IROS
2012
October 7-12
Workshop
Intelligent Robots and Systems
IEEE/RSJ International Conference
Vila Moura, Algarve [Portugal]
Robot Motion Planning: Online, Reactive, and in Real-time

Proceedings of the IROS 2012 Workshop on

Robot Motion Planning
Online, Reactive, and in Real-time

Organizers

Torsten Kroeger

Corrado Guarino Lo Bianco

Preface

Robotic systems have been originally seen as simple substitutes of the human beings in repetitive and dangerous tasks. Consequently, for many years efficiency has represented the main requirement – when not the sole – to be pursued during the synthesis of controllers. Nevertheless, as soon as robots have been introduced in industrial or daily-life environments, the limits of such choice have immediately emerged. It is now evident that actual applications impose to develop systems able to sense the environment and react against unforeseen events. This is certainly true in the case of industrial applications, but it becomes fundamental for systems where a close interaction between humans and robots is expected.

In this sense, online motion planning recently has drawn increasing interest, as robots have started to gain the sensorial and actuation capabilities to effectively perceive their environment. Online planning schemes must evidently be characterized by the same optimality capabilities of their early offline predecessors. Numerous interaction control and motion schemes have been developed, and some of them have already reached the industrial field. However, even though there exists a wide variety of – mostly offline and without reactive behaviors – approaches to problems of motion generation, there is still no unifying concept that connects *(A)* higher-level approaches, that consider cognitive perceptual abilities to perform task-based online motion planning, to *(B)* lower-level approaches that provide the ability of reflex motions triggered by sensor signals and events. Such multilevel planning schemes are typical of human beings that naturally use multiple motion generation layers, which are well connected to each other. For instance, monosynaptic reflexes are triggered in our spinal cord, and, above this most fundamental loop, we can find further anatomic systems for our motor skills, in particular the cerebrum and the cerebellum.

The formulation of a holistic approach for the several layers of online robot motion generation, in combination with autonomous high-level perception-action loops and learning processes, by also taking the dynamic and global uncertainties of the environment into account, still remains an open problem. In order to tackle these interdisciplinary challenges, this full-day workshop intends to bring together world-renowned researchers from fields directly or indirectly related to the field of online motion planning with an emphasis on real-time concepts and reactive motion generation.

We would like to take this opportunity to express our thanks and appreciations to all speakers and authors as well as to all attendees of the workshop for taking part in and contributing to this workshop. We warmly welcome you to Vila Moura!

Torsten Kroeger
Corrado Guarino Lo Bianco

Technical Program

Interactive Session 2

Invited Session 2

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Dynamic Envelopes and Robustly, Continuously Collision-free Trajectories

RayomandVatcha, Jinglin Li, and Jing Xiao
Department of Computer Science
University of North Carolina at Charlotte
xiao@uncc.edu

ABSTRACT

In an unpredictable real-world environment, how the objects move is usually not known beforehand. Thus, whether a robot trajectory is safely collision-free or not has to be tested on-line based on sensing as the robot moves in the environment and taking into account robot motion uncertainty. The problem is more challenging if the robot has a high degree of freedom, such as a mobile manipulator. In this talk, we introduce a general on-line approach to test if a given trajectory segment of the robot, which can have high-DOF, is continuously collision-free, and moreover, if the trajectory segment is *robustly* collision-free, that is, if some deviation of the trajectory within certain "tunnel" of the configuration-time space of the robot is also continuously collision-free. Our method is based on the novel concept of dynamic envelopes [1], which takes advantage of progressive sensing over time without predicting motions of obstacles or assuming specific obstacle motion patterns.

Assume that every obstacle in the unpredictable environment can have a linear speed no greater than v_{max}. Let $R(C)$ be the physical region occupied by the robot at configuration C. To test if the robot at configuration C and future time t is collision-free or not, i.e., if the configuration-time point $x = (C, t)$ is collision-free or not, we define a *dynamic envelope* $E(x, \tau)$ as the closed surface surrounding $R(C)$, such that the minimum distance between $R(C)$ and $E(x, \tau)$ is $v_{max}(t-\tau)$, for sensing time $\tau < t$. If $E(x, \tau)$ is free of obstacle, $x= (C, t)$ is detected collision-free for sure at sensing time τ. As $E(x, \tau)$ is a function of τ, this concept facilitates progressive sensing over a period time to detect if (C, t) is surely collision-free or not *before* time t. Moreover, if the configuration-time point (C, t) is detected collision-free, we will show that a neighborhood of (C, t) is also collision-free. Based on that, we will further introduce an on-line approach to test if a continuous "tunnel" of trajectories in the robot's configuration-time space is collision-free or not by checking if a set of discrete configuration-time points are collision-free or not [2]. Thus, through the concept of dynamic envelopes, we can achieve on-line testing of whether a trajectory segment is continuously and robustly collision-free or not. This approach can be used by a real-time motion planner, such as the RAMP [3], to plan continuously and robustly collision-free trajectories in unpredictable environment.

If a robot has multiple rigid links, a dynamic envelope can be viewed as the union of dynamic envelopes for individual links, which are usually of simple shapes. Therefore, the detection of whether a dynamic envelope intersects an obstacle can be performed quite efficiently via existing fast collision detection algorithms. However, if a robot consists of deformable links, such as a

continuum manipulator, existing intersection detection algorithms based on mesh models of rigid objects are less suitable. We have developed an efficient intersection detection algorithm between an n-section continuum manipulator, with deformable sections, and mesh models of obstacles [4]. The algorithm directly applies to on-line intersection checking between a dynamic envelope of a continuum manipulator and obstacles. The intersection checking for each robot configuration takes only a few percent of the time required by an existing mesh-based collision detection algorithm.

REFERENCES

[1] R. Vatcha and J. Xiao, "Perceived CT-Space for Motion Planning in Unknown and Unpredictable Environments," *Algorithmic Foundation of Robotics VIII* (G.S. Chirikjian, H. Choset, M. Morales, and T. Murphey, Editors), pp. 183-198, Springer, 2010.

[2] R. Vatcha and J. Xiao, "Discovering Guaranteed Continuously Collision-free Robot Trajectories in an Unknown and Unpredictable Environment," *Proceedings of 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2009.

[3] J. Vannoy and J. Xiao, "Real-time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments with Unforeseen Changes," *IEEE Transactions on Robotics*, 24(5):1199-1212, Oct. 2008.

[4] J. Li and J. Xiao, "Exact and Efficient Collision Detection for a Multi-section Continuum Manipulator," *Proceedings of 2012 IEEE International Conference on Robotics and Automation*, May 2012.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

"Design Principles" for Real-World Motion Generation Algorithms

Oliver Brock

Dept. of Computer Engineering and Microelectronics
Robotics and Biology Laboratory
Technische Universität Berlin, Germany

ABSTRACT

Motion planning research has left the simulated world behind and is now generating motion for physical robots in real-world environments. But can we complete this transition into the real world solely through incremental improvements of existing algorithms? Or do the challenges we face - real-time requirements, uncertainty, and the necessity to consider sensing - require more fundamental changes in the way we approach robot motion? Not exclusively for the sake of discussion, I will argue that the answer should be yes! I will propose five "principles" for the design of motion generation algorithms. They deviate from common wisdom in motion planning (as far as I understand it) but help in addressing the challenges mentioned above.

1) Seek incompleteness
2) Balance exploration and exploitation
3) Shift the boundary between planning and control
4) Consider planning and sensing as a whole and not as two integrated parts
5) Know what you can and can't know and deal with uncertainty accordingly

REFERENCES

[1] Nicolas Kuhnen, Arne Sieverling, and Oliver Brock. Motion Generation under Realistic Uncertainty. Manuscript, 2012.
[2] Yuandong Yang and Oliver Brock. Elastic roadmaps - motion generation for autonomous mobile manipulation. Autonomous Robots 28(1):113-130, 2010.
[3] Markus Rickert, Oliver Brock and Alois Knoll. Balancing Exploration and Exploitation in Motion Planning. IEEE International Conference on Robotics and Automation, pp. 2812-2817, 2008.
[4] Brendan Burns and Oliver Brock. Toward Optimal Configuration Space Sampling. Proceedings of Robotics: Science and Systems, pp. 105-112, 2005.

Bootstrapping Online Motion Planning

Sachin Chitta

Willow Garage Inc.
Menlo Park, CA, USA

ABSTRACT

Robots working in the real world will have to deal with dynamic, uncertain environments. They will need to motion plan quickly and generate efficient predictable motions. One approach to this problem is to make motion planning quicker, i.e. develop faster planners that can quickly plan from scratch. Randomized planners have had great success in doing this and are capable of generating new plans very quickly. Another approach to speeding up online motion planning is to take advantage of a priori information, exploiting the structure inherent in most environments and tasks and reusing information from previous plans. In this talk, I will present our approach to using offline information to speed up online planning and make it more realtime. In our first approach, we deal with the problem of motion planning with geometric constraints, using approximations computed offline to speed up online planning. In our second approach, we introduce the concept of Experience Graphs which aim to capture previous motion plans and reuse them when possible while still gracefully degenerating to planning from scratch when necessary. I will present results from both approaches for mobile manipulation tasks using the PR2 robot.

REFERENCES

[1] Motion Planning With Constraints Using Configuration Space Approximations, Sucan, Ioan A.., and Chitta, Sachin, IEEE/RSJ IROS 2012, Vilamoura, Algarve, Portugal, (In Press)
[2] E-Graphs: Bootstrapping Planning with Experience Graphs, Phillips, Michael., Cohen, Benjamin., Chitta, Sachin., and Likhachev, Maxim, Robotics Science and Systems (RSS), 07/2012, Sydney, Australia, (2012)

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Hybrid Planning: Task-Space Control and Sampling-Based Planning

Robert Haschke

***Abstract*— We propose a hybrid approach to motion planning for redundant robots, which combines a powerful control framework with a sampling-based planner. We argue that a suitably chosen task controller already manages a huge amount of trajectory planning work. However, due to its local approach to obstacle avoidance, it may get stuck in local minima. Therefore we augment it with a globally acting planner, which operates in a lower-dimensional search space, thus circumventing the curse of dimensionality afflicting modern, many-DoF robots.**

I. Introduction

Modern two-handed service robots pose enormous challenges to planning and control, especially because they operate in highly cluttered and dynamic environments next to humans, thus demanding efficient, online and real-time capable motion planning and control algorithms. While there exist powerful sampling-based planning methods, which can solve complicated problems (for an overview see [1]), they typically suffer from the curse of dimensionality: the planning effort increases exponentially with the number of degrees of freedom. Modern two-handed, multi-fingered robots easily have more than 50 DoFs, rendering these approaches infeasible for real-world applications.

In order to deal with this complexity, most approaches decompose planning into independent subproblems. For example in grasping, a pre-determined database of grasps for a given object is used to relax the need to plan for the hand motion [2], [3]. Given a set of feasible grasps from this database, the planning can be restricted to the motion of the end effector to reach appropriate hand poses. This planning step is further subdivided into placement of the robot base and subsequent arm motion. However, due to their complexity and their need for pre-computed task knowledge, these algorithms are not yet deployable in unstructured and dynamic environments.

On the other hand, there exist powerful control-based methods, especially the control basis framework of Grupen et al. [4], [5], which provide online-capable approaches to planning and motion generation utilizing gradient-based optimization of suitable cost functions (to reach the object, establish contact, maximize grasp stability, and avoid obstacles and joint limits). However, these methods – due to their local approach – can become trapped in local minima.

The central idea of our work is the integration of local control and global planning into a hybrid approach, which tries to exploit the advantages of both while avoiding their drawbacks: sampling-based planning, which acts globally, is restricted to a low-dimensional, well-suited task-space. This dramatically reduces the search space [6], but also restricts the amount of feasible solutions. To counteract this negative effect, an intelligent local control method exploits the redundancy in the task's null space to increase the success rate of motions between randomly sampled via points.

Sharing the work between local control and global planning allows the global planner to operate on a coarser scale, thus speeding up the overall planning process.

This work was supported by the Center of Excellence Cognitive Interaction Technology (CITEC) and the Honda Research Institute Europe. The author is with the Neuroinformatics Group at Bielefeld University, Germany. rhaschke@techfak.uni-bielefeld.de

II. Hybrid Planning

In the following, we first outline the capabilities of task-space control methods, then we summarize the expansive space tree approach, which we employ for sampling-based planning, and finally introduce the hybrid approach itself.

A. *Task Space Control*

Task-space control methods are founded on the fact, that there exists a (locally) linear correlation of joint movements $\dot{\mathbf{q}}$ and corresponding velocities $\dot{\mathbf{x}}$ of task-space coordinates, which can be easily inverted using the pseudo-inverse of the describing Jacobian $J(\mathbf{q})$ [7]:

$$\dot{\mathbf{q}} = J^{+}(\mathbf{q}) \cdot \dot{\mathbf{x}}\,. \tag{1}$$

In order to deal with numerical instabilities in the vicinity of singularities, several approximative methods were proposed, including singular value decomposition and damped least squares [7]. Redundancy induced by a smaller number of task-space dimensions compared to the number of joints can be exploited to maximize an arbitrary function H. To this end, the gradient $\nabla_{\mathbf{q}} H(\mathbf{q})$ is projected to the null space of J to limit the motion to the redundant space [8]:

$$\dot{\mathbf{q}} = J^{+} \cdot \dot{\mathbf{x}} + N \cdot \nabla_{\mathbf{q}}^{t} H(\mathbf{q})\,, \tag{2}$$

where $N(\mathbf{q}) = \mathbf{1} - J^{+}J$ is the null space projector of J. The main idea of the control basis framework (CBF) [4] is to assume, that the null space motion is generated by a subordinate task controller, J_2, which recursively applies the gradient projection method (2), thus composing complex controllers from simpler ones in an hierarchical fashion:

$$\begin{aligned} \dot{\mathbf{q}} &= \dot{\mathbf{q}}_1 + N_1(\dot{\mathbf{q}}_2 + N_2(\mathbf{q}_3 + \cdots)) \\ &= J_1^{+}\dot{\mathbf{x}}_1 - N_1(J_2^{+}\dot{\mathbf{x}}_2 + N_2(J_3^{+}\dot{\mathbf{x}}_3 + \cdots))\,. \end{aligned} \tag{3}$$

By choosing suitable task representations, one can generate naturally looking, smooth movements in a simple fashion. A major drawback of nowadays motion planning approaches is their attempt to fully specify the end-effector pose in 6D. However, many tasks – due to their inherent symmetry

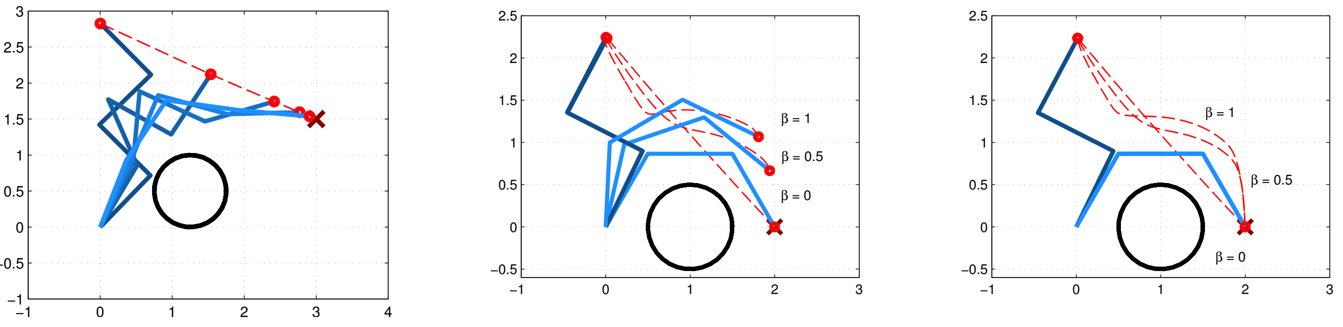


Fig. 1. Goal-directed task-space motion with collision avoidance. Left: Restricting avoidance motions to redundant space yields a straight line motion of the end effector. Middle: Using relaxed motion control (8), the trajectory more strongly avoids the obstacle for larger weights β, but does not converge to the target anymore. Right: Dynamic adaption of β achieves both goals, target reaching and obstacle avoidance (in this example).

– do not require this. For example, grasping a cylindrical object, like a bottle, only requires to align the hand axis with the object axis, the orientation angle around this axis can freely be chosen [9]. To allow even more flexibility, one may specify a task-space interval instead of a distinct target [10]. Platt et al. propose even more abstract controllers, e.g. to maintain force closure, to optimize grasp quality, manipulability, or visibility [5].

Our own implementation of CBF [11], further allows to compose more complex tasks from simpler ones, by (i) stacking Jacobians (solving multiple tasks simultaneously with equal priority), (ii) subtraction of Jacobians (solving relative motion tasks, e.g. left relative to right hand), and (iii) adapting the Jacobian, for example to control the mere distance to a target, i.e.

$$J' = (\mathbf{x} - \mathbf{x}_{\text{goal}})^t \cdot J \tag{4}$$

In the latter case, the task-space motion $\dot{\mathbf{x}}$ is a straight-line towards the goal, much like in classical Cartesian control. However, the redundant space at a given goal distance is the complete sphere around the target and any null space motion is automatically projected onto this sphere. In this manner, we can easily approach spherical objects for grasping from any direction, without the need to precompute a multitude of feasible grasps in advance.

B. Local Collision Avoidance

In the context of motion planning, an important subordinate optimization criterion to be applied in the redundant space is of course collision and joint limit avoidance. Joint limits can be easily avoided minimizing a quadratic or higher-order polynomial function [8], [9]:

$$H_{\mathbf{q}} = \sum w_i \, (q_i - q_i^{\text{ref}})^p \qquad w_i = (q_i^{\max} - q_i^{\min})^{-1}\,, \tag{5}$$

where $\mathbf{q}^{\text{ref}}$ defines a reference pose, e.g. in the middle of the joint range, and the w_i's weight the contribution of individual joints according to their overall motion range.

Local collision avoidance is achieved by a repelling force field originating from each object. To this end, Sugiura [12] proposes to minimize a quadratic cost function defined on the distance $d_{\mathbf{p}} = \|\mathbf{p}_1 - \mathbf{p}_2\|$ between the two closest points $\mathbf{p}_1$ and $\mathbf{p}_2$ on the robot and the obstacle:

$$H_{ca}(\mathbf{p}_1, \mathbf{p}_2) = \begin{cases} \eta\,(d_{\mathbf{p}} - d_B)^2 & d_{\mathbf{p}} < d_B \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Here, d_B acts as a distance threshold below which the force field becomes active and η is a gain parameter. If active, the cost gradient can be computed in terms of the body point Jacobians by applying the chain rule:

$$\nabla^t_{\mathbf{q}} H_{ca} = 2\eta\,(1 - d_B/d_{\mathbf{p}})(J_{\mathbf{p}_1} - J_{\mathbf{p}_2})^t(\mathbf{p}_1 - \mathbf{p}_2)\,, \tag{7}$$

which is easily formulated in the control basis framework. If we employ this cost function for Eq. (2), we yield straight-line task-space movements (e.g. of the end-effector in Cartesian space), while the redundancy is exploited to circumvent obstacles as schematically shown in Fig. 1, left.

To allow more flexible obstacle avoidance, in [13] we proposed a relaxed motion control scheme, which allows deviations from straight-line motions, if the robot gets too close to obstacles:

$$\dot{\mathbf{q}} = J^+(\dot{\mathbf{x}} - \beta\,\dot{\mathbf{x}}_{ca}) - N(\nabla H_{ca} + \nabla H_{\mathbf{q}})\,. \tag{8}$$

Here, additionally to the null-space motion, which minimizes a superposition of both cost functions $H_{\mathbf{q}}$ and H_{ca}, an obstacle avoidance motion $\dot{\mathbf{x}}_{ca}$ directly occurs in task-space as well. This velocity is determined by projecting the cost gradient (7) to the task space:

$$\dot{\mathbf{x}}_{ca} = J\,\nabla^t_{\mathbf{q}} H_{ca}\,. \tag{9}$$

Choosing different values of the weight β, we can smoothly adjust the importance of collision avoidance and target reaching as shown in Fig. 1, middle. However, because both contributions might be contradicting, the target is not always reached with $\beta > 0$. To prevent this, we can ensure, that the goal-directed motion always dominates the collision avoidance motion with a margin ε, if we dynamically adapt β, such that the following condition is fulfilled [14]:

$$\|\dot{\mathbf{x}}\| - \varepsilon \geq \beta \|\dot{\mathbf{x}}_{ca}\|\,. \tag{10}$$

Algorithm 1. Incremental Tree Growing

```
while goal not yet reached do
  if goal bias then
    $\mathbf{p}$ = tree node closest to target $\mathbf{x}_{\text{goal}}$
    $\mathbf{x}_{\text{tgt}} = \mathbf{x}_{\text{goal}}$
  else
    $\mathbf{p}$ = tree node selected according to weights $w_i$
    $\mathbf{x}_{\text{tgt}}$ = task-space via point sampled in vicinity of $\mathbf{p}$
  end if
  $(\bar{\mathbf{x}}, \bar{\mathbf{q}}, t, Q)$ = final motion state of local controller
  add tree node if $t_{\min} < t < t_{\max}$
end while
```

The resulting motion is shown in Fig. 1, right. However, as collision avoidance is the more important objective, it is acceptable to miss the intermediate target. The sampling-based planning component, described in the next subsection, will accommodate for this by globally guiding the search process, providing new via points.

C. Sampling-based Planning

Sampling-based methods randomly grow a tree to explore the whole search space, starting from the initial pose and eventually reaching the targeted pose. The most prominent method, RRT [15], biases its search towards unexplored regions, thus rapidly exploring the whole space. However, we prefer the family of expansive space tree algorithms (EST) [16], because they allow to bias the search in a more fine-grained fashion employing various heuristics. In contrast to the RRT algorithm, EST switches the order of state sampling and tree node selection, performing the following sequence of operations to incrementally grow the tree:

1) randomly select a tree node $\mathbf{p}$
2) sample a new state / via point $\mathbf{x}_{\text{tgt}}$ in vicinity of $\mathbf{p}$
3) extend $\mathbf{p}$ towards $\mathbf{x}$ using a local planner

While sampling-based methods often directly operate on the joint space (to maximally cover the search space), the proposed hybrid planning approach shifts planning to a low-dimensional task-space representation and exploits the powerful task-space controller described in section II-B for local tree extensions. Hence, extensions are more often successful, reducing the need for extensive local refinement of the search tree. Consequently, our sampling-based approach focuses on rapid and coarse-scale exploration of global connectivity. The individual steps of the algorithm and the proposed sampling heuristics are outlined in the following.

1) Node Selection. The major advantage of EST compared to RRT is the possibility to determine, which tree node should be extended next. Plaku et al. bias tree growth towards less covered regions by more frequently choosing tree nodes for extension which have fewer outgoing edges [17]. Assuming a uniform distribution of edge directions and lengths, this yields a reasonable local coverage estimate. However, employing the nontrivial local planner, node extensions more frequently follow similar paths or fail in heavily cluttered environments, because obstacles are avoided in a similar fashion. In this case, node selection should avoid nodes, whose extensions were less successful.

Fortunately, local planning provides various, nontrivial success measures for a node extension, which can be exploited for this additional biasing of the selection process. An important indicator for the presence of obstacles close to the path, is the accumulated magnitude of collision costs: $\mathcal{C} = \int H_{ca}$. However, this measures doesn't account for the direction of the repelling force field. A path should be only considered "difficult", if the costs increase towards the target, i.e. when the goal-direction motion $\dot{\mathbf{x}}$ and the collision avoidance motion $\dot{\mathbf{x}}_{ca}$ are counteracting. In this case the dot product of both vectors becomes negative, leading to the following quality criterion: $Q = \int \dot{\mathbf{x}} \cdot \dot{\mathbf{x}}_{ca}$.

2) State Sampling. In order to optimally cover the local free space in the neighborhood of a tree node $\mathbf{p}$, we propose to apply a sampling strategy which reduces local dispersion [18]. In order to focus the search towards the goal, we apply goal biasing occasionally. To this end, the tree node closest to the target is extended towards the goal. If the local motion controller succeeds to reach the target, we are done. If a node was unsuccessfully used for goal biasing before, the next closest node is used, thus preventing the goal biasing to use the same node over and over.

3) Local Planning. The local motion controller, used to connect a tree node to a newly sampled via point, is limited in duration ($t_{\max}$) to avoid convergence problems in cluttered environments. The local planner returns the reached task-space and joint-space positions $\bar{\mathbf{x}}$ and $\bar{\mathbf{q}}$ of the initial portion of the trajectory, which obeys both joint limit and collision constraints. Additionally, the elapsed control time t and the integrated path quality measure Q is returned.

Finally, the reached state is added as a new node to the tree, if the elapsed control time is between $t_{\min}$ and $t_{\max}$, i.e. if a sufficient path-length could be reached and the controller converged in time. The overall algorithm is summarized in Alg. 1.

D. Task Motion Generation and State Representation

So far, we didn't considered the important aspect of task-space motion generation, i.e. computation of task-space velocities $\dot{\mathbf{x}}$ towards the target. In the past we have employed an algorithm to compute smooth, time-optimal trajectories

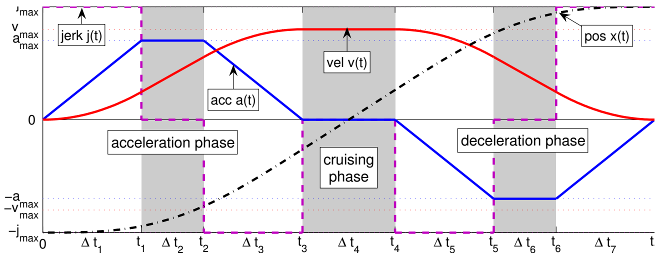


Fig. 2. Time-optimal third-order trajectory profile consisting of seven phases corresponding to maximal jerk (pink), acceleration (blue), and velocity (red) application.

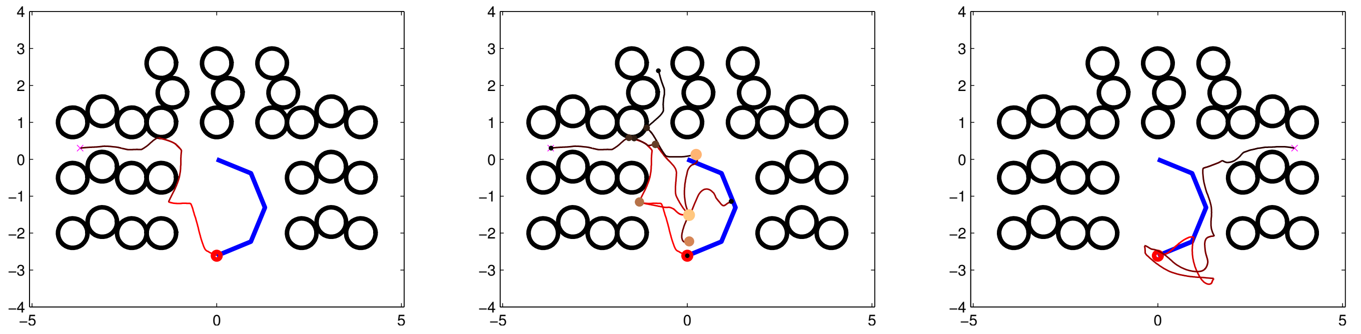


Fig. 3. Motion planning results for a planar 4-joint manipulator moving the end effector (red dot) towards the pink-colored cross. Left and right images show two example trajectories. Middle image illustrates the search tree, with larger and brighter nodes indicating higher exploration weights.

obeying limits on velocity, acceleration and jerk [19]. To this end, a motion trajectory is composed from cubic splines, partitioning the trajectory into phases of applying maximum velocity, acceleration, or jerk as illustrated in Fig. 2. But, the analytic approach involves the computation of zeros of a fourth-order polynomial, which may be ill-posed in certain conditions. Kröger solved this issue with a carefully designed Newton-Raphson iteration algorithm [20].

However, in motion generation, especially for humanoids, it is not important to obtain *the* optimal solution, but it suffices to gain a *very good* one. Hence, we adopt the dynamical-systems approach proposed in [21], [22] utilizing a second-order attractor dynamics (spring-damper system) driving the task-space motion towards the target attractor in a smooth fashion obeying coarse motion limits on velocity, acceleration and jerk:

$$\ddot{\mathbf{x}}(t) = k(\mathbf{x}_{\text{tgt}}(t) - \mathbf{x}(t)) - \gamma\dot{\mathbf{x}}(t)\,, \qquad (11)$$

where k and γ denote the spring and damping constants respectively. Due to its similarity to dynamic motion primitives (DMP) [23], which adds an additional external force $f(t)$ to modulate the shape of the trajectory, it can be easily adapted to imitation learning tasks as well.

According to Eq. 11, the overall state information, which needs to be stored in each tree node, comprises the task-space coordinates $\mathbf{x}$, their velocities $\dot{\mathbf{x}}$, as well as the corresponding joint-space pose $\mathbf{q}$. The latter is required to resolve the redundancy when continuing the search from a specific tree node. That is, although sampling and thus growing of the search tree is performed in task-space primarily, a corresponding secondary tree also exists in joint-space.

III. Results and Discussion

In our publications [13], [24] we demonstrated that the hybrid planning approach finds solutions more often and with fewer tree extensions compared to joint-space methods. However, the more complex local controller takes much more time, such that the overall speedup is limited. Fig. 3 shows exemplary results of a planar 4-joint manipulator moving its end effector (red dot) towards the pink-colored, cross-marked target. The relaxed motion control scheme results in deformed trajectories to better avoid obstacles. Compared to a control scheme, whose avoidance capabilities are limited to the redundant space, relaxed motion control has a higher success rate in complicated situations and takes fewer iterations.

The critical issue of the hybrid planning approach is how to share the workload between the local and global planning. Our preliminary results are encouraging, but there is still room for further improvements by devising improved heuristics for tree node selection and via-point sampling. Summarizing, the shift from perfect towards near-optimal approaches is very promising to realize real-time motion control and planning algorithms for real-world application in many-DoFs, redundant robots.

Acknowledgments

The presented worked was mainly developed by my PhD students Florian Schmidt (control basis framework) and Matthias Behnisch (hybrid planning).

References

[1] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
[2] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," in *Proc. Robotics: Science and Systems IV*, 2008.
[3] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and regrasping tasks," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2009, pp. 2464–2470.
[4] S. Hart, S. Sen, S. Ou, and R. Grupen, "The control basis API – a layered software architecture for autonomous robot learning," in *2009 Workshop on Software Development and Integration in Robotics at ICRA*, 2009.
[5] R. Platt, A. Fagg, and R. Grupen, "Null-space grasp control: theory and experiments," *IEEE Transactions on Robotics*, vol. 26, no. 2, pp. 282–295, 2010.
[6] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009.
[7] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley Publishing Co., Inc., 1991.
[8] A. Liegeois, "Automatic supervisory control of configuration and behavior of multibody mechanisms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 7, no. 12, pp. 861–871, 1977.
[9] M. Gienger, M. Toussaint, and C. Goerick, "Whole-body motion planning — building blocks for intelligent systems," in *Motion Planning for Humanoid Robots*, K. Harada, E. Yoshida, and K. Yokoi, Eds. Springer London, 2010, pp. 67–98.
[10] M. Gienger, H. Janßen, and C. Goerick, "Exploiting task intervals for whole body robot control," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2006, pp. 2484–2490.
[11] F. Schmidt, "Control basis framework," 2011. [Online]. Available: https://github.com/fps/CBF

[12] H. Sugiura, M. Gienger, H. Jannsen, and C. Goerick, “Reactive self collision avoidance with dynamic task prioritization for humanoid robots,” *International Journal of Humanoid Robotics*, vol. 7, no. 01, pp. 31–54, 2010.

[13] M. Behnisch, R. Haschke, H. Ritter, and M. Gienger, “Deformable trees - exploiting local obstacle avoidance,” in *Int. Conf. Humanoid Robots*, 2011, pp. 658–663.

[14] M. Behnisch, “Task level motion planning – Integrating local robot control and global sampling,” Ph.D. dissertation, Bielefeld University, 2012.

[15] S. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” *Computer Science Dept, Iowa State University, Tech. Rep. TR*, pp. 98–11, 1998.

[16] D. Hsu, R. Kindel, J. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *The International Journal of Robotics Research*, vol. 21, no. 3, p. 233, 2002.

[17] E. Plaku, K. Bekris, and L. Kavraki, “Oops for motion planning: An online, open-source, programming system,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2007.

[18] S. R. Lindemann and S. M. LaValle, “Incrementally reducing dispersion by increasing voronoi bias in rrts,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2004.

[19] R. Haschke, E. Weitnauer, and H. Ritter, “On-line planning of time-optimal, jerk-limited trajectories,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008, pp. 3248–3253.

[20] T. Kröger and F. Wahl, “Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events,” *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

[21] M. Gienger, H. Janssen, and C. Goerick, “Task-oriented whole body motion for humanoid robots,” in *IEEE-RAS Int. Conf. Humanoid Robots*, 2005.

[22] M. Toussaint, M. Gienger, and C. Goerick, “Optimization of sequential attractor-based movement for compact movement representation,” in *IEEE-RAS Int. Conf. on Humanoid Robots*, 2007.

[23] A. Ijspeert, J. Nakanishi, and S. Schaal, “Learning attractor landscapes for learning motor primitives,” *Advances in neural information processing systems*, vol. 15, pp. 1523–1530, 2002.

[24] M. Behnisch, R. Haschke, and M. Gienger, “Task space motion planning using reactive control,” in *IEEE-RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2010.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Path Planning for Industrial Robots in Human-Robot Interaction*

Stephan Puls, Patrick Betz, Max Wyden, and Heinz Wörn

Abstract—**In this paper, an online path planning method is presented which has its application in the realm of human-robot interaction and cooperation. Accordingly, a special interest in the safety behavior and effectiveness of the system is taken. Results are presented and discussed with focus on these aspects. Thus, systems gaining greater autonomy in production without safety fences are feasible and enable close human-robot interaction.**

I. Introduction

To achieve human-robot-cooperation in the realm of industrial robotics is a challenging task. In order to realize safety for human co-workers fences are installed or the robot stops its motion in case of human intrusion into the robots working area. Consequently, no real interaction between robot and human sharing space and time can be found.

Due to some progress in the past some modern working cells are equipped with laser scanners for the purpose of foreground detection. But with such a setup no meaningful contribution towards scene understanding can be gained. Thus, no intentional and directed interaction can be achieved.

We are conducting research on scene reconstruction and robot motion planning in a human centered production scenario in order to enable interactive and cooperative systems. Achieving reliable path planning under consideration of the human co-workers pose in the robots working area founds the basis for safe human-robot interaction. Moreover, due to human interactions the path planning needs to adapt to dynamic changes in the environment. In previous work, a framework for human-robot-cooperation (MAROCO) was introduced incorporating a first approach to safe robot motion planning [1, 2]. The here presented work improves and expands the path planning module in the framework.

The remainder of this paper is organized as follows. In Section 2, selected research work on path planning is presented. In Section 3, a short overview of the MAROCO framework and relevant modules is given. In Section 4, the path planning method is detailed and its process sequence is explained. In Section 5, experimental evaluation is given and results are discussed. Finally, Section 6 gives a summary and some hints for future work are mentioned.

*Research is partially supported by BW-Stiftung through the research project SILIA.

S. Puls is with the Institute of Process Control and Robotics (IPR) at the Karlsruhe Institute of Technology (KIT), Karlsruhe, BW 76131 Germany (phone: 0049-721-608-48485; fax: 0049-721-608-47141; e-mail: stephan.puls@kit.edu).

P. Betz and M. Wyden are with the Karlsruhe Institute of Technology, Karlsruhe, BW 76131 Germany. (e-mail: {patrick.betz, max.wyden}@student.kit.edu).

H. Wörn is with the Institute of Process Control and Robotics (IPR) at the Karlsruhe Institute of Technology (KIT), Karlsruhe, BW 76131 Germany (e-mail: woern@kit.edu).

II. Related Work

Path and motion planning for robots is an important means for enabling autonomous robot movement in its environment. Different distinctions of path planning have developed. On the one side, there are purely reactive planners such as behavior based path planners [3] or potential field methods [4]. These approaches are based on local information of the environment and, thus, prone to local minima. On the other side, there are deliberate planners which process global information. These planners are based on searching through a graph which represents the robots configuration space [5]. In order to achieve fast online planning, there is an offline phase in which the search-graph structure is computed.

In [6], a deliberate path planning method based on Dynamic Roadmaps is presented [7]. The system is implemented for a robot arm on a mobile platform which is used in a service oriented scenario. Thus, movement of the robot arm needs to react on changes in the environment and plan its path accordingly. The presented system uses a time-of-flight sensor to detect obstacles and update a voxel model of the environment. This is used to update the search graph. The approach achieves planning times of less than 100 ms and is argued to be faster than human reaction times.

In [8], a hybrid method of reactive and deliberate planners is presented. A roadmap based method is used to determine a path before motion execution. During robot motion the environment model is updated with changed geometric data. In order to avoid collisions the path is adapted based on local information. If the change of the path in the adaptation step is too big a complete re-planning is invoked. The purpose of path adaption is to reduce the number of planner invocations and, thus, increase the overall system performance. Presented results were obtained from simulation only and demonstrate the validity of the approach.

A different approach to accelerate system performance is followed in [9] by using GPU-based parallel algorithms for collision checking. This allows evaluating multiple configurations simultaneously and performing efficient collision queries. Similar to [6], collision free paths can be computed in less than 100 ms.

III. The MAROCO Framework

In order to realize a comprehensive approach to cognitive robotics and enabling human-robot interaction the MAROCO framework was implemented. Sensing of the robots working area is accomplished by a time-of-flight bases camera. Due to occlusion reasons it is mounted at the ceiling, thus, allowing handling of objects and arbitrary robot motion.

Based on the depth information of the working area the human kinematics can be reconstructed without the need of marker [1]. The reconstructed model of the human co-worker is embedded into the overall scene of the working cell (see Fig. 1).

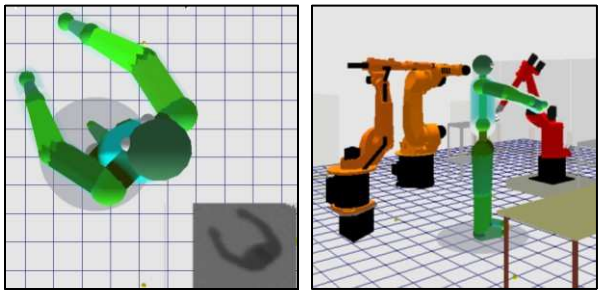


Figure 1. Human kinematics reconstruction based on depth information from time-of-flight camera (left) and the overall scene (right).

The information about the human pose and its relation towards the robots is used for risk assessment and estimation. Due to the fact that safety is a crucial feature in productive interaction scenarios, different methods for risk estimation were implemented and evaluated [10].

Based on the risk assessment different strategies for risk minimization are feasible. In contrast to sole robot velocity adjustments actual re-planning and active collision avoidance is more challenging and rewarding. As shown in Section 5, a robots' capability to maneuver around obstacles increases throughput and productivity.

IV. Method Description

The path planning method is based on the idea of Dynamic Roadmaps and, thus, consists of two phases. An offline phase captures the configuration space and maps it onto a graph structure. The online phase is used to do the actual planning through graph updating and searching.

A. *The Offline Phase*

In the research of the last decade, especially randomized sampling based techniques to capture the free configurations space (CSpace) have gained popularity. These techniques achieve improved performance while not striving for optimality [5].

In our work, the robots' environment can be dynamic due to human interaction. Thus, the free CSpace is not distinguishable from the overall CSpace a-priori. Consequently, we implemented a systematic and deterministic sampling method which intersperses the CSpace in regular intervals. Moreover, virtual fences can be defined to restrict the work space of the robot, thus, accounting for walls, other stationary machinery or adapt the work space depending on task and interaction scenario (see Fig. 2).

Based on requirements defined in EN ISO 10218-1 distance between robot and human and resulting risk have to be assessed and need to influence the robots' velocity. In order to enforce these requirements early in the planning process, a predefined velocity distribution over the work space can be set. Doing so, safety of virtual fences can be enforced. Also, if the interaction space of human and robot is known a-priori, the robots' velocity in that space can be reduced and defined in the offline phase.

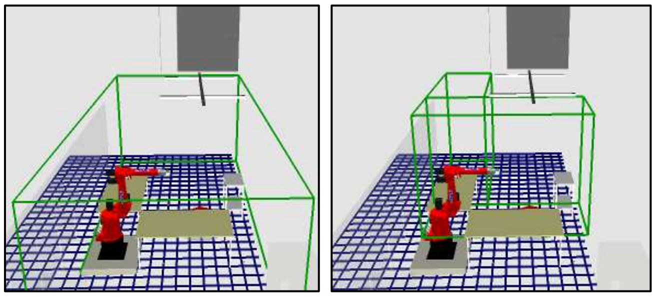


Figure 2. Virtual fences for adapting work space to task.

The sampled configurations are used as nodes in a search graph. Each node is augmented with its predefined maximal velocity if any is given. Each node is then connected with its *k* nearest neighbors.

B. *The Online Phase*

The Dynamic Roadmap method as presented in [7] defines a mapping from work space to configurations in the roadmap. This mapping is based on a discretization of the work space into cells. Each cell is linked to the corresponding nodes and edges in the roadmap. During run-time occupancy of the cells are tested and linked nodes are invalidated.

This procedure has two drawbacks that our approach avoids:

- All configurations linked to an occupied cell are invalidated. This is also done, if only a very little proportion of the cell is actually occupied. This does also depend on cell size, thus resolution.
- Invalidation is only dependent on actual physical obstacles and their spatial circumference. Situation dependent risk estimations cannot be respected.

The human kinematics reconstruction allows us to approximate the human pose geometrically through a sphere model, thus, representing the human kinematics through a dense set of spheres. Such a sphere model is also defined for the robot. This enables fast distance computations and, thus, collision checking (see Fig. 3). Moreover, we use a two-threaded fuzzy logic system for risk estimation of a situation based on the humans head pose and its relations to the robot.

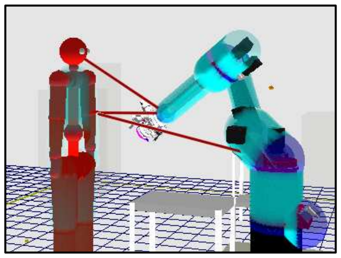


Figure 3. Fast distance computations with sphere models.

Consequently, the update and invalidation of the graph nodes is done during search time. For defining a configuration as colliding two criteria can be used: Distance human to robot or distance in combination with the risk assessment. Our reasoning about risk is that if the human co-worker is seeing what actions the robot is performing it is less risky than if the human is turned around and is distracted. In the latter case, the human might take a step back and walk unintentionally into the robot. Accordingly, greater distances need to be enforced in such a case. Thus, the combined criterion allows for better situation dependent planning and is used in the online phase.

Another important aspect that influences the search time and path quality is the used heuristic function during the A*-search. In contrast to [6] and [7] which use workspace metrics, our approach uses a configuration space metric, namely Euclidian distance. Due to the fact that different configurations can reach the same pose in work space a simple work space metric based on tool center point (TCP) positions is unsuitable. Thus, work space metrics can be devised that consider a set of reference points on the robot surface additionally to the TCP [6, 7]. We argue that the path of the robot is captured best in CSpace because shorter angular distances result in more efficient paths. Thus, the Euclidian metric is applied directly to the configurations avoiding detour computations through work space metrics.

The risk assessment for a given configuration and a human pose can also be used to guide the robots velocity. As stated above, it is required by normative regulations to adapt the robots velocity due to a humans' presence. Thus, the risk value yields a safety value in the interval (0, 1], with 0 being defined as *colliding* and 1 as *safe*. This safety value can be mapped onto the interval [ε, 1] which can be regarded as maximal allowable velocity for robot movement with ε being a minimal velocity so that the robot will move with a safe velocity in risky but non-colliding configurations.

The maximal allowable velocity value can be used two-fold: Firstly, it augments the resulting path so that the robot motion can comply with these parameters. Secondly, the node evaluation function *f*, given in (1), is adapted to prefer faster paths over slower ones by modification of function *g* which is a measure for the distance from starting configuration to current node *i*. The parameter s_k resembles the reciprocal maximal allowable velocity for node *k*.

$$f(v_i) = g(v_i) + h(v_i).$$

$$g(v_i) = \Sigma\,(s_{k+1} \cdot \|v_k - v_{k+1}\|).\quad (k = 0,\ \dots,\ i-1) \tag{1}$$

This modification increases the *g*-value for a node if it has a safety value smaller than 1. Moreover, the heuristic function is unchanged and does not account for future possible slower path traversal in order to be admissible and optimistic. This divergence of functions *g* and *h* leads to more expanded nodes and, thus, longer search times but generally better paths as is shown in Section 5.

C. *Reactivity in Changing Environments*

In order to react on a moving human co-worker the system has to check for imminent collisions along its planned path. It is not efficient to compute collision detections for the whole path during robot motion because possible collision at the end of the path might not actually occur when the robot reaches these configurations. Thus, for determining possible collisions a limited look-ahead is used to check a set of close-future configurations. These are determined by extrapolation along the planned path for a certain distance and computing distances and risk assessment. In order to cope with discretization of the look-ahead a safety clearance for each configuration is defined in which no obstacle is allowed to appear.

If the look-ahead detects an imminent collision during robot motion the robot is stopped and a path re-planning is invoked. The current robot configuration is used as starting configuration whereas the goal is kept unchanged. As soon as the re-planning has found a suitable path the robot continues its motion.

In contrast to our previous work, path re-planning is not preemptive but uses its own thread for computations. This change is necessary due to further developments of additional functionality in the framework. More modules require processing time, thus, continued use of the preemptive approach would require smaller time slices and result in usage of more processing cycles for path finding. Consequently, the dynamics of the work space cannot be captured by the planning module and the system would seem not as responsive.

During path search the core MAROCO modules are processed with a high priority, thus, allowing timely processing of the sensor data and safe stopping of the robot.

V. Experimental Results

For experimental analysis different scenarios were examined. Specifically different approaches to node evaluation during A* search were tested. These evaluation methods are:

- Using Euclidian distance for $g(v_i)$ (Type 1),
- Using modified *g*-function (Type 2),
- Instead of defining a safety value of 0 to be colliding using a value $\varepsilon > 0$ (Type 3).

Moreover, runtime tests were conducted without any re-planning in order to verify the effectiveness of planning in contrast to stopping the robot and waiting for obstacles to disappear.

A sequence of a human moving in the work space was recorded. During evaluation this sequence was played back in a loop and used as sensor input. The sequence consists of 2000 frames in which the human moved freely in the sensor area. The motion consisted of linear and circular paths with habitual and different walking velocities. In total 9000 frames were analyzed for each node evaluation method.

During evaluation different parameters were recorded (see Table I). In order to capture the effectiveness the number of finished paths and canceled paths is of interest. Canceled paths are those that are interrupted before reaching the goal due to invocation of re-planning. The big discrepancy of the number of canceled paths of type 1 and the others is mainly caused by paths that are not even started because the planning returned a colliding look-ahead already in the frame after planning. Thus, the re-planned path was valid in its instance but due to human motion was invalidated the next moment. This is caused by planning very close to the risk-safety boundary which might change drastically if the human is turning around. It resembles the fastest way around the obstacle towards the goal but is not concerned with possible future risk increase. The number of finished paths, on the other hand, counts the times the robot reached its goal.

TABLE I. EVALUATION RESULTS

Evaluation method type	***Type 1***	***Type 2***	***Type 3***
# finished paths	44	46	42
# canceled paths	109	24	15
# searches total	433	316	675
# successful searches	151	70	57
# unsucessful searches	282	246	618
Avg. path length	17.7	19.1	18.5
Avg. time for successful search [ms]	27.33	233.68	62.63
Avg. time for unsuccessful search [ms]	294.89	371.44	227.31
Avg. time search total [ms]	201.58	340.92	213.41

In the case that no path re-planning was used 41 finished paths were recorded. As can be seen in Table 1, all versions of path re-planning were more effective. In case of type 3 the difference is only one more finished path. This is due to the recorded sequence in which the human was almost always in motion, thus, not blocking the robots path for long. For such dynamic and risk dependent scenarios new benchmarks need to be defined in order to truly assess effectiveness. The here presented evaluations give a starting point and help for improving developed algorithms.

Overall, the evaluation method considering maximal allowable velocity (type 2) achieves most finished paths and has less than a quarter of canceled paths compared to basic Euclidian node evaluation (type 1). In comparison of needed search time type 2 is about 8 times slower for successful searches than type 1. In average over all searches the factor decreases noticeably to approximately 1.7. The higher effectiveness of type 2 is contrasted by the higher timely requirements. Type 1 also requires more searches in total due to more canceled paths, thus, reducing effectiveness.

In comparison with [6] and [7], it is notable that our runtimes are slower by a factor of three for type 1 planning. This is due to required forward kinematics computations in order to assess the robot and human pose dependent risk value. Thus, reactivity is slower but greater safety is achieved which in turn decreases the need for many re-planning invocations, especially for type 2 planning. Moreover, as demonstrated in [9], GPU accelerated computation can help improving runtimes. This might also be applicable to our approach.

In Figure 4, the correlation between the number of expanded nodes and the needed runtime for re-planning is shown for each node evaluation method. It can be seen that for all types there is a clear linear correlation between expanded nodes and required search time. Due to the properties of the A* algorithm this is to be expected.

For type 1 there is a clear distinction and clustering. Searches that expand more than 1000 nodes return unsuccessful. A similar cluster of unsuccessful searches can be seen for type 2 but successful searches are stretched over the scale and even a high number of expanded nodes might lead to a found path. This behavior is caused by striving for faster paths rather than shorter ones. In cases of narrow passages in which the risk is high a faster detour is searched. But, instead of failing, this passage can be slowly traversed. Type 3 has overall different characteristics. In order for a search to fail comparably few nodes need to be expanded. This is due to the increase requirement for safety evaluation of a node. Narrower passages get blocked and dead ends are reached.

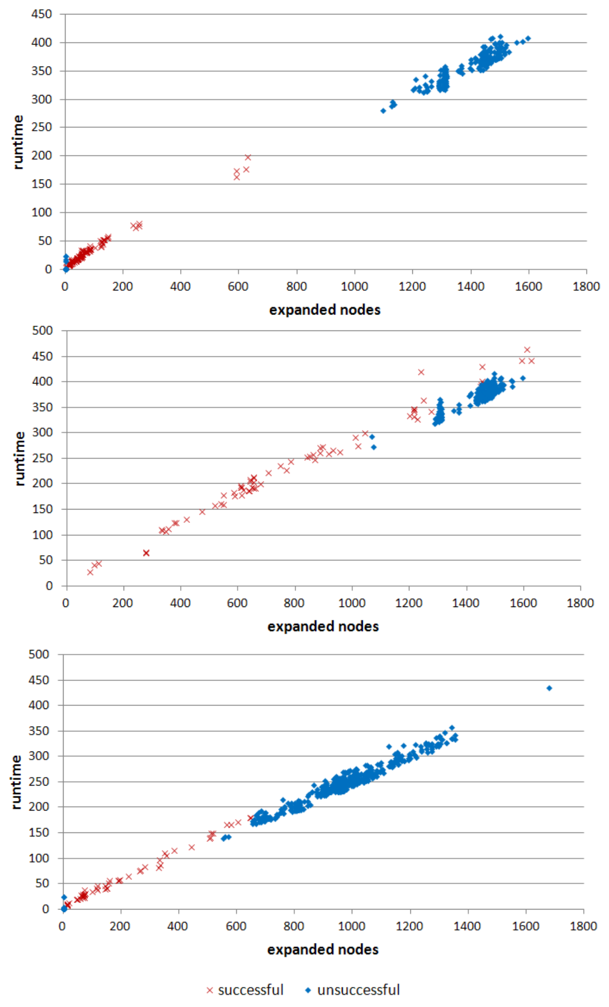


Figure 4. Distributions of runtime over number of expanded nodes for successful and unsuccessful searches. Runtime is given in [ms]. Top: Type 1. Center: Type 2. Bottom: Type 3.

Different planned paths for type 1 and 2 are given in Figure 5. The paths are symbolized by the trajectories of the TCP. The top left image shows the original planned path with the human standing far away and watching the robot. An indication of the overall risk assessment is symbolized by the color of the human model. Thus, green means safe and red shows a high risk value. The color can vary between these two poles.

The human poses depicted in the top right and bottom left images of Figure 5 are similar but the resulted re-planned paths differ noticeably. The images show the outcomes of type 2 and type 1 planning respectively. Type 2 planning keeps a greater safety margin to the human, whereas the type 1 planning finds a shorter path. The resulting higher risk assessment of the type 1 result can be seen due to higher red values of the human model. A short step backwards of the human would result in a new re-plan invocation.

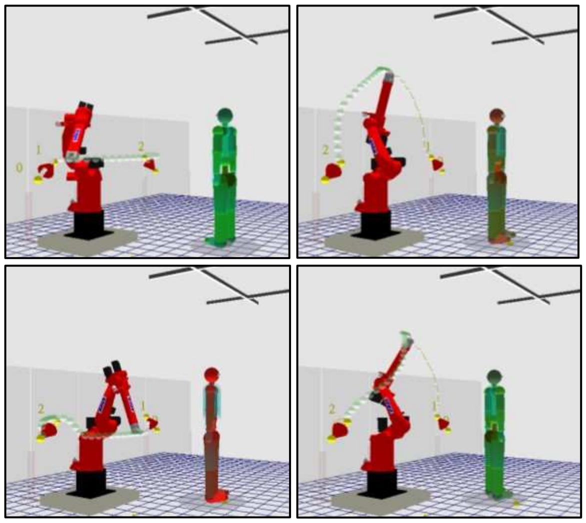


Figure 5. Examples for re-planned paths. Top left: Original plan. Top right: Re-planning type 2. Bottom row: Type 1 re-plans for different situations.

As can be seen in the bottom right image, the human has to move rather close to the original path in order to provoke similar results as type 2 planning (see Fig. 5). Consequently, searching for paths utilizing higher robot velocity results in safer paths.

VI. Summary and Future Work

In this work, we demonstrated and evaluated a reactive online path planning method in conjunction with safe path traversal. Imminent collisions and high-risk situations are detected and path re-planning is invoked accordingly.

For path planning the Dynamic Roadmap approach was adapted in order to cope with situation dependent risk assessment. Instead of invalidating sets of configurations that are linked to an occupied cell in work space each expanded node of the search graph is evaluated separately. This leads to longer processing time compared to [6, 7] but increases safety margins and allows situational robot velocity specification.

For experimental analysis different node evaluation methods were implemented and compared. All methods enabled safe robot motion but achieved differing effectiveness. Even though requiring more processing time during search the method preferring faster paths over slower ones achieved best effectiveness with most finished path traversals and only a few canceled ones.

The presented method allows for effective human co-worker avoidance. In shared work spaces this is a necessity and results in higher productivity and safety at the same time. In contact based cooperative scenarios this approach needs adaption in order to allow intentional "collisions", hence robot guidance or joint object manipulation. Nevertheless, safety is of utmost concern and has to be guaranteed during human interaction.

Reducing the required processing time during search is a further point of improvement. Using acceleration techniques provided by GPU processing might lead to better performances and need further investigation.

References

[1] S. Puls, J. Graf, and H. Wörn, "Cognitive Robotics in Industrial Environments," in *Human Machine Interaction – Getting Closer*, InTech, I. Maurtua (Ed.), Rijeka, Croatia, pp. 213-234, 2012.

[2] J. Graf, S. Puls, and H. Wörn, "Incorporating Novel Path Planning Method into Cognitive Vision System for Safe Human-Robot Interaction," in *Proc. of IARIA Computation World: Cognitive 2009*, Athens, Greece, pp. 443-447, 2009.

[3] R. C. Arkin, *Behaviour Based Robotics*, Cambridge, U.K.: Cambridge Univ. Press, 1998.

[4] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," in *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90-98, 1986.

[5] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[6] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-Time Path Planning for a Robot Arm in Changing Environments," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[7] P. Leven and S. Hutchinson, "A Framework for Real-Time Path Planning in Changing Environments," in *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999-1030, 2002.

[8] E. Yoshida and F. Kanehiro, "Reactive Robot Motion using Path Replanning and Deformation," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[9] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," in *The International Journal of Robotics Research*, 2011.

[10] J. Graf, P. Czapiewski, and H. Wörn, "Evaluating Risk Estimation Methods and Path Planning for Safe Human-Robot Cooperation," in *Proc. of ISR/Robotik 2010*, Munich, Germany, 2010.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

A Synergetic High-level/Reactive Planning Framework with Application to Human-Assisted Navigation

Antonio Franchi, Carlo Masone and Paolo Robuffo Giordano

Abstract— **In this work we present a novel framework for the systematic integration of high-level/mission schedulers, middle-level/cognitive-enabled online-planners and low-level/reactive trajectory modifiers. The approach does not rely on a particular parametrization of the trajectory and assumes a basic environment representation. As an application, the online capabilities of the method can be used to let a mobile robot cooperate with a human taking the role of the middle-level planner. In that case we also describe a rigorous way to bilaterally couple the human and the reactive planner in order to provide an immersive haptic feeling of the planner state. Hardware/Human in-the-loop simulations, with a quadrotor UAV used as robotic platform and a real haptic instrument, are provided as validating showcase of the presented theoretical framework.**

I. INTRODUCTION

Combination of low-level controllers with high-level/sophisticated planning abilities represents one of the crucial issues to enable complex decision making in real-world unstructured scenarios. Furthermore, the presence of a systematic framework to combine these two aspects may also result helpful in many task requiring human-robot interaction and cooperation, e.g., in order to optimally balance the human commitments and robot autonomy.

In these notes we focus on the very common scenario where a mobile robot is tasked to navigate in an environment in order to accomplish some given mission, e.g., exploration, surveillance, monitoring, search-and-rescue, good transportation, mobile-networking, etc.. Some prior knowledge on the environment may be given (e.g., the environment size and a rough map that has been retrieved with a preliminary exploration). The environment is also populated with obstacles and points-of-interests that can only be detected when the robot is sufficiently close (e.g., victims, goods to be loaded, charging docks, stationary antennas, etc.). The planning step is then divided in three phases: mission scheduling, online middle-level planning, and reactive trajectory modifier. Our deformation differs from other well known approaches that apply artificial forces to a sequence of configurations [1], [2], or define trajectory modifications as input functions along the admissible directions of motion [3], because it does not arbitrarily change the path but only some desired geometric properties.

We also propose a systematic way to incorporate a human assistant as online middle-level planner, in order to exploit his/her advanced cognitive capabilities. In the case of a human in-the-loop we propose a haptic algorithm that feeds back to the human a force cue informative of the *global* deformation acting on the desired path rather than on a *local* mismatch between commanded and executed position/velocity, as in all the previous works in our knowledge, see, e.g., [4], [5], [6], [7], [8].

Summarizing, the main contributions of our framework are: i) a systematic integration of three different motion planning layers that does not rely on a particular trajectory parametrization and uses a basic representation of the surrounding environment ii) the possibility of seamlessly applying this framework for including, online and in real-time, a human operator in the planning loop in order to exploit her/his superior cognitive skills, iii) the design of a general force-cue paradigm that closes the loop between the automatic part of the motion planner and the human assistant (when present), and iv) the fact the the proposed force cue is informative of the global deformation of the desired path rather than of the mismatch between direct motion commands and their execution, as in previous bilateral teleoperation frameworks.

This framework is the generalization of the ideas presented in [9] where only the case of human operators and persistent trajectories are considered. In these notes we also present additiona plots from new simulation results.

II. SYNERGETIC TRAJECTORY PLANNING

The proposed planning framework is constituted by three main sub-systems: i) a high-level *task scheduler*, ii) a *middle-level modifier*, iii) a *reactive modifier*, as depicted in Fig. 1. In the case that the middle-level modifier duties are performed by a human assistant (as described in Sec. III) then an additional sub-system is represented by the *bilateral-controller* that provides the assistant with a suitable haptic feedback.

The three fundamental blocks operate in cascade. The task scheduler (TS) periodically generates an *initial reference path* that is intended valid for a given time horizon and it is based on past information. For example, it can be an exploration algorithm that plans the next move based on the current partial map or a coverage method that switches between predefined curve patterns. The middle-level modifier (MM) builds upon the initial reference path in order to generate online a time-varying *reference trajectory*. The underlying idea is that the MM comprises a sophisticated algorithm (or even a human assistant) that is able to promptly take into account new pieces of information gathered by the mobile robot and refine in real-time the reference motion by resorting on some sophisticated/cognitive capabilities. Finally the reactive modifier (RM) goal is to generate online

A. Franchi, C. Masone, and P. Robuffo Giordano are with the Max Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen, Germany {antonio.franchi,carlo.masone, prg}@tuebingen.mpg.de.

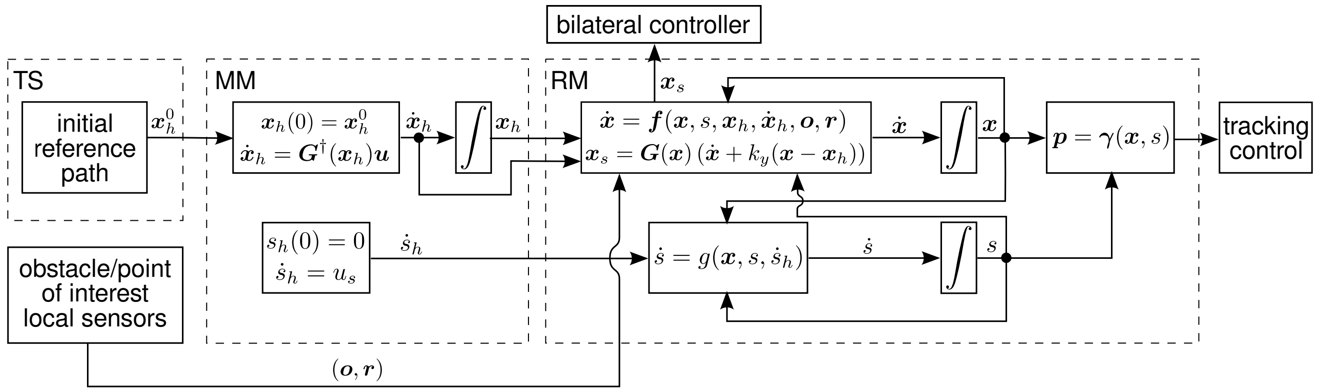


Fig. 1: Block diagram of the planning and control framework, with highlighted the high-level task scheduler (TS), the middle-level modifier (MM), and the low-level reactive modifier (RM).

the actually *tracked trajectory* for the robot motion controller in order to meet the following specifications:

1) ensuring feasibility of the motion, given the robot kinematic and dynamic constraints,
2) let the tracked trajectory be as much as possible similar to the reference trajectory,
3) ensure obstacle avoidance,
4) pass close to the points-of-interest that are located in the vicinity of the reference trajectory.

The reference trajectory, generated online by the MM, is specified as a *geometric path*

$$\gamma : \mathbb{R}^n \times [0, L_h] \to \mathbb{R}^d, \quad \text{s.t.} \quad (x_h, s_h) \mapsto \gamma(x_h, s_h)$$

(where $x_h \in \mathbb{R}^n$ is a vector of shape parameters uniquely defining the curve, L_h is the curve length, and s_h is the arc length of the curve) together with a *timing law* $s_h(t)$ which ultimately determines how the robot should travel the path.

At a certain initial time t_0 the TS provides the initial reference path to the MM in the form of an initial set of geometric parameters x_h^0 for the reference trajectory, Then the MM sets $x_h(t_0) = x_h^0$ and $s_h = 0$ and from that moment the MM is free to modify the reference trajectory by acting on $\dot{x}_h$ and $\dot{s}_h$. The current reference trajectory stays alive until the TS provides a new initial reference path to the MM. This event brings to a reset of the reference trajectory to the new initial reference path, and so on.

The number of parameters n and their kind depends on the specific representation used for γ and in general a larger n results in a higher flexibility of the geometric path. Nevertheless, managing the total number n of parameters required to cope with a typical unstructured environment may demand a too high computational load on the cognitive-planning side (e.g., it may exceed the number of quantities that a human operator can reasonably control at once).

For this reason we consider a vector $y(x_h) = (y_1(x_h) \dots y_m(x_h))^T \in \mathbb{R}^m$, $m \leq n$, defining the m degrees of freedom assigned to the middle-level modifier. The time variation of y is given by

$$\dot{y} = \left(\frac{\partial y_1}{\partial x_h}^T \dots \frac{\partial y_m}{\partial x_h}^T \right)^T \dot{x}_h = \frac{\partial y}{\partial x_h} \dot{x}_h = G(x_h)\dot{x}_h, \tag{1}$$

where matrix $G(x_h) \in \mathbb{R}^{m \times n}$ is assumed to have full row-rank so that the m dofs controlled by the MM are independent.

The action of the MM on the reference trajectory is then obtained by means of the following dynamical system

$$\dot{s}_h = u_s \tag{2a}$$

$$\dot{x}_h = G^\dagger (x_h) u, \tag{2b}$$

where $u_s \in \mathbb{R}$ and $u \in \mathbb{R}^m$ represent the actual MM commands (whose detailed expression in the case of a human operator is described in Sec. III), and $G^\dagger$ is the pseudo-inverse of matrix G.

A. Pure-reactive Dynamics of the Tracked Trajectory

The environment is considered populated by a set of obstacles, described by the vector of obstacle points $o = (o_1 \dots o_{n_o}) \in \mathbb{R}^{d \times n_o}$, and a set of regions of interest, described by the vector of points of interest $r = (r_1 \dots r_{n_r}) \in \mathbb{R}^{d \times n_r}$. Given the reference trajectory parameters (s_h, x_h) and the environment (o, r), the reactive modifier generates the tracked trajectory $p(t)$ as dictated by the following dynamical system:

$$\dot{s} = g(x, s, \dot{s}_h) \tag{3a}$$

$$\dot{x} = f(x, s, x_h, \dot{x}_h, o, r) \tag{3b}$$

$$p = \gamma(x, s), \tag{3c}$$

where $x \in \mathbb{R}^n$ and $s \in [0, L]$ are the shape parameters and the arc length of the tracked trajectory, respectively, and the initial conditions are of the system as set as $x(t_0) = x_h(t_0) = x_h^0$, and $s(t_0) = 0$.

The arc length map (3a) has the following form

$$g(x, s, \dot{s}_h) = g_1(x, s, \dot{s}_h)\dot{s}_h, \tag{4}$$

where $g_1 \in [0, 1]$ is designed so as to cope with the robot motion capabilities. In particular, $g_1 = 1$ if the robot can travel along the path at the desired speed $\dot{s}_h$, and $g_1 \to 0$ (thus, towards a full stop) whenever the speed $\dot{s}_h$ becomes too large for the actuation capabilities of the robot (e.g., because of a too large curvature or a low energy level for the robot batteries).

Concerning the vector field $\boldsymbol{f}$, we first show how to design it in order to meet the first specification, i.e., feasibility. In particular we want to prevent that at any time the geometric properties of $\boldsymbol{\gamma}(\boldsymbol{x},\, s)$ are directly influenced by the variation $\dot{\boldsymbol{x}}$ given by (3b) This is important to ease the action of the robot trajectory tracker which expects presence of some local regularity properties of the curve being tracked[1].

Local geometric properties of a curve are characterized by the k-th derivatives w.r.t. the arc length s, i.e.,

$$\boldsymbol{p}^{(k)}(\boldsymbol{x}, s) = \boldsymbol{\gamma}^{(k)}(\boldsymbol{x}, s) = \frac{\partial^k \boldsymbol{\gamma}}{\partial s^k}(\boldsymbol{x}, s), \quad k \in \mathbb{N}_0$$

where the operator $\cdot^{(k)}$ indicates the k-th geometric derivative. By applying the chain rule, the variation of $\boldsymbol{p}^{(k)}$ due to changes of $\boldsymbol{x}$ and s over time is given by

$$\frac{d}{dt}\left(\boldsymbol{p}^{(k)}(\boldsymbol{x}, s)\right) = \left.\frac{\partial \boldsymbol{\gamma}^{(k)}}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{x},s)} \dot{\boldsymbol{x}} + \left.\frac{\partial \boldsymbol{\gamma}^{(k)}}{\partial s}\right|_{(\boldsymbol{x},s)} \dot{s}. \quad (5)$$

Stacking eq. (5) for the first k derivatives then yields

$$\dot{\boldsymbol{\Gamma}}_{0,k}(\boldsymbol{x}, s) = \boldsymbol{J}(\boldsymbol{x}, s)\dot{\boldsymbol{x}} + \boldsymbol{\Gamma}_{1,k+1}(\boldsymbol{x}, s)\dot{s}. \quad (6)$$

where $\boldsymbol{\Gamma}_{i,j} = [\boldsymbol{\gamma}^{(i)^T} \dots \boldsymbol{\gamma}^{(j)^T}]^T \in \mathbb{R}^{2(j-i+1)}$ and $\boldsymbol{J} \in \mathbb{R}^{2(k+1)\times n}$. Equation (6) shows how the geometric properties of the curve γ at some $(\boldsymbol{x}, s)$ depend on two contributions: the first is due to the parameter change $\dot{\boldsymbol{x}}$ and the second to the longitudinal speed $\dot{s}$. The feasibility requirement on $\boldsymbol{f}$ can then be interpreted as imposing that the first term in (6) is zero when evaluated at the current $(\boldsymbol{x},\, s)$. Therefore, the design of the vector field $\boldsymbol{f}$ must ensure that

$$\boldsymbol{J}(\boldsymbol{x}(t), s(t))\dot{\boldsymbol{x}} = \boldsymbol{0}. \quad (7)$$

Assuming matrix $\boldsymbol{J}$ has a non-empy null-space, an infinity of possible $\dot{\boldsymbol{x}} \in \mathcal{N}(\boldsymbol{J})$ would meet the constraint. To this end, let $\boldsymbol{J}^{\dagger}(\boldsymbol{x}, s)$ represent the pseudo-inverse of $\boldsymbol{J}(\boldsymbol{x}, s)$, and $\boldsymbol{N}(\boldsymbol{x}, s) \in \mathbb{R}^{n\times n}$ a projector matrix spanning the null space $\mathcal{N}(\boldsymbol{J})$, i.e., such that $\boldsymbol{J}\boldsymbol{N} = \boldsymbol{0}$. We then design the vector field $\boldsymbol{f}$ to have the following form[2]

$$\boldsymbol{f}\left(\boldsymbol{x}, s, \boldsymbol{x}_h, \dot{\boldsymbol{x}}_h, \boldsymbol{o}, \boldsymbol{r}\right) = \boldsymbol{N}(\boldsymbol{x}, s)\boldsymbol{f}_1\left(\boldsymbol{x}, \boldsymbol{x}_h, \dot{\boldsymbol{x}}_h, \boldsymbol{o}, \boldsymbol{r}\right). \quad (8)$$

Assuming that matrix $\boldsymbol{J}$ has full row-rank, a well-known choice for the projector operator is $\boldsymbol{N} = (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})$.

In order to meet the remaining three specifications, we then design $\boldsymbol{f}_1$ in (8) as the composition of four terms:

$$\boldsymbol{f}_1 = \boldsymbol{f}_h(\boldsymbol{x}, \boldsymbol{x}_h, \dot{\boldsymbol{x}}_h) + \boldsymbol{f}_o(\boldsymbol{x}, \boldsymbol{o}) + \boldsymbol{f}_r(\boldsymbol{x}, \boldsymbol{r}) + \boldsymbol{f}_i(\boldsymbol{x}). \quad (9)$$

The first term implements a feedforward/proportional action

$$\boldsymbol{f}_h(\boldsymbol{x}, \boldsymbol{x}_h, \dot{\boldsymbol{x}}_h) = \dot{\boldsymbol{x}}_h + k_h(\boldsymbol{x}_h - \boldsymbol{x}), \quad (10)$$

with $k_h > 0$, in order to steer $\boldsymbol{x}$ towards the reference $\boldsymbol{x}_h$; $\boldsymbol{f}_o(\boldsymbol{x}, \boldsymbol{o})$ is a vector field that moves γ away from the obstacles $\boldsymbol{o}$; $\boldsymbol{f}_r(\boldsymbol{x}, \boldsymbol{r})$ is a vector field that attracts $\boldsymbol{\gamma}$ towards the points of interest $\boldsymbol{r}$; and $\boldsymbol{f}_i(\boldsymbol{x})$ is an additional vector field that is left free to the engineer in order to implement some additional internal properties of the curve that may be of interest, e.g., internal elasticity, stiffness, or viscosity.

With regard to $\boldsymbol{f}_o$, each obstacle $\boldsymbol{o}_i$, $i = 1, \dots, n_o$, implements a strictly monotonic and scalar potential $\varphi_{oi} : \mathbb{R}_0 \to \mathbb{R}^+$ such that, for a generic point $\boldsymbol{\gamma}(\boldsymbol{x}, s)$ on the path, $\varphi_{oi} \to \infty$ when $\|\boldsymbol{\gamma}(\boldsymbol{x}, s) - \boldsymbol{o}_i\| \to 0$, $\varphi_{oi} \to 0$ smoothly when $\|\boldsymbol{\gamma}(\boldsymbol{x}, s) - \boldsymbol{o}_i\| \to R_{oi}$ and $\varphi_{oi} \equiv 0$ when $R_{oi} > 0$ where $R_{oi} > 0$ defines the region of influence of the obstacle. The action of the anti-gradient vector field of φ_{oi} on the point $\boldsymbol{\gamma}(\boldsymbol{x}, s)$ is

$$\boldsymbol{f}^{\boldsymbol{p}}_{oi}(\boldsymbol{x}, s, \boldsymbol{o}_i) = -\frac{\partial \varphi(\|\boldsymbol{\gamma}(\boldsymbol{x}, s) - \boldsymbol{o}_i\|)}{\partial \boldsymbol{\gamma}(\boldsymbol{x}, s)}. \quad (11)$$

The overall action exerted on the curve by a single obstacle and projected on the shape parameters space is

$$\boldsymbol{f}_{oi}(\boldsymbol{x}, \boldsymbol{o}_i) = \int_{\boldsymbol{\gamma}} \left.\frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{x}}\right|^{\dagger}_{(\boldsymbol{x},s)} \boldsymbol{f}^{\boldsymbol{p}}_{oi}(\boldsymbol{x}, s, \boldsymbol{o}_i) ds, \quad (12)$$

where the previous considerations on the existence of the pseudo-inverse still hold. Finally, the total action of the obstacles is just the sum over all the elements in $\boldsymbol{o}$:

$$\boldsymbol{f}_o = \sum_{i=1}^{n_o} \boldsymbol{f}_{oi}(\boldsymbol{x}, \boldsymbol{o}_i), \quad (13)$$

We note that, from a practical standpoint, the analytical expression of (11) can be hard to determine, so that a numerical evaluation of the integral may be needed.

By resorting to similar arguments, we define an attractive vector field $\boldsymbol{f}_{ri}(\boldsymbol{x}, \boldsymbol{r}_i)$ for each point of interest $\boldsymbol{r}_i$, and sum each contribution in order to obtain

$$\boldsymbol{f}_r = \sum_{i=1}^{n_r} \boldsymbol{f}_{ri}(\boldsymbol{x}, \boldsymbol{r}_i). \quad (14)$$

B. Obstacle-Crossing Dynamics for the Tracked Trajectory

One drawback of using artificial potentials whose intensity becomes infinite as the distance to the obstacles goes to zero is that it does not allow the tracked trajectory to cross over an obstacle even if this could result in a smaller error norm $e(\boldsymbol{x}, \boldsymbol{x}_h) = \|\boldsymbol{x}_h - \boldsymbol{x}\|$. This limitation is a well known problem in the reactive planning literature and it has been tackled in different ways. For instance, in the elastic strip framework [2] the planner is allowed to temporarily suspend the internal forces keeping two waypoints together and then, when the obstacle is passed, restore them to rejoin the two trajectory branches. However, this formulation is limited to paths defined as a sequences of configurations and it does not guarantee that the two disjoint branches would actually cross to the other side of the obstacle.

Here we propose a procedure that, given an obstacle point $\boldsymbol{o}_i$ on one side of $\boldsymbol{\gamma}(\boldsymbol{x}, s)$, autonomously generates an alternative set of shape parameters $\boldsymbol{x}_{oi} \in \mathbb{R}^n$ such that $\boldsymbol{o}_i$ is on the other side of $\boldsymbol{\gamma}(\boldsymbol{x}_{oi}, s)$. The *alternative path* $\boldsymbol{\gamma}(\boldsymbol{x}_{oi}, s)$ is initialized and generated when $\boldsymbol{o}_i$ induces a big enough deformation on $\boldsymbol{\gamma}(\boldsymbol{x}, s)$. Introducing a threshold $F > 0$, this

[1]The ability to track a sufficiently smooth trajectory is a common property of basically all mobile robots within the scope of this work. This property holds, among the others for all those differentially flat systems [10] whose flat output includes a Cartesian point, or, equivalently, possessing a point that can be linearized through a dynamical feedback [11].

[2]We note that this approach could be seen as an extension of the classical Task-Priority framework developed for robot manipulators and recast to our particular needs, see [12] for more details.

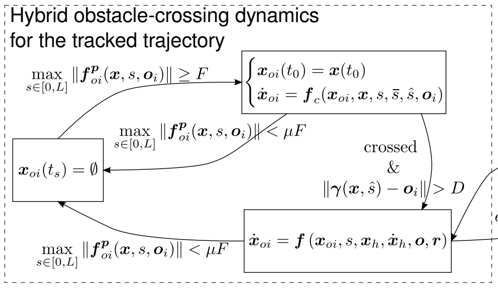

Hybrid obstacle-crossing dynamics
for the tracked trajectory
$\max_{s\in[0,L]} \|f^p_{oi}(x,s,o_i)\| \geq F$
$\begin{cases} x_{oi}(t_0) = x(t_0) \\ \dot{x}_{oi} = f_c(x_{oi}, x, s, \bar{s}, \hat{s}, o_i) \end{cases}$
$\max_{s\in[0,L]} \|f^p_{oi}(x,s,o_i)\| < \mu F$
$x_{oi}(t_s) = \emptyset$
crossed
&
$\|\gamma(x,\hat{s}) - o_i\| > D$
$\begin{cases} x_{oi}(t_s) = x(t_s) \\ x(t_s) = x_{oi}(t_s) \end{cases}$
$e(x_{oi}, x_h) < e(x, x_h)$
&
$\gamma(x,s) \simeq \gamma(x_{oi}, s)$
$\dot{x}_{oi} = f(x_{oi}, s, x_h, \dot{x}_h, o, r)$
$\max_{s\in[0,L]} \|f^p_{oi}(x,s,o_i)\| < \mu F$

Fig. 2: Block diagram of the hybrid obstacle-crossing dynamics for the tracked trajectory.

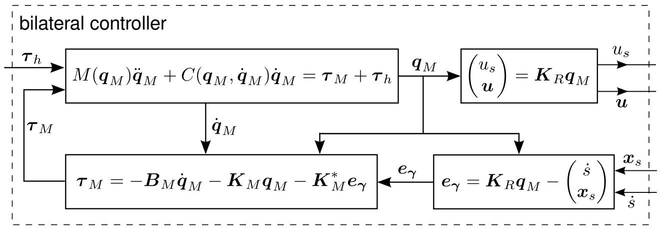

bilateral controller
τ_h
$M(q_M)\ddot{q}_M + C(q_M, \dot{q}_M)\dot{q}_M = \tau_M + \tau_h$
q_M
$\begin{pmatrix} u_s \\ u \end{pmatrix} = K_R q_M$
u_s
u
$\dot{q}_M$
τ_M
$\tau_M = -B_M \dot{q}_M - K_M q_M - K^*_M e_\gamma$
e_γ
$e_\gamma = K_R q_M - \begin{pmatrix} \dot{s} \\ x_s \end{pmatrix}$
x_s
$\dot{s}$

Fig. 3: CM: Block diagram of the bilateral controller.

condition can be expressed in terms of the repulsive force f^p_{oi} in (11) as

$$\max_{s\in[0,L]} \|f^p_{oi}(x, s, o_i)\| \geq F. \tag{15}$$

Note that, by definition, the maximum value of $\|f^p_{oi}\|$ is obtained on the closest point to the obstacle, let this be $\gamma(x, \bar{s})$, which can be computed by solving the problem

$$\bar{s} = \min_{s\in[0,L]} \|\gamma(x, s) - o_i\|. \tag{16}$$

Assume that condition (15) becomes true at time instant t_0, the alternative path is generated by letting the alternative parameters follow a dynamical system of the form:

$$\begin{cases} x_{oi}(t_0) = x(t_0) \\ \dot{x}_{oi} = f_c(x_{oi}, x, s, \bar{s}, \hat{s}, o_i). \end{cases} \tag{17}$$

In order to define $f_c \in \mathbb{R}^n$, we first need to introduce another artificial vector field $f^p_c \in \mathbb{R}^d$, i.e., acting in the Cartesian space. The role of $f^p_c \in \mathbb{R}^d$ is to apply an action that leads a single given point $\gamma(x_{oi}, \hat{s})$ of the alternative path to the other side of the obstacle. Formally, f^p_c is designed as

$$f^p_c(x_{oi}, x, s, \bar{s}, \hat{s}, o_i) = \frac{d\psi(\|\gamma(x_{oi}, s) - o_i\|)}{d\|\gamma(x_{oi}, s) - o_i\|}\, n \tag{18}$$

where ψ is a strictly increasing artificial potential and $n = \frac{o_i - \gamma(x,\bar{s})}{\|o_i - \gamma(x,\bar{s})\|}$.

In fact, the point $\gamma(x_{oi}, \hat{s})$ where the vector f^p_c is applied is given by the intersection of the geometric path $\gamma(x_{oi}, s)$ with the line connecting o_i with $\gamma(x, \bar{s})$. With the same arguments used before, the desired velocity vector f^p_c for the point $\gamma(x_{oi}, \hat{s})$ is realized by a velocity vector in the space of x_{oi} using a pseudo-inversion

$$f_c(x_{oi}, x, s, \bar{s}, \hat{s}, o_i) = \left.\frac{\partial \gamma}{\partial x}\right|^{\dagger}_{(x_{oi}, \hat{s})} f^p_c. \tag{19}$$

When the crossing is completed and $\gamma(x_{oi}, \hat{s})$ is sufficiently far from the obstacle point o_i, then the dynamical evolution of the alternative shape parameters x_{oi} switches to the normal reactive behavior

$$\dot{x}_{oi} = f(x_{oi}, s, x_h, \dot{x}_h, o, r). \tag{20}$$

At this point, the alternative collision free path $\gamma(x_{oi}, s)$ is fully generated. If at some instant t_s it results that $e(x_{oi}, x_h) < e(x, x_h)$, then the alternative path and the tracked one are exchanged, i.e.,

$$\begin{cases} x_{oi}(t_s) = x(t_s) \\ x(t_s) = x_{oi}(t_s). \end{cases} \tag{21}$$

However, the switch is allowed only if $\gamma(x, s) \simeq \gamma(x_{oi}, s)$ to avoid discontinuities in the tracked trajectory for the robot.

To complete the procedure, if the obstacle o_i gets sufficiently distant from $\gamma(x, s)$, i.e. if $\max_{s\in[0,L]} \|f^p_{oi}(x, s, o_i)\| < \mu F$ with $0 < \mu < 1$, then $\gamma(x_{oi}, s)$ is dropped. A block representation of the overall method is depicted in Fig. 2.

The generalization of this procedure to multiple obstacles is straightforward. An alternative path is generated for every obstacle that is applying a strong enough force on the current path, and whenever the current path is switched to another one, all the other alternatives are also reset. This method is not complete at every time instant, however, sequential switches allow to virtually extend the search to large part of the shape-parameter space while still keeping the problem tractable, being the number of path at every instant linear w.r.t. the number of obstacles.

III. Paradigmatic Application to Closed-loop Human-Robot Cooperation

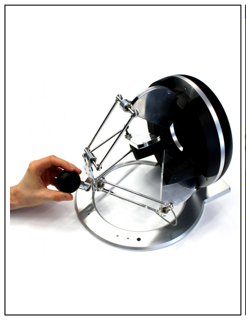
In this section we propose a systematic way to let a human assistant/operator act as MM, thus enabling a fruitful human-robot cooperation. In this case inputs $(u_s, u) \in \mathbb{R}^{m+1}$ are provided by making use of a $(m+1)$-DOF force-feedback device (the master side). The device is modeled as a generic mechanical system whose configuration vector is denoted with $q_M \in \mathbb{R}^{m+1}$, see Fig. 3. The inputs (u_s, u) are computed as

$$\begin{pmatrix} u_s \\ u \end{pmatrix} = K_R q_M, \tag{22}$$

where $K_R \in \mathbb{R}^{m+1\times m+1}$ is a positive definite diagonal matrix of scaling factors.

In order to increase the operator situational awareness the force-feedback τ_M is used to convey information about: i) how well the actual speed $\dot{s}$ is tracking $\dot{s}_h$, i.e., the one commanded by the human via u_s, and ii) how well the whole tracked path $\gamma(x)$ is in agreement with the reference path $\gamma(x_h)$ commanded by the human via u. Therefore, as a haptic cue we consider the linear combination of two errors in a 'PD-like' fashion:

$$e_\gamma = e_{\dot{y}} + k_y e_y. \tag{23}$$


(a)

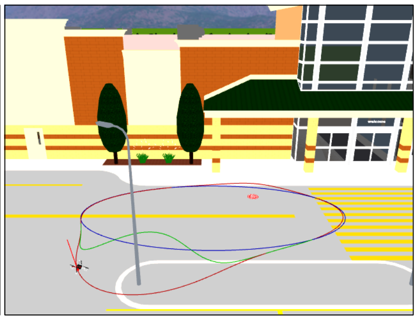

(b)

Fig. 4: simulation setup for human/hardware in the loop simulations. a): haptic device used to command the task path; b): simulation environment with planned paths.

Here, $e_{\dot{y}}$ represents the 'velocity error' part of (23) and is related to the difference between the commanded $(\dot{s}_h, \dot{x}_h)$ and the executed $(\dot{s}, \dot{x})$, i.e., exploiting (1),

$$e_{\dot{y}} = \begin{pmatrix} \dot{s}_h - \dot{s} \\ G(x_h)\dot{x}_h - G(x)\dot{x} \end{pmatrix} = K_R q_M - \begin{pmatrix} \dot{s} \\ G(x)\dot{x} \end{pmatrix}. \tag{24}$$

The second term $k_y e_y$ in (23) represents the 'position error' term and is associated to the mismatch between the desired shape x_h and its actual implementation x, i.e.,

$$e_y = \begin{pmatrix} 0 \\ G(x)\,(x_h - x) \end{pmatrix}. \tag{25}$$

The use of a force feedback based on the entire planned motion is a new feature of our approach w.r.t. to the more classical haptic-cue algorithms where only the local mismatch between the current robot velocity and the commanded one is used, instead.

The master control is then implemented as

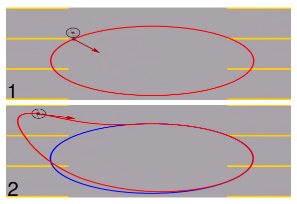
$$\tau_M = -B_M \dot{q}_M - K_M q_M - K_M^* e_\gamma \tag{26}$$

where B_M is a positive definite damping matrix used to stabilize the device, K_M is a diagonal non-negative matrix used to provide a perception of the distance from the zero-commanded velocity, and K_M^* a diagonal positive definite matrix of gains. The resulting scheme is depicted in Fig. 3.

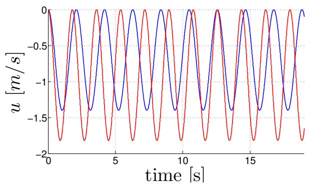
As in all bilateral teleoperation applications, presence of the force feedback τ_M may cause unstable behaviors of the haptic interface because of non-modeled dynamics, communication delays and packet losses, etc. In order to guarantee stability despite all these shortcomings, we make use of the *passive set-position modulation* (PSPM) approach, a very general and flexible framework for guaranteeing stability (passivity) of the master side and of the closed-loop system [13].

A. Human/Hardware-in-the-loop Simulations

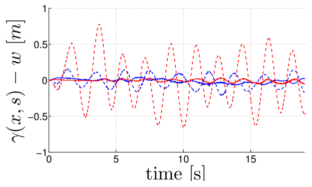
We performed several hardware/human in-the-loop simulations in order to test the proposed framework and its application to the bilateral human/robot cooperation case. The considered physically-simulated robot is a standard quadrotor UAV (see Fig. 4a), the path $\gamma(x, s)$ is parametrized as a fifth-order planar B-spline. We also asked the TS to produce a closed initial path, as described in [9], in order to show the applicability of the proposed framework to typical monitoring/surveillance scenarios where a repetitive motion is often required. The commands (u_s, u) are provided by the assistant through an Omega.6 haptic device (Fig. 4b) with 3 actuated degrees of freedom. A video of the simulations is attached.



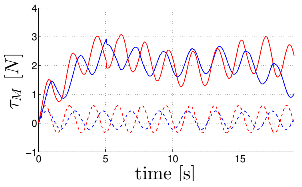
(a)



(b)



(c)

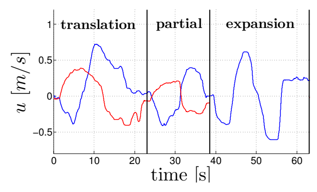


(d)

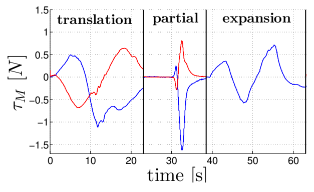
Fig. 5: Simulation comparing 1) the case of no null space projection (solid lines) and 2) when $\dot{x}$ is projected in the null space of $J = \partial\gamma/\partial x|_{(x,s)}$ (dashed lines). The red curve in (a) is the tracked trajectory while the blue one is the reference one. The red arrow represents the desired speed. In all the other plots blue and red stand for the x and y component of the signal, respectively.

In the first simulation we evaluate the effects introduced by the null space projector in (8) to keep the current position of the tracked trajectory invariant to $\dot{x}$. The MM applies a planar sinusoidal translation to the reference trajectory (Fig. 5b) in an obstacle-free environment. The case with the null-space projector (Fig. 5a-2) is compared against the case without it (Fig. 5a-1). Figure 5c shows the tracking error $\gamma(x, s) - w$, where $w \in \mathbb{R}^2$ is the planar position of the UAV. When the null space projection is applied (solid lines) the tracking error is visibly smaller than when no projection is used (dashed lines). This confirms the beneficial effects of keeping the local geometric properties of the curve in the point to be tracked. On the other hand, the deformation introduced by the projector N (see Fig. 5a-2) causes a mismatch between x and the reference x_h, even in free space. This effect is reflected on the force feedback τ_M (Fig. 5d), that becomes informative of the inertia of the path to these local changes.

The second simulation offers an example of how the MM can change the reference trajectory using different maps $G(x_h)$ such as global-translation local-translation and expansion. The MM commands are plotted in Fig. 6a, where vertical black lines indicate the change to a different command map, while the force feedback τ_M is depicted in Fig. 6b. Notice how the force feedback (Fig. 6b) when using the partial translation map (from $t = 25\,$s to $t = 38\,$s) is not null only for a short period. This is due to the fact that the map produces a local modification of the path, therefore the



(a)



(b)

Fig. 6: Second simulation showing how the changes in the reference trajectory (blue curve) affect the tracked trajectory (red curve), and generates a haptic feedback to the human assistant.

null space projection produces a mismatch only when the robot is traveling on this part.

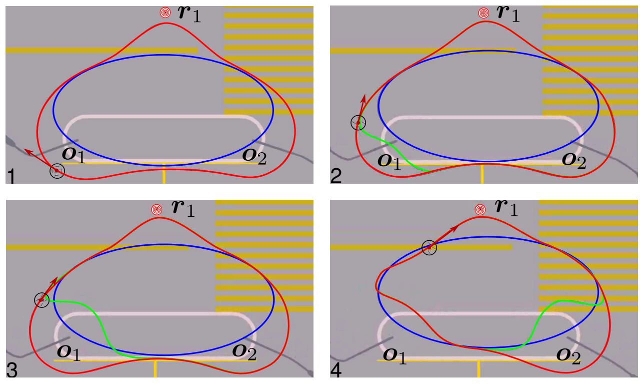
Finally, the third simulation demonstrates the behavior of the obstacle-crossing dynamics for the tracked trajectory described in Sec. II-B. Figure 7a shows four crucial moments of the simulation: 1) the tracked trajectory is deformed by the two obstacles, $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$, and attracted by the target $\boldsymbol{r}_1$; 2) condition (15) is met for $\boldsymbol{o}_1$ and the alternative path (green line) is being generated according to (19); 3) the alternative path reaches a better deformation according to (17), however the RM cannot switch to it because $\boldsymbol{\gamma}(\boldsymbol{x}, s) \neq \boldsymbol{\gamma}(\boldsymbol{x}_{o1}, s)$; 4) the RM has now switched to the alternative path, and another alternative has been generated by $\boldsymbol{o}_2$. All the switches in the replanning are denoted by solid vertical black lines in the plots of Figs. 7b-7c, while a dashed vertical line indicates that the point of interest is detected. Figure 7b shows the evolution of the average error $e(\boldsymbol{x}, \boldsymbol{x}_h) = \sum_{i=1}^{n} |\boldsymbol{x}_{h,i} - \boldsymbol{x}_i\|$. After every path switch the error becomes smaller, confirming the usefulness of the proposed crossing procedure.

Finally, Fig. 7c shows the evolution of the force feedback $\boldsymbol{\tau}_M$. As expected, the force feedback becomes stronger when the tracked trajectory is deformed by the obstacle. Note also that the discontinuities in the feedback when a switch occurs is helpful to inform the human that tracked trajectory has crossed an obstacle and can therefore continue more easily. Similarly, the sudden force that is felt when a point of interest is within range naturally guides the human operator in directing the the reference trajectory towards it.

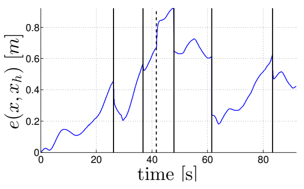
IV. Conclusions

In this work we have presented a new framework for the synergetic combination of an task scheduler, a cognitive-based planner (e.g., sophisticated algorithm or a human assistant), and a reactive (low-level) planner. We also described the design of a bilateral (haptic) connection between the human assistant and the reactive planner which benefits from a novel idea based on the deformation of the whole path. Effectiveness of the proposed approach has been demonstrated through human/hardware in-the-loop physical simulations.

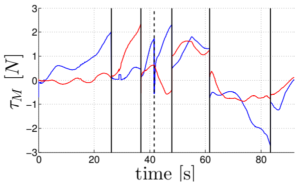
Future developments include the implementation of this framework with a real mobile robot (e.g. a UAV) the extension to a multi-robot scenario.



(a)



(b)



(c)

Fig. 7: Third simulation showing the action of the obstacle-crossing dynamics that leads the tracked trajectory (red curve) to switch to an alternative path (green) in order to better match the reference path (blue).

References

[1] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *1993 IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, May 1993, pp. 802–807.

[2] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[3] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for nonholonomic mobile robots," *IEEE Trans. on Robotics*, vol. 20, no. 6, pp. 967–977, 2004.

[4] N. Diolaiti and C. Melchiorri, "Teleoperation of a mobile robot through haptic feedback," in *IEEE Int. Work. on Haptic Virtual Environments and Their Applications*, Ottawa, Canada, Nov. 2002, pp. 62–72.

[5] D. J. Lee, O. Martinez-Palafox, and M. W. Spong, "Bilateral teleoperation of a wheeled mobile robot over delayed communication network," in *2006 IEEE Int. Conf. on Robotics and Automation*, Orlando, FL, May 2006, pp. 3298–3303.

[6] A. Franchi, P. Robuffo Giordano, C. Secchi, H. I. Son, and H. H. Bülthoff, "A passivity-based decentralized approach for the bilateral teleoperation of a group of UAVs with switching topology," in *2011 IEEE Int. Conf. on Robotics and Automation*, Shanghai, China, May 2011, pp. 898–905.

[7] A. Franchi, C. Secchi, H. I. Son, H. H. Bülthoff, and P. Robuffo Giordano, "Bilateral teleoperation of groups of mobile robots with time-varying topology," *IEEE Trans. on Robotics*, In Press, Electronically published at http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6199993.

[8] A. Franchi, C. Secchi, M. Ryll, H. H. Bülthoff, and P. Robuffo Giordano, "Shared control: Balancing autonomy and human assistance with a group of quadrotor UAVs." *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, 2012.

[9] C. Masone, A. Franchi, H. H. Bülthoff, and P. Robuffo Giordano, "Interactive planning of persistent trajectories for human-assisted navigation of mobile robots," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vilamoura, Portugal, Oct. 2012.

[10] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[11] A. Isidori, *Nonlinear Control Systems, 3rd edition*. Springer, 1995.

[12] B. Siciliano and O. Khatib, *Handbook of Robotics*. Springer, 2008.

[13] D. J. Lee and K. Huang, "Passive-set-position-modulation framework for interactive robotic systems," *IEEE Trans. on Robotics*, vol. 26, no. 2, pp. 354–369, 2010.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Dynamic Movement Primitives for Human Robot interaction

Miguel Prada and Anthony Remazeilles*

***Abstract*— A specialization of the generic *Dynamic Movement Primitives* (DMP) framework is proposed in this article to correctly address a key activity for human robot collaboration that is object exchange. As a first step towards implementing this challenging skill, this paper focuses on the arm motion to reach the initially unknown exchange site. Two improvements related with this application are proposed. First of all a better control of the transition in between the two mains components of the DMP –respectively providing a skill *shape-attractor* and a *goal-attractor*– is described, enabling to define when and how the transition in between these two components occur. Then an extension to handle situations where the goal position varies along time is proposed, which improves the convergence of the trajectory towards a moving target (i.e. the human partner's hand). These two improvements are validated by comparing the obtained behavior with human observations realized through motion capture.**

I. INTRODUCTION

The realization of robotic tasks in non completely controlled environment requires to provide the robotic system with a motion control scheme that adapts its behavior to the observed situation. Sensor-based approaches such as visual servoing [3] define the control law as a closed loop minimization of the error observed in between the current and desired visual feature values. Depending on the framework used, the robot motion can be optimal in the configuration space or in the image feature space. However, these approaches, in their basic versions, are strongly goal-driven and do not allow reproducing more complex skills in which the whole motion profile is as important as the convergence towards the goal.

The learning of complex behaviors can be addressed by *programming by demonstration* approaches, in which the robot imitates a task demonstrated either by a human operator observed with a motion capture system, or by manually moving the robot itself.Statistical approaches are frequently used for the learning. In [11], *Hidden Markov Models* are used to recognize and reproduce nine different full body expressions by a simulated humanoid. Calinon *et al.* propose in [2] to combine *Gaussian Mixture Models* and *Gaussian Mixture Regression* to reproduce several grasping tasks taught through kinesthetics. The *Dynamic Movement Primitives* (DMP) method is another approach studied in that field. Initially introduced by Ijspeert *et al.* [10], the DMP approach relies on a non-linear dynamical system forced to follow a desired trajectory by a parametric forcing term. It is proposed in this article to specialize the basic DMP framework to the special case of human-robot interaction during an object transfer.

Physical human-robot interaction, and specifically object exchange, is a key aspect to get a fluid and efficient human robot collaboration. Several recent works are focusing on this specific situation: [6] shows that the human partner can reduce the complexity of this task by adapting to the robot behavior; [8] implements different velocity profiles for the robot, and compares the results with human-human exchange procedures; direct *vs.* indirect (placing the object on a flat surface for the person to grasp) exchange procedures are compared in [4]; and [1] focuses mostly on making the robot transmit the intent of performing an exchange. In [13] the concrete exchange procedure is handled within an off-line planning scheme. The A^* algorithm is used to estimate the best trajectory to exchange the object with the human partner, based on a $3D$ cost map which combines three cost functions focused on safety, visibility and arm convenience criteria. Once the optimal exchange path is obtained, the actual trajectory to follow is computed with the Soft Motion Trajectory planner, allowing active control of maximum jerks, accelerations and velocities [12]. Nevertheless, the obtained trajectory plan is not explicitly driven by the human observation, and neither designed to adapt to the partner behavior, which is something inherent to the DMP approach proposed here. It is furthermore proved here that the initial stage of exchange location can be skipped by adapting accordingly the DMP framework.

This paper is proposing a DMP specialization for realizing human robot object exchanges. As a first step, the focus is set on the definition of the control system to bring the robotic arm towards the exchange site. Two improvements of the basic DMP framework are proposed, in relation with the exchange application. The first one is related to a better control of the transition between the feed-forward and feedback components of the DMP by introducing a custom weighing function. The second one addresses the dynamic nature of the goal position in exchange motions; a velocity based feedback term is appended to the DMP system which improves convergence with the moving goal.

This present paper is organized as follows: next section provides the needed background related to the DMP. Section III describes the two extensions proposed, and the last section compares the resulting scheme's behavior with real human-human exchange data recorded with motion capture equipment.

*This work was supported in part by the CogLaboration European project under contract FP7-ICT-7-2.1-287888, and by the Fluent National project under contract DPI 2009-13653

M. Prada and A. Remazeilles are with the Assistive Technologies Department of the Health Division of Tecnalia Research and Innovation, Parque Tecnológico de San Sebastin, E-20009 Donostia - San Sebastián, Spain

II. DYNAMIC MOVEMENT PRIMITIVES

A. Original formulation

The DMP framework learns a trajectory from just one reference sample. It can then reproduce it and optionally adapt it to different configurations. This is achieved by using a second order linear dynamical system (i.e. a damped spring-like model) which is stimulated with a non-linear forcing term. Let $x(t)$ denote a one-dimensional trajectory starting at $x(t_0) = x_0$ towards $x(t_f) = g$. In the original DMP framework the following system is introduced [9]:

$$\tau\dot{v} = K(g-x) - Dv + (g-x_0)f(s) \tag{1a}$$

$$\tau\dot{x} = v, \tag{1b}$$

with the forcing term f representing an arbitrary non-linear function as a sum of weighted exponential basis functions:

$$f(s) = \frac{\sum_{i=1}^{N} \psi_i(s)w_i}{\sum_{i=1}^{N} \psi_i(s)} s, \tag{2}$$

and:

$$\psi_i(s) = exp(-h_i(s-c_i)^2). \tag{3}$$

The above dynamical system, named *transformation system* by the authors, is composed of two driving components, aside of the global damping term $-Dv$:

- $K(g-x)$ is an attractor towards the goal position.
- $(g-x_0)f(s)$ represents the contribution of the non-linear forcing term scaled by the $g-x_0$ factor.

The variable s on which the forcing term depends is a phase variable and its evolution is determined by the following decoupled linear system, called the *canonical system*:

$$\tau\dot{s} = -\alpha s \tag{4}$$

This variable evolves exponentially from 1 to 0. It is used to remove the direct time dependency of the forcing term $f(s)$, and provides the complete system with a time scalability by adjusting the parameter τ. The phase variable is also used to weigh the forcing term, enabling this way to continuously shift towards a purely goal-attracted system.

When considering multi-dimensional trajectories, either the complete system above needs to be replicated or, as proposed in [9], a common *canonical system* can be used for all dimensions, with specific *transformation systems* for each dimension.

B. Bio-inspired formulation

In [7], Hoffmann highlights that this formulation has scaling issues when the goal position g is close to the trajectory starting point x_0. Furthermore, this model does not adapt correctly to situations where the goal parameter is set to the opposite side of the trajectory origin x_0 with the respect to the original: the complete trajectory is then completely inverted. A slightly different bio-inspired model is thus proposed, based on evidence obtained on *in vivo* studies on frogs. This modified DMP formulation is:

$$\tau\dot{v} = sK(\frac{f(s)}{s} + x_0 - x) + (1-s)K(g-x) - Dv \tag{5a}$$

$$\tau\dot{x} = v \tag{5b}$$

Similarly, this system is mainly composed of two attractor fields:

- The term $K(g-x)$ is an attractor towards the goal position (from now on referred to as the *goal-attractor*).
- The term $K(\frac{f(s)}{s} + x_0 - x)$ represents an attractor towards the moving point $\frac{f(s)}{s} + x_0$ (the *shape-attractor*).

Each of these attractor fields has its influence weighed according to the evolution of the phase variable: the *shape-attractor*, weighed by s, is predominant in the beginning of the movement, when $s \approx 1$; while the *goal-attractor*, weighed by $(1-s)$, is predominant in the end of the movement, as $s \to 0$.

This formulation bypasses the issues arising when the goal is close to the origin of the trajectory, and vastly improves the adaptation to new goals since the *shape-attractor* does not scale anymore with $(g-x_0)$. Also, the addition of the x_0 component on the *shape-attractor* enables the system to behave properly when the initial starting point is changed. These two properties together make the system affine transform-invariant when learning multi-dimensional trajectories.

C. Trajectory learning

The learning procedure is the same in both models. The first step is to give values to the parameters of the system:

- K and D involve the inherent dynamics of the second order linear system, and determine its response to on-line changes in the goal parameter.
- τ is the time constant and should be set to the duration of the sample trajectory $\tau = t_f - t_0$.
- α determines the decay rate of the phase variable. A value $\alpha \approx 4$ will ensure that $s \approx 0.02$ at $t = \tau$.

Once these values are fixed, the next step is to compute the desired values for the forcing term, by isolating it from (5a) (or (1a) for the first formulation), which results in:

$$f_{des}(s) = \frac{1}{K}(\tau\dot{v} - K(g-x) + Dv + K(g-x_0)s) \tag{6}$$

and then inserting the values of the sample trajectory $x = x(t)$, $v = \tau\dot{x}(t)$ and $\dot{v} = \tau\ddot{x}(t)$, by taking into account the nominal evolution of the phase variable $s = exp(-\frac{\alpha}{\tau}t)$.

With these desired values for the forcing term, the appropriate centers and widths of the basis exponentials in (2) can be set, and the weights w_i can be computed by fitting (2) to (6) by least squares.

D. Limitations with respect to the intended application

Both the above formulations are quite sensitive to variations in the goal from the very beginning, as illustrated on Fig. 1, where a sample trajectory $x(t)$ (black solid line) is learnt and reproduced with the goal changed from 1 to 1.5 from the beginning. In the case of the original formulation (red curve), the contribution of g in both the *shape-attractor* and *goal-attractor* (see (1a)) makes both components scale when the goal is changed. In the case of the bio-inspired formulation, as it can be seen on (5a), the *shape-attractor* is not affected by the goal parameter. Nevertheless, by studying the evolution of the phase variable (Fig. 2)one can observe that more weight is given to the *goal-attractor* for $t > 0.173\tau$ (i.e. starting at less than 20% of the trajectory duration). Thus, from this early moment, any variation of the goal with respect to the reference one has a strong effect which overrides the influence of the *shape-attractor* term.

As previously mentioned, the application we are considering is the arm control during an object exchange with a human partner. The involvement of the human in the loop requires the robotic system to deal with the exchange location uncertainty. It also naturally constraints the robot motions to be human-friendly or fluent.

One of the means to improve the fluency of object exchange is to overlap the motion of the robot with the motion of the human partner, without waiting for the human to reach a stable position to start moving. A solution to achieve this is to launch the robot motion using an estimation of the exchange site, as proposed in [13]. Nevertheless, this initial guess would still need to be adjusted on-line to adjust the robot motions to the human behavior.

To avoid this initial estimation, we are proposing to set the DMP goal to the current position of the hand of the human partner from the beginning of the movement generation. This enables to ensure the convergence towards the exchange site (which is currently assumed to be the human's final hand location). Nevertheless, from this perspective, the fact that the DMP generator is too sensitive to alterations in the goal parameter is considered as a shortcoming, since the initial goal fed to the system can be quite different to the position reached by the non predictable human partner.

In addition, the analysis of the human behavior suggests that reaching motions performed by humans contain two successive components [5]:

- The onset of the movement is performed based on imperfect target information and mostly determined by an internal dynamical model and feed-forward control.
- The final part is dominated by visual feedback control, once the target position information gets more precise.

This evidence supports the objective of initiating the movement with a dominantly feed-forward control policy,

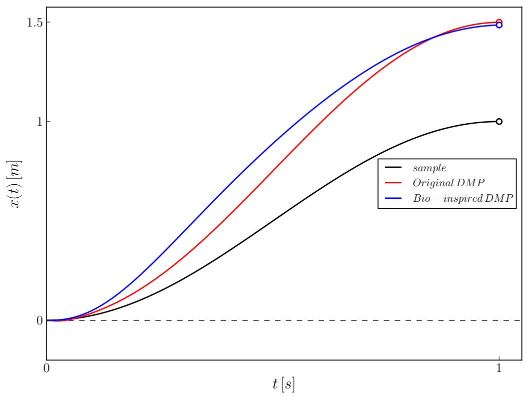


Fig. 1. Sensitivity of both the original (red curve) and the bio-inspired DMP (blue curve) generated motions with respect to a change in the goal. The black curve represents the learn trajectory.

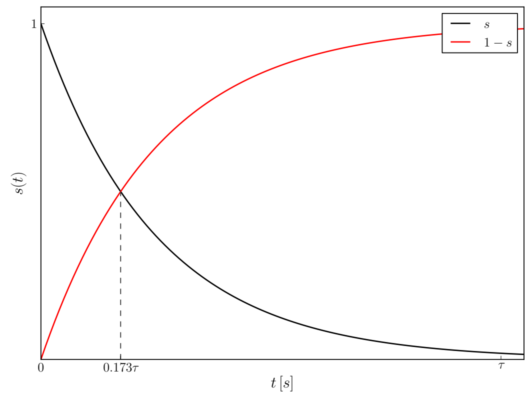


Fig. 2. Evolution of the weights of the *shape-attractor* (in black) and the *goal-attractor* (in red) within the original and bio-inspired DMP models.

and delaying the shift of weights towards the feedback component of the DMP *transformation system* to later in the trajectory. This way the first part of motion is mainly shape-driven, and less dependent on the goal variation, while the second part takes care of the convergence towards the goal. Next section presents the proposed modifications to the DMP method to achieve this desired behavior.

III. Extension of the DMP model

A. Decoupled weighing function

Two approaches are considered to modify the evolution of importance of each term driving the motion generation in the *transformation system*:

- A change in the evolution of the phase variable can change the weight balance between the two components. This can be used to delay the shift of importance from the *shape-attractor* towards the *goal-attractor*.
- A *decoupling* of the weights applied to each of the terms in the transformation system from the phase variable. Instead of weighing the attractors directly with the phase variable, an arbitrary function of the phase variable can be used to compute the desired weights.

The first approach proposed requires to find an appropriate substitute for the *canonical system* with the desired evolution, and in some cases this system might be difficult or even impossible to find without recurring to piecewise or unstable systems. Also, changing the evolution of the phase variable by means of altering the *canonical system* affects all the dimensions of the trajectory being reproduced by the DMP method.

The second approach is interesting in the sense that the *canonical system* can be kept in its original form. Furthermore, each of the *transformation systems* depending on the same phase variable can use a different weighing function if needed. Therefore it is decided to stick with this second approach which is considered more versatile.

The new system equations which use the *decoupling* approach proposed are ($f_w(s)$ and $w_g(s)$ are respectively noted f_w and w_g for notational compactness):

$$\tau\dot{v} = (1-w_g)(f_w + x_0 - x) + w_g K(g-x) - Dv \tag{7a}$$

$$\tau\dot{x} = v \tag{7b}$$

$$\tau\dot{s} = -\alpha s, \tag{7c}$$

where $f_w(s)$ is now defined as:

$$f_w(s) = \frac{\sum_{i=1}^{N} \psi_i(s) w_i}{\sum_{i=1}^{N} \psi_i(s)}. \tag{8}$$

In comparison with (2) the phase variable s is not included in the (8) anymore, since it's not longer required for the forcing term to fade away.

Note that the gain K multiplying the *shape-attractor* in (5a) has been dropped as well, since it does not have any effect at all on the system response; this is obvious by observing how the desired values of the forcing term are computed with (6).

It is proposed to use a weighing function in the shape of a sigmoid similar to the Cumulative Distribution Function (CDF) of the Normal distribution. This function has the advantage of relying on two parameters which easily allow determining when the shift will occur (the mean μ of the Normal distribution) and the duration of the shift (the standard deviation σ of the distribution). Fig. 3 shows several variations obtained by changing these two parameters. The expression for the function is, substituting the dependency on s for dependency on time:

$$w_g(t) = 0.5\left[1 + erf\left(\frac{t-\mu}{\sigma\sqrt{2}}\right)\right], \tag{9}$$

where erf stands for the Gauss error function.

This weighing function has one problem, which becomes evident when the formula for the desired shape of the forcing term $f_w(s)$ is computed. To obtain the desired $f_w(s)$ one needs to isolate it from (7a), resulting in:

$$f_w = \frac{1}{(1-w_g)}(\tau\dot{v} - w_g K(g-x) + Dv) - x_0 + x \tag{10}$$

(where the dependence on s has been dropped again for compactness). It is easy to see that this expression tends to infinity as $w_g \to 1$, thus causing numerical issues. A simple product of the sigmoid function with a linear term (e.g. starting at 0.9 for $t = 0$ and tending towards 1 as $t \to \tau$) solves this problem, while still ensuring that the *shape-attractor* influence is fading away when $s \to 0$ (i.e. $t \to \tau$). This results on the functions shown in Fig. 4.

By using the Decoupled DMP formulation proposed in (7a), (7b) and (7c), the moment where the change of goal affects the output of the DMP algorithm can be adjusted at will. Fig. 5 illustrates the behavior of the new formulation proposed. The original trajectory learnt as well as the value of the goal set during execution are the same as the one used in Fig. 1. Three trajectories are generated with different values of μ, showing how the system output is affected. In the three cases, the g parameter is set to its final value $g = 1.5$ from the beginning of the trajectory, but this only affects the trajectory at the chosen point in time. Notice that the rightmost case, with $\mu = 0.7$, switches to the *goal-attractor* too late for the trajectory to reach the goal at $t = \tau$, although it will reach it shortly after, since by that time the system is almost purely a stable linear second order system.

B. Adpatation for dynamic goals

As previously mentioned, and as a first simplification, the DMP goal is set to the position of the human partner's hand. If [13] proposes to realize an off-line estimation of the best exchange location, our approach presents the advantage of avoiding such estimation, while maintaining a reactive process so that the robot adapts to the human behavior and not the contrary.

However, in some cases, and even if the modification explained in the previous section is in place, the fact of using the human's current hand position as goal at each instant in the motion generation may introduce some undesired oscillations in the resulting trajectory. The example on Fig. 6 shows this effect with a set of data from real human motion. In this figure the black line shows the original trajectory used to learn the robot motion; the blue line shows the observed motion of the human partner, with whom the robot is performing the exchange operation; the red line represents the generated trajectory (with the DMP modifications

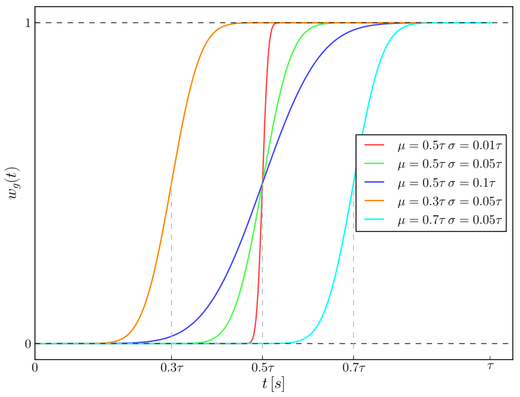


Fig. 3. Weighing function $w_g(t)$ with different sets of parameters

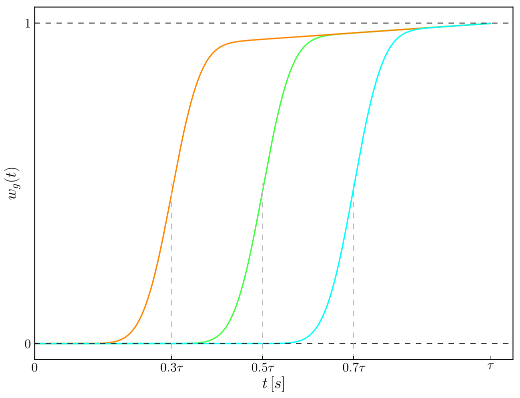


Fig. 4. Weighing function modified to avoid divide-by-zero numerical errors using the same color code as in Fig. 3

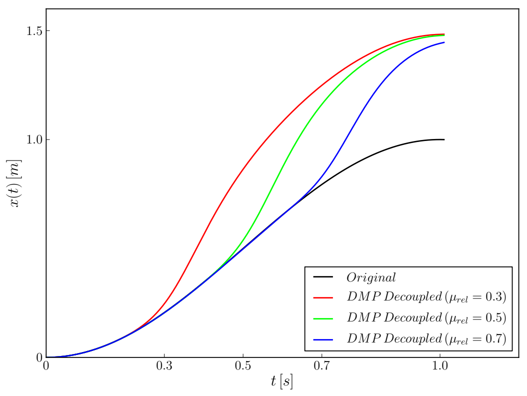


Fig. 5. Decoupled DMP with different values of μ and $\sigma = 0.05$.

presented) as response to the observed movement. It can be seen that, given that the partner's position is lagging with respect to the robot's one when the shift of weights is done in favor of the *goal-attractor*, the robot motion reverses for a certain time lapse. This oscillation is not desired, and a gentle deceleration would be much more convenient.

To alleviate this issue, a modification of the model is proposed which improves the smoothness of the convergence towards a moving goal. This modification consists in adding a velocity feedback term to the *transformation system*, resulting in:

$$\tau\dot{v} = (1 - w_g)(f_w + x_0 - x) + w_g[K(g - x) + K_v\dot{g}] - Dv \quad (11)$$

Fig. 9 on the following section shows the response trajectory generated to the same observed human motion, with the velocity feedback term in place.

IV. Experimental validation

To validate the proposed technique before implementing it onto a real robotic system, some tests have been performed on real data involving two persons exchanging different objects from different locations, as shown in Fig. 7. Markers were installed on the human bodies, mainly on the right arm of each partner (on the shoulder, elbow and hand), although in the present study only the hand markers are effectively used. Markers were tracked using a Vicon motion capture system. The DMP version presented in this paper was used to learn the three Cartesian dimensions of the right hand motion data from a selected sequence. Then data from different sequences have been used as observed human motions, and the resulting generated trajectories have been compared to the recorded response of the partner.

The resulting behavior for one specific data set is shown in Figs. 8, 9 and 10. In each of these figures the black solid line represents the sample trajectory used for learning, the blue solid line represents the data used as "observed" Human hand position, the red solid line represents the output of the proposed DMP method, the dotted blue line represents the real recorded response of the other Human partner to the movement in the solid blue line, and the solid green line shows the response of the bio-inspired DMP formulation under the same conditions. The measured positions are in millimeters, and the reference used for the data capture is located on the floor between the two users, oriented as shown in Fig. 7, where the XYZ axis are colored in RGB order.

Also, for every motion dimension being learnt the same set of parameters has been used for the weighing function: $\mu = 0.7$ and $\sigma = 0.05$.

It can be seen that the generated trajectories adjust to the observed partner trajectory without loosing the inherent dynamics of the sample trajectory from which they were learnt.

It is also evident that the trajectory generated resembles much more closely the real recorded response of the human partner than the response of the bio-inspired DMP method. This supports the idea that the previous versions of the DMP do actually require an initial estimation of the exchange location, since using the current hand position of the partner as goal creates some unpredictable and undesired effects in the motion generated, especially on Figs 8 and 9. As illustrated on these examples, The extended model we are proposing does not require such initial estimation to provide a satisfactory behavior.

V. Conclusions

This article has proposed an extension of the DMP framework to correctly learn and reproduce the human arm approach during an object transfer procedure. By changing the phase variable behavior, we obtain a better control of

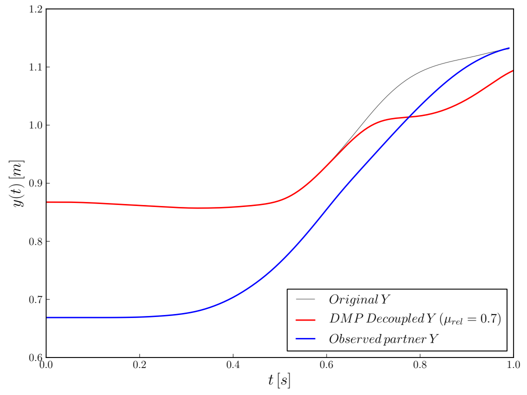


Fig. 6. Oscillation with the DMP proposed in section III-A.

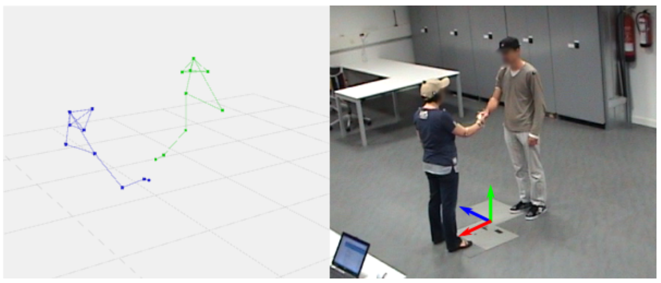


Fig. 7. Motion capture data aquired (left) and a picture of the capture sessions (right).

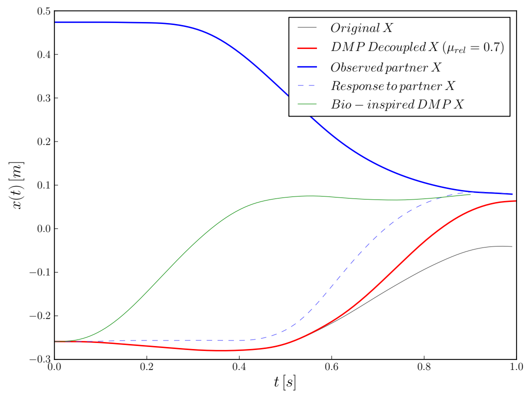


Fig. 8. Evolution of the generated trajectory in the X axis.

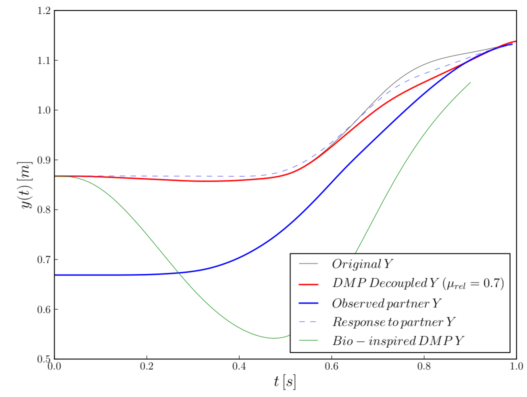


Fig. 9. Evolution of the generated trajectory in the Y axis.

the transition in between the *shape-attractor* and the *goal-attractor*, thus avoiding the need for an exchange location estimation. Furthermore, by adding in the *transformation system* a compensation for the goal velocity, the model obtained improves its convergence towards moving targets. It would be interesting to investigate how these improvements could benefit other applications of the DMP framework.

These experiments do not take yet into account the response of the human partner to the robot motion; indeed, the behavior of the human might not be equivalent when interacting with a person or with a robot. In order to complete the validation of our approach and to analyze the perception and reaction of the human when interacting with such system, at the time of writing this article, this method is being implemented onto a real robotic setup. The equipment used is a Kuka LWR robot, mounted onto a vertical structure to resemble the configuration of a human shoulder and arm; and a Kinect device to capture the motion of the human partner in front of the robot.

Finally, one of the main issues that will need to be tackled regarding such application is the triggering of the robotic motion start to get a perfect timing with the human partner. The proper implementation of such a triggering method will indeed highly influence the real time behavior of the presented technique.

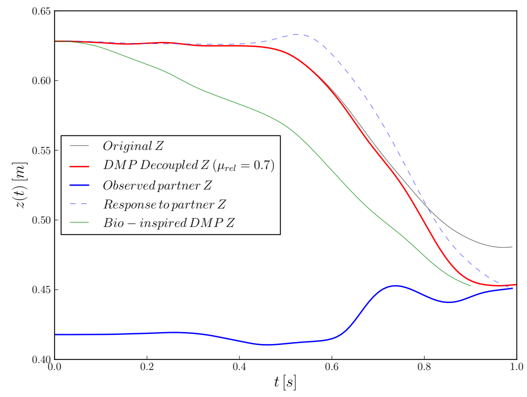


Fig. 10. Evolution of the generated trajectory in the Z axis.

References

[1] M. Cakmak, S. Srinivasa, M.K. Lee, S. Kiesler, and J. Forlizzi. Using Spatial and Temporal Contrast for Fluent Robot-Human Hand-overs. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2011.

[2] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE transactions on systems, man, and cybernetics*, 37(2):286–298, April 2007.

[3] F. Chaumette and S. Hutchinson. Visual servo control, part 2: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, March 2007.

[4] Y.S. Choi, T. Chen, A. Jain, C. Anderson, J. Glass, and C. Kemp. Hand It Over or Set It Down: A User Study of Object Delivery with an Assistive Mobile Manipulator. In *IEEE International Symposium on Robot and Human Interactive Communication*, 2009.

[5] M. Desmurget and S. Grafton. Forward modeling allows feedback control for fast reaching movements. *Trends in cognitive sciences*, 4(11):423–431, November 2000.

[6] A. Edsinger and C. Kemp. Human-Robot Interaction for Cooperative Manipulation : Handing Objects to One Another. In *IEEE International Symposium on Robot and Human Interactive Communication*, 2007.

[7] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. *IEEE International Conference on Robotics and Automation*, pages 2587–2592, May 2009.

[8] M. Huber, M. Rickert, A. Knoll, T. Brandt, and S. Glasauer. Human-Robot Interaction in Handing-Over Tasks. In *IEEE International Symposium on Robot and Human Interactive Communication*, pages 107–112, 2008.

[9] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Learning Nonlinear Dynamical Systems Models. *Neural Computation*, pages 1–33, 2010.

[10] A. Ijspeert, J. Nakanishi, T. Shibata, and S. Schaal. Nonlinear dynamical systems for imitation with humanoid robots. *IEEE-RAS International Conference on Humanoid Robots*, 2001.

[11] D. Lee and Y. Nakamura. Mimesis scheme using a monocular vision system on a humanoid robot. In *IEEE International Conference on Robotics and Automation*, pages 10–14, April 2007.

[12] E. Sisbot, L. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5), 2007.

[13] E. Sisbot, L. Marin-Urias, X. Broquere, D. Sidobre, and R. Alami. Synthesizing robot motions adapted to human presence. *International Journal of Social Robotics*, 2(3), 2010.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Reactive Humanoid Motion Planning for Reaching Tasks

Eiichi Yoshida and Fumio Kanehiro.

Abstract— **This paper addresses a reactive motion planning framework that allows a humanoid robot, supposedly teleoperated, to perform reaching tasks in complex environments with uncertainty like a damaged plant using measured information like voxel map or point clouds. Since sufficient reactiveness is required for smooth human operation, we have developed an efficient computation method to update the environment information and to plan or replan a feasible whole-body reaching path within a second when necessary. This highly responsive planning scheme benefits from rapid computation of whole-body stable posture using approximated center of gravity and analytical inverse kinematics, combined with effective representation of 3D environment using sphere tree that can be rapidly updated when environmental changes occur. We validate the proposed method in a cluttered plant environment with moving object.**

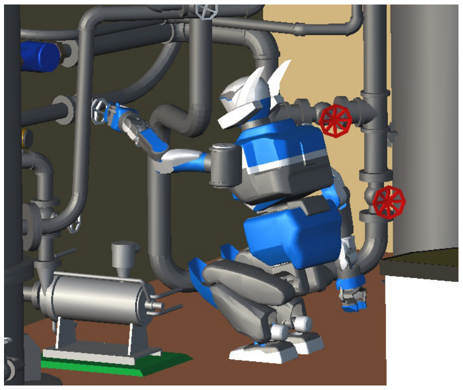


Fig. 1. Example of a reaching task in complex plant environment

I. Introduction

Probabilistic sampling-based motion planning methods have recently made great progress in its efficiency and gained strong attention in many application areas. a collision-free path that connects the initial and goal configurations is computed using a roadmap composed of nodes and edges that represent admissible configurations and local paths respectively. Two roadmap building mechanisms are identified as mainstream of sampling-based method: diffusion (e.g. Rapidly-exploring random tree, RRT) and sampling (e.g. Probabilistic RoadMap, PRM) [1], [2].

One of the most challenging applications for motion planning is the humanoid robot, which is currently expected to work to replace humans in difficult situations than ever. In hazardous environments like a damaged nuclear plant including unknown obstacles, a teleoperated humanoid seems to be a reasonable solution than fully autonomous one. In this case, the operator may want to give commands with certain abstraction level like “reach that point” or “rotate that valve”, to perform such a motion shown in Fig. 1, from a mobile or tablet interface. The robot should have minimum autonomy that can interpret and execute those commands to execute its motion. We can here benefit from the capacity of the sampling planning approach that can handle many degrees of freedom (DOFs) efficiently.

Although many general planning algorithms have been already proposed in the literature for humanoid motion planning, there are still two practical and critical problems to be addressed. The first one is time spent for the planning. Since the robot is teleoperated by a human, it is a very important factor. The planning must be done within a few seconds in order not to keep the human operator waiting too long. The second is that a polyhedral model of the environment is not given a priori. The environment is measured by sensors on the robot and its model is constructed while the robot is exploring.

For this purpose, we have recently developed an efficient motion planner [3] that can generate humanoid whole-body motion quickly in complex environments such as plants with many pipes, using approximated inverse kinematics computation guaranteeing stability and bounding volumes with sphere trees that can model measure point clouds or voxel map. As this is one-shot planning that assumes complex but fixed environments, in this research we present reactive motion planning method in changing environments.

In our previous study [4] we have proposed a reactive motion method that combines the replanning and deformation methods. Once a collision-free path is planned and starts being executed, the robot keeps executing the path as long as the path remains feasible with necessary local path deformation according to the motion of obstacles. If the executed path becomes infeasible even after deformation, the replanning is activated to find an alternative path through queries on the updated roadmap. We have validated the effectiveness of this method by applying it to redundant manipulators. However, further improvement was necessary so that the method can work with humanoid robots as the computation was still heavy.

The efficient motion planning method we have proposed [3] meets the requirements of fast computation to establish reactive planning framework for a humanoid robot to move in complex environment using measured information.

Eiichi Yoshida and Fumio Kanehiro are with CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT, and F. Kanehiro is also with Humanoid Research Group, both belonging to Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan (e.yoshida@aist.go.jp)

II. Replanning framework

A. Parallel Planning and Execution

In this section we first present briefly our reactive planning framework composed of parallel execution and planning [5] that satisfies the following specifications.

- In the event of environmental changes, if collisions are anticipated on the planned path being executed, the planner starts replanning immediately. As soon as another collision-free path is obtained again, it is executed.
- During the path replanning, the robot continues its motion unless it approaches obstacles within the specified safety distance. The path is executed in such a way that it decelerates when approaching the obstacle and makes a complete stop at a safety distance.
- There may be a case where the anticipated collision on the path is removed during the replanning. In this case, the replanning is canceled and the robot continues executing the original path without stopping.

Figure 2 illustrates how the reactive planner works with different states. It has the feature of having two "threads", Execution and Planning, running simultaneously. In Fig. 2, the texts in box correspond to the "states" of the planner. State transition occurs when a "signal" is received by the thread solid arrows in Fig. 2) or when the internal status of the thread changes (dotted arrows in Fig. 2). The signals and internal status changes causing state transition are indicated by italic and underlined texts in Fig. 2.

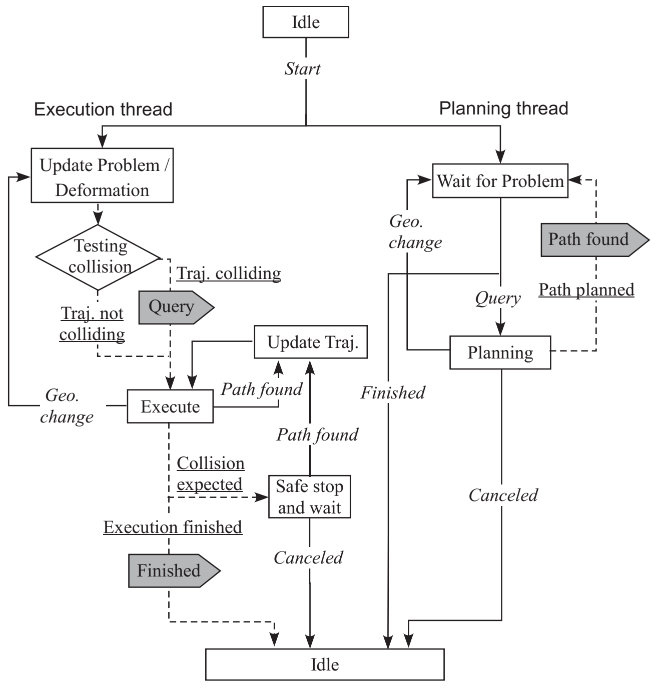


Fig. 2. State transition diagram of replanning. The replanning is performed by parallel threads of Execution and Planning that exchange signals. The states are shown in the boxes and the signal emissions are indicated by shadowed boxes. The italic and underlined texts depict the signals and internal status changes that bring about state transitions respectively.

We assume that the environmental changes can be detected by sensing mechanisms in an appropriate manner, to send "Geo. change" signals to the threads as soon as those changes are observed.

If the Execution thread at "Execute" state receives during path execution, it verifies if replanning is necessary. If it is the case, then Execution thread sends "Query" signal to the Planning thread to start the replanning. If the replanning is successful, the Execution thread updates the path and executes the replanned path without stopping. If the robot goes within safety distance to the obstacle before the planner finds a collision-free path, the Execution thread makes the robot stop by decelerating and wait for the planner to return another collision-free path. The planning fails if a feasible path is not found within the specified time. Of course there are limitations in the speed of the moving obstacles that can be avoided with respect to the robot capacity [5].

The proposed framework is implemented on RT (Robot Technology) Middleware which has been proposed as software platform [6] as a software unit called an RT Component. RT Middleware encourages the modularization and the reuse of software in the robotic field. Other software modules communicating with the motion planner, like the robot controllers and sensor systems to detect the environmental changes, can also be implemented as RT Components.

B. Roadmap Reuse

The replanning is performed based on an incrementally updated roadmap to benefit from the knowledge acquired during the previous exploration of the environment. Two kinds of roadmap, working and learning roadmaps are utilized for this replanning. As shown in Fig. 3 the learning roadmap stores the information about the environment over the whole planning time, whereas the working roadmap is continuously updated so that it includes only the valid part involved in the current replanning problem [5]. As a result, the working roadmap remains compact but reflects the most recent changes in the environment.

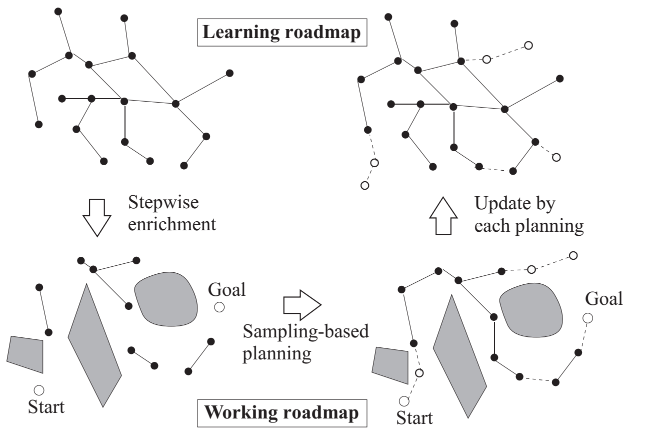


Fig. 3. Learning and working roadmaps.

III. Efficient whole-body motion planning

A brief overview of efficient whole-body humanoid motion planning introduced in [3] is provided in this section. In general, time consuming processes of the motion planning are (1) a collision detection between the robot and the environment and (2) a projection of a sampled configuration onto constrained manifolds. Since these processes are called so many times to find an initial path and optimize it, they should be done efficiently. We adopt a collision model using sphere trees and a projection that satisfies stability and kinematic constraints by maintaining approximated center of gravity (COG) position and computing arm and leg configurations with analytical inverse kinematics.

A. *Collision models using sphere tree*

We here assume that the environment around the robot is measured by sensors such as a stereo vision system or a laser range finder while the robot is exploring and those measurements are accumulated as a voxel map. Since assigning a small cube to each voxel is memory-consuming, we represent the environment by a sphere tree [7]. A sphere is assigned to each voxel with the diameter equivalent to voxel resolution. A sphere tree is composed of many spheres and is used to detect collisions during the planning and to compute distances to reshape the path to avoid collisions caused by balance compensation. The sphere tree is constructed by a top-down approach using aligned bounding box (AABB) by dividing the groups into single sphere. The robot shape is also approximated to detect collisions quickly and make it easy to compute distances. The robot shape is approximated by spheres and capped cylinders since it is easy to compute distances.

B. *Projection satisfying stability and kinematic constraints*

The configuration projection unfortunately tends to be computationally heavy because a humanoid robot must respect many constraints while reaching such as feet position/orientation and COG position. Due to high redundancy, the usual approach is to solve whole-body inverse kinematics numerically through iterative convergence computation. It is however obvious that analytical solutions of inverse kinematics should be used for quick planning.

The reaching task can be naturally defined by a goal position $\boldsymbol{p}_e = (x_e, y_e, z_e)^T$ and orientation $\boldsymbol{rpy}_e = (\phi_e, \theta_e, \psi_e)^T$ of the end effector. To compute the corresponding robot posture by projection, we define a configuration as follows.

$$\boldsymbol{q}_{goal} = [\boldsymbol{p}_e^T \ \boldsymbol{rpy}_e^T \ z_t \ \boldsymbol{rpy}_t^T] \tag{1}$$

This is concatenation of the end-effector position and orientation $\boldsymbol{p}_e$, $\boldsymbol{rpy}_e$, the height of the trunk z_t and an orientation of the trunk base $\boldsymbol{rpy}_t = (\phi_t, \theta_t, \psi_t)^T$. A sampled $\boldsymbol{q}_{goal}$ is projected so that it does not violate the stability and kinematic constraints, assuming that:

1) the whole mass concentrates on a point fixed to the trunk link at $\boldsymbol{COG}_{approx}$ as shown in Fig. 4.

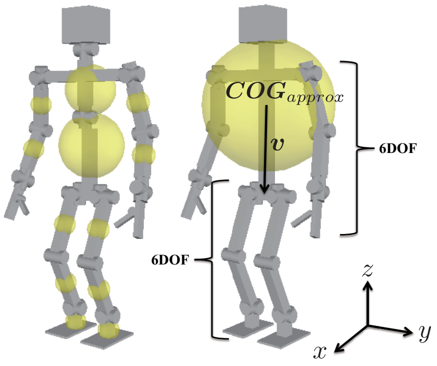


Fig. 4. The original kinematic chain(left) and the simplified kinematic chain used to find goal postures(right). Some of joints are fixed and the original kinematic chain is split into four 6DOF chains connected through the trunk. Distributing masses are assumed to be concentrating on the trunk.

2) the arms and legs are composed of six DOFs. This is the case of our humanoid robot, HRP-2 [8].

First, based on this assumption 1, we can determine the trunk horizontal position easily so that $\boldsymbol{COG}_{approx}$ does not move. This can be done by computing the trunk base position $\boldsymbol{p}_t$ from $\boldsymbol{COG}_{approx}$ based on a fixed relative vector $\boldsymbol{v}$ from $\boldsymbol{COG}_{approx}$ to the origin of the trunk link, its sampled orientation $\boldsymbol{rpy}_t$ and height z_t.

Then from this trunk base position, angles of the arms are computed by solving analytical solutions of inverse kinematics using the trunk position and orientation $\boldsymbol{p}_t$, $\boldsymbol{rpy}_t$ and end-effector position and orientation $\boldsymbol{p}_e$, $\boldsymbol{rpy}_e$. The joint angles of legs are calculated to keep the feet positions in the same way. We have verified that the error of approximation of COG is within 2[cm] in most of the cases [3] and those errors are compensated during the execution time.

C. *Simulation of reaching motion*

A reaching motion is planned using RRT-Connect [9]. The initial configuration and goals obtained by the projection are used as goals for search trees. For the reaching motion planning as well, only analytical solution of inverse kinematics is used to find solutions quickly. The configuration space for reaching motions is defined as follows:

$$\boldsymbol{q}_{plan} = [\boldsymbol{q}_{arm}^T \ z_t \ \boldsymbol{rpy}_t^T] \tag{2}$$

where $\boldsymbol{q}_{arm}$ is an array of joint angles of an arm used to reach. While RRT-Connect grows a tree, the horizontal position of the trunk base is determined in the same way the projection described above to keep the robot balance. Leg joint angles are computed by solving analytical solution of inverse kinematics as well.

IV. Reactive reaching motions

The efficient planning method introduced the previous section has been integrated into the reactive planning framework presented in Section II, using a motion planning software KineoWorks$^{\text{TM}}$ [10].

We employed the plant environment shown in Fig. 1 as an example of complex environments. In addition to the

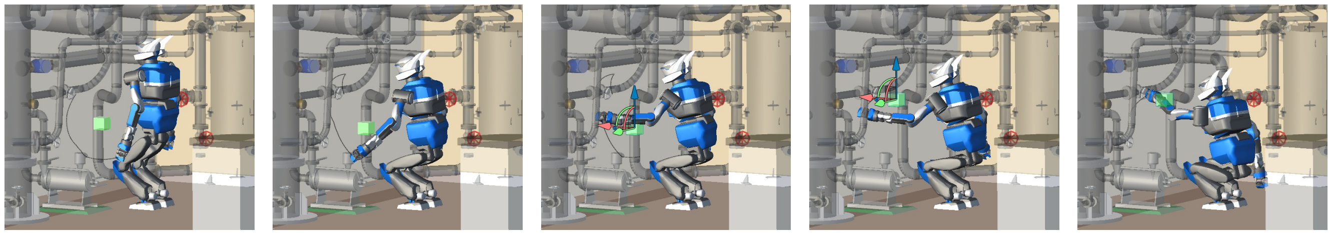

(a) (b) (c) (d) (e)

Fig. 5. Reaching motion replanned to avoid a moving obstacle in a complex environment

one-shot motion planning [3], reactive path replanning is performed in case of there are (possibly unknown) moving obstacles that are also measured as voxels or point clouds represented as sphere tree here.

As we assume all this point information comes from sensors, we actually do not have to distinguish static and moving obstacles, but we just need to update the newly measured region. The whole environment of 4m x 5m in Fig. 1 is represented by around 25,000 points with the resolution of 2 cm which are modeled as spheres of radius 1 cm. The average time required for sphere-tree model reconstruction was 28.3 ms on average with Intel processor Core i7 CPU with 2.70GHz. Although an optimal data management is preferable in case of partial changes, collision model updating is not a bottleneck in this scale of environment.

Figure 5 shows snapshots of replanning process to a valve in a plant environment, where the obstacles are displayed as transparent for better visibility. The green cube simulates an unknown or moving obstacle that is detected only when the robot gets closer to the goal. A collision-free path of reaching with the left hand is first planned as shown in Fig. 5a. When the obstacle moves downwards, new path is immediately replanned by avoiding outside (Fig. 5b, c). The obstacle finally comes upwards, which leads the replanned path to avoid underneath (Fig. 5d, e).

Figure 6 is the final configuration of the planned motion. We can observe that the left arm reaches the goal avoiding the static environment (the pipe) and moving obstacle.

The average planning time was 96 ms, including 10ms for average 2671 collision computation. With this example we can conclude that the proposed planning framework can provide a collision-free motion in a changing environment within a second for reaching tasks, which leads to comfortable teleoperation by human operator.

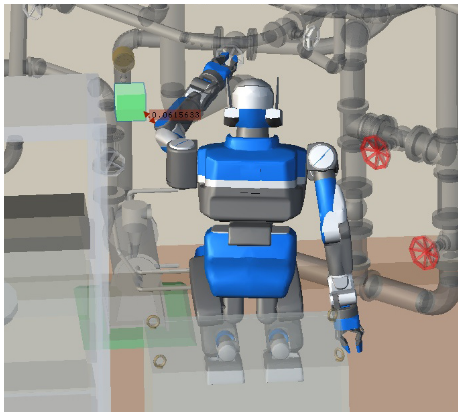


Fig. 6. Final configuration of the reaching motion

V. Conclusions

In this paper we presented an efficient reactive planning framework for a humanoid robot performing reaching tasks. We integrated an efficient environment modeling and whole-body stable configuration computation with approximated COG and analytical inverse kinematics into a reactive planning framework in changing environments. We could show that a replanning can be finished within a second even in a complex environment with moving obstacle. This provides a sufficient autonomy required for a humanoid robot that can accept high-level task commands from human operators.

In this paper we focused on the feasibility of reactive planning, and obviously path execution becomes the next important issue. By integrating the real-time execution also presented in [3], we will validate the proposed reactive planning with first realistic simulator then a physical robot.

Future work also includes extension of the proposed method to a variety of motions other than reaching, including walking or more complex tasks like object manipulation or repairing.

References

[1] H. Choset *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2006.

[2] S. LaValle, *Planning Algorithm*. Cambridge University Press, 2006.

[3] F. Kanehiro, E. Yoshida, and K. Yokoi, "Efficient reaching motion planning and execution for exploration by humanoid robots," in *Proc. 2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, in press.

[4] E. Yoshida and F. Kanehiro, "Reactive robot motion using path replanning and deformation," in *Proc. 2011 IEEE Int. Conf. on Robotics and Automation*, 2011, 5457–5462.

[5] E. Yoshida, K. Yokoi, and P. Gergondet, "Online replanning for reactive robot motion: Practical aspects," in *Proc. 2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010, 5927–5933.

[6] N. Ando *et al.*, "RT-middleware: Distributed component middleware for RT (robot technology)," in *Proc. 2005 IEEE/RSJ Int. Conf. On Intelligent Robots And Systems (IROS2005)*, 2005, 3555–3560.

[7] S. Quinlan, "Efficient distance computation between non-convex objects," in *Proc. 1994 IEEE Int. Conf. on Robotics and Automation*, 1994, 3324–3329.

[8] K. Kaneko *et al.*, "The humanoid robot HRP-2," in *Proc. 2004 IEEE Int. Conf. on Robotics and Automation*, 2004, 1083–1090.

[9] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. 2004 IEEE Int. Conf. on Robotics and Automation*, 2004, 995–1001.

[10] J.-P. Laumond, "Kineo CAM: a success story of motion planning algorithms," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, 90–93, 2006.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Efficient Motion-based Task Learning

Nicholas Malone[1], Aleksandra Faust[1], Brandon Rohrer[2], John Wood[3], Lydia Tapia[1]

Abstract— **Generating motions for robot arms in real-world complex tasks requires a combination of approaches to cope with the task structure, environmental noise, and hardware imperfections. In this paper we present an efficient framework for adaptive motion task learning on real hardware that consists of task transfer, probabilistic roadmaps (PRM), and an online reinforcement learning algorithm. Online refers to the agent making decisions and then receiving information about that decision immediately after the decision has been made, instead of receiving a complete training set. The task transfer jump starts training on the hardware with knowledge learned in simulation. To achieve faster trainings speeds we integrate a PRM with the learning agent. For motion-based task learning, we use a reinforcement learning algorithm loosely based on human cognition. We demonstrate the framework by applying it to two pointing tasks on a 7 degree of freedom Barrett Whole Arm Manipulator (WAM) robot. The first task has a stationary target and illustrates the ability of the framework to quickly adapt and compensate for hardware noise. The second task goes a step further and introduces a non-stationary target, demonstrating the framework's ability to adapt quickly to a new environment and new task.**

I. Introduction

In order to perform tasks, robots must be able to adapt to a changing environment and problems. In order to process real world information, online planning has to process higher volumes of data with tighter deadlines at every time step. The planning is subject to hardware imperfections and errors in reading sensory information. Machine learning techniques, especially online reinforcement learning (ORL) is a useful tool for robotics motion learning and planning. It provides a closed-loop feedback system continuously incorporating current environment information into the planning and producing the motions required to perform a task. However, online reinforcement learning comes with several challenges that make it potentially problematic to use on real hardware.

Implementation of an ORL algorithm must be carefully designed to be safe for the robot both in terms of collision avoidance and producing motions that don't strain hardware. Training the ORL agent from scratch on real hardware can cause wear and tear to the hardware and thus change the dynamics of the system. Furthermore, motions take longer time to execute on hardware than in simulation, and the training phase could become impractically lengthy. Lastly, the sheer size of real world state spaces and physical laws of motion that need to be processed at every time step in real-time could make ORL prohibitively computationally expensive.

We propose a framework based on ORL that successfully overcomes the challenges above and learns motion-based tasks suitable for a real robot. To jump start the learning on hardware, and avoid a lengthy training phase, we transfer the knowledge from a task trained in simulation. To achieve performance suitable for a physical system and ensure the safety of the system, we rely on probabilistic roadmaps (PRM) for dimensionality reduction. The state space information reduced by the PRM is passed to our learning agent, which learns to produce efficient motion plans. We use a Brain Emulation and Cognition Architecture (BECCA) [8] agent. It is an adaptive online reinforcement learning algorithm paired with an unsupervised hierarchical feature creator. BECCA's algorithm contains a decay feature, allowing the agent to forget features and motion plans over time. This feature is especially useful for changing environments, as the agent continuously learns and updates plans based on the current feedback from the environment.

To demonstrate the framework, we implement it on a pointing task on a 7 DoF WAM using all 7 degrees of freedom. The robot needs to autonomously learn how to point at a target location in its environment regardless of the start position. In the first series of the experiments, the target location is stationary. In the second series of experiments the target location moves. We assess the performance of the framework by measuring how well the agent adapts to hardware imperfections and measurement noise. We also examine the performance of the framework by looking into time savings obtained by using transfer learning.

Our results show near-identical performance between simulation and transferred hardware runs. We show between 100 to 600 time steps of savings obtained by using transfer learning, and demonstrate an agile agent that quickly adapts to the new environment within 500 time steps. The work here extends our previous work in [4]. Previously we utilized transfer to accelerate our experimental procedures without

[1]Nicholas Malone, Aleksandra Faust, Lydia Tapia are with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, {nmalone, afaust, tapia}@cs.unm.edu

[2] Brandon Rohrer is with the Intelligent Systems, Robotics, and Cybernetics Group, Sandia National Laboratories, PO Box 5800, Albuquerque, NM, 87185-1010, brrohre@sandia.gov

[3]John Wood is withThe Manufacturing Engineering Program, University of New Mexico, Albuquerque, NM 87131, jw@unm.edu

BECCA development was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. Tapia is supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. Partial support was provided by SNL-3 PO #1110220, for Automated Systems Research, issued to the UNM Manufacturing Engineering Program.

much discussion of the exact transfer process. Here we delve deeper into the ramifications and possibilities of transfer learning for robotics and reinforcement learning.

The rest of this paper is organized as follows: section II gives an overview of the related work. Section III discusses the hardware in more detail. Section IV discusses our methodology, and section V presents our experimental results. Finally, section VI concludes the paper with the framework's benefits to online, reactive motion-based learning.

II. Related Work

Taylor and Stone defined a taxonomy of transfer learning in the reinforcement learning domain in [10]. Using that terminology, our source task is a simulated pointing task. We have two target tasks. In one, the target task has the same goal and algorithm in both simulation and hardware runs. In the other the target is moved but it still has the same algorithm. The transferred knowledge is a set of feature groups and a Q-function.

There is active research on WAM training through human demonstration. A WAM system is represented as a canonical system of motor primitives [6]. The direct policy search class of reinforcement algorithms learns the parameters of the canonical system, while using the demonstration as an initial policy [6]. This line of research has produced a WAM capable of playing table-tennis [5], performing a ball-in-a-cup task [2], and flipping a pancake [3].

Unlike the above approaches which approximate the WAM model, the BECCA agent is agnostic to the type of the system and environment. At every time step the agent receives two signals: a sensory vector and a reward signal. The sensory vector is passed to a hierarchical feature creator. The resulting features and the reward signal are passed to the reinforcement learner.

PRMs are a method for solving complex path planning problems [1], and they tackle these complex problems by working in *conformation space* (C-space). PRMs work by building a roadmap. A roadmap is a graph where configurations are nodes and connections are edges. First, a set of configurations are sampled. Then, for each sampled node a set of candidates nodes are selected to form connections. PRMs have been extended to work in a wide variety of environments, ranging from simple open environments, to complex narrow passageways [1]. They have also been used in environments with moving obstacles [11].

III. Hardware Platform

The Barrett Whole Arm Manipulator (WAM) platform is a 7 degree of freedom (DoF) robotic arm. It is cable-driven and controlled with position encoders and torque estimation. The WAM has been connected to a GE Intelligent Platforms reflective memory network in a hub design that allows multiple computers to share memory at speeds ranging from 43 MB/s to 170 MB/s. The reflective memory networks allow remote computers to handle the planning and learning processing, while leaving a small and fast computer on-board the WAM to handle simple motion control.

IV. Methods

We propose a framework for online motion-based task learning that includes knowledge transfer from simulation to hardware. Subsection IV-A discusses transfer methodology, and subsection IV-B explains the agent in more detail. Subsection IV-C contains details on PRM implementation in our framework.

To test the performance of the framework, we implement it on a WAM and use two series of tasks: with a stationary target which is described in IV-D, and with a changing target described in IV-E.

A. Transfer Learning

Transfer learning typically refers to utilizing information learned in the past on a task in the present [10]. This past learning can be transferred to a new task or to the same task under different constraints. Transfer learning has also been utilized in transferring knowledge from one robot to another robot that may have a different internal architecture to represent the world [10]. Taylor and Stone [10] define jump start and time to threshold performance as two metrics for transfer learning. Jump start defines the amount of gain an agent initially recieves from transfered knowledge. Time to threshold performance defines the amount of time it takes an agent to reach the threshold performance, which is the best the agent can do at a given task.

In this paper, we consider a much narrower version of transfer learning. We transfer learned knowledge of a single task between a perfect simulation of a robot to imperfect robotic hardware. In simulation the robot always receives the exact same joint angles for a particular state, but in hardware the joint angles are subject to small error so re-entering the same state will not have the exact same state information. The source task uses the same learning agent, parameters, and reward function as the target task. The only difference is that the source task interacts with the WAM simulator while the target task interacts with the WAM hardware.

The WAM simulator is a simple kinetic simulator, representing the arm with seven points each corresponding to one degree of freedom. The arm moves in the simulator by simply adding the state and action vectors. The simulator does not inject noise, and performs perfect movements. The WAM arm, on the other hand, performs the movements as described in III. The resulting motion is subject to error in performing the movement.

When performing the transfer, we transfer the entire agent with all its internal states and accumulated experience. We only change the world model that it interacts with from the simulator or the WAM interface.

B. BECCA

BECCA is a general reinforcement learner [4]. It takes an input sensory vector as well as a scalar reward from the world and then produces an output action vector. Internally BECCA combines an unsupervised feature creator with a reinforcement learning agent. Figure 1 shows an overview of the architecture.

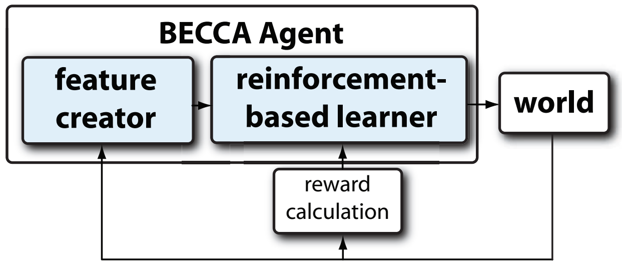


Fig. 1: BECCA's Architecture - At every time step, BECCA makes an observation in the world, extracts features from the sensory input, performs an action in response to the input, and receives a reward.

The feature creator identifies repeated patterns in the input vector and groups loosely correlated elements [7] [8]. The groups are considered to be subspaces, and the unit vectors of these subspaces are the extracted features. New inputs are projected onto each feature and the single feature, in each group with the greatest response magnitude is added to an active feature vector [8] [7] [9]. All other features with smaller response magnitudes are not added to the active feature vector. The active feature vector is then passed to the reinforcement learner, and on the next time step is fed back into the feature creator so that more complex features can be generated.

The reinforcement learner consists of a cause-effect table. The cause is the working memory, and the effect is the current active feature vector. Working memory is simply the sequence of actions that the agent has chosen in the last few time steps. The cause-effect pairs are then associated with an experienced reward. To use this model, the reinforcement learner compares its current working memory and the current active feature vector to the elements in the table. It then chooses the entry which is associated with the highest reward and takes the next action in the cause sequence.

In terms of traditional Markov Decision Process (MDP)-based reinforcement learning, the cause-effect pairs are equivalent to action-state pairs. The cause-effect table with the working memory and its expected reward roughly corresponds to a Q-function in traditional MDP-based reinforcement learning. However, BECCA's model does not assume the Markovian property and might depend on more than one previous state.

As time progresses, less frequently observed cause-effect transitions fade from the memory and the cause-effect table. This makes BECCA inherently able to adapt to new situations and environments at the cost of a steeper learning curve.

While BECCA is mostly automated, an engineer must design a task to interface with BECCA via sending sensory vectors and interpreting action vectors. Such an interface is called a task. A task simply defines what information from the world will be sent to the agent, and in what format. Note that BECCA is agnostic to the format. The task also defines how to read an action vector and move the robotic actuators. Again, note that BECCA is agnostic to how this is defined, and it will learn whatever format the engineer devises.

C. Probabilistic Roadmaps

In this paper, we use the PRMs combined with learning agent techniques from our previous work [4] to build a roadmap for the reinforcement learning agent to navigate. The learning agent is provided with the configurations and the adjacency information. It is constrained to making straight line movements along the edges in the adjacency matrix, thus constraining the reinforcement learner to learn how to navigate the roadmap. Each experiment generates a new random roadmap, except for when a transfer occurs. During a transfer, the previously learned roadmap is preserved. The PRM is the underlying state space provided to the learning agent.

D. Pointing Task with Stationary Target

The sensory vector is a n element binary vector, since the PRM contains n nodes. Each node represents a particular configuration of the robotic arm. When the robot is at a particular configuration the corresponding element in the sensory vector is set to 1.

Algorithm 1 shows how the pointing task is constructed. The action vector is a 4 element long binary vector and is parsed by the $interpret$ function. In this task, we have constrained BECCA to only return a single 1 in the action vector. The $interpret$ function in Algorithm 1 does the following: The 1 in the action vector represents BECCA selecting to move to one of the 3 neighbors, and the 4^{th} element is interpreted as staying at the current configuration. For example the action vector $[0, 1, 0, 0]$ is interpreted by the task as selection to move to the second neighbor of the current configuration in the roadmap. The function then returns the configuration of the selected neighbor.

E. Pointing Task with Non-stationary Target

The formulation and the setup of the non-stationary target task is the same as in Section IV-D. The reinforcement

Algorithm 1 Task Step

```
Require: Task
1: $Task.agent.action$ = $[0, 0, 0, 0]$
2: while $not\ coverging$ do
3:     $newLocation \leftarrow$ interpret($task.agent.action$)
4:     $sendToWAM(newLocation)$
5:     $task.currentPosition \leftarrow$ read current WAM location
6:     $task.SensoryInput \leftarrow task.currentPosition$
7:     $task.reward \leftarrow task.calculateReward()$
8:     $task.agent \leftarrow agent_{step}(SensoryInput, Reward)$;
9: end while
```

learner is trained on an initial pointing task and then transferred to hardware, however upon being transferred the goal state is changed. Thus, the learning agent must compensate for the changed goal, while learning to adapt to the dynamics of the hardware system. Specifically, for this task the goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

V. Experiments

We perform two experiments. One experiment is stationary target pointing task and the other is a non-stationary pointing task. All experimental results are averaged over five executions. Throughout the experiments, we measure the performance on the learning agent by measuring its cumulative reward. When the learning agent is transitioned from simulation to physical hardware, it is placed in a configuration that is as far as possible from the goal configuration.

We present the performance of the learning agent on hardware compared to performance in simulation. The agent executes in time steps but the graphs are shown in blocks, where 1 block equals 100 time steps. We look at the time savings brought on by using transfer learning, and the initial boost of performance that was obtained by knowledge transfer. In case of the non-stationary task, we will look at the time it takes the agent to react to a change in environment and recover to the previous level of performance

Each experimental run is executed on a new roadmap of 50 configurations generated using PRMs. Each configuration is connected to 3 neighbors and itself. A random point in the 50 configurations is chosen as the goal. The goal node is given a reward of 100. The neighbors of the goal are given a reward of 10 and the neighbors of the neighbors are given a reward of 0.1. All other configurations are given a reward of 0.

A. *Pointing Task with Stationary Target*

Figure 2 shows the cumulative reward of the pointing task with the stationary target in simulation and on hardware. The vertical line indicates the transition from the simulation to the hardware. The results show near-seamless transition, and the average performance of the agent on hardware very close to the performance in the simulation.

Table I shows the average cumulative reward for each experiment after stabilization, before and after transition to real hardware. Stabilization in simulation occurs at 20 blocks. The performance of the agent on the hardware outperforms the agent in simulation by 154 units of reward.

To better demonstrate the advantages of using the transfer learning in our framework, the pointing task with stationary target experiments were run again in a different manner. Five completely untrained learning agents were run on hardware for 20 blocks and the results averaged together. Then five agents which were trained for 100 blocks in a simulation were run on hardware for 20 more blocks and averaged together. Figure 3 shows the comparison of the stationary pointing task using transfer to the same task without using transfer. The advantages of using transfer are seen primarily in the jump start and the time to threshold metrics. Table II shows the transfer metrics for the three experiments. Jump start shows the immediate gain from using the transfer. The pointing task starts very close to the threshold performance using the transfer and has a jump start gain of 5716. In all random runs the transfered learning agent outperforms the non-transfered learning agent (Table II). Furthermore the transferred task reaches the threshold performance in 2 blocks compared to 7 blocks without transfer (Figure 3). It is important to note the time saved by using transfer learning. Table III shows the run times for simulation versus hardware for 20 blocks. It is clear that simulation is faster by up to 1 hour and 55 minutes. Using transfer learning it takes significantly less physical time on the robotic hardware for the agent to perform the given task as near optimal levels. This not only saves valuable time but it also saves valuable wear and tear on the hardware.

It is important to note that the learning algorithm is not executing pre-planned paths. It learns from experience which paths lead to highest reward and attempts to follow those paths. The paths learned in simulation provide BECCA with a strong foundation to work from, however each execution of the learning problem finds different paths due to the randomness of exploration. Thus, it is possible to witness executions of BECCA on the same underlying roadmap with slightly varying performances.

TABLE I: Average cumulative rewards in simulation and on hardware after the stabilization for 7DoF task with a stationary target and 7DoF task with a non-stationary target

Task	Simulation	Hardware
Stationary Target	7460.3	7614.8
Nonstationary Target	7460.3	7491.5

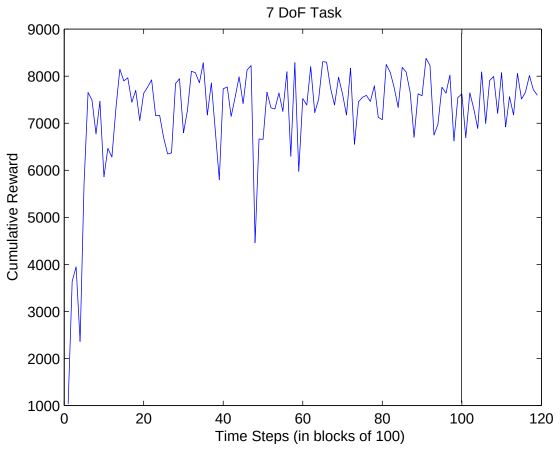


Fig. 2: Cumulative reward for the pointing task with stationary target per time step. The vertical line indicates where the learning agent was transitioned from simulation to real hardware.

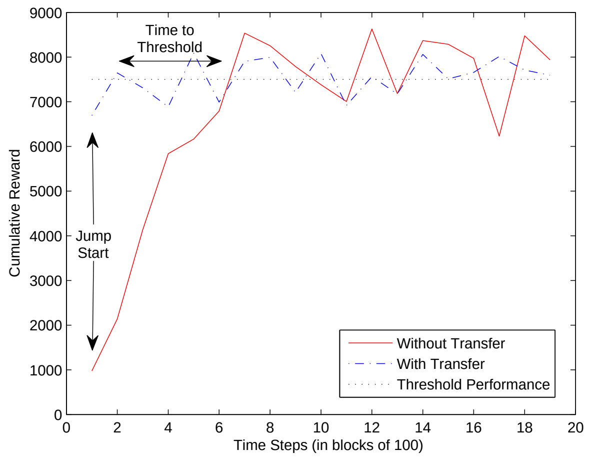


Fig. 3: Cumulative reward for the pointing task running on hardware with stationary target task with transfer and without transfer per time step. Transfer is when an agent trained in simulation is transferred to hardware. Jump start shows the initial gain obtained by using the transferred knowledge. Time to threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

TABLE II: Transfer Metrics for stationary and non stationary tasks. Jump start shows the gain from using transfer. Threshold gain shows the reduction in time steps needed to reach the threshold performance

Task	Metric	Average	min	max
Stationary	Jump Start (reward)	5716	2757	9280
	Threshold Gain (steps)	500	200	700
Non-stationary	Jump Start (reward)	1313	364	1702
	Threshold Gain (steps)	100	100	400

B. *Pointing Task with Non-stationary Target*

In this experiment the reinforcement learner is trained on an initial pointing task and then transferred to hardware. However, upon being transferred, the goal state is changed. Thus, the learning agent must compensate for the changed environment. The goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

Figure 4 shows the results of 100 blocks of simulation and then 20 blocks of running on hardware where the goal has changed. Initially there is a steep performance drop, but the reward does not drop to zero. The agent quickly recovers and learns the new reward structure within 6 blocks. This shows the online nature of the BECCA algorithm. It is able to first learn one environment and then be placed into a slightly different environment but able to compensate for the change quickly.

TABLE III: Average time in minutes to run 20 blocks in simulation and on hardware for 7DoF task with a stationary target and 7DoF task with a non-stationary target

Task	Simulation (min)	Hardware (min)
Stationary Target	23	122
Non-stationary Target	24	121

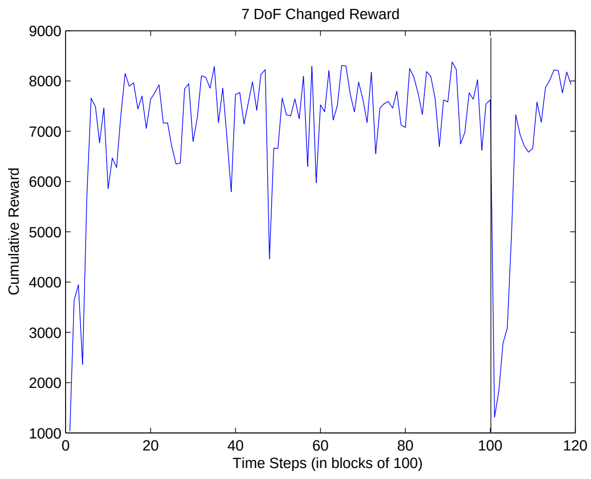


Fig. 4: Cumulative reward for running in simulation and then transferring the task to hardware. The transfer occurs at 100 blocks.

Figure 5 is a comparison between the agent having previously learned a pointing task to an agent without any prior knowledge. However, the agent with knowledge has learned to point to a different goal in simulation before being run on hardware. The untrained agent is also run on hardware but has a stationary target. Thus, the transferred agent has some information about the structure of the environment but it does not have the exact reward structure as the goal was moved before being placed on real hardware. The figure shows that the agent with prior knowledge has a small jump start of 1313 units of reward and reaches the threshold performance 1 block faster than the agent without transferred knowledge.

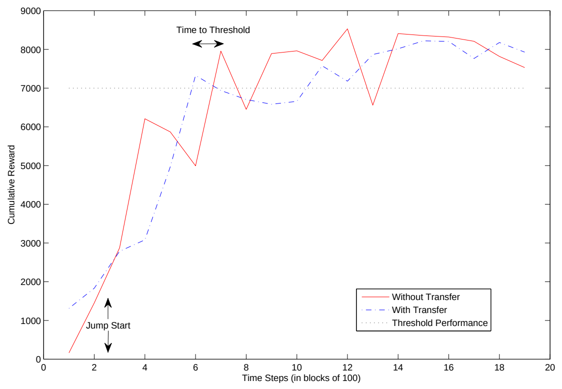


Fig. 5: Cumulative reward for the pointing task running on hardware with a non-stationary target task with transfer and without transfer per time step. Jump start shows the initial gain obtained by using the transferred knowledge. Time to threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

TABLE IV: Average Time for Convergence to Threshold Performance

Task	w/o Transfer (min)	w/ Transfer (min)
Stationary Target	40.8	11.7
Non-stationary Target	41.1	33.0

C. Timing

Timing data is collected by simply measuring the difference between start time and stop time for runs. Table III shows the timing data for running the learning algorithm in simulation versus on real hardware. The run time on hardware is approximately 5 times longer due to the amount of time it takes to for the arm to move between configurations. Each move on the WAM takes approximately 3.5 seconds to compute and execute. This computation time includes the feature extraction and action decision time for the learning algorithm. In contrast, in simulation it only takes 0.5 seconds of time to execute a complete move.

Since BECCA is an online learning algorithm, it can adapt to changes in real time. However, because it is an unsupervised learning agent it still requires repeated examples of the new environment. Thus, at 3.5 seconds an action BECCA will take around 30 minutes to adapt to a changed environment when running only on real hardware. The real time metrics that we are interested in are amount of time it takes to converge from an initial state to a the threshold performance state. This time is important because it represents the amount of time in which the robot is learning instead of performing the desired task. This metric is recorded by simply measuring the difference between the start time of run and the time of each step. Table IV shows the average time for reaching the threshold performance with and without transfer learning. This table shows that transfer learning reduces the learning time by 29 minutes for a stationary target and 8 minutes for a non-stationary target.

VI. Discussion

We demonstrated an efficient online motion-based task learning framework based on reinforcement learning that works in high-dimensional spaces in real-time, is reactive to changes in the environment, performs safe hardware motions, and efficiently learns on hardware. We demonstrated the framework by implementing it on a 7 DoF WAM using all joints to produce pointing motions with both stationary and non-stationary targets. The framework is robust and extensible to other robotics systems as well as with different model formulations, and for a large variety of tasks as well.

Dimensionality reduction and collision checks can be handled through PRMs for any motion-based task. When PRMs are used in this manner, they impose hard limits on the system. For example, self-collision states tend to be invariant to the type of environment or the task, and are good candidates to be precomputed ahead of time. When there is error in the model used for simulation caused by noisy sensor data, the robot can explore the validity of the simulation's roadmap and learn how to efficiently navigate in the physical environment.

Transfer learning can be used to avoid early learning phases when the agent's performance tends to be erratic, to reduce wear and tear on robot, and to speed up the learning process on the real hardware. It can be a powerful techniques to mitigate the long convergence times of reinforcement learning. Combining transfer learning, reinforcement learning and probabilistic roadmap methods produces a powerful framework for solving complex robotic tasks. By harnessing each method's strengths, the weaknesses of the other methods can be mitigated.

An online reinforcement learning algorithm is a suitable candidate for a planner when paired with the above techniques. Such a reinforcement learner continuously learns and updates its policy by incorporating most recent experience from the environment and produces motion plans that are adaptive, real-time, and reactive. Hardware soft limits can be implemented through the reward function.

VII. Acknowledgmetns

We would like to thank Dr. Dave Vick for his help with the robotic hardware setup.

References

[1] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[2] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21:849–856, 2009.

[3] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 3232–3237, Taipei, Taiwan, October 2010.

[4] N. Malone, B. Rohrer, L. Tapia, R. Lumia, and J. Wood. Implementation of an embodied general reinforcement learner on a serial link manipulator. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 862 –869, may 2012.

[5] K. Mülling, J. Kober, and J. Peters. A biomimetic approach to robot table tennis. *Adaptive Behavior*, 19(5):359–376, 2011.

[6] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682 – 697, 2008.

[7] B. Rohrer. Biologically inspired feature creation for multi-sensory perception. In *BICA*, 2011.

[8] B. Rohrer. A developmental agent for learning features, environment models, and general robotics tasks. In *ICDL*, 2011.

[9] B. Rohrer. Becca: Reintegrating AI for natural world interaction. In *Submitted to AAAI Spring Symposium*, 2012.

[10] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, Dec. 2009.

[11] Y. Wu. An obstacle-based probabilistic roadmap method for path planning. Master's thesis, Department of Computer Science, Texas A&M University, 1996.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Autonomous Waypoint Generation Strategy for On-Line Navigation in Unknown Environments

Sanjeev Sharma and Matthew E. Taylor

Abstract— **This paper introduces a reinforcement learning (RL) based autonomous waypoint generation strategy (AWGS), for on-line path planning in unknown environments. An RL agent intelligently analyzes its surroundings and generates waypoints within the robot's field of view. The RL agent uses an MDP for waypoint generation that is formulated to be independent of the domain, robot model, and state space dimensionality. The RL agent requires no environment-specific information beyond the robot's field of view. A path to the selected waypoint is then generated by a path planner. AWGS is applicable to many path or motion planners. However, for brevity, this paper focuses on path planning without the robot's dynamics constraint. Experiments (i) compare the performance of RL agent's policies with RRTs and $A^\star$, and (ii) show that AWGS can: (a) be trained and then used with different robot models, domains, and state-spaces, and (b) successfully navigate in environments with non-convex obstacles.**

I. Introduction

Autonomous robots are becoming increasingly common in domestic, commercial and military settings. Such robots must be able to quickly plan a safe route from their current location to the goal, while still being flexible and adaptable to unforeseeable obstacles and environmental uncertainties. Despite recent successes, e.g., the Mars rovers [1] and DARPA Urban Challenge [2], on-line navigation in unknown environments remains a challenging problem. This paper presents an autonomous waypoint generation strategy (AWGS) that uses reinforcement learning (RL) [3] for on-line navigation of mobile robots in unknown 2D- and 3D-environments. In AWGS, an RL agent analyzes the local surroundings of the robot and generates a waypoint in its field of view (FOV). An underlying path planner (ECAN [4]) then plans a path to reach the waypoint. One of the key insights is that the RL agent's MDP is formulated to be independent of the domain and space dimensionality, allowing it to generalize its waypoint generation policy across different environments. This space-independence also enables planning the waypoints in 3D, using the identical computations as in 2D, which automatically generalizes learning from 2D- to 3D-space (and vice-versa) and makes AWGS effective for the 2D- and 3D-navigation problems.

Using waypoints or sub-goals for reliable navigation in 2D-environments has been widely discussed. Shiller [5] proposed *exit-points*, fixed locations on the boundary of obstacles for on-line navigation in known environments, which is restricted to 2D-space. Krogh *et al.* [6] presented a geometrical method for selecting sub-goals corresponding to the edge and vertices of *convex polygonal* obstacles in unknown 2D-environments. Thus, the approach may not be applicable for arbitrarily shaped obstacles. Maida *et al.* [7] placed local sub-goals inside a rectangular arena of fixed dimensions, constructed around the robot, at a fixed distance from the robot and at the intersection of line segments (forming the path) avoiding the obstacles. The approach is thus limited to the extraction of waypoints along the generated path (in 2D). In contrast, AWGS intelligently selects a waypoint for the planner, which then plans a path to the waypoint, and is thus not restricted by the abilities of a planner. Wang *et al.* [8] selected midpoints between obstacles, in front of the robot, as sub-goals in 2D-environments. A robot moving in a 3D-space may pass over an obstacle instead of searching for an opening between obstacles. AWGS can select waypoints to pass in between obstacles or to go over an obstacle in 3D-space. While the previous approaches are limited to 2D-space, AWGS is applicable in both the 2D- and 3D-space.

During the last decade, sampling based planners like RRTs [9] have been successfully demonstrated in many robotics applications. RRT methods explore the environment by randomly sampling it to construct trees. On-line planning with RRTs in structured (2D-) environments (e.g., following a road) has been addressed in [10]. The constraint that the vehicle should follow a road effectively provides a direction (forward) for the expansion of RRTs. In contrast, AWGS works in unstructured (no predefined path or road), and in both 2D- and 3D-, environments by intelligently generating the waypoints in the FOV. Karaman *et al.* [11], [12] proposed RRT* for optimal on-line path planning. However, RRT* requires a complete map of the environment for building an initial feasible plan to the goal (an initial tree), before the on-line planning starts. Then during the execution of this initial plan, a *rewiring* step modifies the tree (on-line) to generate an optimal solution [11]. The requirement of an initial solution to the goal may create difficulties in planning with zero prior knowledge of an environment. On the other hand, AWGS requires no prior map and assumes no information of the environment beyond the robot's field of view.

AWGS is applicable to many planners, however, this paper focuses on using the ECAN planner [4] because: (i) ECAN's implementation is similar for both 2D- and 3D-space; (ii) ECAN guarantees collision avoidance for non-convex shaped robots, which makes the presentation of non-convex shaped robots easier; and (iii) ECAN may fall into oscillations when obstacles are non-convex, making it easier to demonstrate

Sanjeev Sharma is a graduate student in the Computing Science Dept. at University of Alberta, Canada. sanjeev1@ualberta.ca
Matthew E. Taylor is an assistant professor in the Computer Science Dept., Lafayette College, USA. taylorm@lafayette.edu

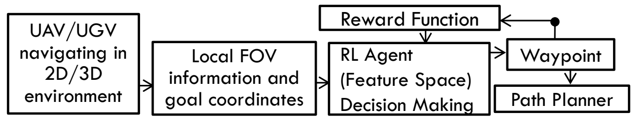

UAV/UGV navigating in 2D/3D environment
Local FOV information and goal coordinates
Reward Function
RL Agent (Feature Space) Decision Making
Waypoint
Path Planner

Fig. 1: AWGS uses domain and space independent RL agent for waypoint selection. The underlying path planner used in experiments is ECAN.

that the AWGS can overcome the shortcomings of a planner[1], and is not restricted by the abilities of an underlying planner. Also, an extension of this paper discussing the multi RL-agents approach and the planning time is available [18].

The contribution of this paper is to present a novel formulation, using RL, for waypoint generation that: (i) works identically for both the 2D- and 3D-navigation problems; and (ii) helps improve the performance of an underlying planner. The rest of this paper proceeds as follows: section-II presents background; section-II-B briefly describes ECAN; section-III describes the feature space construction and the reward function for the RL agent; and section-IV experimentally demonstrates the properties of AWGS.

II. Background

This section provides necessary background, describes the notations used in this paper and presents a selection of related work. This research assumes that the robot: (i) cannot see beyond its field of view (FOV); and (ii) knows its coordinates z_a^t, at each time-step t, and the coordinates of the goal z_g. A point-cloud of m obstacles at time-step t is represented as O^t with $z_{o_i}, i = 1, ..., m$ representing the location of i^{th} point-obstacle in the cloud. A set of n-dimensional positive definite symmetric matrices is denotes as S^n_{++}.

A. Reinforcement Learning

The underlying reinforcement learning (RL) problem of waypoint generation in AWGS is solved as an MDP [3]. In RL, a state-action value function $Q^\pi(s, a)$ for a policy π predicts the long-term reward if an agent takes action $a \in A$ in state $s \in S$ and thereafter follows policy $\pi : S \to A$. The agent's probability of taking an action a in s is given by $\pi(s, a)$. The agent's task is to learn a policy π that maximizes the total expected reward from any $s \in S$, where the reward may be discounted by a discount factor $\gamma \in [0, 1]$. By taking action a in s, agent makes a transition to state s', and receives a reward $r(s, a, s')$. The value function is approximated using a linear function approximation architecture: $Q(s, a) = \phi(s, a)^T w$, where $\phi(s, a) \in \mathbb{R}^k$ is the state-action feature vector for (s, a) and $w \in \mathbb{R}^k$ is learned using samples.

B. ECAN Navigation: Convex Programming Formulation

For navigation, ECAN forms a locally maximal ellipsoid Ψ^t, around the robot while taking the layout of local obstacles into account, oriented in such a way that favors progress towards the goal. The ellipsoid is constrained: (i) to contain the robot, (ii) to keep goal location on the boundary or outside, and (iii) to keep all the obstacles outside its boundary — ensuring collision avoidance. On time-step t, the ellipsoid Ψ^t is parameterized by variables (P^t, q^t, r^t) and is defined as $\Psi^t = \{x | x^T P^t x + x^T q^t + r^t \le 0\}$, where $P^t \in S^n_{++}, q^t \in \mathbb{R}^n, x \in \mathbb{R}^n, r^t \in \mathbb{R}$ and $n = 2$ for 2D navigation and $n = 3$ for 3D navigation. Let $z \in \mathbb{R}^n$ be an arbitrary location in the space and let $\Psi^t(z) = z^T P^t z + z^T q^t + r^t$. The ellipsoid formation problem is then solved as semi-definite programming (SDP): (α, β are trade-off parameters and I is the identity matrix with the same dimensionality as P^t)

$$\begin{aligned} &\textbf{minimize } \Psi^t(z_g) + \alpha ||\Psi^t(z_a^t)||_2 + \beta \textstyle\sum_i \Psi^t(z_{o_i}) \\ &\textbf{subject to } \Psi^t(z_a^t) \le -1;\ \Psi^t(z_g) \ge 0;\ \Psi^t(z_{o_i}) \ge 1 \\ &\qquad P^t \succeq I;\ z_{o_i} \in O^t;\ i = \{1, ..., m\}. \end{aligned}$$

The first constraint ensures that the (point-) robot lies inside Ψ^t, the second constraint ensures that the goal lies outside or on the boundary of Ψ^t, the third constraint ensures that all the obstacles in the point-cloud (O^t) lie outside Ψ^t, and the positive definite constraint on P^t ($P^t \succeq I$) ensures that Ψ^t is an ellipsoid. Objective $\Psi^t(z_g)$ orients the ellipsoid to point towards the goal; $||\Psi^t(z_a^t)||_2$ checks the ellipsoid's unbounded growth along its principal axis [4]; and $\sum_{i=1}^m \Psi^t(z_{o_i})$ forms a locally maximal ellipsoid, around the robot, bounded by surrounding obstacles. If the robot is finite (i.e., not a point mass), the first constraint is replaced by a constraint that the convex-hull of the robot should lie inside Ψ^t. Next, a navigation direction and a step-length (Δs) are computed using the obtained ellipsoid. However, the navigation direction is biased with the ellipsoid's orientation (often pointing towards the goal), potentially allowing the agent to become trapped between non-convex obstacles [4]. Within the AWGS, z_g in the SDP is replaced by the next waypoint. Thus, instead of planning to reach the goal, ECAN in AWGS always plans to reach the next waypoint.

C. Related Work: Reinforcement Learning

The RL agent in AWGS learns a domain and space independent policy. The reutilization of a policy has been widely addressed in transfer learning [13]. In this context, state of art methods typically suffer when generalizing the learned policy from one 2D-environment to another 2D-environment — at least some amount of retraining in new environment is required (see for example [14]–[17]).

III. Feature Space Construction

This section describes the computation of parameters required to construct the feature space for the RL agent. These parameters are readily available during the navigation and are computed identically in 2D- and 3D-space. Obstacles are represented using a potential map (one of the parameters of feature space). The potential map helps the RL agent to classify safe and unsafe regions for waypoints. The geometric parameters help the RL agent to make progress towards the goal, while ensuring safety. The RL agent's task is to generate a waypoint in the FOV, while taking collision avoidance into account, so that the robot can eventually reach the goal by following waypoints.

[1]AWGS has been implemented successfully for RRTs, $A^\star$, unconstrained 2D- and 3D-splines, and motion planning: (i) with Mixed Integer Programming and (ii) of a Car Like Robot: http://www.searching-eye.com/awgs.mp4

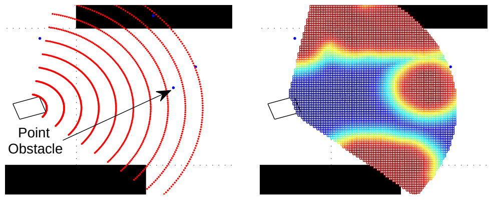


Fig. 2: (a) shows discretized grid-points in the FOV and (b) shows the corresponding potential map; red regions represent higher grid-point values.

The next section introduces the potential map. Then the geometric parameters, which allow goal-directed generation of waypoints, are discussed in section-III-B. Finally the MDP for the RL agent is formulated in section-III-C.

A. Potential Map

A local potential map represents obstacles, for the RL agent, in the robot's FOV at each time-step t. To form the potential map, the FOV is discretized. Using polar coordinate system, robot's view is restricted by $\langle R_{FOV}, \theta_{FOV} \rangle$ in 2D-space and by $\langle R_{FOV}, \theta_{FOV}, \phi_{FOV} \rangle$ in 3D-space. The FOV is thus defined by $\langle r, \theta \rangle$ and $\langle r, \theta, \phi \rangle$ in 2D- and 3D-space respectively, where $r \in (0, R_{FOV}]$, $\theta \in [-\theta_{FOV}, +\theta_{FOV}]$ and $\phi \in [-\phi_{FOV}, +\phi_{FOV}]$. The parameters dr, $d\theta$ and $d\phi$ represent discretization along the respective polar coordinates. The total number of grid-points N, in the robot's FOV is $((2\theta_{FOV}/d\theta) + 1)R_{FOV}/dr$ and $((2\theta_{FOV}/d\theta) + 1)((2\phi_{FOV}/d\phi) + 1)R_{FOV}/dr$ in 2D- and 3D-space respectively. One of these grid-points is then selected as a waypoint by the RL agent. The obstacles in the FOV are converted into an equivalent point-cloud. When obstacles are discovered in the FOV, then the grid-points lying on the faces/edges of these obstacles are added to the point-cloud O^t. Let there be m point-obstacles in the point-cloud (O^t). The potential map, for each of the N grid points, is then computed as:

$$V_q \leftarrow \max_{j:1,\ldots,m} \exp\left(\frac{-(\max\{0, ||l_q - z_{o_j}||_2 - \delta\})^2}{\sigma^2}\right)$$

where $q = \{1, ..., N\}; V_q \in [0, 1]$ is the value of q^{th} grid-point in the potential map; δ is the radius of the smallest circle encircling the robot ($\delta = 0$ for a point robot); and l_q and z_{o_j} are locations of the q^{th} grid-point and the j^{th} obstacle in the point-cloud, respectively. If $m = 0$ then $V_q = 0, \forall q = \{1, ..., N\}$. Fig. 2 shows the grid-points and corresponding potential map in 2D-space.

B. Geometric Parameters Computation: θ^{gr},ρ^{gr},ζ

After computing the potential map, three geometric parameters ($[\theta^{gr}_j, \rho^{gr}_j, \zeta_j]$) are computed for each of the grid-points in the FOV. These parameters, together with the potential map, enable environment independent generation of the waypoints, while ensuring safety. The first two parameters measure the progress towards the goal if the j^{th} grid-point is selected as the waypoint. The first parameter $\theta^{gr}_j \in [-\pi, \pi]$ is an angle between the vectors connecting z^t_a to z_g and z^t_a to the j^{th} grid-point (l_j). The second parameter is a normalized distance of the j^{th} grid-point from the goal location, $\rho^{gr}_j = d^{gr}_j/(2||z^0_a - z_g||_2)$, where $d^{gr}_j = ||z_g - l_j||_2$ and z^0_a is the robot's location at $t = 0$. Fig. 3 (left) depicts these parameters. The third parameter (ζ_j) is a Boolean variable. $\zeta_j = 1$ if a line-segment between the j^{th} grid-point and the robot's current location (z^t_a) intersects any finite obstacle; $\zeta_j = 0$ otherwise. It effectively determines the regions hidden from the robot's view — thus classifying the potentially unsafe regions (Fig. 3, right). The RL agent (as discussed in the following section) gets a large negative reward for selecting the grid-points with $\zeta_j = 1$ as the waypoints.

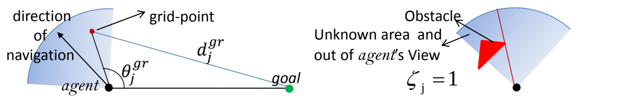


Fig. 3: The left figure shows the first two geometric parameters. The right figure shows a situation where the parameter $\zeta_j = 1$ for the grid-points which lie above the obstacle (red-triangle) and to left of the red-line.

C. Feature Space, Value Function and Reward Function

Once the parameters are computed, state-action features are computed for the RL agent. The robot's current location represents state of the RL agent at time-step t, while an action corresponds to selecting one of the grid-points as the waypoint. After the waypoint is selected, it becomes the current goal for ECAN, which then moves the robot a certain distance Δs (if the environment is uncertain) or to the waypoint (if the obstacles in the FOV are known to be static). The entire algorithm is then iterated. The $\Delta s = \min(\delta_1, \delta_2)$, where δ_1 is arbitrarily fixed and δ_2 is computed, with constraint that the robot remains inside Ψ^t, using quadratic programming (see [4]). The robot may not actually navigate to the location suggested by the current action (waypoint) of the RL agent — an action may be only partially executed. The resulting formulation may violate the Markov property, however, it is treated here as an MDP. The state-action feature vector corresponding to the j^{th} grid-point is computed as: $\Phi_j = [1, 2S(\rho^{gr}_j), \cos(\theta^{gr}_j)\exp(-\rho^{gr}_j), V_j, \zeta_j]^T$, where $S(x)$ represents the sigmoid function of $x \in \mathbb{R}$. The RL agent's policy is then learned using reinforcement learning (sarsa), with an appropriate reward function.

Reward Function: The reward function is designed such that it penalizes the RL agent for generating the waypoints in obstructed regions of the FOV ($\zeta_j = 1$) or close to the boundary of obstacles (measured by the grid-point's value in the potential map). If the RL agent selects the goal location z_g as the waypoint, it gets a large positive reward. For selecting the j^{th} grid-point as the waypoint, RL agent receives a reward:

$$r = -10^3\zeta_j - \alpha_1 V_j + \max\{\alpha_2 500, -5\}.$$

$\alpha_2 = 1$ if the waypoint is defined at the goal z_g, and -1 otherwise. The max function respectively returns $+500$ when the waypoint is at the goal and -5 otherwise, encouraging the RL agent to reach the goal in minimum possible waypoint iterations[2]. α_1 is a constant that controls penalty for defining waypoints close to obstacles.

[2]We use number of waypoint iterations (or simply, iterations) in our experiments to measure the performance of a policy.

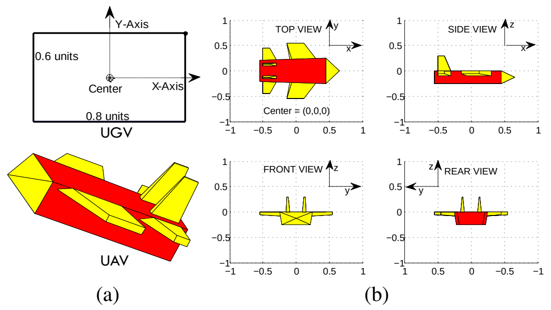


Fig. 4: Finite robot models: (a) upper- UGV with x-axis as the direction of motion; lower- UAV; (b) dimensions of UAV with the help of grid markings.

IV. Experiments

AWGS is evaluated with three categories of experiments showing: (i) domain, robot, and space independent waypoint generation; (ii) a comparison of the performance of optimal policies with RRTs and $A^\star$; and (iii) the planning capabilities in unknown complex 2D- and 3D-space. The finite robot models used in the experiments are depicted in Fig. 4. The default values of parameters are: $\langle\delta_1, \alpha_1\rangle = \langle 1, 200\rangle$; $\langle R_{FOV}, dr, \phi_{FOV}, d\phi, \sigma^2\rangle = \langle 5, 0.2, 40^\circ, 2^\circ, 0.5\rangle$; $\langle\theta_{FOV}, d\theta\rangle = \langle 60^\circ, 1^\circ\rangle$ for 2D and $\langle 40^\circ, 2^\circ\rangle$ for 3D; $\langle\alpha, \beta\rangle = \langle 0.1, 5\times10^{-4}\rangle$ in 2D and $\langle 0.1, 10^{-4}\rangle$ in 3D (trade-off parameters in the SDP in ECAN); and learning rate in SARSA [3] is 0.01 with discount factor 0.9. $A^\star$ was implemented in 2D domains by discretizing the domains in 500×500 grid-cells.

A. Robot-Model Independence and Policy Evaluation

These experiments show that the RL agent's policy is independent of the specific robot model, e.g., dimensionally different robots and FOV. The RL agent is first trained for a point-robot with default FOV parameters. 120 training episodes in the domain of Fig. 7a are used with 50 to 400 random point obstacles. An episode ends when the robot reaches the goal, or the number of waypoint iterations exceeds 200. This policy is then used as an initial policy while training the RL agent on a (2D-) finite robot (Fig. 4, UGV) with $\theta_{FOV} = 80^\circ$. Also, the RL agent learns a new policy for the finite robot from scratch. After every 5 training episodes, both policies are tested in the domain shown in Fig. 7a, with 100 random point obstacles, in 10 different start-goal configurations. In this experiment, obstacles in the FOV are assumed to be static — once the RL agent generates a waypoint, robot reaches it using ECAN, and then the next waypoint is generated. Fig. 5a – 5c show the planning results when learning with point-robot's policy as the initial policy (*transfer*) and when learning from scratch. Navigation to the goal is considered successful if the robot reaches it without colliding with any obstacle and using at most 200 waypoint iterations. If the robot collides with an obstacle, the path-length for that navigation problem is set to 500 and number of iterations is set to 200. Fig. 5a shows the average path-length, averaged over 10 start-goal configurations. Fig. 5b shows the average number of waypoint iterations required to reach the goal and Fig. 5c shows the probability of success, after every 5 training episodes, in 10 different start-goal configurations (i.e., this graph displays the number of experiments in which the robot reached the goal without collision and without violating the 200 iterations limit). These experiments show that the RL agent's policy is independent of the robot-model and FOV — re-learning for the finite-robot with the point-robot's policy (transfer) does not show any improvement. Additionally, the performance of learning from scratch converges to the performance of the point-robot's policy, requiring at least 100 training episodes.

To evaluate the performance of RL agent's optimal policy learned in this domain, the average path-length, over 10 start-goal configurations, produced by RRTs and $A^\star$ are shown in Fig. 5a. In each of the 10 start-goal configurations, RRTs were run 50 times with 10^5 RRT-iterations in each run, producing trees with an average of 75000 edges, in each of the 10 configurations. As shown, the average path-length in 10 start-goal configurations is smaller for AWGS which plans in unknown environments, as compared to RRTs exploring the entire environment. However, the paths are longer than the global $A^\star$ search. Average path-length in each of the 10 configurations is also shown in Fig. 6 (left).

B. Domain Independence and Policy Evaluation

Navigation in unknown environments requires the MDP to be domain independent. Experiments in this section show that the RL agent learns a domain-independent policy to generate waypoints. Since the RL agent's policy is robot-independent, the point-robot's policy learned for the domain in Fig. 7a is used as an initial policy for re-learning (*transfer*) in the domain of Fig. 7b, for the finite-robot. Also, a new policy is learned from scratch in this domain to compare how well the transferred policy performs in this new domain. The environment is again assumed static — the next waypoint is generated only when robot reaches the current waypoint, using ECAN. Fig. 5d – 5f compare the performance of learning with transfer and learning from scratch. Both policies are tested after every 5 training episodes, in 15 different start-goal configurations. The policy learned for the domain in Fig. 7a performs optimally in the new domain with non-convex obstacles even without any training. The new policy learned from scratch takes 100 training episodes for a similar performance, and converges to that performance — showing that the initial transfer policy is optimal. Thus, learning is domain-independent — enabling AWGS to plan in unknown environments.

Again the performance of transferred policy (which is optimal for the RL agent) is evaluated with RRTs and $A^\star$. Fig. 5d shows the average path-length over 15 start-goal configurations for RRTs and $A^\star$. Again the AWGS produces shorter paths as compared to RRTs but longer than $A^\star$. Also, the length of the paths produced by AWGS, RRTs and $A^\star$ in each of the 15 configurations is shown in Fig. 6 (right). The average for RRTs was again taken over 50 trials in each configuration, with 10^5 RRT-iterations in each trial, producing trees with an average of 68550 edges in each configuration. The paths produced by AWGS are again shorter than RRTs (for all but one configuration) even when

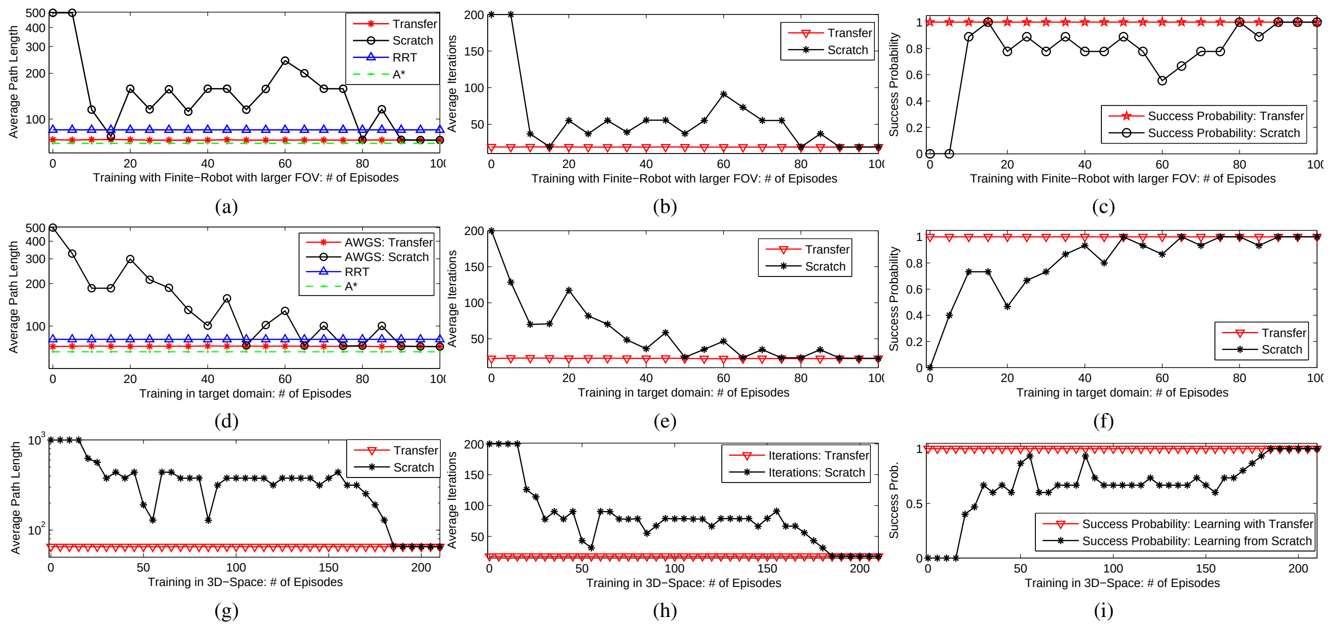


Fig. 5: a–c empirically show that 1) the RL agent's policy is robot-model independent and can be reused for dimensionally different robots and FOV, and 2) the learned policy produces paths shorter than RRTs and longer than $A^\star$. d–f show that the RL agent performs optimally in novel domain without any retraining and that the performance remains constant during additional training. The learned policy again produces shorter paths than RRTs, but produces longer path than $A^\star$. g–i show that 1) the 2D-space policy performs optimally in 3D-space even with no training in 3D and 2) learning from scratch converges to the same performance.

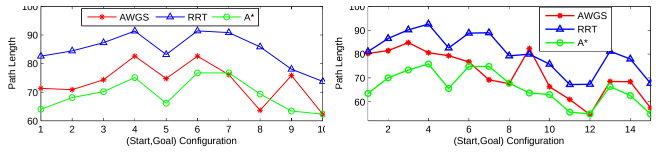


Fig. 6: Comparison of average path-length produced by RRTs, AWGS and $A^\star$ in each of the 10 and 15 configurations in convex (left) and non-convex (right) domains respectively.

planning in unknown environment, while RRTs explore the entire environment.

C. Space Independence

This section experimentally shows that the RL agent's MDP is independent of the space-dimensionality, e.g., training in 2D-space and planning in 3D-space without training from 3D-samples. As in previous experiments, two policies are learned: (i) *transfer-policy*: learning with 3D-samples using the 2D-space policy as an initial policy and (ii) *scratch-policy*: learning from scratch with 3D-samples, for the UAV (Fig. 4). Since the RL agent's policy is environment independent, the test and the training environments are the same. Episodes are defined in the same way as in previous two experiments. After every 5 training episodes, both the policies are tested in the domain shown in Fig. 8, with 15 different start-goal configurations and with 100 random point obstacles. The 2D-space policy was learned in Fig. 7b domain for 200-episodes, with 50 to 1000 random point obstacles. As shown in Fig. 5g – 5i, the transfer-policy performs optimally without any 3D training, while the scratch-policy requires 20-episodes to start reaching the goal (Fig. 5i) and at least 210-episodes to approximate the performance of transfer-policy. Additional training after transfer does not improve the policy's performance. The performance of learning from scratch again converges to the performance of transfer — the initial 2D-space policy performs optimally.

D. Planning in 2D: Convex and Non-Convex Obstacles

This section shows sample planning results and compares the performance of AWGS (using ECAN as planner) with ECAN to show that AWGS can overcome the shortcomings of a planner. Fig. 7a shows the sample trajectories planned on-line by AWGS, with finite-robot and with $\Delta s = \min(\delta_1, \delta_2)^3$, in a cluttered environment with convex obstacles and 401 random point obstacles. The robot's FOV is also shown for comparison. AWGS successfully navigates the robot to the goal(s), with RL agent defining the waypoints and convex constraints in ECAN ensuring collision avoidance for the finite-robot. Fig. 7b shows navigation with AWGS in a domain cluttered with non-convex obstacles. AWGS successfully avoids the traps in between the obstacles.

Both AWGS and ECAN alone were tested in 100 different start-goal configurations in Fig. 7b. AWGS successfully reached the goal in all runs, while ECAN failed to reach the goal because the ellipsoid always points towards the goal and the robot gets trapped in the concavity of obstacles. These experiments show that AWGS can successfully plan in unknown environments cluttered with convex or non-convex obstacles even when underlying planner is prone to local oscillations among non-convex obstacles. Also, AWGS avoids getting trapped in the concavity of obstacles even when the potential map, one of the features in the RL agent's

[3]Once the waypoint is selected by the RL agent, ECAN plans the path to navigate the robot by a distance Δs and then a new waypoint is generated.

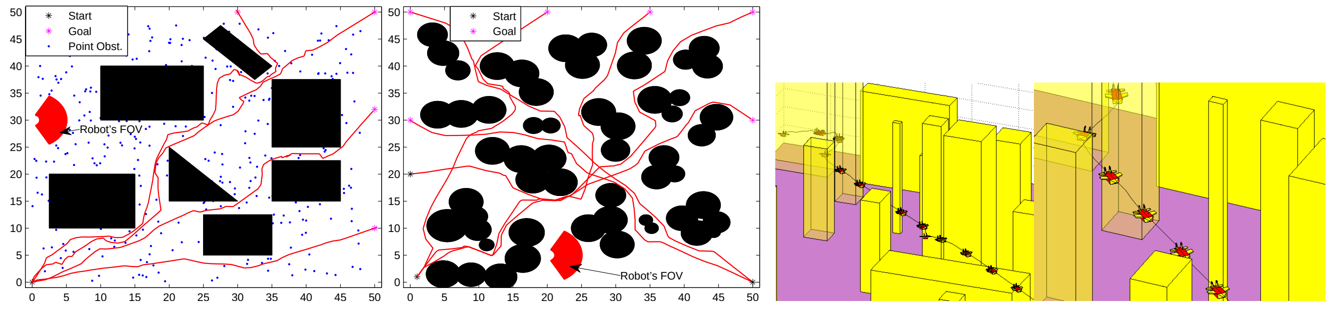


Fig. 7: (a) Sample trajectories to 4 different goals with finite robot model, (b) planning in unknown environment cluttered with non-convex obstacles; robot's FOV also shown for comparison, and (c) UAV navigating in between convex obstacles (left) and a zoomed figure (right), depicting UAV clearly.

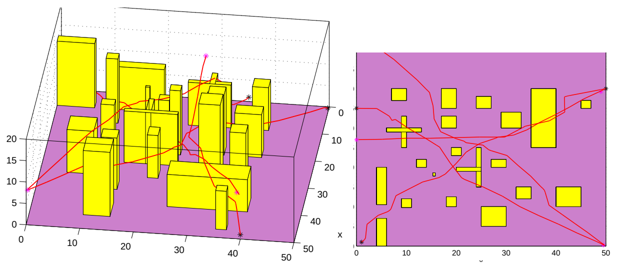


Fig. 8: Showing sample trajectories (left) planned in 3D environment with convex and non-convex obstacles along with projection on x-y (right) plane.

state-action feature vector, may succumb to local minima. The geometric parameters help in goal-directed selection of the waypoints and avoid these local minima.

E. 3D-Space Planning

This section shows successful planning with AWGS ($\Delta s = \min(\delta_1, \delta_2)$) in unknown 3D-spaces, in the presence of both convex and non-convex obstacles. Fig. 7c shows on-line planning for the UAV — the FOV parameters were again set to defaults. AWGS successfully avoids collision and plans directly with finite size UAV in 3D-space cluttered with convex obstacles. Fig. 8 shows sample planning results with AWGS in an environment cluttered with both convex and non-convex 3D-obstacles.

Again, both AWGS and ECAN alone were tested in 100 different start-goal configurations in Fig. 8 domain. AWGS successfully reached the goal in all 100 test cases, while ECAN only succeeded in 36 test cases. ECAN easily gets trapped in between the two non-convex obstacles, while AWGS, due to waypoints, easily avoids getting trapped in between the non-convex obstacles.

V. Conclusion, Discussion and Future Work

This paper presented a novel waypoint generation strategy that facilitates navigation in unknown 2D- and 3D-space. While the existing randomized planners plan once the configuration of obstacles in the environment is known, AWGS on the other hand requires no environment specific information beyond the robot's FOV and produces relatively shorter paths. The waypoint generation is independent of the robot models and space-dimensionality. This makes it possible to learn the RL agent's policy for any kind of robot model and in either 2D- or 3D-space. Future work (i) will present on-line non-holonomic motion planning in unknown environments with AWGS, and (ii) involves implementation of the AWGS on a quad-rotor flying robot and analysis of the performance with noisy robot's coordinate detection.

An implementation of AWGS in Python will be made available at: http://www.searching-eye.com/awsf/

References

[1] K.L. Wagstaff, "Smart Robots on Mars: Deciding Where to Go and What to See", in *Juniata Voices*, vol. 9, 2009.

[2] S. Thurn, "Why we compete in DARPA's Urban Challenge autonomous robot race", in *Communications of the ACM*, 2007.

[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[4] S. Sharma, "QCQP-Tunneling: Ellipsoidal Constrained Agent Navigation", in *IASTED International Conference on Robotics*, 2011.

[5] Z. Shiller, "Online Suboptimal Obstacle Avoidance", in *IJRR*, 19(5), 480-497, 2000.

[6] B. H. Krogh and D. Feng, "Dynamic Generation of Subgoals for Autonomous Mobile Robots Using Local Feedback Information", in *IEEE Transactions on Automatic Control*, 34(5), 483-493, 1989.

[7] A.S. Maida, S. Golconda, P. Mejia, A. Lakhotia and C. Cavanaugh, "Subgoal-based local navigation and obstacle avoidance using a grid-distance field", in *Int' Journal of Vehicle Autonomous Systems*, 2006.

[8] D. Wang, D. K. Lit, N. M. Kwok and K. J. Waldron, "A Subgoal-Guided Force Field Method for Robot Navigation", in *Int' Conf. on Mechatronics and Embedded Systems and Applications*, 2008.

[9] S.M. LaValle and J.J. Kuffner, "Randomized Kinodynamic Planning", in *IJRR*, 20(5), 378-400, 2001.

[10] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli and J.P. How, "Motion Planning for Urban Driving using RRT", in *ICRA*, 2008.

[11] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning", in *RSS*, 2010.

[12] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime Motion Planning using the RRT*", in *ICRA*, 2011.

[13] M.E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey", in *JMLR*, 10(1), 1633-1685, 2009.

[14] K. Ferguson and S. Mahadevan, "Proto-Transfer Learning in Markov Decision Process using Spectral Methods", in *ICML Workshop on Transfer Learning*, 2006.

[15] L. Frommberger, "Generalization and Transfer Learning in Noise-Affected Robot Navigation Tasks", in *Artificial Intelligence: EPIA Lecture Notes in Computer Science*, 308-519, 2007.

[16] L. Frommberger, "A Generalizing Spatial Representation for Robot Navigation with Reinforcement Learning", in *FLAIRS*, 2007.

[17] L. Frommberger and D. Wolter, "Structural Knowledge Transfer by Spatial Abstraction for Reinforcement Learning Agents", in *Adaptive Behavior*, 18(6), 507-525, 2010.

[18] S. Sharma and M.E. Taylor, "Autonomous Waypoint Selection for Navigation and Path Planning: A Navigation Framework for Multiple Planning Algorithms", Technical Report, 2012. http://www.searching-eye.com/awsf/awsf.pdf

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Adaptive Person-Following Algorithm Based on Depth Images and Mapping*

Guillaume Doisy[1], Aleksandar Jevtić[2], Eric Lucet[2] and Yael Edan[1]

Abstract— **Person following by a mobile autonomous robot includes two tasks, person tracking and safe robot navigation. Two person-following algorithms that use depth images from a Microsoft Kinect sensor for person tracking are proposed. The first one, the *path-following algorithm*, reproduces the path of the person in the environment. The second one, the *adaptive algorithm*, uses in addition a laser range finder for localization and dynamically generates the robot's path inside a pre-mapped environment, taking into account the obstacles locations. The Kinect was mounted on a pan-tilt mechanism to allow continuous person tracking while the robot followed the generated path. The two algorithms were tested and their performance compared in a series of trials where the robot had to follow a person walking in an environment with obstacles. With both algorithms the robot could perform continuous person tracking when the obstacles were lower than the height of the camera mount. With the adaptive algorithm the distance travelled by the robot was 29.6% shorter than with the path-following algorithm; however the path-following algorithm does not require a pre-build map of the environment.**

I. INTRODUCTION

Person following for mobile robots is advantageous in applications that require close human-robot interaction. In some applications, such as for a robot companion, having this feature is very important. There are many challenges in development of efficient and human-like person following robot behaviour, e.g., safety of humans and robots, user acceptance or ethical issues.

Person following consists of two tasks, namely person tracking and robot navigation. In real-world applications the person-following algorithms must take into account the environment constraints. For indoor applications mapping the environment allows safer and more efficient robot navigation, but often it must also consider the movement of objects and other people. In such situations, the person-following behavior must be adaptive so the robot can update the path to the desired destination point taking into account the new constraints.

Person detection and tracking is the first necessary step in the person following task. Many proposed person-following algorithms use vision as input [1], [2]. Measurements from a laser range finder (LRF) can be used to extract the patterns of the person's legs [3]; however, similar patterns can represent chairs and tables which makes correct detection difficult. To improve the tracking performance some authors proposed fusion of LRF with infrared camera [4] or with omnidirectional camera [5].

Depth images have been used for visual tracking [6]. They provide information about the distance of the objects in the image. Detection results can be improved through fusion with measurements from other sensors [7] or they can be compared with the stored templates in a pre-built knowledge base [8]. Some methods propose using input from multiple cameras [9], [10].

Recently released Microsoft Kinect sensor is a low-cost and efficient alternative for depth image-based person tracking [11], [12]. Many research groups reported their activity in working on Kinect features, but few have published their results on applications to person tracking [10], [13], SLAM [14], or improved environment mapping [15].

Person following with a mobile robot must first include person detection and tracking. Further, robot navigation and path generation are applied. The initiation of these tasks can be human-operated or autonomous [16]. Various methods for person-following propose using LRF measurements for person legs detection [17], [3]; however, detected patterns are easily confused with tables and chairs, or other people's legs. Color and texture of the person's clothes were used for vision-based tracking and following in [18], [19]. Fusion of LRF and vision-based sensors showed improved person detection [20], [21], [22], [23]. Some authors proposed combining vision-based detection with RFID tracking [24], [25].

A person-following algorithm based on direction following was proposed in [26]. The input from a pair of stereo cameras was used to combine feature detection with pre-built motion models. Another method for robot motion planning based on the learned human motion patterns was proposed in [27].

Mapping of the environment in which the robot operates can simplify the motion-planning task [28]. Mapping and SLAM for mobile robots is a vast field of study [29]. Numerous methods have been proposed based on the input from LRF [30], vision-based sensors [31], 3D images [32], [14], time-of-flight cameras [33], etc. In this paper, path planning using a pre-built map is proposed. This method is compared with a method that does not benefit from mapping, and shows how mapping can allow the robot to adapt to the distribution of obstacles.

*This research was supported by the FP7 EU-funded ITN in the Marie-Curie People Programme: INTRO, grant agreement no. 238486, and partially supported by the Paul Ivanier Center for Robotics Research and Production Management, and by the Rabbi W. Gunther Plaut Chair in Manufacturing Engineering, Ben-Gurion University of the Negev.

[1]G. Doisy and Y. Edan are with the Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer Sheva 8410, Israel doisyg@post.bgu.ac.il, yael@bgu.ac.il

[2]A. Jevtić and E. Lucet are with Robosoft, Technopole d'Izarbel, F-64210, Bidart, France {aleksandar.jevtic, eric.lucet}@robosoft.fr

II. METHODOLOGY

A. Algorithms

Two person-following algorithms are developed and compared: a *path-following algorithm* and an *adaptive algorithm*. These two algorithms use other algorithms to control the robot and to track the position of the person and estimate its position (described in Section III).

B. Hardware

The two algorithms were implemented on a generic differential drive mobile platform with two propulsive wheels and two castor wheels, which comes with basic navigation functions (Robosoft robuLAB10 robotic platform). The robuLAB10 was customized with a rigid structure including three tubes and a tray for laptop PC (Figure 1). On the top of this structure a TRACLabs Biclops pan-tilt mechanism (PT-M) and a Kinect sensor were added. For navigation purposes, the base is equipped with a SICK S300 LRF, which is positioned at the height of $0.24m$ and provides distance measurements of up to $30m$ in an angular field of view of $270°$.

The pan-tilt mechanism has a tilt range of $120°$ and a pan range of $360°$ with a maximum angular velocity of $170°/s$ and a maximum angular acceleration of $3000°/s^2$. The precision of the angular position measurements is $\pm 0.01°$. The mechanism can support a maximum payload of $4kg$ which is more than the weight of the Kinect sensor. In all experiments, the tilt value was set to $0°$ and person tracking was performed only in the horizontal plane, using the pan axis of the pan-tilt mechanism only. The communication between the laptop PC and the mechanism is maintained via a USB port with a data transfer rate of up to $416kbps$.

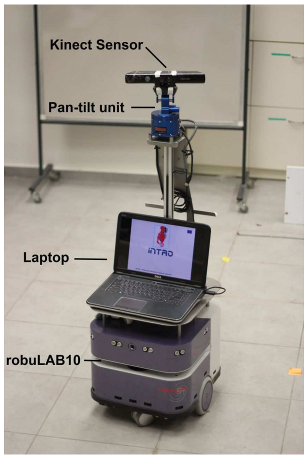


Fig. 1. RobuLAB10 robotic platform with Biclops pan-tilt mechanism, Kinect sensor, and laptop PC.

The Kinect sensor is equipped with an infrared light projector, a depth sensor, a RGB camera, and a multi-array microphone. It also has a motorized tilt that was disabled and was used only for sensor positioning. The depth sensor range is from $0.8m$ to $6m$ with the vertical viewing angle of $43°$ and horizontal viewing angle of $57°$. It provides depth images at the resolution of 640×480 pixels at the maximal frame rate of $30fps$. The Microsoft Kinect SDK provides person detection and person joints position tracking features up to $4m$.

C. Experimental setup

Two sets of experiment were conducted. The first set focused on the performance evaluation of the path-following person following algorithm and the second set focused on the adaptive person following algorithm. In all experiments, the person was instructed not to assist the robot and to walk at a constant speed along a marked path on the ground, regardless of the robot's tracking and/or following performance. This marked path on the ground makes the person travel around obstacles as seen in Figure 3 and 4.

D. Performance analysis

The following performance metrics were used for each trial of each experimental setups to evaluate the proposed person-following algorithms: 1) Path-completion ratio: the length of the ground path from the person start point to the closest point of the robot end point, divided by the total length of the ground path, 2) Number of loss-of-track events: number of events when tracking of the person was lost in a single trial; loss of tracking is defined when no position estimation is provided by the Kinect SDK for a period longer than 500ms, and 3) Robot path length to person path length ratio: the distance travelled by the robot divided by the distance travelled by the person.

For each set of experiment, 10 trials were conducted. For the path-following algorithm evaluation, the error between the person's path and robot's path was computed in addition to the metrics described above. This path error is calculated by resampling robot path data to regular space interval of 1cm and calculating for each resampled point of the robot path the closest distance to the ground path followed by the person.

III. ALGORITHMS

A. Robot control

The robuLAB10 platform uses Robosoft robuBOX open source library. The robuBOX is based on the Microsoft Robotics Developer Studio (MRDS) and written in C#. Its most important component is the Core, which contains the definitions of robots actuators and sensors. All other components interact through these definitions either by implementing or using them. For robot navigation four robuBOX features were exploited, namely the obstacle collision detection, the localization, the differential-drive controller and the path follower. The localization component uses odometry from the wheels to estimate its position and readings from a LRF continuously correct the odometry error if a map of the environment is available.

The obstacle collision detection feature uses the LRF distance measurements and applies two parameters to control the robot's motion. At distances between $0.3m$ and $1m$ from an obstacle the robot speed is reduced proportionally to the distance value. The robot is finally stopped at the distance of $0.3m$ from the obstacle. The distances are calculated within the robot frame with its origin in the point P_m located at mid-distance of the actuated wheels.

The differential-drive controller is used to set robot's linear and angular speeds. The wheels' velocities are derived from these values by the robot's low-level controller.

The path follower feature allows the robot to follow a list of path points that are added to the buffer and executed sequentially. The path follower implements Morin-Samson's path following with no orientation control [34]. We consider a path $\mathcal{C}$ in the plane of motion, as illustrated on Figure 2. Let us define three frames $\mathcal{F}_0$, $\mathcal{F}_m$, and $\mathcal{F}_s$, as follows. $\mathcal{F}_0 = \{0, \overrightarrow{i}, \overrightarrow{j}\}$ is a fixed global frame, $\mathcal{F}_m = \{P_m, \overrightarrow{i_m}, \overrightarrow{j_m}\}$ is a frame attached to the mobile robot with its origin in the point P_m, and $\mathcal{F}_s = \{P_s, \overrightarrow{i_s}, \overrightarrow{j_s}\}$, which is indexed by the path's curvilinear abscissas, is such that the unit vector $\overrightarrow{i_s}$ tangents $\mathcal{C}$. The control point P is attached to the robot chassis, with the coordinates (l_1, l_2) expressed in the basis of $\mathcal{F}_m$. In the experiments the following values were set: $l_1 = 0.15m$ and $l_2 = 0$.

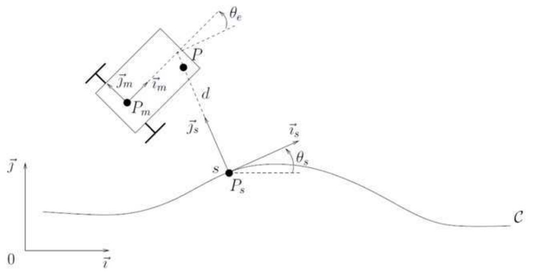


Fig. 2. Representation of the path in the robot motion plane. (Morin & Samson, 2008)

To determine the equations of motion of P with respect to the path $\mathcal{C}$ let us define d as the distance between P and $\mathcal{C}$, and $\theta_e = \theta_m - \theta_s$ as the angle characterizing the orientation of the robot chassis with respect to the frame $\mathcal{F}_s$. Where θ_m is the orientation of the robot chassis in the global frame $\mathcal{F}_0$.The control objective is to stabilize the distance d at zero. For that, the following feedback control law was applied:

$$u_2 = u_1 \left(\frac{\tan \theta_e}{l_1} - k_0 \cdot d \right) \tag{1}$$

where u_1 and u_2 represent the intensities of the robot's longitudinal and angular velocity, respectively, and k_0 is a constant. The detailed proof that d exponentially converges to zero when u_1 is constant and $\theta_e \in (-\pi/2, \pi/2)$ can be found in (Morin & Samson, 2008). The following values were set to $u_1 = 0.5m/s$ and $k_0 = 20$. As a measure of precaution, the maximal heading error was set to $\theta_{e,max} = 60°$, which initiates a recovery procedure that stops the robot and sends it to the last path point in the buffer.

B. Person tracking and position estimation

Tracking of person's skeleton joints is performed for each depth-image frame in the Kinect SDK, using no temporal information [12]. The algorithm uses the variation in depth to find different body parts and applies Random Decision Forests to compute estimated joint positions. It is also able to distinguish between two different persons. The 3D position of the head joint outputted by the algorithm was used to estimate the ground X and Y position of the person. This allows keeping track of the person position in presence of obstacles small in height causing an occlusion of the lower body parts. The outputted person ground position estimation is in the frame of reference of the Kinect sensor. It must be converted in the global frame of reference in order to be used by the path-following algorithm.

To calculate the position estimation in the global frame three direct orthonormal frames of reference were considered:

1) The fixed global frame $\mathcal{F}_0 = \{0, \overrightarrow{i_0}, \overrightarrow{j_0}\}$.
2) The frame attached to the robot $\mathcal{F}_m = \{P_m, \overrightarrow{i_m}, \overrightarrow{j_m}\}$. P_m is at the center of the robot and both $\overrightarrow{i_m}$ and $\overrightarrow{j_m}$ are in the horizontal plane; $\overrightarrow{i_m}$ is pointing in the forward direction of the robot.
3) The frame attached to the Kinect sensor $\mathcal{F}_k = \{P_k, \overrightarrow{i_k}, \overrightarrow{j_k}\}$. P_k is at the center of the Kinect sensor and both $\overrightarrow{i_k}$ and $\overrightarrow{j_k}$ are in the horizontal plane; $\overrightarrow{i_k}$ is pointing in the forward direction of the sensor.

P_m in $\mathcal{F}_0$, denoted $P_{m(\mathcal{F}_0)}$, and the angle between $\overrightarrow{i}$ and $\overrightarrow{i_m}$, denoted $\theta_{m(\mathcal{F}_0)}$, are known from odometry. P_k in $\mathcal{F}_m$, denoted $P_{k(\mathcal{F}_m)}$, is known from the hardware configuration of the robot: $P_{k(\mathcal{F}_m)} = (-0.08, 0)$. The angle between $\overrightarrow{i_m}$ and $\overrightarrow{i_k}$, denoted $\theta_{k(\mathcal{F}_m)}$, is given by the pan axis position measurement of the pan-tilt mechanism. The position of the person in $\mathcal{F}_k$, denoted $Person_{(\mathcal{F}_k)} = (X_{Person(\mathcal{F}_k)}, Y_{Person(\mathcal{F}_k)})$ is given by the output of the Kinect sensor. The angle between the forward direction of the Kinect sensor, $\overrightarrow{i_k}$, and the person, denoted $\theta_{Person(\mathcal{F}_k)}$, can be calculated:

$$\Theta_{Person(\mathcal{F}_k)} = \tan \left(\frac{Y_{Person(\mathcal{F}_k)}}{X_{Person(\mathcal{F}_k)}} \right) \tag{2}$$

The position of the person in $\mathcal{F}_m$, denoted $Person_{(\mathcal{F}_m)} = (X_{Person(\mathcal{F}_m)}, Y_{Person(\mathcal{F}_m)})$ can be calculated:

$$\begin{aligned} Person_{(\mathcal{F}_m)} = Person_{(\mathcal{F}_k)} * \\ \begin{pmatrix} \cos\left(\theta_{k(\mathcal{F}_m)}\right) & \sin\left(\theta_{k(\mathcal{F}_m)}\right) \\ -\sin\left(\theta_{k(\mathcal{F}_m)}\right) & \cos\left(\theta_{k(\mathcal{F}_m)}\right) \end{pmatrix} + \\ P_{k(\mathcal{F}_m)} \end{aligned} \tag{3}$$

The angle between the forward direction of the robot, $\overrightarrow{i_m}$, and the person, denoted $\theta_{Person(\mathcal{F}_m)}$, can be calculated:

$$\Theta_{Person(\mathcal{F}_m)} = \tan \left(\frac{Y_{Person(\mathcal{F}_m)}}{X_{Person(\mathcal{F}_m)}} \right) \tag{4}$$

Finally, the position of the person in $\mathcal{F}_0$, denoted $Person_{(\mathcal{F}_0)} = (X_{Person(\mathcal{F}_0)}, Y_{Person(\mathcal{F}_0)})$, can be calculated:

$$\begin{aligned} Person_{(\mathcal{F}_0)} = Person_{(\mathcal{F}_m)} * \\ \begin{pmatrix} \cos\left(\theta_{m(\mathcal{F}_0)}\right) & \sin\left(\theta_{m(\mathcal{F}_0)}\right) \\ -\sin\left(\theta_{m(\mathcal{F}_0)}\right) & \cos\left(\theta_{m(\mathcal{F}_0)}\right) \end{pmatrix} + \\ P_{m(\mathcal{F}_0)} \end{aligned} \tag{5}$$

C. Pan-tilt mechanism control

In order to make the Kinect sensor always point in the direction of the person tracked, a control law of the pan axis of the pan-tilt mechanism was developed. The output of this control law is an angular speed command of the pan axis, denoted $\dot{\theta}_{k(\mathcal{F}_m)(Command)}$.

A first approach to compute the speed command was to implement a P-controller using the angular position of the person in the Kinect frame, $\theta_{Person(\mathcal{F}_k)}$, as the measurement and a $0°$ angle as the target, $\theta_{Person(\mathcal{F}_k)(Target)}$.

$$\begin{aligned}\dot{\theta}_{k(\mathcal{F}_m)(P-control)} &= K_{p(Pan)} \cdot error \\ &= K_{p(Pan)} \cdot \\ &\quad \left(\theta_{Person(\mathcal{F}_k)(Target)} - \theta_{Person(\mathcal{F}_k)}\right)\end{aligned} \tag{6}$$

$\theta_{Person(\mathcal{F}_k)}$ is given by equation (2) and $\theta_{Person(\mathcal{F}_k)(Target)} = 0°$.

Then the angular speed command is set equal to the output of the P-controller:

$$\dot{\theta}_{k(\mathcal{F}_m)(Command)} = \dot{\theta}_{k(\mathcal{F}_m)(P-control)} \tag{7}$$

We used $K_{p(Pan)} = 4\ s^{-1}$. This first approach using equation (6) for computing the speed command is able to maintain the sensor in the direction of the tracked person when the robot is not moving. However, when the robot is moving, the system is not reactive enough to keep track of the person. Loss of tracking happens when the robot is rotating or turning. To compensate for the robot rotation, a second approach was developed. Information from the odometry pose estimation is used to calculate the angular speed of the robot in $\mathcal{F}_0$, denoted $\dot{\theta}_{m(\mathcal{F}_0)}$, from two successive measurements of the robot orientation in the global frame: $\theta_{m(\mathcal{F}_0)(t-1)}$ and $\theta_{m(\mathcal{F}_0)(t)}$.

$$\dot{\theta}_{m(\mathcal{F}_0)} = \frac{\theta_{m(\mathcal{F}_0)(t)} - \theta_{m(\mathcal{F}_0)(t-1)}}{T(t) - T(t-1)} \tag{8}$$

where $T(t)$ and $T(t-1)$ are the time of the current measurement of angular speed and the time of the previous measurement of angular speed, respectively.

Using the additive inverse of the angular speed yields a robot rotation compensation speed command.

Finally, the speed command to send to the pan axis of the pan-tilt mechanism is calculated by summing the output of the P-controller and the robot rotation compensation speed command:

$$\begin{aligned}\dot{\theta}_{k(\mathcal{F}_m)(Command)} = \dot{\theta}_{k(\mathcal{F}_m)(P-control)} &+ \\ \dot{\theta}_{k(\mathcal{F}_m)(Counter-rotation)}\end{aligned} \tag{9}$$

This approach using equation (9) is the one used in this work.

This algorithm requires an estimation of the person position. In case of a loss of tracking, the recovery procedure continues to apply the last pan axis speed command for 500 ms and then setting the pan axis to its neutral position, $\theta_{k(\mathcal{F}_m)} = 0°$, while waiting for a new person position estimation.

D. Path-following algorithm

The principle of the path following algorithm is to make the robot take the same path as the person it follows. It uses the succession of person position estimations in $\mathcal{F}_0$, denoted $Person_{(\mathcal{F}_0)}$, and is calculated from equation (5), to generate a set of points to send to the robot path follower previously described in the robot control section. However, the $Person_{(\mathcal{F}_0)}$ points cannot be directly sent to the robot path follower. They are too noisy when the robot is moving, as described in the experimental results.

Hence the $Person_{(\mathcal{F}_0)}$ points are first filtered:

- Points which imply that the person accelerates faster than 1 g are ignored.
- Points which imply that the person moves faster than $1.5m/s$ are ignored.
- Jitter reduction of $15cm$ radius is applied: if a point is not farther than $15cm$ from the previous point, it is ignored.

Then the path connecting the succession of points is smoothed using a moving average technique of span 5. Finally, as the robot path follower needs a path with points separated by an interval of $2cm$ to properly work, points are interpolated by using uniform cubic B-splines. This also ensures further smoothing of the path. After filtering, smoothing and interpolation, the output point, denoted $Person_{(\mathcal{F}_0)(Filtered)}$ is sent to the robot path follower.

E. Adaptive algorithm

The idea of the adaptive algorithm is to continuously recompute the best path for the robot to go to the person taking into account the obstacles in the environment. Hence, if a shorter way than the path the person took to go to its current position exists, the robot will be able to use it. The optimal path is computed using an implementation of the Karto library which uses the Monte Carlo Localization algorithm [35].

The adaptive algorithm uses the filtered and smoothed person position estimation, $Person_{(\mathcal{F}_0)(Filtered)}$, described in the previous section. Each time a position estimation is received, it is compared to the last position estimation used to generate the robot path. If the distance that separates these two position estimations is superior to 50 cm, a new path using the last position estimation is computed and sent to the robot. This approach is needed in order to limit the frequency of the re-computation of the path which, when too high, saturates the computer and makes the robot oscillate and change its course too often.

IV. RESULTS AND DISCUSSION

A. Path-following algorithm

For each of the 10 trials the robot was able to follow the person until the end of the path (Table I). The average 0.9 loss-of-track event per trial did not affect the performance of the following thanks to the efficiency of the tracking recovery procedure. Figure 3 illustrates this success and shows both

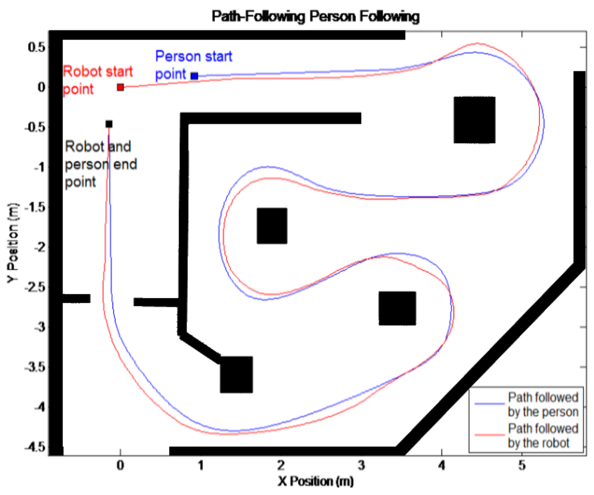


Fig. 3. Person and robot paths of a sample trial of the evaluation of the path-following algorithm.

TABLE I

EXPERIMENTAL RESULTS OF THE EVALUATION OF THE PATH-FOLLOWING ALGORITHM

Path-following algorithm	Average	Max	Min	Standard Deviation
Path-completion success ratio [%]	100	100	100	0
Number of loss-of-track events per trial	0.9	2	0	0.88
Robot path length to person path length ratio [%]	100.5	103.7	97.6	1.6
Path Error [cm]	11.04	40.27	0	7.64

robot and person path close from each other along with the obstacle setups from a typical trial.

The path taken by the person is reproduced accurately with an average path error of 11.04 cm, a standard deviation of 7.34 cm and a maximum error of 40.27 cm (Table I). The agility and accuracy of this method are fully understood when comparing the results with the 40 cm width of our robot. Thanks to this accuracy it is possible to perform person following in an environment with obstacles without the need of detecting and actively avoid the obstacles.

However, when comparing the distance covered by the human and the robot it appears that they are nearly the same. This is due to the principle of this algorithm: the path taken by the person is accurately followed and hence is not optimal; in case of a possible shorter path, it will not be taken by the robot.

B. *Adaptive algorithm*

In term of path-completion ratio the adaptive algorithm performed as good as the path-following algorithm with a 100% completion for all the trials; and similarly it was not affected by the nearly same average 1.1 loss-of-track event per trial. Figure 4 illustrates this success but shows also how the adaptive algorithm enables the robot to take a shorter path when it can. Over the 10 trials the distance travelled by the robot was 70.9% of the distance travelled by the person, with a maximum of 82.1%, a minimum of 57.6% and a standard deviation of 6.5%.

Hence, the adaptive algorithm presents the advantage of minimizing the distance travelled by the robot compared to the path-following algorithm. However, this requires a pre-build map of the environment.

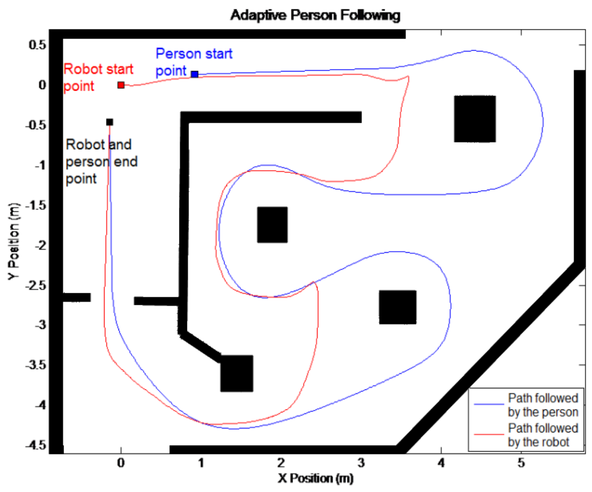


Fig. 4. Person and robot paths of a sample trial of the evaluation of the path-following algorithm.

TABLE II

EXPERIMENTAL RESULTS OF THE EVALUATION OF THE ADAPTIVE ALGORITHM

Adaptive algorithm	Average	Max	Min	Standard Deviation
Path-completion success ratio [%]	100	100	100	0
Number of loss-of-track events per trial	1.1	3	0	1.1
Robot path length to person path length ratio [%]	70.9	82.1	57.6	6.5

V. CONCLUSIONS AND FUTURE WORK

Two person-following algorithms that use depth information from a Kinect sensor were presented. Both use the Kinect sensor mounted on a pan-tilt mechanism for 360-angle tracking and implement path generation from a sequence of estimated person's positions. The path following algorithm generates sequentially a path that reproduces the path taken by the person using each new updated position of the person. On the other hand, the adaptive algorithm, re-computes from scratch the shortest path to the person each time the person has moved more than 50 cm. Both person-following algorithms were equally successful in following the person with a 100% path completion ratio. However, the adaptive algorithm minimized the distance travelled by the robot: it travelled in average 29.1% less than the person it followed whereas the path-following algorithm made the robot travel in average 0.5% more. Yet which algorithm is best to use is subject to discussion. The adaptive algorithm minimizes the distance travelled but presents the important constraint of needing a-priori information about the environment (i.e. a map). This can be an advantage in situations where the cost of travel of the robot is expensive or in situations where the maximum speed of the robot is inferior to the walking speed of the person followed.

Future work should focus on path optimization without a-priori information. The case of the robot standing in the way of the person was not investigated in this work. Hence algorithms must be developed to adapt the path of the robot in order not to block the way of the person when she/he changes suddenly of direction. Furthermore, strategies to recover from complete occlusions from other persons or walls should be improved.

References

[1] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2-3, pp. 90–126, Nov. 2006.

[2] Z. Jia, A. Balasuriya, and S. Challa, "Vision based data fusion for autonomous vehicles target tracking using interacting multiple dynamic models," *Computer Vision and Image Understanding*, vol. 109, no. 1, pp. 1–21, Jan. 2008.

[3] J. M. Martinez-Otzeta, A. Ibarguren, A. Ansuategi, and L. Susperregi, "Laser Based People Following Behaviour in an Emergency Environment," in *Proceedings of the 2nd International Conference on Intelligent Robotics and Applications (ICIRA '09)*, 2009, pp. 33–42.

[4] Y. Motai, S. Kumar Jha, and D. Kruse, "Human tracking from a mobile agent: Optical flow and Kalman filter arbitration," *Signal Processing: Image Communication*, vol. 27, no. 1, pp. 83–95, Jan. 2012.

[5] M. Kobilarov, G. Sukhatme, J. Hyams, and P. Batavia, "People tracking and following with mobile robot using an omnidirectional camera and a laser," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, no. May. IEEE, 2006, pp. 557–562.

[6] Y. Salih and A. S. Malik, "Comparison of stochastic filtering methods for 3D tracking," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2711–2737, Oct. 2011.

[7] R. Muñoz Salinas, E. Aguirre, and M. García-Silvente, "People detection and tracking using stereo vision and color," *Image and Vision Computing*, vol. 25, no. 6, pp. 995–1007, Jun. 2007.

[8] J. Satake and J. Miura, "Robust stereo-based person detection and tracking for a person following robot," in *ICRA 2009 Workshop on People Detection and Tracking*, no. May, 2009.

[9] R. Muñoz Salinas, R. Medina-Carnicer, F. Madrid-Cuevas, and a. Carmona-Poyato, "Particle filtering with multiple and heterogeneous cameras," *Pattern Recognition*, vol. 43, no. 7, pp. 2390–2405, Jul. 2010.

[10] M. Luber, L. Spinello, and K. O. Arras, "People tracking in RGB-D Data with on-line boosted target models," in *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*. IEEE, Sep. 2011, pp. 3844–3849.

[11] L. A. Schwarz, A. Mkhitaryan, D. Mateus, and N. Navab, "Human skeleton tracking from depth data using geodesic distances and optical flow," *Image and Vision Computing*, vol. 30, no. 3, pp. 217–226, Dec. 2012.

[12] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *The 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)*. Colorado Springs, CO, USA: IEEE, Jun. 2011, pp. 1297–1304.

[13] F. Hoshino and K. Morioka, "Human following robot based on control of particle distribution with integrated range sensors," in *2011 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, Dec. 2011, pp. 212–217.

[14] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An Evaluation of the RGB-D SLAM System," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA 2012)*, vol. 3, no. c. IEEE, 2012.

[15] M. Camplani and L. Salgado, "Efficient Spatio-Temporal Hole Filling Strategy for Kinect Depth Maps," in *IS&T/SPIE Int. Conf. on 3D Image Processing (3DIP) and Applications*, 2012.

[16] H. Latif, N. Sherkat, and A. Lotfi, "Information acquisition using eye-gaze tracking for person-following with mobile robots," *Information Acquisition*, vol. 06, no. 03, pp. 147–157, 2009.

[17] E. A. Topp and H. I. Christensen, "Tracking for following and passing persons," in *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*. IEEE, 2005, pp. 2321–2327.

[18] T. Yoshimi, M. Nishiyama, T. Sonoura, H. Nakamoto, S. Tokura, H. Sato, F. Ozaki, N. Matsuhira, and H. Mizoguchi, "Development of a Person Following Robot with Vision Based Target Detection," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*. IEEE, Oct. 2006, pp. 5286–5291.

[19] T. Sonoura, T. Yoshimi, M. Nishiyama, H. Nakamoto, S. Tokura, and N. Matsuhira, "Person Following Robot with Vision-based and Sensor Fusion Tracking Algorithm," in *Computer Vision*, X. Zhihui, Ed. Vienna, Austria: InTech, 2008, no. November, pp. 519–538.

[20] C. Cauchois, F. de Chaumont, B. Marhic, L. Delahoche, and M. Delafosse, "Robotic assistance: an automatic wheelchair tracking and following functionality by omnidirectional vision," in *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*. IEEE, 2005, pp. 2560–2565.

[21] X. Ma, C. Hu, X. Dai, and K. Qian, "Sensor integration for person tracking and following with mobile robot," in *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*. IEEE, 2008, pp. 3254–3259.

[22] A. Konigs and D. Schulz, "Fast visual people tracking using a feature-based people detector," in *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*. IEEE, Sep. 2011, pp. 3614–3619.

[23] V. Alvarez-Santos, X. Pardo, R. Iglesias, a. Canedo-Rodriguez, and C. Regueiro, "Feature analysis for human recognition and discrimination: Application to a person-following behaviour in a mobile robot," *Robotics and Autonomous Systems*, vol. 60, no. 8, pp. 1021–1036, Aug. 2012.

[24] T. Germa, F. Lerasle, N. Ouadah, V. Cadenat, and M. Devy, "Vision and RFID-based person tracking in crowds from a mobile robot," in *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, vol. 3. IEEE, Oct. 2009, pp. 5591–5596.

[25] T. Germa, F. Lerasle, N. Ouadah, and V. Cadenat, "Vision and RFID data fusion for tracking people in crowds by a mobile robot," *Computer Vision and Image Understanding*, vol. 114, no. 6, pp. 641–651, Jun. 2010.

[26] Z. Chen and S. T. Birchfield, "Person following with a mobile robot using binocular feature-based tracking," in *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*. IEEE, Oct. 2007, pp. 815–820.

[27] K. Qian, X. Ma, X. Dai, and F. Fang, "Robotic Etiquette: Socially Acceptable Navigation of Service Robots with Human Motion Pattern Learning and Prediction," *Journal of Bionic Engineering*, vol. 7, no. 2, pp. 150–160, Jun. 2010.

[28] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, May 2012.

[29] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe, "A review of recent developments in Simultaneous Localization and Mapping," in *2011 6th International Conference on Industrial and Information Systems*. IEEE, Aug. 2011, pp. 477–482.

[30] J. Miura, J. Satake, M. Chiba, Y. Ishikawa, K. Kitajima, and H. Masuzawa, "Development of a Person Following Robot and Its Experimental Evaluation," in *Proceedings of the 11th International Conference on Intelligent Autonomous Systems*, Ottawa, Canada, 2010, pp. 89–98.

[31] W. L. D. Lui and R. Jarvis, "A pure vision-based topological SLAM system," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 403–428, Feb. 2012.

[32] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The Office Marathon: Robust navigation in an indoor office environment," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*. Ieee, May 2010, pp. 300–307.

[33] S. Almansa-Valverde, J. C. Castillo, and A. Fernández-Caballero, "Mobile robot map building from time-of-flight camera," *Expert Systems with Applications*, vol. 39, no. 10, pp. 8835–8843, Aug. 2012.

[34] P. Morin and C. Samson, "Motion control of wheeled mobile robots," pp. 799–826, 2008.

[35] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Collective Motion Pattern Scaling for Improved Open-Loop Off-Road Navigation

Frank Hoeller, Timo Röhling, Markus Ducke, and Dirk Schulz

Abstract— **This paper presents an adaptive navigation system which is able to steer an electronically controlled ground vehicle to given destinations while it adjusts to changing surface conditions. The approach is designed for vehicles without a velocity controlled drive-train, making it especially useful for typical remote-controlled vehicles without upgraded motor controllers. The vehicle is controlled by sets of commands, each set representing a specific maneuver. These sets are combined to form trajectories towards a given destination. While one of these sets of commands is executed the vehicle's movement is measured to refine the geometry of all maneuvers. A scaling vector is derived from the changes in dimensions of the bounding boxes of the assumed and the actual path, which is then used to collectively update all known maneuvers. This enables the approach to quickly adapt to surface alterations. We tested our approach using a 300 kg Explosive Ordnance Disposal (EOD) robot in an outdoor environment. The experiments confirmed that the Collective Motion Pattern Scaling significantly increases the adaptation performance compared to an approach without collective scaling.**

I. INTRODUCTION

In the design of robot systems operating in unstructured outdoor environments special care has to be taken that the robots do not accidentally collide with obstacles in their vicinity. Compared to indoor situations the robot can suffer drastically more damage from the more hazardous surroundings. The risk is increased by different ground surfaces, which have a distinct effect on the wheel grip. The resulting deviation has to be anticipated to ensure the reproducibility of planned motions, and thus making collision avoidance possible.

Additional complications arise for robots which were designed for remote-control. Such robots are usually only equipped with relatively simple motor controllers missing an appropriate servo loop for interpreting velocity commands. Unfortunately, this is a requirement for most classic navigation algorithms. Moreover, the impact of adhesion changes is further intensified by such open-loop controllers because no matter how much the wheel grip, and thus the behavior of the robot changes, there is no feedback of the actual movement. Of course, these problems could easily be solved by using a wheel encoder or similar, but adding such to existing, especially commercial robots is rarely possible. Mostly a time-consuming redesign or an expensive new acquisition are the only options.

In this article we present an approach, which allows a mobile robot with any kind of electronic motor controller to operate in outdoor environments while adjusting to changing surface conditions to provide safety and effectiveness. All components of our system follow a local navigation paradigm and do not need global information on the environment, neither of surface characteristics nor on obstacles. Instead, the system decides solely based on the robot's sensory input.

Frank Hoeller, Timo Röhling, Markus Ducke, and Dirk Schulz are with the Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Germany

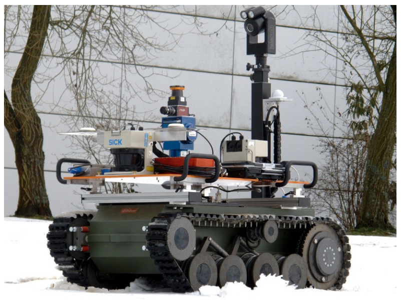


Fig. 1. A Telerob Teodor robot equipped with sensors for basic autonomous navigation.

The motion planning developed for our robot composes paths by combining predefined Motion Patterns. Each Motion Pattern consists of a set of robot commands and a series of poses that represent the robot's movement when the command set is executed by the controllers. With these Motion Patterns, the local navigation module repeatedly computes trees of collision-free command sequences. From each tree a path is extracted which brings the robot close to the destination coordinate as fast as possible.

If one is able to measure the robot's motion on the fly, e.g using SLAM (simultaneous localization and mapping) techniques or an INS (inertial navigation system), one can also monitor movement trajectories. Compared to drive-trains with servo loop that only regard the motor speeds, the results of command sequences can be observed on a larger scale, which allows to tackle the surface traction problem in a novel way: The collected trajectory data is used to update the previously measured movement trajectory of the corresponding Motion Patterns. Furthermore, the detected changes are propagated to all other Motion Patterns by calculating a scaling vector from the alteration in dimensions of a trajectory. The upgraded Motion Patterns are handed over to the planning process and used for the tree generation from then on. Note, that it is not possible to adapt the command sequence to match the desired trajectory because the mapping from trajectories to commands is unknown.

The remainder of this article is organized as follows: After discussing related work in Section II, we introduce our Motion Pattern based local navigation approach in Section III, followed by a description of the learning and collective scaling procedures in Section IV. Before we conclude, we describe some experiments to illustrate the capabilities of our approach. We implemented our approach on a Telerob Teodor EOD robot (Fig. 1) and verified its feasibility in outdoor settings.

II. RELATED WORK

In the field of outdoor robotics the terrain always is of special interest. The analysis and classification of different surfaces regarding their traversability has been addressed by many different authors. The classification of the different surface types using vibration sensors is very popular because this sensing mode is not vulnerable to lighting or perspective issues. Brooks et al. [1] attached this kind of sensor to axle arms and classified different terrains by traversing them. They used offline learning in combination with a voting mechanism to enable the system to identify a set of different surfaces. Unfortunately, classification approaches of this type can only identify known terrains without deriving information concerning the behavior of a vehicle on the respective surface. The identified terrain type would have to be associated with a parameter set for the trajectory generator in an intermediate step. Furthermore, this would make the navigation dependent on the a priori learned identification data, which would violate the intended local navigation paradigm.

The DARPA Grand Challenge winning robot Stanley [12] also uses a vibration sensor to regulate its maximum speed. Unwilling to catalog every possible terrain type, Stavens et al. evaluate the occurring vibrations to limit Stanley's speed according to observed human driving behaviors [10]. A similar approach is presented by Castelnovi et al. [2], but instead of sensing vibrations the authors used a 2D laser range finder aimed downwards to calculate a ruggedness grade. Based on this result the robot's top speed is reduced, resulting in a decreased number of terrain-related incidents. A combined approach was later proposed by Stavens et al. as an upgrade for Stanley's system [11]. Here a learning component associates vibration intensities with surface profiles measured with a forward-facing 3D laser distance scanner. This way the system can automatically learn terrain-speed-associations. Thus the maximum speed can be adjusted before the vibration sensor detects a surface transition and the vehicle is exposed to less shock. These approaches show several similarities to the technique presented in this paper as the analysis of the ground directly affects the local navigation. Nevertheless, only the maximum translation velocity is altered. Rotation velocities are not addressed at all, which is not necessary for a robot like Stanley.

Another interesting approach by Martinelli et al. [5] also uses laser range finders, but in combination with SLAM and Kalman filter techniques. The system is able to determine the systematic component of the odometry error by using the wheel encoder readings and the estimated SLAM position. The non-systematic error, which is more interesting in outdoor applications, can also be determined by a Kalman filter applied on a history of robot states. These are provided by the former Kalman filter, making the estimation indirectly dependent on wheel encoders and closed-loop control.

Crusher, a six-wheeled robot for extreme outdoor environments also suffered from trajectories differing from planned paths. Seegmiller at al. proposed an approach to automatically calibrate a dynamic model [9]. It linearizes the nominal vehicle model and then calibrates the dynamics to explain the observed prediction residuals using a Kalman filter. Their system, just as our system, takes advantage of the precise short-term localization techniques or sensors available. Although results are impressive, their approach is again tailored to a velocity driven model which unfortunately is not applicable to our robot.

The combination of motion templates and learning has been used widely in the area of walking robots. In this field, learning techniques are mostly applied to improve walking policies that were derived from simulations [6] or observed from human walking [7]. Furthermore, due to the complexity of biped locomotion, every walking robot is equipped with many sensors to determine its stance and to allow closed-loop control.

Similarly, the concept of motion template based learning has also been employed to simplify the learning of complex motions [8]. In contrast to our approach the templates are parameterized, so they can be adjusted to fit the desired trajectory. This implies a feasible correlation between parameter input and drive-train behavior.

III. LOCAL NAVIGATION WITH MOTION PATTERNS

The core of the overall approach is a local navigation planning component that directly controls the robot and steers it on a collision-free path from its current position to a given destination in configuration space. For this purpose special precautions for the open-loop motor controllers have to be taken. Since remote-controlled robots lack a velocity regulator circuit, the control commands influence the motor power directly. This induces that their outcome depends on many factors and is far too complex to compute in an online approach. To make motion planning still possible, we introduce Motion Patterns. The first component of a Motion Pattern *MP* is a series of robot control commands $U = (u_1, \ldots, u_T)$. A command u_t can be of any type and dimension: when used with a Teodor robot they are motor power commands, when controlling a car they probably are throttle position and steering angle. A command sequence U is immutable, which implies that Motion Patterns cannot be parameterized e.g. regarding their velocity. The second component of a Motion Pattern is an array of oriented relative positions $R = (\Delta r_1, \ldots, \Delta r_T)$. It represents the trajectory on which the robot would probably move when the command series is sent to the robot, so each pose Δr_t describes the relative position of the robot after it executed the command sequence from u_1 to u_t. The Motion Patterns can now be combined to form motion paths $P = (R_1, \ldots, R_k)$. Of course it has to be checked if concatenated Motion Patterns fit together so that no harsh velocity change creates

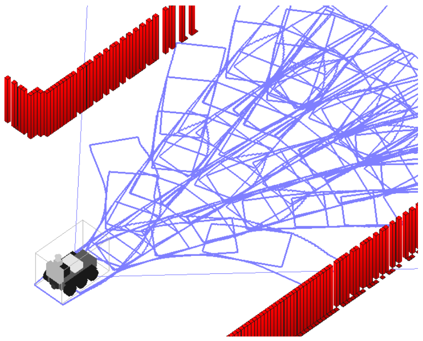


Fig. 2. A planning tree of collision free paths that has been build using Motion Patterns in an 2-dimensional simulated environment.

unexpected movement trajectories. The created paths can then be checked for collisions e.g. by using an occupancy grid [4]. The corresponding sequence of series of commands $C_P = (U_1, \dots, U_k)$ can be merged to a large array of robot commands which can be executed by the robot sequentially. Notice that the number, shape, and complexity of Motion Patterns are not restricted, but definitely have an impact on the planning process. In general, with fewer patterns a larger range can be covered, while more patterns increase the quality of the resulting paths. Since basic navigation capabilities were sufficient for the following learning algorithm, a set of five different Motion Patterns was used.

Based on the model above, we can now also build a collision-free tree of Motion Patterns and extract the best path towards the destination. The path planning process is described in detail in [4] and an example tree generated by this technique can be seen in Fig. 2. The theoretical principles of a very similar navigation approach are examined in [3] extensively.

IV. MOTION LEARNING USING COLLECTIVE MOTION PATTERN SCALING

A problem arising from our kind of local navigation is its sensitivity to surface and traction changes. Motion Patterns are created for specific surfaces only. And it is unlikely that the surface or the surface's condition always remains constant, especially in outdoor scenarios.

To compensate for this, the local navigation has been extended by a learning mechanism. While the command set of a selected Motion Pattern *MP** is executed, the robot's reactions are measured. For this purpose the robot's movement trajectory *M*, consisting of the x, y and φ deltas, is recorded. The saved trajectory R_{MP*} inside the Motion Pattern *MP** can directly be updated with this updated measurement. This technique was combined with an exponential smoothing (see below) to form the first version of the learning Motion Pattern based local navigation. The potential of this basic system has been shown in [4]. Both learning approaches, the basic version from [4] and the improved version discussed here, as well as the planning mechanism assume, that applied Motion Patterns yield in similar trajectories repeatedly. The exponential smoothing can cope with singular discontinuities, but it is not able to handle continuously changing results, i.e. occuring on slopes or rough terrain. Both methods are computationally simple and the calculation effort is negligible compared to the time needed for path planning.

The trajectory update inside the Motion Pattern can be regarded as information gain. We would like to propagate this additional knowledge to all other Motion Patterns as well to improve the adaption speed considerably. For this purpose we calculate a vector of scaling factors $V = (x_s, y_s, \varphi_s)^T$ by comparing the recently measured trajectory and the prediction R_{MP*} saved inside the Motion Pattern *MP**. x_s and y_s regard the positions inside the trajectory, and φ_s the trajectory's yaw information. The first approach would be to compare the final pose of the measured trajectory *M* and the predicted trajectory from *MP** to compute these factors. Unfortunately, this would decrease the robustness for trajectories which have a final lateral position close to its initial value (e.g. double-lane change maneuvers). In these cases, it is likely that the measurement noise exceeds the position change, which would generate enormous scaling factors. A similar problem occurs for patterns which only include a movement in only one dimesion, e.g. forward motions or in-place turns. Although these patterns can be scaled, they cannot be used to calculate a reasonable scaling vector V because their movement in the unaddressed axes is only caused by noise and would again generate exaggerated scaling factors. For this reason, patterns like these are taken out of the learning process, because it is not possible to gain information from something, that did not change. To adresse the former case of this problem, the length and the width of the bounding boxes (*BB*) around the two considered trajectories are used to calculate the scaling factors x_s and y_s. The yaw scaling factor φ_s is calculated similarly to the position factor: here the interval between the minimum and maximum of all recorded yaw angles is used. Now we can compute a quotient for every dimension and compose the scaling vector V:

$$V(M, MP*) = \begin{pmatrix} x_s \\ y_s \\ \varphi_s \end{pmatrix} = \begin{pmatrix} \frac{\text{Length}_{BB}(M)}{\text{Length}_{BB}(MP*)} \\ \frac{\text{Width}_{BB}(M)}{\text{Width}_{BB}(MP*)} \\ \frac{|\text{AngleInterval}(M)|}{|\text{AngleInterval}(MP*)|} \end{pmatrix} \quad (1)$$

This vector can now be used to collectively scale all Motion Patterns making it possible to update even yet unregarded Motion Patterns. An example application is illustrated in Fig. 3. In this manner the knowledge of change of movement behavior is propagated without the need to wait for every Motion Pattern to be chosen and executed, reducing the adoption time notably.

In both cases, without and with Collective Motion Pattern Scaling, the new predictions are integrated in the Motions Patterns' existing trajectory using a component-by-component exponential smoothing function. This allows continuous learning and at the same time smooths minor surface variations to prevent an oscillating learning behavior:

$$R_{p,t} = (1 - w)M_{p,t} + (w)R_{p,t-1} \quad (2)$$

with $0 \leq w < 1$. Here $M_{p,t}$ is a new trajectory for a Motion Pattern p at time t, measured or derived by collective scaling.

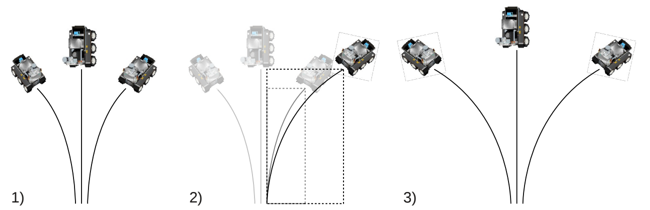


Fig. 3. Evolution of a Motion Pattern set with three patterns: 1) Initial set before a Motion Pattern is chosen and executed. 2) After measuring a Motion Pattern's (MP*) result the bounding box changes are analyzed. 3) All Motion Patterns are scaled yielding to a new set.

$R_{p,t-1}$ represents the existing prediction of the pattern and $R_{p,t}$ the updated prediction. To limit the impact of new measurements w is introduced. Setting it to 0.5 turned out to work fairly well. The whole learning approach is outlined in Algorithm 1.

Algorithm 1 Collective Motion Pattern Scaling

```
 1: while $inMotion$ do
 2:     get next Motion Pattern *MP* from planner
 3:     send *MP* to motor controllers and record motion
 4:     let *M* be the recorded robot motion
 5:     calculate scaling vector *V*                    ▷ see eqn (1)
 6:     UPDATEMOTIONPATTERN(*MP*, *M*)
 7:     for all other MotionPatterns do
 8:         UPDATEMOTIONPATTERN(*MP*, *MP* $\cdot V$)
 9:     end for
10: end while

11: procedure UPDATEMOTIONPATTERN($mp$, *update*)
12:     let $w$ be the impact reduction
13:     $mp \leftarrow (1-w) \cdot mp + (w) \cdot update$   ▷ see eqn (2)
14: end procedure
```

The idea behind this mechanism is that changes in road grip in general affect the forward and lateral movement achieved when executing a command sequence, and that these changes can be approximately captured by the bounding box around the resulting trajectory. Although we disregard it in our learning approach, there certainly is a relation between the length and the width of a trajectory: If the ground surface changes from a sticky to a slippery nature, an executed turn would be wider than before. The resulting bounding box of the trajectory would be longer and correspondingly narrower. But to model this relation more information about the shape of the trajectory would be needed. By the nature of our approach, these are not available, so further detailed analysis of the trajectory would require more computational time, which at the moment is beyond the time frame of our online application. Surface transitions are likely to change also the orientation of the robot while executing a Motion Pattern. To be able to predict complete poses, the position scaling technique is reduced to one dimension and applied again to the orientation values. In a one-dimensional space a bonding box diminishes to an interval enclosing all occurring yaw angles.

Although not required for the learning technique presented here, it is reasonable that most sets of Motion Patterns contain a number of symmetric patterns, e.g. a left and a similar right turn. If one of these patterns is executed and measured, a very precise prediction for the other is generated, which lessens the error margin of the approximation and thus increases the effectiveness of the collective scaling.

V. EXPERIMENTS

The system described in the previous chapters has been tested in simulations but using data from a real robot. The Collective Motion Pattern Scaling algorithm proposed here is compared with its predecessor, whose performance is shown in [4]. To demonstrate the performance of the learning and prediction techniques a controlled environment is essential. Although this might be realizable in simulations, the authors have chosen a different approach.

A. Testing Approach

The following experiment aims at investigating the results of the two mentioned algorithms on a transition from a surface A to another surface B. To maximize the quantitative outcome of the experiments the data acquisition was separated from the algorithm tests. For this purpose, the different surfaces were traversed separately to collect data sets for each surface. Later, these sets are used in combination to simulate surface transitions.

In the data collecting stage the robot executed a large number of Motion Patterns on different ground surfaces, one surface at a time. The driven trajectory of each Motion Pattern was collected, resulting in a sample set RP_S of 300 recordings per Motion Pattern for every considered surface S. As preparation for the second stage a set of averaged Motion Patterns is calculated from the 300 recordings for each RP_S. The result is a set of Motion Patterns IP_S with very precise trajectory predictions for the regarded surface S. When reproducing a terrain change from a surface A to a surface B, the set IP_A is used as initial Motion Pattern database representing the already learned surface type A.

For the next step a test sequence *TS* of Motion Patterns, which is executed after the virtual terrain change, has to be determined. To show the propagation capabilities, two sequences were chosen and are used alternately. Each sequence consists of three turning patterns resulting in a left-left-right and accordingly a right-right-left motion. Note that the left and the right turn Motion Patterns roughly are mirrored equivalents and have a total length of about a second. The test sequence has to be compiled of turning patterns because straight movements only allow one-dimensional corrections (see Section IV). For each run of the test sequence the initial Motion Pattern database IP_A is used to create identical starting conditions.

In the next step the test sequence *TS* is processed one pattern after another. Before a Motion Pattern's command sequence is sent to a robot, the two algorithms to be tested already have a prediction of the prospective trajectory. The accuracy of these trajectories will be inspected. Instead of

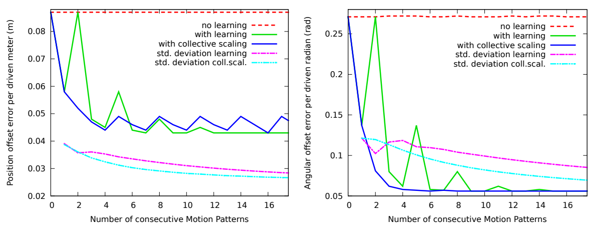


Fig. 4. Error developing of a Teodor robot changing from cinder to grassland during an experiment with extended run-time.

sending the command sequence of each Motion Pattern to a real robot and recording its motion anew, the pattern is processed offline: In exchange for the online motion recording, we use a random trajectory recording from the initially measured collection of trajectories RP_B of surface B. The trajectory is given to the two algorithms to initiate the learning process to update their predictions for the next request. We assume that the trajectories recorded for a pattern on one particular surface are exchangeable i.e. that the trajectories are randomly distributed given a surface. This allows us to do quantitative evaluations using sampling techniques. Further, this enables us to easily process larger numbers of test repetitions.

The processing of the test sequence is repeated 30,000 times. The random trajectories from RP_B are chosen anew every time, but in a fashion that both learning algorithms receive the same data in every run. At the end of every pattern of *TS*, the final position of the recording is compared with the two position predictions provided by the two learning algorithms.

A disadvantage of this simulation method is that surface changes can only be simulated in between Motion Patterns, which is unlikely to happen in the real world. But even in the likely cases the learning mechanisms would adjust the trajectory predictions in the right direction, still reducing planning error margins. The learning process would only be slowed down, and both algorithms would be affected in the same way.

B. Testing Results

The input data for this simulation study was recorded using a tracked Telerob Teodor EOD robot which was upgraded with the required sensors to enable autonomous behavior. In addition, an INS was installed to enable easy and precise movement recordings. For the first phase of the experiment the robot was operated on the following four surfaces to collect the data for the sample sets RP_S: tarmac, grassland, chunky gravel, and volcanic cinder (commonly used for soccer fields). During the experiment the robot always moved at its maximum speed of about 1.0 m/s. The generated sample sets were used to simulate all possible surface transitions.

Table I shows a comparison of four processing methods including a naive and an omniscient policy: a) no learning while keeping the database, b) single pattern learning from [4], c) learning with collective scaling, d) a priori correctly chosen pattern database with learning deactivated showing the minimal possible errors. The error reduction percentages for the normal learning and the learning with collective scaling are presented in Table II. As in the simulated experiment, the reduction of the orientation error is noticeably higher then the position error reduction. The angular error reduction ranges from 7.3 % up to 27.9 % with one exception that probably resulted from similar surface properties. The position error reduction on the other hand only reaches 17.8 % at best, but is normally below 10 %, sometimes even negative. The measured orientation error values are quite large, but it has to be kept in mind that these are normalized. Just as the position error are scaled with respect to the number of driven meters, the rotation errors are scaled with respect to the covered angle. For the patterns examined here the scaling factor is around 3.

Note that the position error reduction can artificially be improved by increasing the pattern length, taking advantage of the large orientation error reduction. When the pattern length is doubled most of the negative position error reductions raise well above zero. To keep the results realistic, these oversized patterns were not used.

Fig. 5 depicts the error distributions of the two tested approaches when changing from cinder to grassland as an example. The distribution of the position error is not as well formed as the distribution of the angular errors, but still a shift towards small errors can be seen. Most of the error distributions of the other transitions with a crucial reduction look similar.

Since both learning algorithms use the same exponential smoothing function, both errors will correlate with the minimal error if the experiment is continued long enough. Fig. 4 shows the error development of the two algorithms when the chosen Motion Pattern sequence is processed repeatedly without resetting the initial pattern database. Both algorithms reach the minimal error during the extended time span, but especially in the beginning the differences are very large. This confirms that the beginning phase is crucial for a fast adaption, and encourages the concentration on this phase. The position error graph also reveals the reason for the poorer performance of the position estimation: After reaching a value of about 0.045 m, the error begins to oscillate in a range of approx. 0.005 m. These 5 mm are the error caused by the propagation of the Collective Motion Pattern Scaling. At this point the algorithm has reached its highest absolute accuracy; even when extending the Motion Pattern length, the amplitude does not increase.

The long-term experiment also enables us to evaluate the variance of the occurring errors. After the first three patterns the standard deviation with collective scaling is higher than the standard deviation of the simple learning method. This is caused by the exponential smoothing: When it is used, it intentionally prevents instant adaption and generates intermediate trajectory predictions while converging towards the measured behavior. When adding collective scaling, these intermediate predictions appear more often due to the greater number of adjusted Motion Patterns, especially immediately after a surface transition. As soon as the adaption to a new surface is completed, the deviation decreases and stays below

TABLE I

RESULTS OF THE SIMULATIONS: POSITION (FIRST ROW) AND ORIENTATION (SECOND ROW) ERRORS OF ALL POSSIBLE TRANSITIONS. VALUES DENOTE THE ERROR PER DRIVEN METER/RADIAN. A) NO LEARNING, B) SIMPLE LEARNING, C) COLLECTIVE SCALING, D) OMNISCIENT.

to	cinder				tarmac				grassland				gravel			
from	a)	b)	c)	d)	a)	b)	c)	d)	a)	b)	c)	d)	a)	b)	c)	d)
cinder	-	-	-	-	0.043	0.043	0.042	0.035	0.087	0.077	0.063	0.039	0.063	0.061	0.058	0.046
	-	-	-	-	0.041	0.043	0.042	0.037	0.271	0.226	0.162	0.067	0.164	0.147	0.126	0.099
tarmac	0.059	0.055	0.056	0.041	-	-	-	-	0.060	0.057	0.051	0.039	0.054	0.054	0.057	0.046
	0.107	0.100	0.091	0.030	-	-	-	-	0.204	0.172	0.129	0.067	0.113	0.111	0.109	0.099
grass	0.108	0.093	0.085	0.041	0.071	0.064	0.060	0.035	-	-	-	-	0.063	0.060	0.059	0.046
	0.372	0.313	0.231	0.030	0.256	0.214	0.154	0.037	-	-	-	-	0.165	0.151	0.133	0.099
gravel	0.060	0.056	0.055	0.041	0.045	0.044	0.046	0.035	0.064	0.060	0.057	0.039	-	-	-	-
	0.191	0.166	0.134	0.030	0.090	0.079	0.066	0.037	0.136	0.119	0.098	0.067	-	-	-	-

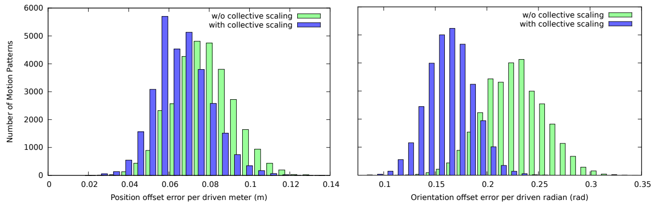


Fig. 5. Result of a simulation: The position (left) and orientation (right) error distribution of a Teodor robot changing from cinder to grassland without and with collective scaling.

TABLE II

THE POSITION (FIRST ROW) AND ORIENTATION (SECOND ROW) ERROR REDUCTIONS OF THE TRANSITION EXPERIMENTS (99% CONFIDENCE).

transition from / to	cinder	tarmac	grassland	gravel
cinder	-	1.5 %	17.8 %	3.4 %
	-	18.5 %	27.9 %	13.5 %
tarmac	-0.5 %	-	10.3 %	-4.9 %
	8.5 %	-	24.9 %	1.2 %
grassland	9.1 %	5.0 %	-	-1.1 %
	26.0 %	27.8 %	-	7.3 %
gravel	0.3 %	-5.4 %	4.5 %	-
	18.4 %	15.6 %	17.2 %	-

the deviation of the simple method.

VI. SUMMARY AND CONCLUSION

In this paper we presented an adaptive navigation system based on predefined motion templates called Motion Patterns. The system has the ability to incorporate the actual robot movement into the Motion Patterns. The updating process is not limited to the actually driven Motion Pattern. In most cases the system is also able to derive adjustment information for all other patterns as well. The soundness of our approach has been shown in a simulation study using real-word data. The system proved it can efficiently learn the robot's behavior after transitions between different surface types while outperforming the previous approach without collective scaling. Future work will focus on further improving the performance of the motion learning and adapting mechanisms. Decomposing a Motion Pattern's recorded trajectory in a series of straight lines and connection angles could lead to an improved geometrical understanding.

Acknowledgements: The presented system is part of the project RoboGasInspector, funded by the German Federal Ministry of Economics and Technology based on a resolution of the German Bundestag.

REFERENCES

[1] C.A. Brooks and K. Iagnemma. Vibration-based terrain classification for planetary exploration rovers. *Robotics, IEEE Transactions on*, 21(6):1185 – 1191, December 2005.

[2] M. Castelnovi, R. Arkin, and T.R. Collins. Reactive speed control system based on terrain roughness detection. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 891 – 896, April 2005.

[3] E. Frazzoli, M.A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *Robotics, IEEE Transactions on*, 21(6):1077 – 1091, Dec. 2005.

[4] F. Hoeller, T. Röhling, and D. Schulz. Offroad navigation using adaptable motion patterns. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, June 2009.

[5] A. Martinelli, N. Tomatis, A. Tapus, and R. Siegwart. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1499 – 1504 vol.2, Oct. 2003.

[6] J. Morimoto and C.G. Atkeson. Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Auton. Robots*, 27(2):131–144, 2009.

[7] J. Morimoto, J. Nakanishi, G. Endo, G. Cheng, and C. G. Atkeson. Poincare-map-based reinforcement learning for biped walking. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2381–2386, 2005.

[8] G. Neumann, W. Maass, and J. Peters. Learning complex motions by sequencing simpler motion templates. In *Proc. of the International Conference on Machine Learning (ICML)*, 2009.

[9] Neal Seegmiller, Forrest Rogers-marcovitz, Greg Miller, and Alonzo Kelly. A unified perturbative dynamics approach to online vehicle model identification.

[10] D. Stavens, G. Hoffmann, and S. Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[11] D. Stavens and S. Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. In *Proc. Conference on Uncertainty in AI (UAI)*, pages 13–16, 2006.

[12] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 2006.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Towards the implementation of a MPC-based planner on an autonomous All-Terrain Vehicle

Luca Bascetta, Davide Cucci, Gianantonio Magnani, Matteo Matteucci,
Dinko Osmanković, Adnan Tahirović

Abstract— **Planning and control for a wheeled mobile robot are challenging problems when poorly traversable terrains, including dynamic obstacles, are considered. To accomplish a mission, the control system should firstly guarantee the vehicle integrity, for example with respect to possible roll-over/tip-over phenomena. A fundamental contribution to achieve this goal, however, comes from the planner as well. In fact, computing a path that takes into account the terrain traversability, the kinematic and dynamic vehicle constraints, and the presence of dynamic obstacles, is a first and crucial step towards ensuring the vehicle integrity.**
The present paper addresses some of the aforementioned issues, describing the hardware/software architecture of the planning and control system of an autonomous All-Terrain Mobile Robot and the implementation of a real-time path planner.

I. INTRODUCTION

The popularity of the research on wheeled mobile robots has been recently increasing, due to their possible use in different outdoor environments. Planetary explorations, search and rescue missions in hazardous areas [1], surveillance, humanitarian de-mining [2], as well as agriculture works such as pruning vine and fruit trees, represent possible applications for autonomous vehicles in natural environments. Differently from the case of indoor mobile robotics, where only flat terrains are considered, outdoor robotics deals with all possible natural terrains. The unstructured environment and the terrain roughness, including dynamic obstacles [3], and poorly traversable terrains, make the development of an autonomous vehicle a challenging problem.

The aim of our research is to develop an All-Terrain Mobile Robot (ATMR), based on a commercial All-Terrain Vehicle (ATV), that is suitable for a wide range of different outdoor operations. The ATMR should be able to operate in any natural environment with a high level of autonomy. The advantage of using ATVs is represented by their good traversability potential for poorly traversable terrains and by the short time spent for reaching the goal, as well as by the possibility to operate in unsafe environments. On the other hand, the main disadvantage of ATVs is their low stability margin due to dynamic constraints, roll-over and excessive side slip [4].

The research leading to these results has received funding from Filas, the Regional Development Agency of the Lazio Region in Italy, under grant agreement No. FILAS-RS-2009-1290 - Project QUADRIVIO.

L. Bascetta, D. Cucci, G. Magnani and M. Matteucci are with Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo Da Vinci 32, 20133, Milano, Italy (email: {bascetta, cucci, magnani, matteucci}@elet.polimi.it).

D. Osmanković and A. Tahirović are with the Department of Automatic Control and Electronics, Faculty of Electrical Engineering, University of Sarajevo, Bosnia and Herzegovina (email: {dinko.osmankovic,atahirovic}@etf.unsa.ba).

ATVs are highly unstable, especially during fast turns and uphill/downhill riding, and a roll/tip-over can often occur. To overcome those problems the development of some active control systems [5], and in particular an Anti-Roll-over System [6], would certainly enhance their drivability. Moreover, it becomes necessary once the vehicle is teleoperated or autonomous. The design and development of an All-Terrain Mobile Robot is thus a challenging task, especially when a high level of autonomy is required. Indeed, due to the complex tasks the robot is supposed to perform, the design of the entire control architecture is anything but trivial [7]: different kind of requirements come from software engineering (e.g. modularity or maintainability), control theory (e.g. stability, robustness, hard real-time-ness) and mobile robotics (e.g. path planning, obstacle avoidance). The hardware/software architecture should fulfil them all in the simplest way.
A natural way to achieve those requirements is to design a multi-layered software architecture, in order to map higher levels of algorithmic abstraction to the top layers of the architecture. The control level that will act as an interface from these high level tasks (action planning, goal prioritisations, etc.) and the vehicle itself will be called "virtual rider". The aim of the virtual rider is to interpret commands from planner and execute them avoiding dangerous manoeuvres that could result in instability. Together with the virtual rider algorithm, a low level control software will be necessary in order to execute simple commands such as steering or braking.

All the aforementioned issues, crucial to ensure the vehicle integrity, can be addressed at two different levels. On one side, the virtual raider should operate in real-time to keep, as much as possible, the vehicle in a safe condition, or to recover it from dangerous situations. On the other, the planner plays a crucial role in computing a safe path, that a priori avoids dangerous manoeuvres.

The present paper describes the implementation and preliminary validation of a MPC-based planner that allows to compute in real-time a path from a starting to a goal position, taking into account obstacles, terrain characteristics and vehicle dynamic and kinematic constraints. The planner is implemented using an optimal control software (ACADO) to solve an initial value optimal control problem in receding horizon manner. Performance function and cost-to-go term are based on the terrain roughness. We locally interpolate roughness data at each time horizon with differentiable functions making it possible to use optimal control techniques provided by ACADO.

II. THE ATMR

The vehicle considered in this research (see Figs. 1 and 2) is a YAMAHA GRIZZLY 700, a commercial fuel powered All-Terrain Vehicle (ATV) equipped with an electric power steering (EPS).
The GRIZZLY 700 is a utility ATV and is thus specifically designed for agriculture work. As a result it has a total load capacity of 130 Kg, and it is equipped with a rear tow hook. The main characteristics of the vehicle are listed in Table I.

Main characteristics of the vehicle

Engine type	686cc, 4-stroke, liquid-cooled, 4 valves
Drive train	2WD, 4WD, locked 4WD
Transmission	V-belt with all-wheel engine braking
Brakes	dual hydraulic disc (both f/r)
Suspensions	independent double wishbone (both f/r)
Steering System	Ackermann
Dimensions (LxWxH)	2.065 x 1.180 x 1.240 m
Weight	296 Kg (empty tank)

TABLE I
VEHICLE CHARACTERISTICS

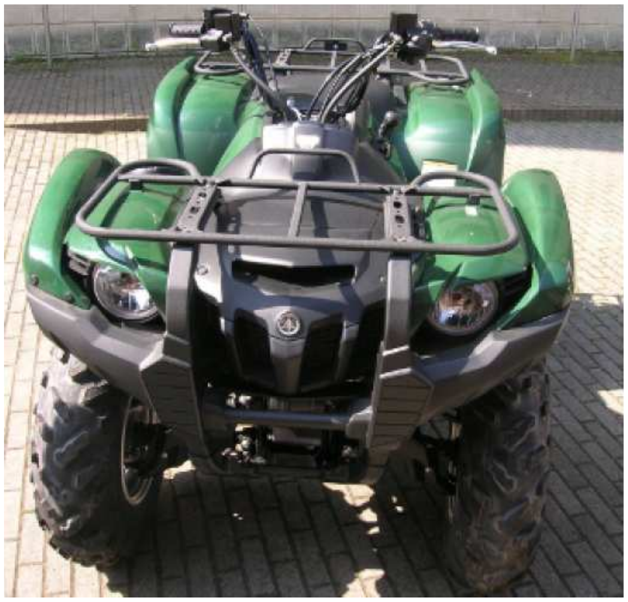


Fig. 1. The Yamaha Grizzly 700 ATV

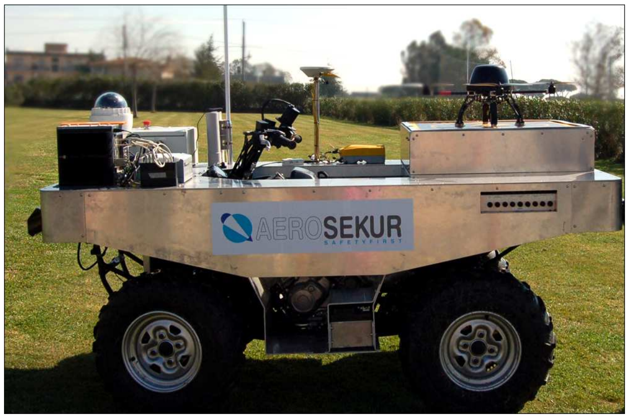


Fig. 2. The vehicle with the new cover

For the purposes of the project, the original vehicle cover has been removed and substituted with an aluminium cover, that allows to easily accommodate for the control hardware and the sensors (Fig. 2).

III. CONTROL SYSTEM ARCHITECTURE

In order to make the vehicle teleoperated, or even autonomous, an on-board hardware/software control platform has to be added. While the implementation of the whole architecture is still under development, a functional diagram that shows the main components of the control system and their relationships is shown in Fig. 3.

The architecture can be divided into three different layers. The top level is a high level planner responsible for the task acquisition and for the medium-long range navigation and planning functionalities. The virtual rider is an intermediate level and is responsible for short range navigation, planning and vehicle stabilisation. It has to ensure vehicle integrity with respect to roll-over/tip-over instabilities, obstacles and terrain traps, etc., replacing the typical low-level riding skills of a human. The lower level represents an interface between the vehicle commands and the virtual rider. Such level interacts with the vehicle measuring the steering angle, throttle ratio, vehicle speed, etc., and acting on the steering column, the throttle leverage and/or the brake pedal through suitable sensors and actuation systems (see [8] for further details).

To implement such a complex architecture that includes high level and low level tasks, the former characterised by an heavy computational load but slower sampling frequencies, the latter being simpler but needing a faster time response, a multi-layered and multiprocessor hardware/software architecture is required. In this way, one can separate complex (localisation and navigation on rough terrains, obstacle avoidance, sensor fusion, etc.) from simple tasks (motion control and servo actuation) and faster from slower ones.
The hardware/software architecture should be as modular as possible, in order to be simply reconfigurable and upgradeable. Indeed, the different computational complexity of the tasks calls for different layers of the control system, thus a multiprocessor architecture is an obvious choice. On the other hand, the navigation control system requires the complete knowledge of the state of the vehicle (in terms of what the sensors are perceiving) to take the best decision autonomously. Thus a very large amount of data must be shared between the system's layers.

The selected hardware architecture (Fig. 4) consists of:

- a **low level** CPU (PLC) with several I/O modules to perform the control of the steering angle, the throttle position, the pressure of the hydraulic braking circuit, etc. An industrial PLC provided by B&R AUTOMATION (X20 CPU: Celeron 650, 64 MB DRAM, 1 MB SRAM, maximum bus frequency 2 kHz) was selected for its

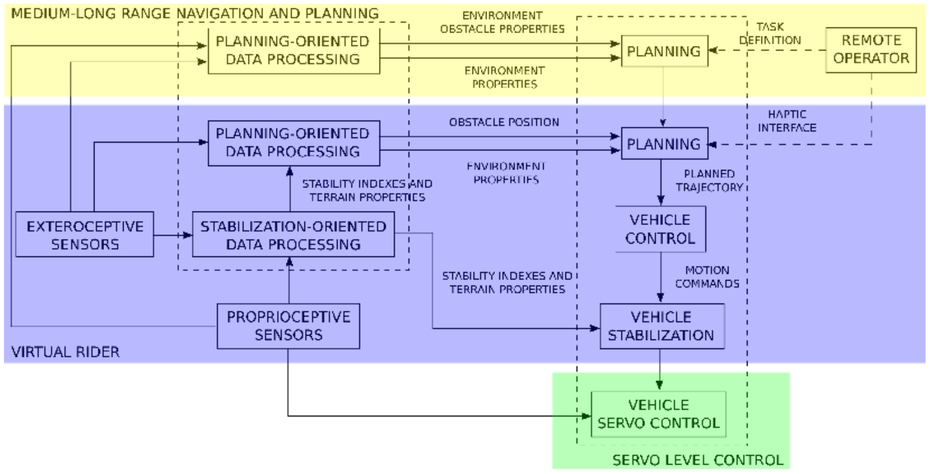


Fig. 3. Functional diagram of the controller architecture

dependability and robustness. Indeed, the choice of a PLC is a good compromise between the hard real-time requirement and the possibility of high level programming.

- a **high level** PC to implement the high level algorithms: the so called "virtual rider" (vision, terrain perception, localisation and mapping, obstacle and roll-over avoidance, etc.), the medium-long range navigation an planning, etc. For this purpose, an Industrial PC (2.16 GHz Intel Core Duo T7400, 2 MB L2 cache, 1024 MB DDR2 RAM, 5 PCI slots) provided by B&R AUTOMATION was selected.

A standard Ethernet communication link was selected to connect the two CPUs.

IV. SOFTWARE ARCHITECTURE

Fig. 5 shows the main modules of the software architecture implemented on the ATMR (as previously introduced, the functional architecture presented in Section III has been only partially implemented).
In the upper part there are the modules running on the PC, while in the lower part the tasks running on the PLC. On the PC two different middle-wares have been used to implement the overall system: ROS [9] and OROCOS [10]. While the former provides useful functionalities out of the box (e.g., laser sensor acquisition, mapping and planning) and thus it helps in speeding up the development, the latter has been used for critical control tasks having hard real-time requirements.

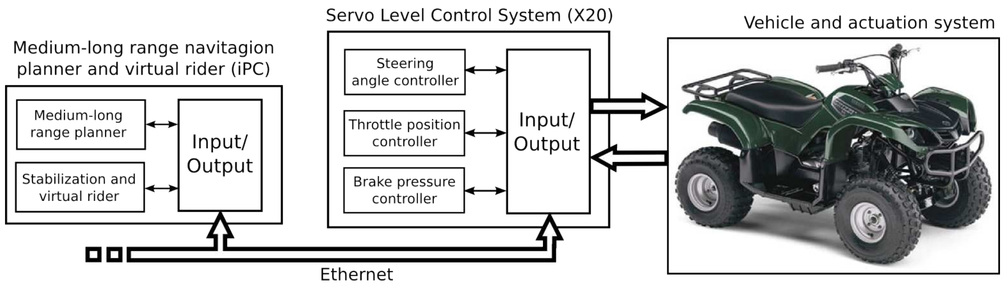
As already stated, the PLC runs the low-level actuator control loops and the sensor acquisition functionalities (e.g., speed, steering angle, stability indexes, etc.). The set points are sent to the low-level control loops (i.e., *speed*, *steer*, and *brake*) by the OROCOS task named MULTIPLEXER, which has the role of deciding whether the ATMR should be teleoperated, i.e. guided by a wireless JOYPAD, or a REMOTE CONTROL STATION (RCS), or autonomous, i.e. the CONTROLLER is in charge of trajectory following.

A simple trajectory follower has been implemented, decoupling the geometrical path following, that is accomplished acting on the steering angle, from the speed control. Following this idea, two independent PID control loops have been realised: one controlling the steering angle on the basis of the vehicle alignment and distance error [11], computed by the SEQUENCER module, the other one regulating the vehicle speed.

The ATMR position is estimated by an EXTENDED KALMAN FILTER (EKF) that uses the Ackerman kinematic model and integrates speed and steer measurements from the ATV sensors together with the position provided by a RTK-GPS with external correction (up to few centimetres accuracy). At the present stage, the magnetometer measurements of an inertial measurement unit are used to initialise the EKF heading estimate, but we plan to integrate them in the EKF once a proper dynamic model of the vehicle is developed.

The pose computed by the EKF module is also provided to the modules implemented under ROS, and it is used to align the point clouds acquired by a Sick LD-MRS laser range finder with the map of the environment.
This map in turn is used by a Model Predictive Control (MPC) based planner to generate the desired trajectory for the ATMR. This task is performed by a set of ROS nodes since most of the routines where already available under that middle-ware and the loose real-time requirements of planning were satisfied by ROS scheduling. It should be noticed that planning is a critical aspect when moving in rough terrains since, by carefully taking into account the constraints of the vehicle, safe trajectory can be planned. The result of this planning activity is then fed to the SEQUENCER module to be executed under real-time conditions.

When navigating an unknown environment unexpected, or unmapped, obstacles might appear; in this case the map need to be updated and the planning activity re-executed to take into account the new information. In our case the MPC based planner is re-executed continuously so this map update is managed in a natural way. However, MPC planning might



Servo Level Control System (X20)
Vehicle and actuation system
Medium-long range navitagion planner and virtual rider (iPC)
Steering angle controller
Throttle position controller
Brake pressure controller
Input/ Output
Medium-long range planner
Stabilization and virtual rider
Input/ Output
Ethernet

Fig. 4. Hardware architecture

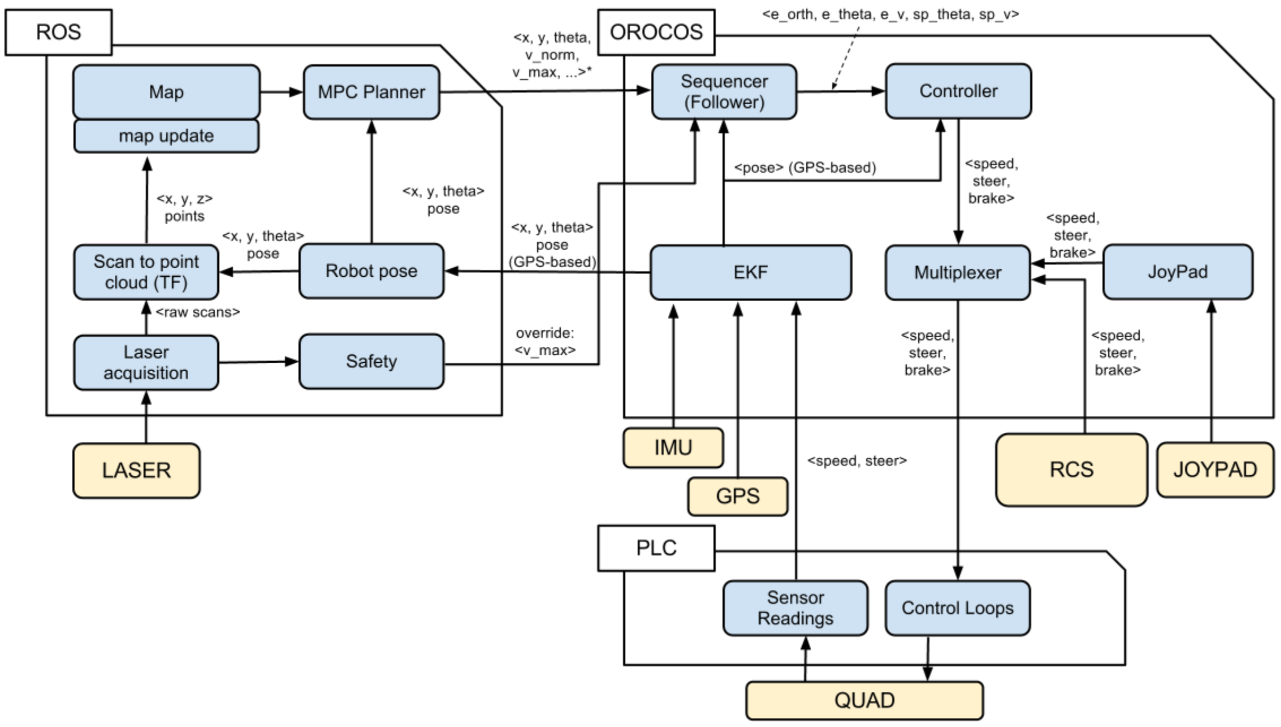


Fig. 5. Software architecture

take some time to compute a new plan or the computation can even fail. To cope with this possibility, a SAFETY module that, observing the point cloud generated by the sensor, overrides the maximum speed allowed for trajectory following has been introduced.

V. A MPC-BASED PLANNER

Including the vehicle model into the motion planning stage provides a planner which generates trajectories that can be easily followed by a mobile robot. This especially comes to the fore when a vehicle moves with high speed and operates on rough terrains. Using a simpler planner that does not take into account the mobile vehicle model might cause a fatal error due to the difference between the planned and executed trajectories. For this reason, the gradient based algorithms such as the navigation function or a variant of the D^* [12], [13], [14], in our case are not considered an acceptable solution.

Finding an optimal path on rough terrains, given a vehicle model and all information about the terrain, can be expressed as a two point boundary value optimal control problem (OCP). Including the terrain shape into an objective function for the OCP might result into a problem difficult to solve. Namely, the OCP softwares, including ACADO [15], the software used in this work, require a differentiable objective function. To overcome this problem, a kind of interpolation of the terrain shape must be applied. However, such an interpolation might be computationally intensive even for medium size terrains, and finding the best path solving an OCP might be impractical for real-time implementation.

The approaches [16], [17], [18], [19], [20], all consider the vehicle model to find the final path from an initial to the goal position. They use an appropriately selected state-space sampling technique, in which a planner propagates the vehicle model over these states toward the goal position. However, these approaches might easily miss some key state spaces yielding a solution being far from optimal. If the vehicle discovers different information during the execution, these approaches re-plan from scratch finding a new path to the goal position. In case of uncertain terrains, a frequent complete replanning makes it difficult to use the approach for real-time implementation.

In this work, we use an adapted real-time Model Predictive Control (MPC) based motion planner, introduced in [21] and [22]. At each time sample, the planner finds the best local trajectory (within the sensor range) given the current vehicle state and terrain information. Such an "on-line" optimisation during the task execution is in accordance with the MPC approach, hence the name. The MPC based motion planner easily accommodates for a vehicle model and any form of constraints into the optimisation set-up. In [21], the optimization has been performed using genetic algorithms in order to cover the control space of the vehicle model and to find the best solution at each time sample. In this paper, the objective function and the cost-to-go term are based on the terrain roughness. We locally interpolate roughness data (within the vehicle sensor range) at each time horizon into differentiable functions making it possible to use optimal control techniques provided by ACADO.

The MPC optimization problem can be expressed as an initial value OCP problem with an end-free position (eqs. 1-5). The task of this optimization is to find the input $\mathbf{u}$ of the vehicle (velocity and steering angle momentum for kinematic model) along the optimization horizon $t \in (t_0, t_0+T)$, that is over all potential candidate paths, by minimizing the cost function $J(\mathbf{u})$ given in (1). The integrand $\gamma(\mathbf{x},\mathbf{u})$ represents the local roughness estimated by the vehicle within the sensor range. We use the roughness-based navigation RbNF, which represents a cost-to-go map, to extract a cost-to-go term Γ required by the MPC optimization. The RbNF might be computed as an optimal or approximated cost-to-go map [23]. The former gives better results, but is computationally expensive for large scale terrains. Since in our work we experiment with a small-scale terrain, the computational issue is not addressed. When the vehicle senses new information during the task execution, the RbNF can be updated similarly to [13]. Eqs. (2-5) represent optimization constraints including the differential constraint related to the vehicle model (2), control constraints (3), the safe stopping constraint (4) and the constraint which ensures the decrease of the Γ in order to guarantee that the plan reaches the goal position (5).

$$J(\mathbf{u}) = \int_{t_0}^{t_0+T} \gamma(\mathbf{x},\, \mathbf{u})dt + \Gamma(t_0+T) \tag{1}$$

$$\frac{d}{dt}\mathbf{x} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} \tag{2}$$

$$\mathbf{u}(t) \leq \mathbf{u}_{max} \tag{3}$$

$$v(t_0+T) = 0 \tag{4}$$

$$\Gamma(\mathbf{r}(t_0+T)) < \Gamma(\mathbf{r}(t_0+T_1)) < \Gamma(\mathbf{r}(t_0)) \tag{5}$$

In some rare cases when ACADO fails, bringing back an infeasible solution or no solution, we use a backup strategy to guide the vehicle forward. In those cases, a planner selects a close way-point which is located along the steepest descent of the RbNF and solves for a two point boundary value OCP problem.

In the sequel, the aforementioned advantages of an MPC motion planner are summarised. An MPC based motion planner can easily accommodate for a vehicle model with all the required constraints. The planner might be near optimal (giving the current state information) due to "the optimality principle" since the RbNF is a near optimal estimator of the cost-to-go optimisation term. Since the MPC horizon can be arbitrarily chosen, a terrain shape interpolation required to get a differentiable objective function can be locally applied as in [24]. Having a differentiable objective function allows for using an OCP software. Using a software to solve a local OCP problem, like ACADO, covers much of the control and state space comparing to [16], [17], [18], [19], [20]. Finally, instead of repeating the complete path planning procedure from scratch when the vehicle senses new information, the RbNF can be easily updated similarly to [13].

VI. SIMULATION RESULTS

Fig. 6 illustrates a path generated by an MPC based planner. The path is drawn over a contour plot of the terrain roughness map. The terrain roughness map is computed by using terrain heights as in [25] and [22].
The example shows that the generated path avoids obstacles, follows less roughness regions (blue regions in Fig. 6) and reaches the goal position (start and goal positions are marked with a red and a pink disk, respectively).

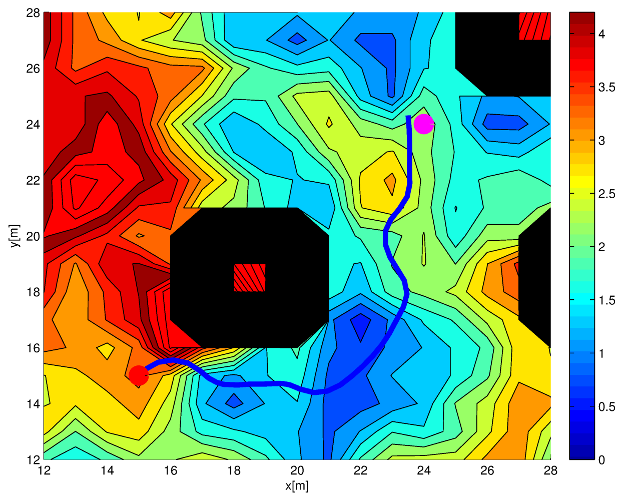


Fig. 6. An example of an MPC based solution

Finding a trajectory from an initial to the goal position using an OCP software is hardly feasible in real-time, especially for a large-scale terrain. The OCP solution finds the control inputs (velocity and steering angle for a kinematic model which is used in the simulations) that minimise traversed roughness, taking into account the required constraints. Some of the possible constraints might include: avoiding obstacles, velocity and steering limitations, vehicle stability and the RbNF decreasing to guarantee reaching the goal (see, e.g. [22]).

In some cases where the terrain is small-scale, it is possible to compute a solution in a reasonable time by an OCP software such as ACADO. For this reason, we have used a small terrain 50m x 50m to compare an optimal and a MPC based solutions exploring the MPC sub-optimality. Fig. 7 depicts 10 simulations in which the same rough terrain and different vehicle initial positions are used. The average sub-optimality of the MPC based path planner can be computed as

$$\alpha = \frac{1}{N}\sum \frac{roughness^{OCP}}{roughness^{MPC}} = 0.43$$

where N is the number of simulations. One might see that in the 9^{th} and 10^{th} simulations, ACADO did not find a feasible solution for the OCP problem (depicted by 0 in the picture).

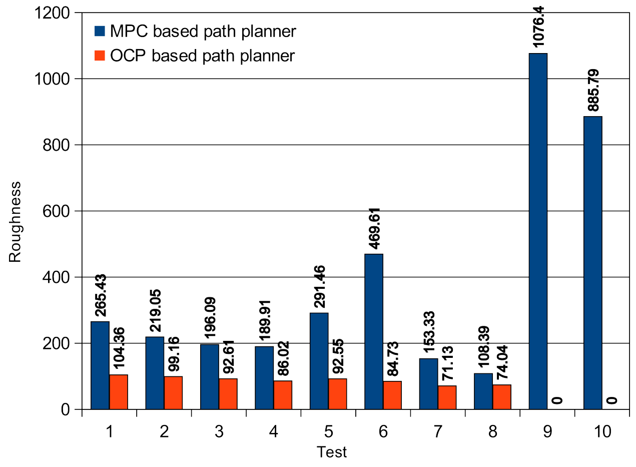


Fig. 7. I: Small-scale terrain. MPC and OCP solutions.

Fig. 8 depicts another example with 10 simulations on the same terrain with the same vehicle initial position and roughness shape, but with different obstacles. There are some examples where a MPC based solution has given a better result. This can be explained by the fact that an OCP software parametrises the control space in order to find the best solution. This might produce a solution that is not necessary the optimal one. In this example, the sub-optimality of the MPC path planner is much higher ($\alpha = 0.93$).

A two boundary value problem is difficult to solve in a feasible time on a large-scale terrain. For this reason, we use three different planners for a 500m x 500m terrain, a MPC based planner, a gradient based planner and a smooth gradient based planner. The gradient based planner is generated by the steepest descent of the RbNF. As already discussed, the gradient based planner is not considered as an acceptable solution in our work, since it does not take the vehicle model into account, and it is hard to predict how

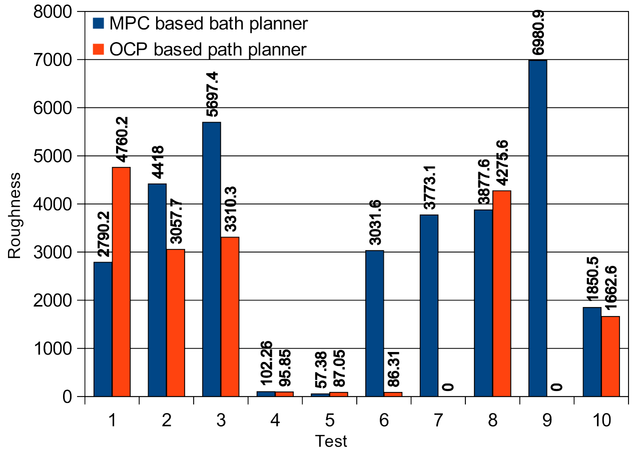


Fig. 8. II: Small-scale terrain. MPC and OCP solutions.

well the vehicle will follow such path. However, in order to validate the MPC based path planner, we introduce a smooth gradient based path planner which picks a point on the path obtained by the gradient based path planner and solves for a two boundary problem. Then, it repeats the procedure going towards the goal position. Fig. 9 compares the two planners on 10 different rough terrains. The sub-optimality of the MPC based path planner is $\alpha = 1.8$, which means that the MPC based planner performs better than the smooth gradient path planner. Again, this can be explained by the fact that the smooth gradient based path planner does take the vehicle model into account but only to follow the gradient based path planner.

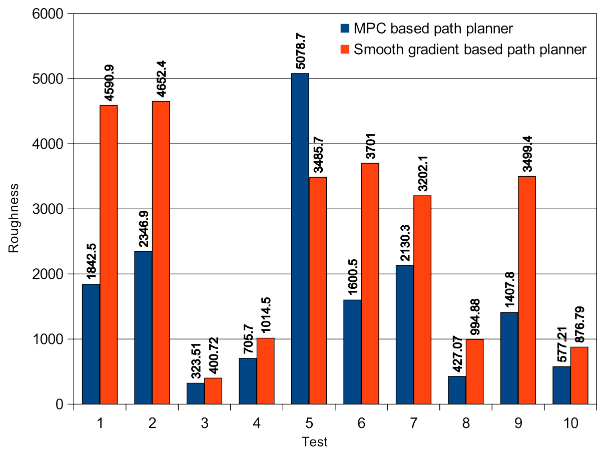


Fig. 9. Large-scale terrain. MPC and smooth gradient based solutions.

VII. CONCLUSIONS

This paper describes part of the work devoted to the development of an All-Terrain Mobile Robot, based on a commercial All-Terrain Vehicle, for high speed riding on difficult terrains.

Among the huge number of functionalities required to autonomously take the vehicle from a start to a goal position

through a safe path, accounting for terrain traversability, obstacles and vehicle constraints, the paper is focused on the hardware/software architecture and, above all, on the real-time implementation of a MPC-based planner. The issues involved in the implementation of the planner, using the open-source solver ACADO, are thoroughly discussed.
The simulation results show the effectiveness of the planner, and compare the paths computed by the MPC planner with those computed using a different approach.
An experimental validation of the MPC planning software, using the vehicle described in Section II, is ongoing. The results will be published soon.

References

[1] A. Garcia Cerezo, A. Mandow, J. Martinez, J. Gomez de Gabriel, J. Morales, A. Cruz, A. Reina, and J. Seron, "Development of ALACRANE: a mobile robotic assistance for exploration and rescue missions," in *IEEE International Workshop on Safety, Security and Rescue Robotics*, 2007.

[2] P. Debenest, E. Fukushima, and S. Hirose, "Proposal for automation of humanitarian demining with buggy robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2003, pp. 329–334.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[4] R. Lenain, B. Thuilot, C. Cariou, and P. Martinet, "Backstepping observer dedicated to tire cornering stiffness estimation: application to an All Terrain Vehicle and a farm tractor," in *IEEE/RSJ International Conference on Intelligent Robotics Systems*, 2007, pp. 1763–1768.

[5] J. van der Burg and P. Blazevic, "Anti-lock braking and traction control concept for all-terrain robotic vehicles," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1997, pp. 1400–1405.

[6] N. Bouton, R. Lenain, B. Thuilot, and J. Fauroux, "A rollover indicator based on the prediction of the load transfer in presence of sliding: application to an All Terrain Vehicle," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1158–1163.

[7] H. Utz, S. Sablatnög, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot applications," *IEEE Transaction on Robotics and Automation*, vol. 18, no. 4, pp. 493–497, 2002.

[8] L. Bascetta, G. Magnani, P. Rocco, and A. Zanchettin, "Design and implementation of the low-level control system of an All-Terrain Mobile Robot," in *International Conference on Advanced Robotics*, 2009.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *IEEE International Conference on Robotics and Automation - Workshop on Open Source Software*, 2009.

[10] H. Bruyninckx, "Open robot control software: the OROCOS project," in *IEEE International Conference on Robotics and Automation*, 2001, pp. 2523–2528.

[11] M. Linderoth, K. Soltesz, and R. Murray, "Nonlinear lateral control strategy for nonholonomic vehicles," in *American Control Conference*, 2008, pp. 3219–3224.

[12] A. Stenz, "Optimal and efficient path planning for partially-known environments," in *Proc. of the IEEE International Conference on Robotics and Automation*, 1994, pp. 3310 – 3317.

[13] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652–1659.

[14] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354 – 363, 2005.

[15] B. Houska and H. Ferreau, "ACADO toolkit - Automatic Control and Dynamic Optimization," http://acadotoolkit.org.

[16] C. J. Green and A. Kelly, "Toward optimal sampling in the space of paths," in *Proc. of the International Symposium of Robotics Research*, 2007, pp. 171 – 180.

[17] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *J. Field Robot.*, vol. 26, no. 3, pp. 308–333, 2009.

[18] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. J. Rob. Res.*, vol. 26, no. 2, pp. 141–166, 2007.

[19] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *Proc. The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, September 2005.

[20] T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *Journal of Field Robotics*, vol. 25, no. 10, pp. 325–345, 2008.

[21] A. Tahirovic and G. Magnani, "General framework for mobile robot navigation using passivity-based MPC," *IEEE Transactions on Automatic Control*, vol. 56, no. 1, 2011.

[22] ——, "Passivity-based model predictive control for mobile robot navigation planning in rough terrains," in *Proc. The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2010.

[23] ——, "A roughness-based rrt for mobile robot navigation planning," in *Proc. the 18th IFAC World Congress*, September 2011.

[24] F. J. Aguilar, F. Agüera, M. A. Aguilar, and F. Carvajal, "Effects of terrain morphology, sampling density, and interpolation methods on grid dem accuracy," *Photogrammetric Engineering & Remote Sensing*, vol. 71, no. 7, 2005.

[25] K. Iagnemma and S. Dubowsky, *Mobile Robots in Rough Terrain: Estimation, Motion Planning and Control with Application to Planetary Rovers*. New York: Springer Berlin / Heidelberg, 2004.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Sensor-based trajectory generation for safe human-robot cooperation

Andrea Maria Zanchettin and Bakir Lacevic

Abstract— **This paper presents a strategy for sensor-based trajectory generation in unstructured environments which guarantees the achievement of the goal position without incurring in local minima. The passivity of the closed-loop system renders this control scheme well-suited for human-robot cooperation, especially when the robot is supposed physically interact with humans. The given control law has been implemented and experimentally tested in a realistic scenario, demonstrating the effectiveness in driving the robot to a given configuration in a cluttered environment without any offline planning phase.**

I. INTRODUCTION AND MOTIVATIONS

Future paradigms in industrial robotics no longer require a physical separation between robotic manipulators and humans. Moreover, to optimize production, humans and robots will be expected to cooperate to some extent. In this scenario, involving a shared environment between humans and robots, common industrial robot controller might turn to be inadequate for this purpose. In order to obtain a natural and safe collaboration, robots will be equipped with sophisticated sensing devices and with human-aware control/planning capabilities.
In the literature many attempts in developing suitable robot reactions to unforeseen events have been presented by means of trajectory adaptation [4], [7] or modifications based on sensor readings, [2], [5]. However, in case of very unstructured environments, a better solution might be achieved with an advanced sensor-based motion and trajectory generation, rather than an online modification of an offline planned path. The aim of this research is to provide tools to overcome the current limitations in off-the-shelf robot controller and propose an online trajectory generator capable of understanding the environment and of computing a sensor-based trajectory to let the robot perform a prescribed task with a suitable level of safety.
A preliminary version of this work is discussed in [8], while paper complements the previous one by adding more details on the actual implementation of the sensor-based trajectory generation and discussing the outcome of more realistic experiments.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 - Challenge 2 - Cognitive Systems, Interaction, Robotics - under grant agreement No 230902 - ROSETTA.

A.M. Zanchettin is with Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza L. Da Vinci 32, 20133, Milano, Italy (email: zanchettin@elet.polimi.it).

B. Lacevic is with the Faculty of Electrical Engineering, University of Sarajevo, Sarajevo, Bosnia and Herzegovina (email: bakir.lacevic@etf.unsa.ba).

II. BACKGROUND MATERIAL

In this Section, the concept of danger field [6] is briefly outlined. Basically, the danger field is a scalar quantity that captures how much is a specific state of the robot (position and velocity) dangerous with respect to a generic point in the workspace. The intuition behind is that the danger field decreases with the distance from the robot whereas it increases with the robot's velocity, particularly if the robot moves towards the location where the field is computed at. For a simple case of a point robot located at $\boldsymbol{r_s} \in \mathbb{R}^3$, moving with the velocity $\boldsymbol{v_s} \in \mathbb{R}^3$, the elementary danger field at the position $\boldsymbol{r_j} \in \mathbb{R}^3$ could be defined as $DF_e = SDF_e + DDF_e$, where

$$SDF_e = \frac{k_1}{\|\boldsymbol{r_j} - \boldsymbol{r_s}\|^{\lambda_1}}, \tag{1}$$

$$DDF_e = \frac{k_2 \|\boldsymbol{v_s}\|^{\lambda_2} \left(1 + \cos\angle\left(\boldsymbol{r_j} - \boldsymbol{r_s}, \boldsymbol{v_s}\right)\right)}{\|\boldsymbol{r_j} - \boldsymbol{r_s}\|^{\lambda_3}}, \tag{2}$$

and SDF_e and DDF_e are the elementary static and kinetic danger fields respectively, $k_1, k_2, \lambda_1, \lambda_2, \lambda_3$ being positive parameters[1]. The elementary danger field can be generalized to its cumulative version that captures the position and velocity of the robot's i-th link by performing a path integration along the straight line that represents the wire model of the link:

$$DF = \int_0^1 SDF_e(s)ds + \int_0^1 DDF_e(s)ds. \tag{3}$$

For a robot with n links, the cumulative danger field induced at the locations of interest $r_j,\ j = 1, \ldots, n_{obst}$ (e.g., the relevant positions of obstacles) can be expressed as:

$$DF = SDF + DDF = \sum_{j=1}^{n_{obst}} \sum_{i=1}^{n} \int_0^1 \frac{k_1 ds}{\|\boldsymbol{r_j} - \boldsymbol{r_{i,s}}\|^{\lambda_1}} + \\ + \sum_{j=1}^{n_{obst}} \sum_{i=1}^{n} \int_0^1 \frac{k_2 \|\boldsymbol{v_{i,s}}\|^{\lambda_2} \rho_{i,j,s}}{\|\boldsymbol{r_j} - \boldsymbol{r_{i,s}}\|^{\lambda_3}} ds, \tag{4}$$

where $\rho_{i,j,s} = 1 + \cos\angle(\boldsymbol{r_j} - \boldsymbol{r_{i,s}}, \boldsymbol{v_{i,s}})$. Knowing $\boldsymbol{r_{i,s}}$ from the forward kinematics and letting $\boldsymbol{v_{i,s}} = \boldsymbol{J_{i,s}}\dot{\boldsymbol{q}}$, where $\boldsymbol{J_{i,s}}$ represents the Jacobian at point $\boldsymbol{r_{i,s}}$ on the manipulator, the cumulative danger field becomes a function of the configuration $\boldsymbol{q}$ and its time derivative $\dot{\boldsymbol{q}}$. Figure 1 shows volumetric representation of the danger field induced by a 6 DOF robotic manipulator. Notice the typical onion-like charecteristic of the danger-field very similar to the

[1]Note that this is a slight generalization of the danger field with respect to [6].

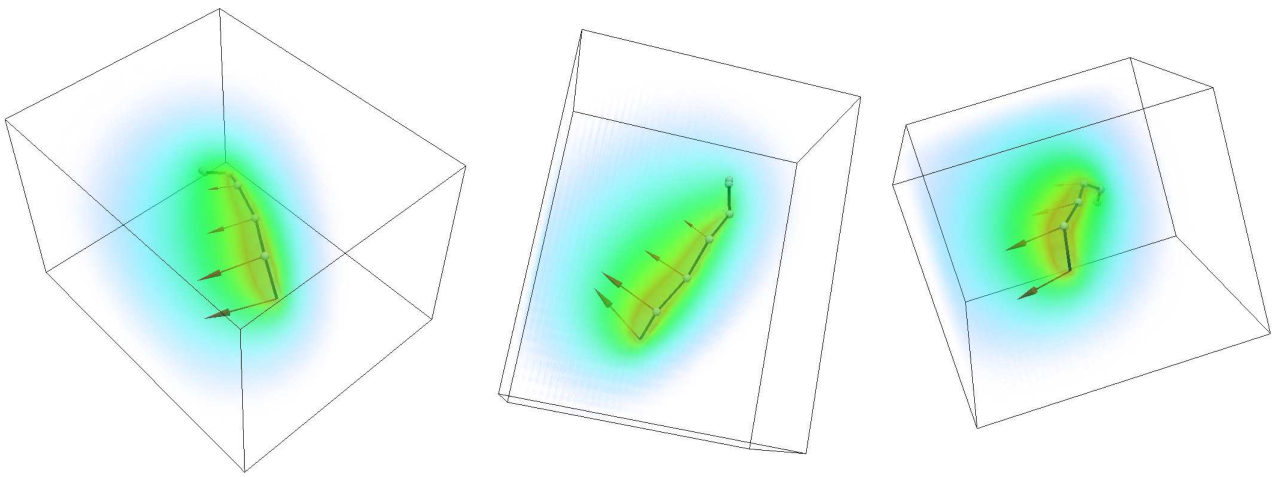


Fig. 1. Danger field around a 6 DOF manipulator: three different perspectives of the same robot state. The velocities of the link endpoints are indicated.

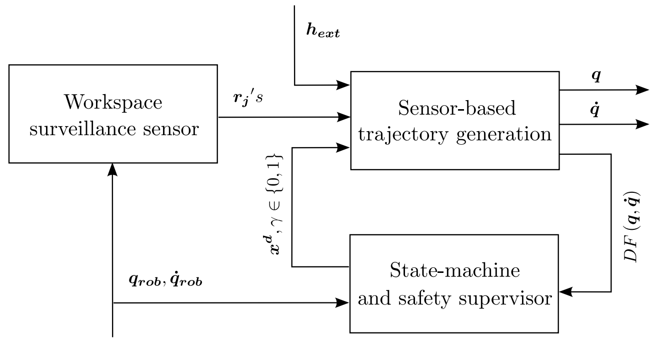
concept of minimum separation distance, developed in [1]. For the computational aspects of the danger field, the reader is referred to [6].

III. IMPLEMENTATION

The sensor-based module for online trajectory generation described in this paper works within a real-time loop of 4 ms, the same as the low-level axis control and consists of three blocks:

- a module for trajectory generation, communicating with the robot controller via a real-time Ethernet connection, see [3], and providing it joint references;
- a module for task description, implementing a state-machine and also responsible of temporary task suspension in case of physical cooperation initiated by the human operator;
- an interface to the workspace surveillance sensor, monitoring and tracking the human or any obstacle and providing a synthetic representation of them to the trajectory generation module.

The three-blocks architecture and their connections are depicted in Fig. 2. In the following, each of the three modules is detailed.


h_{ext}
Workspace surveillance sensor
r_j's
Sensor-based trajectory generation
q
$\dot{q}$
$x^d, \gamma \in \{0,1\}$
$DF(q, \dot{q})$
$q_{rob}, \dot{q}_{rob}$
State-machine and safety supervisor
Fig. 2. Components of the trajectory generation algorithm

A. Sensor-based trajectory generation

For a given desired Cartesian position/orientation $\boldsymbol{x_d}$ computed by the state-machine, a trajectory in the configuration, i.e. $\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)$ space is generated in real-time depending on sensor (proximity and force/torque) readings. In particular, the trajectory is computed by integrating the following dynamic system:

$$\begin{aligned}\ddot{\boldsymbol{q}} =& \boldsymbol{J}^T\left(\gamma \boldsymbol{K_P}(\boldsymbol{q})\,\boldsymbol{e} + \boldsymbol{h_{ext}}\right) - \boldsymbol{K_D}\dot{\boldsymbol{q}} \\ &+ \alpha(\boldsymbol{q})\,\boldsymbol{F}(\boldsymbol{\mu}, \dot{\boldsymbol{q}})\,\boldsymbol{\mu} - \beta(\boldsymbol{q}, \dot{\boldsymbol{q}})\left(\frac{\partial DF}{\partial \dot{\boldsymbol{q}}}\right)^T\end{aligned} \tag{5}$$

where

$$\boldsymbol{\mu} = -\frac{1}{2}\left[\|\boldsymbol{\nabla} U\|\,\|\boldsymbol{\nabla} SDF\| + \boldsymbol{\nabla} U\left(\boldsymbol{\nabla} SDF\right)^T\right]\left(\boldsymbol{\nabla} SDF\right)^T \tag{6}$$

is an evasive action preventing the robot hitting obstacles (or humans) in its working space,

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{\mu}, \dot{\boldsymbol{q}}) &= \begin{cases}\boldsymbol{I_n}, & \text{if } \boldsymbol{\mu}^T\dot{\boldsymbol{q}} \leq 0 \\ \boldsymbol{P}_{\perp}(\dot{\boldsymbol{q}}), & \text{otherwise}\end{cases} \\ \boldsymbol{P}_{\perp}(\dot{\boldsymbol{q}}) &= \boldsymbol{I_n} - \frac{\dot{\boldsymbol{q}}\dot{\boldsymbol{q}}^T}{\|\dot{\boldsymbol{q}}\|^2}\end{aligned} \tag{7}$$

is a weighting matrix, while $\alpha(\boldsymbol{q}), \beta(\boldsymbol{q}, \dot{\boldsymbol{q}}), \gamma$ are positive weighting parameters and $\boldsymbol{K_P}, \boldsymbol{K_D}$ are definite positive matrices. The overall system is depicted in Fig. 3. The dynamic system (5) has different properties such as guaranteed goal achievement with proved absence of local minima, enforced passivity mapping between external wrenches $\boldsymbol{h_{ext}}$ and joint velocities $\dot{\boldsymbol{q}}$ and others. The reader is referred to [8] for a more rigorous explanation of these concepts.

B. Workspace surveillance and geometric representation of obstacles

For workspace surveillance a range camera (MICROSOFT KINECT) with the OPENNI drivers have been selected. The output of the sensor consists of a segment representation

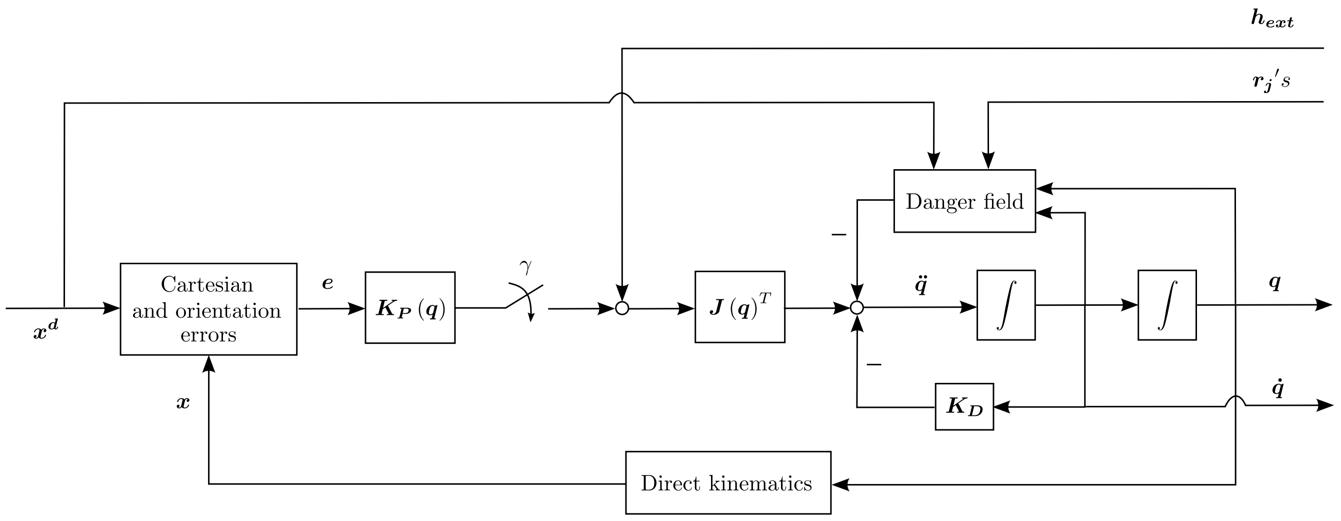


Fig. 3. Block diagram of the sensor-based trajectory generation

of the human silhouette capturing the position of anatomical points along the body (head, shoulder, elbow, wrist, hip, etc.), see Fig. 4.

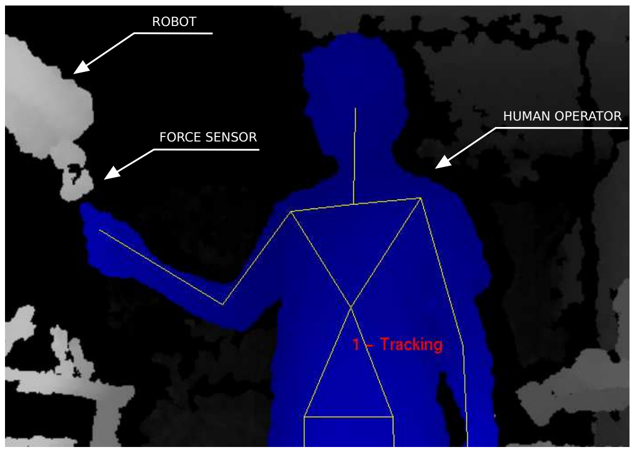


Fig. 4. Robot's workspace as seen from the surveillance range camera

A set of interesting points where to compute the danger field should be selected. Therefore, a simple and fast algorithm compatible with the representation of obstacles provided by the KINECT is described in the following.
While the robot is described as a chain of segments, a generic obstacle (the human operator in this work) can be decomposed in a set of segments, spheres or represented by a more generic triangular meshed surface. Algorithmic primitives based on quadratic optimization have been developed to estimate the closest points of each obstacles to the robot. For example, a segment $\boldsymbol{\mathcal{S}}$ can be parameterized by means of a vertex $\boldsymbol{P}$ and a vector $\boldsymbol{d}$ such that all the points belonging to the segment can be written as follows:

$$\boldsymbol{\mathcal{S}}(t) = \boldsymbol{P} + t\boldsymbol{d}, \;\; 0 \le t \le 1 \tag{8}$$

The minimum distance between two segments, hence the closest point to each link of the robot, can be obtained by solving the following constrained optimization problem

$$\begin{aligned} &\min_{t_1,t_2} \left\|\boldsymbol{\mathcal{S}_1}(t_1) - \boldsymbol{\mathcal{S}_2}(t_2)\right\|^2 \\ &\text{subject to } 0 \le t_1, t_2 \le 1 \end{aligned} \tag{9}$$

On the other hand, a triangle $\boldsymbol{\mathcal{T}}$ is parameterized by means of a vertex $\boldsymbol{V}$ and two vectors $\boldsymbol{e_0}, \boldsymbol{e_1}$. This way all the points belonging to the triangle can be written as follows:

$$\boldsymbol{\mathcal{T}}(u,v) = \boldsymbol{V} + u\boldsymbol{e_0} + v\boldsymbol{e_1} \tag{10}$$

subject to the following constraints

$$\begin{aligned} 0 &\le u \le 1, \\ 0 &\le v \le 1, \\ u + v &\le 1 \end{aligned} \tag{11}$$

Therefore the closest point on the robot link can be computed by simply solving the following optimization problem:

$$\begin{aligned} &\min_{t,u,v} \left\|\boldsymbol{\mathcal{S}}(t) - \boldsymbol{\mathcal{T}}(u,v)\right\|^2 \\ &\text{subject to } 0 \le u, v, t \le 1, u + v \le 1 \end{aligned} \tag{12}$$

In this case study, where only the human operator represents an obstacle, its body is decomposed into four segments (right and left arm and forearm), one triangle (torso) and one sphere (head). The algorithm described so far to compute the closest point to the robot is then applied providing a set of points where the trajectory generation algorithm will compute the danger field. The depicted scenario is sketched in Fig. 5.

C. State-machine and safety supervisor

A state-machine has been implemented to both monitor the execution and eventually suspend the task when the user gets too close to the robot. During the normal execution, the state-machine communicates with the trajectory generation module sending the sequence of Cartesian position/orientation references. When the value of the danger field exceeds a prescribed threshold DF_{thr}^{sup}, the task is suspended by setting $\gamma = 0$. In this situation the trajectory generation is still

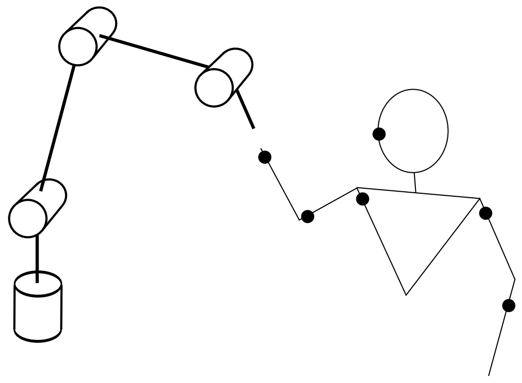


Fig. 5. Points on the human body where danger field will be computed

active but without any reference position/orientation. This is meant to allow the user to manually guide the robot motion, e.g. to teach the robot new positions. When the

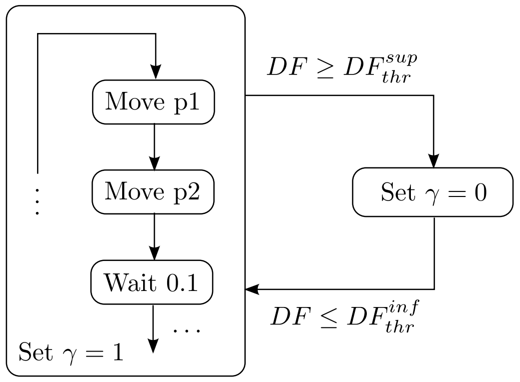


Fig. 6. State-machine governing task execution and suspension

danger field assumes values lower than another threshold $DF_{thr}^{inf} < DF_{thr}^{sup}$, the main task is resumed from the point where it was suspended by setting $\gamma = 1$. Figure 6 sketches the task suspension/resumption mechanism.

IV. EXPERIMENTS

As a validation of the proposed control strategy, some experimental tests have been carried out on an industrial manipulator. The 6 axes ABB IRB-140 robot with 6 kg payload was used for this purpose. The manipulator is equipped with an ATI force/torque sensor mounted on the robot end-effector and interfaced with the controller. All the sensors are acquired and processed within an external real-time LINUX PC interfaced with the ABB IRC5 industrial robot controller using a communication link developed, see [3].

A realistic industrial scenario has been arranged in order to resemble a typical machine tending task: the robot handles a workpiece from a storage station and transports it to a position located on the other side of its workspace.

The operator, regarded as a moving obstacle, is able at any time to enter the working area of the robot for inspection and for this reason a proper safety action should be guaranteed, possibly without interrupting the production. During the experiment, the same production cycle has been repeated 3 times. During the last two repetitions the human operator enters the scene and walks towards the pick-up station approximately at time instants $t = 25s$ and $t = 70s$. This is confirmed by the profile of the danger field, see Fig. 9, which captures a more dangerous situation due to the vicinity of the human. Correspondingly the robot first tries to reduce the speed and then, since the danger field exceeds the desired threshold, suspends the task to allow the physical cooperation with the operator (this situation is highlighted with gray bands in the Figures). The accompanying video shows the execution of the experiment.

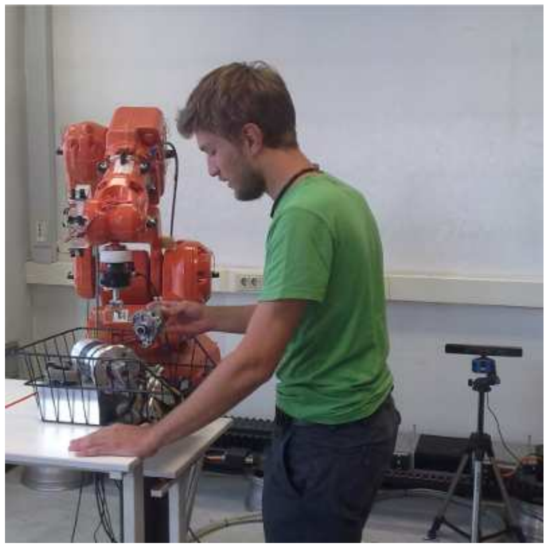


Fig. 7. Human-robot coexistence ($\gamma = 1$)

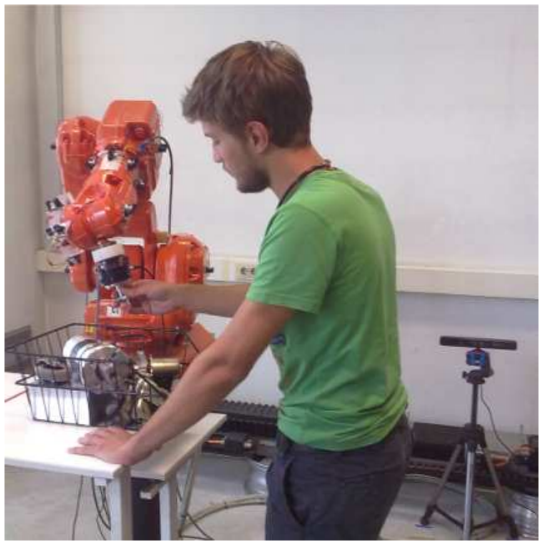


Fig. 8. Human-robot physical cooperation ($\gamma = 0$)

V. CONCLUSIONS

This paper complements [8] detailing the implementation of a newly conceived passivity-based control scheme for robotic manipulators in cluttered environments. The control

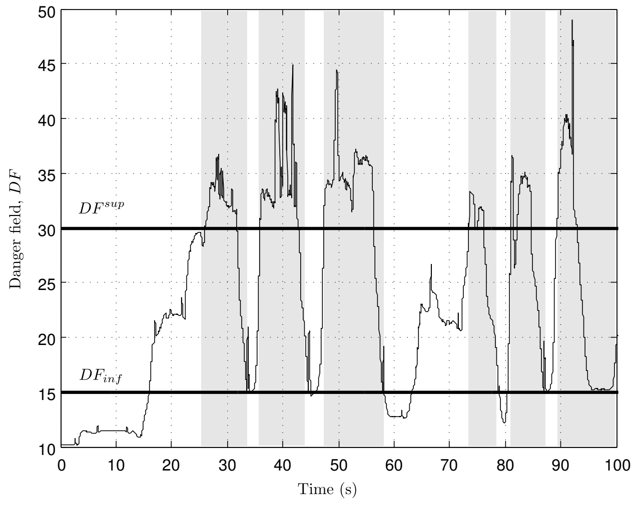


Fig. 9. Profile of the danger field and safety thresholds

law has been experimentally verified in a scenario involving an industrial manipulator physically cooperating with a human operator, demonstrating the possibility to safely move the robot in given configurations without any offline planning phase.

REFERENCES

[1] *ISO/TS 15066 "Robots and Robotic Devices - Collaborative industrial robots".*

[2] L. Bascetta, G. Magnani, P. Rocco, R. Migliorini, and M. Pelagatti. Anti-collision systems for robotic applications based on laser time-of-flight sensors. In *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pages 278 –284, july 2010.

[3] A. Blomdell, I. Dressler, K. Nilsson, and A. Robertsson. Flexible application development and high-performance motion control based on external sensing and reconfiguration of abb industrial robot controllers. In *Proc. ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, Anchorage, AK, jun 2010.

[4] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052, 2002.

[5] S. Haddadin, R. Belder, and A. Albu-Schaffer. Dynamic motion planning for robots in partially unknown environments. In *IFAC World Congress 2011, IFAC 2011*, pages 6842–6850, 2011.

[6] B. Lacevic and P. Rocco. Kinetostatic danger field - a novel safety assessment for human-robot interaction. In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010*, pages 2169–2174, 2010.

[7] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 802–807, may 1993.

[8] A.M. Zanchettin, B. Lacevic, and P. Rocco. A novel passivity-based control law for safe human-robot coexistence. In *IEEE/RSJ 2012 International Conference on Intelligent Robots and Systems, IROS 2012*, 2012.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Online Kinodynamic Trajectory Generation using Nonlinear Filters: a Multi-Dimensional Space Approach

Marcello Bonfè, Cristian Secchi and Enea Scioni

Abstract— **The paper describes trajectory generation algorithms based on nonlinear smoothing filters, which can be applied to kinodynamic motion planning for mobile robots in a two-dimensional or three-dimensional space. The trajectory generators operate fully online to reach a given target point according to a pursuit-based logic. The logic sets reference inputs and dynamic constraints for a set of nonlinear filters, whose output signals are finally used to compute the trajectory compatibly with the kinematic model and the dynamic features of the robot tracking such a trajectory. Since the target point of the trajectory generator can be changed at any time during motion, the algorithm can execute online smoothing of straight-line reference paths, for example composed of via-points assigned by a global planner on the basis of obstacle avoidance rules.**

I. Introduction

Generation of smooth motion trajectories, subject to kinematic and dynamic constraints, is a fundamental issues in robotics. Motion planning is usually separated into the geometric problem (*path planning*), whose solution is a parametric path depending on an unspecified timing law to become executable, and the actual *trajectory planning*, in which the timing law for a given path is designed. Path planning for mobile robots and autonomous vehicles must take into account nonholonomic constraints and/or obstacles, while constraints involved by dynamics laws and actuator bounds must be addressed during the design of either timing plans or tracking control algorithms, see [1] (Ch. 4,7,8).

The literature on path planning describes many solutions to avoid collision with static or moving obstacles, see [2]. If static obstacles are considered, robot motion can be planned in advance and efficient, but computationally demanding, interpolation methods can be applied. However, when the tasks of the robot or the positions of obstacles are not fully known a priori, paths must be adapted or re-planned online, within hard real-time constraints. Moreover, any path must be associated with a feasable timing law to become a trajectory compatible with the dynamic features of the robot.

The trajectory generation solution first introduced by [3] is specifically designed for online execution, thanks to its limited computational demand and to its discrete-time behavior. The output of the proposed trajectory generator is fully specified (in cartesian coordinates) with respect to time and has continuous curvature, so that it is compatible with with *kinodynamic constraints* (i.e. bounded linear/angular velocity and acceleration) of unicycle-like robots. Finally, the input of the trajectory generator can be either a sequence of fixed via-points, along a straight-line path, or a time-varying reference point, which may be provided by *global* planning algorithms implementing obstacle avoidance, but not necessarily precise geometric path design, since the smoothing action of the trajectory generator compensates path discontinuities.

In this paper, we extend the contribution of [3] by analysing the geometric features of generated trajectories and by presenting some obstacle-avoidance use cases (Section III). Moreover, Section IV describes an extension of the motion algorithms to the three-dimensional case. Finally, Section V, contains practical guidelines for the implementation of the trajectory generator on a low-cost DSP or microcontroller and reports real experimental results.

M. Bonfè and Enea Scioni are with the Engineering Department (ENDIF), University of Ferrara, 44122 Ferrara, Italy. E-mail: marcello.bonfe@unife.it, enea.scioni@unife.it

C. Secchi is with the Deparment of Science and Methods for Engineering (DISMI), University of Modena and Reggio Emilia, 42122 Reggio Emilia, Italy. E-mail: cristian.secchi@unimore.it

II. Two-dimensional trajectory generation

The nonlinear filtering approach adopted in this paper has been exploited first, for one dimensional motion, in [4], a paper that describes the design of a Variable Structure (VS) dynamic system acting as a smoothing filter for rough (steps, discontinuous ramps, etc.) position reference signals. The filter achieves perfect tracking of the reference signal in minimum time, compatibly with constraints on the first and second-order derivative of the filter output. The VS system is composed by a chain of two integrators and a nonlinear controller that guarantees the requirement on bounded output derivatives and minimum time response. The block diagram of the filter, in the discrete-time case[1], is shown in Fig.1.

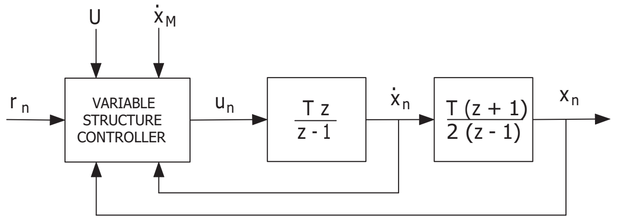


Fig. 1. Block diagram of a nonlinear filter for trajectory generation

The VS controller of the filter receives at each sampling instant nT the following inputs: the position reference signal r_n and its time derivative $\dot{r}_n$, the current outputs of the

[1]All the filters described in the paper are discrete-time systems. The sampling instant nT is dropped in all subsequent equations, to simplify notation, and differentiation/integration are improperly denoted as the equivalent continuous-time operations

integrators ($\dot{x}_n$ and x_n), the bounds U on the acceleration/deceleration and $\dot{x}_M$ on the velocity absolute value. Therefore, such bounds can be changed in real-time. The control law proposed by [4] is a Sliding Mode (SM) controller (see [5]), ensuring that the resulting trajectory always reaches a sliding surface in the error phase plane (with coordinates $y_n = x_n - r_n$ and $\dot{y}_n = \dot{x}_n - \dot{r}_n$), in minimum time, without overshooting and without exceeding the bounds U and $\dot{x}_M$. Once reached the surface, the SM brings the filter towards a perfect tracking condition, which is also achieved in minimum time.

The extension of this method to generate trajectories in the two-dimensional operational space of a nonholonomic mobile robot has been presented by [3]. The key idea is to generate with two separate nonlinear filters, both structured as shown in Fig. 1, linear velocity and orientation of the trajectory, compatibly with dynamic constraints, and then take into account kinematic constraints to obtain first-order time derivatives of the trajectory in the cartesian space, whose subsequent numerical integration defines the desired position vector. Considering the class of unicycle-like mobile robots, kinematic constraints require the robot configuration vector $[x, y, \theta]^T$, being θ the orientation of the robot w.r.t. to the fixed cartesian frame, to comply with:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{1}$$

in which v and ω, respectively *driving velocity* and *steering velocity*, are in most applications assumed as the control inputs. Considering only the first two rows of Eq.(1), it is clear that the generation of two sufficiently smooth signals $v(t)$ and $\theta(t)$ and the integration of $\dot{x}$ and $\dot{y}$ equals to generate a trajectory compatibly with the kinematic model of the unicycle. Dynamic constraints in the generation of $v(t)$ and $\theta(t)$ can be taken into account recalling that the acceleration of a planar trajectory is given by the sum of *tangential* and *radial* acceleration orthogonal vectors, whose lengths are respectively $a_t = \dot{v}$ and $a_r = v\,\dot{\theta} = v\,\omega$. Both components must be bounded to preserve the robot from slipping. Limiting radial acceleration by reducing (without zeroing) the driving velocity when $\omega \neq 0$, involves a limitation also on the scalar curvature of the path, since the latter is $\kappa = \omega/v = \omega^2/a_r$.

Minimization of time and space required to reach a given target position can be achieved by maximizing linear velocity, within actuator limitation, when the robot is oriented towards the target, and instead set driving speed as the ratio between maximum allowed radial acceleration and maximum steering velocity, when the robot must change its orientation to point towards the target. The generation of this change of orientation can be obtained following the so-called *planar pursuit-evasion* approach and applying the equations describing the geometric relationship between a moving target, whose position and linear velocity are denoted as $[x_t, y_t]^T$ and v_t in Fig. 2, and a tracking point, characterized by $[x_d, y_d]^T$ and v_d in the figure. Such pursuit-evasion equations can be found in [6].

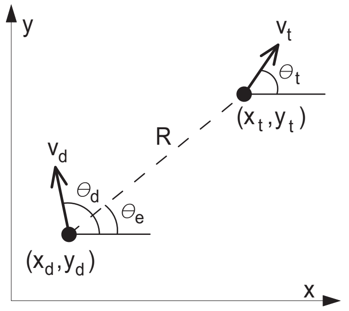


Fig. 2. Target interception geometry

In this section $v_t = 0$ (i.e. fixed target positions) is assumed, but the approach can be easily extended to consider moving targets, as shown in Section III. If the orientation of the trajectory generated by the filter is different from θ_e, then this value should be set as the reference for the nonlinear filter. During the turning phase necessary to align with the target, steering velocity should be the highest possibile, while driving velocity must be set to a maximum compatibly with the bound on radial acceleration. Once that target alignment is achieved, the final position is reached by triggering a deceleration to zero velocity, as soon as the distance from the target is equal to the space required to stop with given bounds on $\dot{v}_d$ and $\ddot{v}_d$.

The block diagram of a nonlinear smoothing filter that generates trajectories for a unicycle-like robot, in the way just described, is shown in Fig. 3.

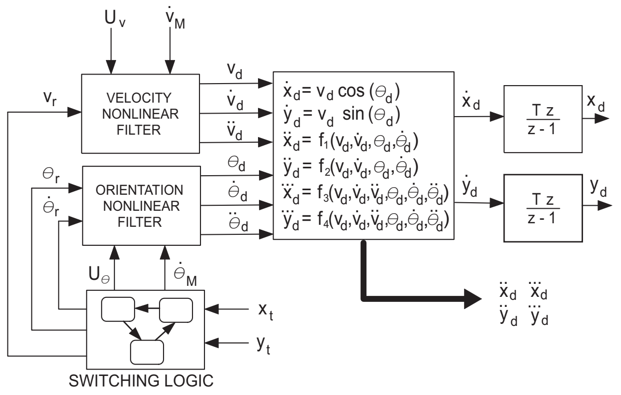

U_v
$\dot{v}_M$
v_r
VELOCITY NONLINEAR FILTER
v_d
$\dot{v}_d$
$\ddot{v}_d$
θ_r
$\dot{\theta}_r$
ORIENTATION NONLINEAR FILTER
θ_d
$\dot{\theta}_d$
$\ddot{\theta}_d$
$\dot{x}_d = v_d \cos(\theta_d)$
$\dot{y}_d = v_d \sin(\theta_d)$
$\ddot{x}_d = f_1(v_d, \dot{v}_d, \theta_d, \dot{\theta}_d)$
$\ddot{y}_d = f_2(v_d, \dot{v}_d, \theta_d, \dot{\theta}_d)$
$\dddot{x}_d = f_3(v_d, \dot{v}_d, \ddot{v}_d, \theta_d, \dot{\theta}_d, \ddot{\theta}_d)$
$\dddot{y}_d = f_4(v_d, \dot{v}_d, \ddot{v}_d, \theta_d, \dot{\theta}_d, \ddot{\theta}_d)$
$\dot{x}_d$
$\dot{y}_d$
$\frac{T z}{z-1}$
$\frac{T z}{z-1}$
x_d
y_d
U_θ
$\dot{\theta}_M$
x_t
y_t
$\ddot{x}_d$ $\dddot{x}_d$
$\ddot{y}_d$ $\dddot{y}_d$
SWITCHING LOGIC

Fig. 3. Block diagram of the nonlinear smoothing filter for mobile robotics

The block *Velocity Nonlinear Filter* of Fig. 3 is a filter with the structure of Fig. 1, but its output is a driving velocity v_d (*instead* of a desired position), that perfectly tracks the discontinuous reference velocity v_r with bounds on first and second-order derivatives ($|\dot{v}_d| \leq \dot{v}_M$ and $|\ddot{v}_d| \leq U_v$). The *Orientation Nonlinear Filter* differs from the previous one only in the calculation of the tracking error, which is limited in the interval $[-\pi; \pi]$. The output of the filter is the desired orientation θ_d tracking at best θ_e with bounded derivatives ($|\dot{\theta}_d| \leq \dot{\theta}_M$ and $|\ddot{\theta}_d| \leq U_\theta$).

The *Switching Logic* is a finite state machine that sets the reference signals v_r, θ_r and $\dot{\theta}_r$ ($\dot{v}_r = 0$ at any time) and the bounds of the orientation filter. The state machine contains four states, namely *Turning*, *Aligned*, *Stopping* and *Limit Speed*, and it formalizes the target approaching sequence previously described (see [3] for the full details of the state machine).

The behavior of the Switching Logic and, therefore, the resulting geometric properties of the trajectories generated by the filter, depend on the following parameters:

- v_M: maximum allowed driving velocity. It is set as a reference for the *Velocity Nonlinear Filter* only when the trajectory is aligned with the target (state *Aligned*).
- A_{RM}: maximum allowed radial acceleration. It affects the resulting curvature of the path and it is used to calculate the driving velocity limit in the states *Turning* and *Limit Speed*. The latter state is required to slow down before starting a curve.
- $\dot{\theta}_B$: maximum allowed steering velocity. It is used to limit the rate of change of the orientation along a curve (i.e. *Turning* state), when the *Orientation Nonlinear Filter* is forced to track θ_e.
- R_{stop}: distance required to decelerate from $v_r = v_M$ to $v_r = 0$. This value, used in the guard condition to switch from state *Aligned* to *Stopping*, can be easily calculated since the output of the velocity filter is a standard profile with trapezoidal first-order derivative (*acceleration*, in this case). R_{stop} is obtained integrating further this velocity profile.

The outputs of the velocity and orientation nonlinear filters must be combined as follows:

$$\begin{aligned}\dot{x}_d &= v_d \cos\theta_d \\ \dot{y}_d &= v_d \sin\theta_d\end{aligned} \tag{2}$$

and integrated to finally obtain $[x_d, y_d]^T$. Higher order time derivatives $\dot{x}_d, \dot{y}_d, \ddot{x}_d, \ldots$ are bounded up to the third order and can be calculated from the full output vectors of the two nonlinear filters, with equations obtained by differentiation of Eq.(2).

Remark 1: Target position $[x_t, y_t]^T$ can be abruptly changed at any time. This event causes $\theta_d \neq \theta_e$ and, therefore, forces a reduction of driving velocity, set to $A_{RM}/\dot{\theta}_B$ before starting the turning phase. In this way, radial acceleration is guaranteed to be bounded by A_{RM}.

Remark 2: The second-order time derivatives of velocity and orientation are bounded, so that also third-order derivatives $[\dddot{x}_d, \dddot{y}_d]^T$ are limited. Moreover, this implies that the curvature of the resulting path is continuous.

III. Via-point commutation and case-studies

The role of the discrete-time nonlinear filter described in Fig. 3 is to calculate online a smooth trajectory approaching and reaching a target position. At each sampling instant, the output vector of the filter is updated according to the error between the current orientation of the trajectory and the one that allows to reach the target point, compatibly with kinodynamic constraints. The resulting cartesian path depends inherently on the time-varying setting of target position. If the aim is to generate a trajectory among obstacles, a key issue that needs to be solved is the adequate placement of a sequence of via-points and, in addition, the definition of conditions for selecting online one of these via-points as the target of the smoothing filter. These tasks could be executed by a higher-level planning algorithm, that is aware of the location of obstacles and of the geometric properties of the trajectory generated by the filter.

In particular, there are two features of the proposed nonlinear filter that must be considered. First of all, the design of the Switching Logic guarantees that radial acceleration is bounded by A_{RM}, by means of a *Limit Speed* state that precedes any turn. Moreover, since $\dot{\theta}_d$ can only vary linearly with a constant rate U_θ, it means that the curvature and its time derivative are bounded by:

$$\kappa_M = \frac{\dot{\theta}_B^2}{A_{RM}}; \quad \dot{\kappa}_M = \frac{U_\theta\, \dot{\theta}_B}{A_{RM}} \tag{3}$$

Therefore, the trajectories generated with the proposed approach have the same properties of Continuous-Curvature paths (CC-paths) described by [7] or [8], namely they are composed of linear segments, clothoid arcs (i.e. arcs with linearly increasing curvature) and circular arcs of radius κ_M^{-1}. The key differences are that the trajectory generator described in previous section does not compute esplicilty the clothoids, since the latter are inherently the output of the nonlinear filter, and that the resulting clothoid arcs are specified w.r.t time, instead of arc length. Moreover, since the driving velocity is exactly $v_d = A_{RM}/\dot{\theta}_B$ in any curved part of the trajectory, by design, there is no need for a time-scaling law to guarantee the geometry of the path or to comply with dynamic constraints, which is instead the approach of [8]. The geometric features of CC-paths (see [7]) can be revisited for the case under study considering the trajectory plotted in Fig.4, including a curve near a via-point denoted with A. In the figure, the current state of the nonlinear filter Switching Logic is highligted by the color of the path. As can be seen, the first part of the curve is a clothoid arc, more precisely an arc whose sharpness is:

$$\sigma_M = \frac{\dot{\kappa}_M}{v_d} = \frac{U_\theta}{v_d^2}. \tag{4}$$

The objective of this analysis is to determine the condition for changing the target point of the nonlinear filter from a via-point to another and obtain a smooth path that remains as much as possible along the straight lines connecting via-points. The key information that is required to evaluate this condition is the length AB, which is the distance from the via-point at which a curve must be initiated to obtain a symmetric trajectory analogous to the paths described by [7]. Assuming, without loss of generality, that B is the origin, from the theory of clothoids we obtain the coordinates of B', the point in which the curvature reaches the maximum

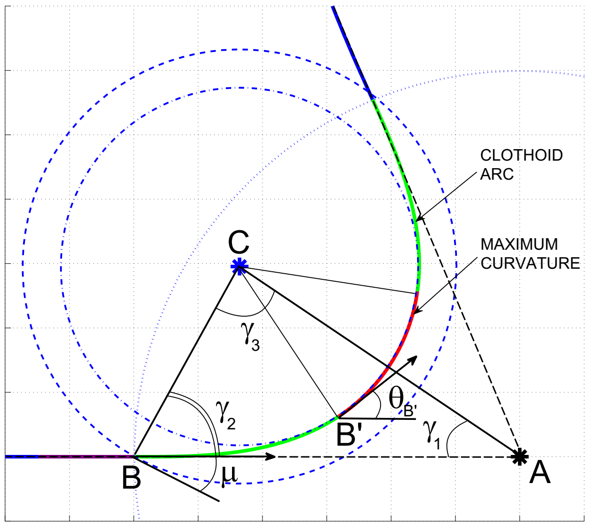


Fig. 4. Trajectory generated by nonlinear filtering and Switching Logic states: *Limit Speed* (purple), *Turning* with increasing curvature (green), *Turning* with maximum curvature (red), *Aligned* (blue)

admissible value (C_F and S_F denote the Fresnel integrals):

$$\begin{aligned} x_{B'} &= \sqrt{\frac{\pi}{\sigma_M}}\, C_F\left(\sqrt{\frac{\kappa_M^2}{\pi\sigma_M}}\right) \\ y_{B'} &= \sqrt{\frac{\pi}{\sigma_M}}\, S_F\left(\sqrt{\frac{\kappa_M^2}{\pi\sigma_M}}\right) \end{aligned} \tag{5}$$

and the orientation:

$$\theta_{B'} = \frac{\kappa_M^2}{2\sigma_M} \tag{6}$$

It is then possible to calculate the length BC, knowing the maximum radius of curvature κ_M^{-1}, and the angles μ, γ_1, γ_2 and γ_3, from basic trigonometry. Finally, applying known theorems on triangles, we obtain:

$$AB = BC\,\frac{\sin\gamma_3}{\sin\gamma_1} \tag{7}$$

It is important to note that all the results, excluding Eq.7, depend only on the bounds applied to the nonlinear filter. If such bounds constant, the only value that should be calculated online and for each via-point is AB. Otherwise, since the bounds of the nonlinear filter can also be changed in real-time, this additional degree of freedom can be used to increase the curvature of the trajectory in case of narrow passages, in which the via-point should be approached as much as possible.

Case study 1: path smoothing among static obstacles

Obstacle-avoiding paths among static obstacles can be obtained by means of a properly defined interaction between the nonlinear filter and a global planner. The latter can place a set of via-points, using any collision-avoidance algorithm, among the obstacles and can calculate for each via-point the commutation distance (which is actually the sum of AB plus the distance required to reduce linear velocity from v_M to $A_{RM}/\dot{\theta}_B$), so that continuous-curvature trajectories smoothing straight-line paths remain collision-free. The complete sequence of via-points, together with related commutation distance, can be stored in a queue and can be managed online by a simple algorithm, extracting from the queue a via-point to be selected as the current target point of the filter and continuously updating the distance from the output of the filter to the via-point to promptly detect the commutation condition. As an example, Fig. 5 shows a trajectory generated applying a sequence of fixed via-points, marked by stars, along a path composed of straight lines among obstacles (grey rectangles). The target point of the nonlinear filter is always changed exactly when the filter output is at a distance from the current via-point such that at the end of the turning phase, triggered by the target commutation, the trajectory is exactly oriented along the straight line connecting the current via-point with the subsequent one.

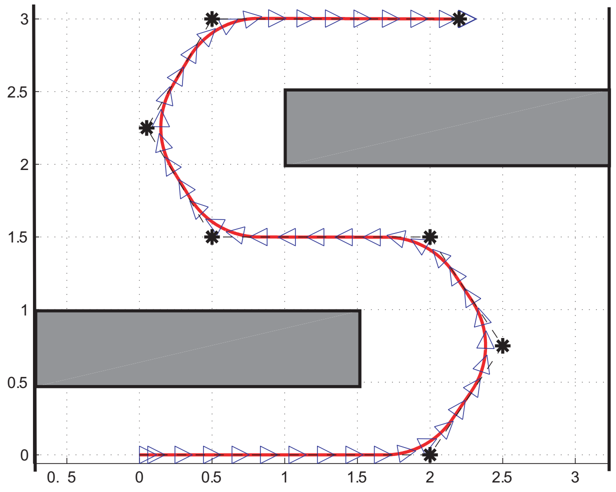


Fig. 5. Straight-line path among obstacles, with via-points (stars) and resulting smooth trajectory generated by the nonlinear filter (red)

Case study 2: avoiding moving obstacles

The target point for the nonlinear filter can also have a given linear velocity. A practical case that may be addressed using this feature of the proposed nonlinear filter is a highway-like context, in which slower vehicles move in the same direction of the controlled mobile robot. If the robot has to pass in front of the slower vehicle, it is possible apply the following target point commutation strategy:

1) when the robot is behind the slow vehicle, the target is set as a point aligned with the back of other vehicle, but shifted at the left or at the right (if admissible) by a given safety distance, and moving at the same speed of the other vehicle;
2) once that the previous target is reached, the target is shifted forward of a given distance and is kept moving with a given velocity, slightly greater than that of the slower vehicle;
3) once the the previous target is reached, the target is set as a point in front of the other vehicle and is kept moving with a given velocity, slightly greater than that of the slower vehicle.

At the end of this manoeuver, the robot has completed the pass and has safely avoided the moving obstacle with a smooth trajectory. An example of a trajectory achieving this behavior is shown in Fig. 6, in which the three different moving target points, as previously described, are denoted with green triangles.

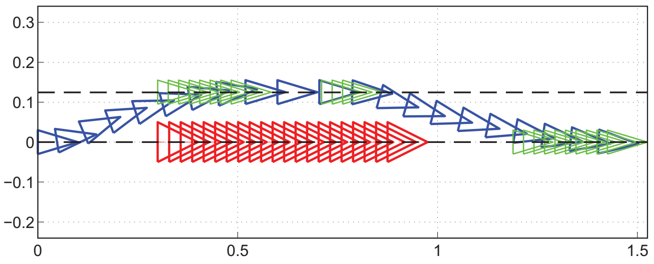


Fig. 6. Smooth trajectory (blue) to pass a slower robot (red) moving in the same direction

IV. Extension to three-dimensional space

Another domain that could be of interest for the application of the proposed online trajectory generation method is the one of Unmanned Aerial Vehicles (UAVs). For a given class of UAVs (e.g. fixed wing aircrafts), the target tracking logic of the nonlinear smoothing filter can be extended to the three-dimensional space by using the pursuit-evasion geometry shown in Fig. 7.

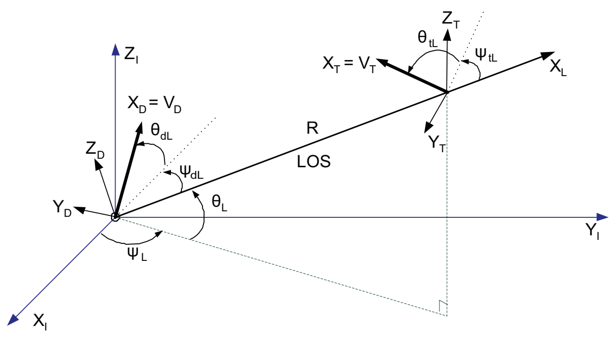


Fig. 7. Pursuit-evasion in three-dimensional space

In particular, it is assumed that the target point may have a velocity directed as the x-axis of an (X_T, Y_T, Z_T) coordinate system, while the output of the trajectory generator has an equivalent linear velocity directed as the x-axis of the (X_D, Y_D, Z_D) coordinate system. These coordinate systems are specified by the coordinates $[x_t, y_t, z_t]$ and $[x_d, y_d, z_d]$ of their origin w.r.t. the reference system (X_I, Y_I, Z_I) and by their azimuth angle ψ and elevation angle θ. In the figure, (ψ_L, θ_L) denotes the orientation of the *line of sight* (LOS) vector w.r.t. the reference system, while (ψ_{tL}, θ_{tL}) and (ψ_{dL}, θ_{dL}) denote the orientation respectively of the target velocity and the velocity of the output trajectory w.r.t. to (X_L, Y_L, Z_L), the LOS coordinate system.

A smooth three-dimensional trajectory reaching the target point, in the sense that the length of the LOS vector R tends to zero, can be computed by a nonlinear filter with a structure similar to the one shown in Fig. 3, but including three different one-dimensional nonlinear filters: one for the linear velocity v_d and two for the azimuth and elevation angles (ψ_d, θ_d), w.r.t. the reference system. The final output of the trajectory generator can be computed according to the following kinematic model:

$$\begin{aligned} \dot{x}_d &= v_d \cos\psi_d \cos\theta_d \\ \dot{y}_d &= v_d \sin\psi_d \cos\theta_d \\ \dot{z}_d &= v_d \sin\theta_d \end{aligned} \tag{8}$$

The switching logic of the three-dimensional trajectory generator is also similar to the one described in Section II, but it sets the reference signals for the two smoothing filters related to orientation angles using the pursuit-evasion equations described, among others, in [9], here omitted for brevity.

Remark 3: The kinematic model of Eq. 8 is compatible with the motion of a fixed wing UAV, assuming that the dynamics of the autopilot can be neglected. This model can also be used for trajectory tracking control design based on dynamic feedback linearization, as shown in [10].

Remark 4: With the nonlinear filter based on the kinematic model of Eq. 8 it is possibile to obtain trajectories with bounded curvature, but it is more difficult to apply a via-point commutation strategy based on the theory of clothoids, as described in Section III. For some applications (e.g. motion planning of robotic manipulators instead of UAVs), it would be preferable to preserve strict alignment with the straight-line path connecting via-points. In that case, the commutation strategy of Section III can be extended to the three-dimensional space by using the two-dimensional nonlinear filter, constrained on the plane containing three subsequent target points, and mapping the output of filter from 2D to 3D by using properly calculated transformation matrices. An example of a three-dimensional trajectory generated by the proposed nonlinear filter from a starting point to a final one with a single via-point is shown in Fig. 8.

V. Implementation and experiments

The proposed nonlinear filter has been tested on a in-house built differential-drive platform, with custom electronics. The trajectory generation algorithm described in Section II have been implemented on a motion control card based on a dsPIC30F by Microchip Technology, which is a 16-bit fixed-point Digital Signal Controller (DSC) executing up to 30 MIPS. The DSC implements also the dynamic feedback linearization controller described in [3].

The performance of the DSC have been optimized using only fixed-point computations and proper scaling of the variables in the code, to avoid as much as possibile the use of division operations. The full implementation of the trajectory generator of Fig.3 is computed by the DSC in less than 700 μs, which allowed to safely set the sampling frequency of the full control system at 256 Hz (i.e. 2^8 Hz or $T = 3.90625$ ms). Finally, a global planner can interact with the

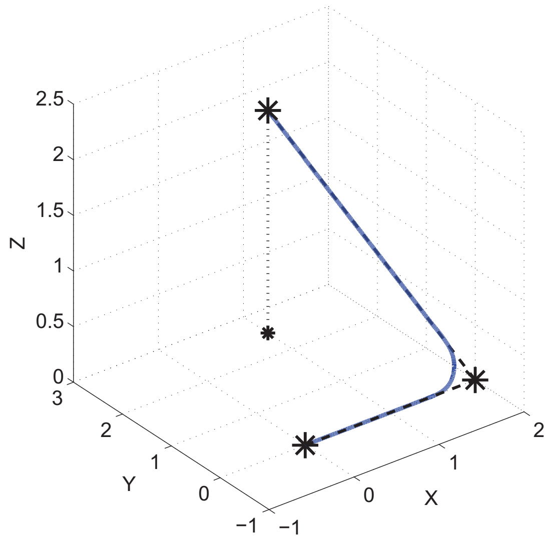


Fig. 8. Example of 3D smooth trajectory generated by the proposed nonlinear filter.

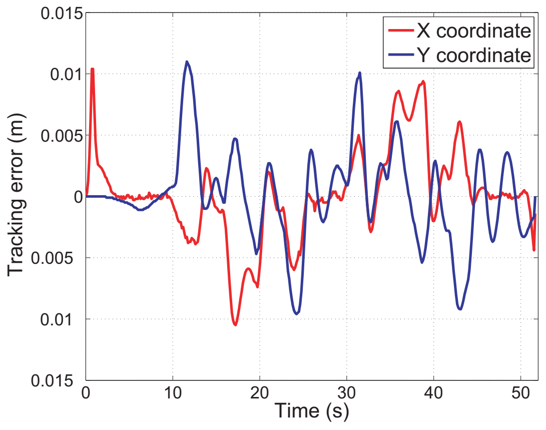


Fig. 9. Tracking error in the cartesian space

trajectory generator by setting the sequence of cartesian via-points, together with related via-point commutation distance, through serial communication.

In its current form, the implementation of the trajectory tracking control loop exploits the feedback obtained from wheel odometry estimates and their numerical differentiation. Even if this estimate may be enhanced using inertial sensors, the control performance can be fairly evaluated on the basis of the norm of the tracking error w.r.t. odometry. This norm, measured during experiments on the reference trajectory of Fig.5, is always lower than 11.5 mm, with an average value of 3.9 mm. Fig. 9 shows the error measurements in the two distinct coordinates. These good tracking performances are better than those described by [11] and [12], which used robots with comparable dynamic performances and similar control methods. It is also important to remark that in these experiments only a 16-bit DSC-based control board has been used, while [11] and [12] used also powerful PCs equipped with Intel Pentium II or Core 2 Duo processors and floating-point capabilities, even though running control loops at slower frequencies (respectively 20 Hz and 35 Hz) because of overall system load.

In particular, Fig. 9 shows that even during initial transient, in which the robot accelerates from zero to maximum linear velocity, tracking is accurate thanks to the feedforward action, the compensation of robot dynamics by means of driving force τ_d and steering torque τ_s control inputs and the smoothness of generated trajectory.

Further information and full source code of the firmware for the motion control board used in the experiments can be downloaded from `http://sact-unife.googlecode.com`.

VI. Conclusion

The paper has described an approach to trajectory generation for mobile robots based on the theory of nonlinear smoothing filters. The trajectories obtained from the filter have continuous curvature and are inherently compatible with kinematic and dynamic constraints of a classical unicycle-like robot. Accurate trajectory tracking can be achieved with a control system based on I/O linearization with dynamic state feedback. A global path planner can interact with the trajectory generator by simply setting sequences of fixed via-points or reference points moving along non-smooth and obstacle-free paths.

References

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modeling, planning and control*, ser. Advanced Textbooks in Control and Signal Processing. Springer-Verlag, 2009.

[2] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] M. Bonfè and C. Secchi, "Online smooth trajectory planning for mobile robots by means of nonlinear filters," in *Proc. of IEEE/RSJ Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010.

[4] R. Zanasi, C. Guarino Lo Bianco, and A. Tonielli, "Nonlinear filters for the generation of smooth trajectories," *Automatica*, vol. 36, p. 439/448, 2000.

[5] V. Utkin, "Variable structure systems with sliding modes," *IEEE Transactions on Automatic Control*, vol. 41, no. 4, 1977.

[6] C. Lin, *Modern Navigation, Guidance and Control Processing*. Prentice-Hall, 1992.

[7] T. Fraichard and A. Scheuer, "From Reed and Shepp's to Continuous-Curvature paths," *IEEE Trans. on Robotics*, vol. 20, no. 6, pp. 1025–1035, December 2004.

[8] E. Szadeczky-Kardoss and B. Kiss, "Path planning and tracking control for an automatic parking assist system," in *European Robotics Symposium 2008*, ser. Springer Tracts in Advanced Robotics, H. Bruyninckx, L. Preucil, and M. Kulich, Eds. Springer Berlin / Heidelberg, 2008, vol. 44, pp. 175–184.

[9] J. Oh and I. Ha, "Capturability of the 3-Dimensional Pure PNG law," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 2, pp. 491–503, April 1999.

[10] N. Bertozzi, P. Castaldi, M. Bonfè, S. Simani, and G. Bertoni, "Guidance and nonlinear active fault tolerant control for general aviation aircraft," in *Proc. of IFAC Workshop on Aerospace Guidance, Navigation and Flight Control Systems*, Samara, Russia, June 30 – July 2 2009.

[11] G. Oriolo, A. De Luca, and M. Vendittelli, "WMR control via dynamic feedback linearization: Design, implementation and experimental validation," *IEEE Trans. on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, November 2002.

[12] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. of IEEE/RJS Conference on Intelligent Robots and Systems*, St. Louis (MO), October 2009.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Multi-axis High-order Trajectory Planning

Ben Ezair[1] Tamir Tassa[1] Zvi Shiller[2]

***Abstract*— This paper presents a trajectory planning algorithm for multi-axis systems. It generates smooth trajectories of any order subject to general initial and final conditions, and constant state and control constraints. The algorithm is recursive, as it constructs a high order trajectory using lower order trajectories. Multi-axis trajectories are computed by synchronizing independent single-axis trajectories to reach their respective targets at the same time.**

The algorithm's efficiency and ability to handle general initial and final conditions make it suitable for reactive real time applications. Its ability to generate high order trajectories makes it suitable for applications requiring high trajectory smoothness. The algorithm is demonstrated in several examples for single- and two-axis trajectories of order $2-6$.

I. Introduction

The trajectory planning problem, in the context of robot motion, is the problem of generating a trajectory in the robot's state space that connects given initial and final states, subject to state and control constraints, and is optimal with respect to some given cost function. A trajectory is essentially a time-parameterized path between two points in the configuration space. While path planning has traditionally been concerned with generating the shortest path that avoids obstacles, trajectory planning is concerned in addition with the robot's dynamic behavior by imposing constraints on the robot's velocity, acceleration, jerk and possibly higher derivatives. Bounding the motion derivatives yields smooth trajectories, which can be tracked with smooth control inputs that do not excite high vibration modes. In addition, they increase tracking accuracy [11]. The number of bounded derivatives in the trajectory is called the order of the trajectory.

Several approaches for trajectory generation have been developed. One approach uses polynomials or other functions to approximate the desired trajectories. Piazzi and Visioli [13] optimize cubic splines to minimize jerk for a specified motion time. Petrinec and Kovacic [12] use 4th and 5th order polynomials to produce smooth multi-axis trajectories. Macfarlane and Croft [10] compute trajectories that are represented by fifth-order polynomials.

Another approach for trajectory generation is to divide the trajectory into segments where the value of the highest derivative is constant in each segment. Liu et al. [9] present an algorithm that produces a third order trajectory that is constructed by dividing the trajectory to seven segments. This approach is used to produce multi-axis trajectories by synchronizing several single-axis trajectories [1], [2], [5]. Haschke et al. [4] emphasize the online capabilities of their algorithm that is designed to produce a third order halting trajectory. Works by Kroger et al. [6], [7] also focus on online algorithms, using a thorough analysis of possible acceleration profiles to handle more general initial and final conditions. Lambrechts et al. [8] produce fourth order trajectories. Nguyen et. al. [11] developed an algorithm that generates trajectories of arbitrary order with zero initial and final conditions and symmetric state and control constraints. It is based on dividing the trajectory into a recursive structure of S-curve segments. The use of S-curves forms also the basis for the algorithm which we present herein.

A. Our algorithm

This paper presents a novel algorithm for planning trajectories of arbitrary order between arbitrary initial and final states (position and its time derivatives), subject to arbitrary constant state and control constraints, which is geared towards minimizing motion time. The generality of our approach makes the algorithm suitable for both online and offline trajectory planning. The algorithm is recursive, as it reduces the original problem of order m to problems of order $m-1$, until it reaches basic problems that can be solved directly. The algorithm is also modular, as it may accept any external solver for the basic trajectory generation problem which is solved directly in order to terminate the recursion. The algorithm is efficient, as demonstrated in several experiments (see Table II in Section II-B.2).

Finally, the basic algorithm is extended to generate multi-axis trajectories by synchronizing single-axis trajectories to reach their respective targets at the same time.

Table I compares our algorithm with a few comparable algorithms that were presented in the above described studies. Most of the comparable algorithms are limited in the order of the trajectories that they may produce, or in the initial and final conditions that they may accept. Our algorithm's main advantage is its generality and flexibility, as it is applicable to a wider range of scenarios than the other algorithms.

II. Single-axis trajectories

We wish to compute a pair $\langle T, x(t)\rangle$, where $x(t)$ denotes the position of a moving object along a given axis, such that (a) $x(t)$ satisfies given initial and final conditions at $t=0$ and $t=T$,

$$x(0) = x_s^0\,,\ x^{(i)}(0) = x_s^i\,,\quad 1 \le i \le m-1\,, \tag{1}$$

$$x(T) = x_f^0\,,\ x^{(i)}(T) = x_f^i\,,\quad 1 \le i \le m-1\,, \tag{2}$$

[1]Ben Ezair and Tamir Tassa are with The Department of Mathematics and Computer Science, The Open University, Israel. ben_e@hotmail.com & tamirta@openu.ac.il

[2]Zvi Shiller is with the Department of Mechanical Engineering and Mechatronics, Ariel University Center of Samaria. shiller@ariel.ac.il

Ref.	Order	Initial & Final Conditions	Online	Optimal
[6]	2	general	yes	yes
[2]	3	zero acceleration	yes	yes
[4]	3	ends at rest	yes	yes
[7]	3	zero final acceleration	yes	yes
[8]	4	rest to rest	no	no
[11]	any	rest to rest	no	no
Ours	any	general	yes	no

TABLE I
COMPARISON OF TRAJECTORY GENERATION ALGORITHMS

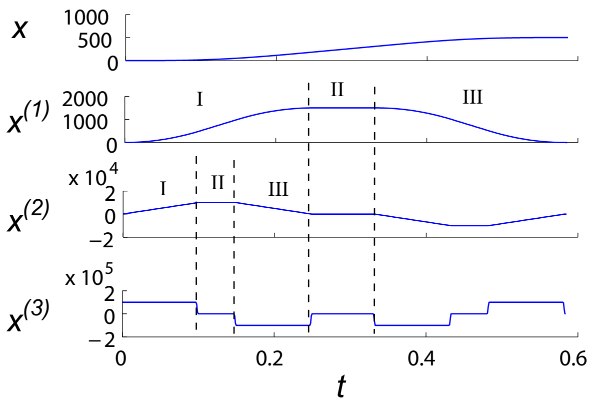


Fig. 1. The recursive structure of the trajectory

where $x^{(i)}(t)$, $i \geq 1$, is the i-th order derivative of $x(t)$; (b) it is constrained by constant lower and upper bounds,

$$x^i_{min} < 0 < x^i_{max}\,, \quad 1 \leq i \leq m \tag{3}$$

$$x^i_{min} \leq x^{(i)}(t) \leq x^i_{max}\,, \quad t \in [0, T]\,, \ 1 \leq i \leq m\,; \tag{4}$$

and (c) the time $T = \int_0^T 1dt$ is minimized. The number $m \geq 1$ of constrained derivatives is called the order of the problem. A pair $\langle T, x(t)\rangle$ that satisfies the initial and final conditions, (1)–(2), and the lower and upper bounds (4) is called a feasible solution. A solution is optimal if is feasible and minimizes T.

This single-axis trajectory planning problem may be viewed as a time optimal control problem of a linear system of ordinary differential equations with m state variables (being the position function $x(t)$ and its first $m-1$ derivatives) and a single control variable (being the m-th derivative $x^{(m)}(t)$), subject to initial and final conditions and state and control constraints. The structure of the optimal control for such problems can be shown to have a bang-zero-bang structure [3].

The solution for the case $m = 1$ is trivial, consisting of a constant velocity motion. The solution for $m = 2$ was derived in [6]. Our approach in solving higher order problems is recursive, as it reduces a problem of order m to problems of order $m - 1$, repeatedly, until $m = 2$, in which case the problem can be solved directly.

A. A single-axis trajectory planning algorithm

1) Overview: The algorithm for computing high order trajectories is motivated by the observation that integrating a bang-zero-bang control profile yields an S-curve structure. A typical S-curve can be divided into three segments: (I) acceleration from the initial state; (II) cruising at a constant velocity; and (III) deceleration to the final state. This structure, as illustrated in Figure 1 for $m = 3$, repeats recursively since the velocity profile, as well as the profiles of higher derivatives, consist of two or more S-curve segments.

The recursive algorithm looks for a solution with an S-curve position profile. The main loop attempts to find the best value for the constant velocity in segment II. Given a candidate value v for that velocity, the algorithm computes the velocity profile in segments I and III by invoking recursion. Specifically, it solves in each of those segments a reduced order trajectory planning problem for the velocity profiles. Once the velocity profiles in all three segments are found, the algorithm checks that the corresponding position profile is a feasible solution. When the resulting solution is non-feasible, the algorithm reduces $|v|$; when the resulting solution is feasible, the algorithm increases $|v|$ in order to reduce motion time. The algorithm terminates when the optimal value of v is found within some predetermined accuracy, and it outputs the found position profile $x(t)$.

2) Detailed description: We proceed to describe the operation of Algorithm 1 that implements the above procedure. The algorithm accepts as inputs the problem order, the initial and final conditions, and the lower and upper constraints. It outputs a feasible solution $\langle T, x(t)\rangle$ which is time-efficient and in some cases optimal.

If $m = 2$ the algorithm outputs the analytic solution (Step 1). Otherwise, we set Δx to be the distance to be traveled (Step 2) and start a binary search for v within the allowed range of values $[x^1_{min}, x^1_{max}]$. The variables v_{min} and v_{max} hold the lower and upper limits of the search range; they are initialized in Step 3. The variable $\hat{v}$ holds the last value of v that produced a feasible solution. It is initialized to an illegal value $(x^1_{max} + 1)$ in Step 3, and so is v.

During the binary search (Steps 4-18), we consider the midpoint of the current range as the candidate value for v (Step 6). Given a candidate value for v, the trajectory planning problem in the acceleration and deceleration segments (I and III) are well defined and can be solved by invoking recursion. Let $v_1(t)$ be the velocity profile in segment I, from the initial value x^1_s to the cruising velocity v, and let $\tau_{1,v}$ denote the duration of that segment. Then in Step 7 we compute $\langle \tau_{1,v}, v_1(t)\rangle$ by solving a problem of order $m - 1$ for $x'(t)$ along that segment. The initial conditions for that reduced order problem are $(x^1_s, \ldots, x^{m-1}_s)$; its final conditions are $(v, 0, \ldots, 0)$ (since we wish to reach the velocity v with all higher derivatives zero); and the lower and upper bounds on the derivatives are in accord with those of the original problem. Similarly, we invoke recursion in Step 8 to compute $v_3(t)$, the velocity profile in segment III, from v to the final value x^1_f, and the corresponding duration $\tau_{3,v}$.

Next, we compute the distance covered in segments I and III, Δx_1 and Δx_3 (Step 9). Δ is the remaining distance that needs to be traveled in the intermediate segment II in order to complete a journey of length Δx. Since the velocity along segment II is constant and equals v, the duration of that segment should be $\tau_{2,v} = \Delta/v$ (Step 10). If $\tau_{2,v}$ is nonnegative, then this tested value of v leads to a valid trajectory; in that case we record that value of v in the variable $\hat{v}$ (Step 11).

The search ends once the lower and upper limits of the search are sufficiently close (Steps 12-14). In that case, we set v, v_{min} and v_{max} to be the last value of v that produced a valid solution. If $\hat{v}$ still equals its initial value $x^1_{max} + 1$ (a forbidden value for v, as it is outside the allowed range $[x^1_{min}, x^1_{max}]$), then the search failed to find a valid v. This may occur if the problem parameters define a range of legitimate v values that is smaller than ε, and, consequently, cannot be captured using a binary search with such accuracy. (We note that instead of using the same value of ε for all levels, we may define for each level i, $1 \le i \le m$, a different value ε_i.) Otherwise, if $\hat{v}$ is a legal value, then the algorithm performs another iteration. Since v, v_{min}, and v_{max} equal the last valid value of v, the subsequent setting of $last\_v$ and v in Steps 5-6 will cause the algorithm to perform the next iteration with $v = \hat{v}$ and then terminate the loop when it examines the termination condition in Step 18.

In case the lower and upper limits of the search are still far apart, we check the value of Δ to determine how to proceed with the search: if $\Delta > 0$, then we examine profiles with higher values of v (Step 15); if $\Delta < 0$, we consider lower values of v (Step 16); if $\Delta = 0$, we terminate the search by setting $last\_v$ to equal v (Steps 17). The search ends when $last\_v = v$. After determining the value v, we compute T and construct the profile of x' as the concatenation of three segments – $v_1(t)$, v,$v_3(t)$ (Steps 19-20). Finally, we integrate $x'(t)$ to obtain $x(t)$ (Step 21).

3) A note on optimality: Algorithm 1 uses a simple greedy approach in the search of a solution with a minimal motion time. The solution is optimal for rest-to-rest motions of order $m \le 3$. (The proof of this claim is deferred to the full version of this paper.) Although Algorithm 1 attempts to minimize motion time, the solution is not necessarily optimal, because the algorithm is based on two assumptions that are not always satisfied:

(A1) The duration of segment II is a continuous and monotonic function of v.

(A2) The velocity during segment II is constant, implying that during this phase all higher derivatives are zero.

The first assumption affects the way the algorithm updates v (steps 15-16). If this assumption is not satisfied, the algorithm may choose a value of v that will result in a non-optimal motion time. The second assumption is more central to Algorithm 1, as it allows us to subdivide the trajectory into two S curves that can be joined together with a simple constant velocity motion. However, this assumption is not always true, e.g. in cases where the optimal trajectory either always accelerates or always decelerates. In such cases, the algorithm would return a solution that is of a different structure than that of the optimal trajectory.

Algorithm 1 ComputeTrajectory

```
Input:
(1) The system order $m \ge 2$.
(2) Initial and final states: $x^i_s, x^i_f$, $0 \le i \le m-1$.
(3) Bounds: $x^i_{min}, x^i_{max}$, $1 \le i \le m-1$.
Output: A feasible solution $\langle T, x(t) \rangle$.
 1: if $m = 2$ then return the analytic solution and stop.
 2: $\Delta x = x^0_f - x^0_s$.
 3: $v_{min} = x^1_{min}$, $v_{max} = x^1_{max}$, $v = \hat{v} = x^1_{max} + 1$.
 4: repeat
 5:   $last\_v = v$.
 6:   $v = (v_{max} + v_{min})/2$.
 7:   $\langle \tau_{1,v}, v_1(t) \rangle \leftarrow$ ComputeTrajectory$[m-1,$
      $(x^1_s, \ldots, x^{m-1}_s), (v, 0, \ldots, 0), \{(x^2_{min}, x^2_{max})\}^m_{i=2}]$
 8:   $\langle \tau_{3,v}, v_3(t) \rangle \leftarrow$ ComputeTrajectory$[m-1,$
      $(v, 0, \ldots, 0), (x^1_f, \ldots, x^{m-1}_f), \{(x^2_{min}, x^2_{max})\}^m_{i=2}]$
 9:   $\Delta x_1 = \int_0^{\tau_{1,v}} v_1(t)dt$;  $\Delta x_3 = \int_0^{\tau_{3,v}} v_3(t)dt$.
10:   $\Delta = \Delta x - \Delta x_1 - \Delta x_3$;  $\tau_{2,v} = \Delta/v$.
11:   if $\tau_{2,v} \ge 0$, $\hat{v} = v$.
12:   if $|v_{max} - v_{min}| \le \varepsilon$ then
13:     $v = v_{min} = v_{max} = \hat{v}$.
14:     if ($\hat{v} = x^1_{max} + 1$) then stop and output "Failed".
15:   elseif $\Delta > 0$ then $v_{min} = v$
16:   elseif $\Delta < 0$ then $v_{max} = v$
17:   else $last\_v = v$ endif
18: until $last\_v = v$
19: $T = \tau_{1,v} + \tau_{2,v} + \tau_{3,v}$.
20: $x'(t) = \begin{cases} v_1(t) & [0, \tau_{1,v}] \\ v & [\tau_{1,v}, \tau_{1,v} + \tau_{2,v}] \\ v_3(t - \tau_{1,v} - \tau_{2,v}) & [\tau_{1,v} + \tau_{2,v}, T] \end{cases}$
21: Return $\langle T, x(t) \rangle$, where $x(t) = \int_0^t x'(\tau)d\tau + x^0_s$.
```

Both assumptions make the algorithm efficient by limiting the number of possible trajectory forms we need to consider. This, in turn, greatly simplifies the search for segment II that connects segments I and III. It is possible to remove these assumptions, while keeping the recursive structure of the algorithm, and produce the optimal trajectory by exhaustively searching for the initial and final conditions of segment II, at the obvious cost of increasing the computational complexity.

B. Experiments

1) Examples of trajectories: We tested Algorithm 1 for high order trajectories (with orders up to $m = 7$) with zero and non-zero initial and final conditions. Figure 2 shows trajectories computed by the algorithm for various values of m, $\Delta x = 50$, zero initial and final conditions ($x^i_s = x^i_f = 0$, $1 \le i \le m-1$); the state constraints in this example were $|x^{(i)}| \le 10^{2+i}$. These results show that motion time and smoothness increase with the trajectory order, because of the added limits on higher derivatives. The $m = 2$ profile in Figure 2 is the fastest, but it is not smooth as already its acceleration profile is discontinuous. The $m = 6$ profile, on the other hand, is the slowest, but it exhibits discontinuities only in its sixth derivative.

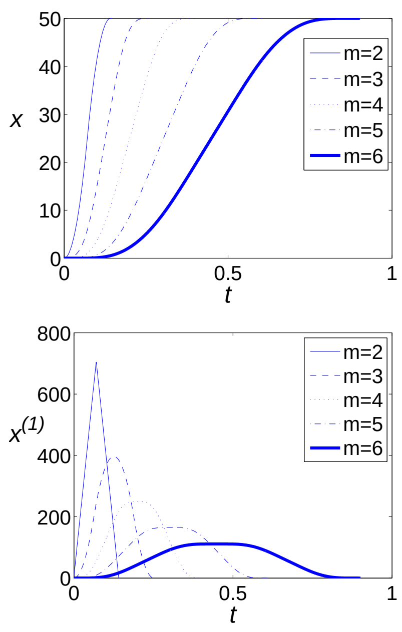


Fig. 2. Trajectory position (top) and velocity (bottom) for $m = 2, 3, 4, 5, 6$

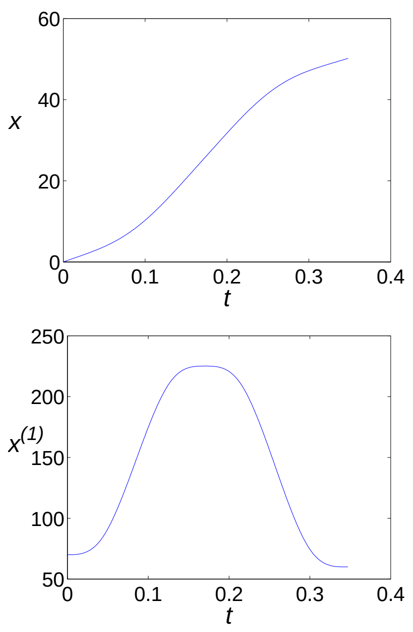


Fig. 3. 4th order trajectory position (top) and velocity (bottom) with nonzero initial and final conditions

Figure 3 shows a solution for the same setting as in Figure 2, for $m = 4$, except that the initial and final conditions on the velocity are nonzero: $x_s^1 = 70$ and $x_f^1 = 60$.

All of the these solutions share the familiar bang-zero-bang pattern.

2) Runtimes: The algorithm was implemented in C++ and was executed as a normal priority process on an Intel Pentium D 3.0 GHz processor, using a normal Microsoft windows XP system.

Table II shows the average runtimes of Algorithm 1 for various values of m, when executed with the same inputs as used to generate the trajectories in Figure 2. The parameter ε_i was set so that the accuracy is 0.01%, i.e., $\frac{x^i_{max} - x^i_{min}}{\varepsilon_i} = 0.0001$ for all i. For each m, the average runtime was computed by averaging several runs of the algorithm. As can be seen in Table II, the runtime changes exponentially with respect to m, since the algorithm is recursive in m. As m is always a small integer, that exponential dependence on m poses no practical problem.

We note that Algorithm 1 may be parallelized, as Steps 7 and 8 are independent of each other and could be executed in parallel. It is therefore possible to reduce the runtime by a factor of up to 2^{m-2} on a multi-core CPU, depending on the number of processes that can be executed in parallel.

Order	Number of runs	Average runtime [s]
3	1000	0.000074
4	1000	0.002141
5	10	0.0625
6	10	1.9515
7	10	80.064

TABLE II
RUNTIMES (SECONDS) FOR SEVERAL PROFILE ORDERS

III. MULTI-AXIS TRAJECTORIES

The single-axis trajectory planning algorithm can be used for solving multi-axis trajectory planning problems. We wish to compute a pair $\langle T, (x_1(t), \ldots, x_n(t))\rangle$, where $(x_1(t), \ldots, x_n(t))$ is a function that connects two points in the Euclidean space $\mathbb{R}^n$ in minimal time, subject to the following constraints: (a) $x_j(t)$ satisfies given initial and final conditions at $t = 0$ and $t = T$,

$$x_j(0) = x_s^{j,0}\,, \quad x_j^{(i)}(0) = x_s^{j,i}\,, \tag{5}$$

$$x_j(T) = x_f^{j,0}\,, \quad x_j^{(i)}(T) = x_f^{j,i}\,, \tag{6}$$

where $1 \le i \le m-1$ and $1 \le j \le n$ (hereinafter the index j denotes the axis while i denotes the derivative order); (b) it is constrained by constant lower and upper bounds,

$$x_{min}^{j,i} \le x_j^{(i)}(t) \le x_{max}^{j,i}\,, \qquad t \in [0, T]\,, \tag{7}$$

where $1 \le i \le m$ and $1 \le j \le n$; and (c) the time $T = \int_0^T 1dt$ is minimized.

To solve the multi-axis trajectory planning problem, we begin by first solving the n independent single-axis problems. For each axis $1 \le j \le n$, we get a single-axis trajectory, $x_j(t)$, that satisfies the initial and final conditions and kinematic bounds along that axis, and completes the journey in minimal time. The goal is now to combine those n single-axis trajectories, each reaching its final position at a possibly different time, into one multi-axis trajectory, $(x_1(t), \ldots, x_n(t))$. This is done by identifying the slowest axis, and then "stretching" the trajectories along the other axes so that they all reach their respective target at the same

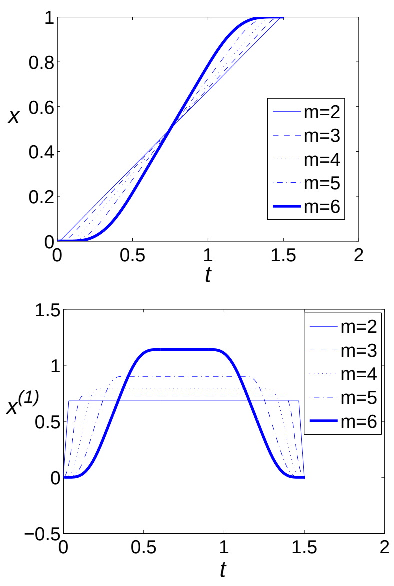


Fig. 4. Trajectories — position (top) and velocity (bottom) for $m = 2, \ldots, 6$, with $T_{\text{ext}} = 1.5$

time. The stretching procedure may be repeated until all single-axis trajectories reach their target at the same time.

In order to stretch a trajectory that was generated by Algorithm 1, we slightly modify the function ComputeTrajectory that the algorithm implements into a new function, called ComputeTrajectory-TimeLimit. That function receives the same inputs as ComputeTrajectory, and one additional positive scalar parameter denoted T_{ext}. It then proceeds to generate a trajectory that complies with the given inputs and takes minimal time that is no less than T_{ext}. To that end, if the modified algorithm generates a trajectory that reaches its goal in less than T_{ext}, it slows down the motion by decreasing the absolute value of v, the constant velocity during segment II. Specifically, if the value of v for the faster-than-T_{ext} solution is positive, the algorithm lowers the upper bound of the binary search so that it examines smaller values for v; if, on the other hand, the value of v for the faster-than-T_{ext} solution is negative, the algorithm sets it as the lower bound of the binary search to explore higher values for v. To achieve the above described functionality, the only modification that needs to be introduced is adding the next command after Step 11: **if** $(\tau_{2,v} > 0)$ and $(\tau_{1,v}+\tau_{2,v}+\tau_{3,v} < T_{\text{ext}})$ **then** $\Delta = -\Delta$.

To illustrate the effect of calling the modified function ComputeTrajectory-TimeLimit with a positive T_{ext}, we ran the algorithm with various values of m, $\Delta x = 1$, zero initial and final conditions ($x_s^i = x_f^i = 0$, $1 \le i \le m-1$), state constraints $|x^{(i)}| \le 2 \cdot 10^{i-1}$, and set $T_{\text{ext}} = 1.5$. The resulting trajectories, for $m = 1, \ldots, 6$, all with travel time of $T = 1.5$, are shown in Figure 4. Note that all trajectories use a cruising velocity well below the upper velocity constraint in order to comply with the given lower bound $T_{\text{ext}} = 1.5$ on the motion time.

Algorithm 2 solves the multi-axis problem, for any number of axes, iteratively by searching for the shortest common motion time. It saves in T_{max} the duration of the currently slowest trajectory, and in $sync$ the number of axes along which it already found a feasible solution with motion time T_{max} (or at least a motion time $T \in [T_{max}, T_{max} + \theta]$, where θ is a small parameter that determines the desired level of accuracy). To that end, after initializing those two variables (Step 1), it starts a cyclic loop over all axes (Steps 2-9) in search of the smallest value of T_{max} for which there is a feasible solution along each of the n axes with motion time $T \in [T_{max}, T_{max} + \theta]$. In order to synchronize the single-axis trajectories, Algorithm 2 computes a trajectory along each axis by invoking the modified Algorithm 1 (namely, the function ComputeTrajectory-TimeLimit) with T_{ext} that equals the current slowest motion time (Step 4). If ComputeTrajectory-TimeLimit succeeds in finding a feasible solution with $T \in [T_{max}, T_{max} + \theta]$, it records that success by increasing $sync$ (Step 5). Otherwise, the found feasible solution ends in time $T > T_{max} + \theta$; in that case, T_{max} is reset to T, and $sync$ is reset to 1 (Step 6). The loop ends only when $sync = n$ (Step 9), since then all single-axis trajectories have the same duration (up to a tolerable difference of θ). The algorithm then stops and returns the found feasible multi-axis solution (Step 10).

Algorithm 2 SynchronizeTrajectories

Input:

(1) The system order $m \ge 1$.
(2) The number $n \ge 1$ of trajectories that need to be synchronized.
(3) An accuracy parameter for the motion time, $\theta \ge 0$.
(4) Initial values: $x_s^{j,i}$, $0 \le i \le m-1$, $1 \le j \le n$.
(5) Final values: $x_f^{j,i}$, $0 \le i \le m-1$, $1 \le j \le n$.
(6) Bounds: $x_{min}^{j,i} \le 0 \le x_{max}^{j,i}$, $1 \le i \le m$, $1 \le j \le n$.

Output:

(1) Total motion time, $T > 0$.
(2) Trajectories $x_j(t)$, $1 \le j \le n$, that satisfy the input constraints, each spanning the time T.

```
1:  $T_{max} = 0$; $sync = 0$.
2:  $j = 1$.
3:  **repeat**
4:      $\langle T, x_j(t)\rangle \leftarrow$ ComputeTrajectory-TimeLimit$[m,$
        $(x_s^{j,0}, \ldots, x_s^{j,m-1}), (x_f^{j,0}, \ldots, x_f^{j,m-1}),$
        $\{(x_{min}^{j,i}, \ldots, x_{max}^{j,i})\}_{i=1}^m, T_{\text{ext}} = T_{max}]$
5:      **if** $T - T_{max} \le \theta$ **then** $sync = sync + 1$
6:      **else** $T_{max} = T$, $sync = 1$
7:      $j = j + 1$.
8:      **if** $j = n + 1$ **then** $j = 1$
9:  **until** $sync = n$
10: Return $\langle T_{max}, (x_1(t), \ldots, x_n(t))\rangle$.
```

Example. This example demonstrates the use of Algorithm 2 to generate a trajectory that passes through four points in the plane with specified velocities and accelerations. The resulting trajectory demonstrates the algorithm's ability to produce a high-order continuous path.

Let $A = (0,0)$, $B = (20,0)$, $C = (20,20)$, and $D = (0,20)$ be four points in the $x-y$ plane. We wish to move a

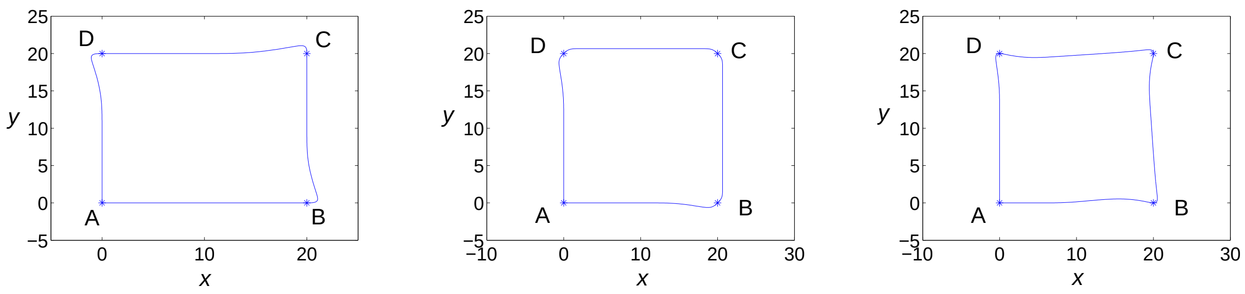


Fig. 5. Trajectories along a square path as described in Example 2: Scenario 2 (left), 3, and 4 (right).

body through these points, $A \to B \to C \to D \to A$, starting and finishing at rest. We consider trajectories of order $m = 3$ with the following bounds along each of the four motion segments: $|x^{(i)}| \leq 10^{2+i}$, $1 \leq i \leq m$.

We examine four scenarios that differ in the inner corner velocities and accelerations, at B, C and D. In Scenario 1, the body reaches a full stop in each inner corner before continuing its motion. In Scenario 2, the velocity in each inner corner is 50 in the direction leading to the corner, and the acceleration there is zero. In Scenario 3, the corner velocities are counterclockwise 45^o rotations of the corresponding corner velocities in Scenario 2 (so that the velocity at B, for example, is $(50/\sqrt{2}, 50/\sqrt{2})$ instead of $(50, 0)$ as it was in Scenario 2); the acceleration in each corner is set to zero. This adjustment of the velocity to the right-angle turn in each corner results in a shorter overall motion time with respect to Scenario 2. Finally, Scenario 4 is identical to Scenario 2 except for the acceleration values in the inner corners. These acceleration values are designed so that the moving body begins accelerating for the next motion segment earlier, in order to reduce the overall motion time. The acceleration values are $(-2000, 2000)$ at B, $(-2000, -2000)$ at C, and $(2000, -2000)$ at D. The trajectories in Scenarios 2, 3 and 4 are shown in Figure 5. (The trajectory in Scenario 1 is not shown since it is a perfect square.)

As expected, the motion time in Scenario 1 is the longest, $T_1 = 0.743$. In Scenario 2, where the body is not forced to stop in each inner point, it is $T_2 = 0.701$. In Scenario 3, in which the corner velocities are better adjusted to the counterclockwise turns in each corner, the motion time reduces to $T_2 = 0.683$. Finally, in Scenario 4, with the added benefit of acceleration conditions, the body completes the journey in time $T_4 = 0.620$.

IV. Conclusion

This paper presented a trajectory planning algorithm for single and multi-axis a trajectories, subject to general initial and final conditions and derivative bounds. It is based on a recursive process that reduces the original high order trajectory problem to lower order problems. The recursion is applied until reaching low orders ($m = 1$ or $m = 2$) for which a direct solution is available. The resulting algorithm is simple and efficient, as was demonstrated in our runtime results. The proposed algorithm can be used off-line to produce high order trajectories, as well as on-line in applications where efficiency and reactiveness are essential.

In this paper we focused on multi-axes trajectories with no concern to geometrical constraints, apart from the initial and final positions. Extending our algorithm to account for geometrical constraints, such as imposed by obstacles or by a specified path, is a subject of future research.

References

[1] I.H. Aguilar and D. Sidobre. On-line trajectory planning of robot manipulators end effector in cartesian space using quaternions.

[2] X. Broquère, D. Sidobre, and I. Herrera-Aguilar. Soft motion trajectory planner for service manipulator robot. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2808–2813. IEEE, 2008.

[3] A.E. Bryson and Y.C. Ho. *Applied Optimal Control*. Blaisdell Publishing Co., Cambridge, MA, 1969.

[4] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3248–3253. IEEE.

[5] I. Herrera-Aguilar and D. Sidobre. Soft motion trajectory planning and control for service manipulator robot.

[6] T. Kroger, A. Tomiczek, and F.M. Wahl. Towards on-line trajectory computation. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 736–741. IEEE, 2006.

[7] T. Kroger and F.M. Wahl. Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *Robotics, IEEE Transactions on*, 26(1):94–111, 2010.

[8] P. Lambrechts, M. Boerlage, and M. Steinbuch. Trajectory planning and feedforward design for high performance motion systems. In *American Control Conference, 2004. Proceedings of the 2004*, volume 5, pages 4637–4642. IEEE.

[9] S. Liu. An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators. In *Advanced Motion Control, 2002. 7th International Workshop on*, pages 365–370. IEEE, 2002.

[10] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: Design for real-time applications. *Robotics and Automation, IEEE Transactions on*, 19(1):42–52, 2003.

[11] K.D. Nguyen, I.M. Chen, and T.C. Ng. Planning algorithms for s-curve trajectories. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1–6. IEEE, 2007.

[12] K. Petrinec and Z. Kovacic. Trajectory planning algorithm based on the continuity of jerk. In *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pages 1–5. IEEE, 6.

[13] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *Industrial Electronics, IEEE Transactions on*, 47(1):140–149, 2000.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Online, Adaptive, and Distributed Multi-Robot Motion Planning for Collaborative Patrolling of Sparse Sensor Networks

Theofanis P. Lambrou and Christos G. Panayiotou

Abstract— **This paper presents an online, adaptive, distributive and collaborative path planning method for a team of autonomous mobile sensors that enables them to navigate through a sparse network of stationary sensors to search for events and improve the spatio-temporal coverage of the sensor field. The mobile sensor nodes have limited communication and sensing ranges and collaborate to *autonomously* plan their trajectories, adapt to the local region they monitor and enhance the area coverage over time under constrains like obstacles, collisions and limited communication. In this context, this paper addresses the trade off between area coverage and mobiles' travelled distance and proposes an adaptive speed model to minimize the distance the mobiles travelled and hence the energy needed for mobility. Finally, simulation results indicate the effectiveness of the proposed approach over a centralized partitioning approach under mobile sensors failures.**

I. Introduction

This article investigates the path planning problem for improving the coverage and detection performance of mixed WSNs consisting of both static and mobile nodes. With recent advances in distributed robotics and low power embedded systems, such mixed WSNs are becoming attractive as covering complectly a large region of interest with static sensors requires excessively dense deployments which implies prohibitive cost. However, controlling the motion of mobile sensor-robots in such distributed environments is often complicated by factors as resource constraints on sensing, motion, communication and computation capabilities, uncertain nature of the environment (e.g. obstacles, hazards, node failures) and distributed-asynchronous information sharing.

Such mixed WSNs are expected to find potential applications in environmental monitoring (e.g. water bodies monitoring) as well as search and surveillance operations. Search and surveillance is a problem that has attracted significant attention over the past years, however, there is significant focus on how to allocate search effort across the environment instead of finding the best search path to follow [1], [2]. Recently, wireless sensor networks (WSNs) have been proposed to address the area monitoring or surveillance problem with either stationary nodes [3], [4], mobile [5], [6], [7], [8], [9] or both types of nodes [10], [11], [12], [13]. Mixed/Mobile WSNs is a new area of research and methods proposed usually considered random mobility models [5], [11], [14] or they do not even consider the actual path that mobile nodes should follow [6],[15] (e.g. solve the redeployment problem). Moreover, other methods proposed for finding the worst-case coverage path [16] do not consider the complete coverage-search problem and provide only a single path between two given points in a centralized and static manner (do not consider changes in the field) and hence do not support multiple mobile nodes.

In [12], an architecture is developed that enables the collaboration of mobile and stationary sensor nodes in WSNs. Mobile sensors plan their trajectories to sample the least covered areas by the stationary sensor nodes. The framework developed is easily scalable to large numbers of mobile sensor nodes and for different WSNs deployments and enables mobile sensors to compute their path on-line using only "local" information and adapt to the sensor field changes. This paper extends and generalizes the framework proposed in [12] by incorporating a probabilistic sensing model and a dynamic speed policy. In addition, the approach is now applicable for mobile nodes with variable speed and sensor fields that include obstacles. The main contribution of this paper is the development of an adaptive speed policy that maximizes the area coverage and at the same time minimizes the total distance travelled by mobiles (energy needed for mobility).

The remaining of the paper is organized as follows. Section II presents the distributed-collaborative path planning framework for mixed WSNs. Section III investigates the performance of the proposed framework and presents the simulation results. Finally, the paper concludes with Section IV.

II. Path Planning Framework

In this section we present a collaborative framework where the mobiles nodes autonomously decide their path to sample the areas least covered. In this architecture, at every step, the mobiles define a "local" area around their current location and identify the biggest coverage hole which becomes their next target point. Target points are then updated in a receding horizon like scheme. This approach works well and given enough time complete area coverage can be achieved.

At this point its worth pointing out that alternative path planning approaches like potential function techniques usually fail to address the problem under consideration as they get stuck in local minima or oscillate between two closest points [17]. Solutions provided to overcome the problem of local minima, when planning with potential functions, like wave-front planner [18], and *navigation functions* [19], [20] still fail to address the problem. Wave-front planner needs to search the entire space for a path each time the path is updated which is computationally intractable. On the

T. Lambrou and C. Panayiotou are with the KIOS Research Center for Intelligent Systems and Networks and the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus. {faniseng,christosp}@ucy.ac.cy

other hand, navigation functions assumed that obstacles are circular disks that do not intersect and the configuration space is bounded by a sphere (or a star) space. These assumptions are not satisfied in our setting.

Therefore, efficient path planning under random static sensor deployments is complicated, especially when the algorithm is intended to be robust to the sensor field density. To resolve these issues we have developed dynamic receding horizon policy which constitutes of appropriate normalizing path cost functions. It should be pointed out that the proposed path planing method is also applicable when static sensors are absent and thus mobile robots plan there trajectories to patrol or cover the given area under surveillance.

A. Sensor Network Model

We consider a mixed sensor network made of a large number of sensor nodes deployed in a large region $\mathcal{A}$ as shown in Fig. 1.

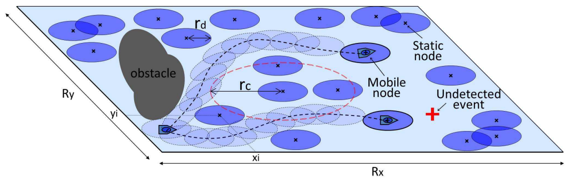


Fig. 1. Mixed sensor network model.

We assume that the region under monitored is a large rectangular area $\mathcal{A} = \mathcal{R}_x \times \mathcal{R}_y$ and a large set $\mathcal{S}$ of $S = |\mathcal{S}|$ static sensor nodes are randomly placed in the area $\mathcal{A}$, at positions $\mathbf{x}_i = (x_i, y_i)$, $i = 1, \cdots, S$. In addition, we assume that a small set $\mathcal{M}$ of $M = |\mathcal{M}|$ mobile sensor nodes are available and their position after the k-th time step is $\mathbf{x}_i(k) = (x_i(k), y_i(k))$, $i = 1, \cdots, M$, $k = 0, 1, \cdots$. For notational convenience, we define the set of *all* sensor nodes $\mathcal{N} = \mathcal{S} \cup \mathcal{M}$ and in this set the mobile nodes are re-indexed as $m = S+1, \cdots, N$, where $N = S + M$.

We assume that all sensors sense the environment according to the probabilistic sensing model [21]. This model is more realistic compared to the Boolean sensing model as it can capture the degradation of a sensor's sensing capability as the distance between the sensor and measuring point increases. In this model, a quantity r_u is defined in order to capture the uncertainty in sensor detection. According to this model, the initial (given one sample) probability that a sensor $s \in \mathcal{N}$ detects an event to a distance r is

$$p_s(r) = \begin{cases} 1, & r \leq r_u \\ e^{-\beta(r-r_u)^\gamma}, & r_u < r < r_d \\ 0, & r \geq r_d \end{cases} \tag{1}$$

where, r_u defines the starting of uncertainty in sensor detection, r_d is the maximum sensing range of the node and the parameters β and γ are adjusted according to the physical properties of the sensor and the environment. This model is more general because it becomes Boolean sensing model when $r_u = r_d$. It is also assumed that all static and mobile nodes have common communication ranges $r_c > r_d$ as well as sensing characteristics and know their location through a combination of GPS and localization algorithms.

The neighborhood of a sensor $s \in \mathcal{N}$ is the set of all sensors nodes that are one hop away, i.e., the nodes that are located at a distance less than or equal to r_c from s. This set is denoted by

$$\mathcal{H}_{r_c}(s) = \{j \; : \; \|\mathbf{x}_s - \mathbf{x}_j\| \leq r_c, \; j \in \mathcal{N}, j \neq s\} \tag{2}$$

where $\|\cdot\|$ denotes the Euclidean norm.

In addition, we consider a set $\mathcal{E}$ of $E = |\mathcal{E}|$ point events that can occur in $\mathcal{A}$ at positions $\mathbf{e}_i = (x_i^e, y_i^e)$, $i = 1, \cdots, E$. These events are uniformly distributed in the areas not monitored by the static sensor nodes and are temporally static, i.e. they occurred continuously in time. The case of temporally dynamic events is also addressed however it is omitted due to space limitations.

B. Event Detection

As previously implied, all static and mobile nodes sense the environment according to the probabilistic sensing model and it is assumed that all sensors' samples are temporally and spatially independent. An event is considered as detected (found) when is occurred at a point that falls within the sensing range r_d of a mobile sensor and the corresponding occurrence point is sensed with probability close to 1, given any concurrent measurements of neighboring static and mobile sensors. In other words, a binary variable $I_D(\mathbf{e}_j)$ is used to indicate whether an event $\mathbf{e}_j$ has been detected or not by a mobile sensor at the current step as follows:

$$I_D(\mathbf{e}_j) = \begin{cases} 1 & if \; P_D(\mathbf{e}_j) \geqslant \tau_d \\ 0 & otherwise \end{cases} \tag{3}$$

where $P_D(\mathbf{e}_j)$ is defined below and τ_d is a pre-defined threshold close to 1

$$P_D(\mathbf{e}_j) = 1 - \prod_{i \in \mathcal{H}_{r_d}(\mathbf{e}_j)} (1 - p_i(r_{ij})) \tag{4}$$

where $\mathcal{H}_{r_d}(\mathbf{e}_j) = \{i \; : \; \|\mathbf{e}_j - \mathbf{x}_i\| \leq r_d, \; j \in \mathcal{E}, \; i \in \mathcal{N}\}$ defines all sensors that are located at a distance less than or equal to r_d from the event $\mathbf{e}_j$[1].

C. Dynamic Coverage $\mathcal{C}_k$

To study the coverage of such mixed sensor networks we define a coverage measure called the dynamic coverage. Unlike static coverage (defined as the instantaneous ratio of covered area by the sensor network to the area of interest), dynamic coverage $\mathcal{C}_k$ is defined as the ratio of covered area by the sensor network to the area of interest during a time interval $[0, k]$. In other words, it defines the probability that a temporally static point event can be detected within a time interval $[0, k]$ by at least one sensor node in the sensor field. A similar measure of dynamic coverage is also considered in [5], [11], [14]. The dynamic coverage depends not only on

[1]Given that $\mathbf{e}_j$ is unknown, the evaluation of eq.4 requires each mobile to receive $p_i(.)$ of its neighbors and assumes that two events occurred at least $2r_d$ apart.

the sensing model, the number of nodes and node placement strategy, but also depends on the mobility behavior of the nodes. Hence, proper motion planning is required to exploit the full advantage of mobile sensors.

To make the concept of dynamic coverage a computationally measurable objective, the entire sensor field area $\mathcal{A}$ is discretized into an $X \times Y$ matrix G_k, $k = 0, 1, \cdots$. Initially, a zero value is assigned at each cell $G_0(i,j) = 0$, $i = 1, \cdots, X$, $j = 1, \cdots, Y$ and then at $k = 1$ (first sample) a value is assigned at each cell $G_1(i,j)$ depending on its distance from the stationary sensors as well as its distance for the initial position of the mobile sensors. Then, at each step k every sensor is sampling the environment and mobile sensors are moving around as well and thus the following updating rule is used for the G_k matrix,

$$G_{k+1}(i,j) = \begin{cases} 1 - (1 - G_k(i,j)) \prod_s (1 - p_s(\bar{r})), \\ \qquad \text{if } (i,j) \in D_{\bar{r}_d}(\bar{\mathbf{x}}_s),\ s \in \mathcal{N} \\ G_k(i,j), \\ \qquad \text{otherwise} \end{cases} \tag{5}$$

where $\bar{\mathbf{x}}_s$ are the coordinates of sensor s (mobile or static) in the grid G_k, $D_{\bar{r}_d}(\bar{\mathbf{x}}_s)$ is the set of grid cells covered by sensor $s \in \mathcal{N}$ with sensing range r_d, $\bar{r}$ is the discretized distance of cell $G_k(i,j)$ from $\bar{\mathbf{x}}_s$ and $p_s(\bar{r})$ is given by eq. (1).

The $\mathcal{C}_k$ represents the *dynamic coverage* over a time interval $[0, k]$ and it is an appropriate quality metric for applications that require coverage of all locations within some time interval. $\mathcal{C}_k$ also represents the probability of detection of static events existing in the sensor field within a time interval $[0, k]$

$$\mathcal{C}_k = \frac{1}{X \times Y} \times \sum_{i=1}^{X} \sum_{j=1}^{Y} G_k(i,j) \tag{6}$$

Therefore, when temporally static events are considered, the objective is to maximize the dynamic coverage rate over a time interval, this objective can be satisfied by finding the near-optimal paths to be followed by the mobile sensors in the sensor field in a distributive and collaborative manner. Its worth pointing out that finding optimal solutions to any arbitrary problem instance is not possible due to the complexity of the problem.

D. Mobile Sensor Node Model

Mobile sensor nodes can move in the sensor field and autonomously plan their trajectories to enhance the dynamic coverage and minimized the detection latency. The state of the m-th mobile node at time k is denoted by its position $\mathbf{x}_m(k)$ and its heading direction $\theta_m(k)$. Each mobile node m is capable to move with variable speed $\upsilon_m(k) \in [\upsilon_{min} \quad \upsilon_{max}]$ and make path planning decisions at discrete time intervals. Note that this model also considers the maneuverability constraints of the mobile platform using some angle ϕ which constrains the maximum allowed difference between $\theta_m(k)$ and $\theta_m(k+1)$ and allows variable speed with maximum velocity of υ_{max}.

Finally, we describe the information required by each mobile in order to run the proposed path planning algorithm. Each mobile uses a *coverage cognitive map*, an $X \times Y$ matrix P_k^m, $m \in \mathcal{M}$ where it keeps the state of the field. Ideally P_k^m should remain $P_k^m = G_k$ at all times k, since the matrix G_k represents the accurate global state of the field which is used for the computation of the dynamic coverage $\mathcal{C}_k$. Clearly, in a dynamic environment where several sensors move, fail or more sensors are added as well as due to limited communication between mobile nodes, it is impossible to guarantee that $P_k^m = G_k$ at all times. However, we emphasize, that the proposed algorithm, that will run by a mobile located at some position $\mathbf{x}_m(k)$, computes its path based *only* on local information, i.e., information in the submatrix of P_k^m that corresponds to the cells $\mathcal{D}_{\bar{r}_c}(\bar{\mathbf{x}}_m(k))$, and thus, it is sufficient to have accurate information only for the $\mathcal{D}_{\bar{r}_c}(\bar{\mathbf{x}}_m(k))$ submatrix. This is easily attainable since the required information can be obtained from the one-hop neighbors.

E. Distributed Path Planning

The path planning method is based on Receding-Horizon approach where at each step the mobile's controller evaluates the cost of moving to a finite set of candidate positions and moves to the one that minimizes an overall cost.

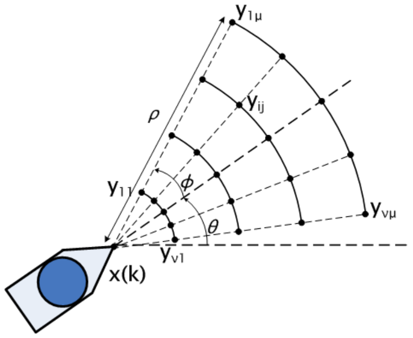


Fig. 2. Evaluation of the mobile node's next step.

As shown in Fig. 2, suppose that during the kth step, the mobile node is at position $\mathbf{x}(k)$ and its heading to a direction θ. The next candidate positions are the $\nu\mu$ points $\mathbf{y}_{11}, \cdots, \mathbf{y}_{\nu\mu}$ that are distributed on a circular sector with center $\mathbf{x}(k)$, radius ρ and angle $\theta - \phi$ and $\theta + \phi$, where $\nu \in \{2n+1, \forall n \in \mathbb{Z}^+\}$. The mobile node evaluates a cost function $J(\mathbf{y}_{ij})$ for all candidate locations $(\mathbf{y}_{11}, \cdots, \mathbf{y}_{\nu\mu})$ and moves to the location $\mathbf{x}(k+1) = \mathbf{y}_{i^*j^*} = \mathbf{x}(k) + \frac{j^*\rho}{\mu}.e^{\mathrm{i}(\theta+\varphi_{i^*})}$ where i is the imaginary unit and i^*j^* are the indexes that minimize $J(\mathbf{y}_{ij})$,

$$i^*j^* = \arg \min_{\substack{1 \le i \le \nu \\ 1 \le j \le \mu}} \{J(y_{ij})\} \tag{7}$$

In this model, θ is the direction that the mobile is heading, ϕ is the maximum angle that the mobile can turn in a single step, $\nu \times \mu$ is the number of candidate positions that are being evaluated for the next step and ρ is the maximum distance that the mobile can cover in one time step when the mobile is moving with its maximum velocity v_{max}. This model allows the mobile node to select an appropriate speed level at each time step and thus to adapt its speed based on the objectives that its tries to achieve. Assuming that the time t_s between two consecutive time steps k and $k+1$ is constant, we get $t_s = \frac{\rho}{v_{\max}} = \frac{j\rho}{\mu v_i}$. Hence, these discretized speed levels are defined by

$$v_i = \frac{j}{\mu} v_{\max},\ j = 1, \cdots, \mu \tag{8}$$

The objective function $J(y_{ij})$ that each mobile is trying to minimize is of the form

$$J(\mathbf{y}_{ij}) = \sum_{o \in \mathcal{O}} w_o J_o(\mathbf{y}_{ij}) \tag{9}$$

where $\mathcal{O}$ is a set of indexes such that the functions J_o, $o \in \mathcal{O}$ are normalized cost functions with $0 \le J_o \le 1$ and are defined to achieve certain objectives. w_o are non-negative constant weights that are used to trade off these objectives. For the purposes of this section, $\mathcal{O} = \{t, s, c, a, b\}$ but other functions can also be included (e.g. a cost function that depend on time or the residual energy of mobile nodes). In order to improve the area coverage, the mobiles should move towards large uncovered regions and on their path, they should try (to the extend possible) to avoid areas that are covered by static sensors or have been covered by other mobile nodes. For the purposes of this paper the following normalized functions have been used: $J_t(\cdot)$ which penalizes positions that are away from large coverage holes, $J_s(\cdot)$ and $J_c(\cdot)$ which penalize positions that are close to regions been covered by other sensors (stationary or mobile), $J_a(\cdot)$ which enable mobiles to avoid obstacles and $J_b(\cdot)$ which prevents mobiles moving outside the region under monitored. Next, we present the formulas of these functions.

a) Target Cost Function: At each step k, the mobile node m uses the information stored in its P_k^m matrix to search for the center of the biggest coverage hole (uncovered region) at a radius r_z from its current location. This can be done efficiently using the zoom algorithm [12]. The zoom algorithm divides the submatrix of P_k^m that corresponds to the cells $\mathcal{D}_{r_z}(\bar{\mathbf{x}}_m(k))$ in a 2D divide-and-conquer manner and outputs the hole center position. The center of the hole becomes the current target destination point $\mathbf{x}_t$ of the mobile. The cost $J_t(\mathbf{y})$ is a function that pulls the mobile towards its target and is a function of the distance between the mobile and the target position. This cost function is given by

$$J_t(\mathbf{y}) = \frac{\|\mathbf{y} - \mathbf{x}_t\|}{r_z} \tag{10}$$

In this function, r_z is the maximum distance between the mobile node and its target and is used for normalization. The radius r_z is an important parameter of the path planning algorithm and previous results [22] indicate that is more beneficial (achieves better area coverage and event detection time) if the dynamic target is determined more closer ("locally") to the mobile as opposed to more "globally". Thus r_z range must be fairly small compared to the sensor field area. Note that since $r_z \le r_c - r_d$ in order to have accurate information, a smaller r_z is advantageous as it implies that less information (i.e. less computation and communication) is needed for the coverage hole estimation.

b) Neighboring Sensor Cost Function: The objective of this function is to push the mobile away from areas covered by other sensors. The cost function $J_s(\mathbf{y})$ used involves a repulsion force that pushes the mobile away from its closest neighbor. The form of this function is given by

$$J_s(\mathbf{y}) = \max_{j \in \mathcal{H}_{r_c}(m)} \left\{ \exp\left(- \frac{\|\mathbf{y} - \mathbf{x}_j\|^2}{r_s^2} \right) \right\} \tag{11}$$

where $\mathcal{H}_{r_c}(m)$ is the set of all nodes in the communication range r_c of the mobile m. The detection range r_d quantifies the size of the region around the mobile m to be repelled by its neighbors.

c) Coverage Cost Function: The cost function $J_c(\mathbf{y})$, similarly to J_s, is designed to push the mobile away from areas that have been covered by other sensors (stationary or mobile) or by itself using the relevant information from the cognitive map of the mobile node. This function takes a larger value if the candidate position is adequately covered by other sensors and a small value otherwise. This cost function is given by

$$J_c(\mathbf{y}) = \frac{1}{\pi r_d^2} \sum_{\{i,j\} \in \mathcal{D}_{\bar{r}_d}(\bar{\mathbf{y}})} P_k(i,j) \tag{12}$$

where $\mathcal{D}_{\bar{r}_d}$ is the set of cells that exist in a discretized disk of the mobile's P_k matrix, centered at the position $\bar{\mathbf{y}}$ with radius of $\bar{r}_d$.

d) Obstacle Avoidance Cost Function: This function enable the robots to avoid hitting obstacles that exist in the environment. The obstacle avoidance cost function J_a is similar to J_s and its form is given by

$$J_a(\mathbf{y}) = \exp\left(- \frac{\big(r_o - \|\mathbf{y} - \mathbf{x}(k)\|\big)^{10}}{r_d^{10}} \right) \tag{13}$$

where r_d is the detection range and r_o indicates the distance of the obstacle's boundary from the mobile's current position $\mathbf{x}(k)$ and its provided by the mobile's on board range-finding sensors such as low cost ultrasonic sensors or infrared sensors. The information provided by range-finding sensors can be combined with the model presented in Fig. 2 to associate each candidate location $\mathbf{y}_{ij}$ with a cost. For instance, the geometry of detectors can be combined with each candidate direction φ_i, $i = 1, \cdots, \nu$ to provide the distance to obstacles associated with the candidate direction.

e) Boundaries Cost Function: For completeness, note that another cost function is used that prevents mobiles from stepping outside the field along with projection which means that mobile sensors return to the interior of the field whenever they reach to boundaries in a manner similar to that of a light wave reflecting on a mirror. This boundary cost function $J_b(\mathbf{y})$ penalizes all candidate positions $\mathbf{y}$ that are not included in the field area $\mathcal{A}$ and is given by

$$J_b(\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} \notin \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

F. Distributed Asynchronous Collaboration Scheme

Since each mobile determines its path autonomously, when two or more mobiles come close to each other it is very likely that the information they will use to estimate the next target position will be the same and as a result they will all estimate the same target location. To avoid this problem we utilize a collaboration protocol that enables mobile nodes to exchange some information in order to avoid moving towards to the same point and search different areas.

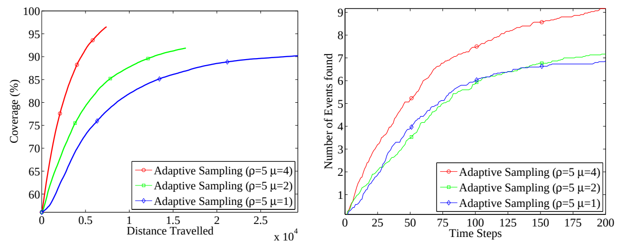
The collaboration protocol developed is as follows: If at step k, the mobile nodes come into communication range r_c when they were out of range in step $k-1$, they exchange their *entire maps* P_k^m *only* if their coverage difference exceeds a predefined threshold τ_C since the last time communicated [2]. If they are in r_c at time $k-1$ then they only exchange their *positions* $\mathbf{x}^m(k)$ and dynamic *target coordinates* $\mathbf{x}_t^m(k)$. Note that the P_k^m is exchanged only in an event driven way.

After a mobile node i has exchanged collaboration messages with its neighboring mobiles it has all the necessary information to execute the collaboration protocol. Thus at first, it merges its cognitive map P_k^i with the cognitive maps P_k^j, $j \neq i$ received from its "new" neighbors, so that it does not explore areas already explored by other mobile nodes. Merging policy is based on a cell value maximization rule. Afterwards, the mobile node i utilizes the current locations $\mathbf{x}^j(k)$ and dynamic target coordinates $\mathbf{x}_t^j(k)$ received by its neighboring mobiles $j \neq i$ (as well as the locations received by its neighboring stationary nodes) in order to update its P_k^i cognitive map and to avoid going towards the same point. Thus, once the mobile i has received all target points from its neighbors, it forms the matrix $D_{r_z}(\bar{\mathbf{x}}_i(k))$ (which is a copy of the set of P_k^i cells that corresponds to the distance r_z from the current position of mobile i) and updates the $D_{r_z}(\bar{\mathbf{x}}_i(k))$ matrix by assuming that these targets points constituted covered areas. Finally, it executes the zoom algorithm [12] where the input is the $D_{r_z}(\bar{\mathbf{x}}_i(k))$ matrix and the output is the dynamic target point $\mathbf{x}_t^i(k)$ of the mobile node i which is definitely different that the target points of its neighboring mobiles. As mobile nodes remain in communication range there is no need to exchange their cognitive maps since their maps are updated accurately using the positions of neighbors. It should be pointed out that proposed scheme is distributed (no need for a central controller) and utilizes only local information available in the neighborhood of the mobile node.

III. Simulation Results

In the first simulation, we investigate the parameters of the adaptive speed policy with respect to the average dynamic coverage and average number of static events detected using monte carlo simulations. We assumed 100 sensor fields with 300 randomly distributed stationary sensors and in each field 10 static events not initially detected exist. The objective is to investigate the adaptive versus the constant speed policy. The key parameter here is μ, for $\mu = 1$ mobiles are moving with constant maximum speed and as μ increases more speed levels and thus speed adaptivity is allowed, however as μ increases computation overhead also increases. These experiments refer to a square sensor field of area $A = 40000m^2$ and the sensors parameters are set to $r_d = 6m$ with $r_u = 4m$, $\beta = \gamma = 1$. The radius r_z where the dynamic target is found is set to $r_z = 19m$ and $r_c = r_z + r_d = 25m$. The weights are set to $w_t = 0.5, w_s = 0.2, w_c = 0.3, w_a = w_b = 1$ and thresholds for event detection and exchange of cognitive maps are set to $\tau_d = 0.999$ and $\tau_C = 5\%$ respectively. The mobile maneuverability parameters are set to $\rho = 5m$ and $\phi = 40°$ while for every decision $\nu\mu$ candidate next positions are considered with $\nu = 5$. Fig. 3 depicts the results for $\mu = 1, 2, 4$.

As shown in Fig. 3 the performance in terms of average dynamic coverage as well as the number of events detected increases but the total distance travelled by mobile nodes decreases!. This is an advantageous and desirable behavior and can be justified because using an adaptive speed policy (i.e. modifying the speed of the mobile at each step) enable mobiles to make more precise movements, which decreases the distance the mobiles moved (i.e. the energy needed for mobility) and simultaneously increases the dynamic coverage performance over time. In other words, if a mobile considers more candidate positions including positions that fall very near to it, (meaning going slower or make more precise navigation) it enables the mobile to go slower when needed but still have the option to go fast and thus decide its next position more accurately.



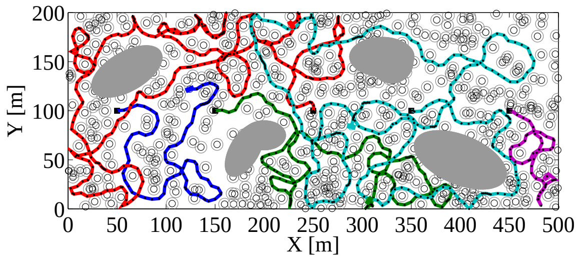
(a) Coverage Vs Distance. (b) Detected static events Vs Time.

Fig. 3. The average coverage and average number of detected static events accomplished by $M = 5$ mobile nodes after 200 moving steps for the adaptive and constant maximum speed policies.

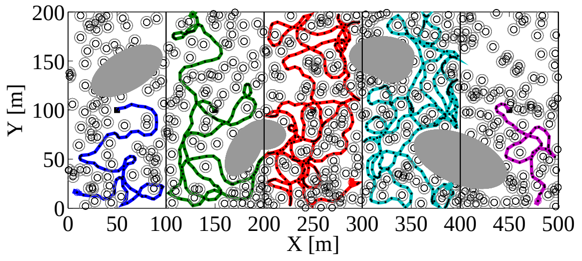
Finally, the last simulation evaluates the robustness of the proposed Distributed Collaborative coverage path planning Algorithm (DCA) with respect to the failures of mobile sensors. The lifetime of mobile sensors is modeled as an exponential distribution with failure rate $\lambda = 1/T$, where T denotes the simulation time. To illustrate the effectiveness of the proposed DCA, we have implemented another Centralized Partitioning coverage path planning Algorithm (CPA). In CPA is assumed that a central controller partitions the area under monitored into m equal partitions, where m denotes the number of mobile sensors, and assigns each mobile to a different partition. Fig. 4 illustrates the paths followed using

[2]Each mobile must keep in its memory a communication matrix where it tracks with which mobiles was in communication during the previous step as well as what was its coverage value since the last time it communicates with another mobile.

the DCA and CPA on the same sparse sensor field with 500 stationary sensors and 4 obstacles. The other parameters used for this simulation are the same as in the previous simulation. Note that in this scenario three mobile sensors fail before the end of the simulation time $T = 300$ time steps.



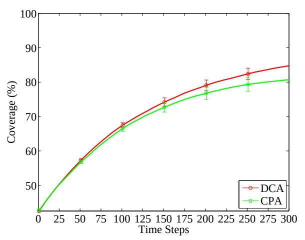
(a) Coverage path planning using DCA.



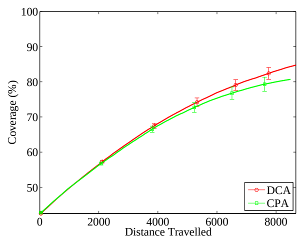
(b) Coverage path planning using the CPA.

Fig. 4. Paths followed using DCA and CPA under mobile sensor failures in a sparse sensor field with obstacles.

The two approaches are evaluated/compared using extensive monte carlo simulations for the case when mobiles failed according to the exponential distribution with failure rate $\lambda = 1/300$. We assumed 100 sensor fields with 500 randomly distributed stationary sensors and for each field identical initial positions and failures of mobile sensors are considered when simulating each algorithm. Results are shown in Fig. 5. As expected DCA outperforms CPA in both coverage and distance travelled performance due to its adaptive and distributed behavior. Therefore the proposed DCA is robust and adaptive to sensor node failures.



(a) Coverage Vs Time.



(b) Coverage Vs Distance.

Fig. 5. DCA vs CPA under mobile sensor failures: Average coverage accomplished by $M = 5$ mobile nodes after 300 moving steps.

IV. Conclusion

This paper presents an efficient adaptive-distributed-collaborative framework for mixed WSNs where autonomous mobile sensors navigate through a sparse stationary WSN searching for events and improving area coverage. An adaptive speed policy have been proposed to improve the performance and the energy consumption of mobile sensors and the robustness of the proposed approach has been evaluated under mobile sensor failures.

Acknowledgments

This work is partly supported by the Cyprus Research Promotion Foundation under grant ΤΠΕ/ΟΡΙΖΟ/0609 (ΒΕ) /06 and co-funded by the Republic of Cyprus and the European Regional Development Fund.

References

[1] A. R. Washburn and M. Kress, *Combat Models*. Springer, 2009.
[2] S. J. Benkoski, M. G. Monticino, and J. R. Weisinger, "A survey of the search theory literature," *Naval Research Logistics*, vol. 38, no. 4, pp. 469–494, 1991.
[3] A. Ghosh and S. K. Das, "Coverage and connectivity issues in wireless sensor networks: A survey," *in Pervasive and Mobile Computing*, vol. 4, no. 3, pp. 303–334, 2008.
[4] M. Cardei and J. Wu, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC Press, 2004, ch. 19.
[5] B. Liu, O. Dousse, P. Nain, and D. F. Towsley, "Dynamic coverage of mobile sensor networks," *CoRR*, vol. abs/1101.0376, 2011.
[6] J. Cortes, "Coverage optimization and spatial load balancing by robotic sensor networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 749 –754, 2010.
[7] W. Li and C. G. Cassandras, "Distributed cooperative coverage control of sensor networks," in *Proceedings of 44rd IEEE CDC*, 2005.
[8] A. Deshpande, S. Poduri, D. Rus, and G. S. Sukhatme, "Distributed coverage control for mobile sensors with location-dependent sensing models," in *Proceedings of the IEEE ICRA'09*, 2009, pp. 3493–3498.
[9] A. Howard, M. Mataric, and G. Sukhatme, "Mobile sensor network deployment using potential fields:a distributed, scalable solution to the area coverage problem," in *DARS*, 2002.
[10] A. Ghosh, "Estimating coverage holes and enhancing coverage in mixed sensor networks," in *Local Computer Networks*, 16-18 Nov. 2004, pp. 68–76.
[11] T. Wimalajeewa and S. K. Jayaweera, "Impact of mobile node density on detection performance measures in a hybrid sensor network," *IEEE Trans. Wireless. Comm.*, vol. 9, no. 5, pp. 1760–1769, 2010.
[12] T. P. Lambrou, C. G. Panayiotou, S. Felici-Castell, and B. Beferull-Lozano, "Exploiting mobility for efficient coverage in sparse wireless sensor networks," *Wireless Personal Communications Journal*, vol. 54, no. 1, pp. 187–201, 2010, iSSN: 0929-6212.
[13] T. P. Lambrou and C. G. Panayiotou, "A testbed for coverage control using mixed wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 527–537, 2012.
[14] E. Yanmaz and H. Guclu, "Stationary and mobile target detection using mobile wireless sensor networks," in *IEEE INFOCOM 2010 Workshop on Computer Communications*.
[15] N. Bartolini, T. Calamoneri, T. La Porta, and S. Silvestri, "Mobile sensor deployment in unknown fields," in *IEEE INFOCOM 2010*.
[16] S. Megerian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Worst and best-case coverage in sensor networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 1, pp. 84 – 92, 2005.
[17] T. P. Lambrou and C. G. Panayiotou, "Improving area coverage using mobility in sensor networks," in *ISYC 2006*, Ayia Napa, Cyprus.
[18] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224 –241, 1992.
[19] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.
[20] S. G. Loizou and K. J. Kyriakopoulos, "Navigation of multiple kinematically constrained robots," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 221–231, 2008.
[21] A. Hossain, P. Biswas, and S. Chakrabarti, "Sensing models and its impact on network coverage in wireless sensor network," in *the 3rd IEEE Conference on Industrial and Information Systems*, Dec 2008.
[22] T. P. Lambrou and C. G. Panayiotou, "On the optimal search neighborhood in mixed wireless sensor networks," in *50th IEEE Conference on Decision and Control*, Orlando, Florida, USA, Dec. 2011.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Design of force-driven online motion plans for door opening under uncertainties

Yiannis Karayiannidis, Christian Smith, Francisco E. Viña, Petter Ögren, and Danica Kragic
Computer Vision and Active Perception Lab., Centre for Autonomous Systems,
School of Computer Science and Communication, Royal Institute of Technology (KTH)
SE-100 44 Stockholm, Sweden.
{yiankar|ccs|fevb|petter|dani}@kth.se

Abstract— **The problem of door opening is fundamental for household robotic applications. Domestic environments are generally less structured than industrial environments and thus several types of uncertainties associated with the dynamics and kinematics of a door must be dealt with to achieve successful opening. This paper proposes a method to open doors without prior knowledge of the kinematics. The proposed method can be implemented on a velocity-controlled manipulator with force sensing capabilities at the end-effector. The velocity reference is designed by using feedback of force measurements while constraint and motion directions are updated online based on adaptive estimates of the position of the door hinge. The online estimator is designed to identify the unknown directions. The proposed scheme has theoretically guaranteed performance which is further demonstrated in experiments on a real robot. Experimental results also show the robustness of the proposed method under disturbances introduced by the motion of the mobile platform.**

I. Introduction

Doors or drawers can be considered typical components in a domestic environment. Hence, a household robot should be able to open doors in a wide range of household applications. A typical example of domestic manipulation may be the task of retrieving a glass from a cupboard. In this case, the task also involves the prerequisite task of opening the door of the cupboard so that the primary task of picking up the glass can be performed. Moreover, in order to bring the glass to its final destination, the robot may have to negotiate doors between rooms or hallways. Furthermore, domestic environments include several types of uncertainty that disqualifies the use of motion control with preplanned trajectories typically used on stiff industrial robots, making the door opening task more challenging. Thus, the motion plans have to be recomputed online in reaction to encountered measurement errors.

Typical sources of uncertainty in the door-opening problem are the location of the hinge in terms of kinematics, and the force model of the dynamic motion of the door. If we also consider a mobile robot, then extra difficulties arise from the disturbances caused by motion of the platform.

Pioneering work on the door opening problem include [1] and [2]. In [1], experiments on door opening with an autonomous mobile manipulator were performed under the assumption of a known door model, using the combined motion of the manipulator and the mobile platform, while in [2], velocity-based estimation of the constraints describing the kinematics of the motion for the door opening problem is proposed. Recent works of [3] and [4] has been inspired by [2]; however, they suffer from ill-defined normalization when the velocity is small and estimation lags. Furthermore, there exist several position-based estimation techniques [5]–[8]; optimization algorithms that uses the end-effector position are used in parallel with controllers that provide the system with the proper compliance in order to deal with inaccurate trajectory planning. On the other hand, off-line methods using prior phases have been also proposed: slowly pulling and pushing in a prior phase [9], probabilistic methods based on a set of motion observations of the objects [10] or based on the use of particle filters and extended Kalman filters for an a priori defined detailed model of the door [11]. Another part of the literature on the door opening problem exploits advanced hardware capabilities to accomplish the manipulation task: combination of tactile-sensor and force-torque sensor [12], clutches that disengage selected robot motors from the corresponding actuating joints for passive joint's rotation [13], exploitation of the compliance of the DLR lightweight robot II [14] and use of the humanoid robot HRP-2 exerting impulsive force on a swinging door [15].

In this paper, we propose a controller which is proved to achieve stable force regulation as well as learning the constraint direction, and thus is able to continously generate online motion plans for smooth door opening in case of uncertainty. The proposed method can be implemented on any velocity controlled manipulator — with force measurements at the end-effector or wrist — and differs from the existing work by simultaneously providing on-line performance while explicitly including the uncertain estimates in the controller.

II. System and Problem Description

A. Notation and Preliminaries

Bold roman small letters denote vectors while bold roman capital letters denote matrices. The generalized position of a moving frame $\{i\}$ with respect to a inertial frame $\{B\}$ (typically located at the robots base) is described by a position vector $\mathbf{p}_i \in \mathbb{R}^m$ and a rotation matrix $\mathbf{R}_i \in SO(m)$ where $m = 2$ for the planar case. We also consider the following normalization and orthogonalization operators:

$$\underline{\mathbf{z}} = \frac{\mathbf{z}}{\|\mathbf{z}\|} \tag{1}$$

$$\mathbf{s}(\mathbf{z}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{z} \tag{2}$$

with $\mathbf{z}$ being any non-trivial two-dimensional vector. Note that in case of $\mathbf{z} = \mathbf{z}(t)$ the derivative of $\underline{\mathbf{z}}$ is calculated as:

$$\dot{\underline{\mathbf{z}}} = \|\mathbf{z}\|^{-1}\mathbf{s}(\underline{\mathbf{z}})\mathbf{s}(\underline{\mathbf{z}})^{\top}\dot{\mathbf{z}}. \tag{3}$$

Furthermore, we denote with $\mathcal{I}(z)$ the integral of some scalar function of time $z(t) \in \mathbb{R}$ over the time variable t, i.e:

$$\mathcal{I}(z) = \int_0^t z(\tau)d\tau \tag{4}$$

B. Kinematic model of robot door opening

We consider the case where the robot's end-effector has achieved a fixed grasp of the handle of a kinematic mechanism e.g. a door in a domestic environment. The term fixed grasp denotes that there is no relative translational velocity between the handle and the end-effector but we place no constraints on the relative rotation of the end-effector around the handle. We consider also that the motion of the handle is inherently planar which implies a planar problem definition.

Let $\{e\}$ and $\{o\}$ be the end-effector and the door frame respectively (Fig. 1); the door frame $\{o\}$ is attached at the hinge which in our case is the center of door-mechanism rotation. The radial direction vector $\mathbf{r}$ is defined as the relative position of the aforementioned frames:

$$\mathbf{r} \triangleq \mathbf{p}_o - \mathbf{p}_e \tag{5}$$

By expressing $\mathbf{r}$ with respect to the door frame and differentiating the resultant equation we get:

$$\dot{\mathbf{R}}_o {}^o\mathbf{r} + \mathbf{R}_o {}^o\dot{\mathbf{r}} = \dot{\mathbf{p}}_o - \dot{\mathbf{p}}_e \tag{6}$$

The substitutions ${}^o\dot{\mathbf{r}} = \dot{\mathbf{p}}_o = 0$ and $\dot{\mathbf{R}}_o = \omega \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{R}_o$, with ω being the rotational velocity of the door, give us:

$$\dot{\mathbf{p}}_e = -\mathbf{s}(\mathbf{r})\omega \tag{7}$$

which describes the first-order differential kinematics of the door opening problem in case of a revolute hinge. Notice that the end-effector velocity along the radial direction of the motion is zero, i.e:

$$\mathbf{r}^{\top}\dot{\mathbf{p}}_e = 0 \tag{8}$$

The latter can be regarded as the constraint on the robot end-effector velocity.

C. Robot kinematic model

In case of velocity controlled manipulators, the robot joint velocity is controlled directly by the reference velocity $\mathbf{v}_{\text{ref}}$. In particular, the reference velocity $\mathbf{v}_{\text{ref}}$ can be considered as a kinematic controller which is mapped to the joint space in order to be applied at the joint velocity level as follows:

$$\dot{\mathbf{q}} = \mathbf{J}^{+}(\mathbf{q})\mathbf{v}_{\text{ref}} \tag{9}$$

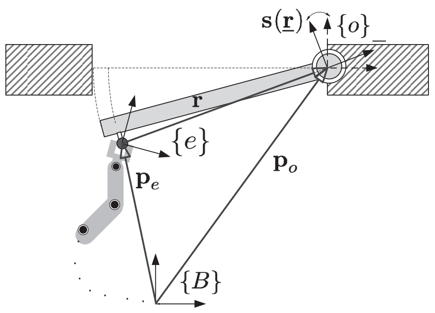


Fig. 1: Kinematics of the door opening

with $\mathbf{q},\ \dot{\mathbf{q}} \in \mathbb{R}^n$ being the joint positions and velocities and $\mathbf{J}(\mathbf{q})^{+} = \mathbf{J}(\mathbf{q})^{\top}\left[\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^{\top}\right]^{-1}$ being the pseudo-inverse of the manipulator Jacobian $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{2\times n}$ which relates the joint velocities $\dot{\mathbf{q}}$ to the end-effector velocities $\dot{\mathbf{p}}_e$; without loss of generality we consider only the translational end-effector velocity $\dot{\mathbf{p}}_e \in \mathbb{R}^2$ and the associated Jacobian. If we consider the typical Euler-Lagrange robot dynamic model, the velocity error at the joint level drive the torque (current) controller $\mathbf{u}(t)$. If we assume a high frequency current control loop with external forces' compensators and weak inertial dynamics, the kinematic model is valid.

D. Control Objective

The objective is to control the motion of the robot to achieve a smooth interaction with an external kinematic mechanism such as a door. In applications which take place in a dynamic unstructured environments such as a domestic environment, it is difficult to accurately identify the position of the hinges and the associated dynamics. Hence, it is difficult to design a priori the desired velocity within the constraints imposed by the kinematic mechanism. The execution of a trajectory which is inconsistent with system constraints gives rise to high interaction forces along the constraint direction which may be harmful for both the manipulated mechanism and the robot.

Let f_{rd} and v_d be the desired radial force and desired tangent velocity magnitudes respectively. If we define the force along the radial direction as $f_r = \underline{\mathbf{r}}^{\top}\mathbf{f}$ with $\mathbf{f} \in \mathbb{R}^2$ being the total interaction force, the control objective can be formulated as follows: $f_r \to f_{rd}$ and $\dot{\mathbf{p}}_e \to \mathbf{s}(\underline{\mathbf{r}})v_d$. These objectives have to be achieved without knowing accurately the $\underline{\mathbf{r}}$ direction which subsequently implies that there are uncertainties in the control variables f_r and $\mathbf{s}(\underline{\mathbf{r}})v_d$. From a high level perspective, we consider that the door opening task is accomplished when the observed end-effector trajectory, which coincides with the handle trajectory, enable the robot to perform the subsequent task which can be for example "get an object" or "pass through the door". Thus the command to halt the door opening procedure is given externally based on the observations of the rotation angle ϑ.

III. Control Design

A. Incorporating Force Feedback in the Velocity Reference

Let us first define an estimated radial direction $\hat{\mathbf{r}}(t)$ based on appropriately designed adaptive estimates of the center of rotation $\hat{\mathbf{p}}_o(t)$:

$$\hat{\mathbf{r}}(t) = \hat{\mathbf{p}}_o(t) - \mathbf{p}_e \tag{10}$$

For notation convenience we will drop out the argument of t from $\hat{\mathbf{r}}(t)$ and $\hat{\mathbf{p}}_o(t)$. We will use the estimated radial direction (10) considering that $\|\hat{\mathbf{r}}(t)\| \neq 0,\ \forall t$ in order to introduce a reference velocity vector $\mathbf{v}_{\text{ref}}$ for controlling the end-effector velocity:

$$\mathbf{v}_{\text{ref}} = \mathbf{s}(\hat{\underline{\mathbf{r}}})v_d - \alpha\hat{\underline{\mathbf{r}}}v_f \tag{11}$$

with α being a positive control gain acting on the force feedback term v_f which has been incorporated in the reference velocity.

We can now introduce the velocity error:

$$\tilde{\mathbf{v}} \triangleq \mathbf{v} - \mathbf{v}_{\text{ref}} \tag{12}$$

where $\mathbf{v} \triangleq \dot{\mathbf{p}}_e$ can be decomposed along $\hat{\underline{\mathbf{r}}}$ and $\mathbf{s}(\hat{\underline{\mathbf{r}}})$ and subsequently expressed with respect to the parameter estimation error $\tilde{\mathbf{p}}_o = \tilde{\mathbf{r}} = \mathbf{p}_o - \hat{\mathbf{p}}_o$ by adding $-\|\hat{\mathbf{r}}\|^{-1}\hat{\underline{\mathbf{r}}}\mathbf{r}^\top\mathbf{v}$ as follows:

$$\mathbf{v} = \mathbf{s}(\hat{\underline{\mathbf{r}}})\mathbf{s}(\hat{\underline{\mathbf{r}}})^\top\mathbf{v} - \|\hat{\mathbf{r}}\|^{-1}\hat{\underline{\mathbf{r}}}\tilde{\mathbf{p}}_o^\top\mathbf{v} \tag{13}$$

Substituting (13) and (11) in (12) we can obtain the following decomposition of the velocity error along the estimated radial direction $\hat{\underline{\mathbf{r}}}$ and the estimated direction of motion $\mathbf{s}(\hat{\underline{\mathbf{r}}})$:

$$\tilde{\mathbf{v}} = \hat{\mathbf{R}}_o \begin{bmatrix} -\|\hat{\mathbf{r}}\|^{-1}\tilde{\mathbf{p}}_o^\top\mathbf{v} + \alpha v_f \\ \mathbf{s}(\hat{\underline{\mathbf{r}}})^\top\mathbf{v} - v_d \end{bmatrix} \tag{14}$$

where $\hat{\mathbf{R}}_o \triangleq \begin{bmatrix} \hat{\underline{\mathbf{r}}} & \mathbf{s}(\hat{\underline{\mathbf{r}}}) \end{bmatrix}$.

In the next step, we are going to design the force feedback v_f employed in the reference velocity $\mathbf{v}_{\text{ref}}$. The force feedback term v_f is derived from the magnitude of the measured force components projected along the estimated radial direction:

$$\hat{f}_r = \hat{\underline{\mathbf{r}}}^\top\mathbf{f} \tag{15}$$

the corresponding force error:

$$\Delta\hat{f}_r = \hat{f}_r - f_{rd} \tag{16}$$

as well as the corresponding force error integral $\mathcal{I}(\Delta\hat{f}_r)$. In particular, for velocity controlled robotic manipulators, we propose a PI control loop of the estimated radial force error $\Delta\hat{f}_r$:

$$v_f = \Delta\hat{f}_r + \beta\mathcal{I}(\Delta\hat{f}_r) \tag{17}$$

with β being a positive control gain. By projecting $\tilde{\mathbf{v}} = 0$ along $\underline{\mathbf{r}}$ we can calculate $\hat{f}_r$ as a Lagrange multiplier associated with the constraint (6) for the system (9):

$$\hat{f}_r = f_{rd} - \beta\mathcal{I}(\Delta\hat{f}_r) + \frac{v_d\underline{\mathbf{r}}^\top\mathbf{s}(\hat{\underline{\mathbf{r}}})}{\alpha\underline{\mathbf{r}}^\top\hat{\underline{\mathbf{r}}}}. \tag{18}$$

Equation (18) is well defined for $\mathbf{r}^\top\hat{\mathbf{r}}(t) > 0$. Equation (18) is consistent to (15) in case of rigid contacts and fixed grasps.

Remark 1: For torque controlled robotic manipulators, the derivative of reference velocity also known as reference acceleration is required in the implementation. In order to avoid the differentiation of the force measurements in case of torque controlled manipulators, the force feedback part of the reference velocity should be designed using only the integral of the estimated radial force error.

B. Update Law Design

The update law for the vector $\hat{\mathbf{p}}_o$ is designed via a passivity-based approach, by defining the output of the system as follows:

$$y_f = \alpha_f\Delta\hat{f}_r + \alpha_I\mathcal{I}(\Delta\hat{f}_r) \tag{19}$$

with α_f and α_I being positive constants. Taking the inner product of $\tilde{\mathbf{v}}$ (14) with $\hat{\underline{\mathbf{r}}}y_f$ (19) we obtain:

$$\begin{aligned} y_f\hat{\underline{\mathbf{r}}}^\top\tilde{\mathbf{v}} &= y_f(-\|\hat{\mathbf{r}}\|^{-1}\tilde{\mathbf{p}}_o^\top\mathbf{v} + v_f) \\ &= -\|\hat{\mathbf{r}}\|^{-1}y_f\mathbf{v}^\top\tilde{\mathbf{p}}_o + c_1\Delta\hat{f}_r^2 \\ &\quad + c_2\mathcal{I}(\Delta\hat{f}_r)^2 + c_3\frac{d}{dt}\left[\mathcal{I}(\Delta\hat{f}_r)^2\right] \end{aligned} \tag{20}$$

where:

$$c_1 = \alpha\alpha_f, \quad c_2 = \alpha\alpha_I\beta, \quad c_3 = \frac{\alpha(\alpha_f\beta + \alpha_I)}{2} \tag{21}$$

Next, we design the update law $\dot{\hat{\mathbf{p}}}_o \triangleq -\dot{\tilde{\mathbf{p}}}_o$ as follows:

$$\dot{\hat{\mathbf{p}}}_o = \mathcal{P}\{\gamma\|\hat{\mathbf{r}}\|^{-1}y_f\mathbf{v}\} \tag{22}$$

Notice that $\mathcal{P}$ is an appropriately designed projection operator [16] with respect to a convex set of the estimates $\hat{\mathbf{p}}_o$ around $\mathbf{p}_o$ (Fig. 2) in which the following properties hold: i) $\|\hat{\mathbf{r}}\| \neq 0, \quad \forall t$, in order to enable the implementation of the reference velocity and calculate estimated radial force and ii) $\mathbf{r}^\top\hat{\mathbf{r}} > 0$; which is required for the system's stability. It

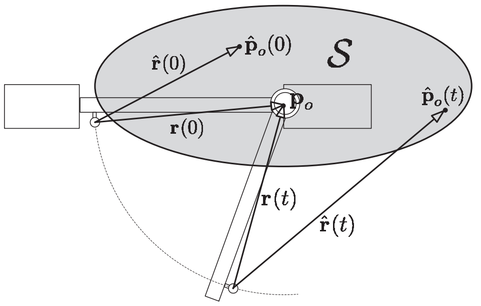


Fig. 2: Convex set $\mathcal{S}$ for the projection operator $\mathcal{P}$

is clear that the update law (22) gives rise to the potential owing to estimation error i.e. $\frac{1}{2\gamma}\tilde{\mathbf{p}}_o^\top\tilde{\mathbf{p}}_o$ and allow us to use the following function $V\left(\mathcal{I}(\Delta\hat{f}_r), \tilde{\mathbf{p}}_o\right)$ in order to prove Theorem 1 for velocity controlled manipulators. In particular $V\left(\mathcal{I}(\Delta\hat{f}_r), \tilde{\mathbf{p}}_o\right)$ is given by:

$$V\left(\mathcal{I}(\Delta\hat{f}_r), \tilde{\mathbf{p}}_o\right) = c_3\mathcal{I}(\Delta\hat{f}_r)^2 + \frac{1}{2\gamma}\tilde{\mathbf{p}}_o^\top\tilde{\mathbf{p}}_o \tag{23}$$

and is positive-definite with respect to $\mathcal{I}(\Delta\hat{f}_r)$, $\tilde{\mathbf{p}}_o$ and Theorem 1 is stated below:

Theorem 1: The kinematic controller $\mathbf{v}_{\text{ref}}$ (11) with the update law (22) applied to the system (9) achieves the following objectives: $\hat{\underline{\mathbf{r}}} \to \underline{\mathbf{r}}$, $\mathbf{v} \to \mathbf{s}(\underline{\mathbf{r}})v_d$, $\mathcal{I}(\Delta f_r) \to 0$ and $f_r \to f_{rd}$, which are equivalent with the control objective of smooth door opening stated in Section II-D.

Proof: Substituting (11) in (9) and multiplying by $\mathbf{J}(\mathbf{q})$, implies $\tilde{\mathbf{v}} = 0$. Differentiating $V\left(\mathcal{I}(\Delta\hat{f}_r), \tilde{\mathbf{p}}_o\right)$ with respect to time and substituting $\tilde{\mathbf{v}} = 0$ and (22) we get: $\dot{V} = -c_1\Delta\hat{f}_r^2 - c_2\mathcal{I}(\Delta\hat{f}_r)^2$; note that $\dot{V}$ has extra negative terms when the estimates reach the bound of the convex set and the projection operator applies and thus the stability properties of the system are not affected. Hence, $\mathcal{I}(\Delta\hat{f}_r)$, $\tilde{\mathbf{p}}_o$ are bounded and we can prove the boundedness of the following variables: (a) $\hat{f}_r$ is bounded, given the use of projection operator in (18), (b) $\mathbf{v}_{\text{ref}}$ is bounded, (c) $\dot{\mathbf{q}}$ is bounded, given the assumption of a non-singular manipulator in (9), (d) $\dot{\hat{\mathbf{p}}}_o$ is bounded, given (22) and the boundedness of $\mathbf{v}$.

The boundedness of the aforementioned variables implies that $\dot{\hat{f}}_r$ and subsequently $\ddot{V} = -2\Delta\hat{f}_r[c_1\dot{\hat{f}}_r + c_2\mathcal{I}(\Delta\hat{f}_r)]$ are bounded and thus Barbalat's Lemma implies $\dot{V} \to 0$ and in turn $\mathcal{I}(\Delta\hat{f}_r),\ \Delta\hat{f}_r \to 0$. Substituting the convergence results in (9) and (18) we get $\mathbf{v} \to \mathbf{s}(\hat{\underline{\mathbf{r}}})v_d$ and $\hat{\underline{\mathbf{r}}}^\top \mathbf{s}(\underline{\mathbf{r}}) \to 0$ for $\lim_{t\to\infty} |v_d| \neq 0$ (or for a v_d satisfying the persistent excitation condition) respectively; the latter implies $\hat{\underline{\mathbf{r}}} \to \underline{\mathbf{r}}$. Since the estimated direction of the constraint is identified we get: $\mathbf{v} \to \mathbf{s}(\underline{\mathbf{r}})v_d$, $\mathcal{I}(\Delta f_r) \to 0$ and $f_r \to f_{rd}$. □

C. Summary and Discussion

The proposed method is based on a reference velocity (11) which is decomposed to a feedforward velocity on the estimated direction of motion and a PI force control loop on the estimated constrained direction. The estimated direction is obtained on-line using the update law (22) and the definition of the radial estimate (10). The use of (22) and (10) within a typical velocity reference like (11) enables the proof of the overall scheme stability as well as the proof that the estimates converge to the true values, driving the velocity and radial force to their desired values. Note that the proposed control scheme can easily be implemented on a common robotic setup with a velocity-controlled robotic manipulator with a force/torque sensor in the end-effector frame.

It is also clear that the proposed method is inherently on-line and explicitly includes the uncertain estimates in the controller, as opposed to the state of the art for door opening (as described in Section I), which assumes that the estimate obtained in each step is approximately equal to the actual value. The proposed method can be also combined with off-line door kinematic estimation; in this case the off-line estimates can be used as the initial estimates of the estimator (22). However, our scheme is proven to work satisfactorily even in the case of large estimation errors, where off-line methods fail. Last but not least, the proposed method can be also be applied to other types of robot manipulation under kinematic uncertainties. We have chosen here the door opening problem since it is very challenging, but can be described in terms of concrete motion constraints.

IV. Experimental Evaluation

The performance was evaluated on a real robot system. The arm used is constructed from Schunk rotary modules, that can be sent velocity commands over a CAN bus. The modules incorporate an internal PID controller that keeps the set velocity, and return angle measurements. In this setup, the modules are sent updated velocity commands at 400 Hz. Angle measurements are read at the same frequency. The arm has an ATI Mini45 6 DoF force/torque sensor mounted at the wrist. The forces are also read at 400 Hz in this experiment. The force readings display white measurement noise with a magnitude of approximately 0.2 N, apart from any process noise that may be present in the mechanical system. In the experiment, we actuate the second and fifth joints, and start the experiment with the end-effector firmly grasping the handle of a cupboard door. The cupboard door is a 60 cm width IKEA kitchen cupboard, with multiple-link hinges, so that the centre of rotation moves slightly (<1 cm) as a function of door angle. The handle of the door has been extended an additional 5 cm to accomodate the width of the fingers on the parallel gripper.

We examine two scenarios. The first scenario assumes a large error in the initial estimate (initial error of 50°), but a stationary platform, while the second scenario assumes a smaller initial error (initial error of 5°) but with a moving base. The desired velocity value v_d=0.05 m/s for both scenarios. The controller gains are set to $a_f = 0.1$, $a_I = 0.05$, $\alpha = 0.001$, $\beta = 0.1$, $\gamma = 0.5$. These gains have not been tuned specifically for the robot configuration or problem parameters, in order to show the generality of the approach. Fig. 3 shows the robot performing the task in the first case, and Fig. 4 shows the robot performing the task in the second case. Note that the base motion allows the cupboard to be opened to a wider angle. The base motion was 0.3 m along a straight line, driven by a human operator at approximately 0.04 m/s. The base motion was not modelled or included in the controller, but treated as an external disturbance.

The experimental results are shown in Figs. 5 and 6 for stationary and moving base respectively. In the stationary case, both force error and estimation error converge to zero in approximately 4 s. In the moving base scenario, we see larger force errors and slower convergence. This is to be expected, as the base motion continuously injects new errors into the system.

V. Conclusions

This paper proposes a method for manipulation with uncertain kinematic constraints. It is inherently on-line and real-time, and convergence and stability is analytically provable. The method can be used with any velocity controllable manipulator with force measurements in the end-effector frame. In this paper, the method has been applied to the task of opening a door with unknown location of the hinges, while limiting the interaction forces. In particular, a velocity reference is designed using force and position measurements to deal with the door opening problem in the presence of incomplete knowledge of the door model. In

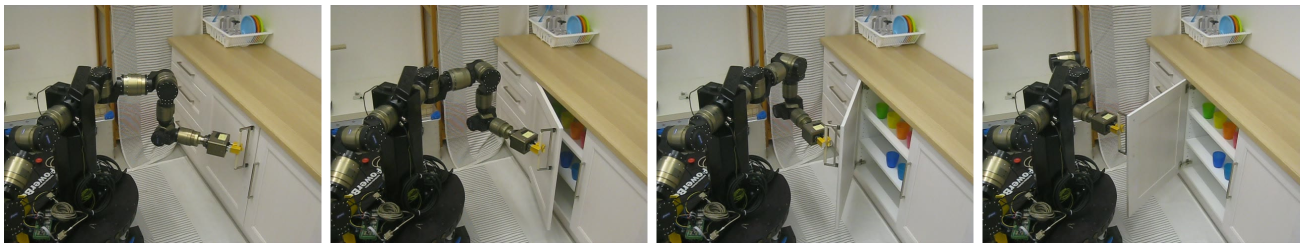


Fig. 3: The robot performing the door opening experiment. v_d = 0.05 m/s. The images are taken at $t = 0$ s, $t = 1.5$ s, $t = 3.6$ s, and $t = 5.7$ s, respectively.

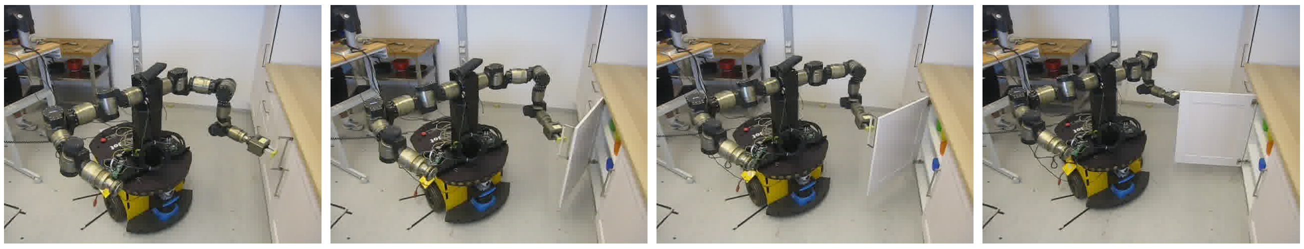


Fig. 4: The robot performing the door opening experiment, with moving base. v_d = 0.05 m/s. The images are taken at $t = 0$ s, $t = 2.2$ s, $t = 4.8$ s, and $t = 9.2$ s, respectively.

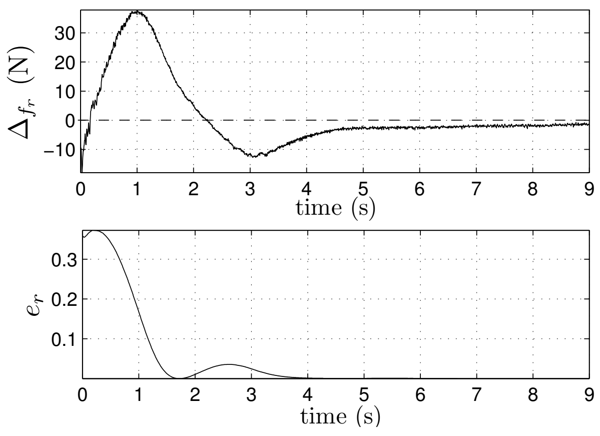


Fig. 5: Radial force (upper plot) and estimation error (lower plot) responses - robot experiment, stationary base, high error in initial estimate $\hat{\mathbf{p}}_{o1}(0)$

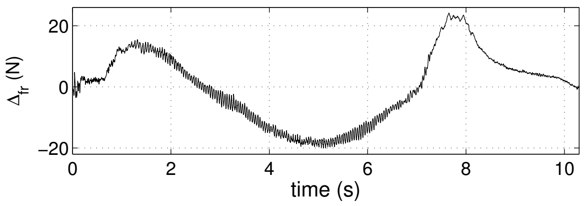


Fig. 6: Radial force error responses - robot experiment, moving base, smaller error in initial estimate $\hat{\mathbf{p}}_{o2}(0)$

the velocity reference, the constraint direction is explicitly considered uncertain by including online estimates based on the adaptation of the hinge's location. Convergence results are theoretically proved. An experiment on a real robot show that the estimates converge to the actual values even for large initial errors in the estimates as well as that the method can achieve smooth door opening even in case of disturbances due to the motion of the robotic mobile platform. Future work includes applying the proposed method to a wider range of domestic manipulation tasks with uncertainties in the kinematic constraints.

Acknowledgment

This work has been supported by the Swedish Research Council (VR) and Swedish Foundation for Strategic Research (SSF), and the European Union FP7 project RoboHow.Cog (FP7-ICT-288533). The authors gratefully acknowledge the support.

References

[1] K. Nagatani and S. Yuta, "An experiment on opening-door-behavior by an autonomous mobile robot with a manipulator," in *1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95*, vol. 2, aug 1995, pp. 45 –50.

[2] G. Niemeyer and J.-J. Slotine, "A simple strategy for opening an unknown door," in *1997 IEEE International Conference on Robotics and Automation*, vol. 2, apr 1997, pp. 1448 –1453 vol.2.

[3] E. Lutscher, M. Lawitzky, G. Cheng, and S. Hirche, "A control strategy for operating unknown constrained mechanisms," in *IEEE International Conference on Robotics and Automation*, may 2010, pp. 819 –824.

[4] D. Ma, H. Wang, and W. Chen, "Unknown constrained mechanisms operation based on dynamic hybrid compliance control," in *IEEE International Conference on Robotics and Biomimetics*, dec. 2011, pp. 2366 – 2371.

[5] L. Peterson, D. Austin, and D. Kragic, "High-level control of a mobile manipulator for door opening," in *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2000, pp. 2333 –2338.

[6] A. Jain and C. Kemp, "Pulling open novel doors and drawers with equilibrium point control," in *9th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2009*, dec. 2009, pp. 498 –505.

[7] ——, “Pulling open doors and drawers: Coordinating an omni-directional base and a compliant arm with equilibrium point control,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2010, pp. 1807–1814.

[8] M. Prats, S. Wieland, T. Asfour, A. del Pobil, and R. Dillmann, “Compliant interaction in household environments by the Armar-III humanoid robot,” in *8th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2008*, dec. 2008, pp. 475 –480.

[9] W. Chung, C. Rhee, Y. Shim, H. Lee, and S. Park, “Door-opening control of a service robot using the multifingered robot hand,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3975–3984, oct. 2009.

[10] J. Sturm, C. Stachniss, and W. Burgard, “A probabilistic framework for learning kinematic models of articulated objects,” *Journal of Artificial Intelligence Research*, vol. 41, pp. 477–526, 2011.

[11] A. Petrovskaya and A. Y. Ng, “Probabilistic mobile manipulation in dynamic environments with application to opening doors,” in *Proc. of the 20th International Joint Conference on Artifical Intelligence*, Hydrabad, India, January 2007, pp. 2178–2184.

[12] A. Schmid, N. Gorges, D. Goger, and H. Worn, “Opening a door with a humanoid robot using multi-sensory tactile feedback,” in *IEEE International Conference on Robotics and Automation (ICRA)*, may 2008, pp. 285 –291.

[13] C. Kessens, J. Rice, D. Smith, S. Biggs, and R. Garcia, “Utilizing compliance to manipulate doors with unmodeled constraints,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, oct. 2010, pp. 483 –489.

[14] C. Ott, B. Bäuml, C. Borst, and G. Hirzinger, “Employing cartesian impedance control for the opening of a door: A case study in mobile manipulation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop on mobile manipulators: Basic techniques, new trends & applications*, 2005.

[15] H. Arisumi, J.-R. Chardonnet, and K. Yokoi, “Whole-body motion of a humanoid robot for passing through a door - opening a door by impulsive force -,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, oct. 2009, pp. 428 –434.

[16] P. A. Ioannou and J. Sun, *Robust Adaptive Control*. Upper Saddle River, NJ:Prentice Hall, 1996.

**Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time**
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Real-Time Robot Trajectory Generation with Python*

Morten Lind[1], Lars Tingelstad[1] and Johannes Schrimpf[2]

Abstract— **Design and performance measurements of a framework for external real-time trajectory generation for industrial robots is presented. The framework is implemented entirely in Python. It serves as a proof of concept for performing real-time trajectory generation in Python, from a PC with connection to the motion controller in an industrial robot controller. Robotic applications requiring advanced, custom trajectory generation, and a high level of integration with sensors and other external systems, may benefit from the efficiency of Python in terms of reduced development time, lower code complexity, and a large amount of accessible software technologies.**

The presented framework, dubbed PyMoCo, supplies a set of simple trajectory generators, which are comparable to those found in contemporary industrial robot controllers. Designing and implementing new trajectory generators and integrating or extending the included trajectory generators is central to the design of PyMoCo. Laboratory applications involving real-time sensor- and vision-based robot control has demonstrated the usability of PyMoCo as a motion control framework and Python as a robotics application platform. For robotics applications with a control frequency not exceeding a couple of hundred Hz, computation deadlines no shorter than some couples of milliseconds and jitter tolerance at the order of a millisecond, PyMoCo may be considered a feasible and flexible framework for testing and prototype development.

I. Introduction

Robotic tasks of limited complexity such as simple positioning tasks, trajectory following or pick-and-place applications in well structured environments, are straightforward to develop and integrate in the application platform of the native robot controller using current commercial robot control software (*de Schutter, et al. (2007)* [1]).

If robots communicate or interact with other robots or systems, the implementation is most often based on vendor-specific proprietary protocols and with limited performance specifications that preclude online sensor-based control (*Decré (2010)* [2]). However, there is a strong market pull for more flexible and cost effective robotic systems which are able to integrate a multitude of sensors and operate in unstructured environments. An example of this is the increased use of industrial robots in small and medium-sized manufacturing enterprises, often characterized by a combination of low-volume, high variety, and custom-made goods (*EURON (2005)* [3]). In order to meet these requests from the industry, new methods for programming and system integration are needed.

Many research laboratories therefore attempt to circumvent the application platform of the native robot controller, which either precludes real-time interaction or does not offer an appropriate set of technologies for solving the pertinent problem, in order to directly interface the motion control level. The motion control level is described as the entity providing a real-time interface for addressing the joint configuration space of the robot arm at an intermediate-level frequency; in the range from 100 Hz to 1 kHz. The ability to address the motion control level from an external application platform may thus give full control of choosing hardware peripherals, programming software and control algorithms (*Decré (2011)* [2]). The motion control level is often referred to as *low-level control* in literature.

A. Related Work

Applications that utilize low-level interfaces, to the motion control level, are usually implemented with compiled, intermediate-level languages, such as C or C++, and deployed on some real-time operating system (OS) platform, such as VxWorks, QNX, OS-9 and RTAI+Linux. The obvious reasons for these choices are among requirements to hard real-time performance; efficiency of computation with short cycle times; and latency tolerance on the time scale of microseconds.

Dallefrate et al. (2005) [4] used RTAI+Linux to control the Mitsubishi PA10 robot at the motion control level in 1 kHz over Arcnet.

Kubus et al. (2010) [5] modified Stäubli controllers and gained external joint level position control rates of 10 kHz and 250 Hz from a QNX system on a standard PC.

Buys et al. (2011) [6] present a teleoperation setup using two KUKA Light-Weight Robots (LWR) coupled to a Willow Garage Personal Robot (PR2). The two KUKA LWR robots are controlled over the KUKA Fast Research Interface (FRI) (*Schreiber et al. (2010)* [7]) for the KUKA KRC2LR industrial controller from an external control unit running RTAI+Linux. The communication is based on the UDP protocol and has a configurable communication rate of up to 1 kHz. The application was integrated using the two component based robotic frameworks OROCOS (Open Robot Control Software) (*Bruyninckx (2001)* [8] and *Bruyninckx et al. (2003)* [9]) and ROS (Robot Operating System) (Quigley et al. (2009) [10]).

A contemporary overview of the directly available low-level accessibility in some industrial robot controllers can be

*The work presented has been financially supported by the The Research Council of Norway through the research programmes “SFI Norman”, “BIA Robust, industriell sømautomatisering” and “KMB Next Generation Robotics”.

[1]Department of Production and Quality Engineering, Norwegian University of Science and Technology, Trondheim, Norway

[2]Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway

e-addresses: `{morten.lind, lars.tingelstad, johannes.schrimpf} at ntnu.no`

found in *Kröger and Wahl (2010)* [11].

B. Motivation and Goals

The work underlying this paper is motivated by the desire for making quick prototype development of real-time, sensor-based robotics applications in a laboratory setting with industrial robots. Our main application domain is industrial manufacturing automation, and all laboratory projects involve industrial robots for various types of tasks, ranging from standard offline programmed robot control to sensor-based real-time trajectory generation.

The presented work started out as a simple need for experimenting with motion control interfacing, and developed into the robot control framework we call PyMoCo. When developing real-time robotic applications, there are many demanding issues involved. We aim at addressing two of these:

- Maintenance and knowledge of specialized real-time operating systems and platforms (hardware and software).
- Development of C/C++ applications on real-time enabled software frameworks or platforms.

The goals of the presented work were to establish a sufficiently stable real-time framework which is:

- based on a stock GNU/Linux kernel and a freely available operating system,
- and using a high-level scripted programming language in pure user-mode.

Obtaining these two goals may have driven our development away from supplying directly usable industrial solutions. On the other hand it has been the enabling factor for having many researchers as well as projects making progress in advanced sensor-based robot control applications.

The specific choices of using stock Real-Time Linux[1] kernels with the Debian/Ubuntu operating systems and Python as the programming language were well-considered in terms of previous experiences and expertise.

As will be demonstrated later, see Section III, there is not much effect on the performance from using the Real-Time Linux kernel compared to using a standard Linux kernel. The major concern towards real-time performance regards the Python run-time efficiency and the implementation of PyMoCo. While there exist a possibility, however remote, that Real-Time Linux may some day guarantee an upper bound to latency, the Python run-time system in its current form, and possibly far into the future, does not possess hard real-time quality.

The efficiency of using Python as a development language, and even as an end-target platform has been well known for some time (*van Rossum (1998)* [12]). Further, the general scientific computational performance of Python is well document by many papers and projects; see e.g. the comprehensive paper by *Cai et al. (2005)* [13].

It is the purpose of this paper to give an overview of PyMoCo at the design and architectural level and to convey an impression of its level of feasibility as a software technology for real-time trajectory generation in prototype development of sensor-based applications of industrial robots.

C. Paper Outline

The remainder of this paper is outlined as follows. An overview of PyMoCo is presented in Section II, performance test setup and results are presented and discussed in Section III, and general discussion and mention of further work is presented in Section IV.

II. PyMoCo Overview

PyMoCo is a free and open source[2] software framework implemented entirely in Python, using the efficient NumPy[3] library for numerical computations. This section gives an overview of the architectural structure of PyMoCo.

The development of PyMoCo has been proceeding over the past five years and by now amount to some 4500 lines of Python source code[4]. It includes back-ends to two different robot types: A software-modified Universal Robots[5] controller and hardware-modified Nachi Robotics AX10 and AX20 controllers.

A. Applications

Though PyMoCo is a work in progress it has played a central role in many manufacturing automation prototype projects at our research laboratories.

The dual robot, real-time sensor-based sewing cell described by *Schrimpf et al. (2012)* [14] has a setup that uses PyMoCo trajectory generators.

Lind (2012) [15] used PyMoCo in the development of a joint offset calibration method for industrial robots.

Tingelstad et al. (2012) [16] used PyMoco for a tight tolerance compliant assembly task of critical aero engine components.

Schrimpf et al. (2011) [17] used PyMoCo for a real-time sensor-based control system with multiple sensors in a line-following application.

Lind and Skavhaug (2011) [18] used PyMoCo's ToolLinearController trajectory generator intensively for a real-time emulated production system setup involving several robots.

B. Architecture and Design

The PyMoCo run-time provides three core interfaces to the trajectory generation and application level systems. These are described in the following.

1) RobotDefinition Interface: is a placeholder for all static information about the robot in use. It provides such information as static link transforms; joint transform parameters; translators between different joint spaces: actuator, encoder, and serial; the home pose of the robot; and it is a factory for a set of joint transform function objects for the robot.

[1]`https://rt.wiki.kernel.org/`

[2]PyMoCo can be branched from Launchpad: `https://launchpad.net/pymoco`

[3]`http://numpy.org/`

[4]Measured using David A. Wheeler's 'SLOCCount' `http://www.dwheeler.com/sloccount/`.

[5]`http://www.universal-robots.com/`

2) FrameComputer Object: is the computational entity for all kinematics computation. It is currently a single, unspecialized class for unified kinematics computation for all robot structures. For joint transform objects and static link transforms, it relies on information retrieved from the RobotDefinition interface at construction time.

3) RobotFacade Interface: is the main interface covering the robot specific backend subsystem. Ultimately, in the backend subsystem, there is a connection to the motion controller entity of the operating robot. At any time, some trajectory generator must be answering real-time requests propagated from the robot motion controller through the robot facade subsystem.

For illustrating the relationships of entities in setups for real-time trajectory generation and robot application control involving PyMoCo, two UML object diagrams are shown and described in the following.

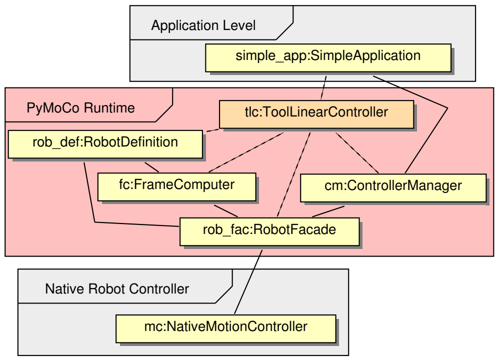


Fig. 1. UML object diagram giving an overview of a simple PyMoCo application, utilizing built-in trajectory generators managed by a ControllerManager object from PyMoCo.

The most simple runtime setup using PyMoCo for robot control, illustrated by the UML object diagram in Fig. 1, uses an object of the ControllerManager class, included with PyMoCo. The ControllerManager class is managing the switch of trajectory generators at the request of the application code, ensuring that the switch will not skip a control cycle request from the motion control level.

In the diagram in Fig. 1 weak or temporary associations are represented by dashed lines and more persistent object associations are illustrated by solid lines. The trajectory generator is exemplified by an object of the ToolLinearController class. It uses the core PyMoCo entities and provides its operational interface to a simple application; which is not specified by PyMoCo. The simple application, developed and provided by the user, thus only has to interface with the ControllerManager object and the trajectory generator objects that it requests from the controller manager.

Fig. 1 also indicates a layered structure, where the native robot controller containing the motion controller is lowest, the PyMoCo run-time system is in the middle, and the application level at the top. In a simple setup as the one illustrated, PyMoCo may be considered more as a software service than a software framework, since the client system, i.e. the simple application, is cleanly separated from the PyMoCo code.

The specific set of trajectory generators that are managed by the controller manager are the ones supplied with PyMoCo, and they will be discussed shortly in Section II-C.

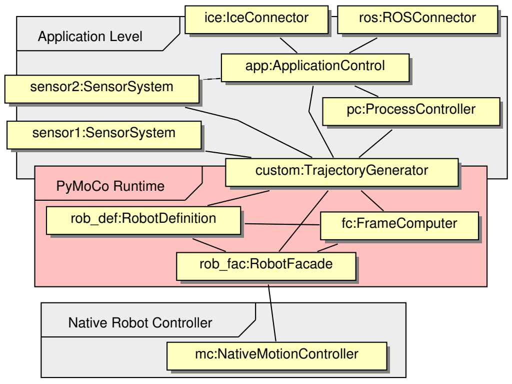


Fig. 2. Overview of an advanced PyMoCo application, utilizing the core PyMoCo objects and implementing custom trajectory generators with PyMoCo resources.

A more advanced, and realistic setup for sensor-based real-time trajectory generation, is illustrated in Fig. 2. It shows an application control at the application level which is strongly integrated with network communication systems, illustrated by connectors over ZeroC Ice™(*Henning (2004)* [19]) and ROS; process control; sensor systems which naturally connect externally; and with a custom trajectory generator. The custom trajectory generator is developed using the PyMoCo software framework resources and takes on the real-time obligations toward the pertinent robot motion controller through the robot facade.

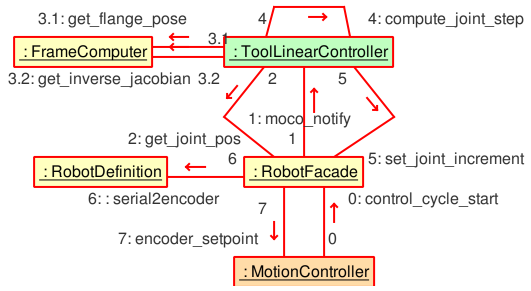


Fig. 3. The real-time cycle illustrated as a UML collaboration diagram among core PyMoCo entities, the robot motion controller, and a trajectory generator (of the class ToolLinearController).

The detailed mechanisms of the control cycle involving the core elements of PyMoCo may be perceived from the UML collaboration diagram in Fig. 3. The focus here is on the computational real-time cycle from the trajectory generation level and down, and hence the application logic and control is not included. The MotionController class is not a real class, but included for representing the motion controller in the native robot controller. The trajectory generator used for illustration here is, again, of the class ToolLinearController.

The control cycle is started by a notification from the motion controller to the robot facade in PyMoCo; typically containing a lot of status information such as encoder readings, velocities, etc. The robot facade propagates the notification internally to a subscribing trajectory generator, which then, using PyMoCo run-time facilities, computes a control step in response. This control step is returned in serial joint kinematics coordinates to the robot facade, which translates it to encoder values and then in turn responds to the motion controller.

C. Included Trajectory Generators

A (real-time) trajectory generator is an entity which ultimately carries the real-time responsibility of timely responding to the motion controller request for a new control-setpoint in joint space; or rather the joint encoder space. Neither the motion controller or the trajectory generators are core PyMoCo entities.

PyMoCo includes a set of simple trajectory generators. They cover the typical trajectory generators that are represented by motion commands in the application platforms of standard industrial robot controllers. None of the included trajectory generators implement any advanced strategies for dealing with arm configuration singularities, joint speed or acceleration violations, joint limits, or other types of circumstances that may lead the motion control system to fail. Self-motion, or internal, singularities are dealt with by using a configurable singular value cutoff in the inverse Jacobian computation; which is probably the simplest possible strategy.

The most common trajectory generators in standard robot controller are included: joint space linear motion, tool space linear motion, and real-time correction-responsive tool space linear motion. An additional two real-time responsive trajectory generators are included, which are rarely found in standard robot controllers, but immensely useful in real-time sensor-based robot control: tool space velocity motion and joint space velocity motion. The tool space velocity generator is the most frequently used in real-time sensor-based robot control applications at our laboratories.

III. Real-Time Performance

The high flexibility and versatility of Python as an application platform for robot control, and as the implementation language of PyMoCo alike, come at the cost of computational performance and real-time quality. The real-time performance of a PyMoCo-based application is thus crucial to investigate. It is the outcome of such an investigation which will clarify whether PyMoCo is usable and feasible, and, if at all, for which applications and robots.

This section presents results of an experimental setup based on the Universal Robots controller. The Universal Robots UR5 robot is used extensively in our laboratories, since it may be externally controlled and exhibits fairly low control delay and short motion response time; see *Lind et al. (2010)* [20].

Schrimpf et al. (2012) [21] compares three different setups for real-time trajectory generation; one of which is PyMoCo and the others based on OROCOS kinematics. Though their experiments are performed on one PC using local loop back networking, and thus do not measure the over-the-wire performance, the comparison is instructive. The purpose of the experiments presented in this section is different, in that it aims at making absolute, over-the-wire, realistic performance tests that are valid for PyMoCo-based trajectory generation applications.

A. Experiment Setup

The motion controller in the Universal Robots controller is interfaced at $125\,\mathrm{Hz}$, i.e. a control period of $8\,\mathrm{ms}$, and requires a response in $4\,\mathrm{ms}$. In the real controller, the native application platform and trajectory generator can be shut down, and a custom "router" application started. This router application listens for external connections over TCP, and mediates contact with the motion controller internally in the robot controller. The router application, representing the motion controller, can be emulated on an ordinary PC, the purpose of which it is to log the response times from a PyMoCo application running off another PC and connecting through a switch.

All hardware used is consumer grade and not of highest performance. Two PCs, both with an Intel i7 processor are used for performance measurements, connected through a standard $100\,\mathrm{Mbit\,s^{-1}}$ switch, and using the on-board Ethernet cards. The most important hardware to detail is the PC running the PyMoCo application. It is an Intel i7-860 processor running at $2.80\,\mathrm{GHz}$ with four cores and two threads per core.

Both PCs use the stock GNU/Debian Linux systems with Preempt-RT patched kernels of version 3.2.0-3-rt-686-pae; i.e. Real-Time Linux kernels. The most important software versions to mention are Python, 2.7.3rc2, and NumPy, 1.6.2-1. All software and kernels involved are taken from the official Debian testing repositories[6].

Starting from a standard Debian desktop installation, a checklist of simple tweaks to ensure the best possible real-time performance was followed:

1) Switch to single user mode. (`$ telinit 1`)
2) CPU frequency scaling should be set to "performance". (`$ cpufreq-set -c [0..7] -g performance`)
3) Disable garbage collection in the Python code for the real-time critical computations. (`gc.disable()`/`gc.enable()`)
4) Put the control process in a real-time scheduler queue. (`$ chrt 99 ...`)
5) Run the RT-critical processes from a remote login-shell. (`$ ssh ...`)
6) Boot the Real-Time Linux kernel.

All experiments were conducted at a length of $100\,000$ samples, which at $125\,\mathrm{Hz}$ amounts to about $13\,\mathrm{min}$ running time.

[6]`http://ftp.debian.org/debian/dists/testing/`

B. Best Condition Performances

The most important experiments were to measure the inherent response time of the PyMoCo run-time, by using the ZeroVelocityController, and to performance test the two most useful of the included trajectory generators: ToolVelocityController and ToolLinearController. All experiments were executed under the best obtainable real-time conditions, as per the check list in Section III-A.

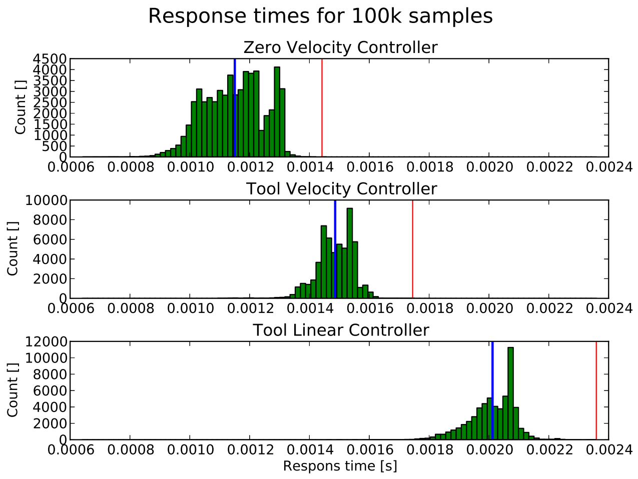


Fig. 4. Response time distribution for three different controllers. Average response time is marked by a vertical blue line and worst-case is marked by a vertical red line.

The results are visually observable from Fig. 4. Statistical summaries of the response time samples are shown in Table I.

Trajectory Generator	Worst [s]	Average [s]	Std. dev. [s]
ZeroVelocityController	0.00144	0.00115	0.00010
ToolVelocityController	0.00174	0.00149	0.00006
ToolLinearController	0.00236	0.00201	0.00008

TABLE I
STATISTICS OF MEASUREMENTS UNDER BEST REAL-TIME CONDITIONS.

These results show that the ToolLinearController is computationally much heavier than the ToolVelocityController; which was expected since it performs various checks along its path to control bounded acceleration ramp-up and ramp-down of the velocity. More importantly, the results also show that both of the usable controllers are well within the $4\,\mathrm{ms}$ response time required by the Universal Robots motion controller. The inherent response time of PyMoCo indicated by the worst-case response time of the ZeroVelocityController gives the impression of the availability of control computation time for any useful trajectory generator. In case of a required $4\,\mathrm{ms}$ response time, there is of the order of $2.5\,\mathrm{ms}$ time available for any trajectory generator in each control cycle.

C. DH-Kinematics Performance

PyMoCo has a native kinematics formulation which is flexible for specifying separately static link transforms and joint transform functions. However, a DH formulation of the kinematics is also supported, reducing the number of matrix multiplications in the forward kinematics computation. The DH formulation was used in one run with the ToolVelocityController under the same conditions as the ones used in Table I. The comparable statistical results are seen in Table II

Kinematics	Worst [s]	Average [s]	Std. dev. [s]
PyMoCo	0.00174	0.00149	0.00006
DH	0.00180	0.00151	0.00007

TABLE II
MEASUREMENT OF KINEMATICS IN DH FORMULATION.

It turned out that the DH formulation, contrary to the expected, was slightly inferior to the standard formulation in PyMoCo. This can be traced to NumPy being relatively inefficient in assigning matrix element compared to multiplying matrices.

D. System Tweak Performance Effects

The last experiments addressed the effects of individual omission of the various real-time enhancement tweaks, shortlisted in Section III-A. Results are given in Table III.

Tweak	Worst [s]	Average [s]	Std. dev. [s]
All tweaks	0.00174	0.00149	0.00006
- Single user	0.00249	0.00206	0.00009
- CPU freq. sched.	0.00265	0.00212	0.00007
- Disable GC	0.00223	0.00156	0.00009
- RT scheduling	0.00203	0.00155	0.00005
- RT kernel	0.00183	0.00125	0.00005

TABLE III
EFFECT OF VARIOUS TWEAKS ON REAL-TIME PERFORMANCE.

It is observed from the table that all tweaks have significant effect on the worst-case performance. The lower average and higher worst-case response times of the standard kernel are natural, since the low-level real-time enhancements in the real-time kernel sacrifice some computational efficiency for gaining lower worst-case latency. The fact that the performance difference between a standard and a real-time kernel is so low is evidence of the flow of the real-time patches into the mainline Linux kernel over the recent years.

IV. DISCUSSION AND FURTHER WORK

This paper has presented an overview of the structure of PyMoCo, a flexible, Python-based software framework for trajectory generation and motion controller interfacing.

Various real-time performance experiments for assessing its usability have been conveyed and the results have been presented and discussed. Under the presented experiment conditions, in terms of hardware, software, and system setup, it can be inferred that PyMoCo may be a usable software technology for trajectory generation in robot control applications where the over-the-wire response time limit is no lower than about some $3\,\mathrm{ms}$.

The main contribution of PyMoCo is to provide users with a very flexible framework for building real-time sensor-based robot control applications at the laboratory prototyping stage. Many laboratory prototyping projects have already utilized PyMoCo, and it is considered good for learning and fast prototyping. However, being tied to the Python language and the Python run-time platform, it has no outlook of becoming industrially real-time reliable.

The computational performance of contemporary CPUs together with the current implementation and design of PyMoCo is the limiting factor for its use in various setups. For instance, it is currently precluded that a KUKA LWR could be controlled by a PyMoCo-based application over FRI with maximum control rate. However, with CPU performance increasing over time, such setups may be achievable for PyMoCo in a not too distant future.

Notwithstanding the automatic performance gains of future CPUs, there are a whole range of possibilities for increasing the inherent performance of a pure Python application. These range from downright porting of functional code to C/C++ extension modules, whereby some flexibility may be lost; over Cython (*Behnel et al. (2011)* [22]) for automated translation and compilation of computationally critical code blocks; with PyPy, a very fast re-implementation of the Python run-time; to simply using more optimal and specialized technologies within PyMoCo, e.g. integrating PyKDL for kinematics computations as demonstrated by *Schrimpf et al. (2012)* [21].

Among useful and functional features that will be addressed in the future work with PyMoCo are facilities for trajectory blending. The methods described by *Lloyd and Hayward (1993)* [23] and *Volpe (1993)* [24] are under consideration.

References

[1] J. D. Schutter, T. D. Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, May 2007.

[2] W. Decré, "Optimization-Based Robot Programming with Application to Human-Robot Interaction," Ph.D. dissertation, Katholieke Universiteit Leuven, 2011.

[3] Euron, "Sectorial Report on Industrial Robot Automation," European Robotics Network, Tech. Rep., July 2005, http://www.euron.org/miscdocs/docs/euron2/year2/dr-14-1-industry.pdf.

[4] D. Dallefrate, D. Colombo, and L. M. Tosatti, "Development of robot controllers based on PC hardware and open source software," in *Seventh Real-Time Linux Workshop*, Nov. 2005. [Online]. Available: http://www.realtimelinuxfoundation.org/events/rtlws-2005/ws.html

[5] D. Kubus, A. Sommerkorn, T. Kröger, J. Maaß, and F. M. Wahl, "Low-level control of robot manipulators: Distributed open real-time control architectures for stäubli rx and tx manipulators," in *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, D. Kubus, K. Nilsson, and R. Johansson, Eds. Technical University of Braunschweig, 2010, pp. 38–45. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010

[6] K. Buys, S. Bellens, W. Decre, R. Smits, E. Scioni, T. D. Laet, J. D. Schutter, and H. Bruyninckx, "Haptic coupling with augmented feedback between two KUKA Light-Weight Robots and the PR2 robot arms," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, Sept. 2011, pp. 3031–3038.

[7] G. Schreiber, A. Stemmer, and R. Bischoff, "The Fast Research Interface for the KUKA Lightweight Robot," in *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, D. Kubus, K. Nilsson, and R. Johansson, Eds. Technical University of Braunschweig, 2010, pp. 15–21. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010

[8] H. Bruyninckx, "Open Robot Control Software: the OROCOS project," in *International Conference on Robotics and Automation*, vol. 3. IEEE, 2001, pp. 2523–2528.

[9] H. Bruyninckx, P. Soetens, and B. Koninckx, "The Real-Time Motion Control Core of The Orocos Project," in *International Conference on Robotics and Automation*, vol. 2. IEEE, Sept. 2003, pp. 2766–2771.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009. [Online]. Available: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

[11] T. Kröger and F. M. Wahl, "Low-level control of robot manipulators: A brief survey on sensor-guided control and on-line trajectory generation," in *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, D. Kubus, K. Nilsson, and R. Johansson, Eds. Technical University of Braunschweig, 2010, pp. 46–53. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010

[12] G. v. Rossum, "Glue It All Together With Python," in *Workshop on Compositional Software Architectures*, C. Thompson, Ed. Object Services and Consulting, Inc., Feb. 1998. [Online]. Available: http://www.objs.com/workshops/ws9801/papers/paper070.html

[13] X. Cai, H. P. Langtangen, and H. Moe, "On the performance of the Python programming language for serial and parallel scientific computations," *Scientific Programming*, vol. 13, no. 1, pp. 31–56, 2005. [Online]. Available: http://iospress.metapress.com/content/xawr0dx9xg61nb7q/

[14] J. Schrimpf, L. E. Wetterwald, and M. Lind, "Real-Time System Integration in a Multi-Robot Sewing Cell," in *International Conference on Intelligent Robots and Systems*. IEEE/RJS, Aug. 2012, accepted.

[15] M. Lind, "Automatic Robot Joint Offset Calibration," in *International Workshop of Advanced Manufacturing and Automation*, K. Wang, O. Strandhagen, R. Bjartnes, and D. Tu, Eds. Trondheim, Norway: Tapir Academic Press, June 2012.

[16] L. Tingelstad, A. Capellan, T. Thomessen, and T. K. Lien, "Multi-Robot Assembly of High-Performance Aerospace Components," in *IFAC Symposium on Robot Control*, 2012, accepted.

[17] J. Schrimpf, M. Lind, and G. Mathisen, "Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking," in *International Conference on Intelligent Robots and Systems*. IEEE/RJS, Sept. 2011, pp. 2861–2866.

[18] M. Lind and A. Skavhaug, "Using the blender game engine for real-time emulation of production devices," *International Journal of Production Research*, vol. 0, no. 0, pp. 1–17, 2011, online available, iFirst.

[19] M. Henning, "A New Approach To Object-Oriented Middleware," *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, Aug. 2004.

[20] M. Lind, J. Schrimpf, and T. Ulleberg, "Open Real-Time Robot Controller Framework," in *CIRP Conference on Assembly Technologies and Systems*, T. K. Lien, Ed. NO-7005, Trondheim, Norway: Tapir Academic Press, June 2010, pp. 13–18.

[21] J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, "Implementation Details of External Trajectory Generation for Industrial Robots," in *International Workshop of Advanced Manufacturing and Automation*, K. Wang, O. Strandhagen, R. Bjartnes, and D. Tu, Eds. Trondheim, Norway: Tapir Academic Press, June 2012.

[22] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The Best of Both Worlds," *Computing in Science Engineering*, vol. 13, no. 2, pp. 31 –39, Apr. 2011.

[23] J. Lloyd and V. Hayward, "Trajectory generation for sensor-driven and time-varying tasks," *International Journal of Robotics Research*, vol. 12, no. 4, p. 380, Aug. 1993.

[24] R. Volpe, "Task space velocity blending for real-time trajectory generation," in *International Conference on Robotics and Automation*, vol. 2. IEEE, May 1993, pp. 680–687.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

A holistic framework for planning, real-time control and model learning for high-speed ground vehicle navigation over rough 3D terrain.

Nima Keivan[1], Steven Lovegrove[1] and Gabe Sibley[1]

Abstract— **This paper describes a local planning, control and learning framework enabling high-speed autonomous ground-vehicle traversal of rough 3D terrain replete with bumps, berms, banked-turns and even jumps. We propose an approach based on fast physical simulation and prediction, which we find offers numerous benefits: first, it takes advantage of the full expressiveness of the inherently non-linear, highly dynamic systems involved; second, it allows for the fusion of local planning and model-based feedback control all within a single framework; third, it allows vehicle model learning. The final and most important reason to use physical simulation as a unifying framework is that it works well in practice. The system is experimentally validated on a high speed nonholonomic remotely controlled vehicle on undulating terrain using a scanned 3D ground model and motion capture ground-truth data. Parameter reduction is achieved with the use of cubic curvature control primitives and a fast precomputed lookup table.**

I. INTRODUCTION

Recent developments in path planning and navigation have enabled operation in increasingly challenging environments. The use of motion primitives [9] and stochastic search methods such as RRT and RRT* [8] [6] have resulting in algorithms that successfully navigate complex obstacle fields even in higher order configuration space. A major advantage of these methods is that they can employ nonlinear dynamics models thereby enabling physically accurate planning in complex environments without approximation or linearization. However, this advantage comes at a performance price as stochastic methods invariably sample infeasible trajectories. Conversely, optimization based methods [4] employ effective initial guesses and numerical or analytical optimization techniques to rapidly converge on optimal paths. However due to the reliance on the accuracy of the initial guess, these methods are susceptible to failure or suboptimal performance depending on the quality of this guess.

The quality, optimality and methodology of the plans notwithstanding, their open loop performance in real robots is inevitably impaired by the existence of imperfections or extraneous inputs that may not have been included in the dynamics model. Therefore for real-life applications, some form of closed loop control is desired. Moreover, both the planner and control systems rely on an accurate model in order to properly control and plan for the robot. Due to the difficulty of obtaining accurate model parameters, it is desirable to learn model parameters by observing the response of the robot to control inputs. Recent developments in Model Predictive Control(MPC) [3] and Learning-based Model Predictive Control (LBMPC) [2] [10] have strived to both implement model-based control schemes and to improve the underlying model parameters by observing the response of the system to inputs. The advantage that these schemes hold over more traditional control methods is twofold: the incorporation of increasingly complex models, and the ability to generate control policies over a predicted series of timesteps into the future. The latter offers clear advantages when controlling an infeasible trajectory or one that was made using an inaccurate model.

An alternative to model predictive control, traditional feedback systems use static and/or dynamic feedback of the state to determine the controls for the next time steps. Recent developments in this field have resulted in methods allowing the calculation of Lyapunov functions for nonlinear systems [5] and defining graphs of Lyapunov-stable region around states as in the case of LQR-Trees [11]. Generally speaking, these methods rely on the linearization of the state transfer function in order to analytically obtain control policies.

[1]N. Keivan, S. Lovegrove and G. Sibley are with the Department of Computer Science, George Washington University, Washington DC, nima|slovegrove|gsibley@papercept.net

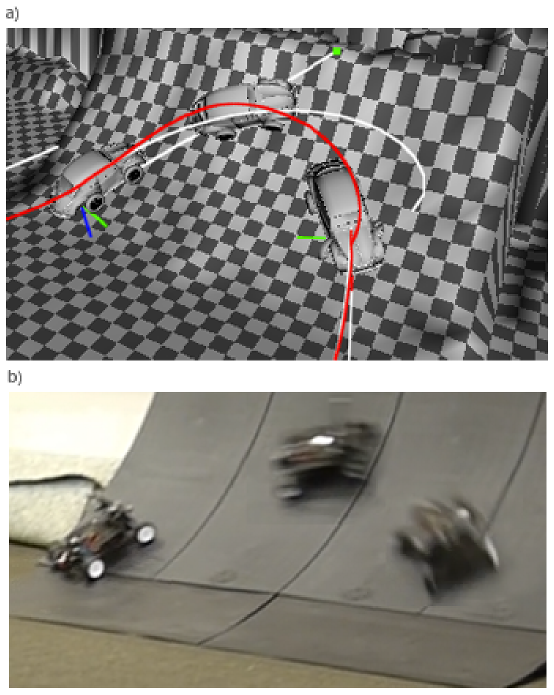


Fig. 1. a) Local plan (red curve) generated between two points on a 3D scanned quarter pipe ramp and the simulated vehicle tracking the plan in open-loop mode. b) Motion captured test vehicle performing the same manoeuvre.

Considering that the planning, MPC and model learning systems all utilise a model of the system, a unified system could be conceived to utilise the same model to perform all three tasks. The main contribution of this paper is such a system encompassing planning, control and online model learning using a unified, simulation-based model and operating in real time. We use a singular boundary value solver in all three cases in conjunction with cubic curvature polynomials for parameter reduction [7] . This allows accurate planning and control in full 3D environments and allows the learning of physical model parameters such as wheel radius, steering angle ratios and friction coefficients. Ground-truthed experimental evidence is also presented showcasing the results of the system planning over waypoints on undulating terrain and subsequently tracking the trajectory on a high speed nonholonomic robotic platform with on-line model learning.

II. METHODOLOGY

The different components of the planning and control system rely on a unified boundary value solver which produces a control law in order to navigate the robot between the start and goal 6DOF poses. For the purposes of this paper, we have implemented a parameter reduction and boundary value solver to plan for and control a nonholonomic remote control robot through high speed trajectories on undulating terrain. This formulation relies on a good initial guess for the steering and acceleration commands between two waypoints w_1, w_2 each parametrized as $[x, y, z, p, q, r, v]$ where v is the desired velocity with which the robot should reach the 6DOF coordinates of the waypoint. The optimization is facilitated with an initial guess utilising cubic curvatures for steering, and a linear velocity profile between waypoints.

A. Dynamics Model

The centrepiece of the system is the dynamics model. We use the Bullet Physics Engine [1] to simulate the dynamics of a vehicle with nonholonomic constraints on 3D terrain. A multithreaded framework allows the full use of modern multicore processes resulting in quick simulations for finite-difference based optimization. Traditionally, the state transfer function is defined as:

$$\check{x} = f(x, u, p)$$

Where x is the current state, u is the control input, and p defines the model parameters. In the case of the numerically integrated Bullet Physics model, the state transition is defined as:

$$x_{t+1} = F(x_t, u, p)$$

Where $F(x, u, p)$ encompasses the entirety of the dynamics of the vehicle and interaction with the terrain. In general terms, this function can be replaced with any simulation system resulting in an update in the state, given the control inputs, previous state and model parameters.

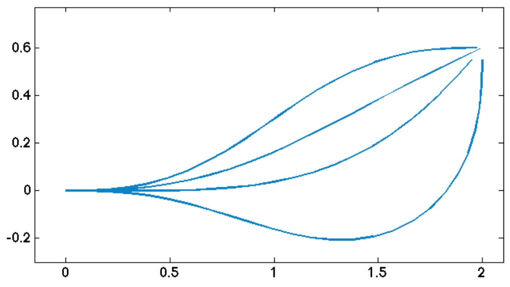


Fig. 2. Integration of cubic curvature polynomials in Cartesian space between $[0, 0, 0, 0]$ and $[2, 0.6, \pi, 0]$ with varying values of π

B. Parameter Reduction

The boundary value solver used relies on the reduction in control space dimensionality with the use of a control law. In the proposed system, we have employed cubic curvature polynomials [7] as a means to parameter reduction.The trajectory curvature is parametrized as a function of the travelled distance in the following form:

$$\kappa = a + bs + cs^2 + ds^3 \tag{1}$$

Where a is the starting curvature, $b, c,$ and d are the cubic polynomial coefficients and s is the distance travelled along the trajectory. Individual polynomials are constrained using the endpoints coordinates $[x, y, \theta, \kappa]$. To obtain the cubic parameters necessary to reach the desired endpoint, a precomputed lookup table is employed followed by a Gauss-Newton optimization using the analytical Jacobian of the polynomial. Figure 2 shows an example of cubic curvature polynomials integrated in 2D Cartesian space. However since the planner operates in 3D space, we project the curvature polynomial onto a 2D plane, with a normal which is linearly interpolated between the normals of the two waypoints as shown in Figure 3. This allows the 2D curvature to better estimate the control law that will guide the vehicle and resolves singularities from waypoints perpendicular to the ground plane.

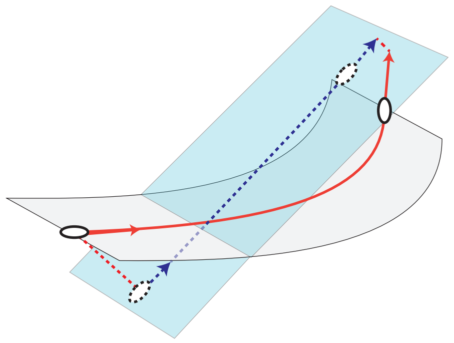


Fig. 3. The projected 2D trajectory (dotted blue) between two 3D waypoints on a curved manifold. This serves to better estimate the trajectory between the waypoints as well as eliminate singularities when projecting waypoints which are perpendicular to the ground plane.

C. Model Compensation

The linear velocity profile used between waypoints employs a constant acceleration model. However, due to the underlying physics-based vehicle model, this simple acceleration control law does not constitute a good initial guess. To improve upon this guess, several compensation factors are utilised to mitigate the extraneous influences introduced by the terrain and vehicle dynamics. Compensations are applied iteratively after each physics model update. This allows folding in detailed terrain information such as slope and also simulated vehicle parameters such as suspension force and extension. Furthermore, the underlying constant acceleration model remains valid once all other factors are compensated for.

1) Gravity Compensation: The constant acceleration model used between waypoints is by definition unable to account for terrain slope and undulation effects. We have implemented a simplified compensation model which accounts for the axial forces imparted by wheel interaction with inclined terrain (See Fig. 4a). The position of the wheels as well as the corresponding contact normal is obtained after each simulation step, and used to compensate the acceleration model for the following step.

2) Steering Compensation: Figure 4b shows the axial force component imparted as a result of the front wheel deflection during cornering. This force results in significant deceleration during tight turns and is compensated for in a similar fashion to gravity compensation at the end of each simulation step.

3) Friction Compensation: Friction compensation is undertaken iteratively similar to previously discussed factors. At each timestep, the friction forces on each wheel are calculated by the physics model. This is then used to offset the constant acceleration model accordingly. We have opted to use a simple friction model based on static/dynamic coefficients of friction, and the normal forces imparted on the springs. This information is readily available from the physics-model at each simulation step.

D. Boundary Value Solver

The boundary value optimization is performed by minimizing the trajectory cost C which we have defined as the 6 dimensional residual between the destination waypoint and the simulation endpoint. The optimization is performed by first solving a Gauss-Newton iteration with line search, and if the error norm is not reduced, a coordinate descent step is performed if possible. The Jacobian of the forward simulation is defined as:

$$J = \begin{pmatrix} \frac{\partial c_1}{\partial p_1} & \cdots & \frac{\partial c_1}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_n}{\partial p_1} & \cdots & \frac{\partial c_n}{\partial p_n} \end{pmatrix} \tag{2}$$

Where p_n is a control law parameter (such as a curvature polynomial coefficient) and c_n is a cost parameter. In the presented implementation, the cost is calculated as projected back onto the 2D plane of the cubic curvature polynomial

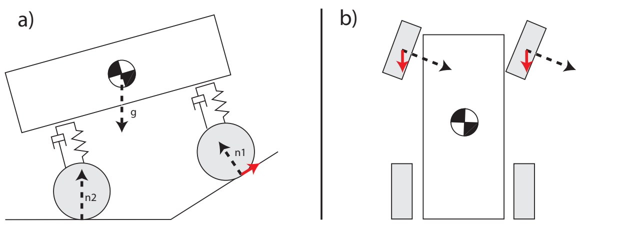


Fig. 4. a) Axial forces (in red) resulting from wheels on inclined terrain. b) axial forces (in red) resulting from front wheel steering deflection.

(See Fig. 3) and is parametrized as $[x, y, \theta, v]$. Each column $\frac{\partial c_1}{\partial p_j} \cdots \frac{\partial c_n}{\partial p_j}$ of J is calculated by pushing forward the dynamics model using a set of control parameters p with perturbations $\pm\epsilon$ along dimension j. This computation is accelerated by the use of a multithreaded forward physics model, solving for all dimensions of the Jacobian simultaneously. The Gauss-Newton delta (δp) is then calculated by Cholesky factorization as follows:

$$\begin{aligned} J^T J &\rightarrow R^T R \\ R^T y &= J^T b \\ R\delta p &= y \end{aligned}$$

Where b is the vector of residuals calculated by running the current parameters p and obtaining the endpoint error(s). The validity of the assumption of quadratic convergence made by this optimization is dependent on many factors including interactions with the terrain and the dynamics model. After obtaining the Gauss-Newton δp, we perform a multithreaded line-search step by pushing forward the physics model simultaneously with several scaled values of δp.

$$p_{n+1} = p_n + \lambda(\delta p)$$

Where $\lambda \leq 1$ the a scaling factor. If none of the scaled values of δp improve upon the error norm, we perform a coordinate descent if possible, by using the best norm obtained when calculating the Jacobian (Eq. 2) by finite-differences. The optimization ceases if the either the error norm is improved past a certain threshold, or if we are in a local minimum as indicated by the inability to perform a coordinate step to reduce the error norm.

E. Real-Time Control

In this section we present an MPC-like real-time control scheme based on fast replanning to account for inaccuracies and extraneous influences. Similar to MPC based control systems, our approach is formulated by constantly optimizing the trajectory ahead of the vehicle by solving new *control plans* which provide a viable control law from the vehicle's current position, to a point on the trajectory further ahead. As part of the holistic approach, we have used the same boundary value solver previously described to plan between waypoints, in creating the control plans. Due to the unified

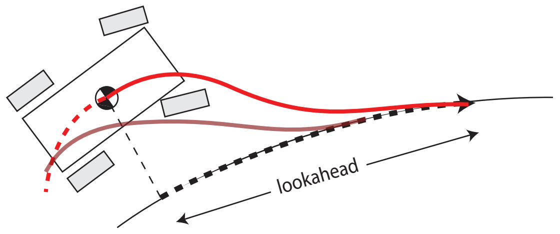


Fig. 5. Replanning-based control. The active control plan's (red) initial curvature matches that of the vehicle's current path curvature. Each control plan is optimized to plan to a point ahead on the trajectory based on a pre-define lookahead time.

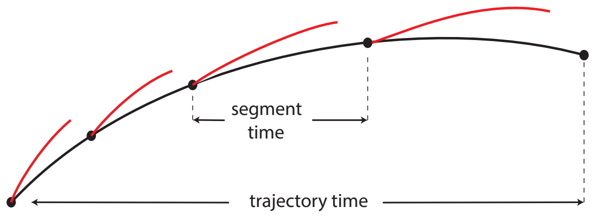


Fig. 6. The observed trajectory (black) shown with segments and their respective simulation results (red). The disparity between the simulation endpoint and the segment endpoint is the residual which is minimized in the optimization. If the residual is zero, the model perfectly matches the real vehicle's performance.

underlying steering control law, the control plans tend to converge back onto the original trajectory, thereby avoiding the pitfalls of follow-the-leader trajectory trackers which are prone to diverge if the target vehicle is set too far ahead, or oscillate if the target vehicle is too close. However, in order to achieve this behaviour, the starting curvature (denoted by the constant a in Eq. 1) of the control cubic curvatures has to match that of the vehicle's instantaneous path curvature as shown in Figure 5. This is also a requirement for smooth steering between control plans, as each will start with a curvature equivalent to that of the vehicle's current path. Consistent starting curvature is guaranteed by setting the constant a before the 2D optimization which solves Eq. 2. However, if the initial curvature required is too high, the 2D cubic required to solve the control plan might be infeasible. In these cases, the control system falls back to an initial curvature of zero to solve for a feasible control plan. Ideally, an alternative control law formulation would enable control plans that converged rapidly with any choice of initial curvature.

Furthermore, each control plan takes a small amount of time to optimize via the boundary value solver. During this time, the vehicle will be following the previous control plan. Our implementation includes a timestamp which is used to smoothly interpolate between plans as they become available. The control plans also constitute an any-time algorithm, as the optimization simply improves upon the quality of the initial guess with each iteration. The more time given to the algorithm, the higher the accuracy of the endpoint will be. It must be noted that each control plan is a valid set of controls that should ideally converge the vehicle back onto the trajectory in open-loop mode, given an accurate vehicle and terrain model. Therefore real-time control is established if the time taken to optimize tractable control plans is less than the lookahead time, allowing constant replanning without ever exceeding the bounds of a single control plan.

F. Online Model Learning

The quality of the control and planning provided by the aforementioned system relies significantly on the quality of the underlying model. Since a full physics-based model is used in the optimization, the number of parameters which could be adjusted rules out the possibility of manual tuning. We propose an optimization based learning system which is formulated almost identically to the control and planning systems, and which serves to tune select parameters in the model to match those of the real vehicle. The proposed methodology involves observing the vehicle and also the control commands given to it for a period of time, and optimizing the underlying physics model to replicate the observed behaviour, given the same control commands. We formulate the optimization by splitting the observed trajectory into segments as shown in Figure 6. Each segment is then simulated and a Jacobian formed as per Eq. 2 but where p_n is a model parameter which is changed by $\pm\epsilon$. Due to the nature of the optimization, we can obtain J^TJ and J^Tb for the trajectory directly as follows:

$$J^TJ = \sum_{i=0}^{n} J_i^T J_i$$

$$Jb = \sum_{i=0}^{n} J_i^T b_i$$

Where J_i is the Jacobian if the ith segment, n is the total number of segments, and b_i is the error vector of the ith segment which we have defined as the disparity between the observed trajectory and the final position of the simulated vehicle (See Fig. 6). We then apply the resulting δp to the model and repeat the process. As per the planning and control optimization formulations, the learning system implements a Gauss-Newton and line search stage which is followed by a coordinate descent stage if needed. The optimization ends when either a sufficiently small norm is obtained, or if a local minimum is detected. The learning system is implemented as an on-line algorithm allowing continued refinement of model parameters.

III. Results

The planner, control and learning systems were experimentally validated in a motion captured environment, using a terrain model which was 3D scanned using a Microsoft Kinect sensor combined with the motion capture system and a fusion algorithm. Due to the simulation-based model, any method of obtaining 3D terrain data could be used to simulate the dynamics. This includes real-time acquisition using cameras, laser scanners and/or dense tracking and mapping (DTAM) methods. In our experiments the waypoints were manual placed over the terrain in order to put the vehicle through desired manoeuvres including straight tracks, curves, steep inclines and jumps.

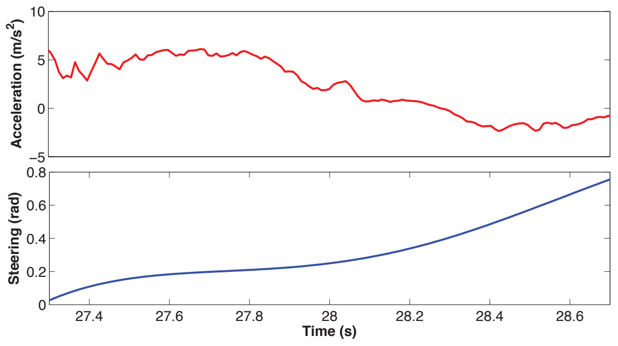


Fig. 7. Acceleration (red) and steering (blue) commands generated by the planner after optimization for the ramp manoeuvre depicted in Fig. 1. Note the gravity compensation during the uphill and downhill sections resulting in acceleration and braking forces.

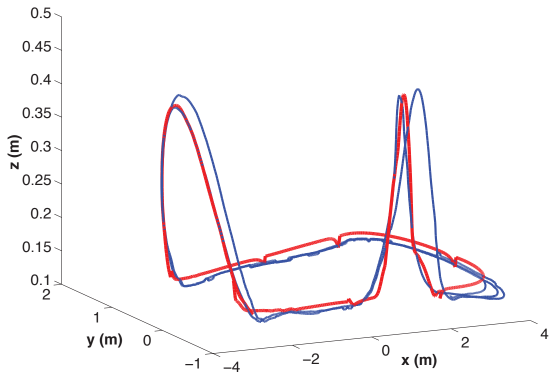


Fig. 8. Motion capture results (blue) of the control system running on a planned trajectory (red) with a small jump (center) and quarter-pipe ramp manoeuvre (left). The model was tuned using a combination of the learning system and manual adjustments.

A. Boundary Value Solver

Due to the quality of the initial guess, the boundary value solver successfully resolves feasible trajectories between waypoints. Furthermore the underlying model follows these trajectories precisely in open-loop control. However the choice of waypoints heavily influences the success or failure of the planner, as is expected. Figure 1 shows a plan generated between two waypoints on a scanned 3D model of a quarter-pipe ramp and Figure 7 shows the resulting acceleration and steering commands. Gravity compensation can be seen in Figure 7 and is vital to the feasibility of this plan as the vehicle needs to accelerate uphill and decelerate downhill in order to maintain velocity. The local planner can fail if the waypoints are poorly positioned or their velocities chosen improperly, for example if two waypoints are placed either side of a wall.

B. Real-Time Control

The real-time controller was tested on a trajectory including a small jump and sharp turn over a quarter-pipe ramp. Figure 8 shows the resulting vehicle path (blue) compared to the planned trajectory (red). It must be noted that the model used in this experiment was tuned using a combination of the learning system and manual adjustments. Manual adjustments were necessary as model-learning was not performed in high acceleration scenarios such as jumping, and small adjustments to the learned parameters was required to optimize the performance. The results obtained show that the vehicle is capable of accurately tracking the trajectory even in challenging manoeuvres, however as seen in Figure 8 , divergence can still be observed in the case of jumps where the steering is ineffective. This has the tendency to disrupt the real-time controller, as the boundary value Jacobian becomes invalid if changes in control input do not adequately perturb the endpoint of the control plan. This is the case when the vehicle is airborne.

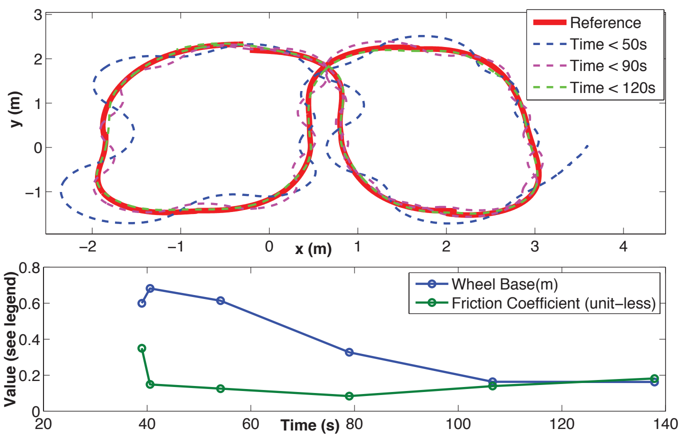


Fig. 9. Motion capture results of the control system (top graph) running on a planned figure-eight trajectory (red) while performing online model learning. The trajectories at various times show the improvement in tracking as the model parameters converge (bottom graph) to stable values.

C. Online Model Learning

The learning system was tested on a figure-eight trajectory over two model parameters: friction coefficient and wheel base. These model parameters, while not being all encompassing in their influence over the model have significant impact on the steering and acceleration response of the vehicle. They were chosen to test the learning system's response to deviations from the trajectory. While these parameters have underlying physical units, it is not expected that the values obtained from the model learning will reflect these values. This is due to the existence of unknown factors such as servo command coefficients, that will ultimately change the physical bases of the parameters. Nevertheless, the learned wheel-base parameter was observed to converge reliably to a value close to the actual vehicle wheelbase of $0.28m$. However, it is expected that learned values will bring the underlying model closer to the real-world vehicle, and therefore improve planning and control. Figure 9 shows the results of model-learning on a flat figure eight trajectory at different time intervals. It is evident that as the model parameters for wheel base and friction change, the adherence of the vehicle to the intended trajectory is significantly improved.

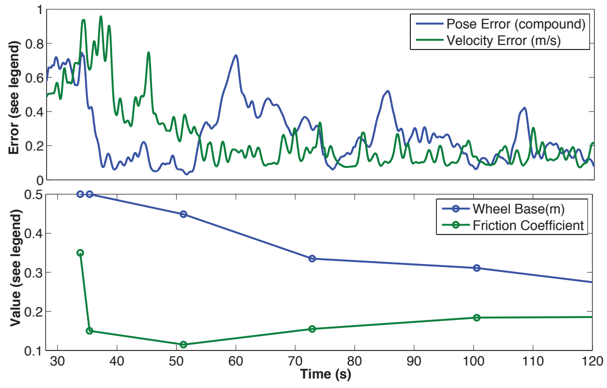


Fig. 10. Aggregated 6 DOF pose (blue) and velocity (green) error as a function of time performed on the figure-eight trajectory of Figure 9. The effect of the convergence of parameters can be seen on the repeated trajectory error pattern.

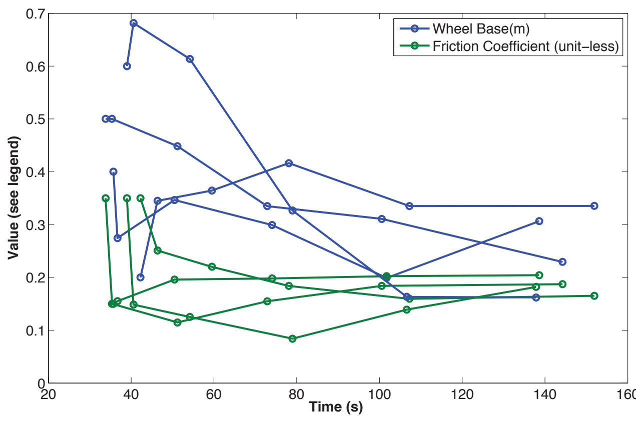


Fig. 11. Model parameter evolution during on-line learning for multiple experiments on the figure-eight trajectory of Figure 9.

Figure 10 shows the aggregated position and velocity error for a single run on the figure-eight trajectory, while performing model learning. It can be seen that the repeated error pattern around the trajectory monotonically decreases as the parameters converge. This also corresponds to a significant visible reduction in overshoot and improved tracking. Figure 11 shows the parameter convergence over multiple runs on the figure-eight trajectory and with different starting values for the parameters. It can be seen that the parameters, while not perfectly converging to the same point every time, tend to arrive at similar values.

IV. CONCLUSIONS

We have presented a holistic solution to local planning, real-time control and model learning which uses a unified simulation-based underlying physics model, folding in complex vehicle and terrain dynamics. The presented solution uses cubic curvature control laws to reduce the dimension of the control space while employing iterative compensation to deal with extraneous effects such as friction, terrain slope and steering deceleration. The solution was experimentally validated on a motion-captured vehicle and shown to execute manoeuvres on banked terrain and small jumps in real-time. However the real-time control system is still susceptible to disruption over jumps, and the model learning system has not been fully validated with a large number of parameters. Further experiments should also evaluate the ability to produce physically accurate model parameters. However, both systems have been shown to work well in practice and have produced good results in a challenging setting. Future work will be aimed towards testing the system on more challenging terrain, validating the model-learning with more parameters and implementing global planning solutions to place waypoints using the same holistic approach.

REFERENCES

[1] Bullet physics engine. `http://bulletphysics.org`. [Online; accessed July-2012].

[2] Anil Aswani, Patrick Bouffard, and Claire Tomlin. Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter. In *ACC 2012, to appear*, Montreal, Canada, June 2012.

[3] Thomas Howard, Colin Green, and Alonzo Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *Proceedings of the 7th International Conferences on Field and Service Robotics*, July 2009.

[4] Thomas M. Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *Int. J. Rob. Res.*, 26(2):141–166, February 2007.

[5] Tor A. Johansen and N-Trondheim Norway. Computation of lyapunov functions for smooth nonlinear systems using convex optimization. *Automatica*, 36:1617–1626, 1999.

[6] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[7] Alonzo Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. Pittsburgh, PA, June 2001.

[8] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[9] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[10] Neal Seegmiller, Forrest Rogers-Marcovitz, and Alonzo Kelly. Online calibration of vehicle powertrain and pose estimation parameters using integrated dynamics. In *IEEE International Conference on Robotics and Automation*, May 2012.

[11] R. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

**Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time**
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Plane Extraction and Map Building Using a Kinect Equipped Mobile Robot

Ahmad Kamal Nasir, Christof Hille, Hubert Roth
Lehrstuhl für Regelungs- und Steuerungstechnik (RST), Department Elektrotechnik und Informatik,
Fakultät IV Universität Siegen, 57068, Siegen Germany, (email: {ahmad.nasir, christof.hille, hubert.roth}@uni-siegen.de)

***Abstract*— This research work describes a method to create a geometric feature (3D plane) based map using a differential drive mobile robot in an indoor environment. Two algorithms namely Hough Transformation and Random Sample Consensus are used separately to extract multiple 3D planes from the point cloud data and the results are compared. An Octree based data structure is used to create and store the generated planner map of the environment. Furthermore a simple error analysis on the estimated parameters of the plane from both RANSAC and Hough Transformation algorithm in a test environment is presented.**

I. Introduction

3D Map building is fundamental to the autonomous navigation of the mobile robots in real world environment. Furthermore it could help mobile robots to reason about environment. State of the art mobile robots use 3D range scanning devices such as laser scanner, time of flight cameras, stereo cameras and RGB-D [21] cameras to sense the spatial environment and construct the map from acquired point clouds. Traditional computer vision solutions to construct 3D maps from multi-view videos or related images are computational resource demanding and time consuming. Geometric features such as lines and planes are prevalent into the manmade environments such as offices and factory floors. Mobile robots can use such geometric features to construct a map for collision free autonomous navigation and localization in such environments.

In this research work we demonstrate building a 3D geometric map of an office environment by using a ground mobile robot equipped with a Microsoft Kinect camera. It is an inexpensive camera which provides a color image stream and a depth image stream in an indoor environment in real time which can be very useful for dense 3D color mapping in cluttered indoor environments. Despite of the impressive acquisition rate the raw data is unsuitable for navigation and real-time 3D mapping because of the enormous amount of the data to be processed. Therefore, geometric features such as planes are extracted from the raw 3D point clouds.

To create the model of the environment several scans have to be fused. The fusing process is easy if the position of the scanner is known otherwise scan registrations have to be performed to estimate the pose of the scanner. In this research work we have not concentrated on the scan registration and a basic analysis of Kinect range measurement error is performed, but have not used to correct the measurement because of small error. We have also assumed that the mobile robot has been already localized thus an accurate mobile robot pose is available for mapping. For detailed Kinect sensor range measurement error model one could refer [22], [8].

A. Related Work

Various research works [1], [2], [3], [4] have been done until now to extract the 3D planes from the point cloud data acquired from different range sensor devices and build the 3D map of the environment. Asad [4] has proposed a mapping system for mobile robots which used height maps created from range images for path planning. Pathak [1] proposed a method for 3D mapping by a mobile robot, furthermore, his proposed method utilizes the uncertainty of the plane parameters to compute the uncertainty in the pose computed by scan registration. Weingarten et al. [3] proposed a method for plane fitting for laser range scanner data and fuses matching planes together to find a compact 3D model. Andreasson et al. [5] uses an approach which fuses both color and range information to detect 3D planes.

Apart from various mapping algorithms for mobile robots different sensors have also been used in combination with mapping algorithms to map 3D environments. Such sensors include laser scanners [19], stereo vision and monocular cameras [11] and time of flight camera [20]. 2D laser scanners are limited in use for mapping environments which contains simple geometric shapes; furthermore the obstacles which are above or below the scanned planes cannot be detected e.g. downward stairs. Where the stereo systems are dependent on lighting conditions and cannot detect planes in homogenous regions. Kinect sensor has brought acquiring colored 3D point clouds cheaper and quicker which in the past require expensive time of flight cameras. Furthermore, to acquire colored point clouds the system consisting of time of flight camera and image camera must be setup and calibrated. But Kinect combines the 3D range finding capability and the color information.

Recently most of the research work which uses Kinect camera [6], [7], [8] has focused on extracted plane segmentation because of the sparsity, measurement range limitation and occlusion of the measurements. These research works have used the color image to complement the range limitation and sparsity of the depth measurement. The intensity information can help in segmentation of the 3D point cloud data by detecting edges in the intensity images corresponding to the area of interest in the 3D point cloud.

A common approach for mapping is to align point clouds by finding rotation and translation between consecutive 3D scans [13]. Henry [12] maps the environment using ICP and SIFT features. There exist numbers of other methods which extract the 3D planes from the raw point clouds. Borrmann [15] uses the Hough transform to extract the 3D plane from the raw point clouds. Triebel [16] uses expectation maximization, Gallo [17] used RANSAC to extract the planes and Pathek [18] used the split and merge techniques to detect the planes.

Our focus in this research work is to build a framework for 3D mobile robot mapping which can be used in real time for SLAM, obstacle avoidance and path planning. Semantic mapping [14] can be applied as a post processing step to group the related geometric features in the map. The paper is organized as follows; section II describes the methodology of our research work in details. Section III describes the implementation of plane extraction algorithms and mapping. Section IV discusses the result obtained from the experiment. Section V concludes our work.

II. Methodology

This research work uses the plane detection algorithms to detect the planes from the raw Kinect data and registers them using octree data structure. Since Kinect sensor acquires enormous amounts of data, 9.2 million 3D points in one sec, it is challenging to process the data in real time because of the limited amount of computation resources available on mobile robots, furthermore, raw 3D point clouds from Kinect sensor are not directly useable. Some processing is required to reduce this amount of data to extract features information present in the raw 3D point cloud. The features could be point features, line features, color segmentations and shape detections. Extracting multiple geometric features from the range data is computationally demanding and directly related to the number of parameters required to represent the geometric model to be found in the raw point clouds. In our geometric mapping approach we have used 3D planes as geometric features because a plethora of 3D planes are available in structured environments. We have tested two algorithms namely RANSAC and Hough transformation to extract the 3D planes from the raw point cloud so that we can compare the performance of real-time geometric map building from the Kinect equipped ground mobile robot

A. Data Association

Octrees are data structures which can be used to partition the three dimensional space in an efficient way. The base octree is represented by a cube or a cuboid and is called root node. Its volume can be further discretized into eight new octants, which partition the space of their parent node. Depending on the needs of the application the depth of nodes can vary. Fig. 1 depicts an octree with a root node, a node and one leaf node, where data can be stored.

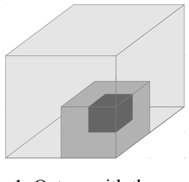


Figure 1. Octree with three nodes.

The main advantage of this approach is that there is no need to create child nodes, where there is no data for them and it's always possible to create a root node, if one exceeds the space given by the actual octree. Octrees offer a great flexibility to store and access 3D data. In most approaches each node represents a voxel in 3D which could be further discretized depending on the scan information during storing or the desired accuracy during rendering as described in [25], [26] and [31]. Once a point cloud is stored into an octree, the partitioning of the space can also be used to limit the data points to extract geometric features and therefore to help the extraction algorithms like in [27]. The data to be stored in the nodes are of course up to the user. In this research work planes in the environment are identified by their orientation and their center point. Each center point is located in one leaf node of the octree. In our octree discretization the root node has an edge length of 40 m and the leaf nodes have an edge length of 0.625 m. The idea to use the octree data structure is to simplify data association for detected planes. One leaf node is supposed to store only one plane. If a second plane should be registered into a node which already contains a plane, it is possible to decide which plane is the true plane. Since in this contribution the position of the plane greatly depends on the odometry of the mobile robot, which is not very accurate after a few meters of movement, it is almost impossible to decide which plane is true. Therefore each plane which is registered in a leaf node overwrites the existing one, in case there is one.

Uniquely registered planes in the octree can be easily rendered into a 3D map. This is a simple but efficient way to reduce the amount of data in the map. Another advantage of the octree structure is the possibility to implement efficient path planning and collision avoidance algorithms. Jung [32] presents one of the first approaches which use a three dimensional octree map of voxel information to plan a path for a manipulator robot. Kazakov et al. [28] propose an octree based map for large scale environments to simplify the problems of path planning and collision detection for mobile robots. In their experiments they successfully show that their suggested path planning algorithms are able to suggest a path and to correct it, if necessary.

B. Plane Extraction

1) Hough Transform

Hough transform is a well known algorithm in computer vision society to detect multiple models in the data compared to RANSAC which in its basic form assumes there is a single model present in the data. It can detect lines, planes, spheres and other parameterizable geometric objects in the input data. In spite of the robustness of the method against noisy data one drawback of this algorithm is its high computational requirement therefore many variations of the Hough transform exists to detect the desired model parameters. Apart from standard Hough transform other variations which exists are, probabilistic Hough transform, random Hough transform, adaptive probabilistic Hough transform and progressive probabilistic Hough transform. The plane equation in Hesse normal form can be defined by a point p on the plane with normal vector n to the plane which is at a distance ρ from the origin, which is collinear to normal vector as shown in fig. 4. The normal vector or ρ makes an angle θ with the z-axis and its projection in the x-y plane makes an angle φ with the x-axis. Therefore, the equation of the plane can be defined as

$$p_x \cdot \cos(\theta) \cdot \sin(\varphi) + p_x \cdot \sin(\theta) \cdot \sin(\varphi) + p_x \cdot \cos(\varphi) = \rho. \quad (1)$$

The dimension of the Hough space is basically equal to the number of parameters of our model i.e. (θ, φ, ρ). Each plane in $\mathbb{R}^3$ corresponds to a point in the Hough space and each point in $\mathbb{R}^3$ corresponds to a surface in Hough space. This surface represents all the possible planes where the point could belong to. Therefore, the transformation of the points $p_i \subset P$ from $\mathbb{R}^3$ to Hough space will generate surfaces in Hough space. The intersection of three surfaces in Hough space results in a point in Hough space which corresponds to a plane in $\mathbb{R}^3$ on which the three points which generates the surface lies on. All points whose surfaces in Hough space intersect at a point correspond to the same plane in $\mathbb{R}^3$.

We have used a random Hough transform to detect the 3D planes in the raw 3D point clouds. Instead of generating surfaces for each point p_i in $\mathbb{R}^3$ into Hough space, which is very time consuming we used the fact that a plane corresponds to a single point in Hough space, therefore, it is very fast to compute a plane from three random points from a small circular region and transform the estimated plane to Hough space, this results into a significant faster algorithm for real time implementation. The pseudo code of the randomized Hough transform is as follows:

```
Do until DetectedPlanes < 8 and TotalPoints > MinPoints
  Randomly select $(P_1, P_2, P_3)$ from a random circular region
  Calculate plane from $(P_1, P_2, P_3)$
  Transform the Calculate plane from $\mathbb{R}^3$ to Hough space
  If local maxima is found in Hough space
    Delete points corresponding to plane from input points
    Calculate plane boundries
    Reset Hough space
  End if
End Do
```

Algorithm: Randomized Hough Transform Algorithm

The Hough discretization size of the Hough transform depends on the accuracy required and the available memory. For our implementation we have discretized the Hough space into 1cm for ρ from 1cm to 500cm, 1° for φ from -180° to 180° and 1° for θ from 0° to 180°. Using the above discretization the memory requirement for Hough space is found to be 125MB. We have found out that the predominant part of the time required by the randomized Hough transform is required to reset the Hough space, therefore the choice of discretization for plane parameters has been chosen based on the possible orientation of the planes in the input raw 3D point clouds.

2) *RANSAC*

A common approach to identify geometric objects in a scene of 3D points is the use of the RANSAC algorithm, which was introduced by Fischler [24]. It overcomes the weakness of a normal least square approach which can't distinguish between inliers and outliers in the measurement data. Instead of using all the available data to calculate the model parameters it checks for each data point if it should be considered as an inliers or an outlier to the model. Therefore in the end only inliers are used to derive the object parameters. In this paper the RANSAC algorithm is used to identify planes in clouds of 3D points. A basic RANSAC algorithm is used here. It takes in a first step three non-collinear, random points $P_1(x_1,y_1,z_1)$, $P_2(x_2,y_2,z_2)$ and $P_3(x_3,y_3,z_3)$ out of the point cloud and sets up a plane equation using the equation

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0. \quad (2)$$

In the second step it sorts the remaining points into inliers, which have a minimum distance to the plane, and outliers, which have a greater distance to the plane than a defined threshold. The point to plane distance threshold for RANSAC is chosen to be 1 cm because the surfaces in our test environment are very flat and also we have small errors in Kinect's range measurements. A distance between a 3D point and a plane is calculated by

$$d = x\,cos\alpha + y\,cos\beta + z\,cos\gamma + p, \quad (3)$$

where d is the distance from the point P(x,y,z) to the plane and cosα, cosβ, cosγ and p are the parameters of the plane in the hessian form. Steps one and two are repeated until a satisfying number of inliers are found or a maximum number of iterations are executed. The number of satisfying inliers is 50,000 and the maximum number of iterations are 500. The identified inliers are used to calculate the plane parameters using the hessian description of a plane which is given as:

$$x\,cos\alpha + y\,cos\beta + z\,cos\gamma + p = 0, \quad (4)$$

which can be rewritten as

$$\underline{AX} = 0, \quad (5)$$

where $\underline{A}$ represents a matrix containing the plane parameters in each row which solves the equation for the matrix $\underline{X}$ which contains the homogeneous coordinates of each point P in its columns.. This homogeneous equation must be solved by using least square techniques. The solution of the above mentioned over-determined system is found by using the Singular Value Decomposition (SVD) implementation of the library Sho [23].

The RANSAC algorithm is applied for a maximum of 8 iterations on the outlier point cloud from the previous iteration, but stops if the amount of remaining points drops below a threshold. This avoids that points could be part of different planes and helps the algorithm to find smaller planes in the point cloud. Much more complex RANSAC algorithms do exist like [27], [30] but since in this research work only planes are extracted and no spheres, cylinder or tori, this complexity is not needed.

Since the robot pose in the world coordinate system is known from the odometry, it is possible to register a detected plane in the world coordinate system and to construct a map.

III. Experiment

During all experiments a laptop, having an Intel® Core™2 Duo processor, running at 2.8 GHz, equipped with 8 GB RAM and running with a 64 Bit Linux, is used to acquire and to store all the sensor information needed for the different experiments at a rate of 5 Hz. The amount of Kinect sensor data is very huge, which makes storing it to a problem where traditional HHDs are a bottle neck. Therefore the Laptop is equipped with an SSD which allows data storing at very high data rates, which helps to acquire the data at defined time stamps. Later the offline data sets are processed to extract the needed data for the different experiments.

Kinect sensor provides a horizontal and vertical field of view of 58° and 44° respectively and the angular resolution of 0.08°. Furthermore, the device can be tilted ±30°. The working depth measurement range is between 0.8m and 5m. Kinect consumes about 250mA at 12V DC. It can acquire both depth stream and color stream of full VGA resolution (640x480) at 30Hz. Park [9] has found out also that the Kinect variance of measurements error for dark objects is smaller compared to the Hokuyo UBG-04LX-F01 laser scanner. Since the color and depth cameras in Kinect are factory calibrated therefore it is now an easy task to correspond the pixel information in both cameras. One drawback of the Kinect is, it works only in indoor environments without sunlight.

To validate the approach and to compare the different plane extraction algorithms presented in chapter II, three experiments are conducted.

A. Experiment I

In the first experiment the Kinect sensor is mounted on a KUKA R16 manipulator robot and looks on a cuboid which is used as a reference object. All of its planes are orthogonal to each other and in this experiment two planes are in the field of view of the Kinect. The robot is used to change the view on the scene in defined angles. Under all changed view conditions the algorithms should be able to detect an angle around 90° between both detected planes of the reference object. The robot pose configuration is shown in fig. 2. Both Hough transform and RANSAC are supposed to identify the angle between the two planes of the reference object which correspond to the sides of the object. All together four different four different views of the cuboid are taken with the Kinect sensor. The robot angles of the four points clouds are listed in table I.

TABLE I. CONFIGURATION OF THE ROBOT AXIS DURING EXPERIMENT

Robot Axis	View 1	View 2	View 3	View 4
A1 [°]	1.59	1.59	1.59	5.59
A2[°]	90	90	90	90
A3[°]	-90	-90	-90	-90
A4[°]	0	0	0	0
A5[°]	-50	-50	-50	-50
A6[°]	0	20	-20	0

B. Experiment II

Here the Kinect sensor is placed on rotary indexed table which is placed on a table against a flat wall. The distance between wall and Kinect's front wall facing side is measured to be 106 cm and the data is recorded. In a next step the Kinect is moved 50 cm further away from the wall and once more the data is recorded. From both positions we have calculated the standard deviation of the Kinect's measurement range error, which will be discussed in the next section.

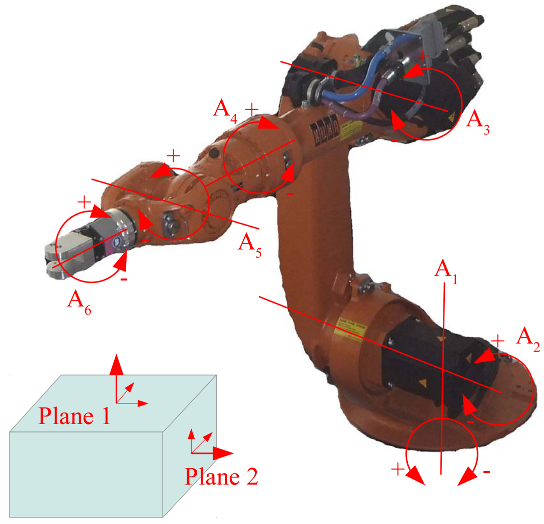


Figure 2. Coordinate system of the KUKA robot.

C. Experiment III

In the third experiment the mobile robot moves inside the corridor of a building within the campus. It starts in the lower right side of the corridor and drives through it counter clock wise as is it depicted in fig. 3. TOM3D has a differential drive wheel base with two wheels and one castor wheel. Each wheel is equipped with a DC-motor integrated with gearbox and quadrature encoder. From those two encoders the x and y position of the robot and its orientation in the global coordinate system is calculated.

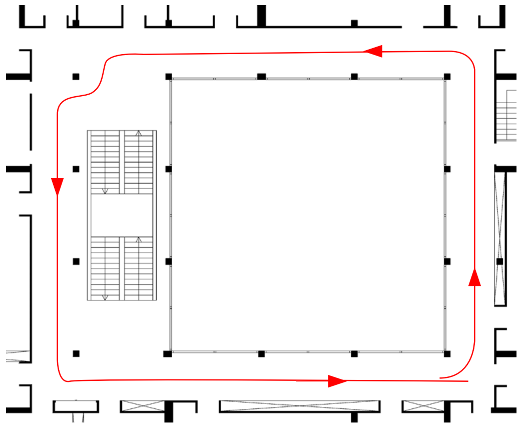


Figure 3. Driving path of TOM3D on the first floor in the H-F building of the campus at University of Siegen.

An electronic control board is designed for this robot based on a 16-bit Infineon MCU. The robot's firmware is designed in a way that all sensor information is pre-processed at robot's control unit and reports are sent to a PC via RS 232. On the top of the robot the Kinect sensor is mounted at an orientation of -90° around the robot's z-axis. Rotations around the x-axis and y-axis are set to zero, because the ground should not be in the field of view since the existence of the ground plane is preconditioned. TOM3D's and Kinect's coordinate systems are shown in Fig 4. It also shows that the normal vector of the plane is described by the spherical angles ϑ and φ where the shortest perpendicular distance from the origin to the plane is described by ρ. If the Kinect would look straight forward into the moving direction of the robot, most information would be lost since the Kinect has maximum measurement range of 4 m and the parts of the corridor have a length of 10-16 m. Therefore the Kinect sensor looks to the right side of the robot, where most geometric information of the surrounding is located. The global coordinate system is placed at robot power-up position.

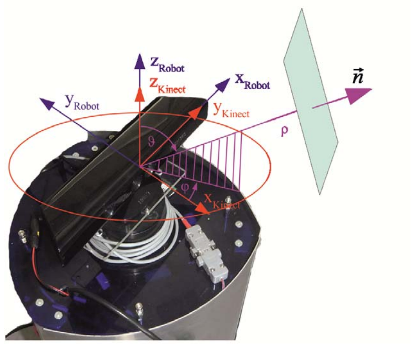


Figure 4. Coordinate system of Kinect and Robot; representing the plane's normal vector in spherical coordinates.

IV. Results

In this section the results of the experiments described in chapter 3 are discussed. It is also divided into three subchapters where the results of the different experiments will be presented and discussed.

A. *Experiment I*

From the first experiment the reference is the angle between the two planes on the cuboid which is 90°. Tab II shows the results on different views. Taken into account that the sides of the reference cube are not truly 90° due to production tolerances, we think that this result is acceptable for our approach.

B. *Experiment II*

In the second experiment the distance between a wall and the Kinect sensor is changed. We have measured the standard deviation of measurement error in rho for RANSAC and Hough transform to estimate the accuracy of the distance measurement from Kinect sensor. Tab III lists the results. First of all we can see the standard deviation of the Kinect measurement error increases with the distance, furthermore since the standard deviation calculated from both algorithms are almost the same therefore this in fact is the standard deviation of the Kinect's measurement irrespective of the algorithm used. Because of the small standard deviations of Kinect's measurement error we think it is still useful to map environment with this sensor.

TABLE II. Detected angles between two orthogonal planes in different views

Data Set	View 1	View 2	View 3	View 4
Angle between planes extracted by RANSAC	87.7°	88.3°	87.3°	87.1°
Angle between planes extracted by Hough Transform	87.1°	87.4°	87.2°	86.3°

TABLE III. X-coordinates of the plane base points at different positions

Standard deviation in ρ measurement error	Position 1 106 cm	Position 2 156 cm
RANSAC	3.9 mm	8.8 mm
Hough Transform	2.5 mm	7.4 mm

C. *Experiment III*

From the resulted 3D generated map by the RANSAC, fig. 6, and the Hough Transform, fig. 5, both produce a visually comparable result. The difference between the two resulted maps is in the top left corner, where the RANSAC fails to find the correct planes, because the corresponding point clouds contain a high number of invalid points. In term of the execution time RANSAC took on average 50 ms to extract the first plane, whereas the Hough Transform took an average of 170 ms to extract a plane. Since no loop closure was used the difference between start and end point in both maps was expected.

V. Conclusion

The approach described in this paper is a good base to expand the framework for data association and loop closure. We could extract the plane boundaries by clustering the 3D points based on their position and color information, therefore more accurate plane boundaries would be expected if the objects have uniform color and smaller data association uncertainty because the additional color knowledge is added to the system.

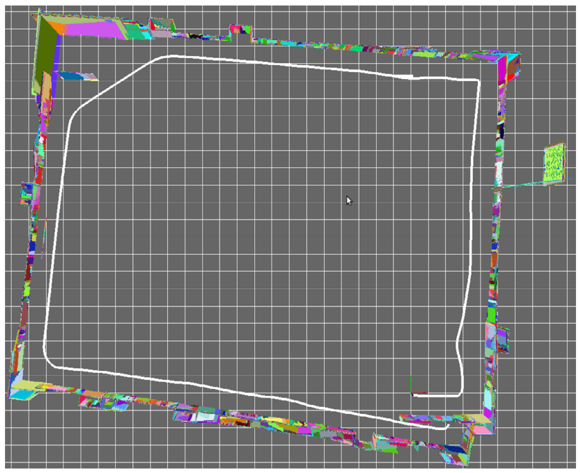


Figure 5. 3D Map generated using Hough Transform

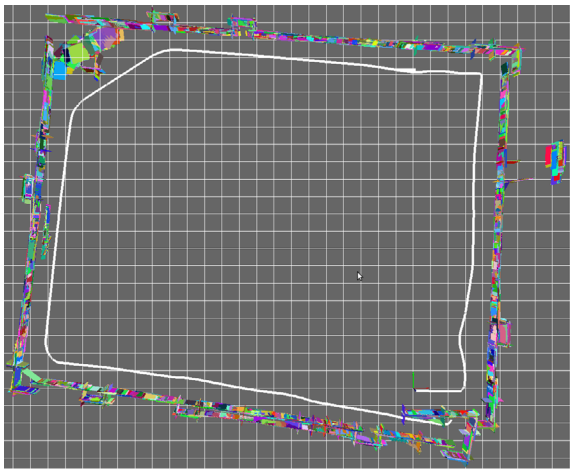


Figure 6. 3D Map generated using RANSAC

References

[1] Pathak, K., Vaskevicius, N., Poppinga, J., Pfingsthorn, M., Schwertfeger, S., Birk, A., "Fast 3D mapping by matching planes extracted from range sensor point-clouds," Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on , vol., no., pp.1150-1155, 10-15 Oct. 2009

[2] Bayro-Corrochano, E., Bernal-Marin, M., "Generalized hough transform and conformal geometric algebra to detect lines and planes for building 3D maps and robot navigation," Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on , vol., no., pp.810-815, 18-22 Oct. 2010

[3] Weingarten, J.W., Gruener, G., Siegwart, R., "Probabilistic plane fitting in 3D and an application to robotic mapping," Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on , vol.1, no., pp. 927- 932 Vol.1, 26 April-1 May 2004

[4] Asada, M., , "Map building for a mobile robot from sensory data," Systems, Man and Cybernetics, IEEE Transactions on , vol.20, no.6, pp.1326-1336, Nov/Dec 1990

[5] Andreasson, H., Triebel, R., Burgard, W., "Improving plane extraction from 3D data by fusing laser data and vision," Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on , vol., no., pp. 2656- 2661, 2-6 Aug. 2005

[6] D. Holz, S. Holzer, R. B. Rusu and S. Behnke, ”Real-Time Plane Segmentation using RGB-D Cameras”, in Proc. of IEEE RoboCup Symposium, 2011.

[7] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, ”Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Human Environments”, In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2009.

[8] Erdogan, C., Paluri, M., Dellaert, F., , "Planar Segmentation of RGBD Images Using Fast Linear Fitting and Markov Chain Monte Carlo," Computer and Robot Vision (CRV), 2012 Ninth Conference on , vol., no., pp.32-39, 28-30 May 2012

[9] Chan-Soo Park, Sung-Wan Kim, Doik Kim, Sang-Rok Oh, "Comparison of plane extraction performance using laser scanner and Kinect," Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on , vol., no., pp.153-155, 23-26 Nov. 2011

[10] Moghadam, P., Wijesoma, W.S., Dong Jun Feng, , "Improving path planning and mapping based on stereo vision and lidar," Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on , vol., no., pp.384-389, 17-20 Dec. 2008

[11] Biswas, J., Veloso, M., , "Depth camera based indoor mobile robot localization and navigation," Robotics and Automation (ICRA), 2012 IEEE International Conference on , vol., no., pp.1697-1702, 14-18 May 2012

[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In the 12th International Symposium on Experimental Robotics, 2010.

[13] Nuchter, A., Surmann, H., Lingemann, K., Hertzberg, J., Thrun, S., , "6D SLAM with an application in autonomous mine mapping," Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on , vol.2, no., pp. 1998- 2003 Vol.2, April 26-May 1, 2004

[14] Olufs, S., Vincze, M., , "Towards efficient semantic real time mapping of man-made environments using Microsoft's Kinect," Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on , vol., no., pp.130-137, 7-11 Dec. 2011

[15] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nuchter. The 3d hough transform for plane detection in point clouds - A review and a new accumulator design. Journal 3D Research, 2(2), March 2011

[16] R. Triebel, W. Burgard, and F. Dellaert. Using hierarchical em to extract planes from 3d range scans. In IEEE ICRA, 2005

[17] Orazio Gallo, Roberto Manduchi, and Abbas Rafii. CC-RANSAC: fitting planes in the presence of multiple surfaces in range data. Pattern Recognition Letters, 2011

[18] Kaustubh Pathak, Andreas Birk, Narunas Vaskevicius, and Jann Poppinga. Fast registration based on noisy planes with unknown correspondences for 3d mapping. IEEE Transactions on Robotics, 26(3):424–441, 2010

[19] M. Y. Yang and W. Forstner, “Plane detection in point cloud data,” Department of Photogrammetry Institute of Geodesy and Geoinformation, University of Bonn, Tech. Rep., 2010.

[20] N. Vaskevicius, A. Birk, and K. Pathak, “Fast plane detection and polygonalization in noisy 3d range images,” in IEEE International Conference on Intelligent Robots and Systems, 2008.

[21] http://www.microsoft.com/en-us/kinectforwindows/ (accessed: Aug. 10, 2012). Internet, 2012.

[22] Dube, D., Zell, A., , "Real-time plane extraction from depth images with the Randomized Hough Transform," Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on , vol., no., pp.1084-1091, 6-13 Nov. 2011

[23] Microsoft Sho Library. *http://msdn.microsoft.com/en-us/devlabs/gg585581* (accessed: Aug. 10, 2012). Internet, 2012.

[24] M. A. Fischler and R. C. Bolles, "Random Samples Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography", Communications of the ACM, Vol. 24, Num. 6, pp. 381-395

[25] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige and W. Burgard, "Hierarchies of Octrees for Efficient 3D Mapping", Proceedings of the IEEE/RSJ Internation Conference on Intelligent Robots and System (IROS), San Francisco, CA, USA, 2011

[26] K. M. Wurm, A. Hornung, Maren Bennewitz, C. Stachniss and W. Burgard, "OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems", Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation, 2010.

[27] R. Schnabel, R. Wahl and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection", Computer Graphics Forum (June 2007), 26:2(214-226)

[28] K. Kazakov, V. Semenov and V. Zolotov, "Topological Mapping Complex 3D Environment Using Occupancy Octrees", The 21st International Conference on Computer Graphics and Vision GraphiCon’2011, September 26–30, 2011, Moscow, Russia

[29] J. Weingarten, G. Gruener and R. Siegwart, "A Fast and Robust 3D Feature Extraction Algorithm for Structured Environment Reconstruction", Proceedings of ICAR, 2003, Coimbra.

[30] J. Bauer, K. Karner, K. Schindler, A. Klaus and C. Zach, "Segmentation of building models from dense 3D point-clouds", 27th Workshop of the Austrian Association for Pattern Recognition 2003, Laxenburg

[31] A. Chávez and H. Karstoft, "Improvement of Kinect Sensor Capabilities by Fusion with Laser Sensing Data Using Octree", Sensors 2012, 12(4), 3868-3878

[32] D. Jung and K. K. Gupta, "Octree-Based Hierachical Distance Maps for Collision Detection", Journal of Robotic Systems Volume 14, Issue 11, pages 789–806, November 1997

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Direct Trajectory Generation for Vision-Based Obstacle Avoidance

Roel Pieters, Alejandro Alvarez-Aguirre, Pieter Jonker and Henk Nijmeijer

Abstract— **In this paper, a method for direct trajectory generation for robotic manipulators is proposed. The method is specifically designed for obstacle avoidance and can incorporate kinematic constraints into the avoidance motion. In particular, a point-to-point and multi-point trajectory generator is proposed in which different levels of constraints can be established on via- and end-points. The approach is direct in the sense that at every time instant a new trajectory is generated, which allows the trajectory to be changed at runtime. Moreover, when the final conditions of the trajectory are altered such that constraints would be violated, an optimization procedure is employed in order to extend the execution time of the trajectory. This effectively makes the trajectory time-optimal. The approach is experimentally verified with a 7-DOF anthropomorphic manipulator with time-synchronization of multiple trajectories.**

I. Introduction

With the increasing demand of integrating robotics into every day life and industry, safety requirements are still a driving factor. In a human-centred environment, robot motion has to be as smooth as possible and safety has to be guaranteed. This implies a safe replanning of motion when obstacles are detected. As current state-of-the-art approaches differentiate between obstacle avoidance (i.e., path planning) and traditional positioning (i.e., trajectory planning), the problem of avoidance is usually solved by designing a new path. This means that predefined kinematic constraints for the trajectory are not taken into account for obstacle avoidance and only a reactive motion guides the robot away from objects (e.g. potential field, roadmap).

This combination of traditional motion control with direct and online replanning of trajectories is the concept of this paper. In particular, our approach designs a new trajectory at every iteration, even when no obstacle is detected. This enables the generation of a trajectory only for the next state of the motion system based on current state and events. As such, sudden, unexpected actions that need replanning of motion can be taken into account. Direct trajectory design is presented for point-to-point and multi-point positioning, for different levels of constraints. More specifically, a different choice of constraints on (via-)points and on the complete trajectory itself will result in a different motion design. In order to guarantee that motion bounds are not violated, an optimization procedure is employed which regards the extension of the execution time in order to meet predefined kinematic constraints.

Roel Pieters, Alejandro Alvarez-Aguirre, Pieter Jonker and Henk Nijmeijer are with the Eindhoven University of Technology, Department of Mechanical Engineering, Dynamics and Control Group, PO Box 513, 5600 MB Eindhoven, The Netherlands, r.s.pieters@tue.nl, a.alvarez.aguirre@ieee.org, p.p.jonker@tue.nl, h.nijmeijer@tue.nl

A. Related work

Trajectory design for robot motion control is one of the earliest fields of research in robotics. Traditional approaches that are now accepted as standard implementations can be found in many well-known textbooks (see for example [1], [2]). It is well known that path planning [3] and trajectory planning [4] are two different topics. The former considers only the geometry of positioning, while the latter considers time and can thus include constraints on for instance velocity and acceleration. This difference is of importance, as commonly, replanning of motion (e.g., obstacle avoidance) is designed on the path planning level and motion is designed and constrained separately by motor controllers.

Traditional trajectory generation is commonly based on the assumption that initial and final conditions (e.g. velocity and acceleration for a 5^{th} order polynomial) are equal to zero. The work of Ahn et al. [5] proposes a method, denoted arbitrary states polynomial-like trajectory (ASPOT), which designs a trajectory with arbitrary initial and final conditions. The method generates the trajectories online, however, constraints are not taken into account.

The work of Thompson et al. [6] describes a trajectory generation approach which explicitly considers the presence of obstacles. The method entails adding a fourth order term to a cubic polynomial and a cost function to the state equations. Solving for the parameters of the polynomial given initial and final conditions then generates polynomial trajectories which minimise the cost function.

The work of Namiki et al. [7] presents an online trajectory generator for catching a flying ball in mid-air. A 5^{th} order polynomial is used to describe all possible target trajectories in the neighbourhood of the catching point. The parameters of the trajectory are optimized depending on the dynamics and the kinematics of the manipulator and the object. A final trajectory is then generated such that the end-effector can catch the target at one point, and a match between the position, velocity, and acceleration of the target and the end-effector is satisfied.

Motion planning in visual servoing can be found in [8]. In this, visibility and workspace constraints are considered while minimizing a cost function (e.g., spanned image area, trajectory length). As trajectories are designed in image space, only a (image) path is designed.

A complete framework for generating trajectories online is presented in the work of Kröger [9], [10]. Particularly motion systems subject to unforeseen events benefit from this approach by being able to directly react to events and switch between different control methods or domains. As such, this is a hybrid switched systems approach to robotic manipulation and is motivated to generate motion with arbitrary initial conditions. Experiments present a trajectory generation in which the final conditions can be specified up to and including velocity (i.e., 3^{rd} order) and the acceleration is set to zero.

Our approach does not consider switching between different control methods but considers changes of the trajectory itself. In particular, this entails changes due to obstacle avoidance and therefore adaptation of intermediate and final conditions (i.e., position, velocity and acceleration on via-point and end-point). This means that every iteration, a new trajectory is designed with different conditions. As such, visual measurements can be incorporated in a constrained manner.

The problem of reaching a given constraint is solved online by a computationally efficient optimization procedure that iterates to a required motion bound (e.g. velocity, acceleration) by extending the execution time. This guarantee effectively makes the generated trajectory time-optimal. Finally, synchronization between constraints as well as synchronization between DOFs is considered prior to motion generation.

The remainder of this paper is organized as follows. Section II recalls traditional trajectory planning with a minimum jerk polynomial. Section III discusses the proposed direct trajectory generation method and its use for obstacle avoidance. Finally, section IV presents simulation and experimental results.

II. Traditional Motion Generation

Traditional motion generation can be divided in two categories: offline motion planning and sensor-based motion planning (e.g., visual servoing). Commonly, sensor-based motion planning designs a path, whereas offline motion planning designs a trajectory. The difference between path planning and trajectory planning is that a path only takes geometric considerations into account. A trajectory includes time and can therefore specify kinematic constraints. Following, both methods are explained and compared in more detail.

A. Vision-based vs. Offline Motion Planning

In offline motion planning, a trajectory is designed before any motion is executed. This trajectory cannot be changed at runtime, however, constraints on the trajectory can be easily considered. A common procedure is to execute multiple trajectories successively, where subsequent trajectories can account for changes in conditions or constraints. This implies that, while executing motion, the system is blind to any changes.

Sensor-based motion planning considers the motion of a system to be dependent on the sensor at hand, which means that the motion is directly modified based on the sensor readings. The design of this motion is usually highly simplified, as incorporation of sudden events is fairly complex or too time-consuming. In particular, in vision-based control, common procedure is to use an image error $\mathbf{e}$ as feedback to control the velocity $\mathbf{v}_m$ of a manipulator:

$$\mathbf{v}_m = \mathbf{L}_{\mathbf{e}}^{\dagger}\dot{\mathbf{e}}, \quad \text{and} \quad \dot{\mathbf{e}} = -\lambda\mathbf{e}, \tag{1}$$

where $\mathbf{L}_{\mathbf{e}}^{\dagger}$ is the pseudo-inverse of the interaction matrix [11] and $\lambda > 0$ serves as a gain. This exponential error decrease can lead to non-smooth or undesirable robot motion. The initial error (step at $t = 0$) is discontinuous and kinematic constraints on the trajectory are not included. Furthermore, missing or delayed measurements have to be dealt with by e.g. an observer, otherwise instability of the system may occur.

An additional difference between both traditional methods is their execution time. Traditional offline motion planning defines a single control structure known as trajectory tracking, which can be executed at a high rate (e.g., 1 [kHz]). On the other hand, sensor-based control requires more processing time to compute a motion command (e.g., force or visual control). This gives rise to a local control loop that guarantees stability and a global loop that computes the path.

Closer inspection suggests that if both approaches could be adapted into one, the advantages of both could account for an improved motion design. This approach fits perfectly in a motion control scheme where direct reactions to sensor readings are eminent. More specifically, we propose an approach of direct trajectory generation as solution to the obstacle avoidance problem. Where path planning would direct an avoidance procedure merely on path planning level, our direct trajectory generation method considers the avoidance procedure on trajectory planning level and, as such, can consider motion constraints directly.

B. Minimum Jerk Trajectories

For simple trapezoidal trajectories, discontinuities occur during transition from constant to zero acceleration and during velocity reversal. This discontinuous acceleration will cause infinite jerk, which in turn tends to cause overshoot, electric noise in the power source and unwanted vibrations [12]. Furthermore, the larger the magnitude of the jerk is, the larger the variation of acceleration is. Smooth motion can therefore be obtained by choosing the trajectory as a 5^{th} order polynomial. This implies that its 6^{th} derivative is zero, which will minimize the integrated square jerk [13]:

$$f_{J_{min}}(t) = \int_{t=0}^{T} \dddot{q}^2(t)dt = \int_{t=0}^{T} \left[\frac{d^3q(t)}{dt^3}\right]^2 dt. \tag{2}$$

This trajectory has the form:

$$q(t) = a_1 + a_2t + a_3t^2 + a_4t^3 + a_5t^4 + a_6t^5 \tag{3}$$

for $0 \leq t \leq T$. The velocity and acceleration can be written as

$$\begin{aligned} \dot{q}(t) &= a_2 + 2a_3t + 3a_4t^2 + 4a_5t^3 + 5a_6t^4, \\ \ddot{q}(t) &= 2a_3 + 6a_4t + 12a_5t^2 + 20a_6t^3. \end{aligned} \tag{4}$$

By setting up a system of linear equations a Vandermonde [4] matrix $\mathbf{T}$ can be formulated. This effectively performs a polynomial interpolation, since solving the system of linear equations $\mathbf{Ta} = \mathbf{q}$ for $\mathbf{a}$ is equivalent to finding the coefficients of the polynomial.

A 5^{th} order polynomial implies that $\mathbf{T}$ becomes 6×6. With the points (t_k, q_k), $k \in \{i, f\}$ and considering additional constraints regarding initial (i) and final (f) velocities and accelerations, we can build the vectors $\mathbf{q}$, $\mathbf{a}$, and matrix $\mathbf{T}$ of order $n + m$ (i.e. $n + 1 = 2$ points, $m = 4$ additional constraints) as:

$$\mathbf{q} = \begin{bmatrix} q_i & q_f & v_i & \alpha_i & v_f & \alpha_f \end{bmatrix}^T = \mathbf{Ta} =$$
$$\begin{bmatrix} 1 & t_i & t_i^2 & t_i^3 & t_i^4 & t_i^5 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_i & 3t_i^2 & 4t_i^3 & 5t_i^4 \\ 0 & 0 & 2 & 6t_i & 12t_i^2 & 20t_i^3 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}. \tag{5}$$

The coefficients can then be recovered with $\mathbf{a} = \mathbf{T}^{\dagger}\mathbf{q}$, where $\mathbf{T}^{\dagger}$ is the pseudo-inverse of $\mathbf{T}$. Aforementioned method generates a minimum jerk trajectory for one degree of freedom. For multiple degrees of freedom, an equal number of trajectories has to be generated.

C. Trajectory Synchronization

Incorporating kinematic constraints into basic motion profiles is fairly straightforward. The duration (i.e., execution time) of the trajectory scales linearly (velocity) or square-root linearly (acceleration) with the affected constraint [4]:

$$t_{s,l} = \left\{ \frac{15}{8}\frac{h}{v_{max}}, \sqrt{\frac{10\sqrt{3}}{3}\frac{h}{\alpha_{max}}} \right\} \tag{6}$$

where $t_{s,l}$, $l \in \{v, \alpha\}$ is the execution time, $h = q_f - q_i$ and v_{max} and α_{max} are the maximum velocity and acceleration respectively. The constraint that is most critical is then chosen as the execution time. Considering multi-dimensional trajectories, it is highly unlikely that all motion is finalized at the same time instant. The standard approach is then to evaluate all constraints over all degrees of freedom and select the time that is most critical.

III. Direct Trajectory Generation for Obstacle Avoidance

Incorporating obstacle avoidance into a trajectory planner implies that constraints can be directly taken into account. In order to show the flexibility of the proposed approach, several methods for avoidance are presented, namely, obstacle avoidance for point-to-point motion and multi-point motion.

A. Direct Trajectory Generation

Direct motion planning requires that the order of the trajectory ($\mathcal{C}^n$) and the global constraints have to be defined on beforehand. An outline of the proposed trajectory generation method is shown in pseudo-code in Algorithm 1. When an obstacle blocks the original end-point, a new end-point position q_s is computed (from vision) for avoidance. The proposed algorithm allows for event-based or rate-based obstacle avoidance (see Algorithm 1: line 1). For event-based avoidance, a trajectory update is incorporated only when an event occurs. In rate-based avoidance, the trajectory is updated continuously enabling even small disturbances to be incorporated. One downside on the latter approach is that noise can affect the generation quite significantly.

Algorithm 1 Direct Trajectory Generation (DTG)

```
Input: $\mathcal{C}^n, \mathbf{q}, q_s$                              {initial conditions}
Output: $\mathbf{S}(k+1)$                                {next step state}
1: if $q_s > 0$ || mod $(i, 10) = 0$ then {event or rate-based}
2:     compute $t_{ev}$, $t_{e\alpha}$                       {see algorithm 2}
3:     $q_i = q(k-1)$                             {update $\mathbf{q}, \mathbf{T}, t_f$}
       $q_f = q_s$
       $v_i = v(k-1)$
       $\alpha_i = \alpha(k-1)$
       $t_f = t_s + t_e - \Delta t$                             {see (9)}
       $\mathbf{a} = \mathbf{T}^{\dagger}\mathbf{q}$
       $\mathbf{S}(k+1) = [q_{k+1}, \dot{q}_{k+1}, \ddot{q}_{k+1}]^T$      {see (3) and (4)}
4: end if
```

B. Obstacle Detection

For obstacle detection the 'SURF' [14] feature detector is employed. Greyscale images of planar obstacles are preloaded in memory of the pc and searched for continuously. Subsequent processing involves a homography estimation and decomposition [15] to obtain a Cartesian position (rotation is not considered for avoidance). As only a scaled translation can be recovered from the homography decomposition, a large margin is taken to avoid the obstacle.

C. Point-to-Point vs. Multi-point

When designing a trajectory with two points, the final point is the only variable that can be altered to avoid obstacles. This implies that after the trajectory moves away from an obstacle, still a new goal position has to be employed to move to a final end-goal.

If the trajectory is designed with multiple points, more design choices become available. For instance, via-points can be used to manoeuvre around (multiple) obstacles where each via-point accounts for one obstacle. The final point then does not necessarily need to be adapted as via-points take care of avoidance. Furthermore, the constraints on the via-point(s) can be limited to only position, as a continuous $\mathcal{C}^2$ trajectory is already guaranteed. Moreover, this choice is preferable, since in higher order trajectories, the behaviour becomes more oscillatory (i.e., Runge's phenomenon). With the addition of n via-points, the degree of the trajectory will grow with either $1n$, $2n$ or $3n$ degrees depending on the number of constraints. Unfortunately, if a via-point increases the order of the polynomial trajectory, a minimum-jerk trajectory is no longer guaranteed.

D. Constraint Optimization

When considering that a new trajectory can be generated at any arbitrary state, a relation between time and constraints is difficult to obtain. A straightforward solution is to optimize these constraints online. This implies that, every iteration, the constraints are evaluated and if, due to a redesign of the trajectory, these would be violated, additional time is added to the trajectory. The location of a current constraint (maximum or minimum) is found by computing the zero-crossings of the derivative (roots) of the considered polynomial and its magnitude by evaluating the original polynomial at these roots. A steepest descent optimization routine [16] is sufficient to accommodate for an eventual constraint mismatch and does not need to be executed in one iteration. For a velocity and acceleration constraint this is respectively expressed as

$$\begin{aligned} t_{ev} &= d_v(|v_m| - v_{max}), \\ t_{e\alpha} &= d_\alpha(|\alpha_m| - \alpha_{max}), \end{aligned} \tag{7}$$

in which $d_v > 0$ and $d_\alpha > 0$ define the rate of convergence, v_{max} and α_{max} denote the predefined constraints, and v_m and α_m the computed constraint (maximum or minimum) of the current trajectory.

As the optimization converges quite rapidly, the computations can be spread out over several iterations. Algorithm 2 presents more details of the optimization procedure in pseudo-code for point-to-point motion. For multi-point trajectories, the root solving problem becomes higher order, however, the solution method remains the same.

Algorithm 2 Constraint Optimization

```
Input: $\mathcal{T}(\mathbf{a}), v_{max}, \alpha_{max}$            {trajectory and constraints}
Output: $t_{ev} \,||\, t_{e\alpha}$                {extra time to satisfy constraint}
 1: $\mathcal{T}_{vc} = 2a_3 + 6a_4 t + 12a_5 t^2 + 20a_6 t^3$ {velocity constraint}
 2: $\mathcal{T}_{\alpha c} = 6a_4 + 24a_5 t + 60a_6 t^2$   {acceleration constraint}
 3: if $\mathcal{T}_{vc}$ then
 4:     $\mathcal{T}_{vc} = 0$          {find roots and sort descending in $\mathbf{r}$}
 5:     $t_m = \mathbf{r}(2)$                              {time of maximum}
 6:     $v_m = a_2 + 2a_3 t_m + 3a_4 t_m^2 + 4a_5 t_m^3 + 5a_6 t_m^4$
 7:     if $v_m > v_{max}$ then
 8:         $t_{ev} = d_v(|v_m| - v_{max})$                 {steepest descent}
 9:     end if
10: end if
11: if $\mathcal{T}_{\alpha c}$ then
12:     $\mathcal{T}_{\alpha c} = 0$          {find roots and sort descending in $\mathbf{r}$}
13:     $t_m = \arg\max\{\mathcal{T}_{\alpha c} = 0\}$            {time of maximum}
14:     $\alpha_m = 2a_3 + 6a_4 t_m + 12a_5 t_m^2 + 20a_6 t_m^3$
15:     if $\alpha_m > \alpha_{max}$ then
16:         $t_{e\alpha} = d_\alpha(|\alpha_m| - \alpha_{max})$             {steepest descent}
17:     end if
18: end if
```

The additional time needed to avoid violating a constraint is added to the original trajectory time. The fact that every iteration a new trajectory is generated implies that the trajectory time is continuously decreasing (except when t_{ev} or $t_{e\alpha}$ are added) and is equal to zero at the end of the trajectory. More specifically, at $t = 0$ and $t = t_f$ it holds that

$$t(0) = t_f, \quad \text{and} \quad t(t_f) = 0. \tag{8}$$

When computing the trajectory online, the initial and final time is defined as

$$t_i = 0, \quad \text{and} \quad t_f = t_s + t_e - \Delta t \tag{9}$$

where t_s is obtained from (6) and t_e is obtained from (7). Δt is the ascending trajectory time approximated as $\Delta t = T_l j$, with T_l the local loop time with iteration count j.

For a multi-dimensional trajectory, synchronization entails that the final time t_f of all trajectories is equal. The via-point time is obtained similarly to the final time defined in (9). When a trajectory is altered during runtime and additional time t_e is added, this is passed on to all other trajectories. A possible constraint violation due to this addition is dealt with by the constraint optimization procedure.

Typical for vision-based control systems is the separation of a visual loop with cycle time T_v and a local control loop with cycle time T_l, where $T_v > T_l$. As every cycle a new trajectory is generated, the real-time requirement of the visual loop is now no longer necessary.

IV. Experimental results

In order to show that the method can generate from an arbitrary state the desired motion profiles as explained in section III, first results are shown for a single DOF. Following, results are presented with a 7-DOF redundant manipulator, where motion is designed in Cartesian space.

A. Experimental Results for a Single DOF

Direct trajectory generation for a single DOF is shown in Fig. 1, for constraint optimization of acceleration. The end-point of the trajectory is changed at $t = 1.2$ $[sec]$. It can be seen that the bound of $|\alpha_{max}| = 1$ $[m/s^2]$ is not exceeded. Closer inspection of t_f shows that directly after the change in conditions, the execution time is increased to comply with the initially imposed bounds. In this case (for clarity), it is chosen to optimize to the new constraints in several iterations (one optimization step per iteration). However, due to limited number of steps necessary for convergence, it is also possible for the optimization to converge within one iteration.

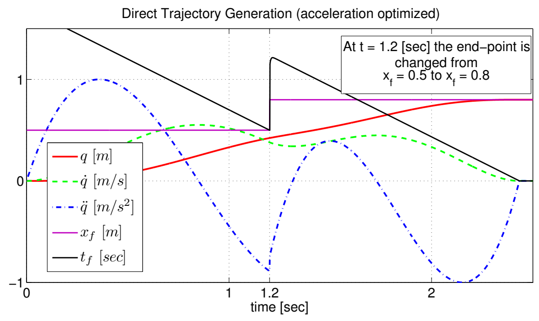


Fig. 1. Direct trajectory generation with online end-point change. In order to comply with desired constraints ($|\alpha_{max}| = 1$ $[m/s^2]$), the end-time of the trajectory t_f is iteratively increased directly after $t = 1.2$ $[sec]$. Note that the acceleration profile is continuous.

B. Manipulator Kinematics: AMOR Arm

The selected robotic manipulator is the AMOR[1] anthropomorphic arm from Exact Dynamics, B.V.[2] (see Fig. 2, and [17]), which has 7-DOF and a gripper on its end-effector. All the matrices required for the implementation of the inverse kinematics algorithm are developed in `C/C++`. For visual processing, an industrial camera (Prosilica GE680M) takes greyscale images which are processed using the computer vision library Opencv [18]. The output of the obstacle detection algorithm using SURF is shown in Fig. 3.

C. Results for Point-to-Point Whole-Arm Movements

In order to assess the obstacle avoidance method, a scenario is developed in which a robotic manipulator should execute a predefined planar positioning task, and is blocked by an obstacle at certain time and location. This means that from any arbitrary state, the manipulator should be guided to a new (online updated) end-point, while maintaining certain kinematic constraints. Fig. 4 and Fig. 5 show the simulation and experimental results for this scenario. New end-conditions are computed when the obstacle is detected and constraints are not violated.

D. Results for Multi-Point Whole-Arm Movements

A similar scenario is developed to generate the motion for obstacle avoidance with a multi-point trajectory containing 3 points (one via-point is added with only a position constraint, thus still ensuring $\mathcal{C}^2$ continuity). This allows controlling more parameters of the trajectory compared to point-to-point motion. The via-point and end-point are both determined to avoid the obstacle. Fig. 6 and Fig. 7 show the simulation and experimental results of this scenario. Once more, new

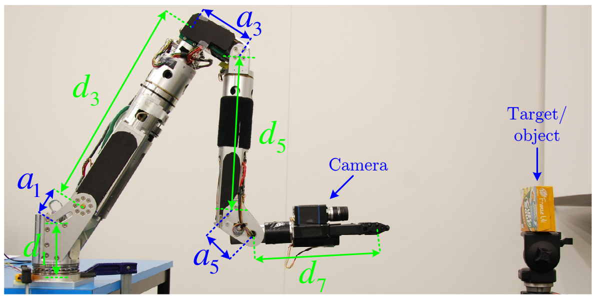


Fig. 2. Redundant 7-DOF AMOR robotic manipulator.

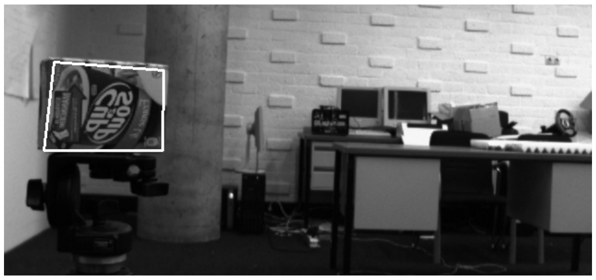
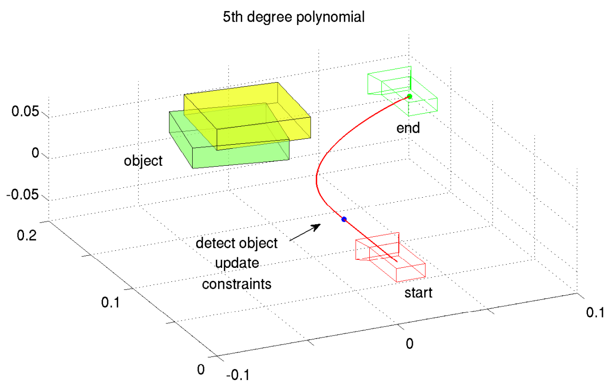


Fig. 3. Output of obstacle detection using SURF feature detector. The detected object is outlined with a white rectangle.

[1] http://www.amorrobot.com [2] http://www.exactdynamics.com



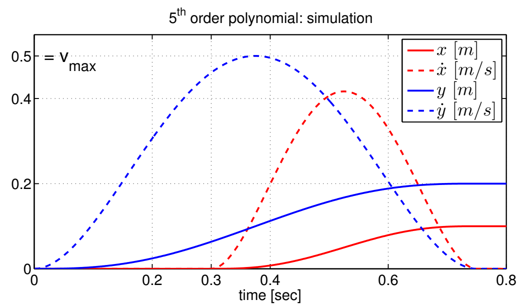
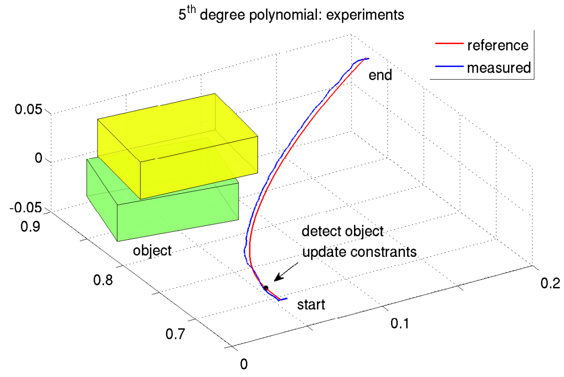


Fig. 4. Simulation for direct, online obstacle avoidance with a 5th degree polynomial (2 points, 3 constraints each). The object is smoothly avoided when detected (at $t = 0.3$ $[sec]$) with velocity constraint $v_{max} = 0.5$ $[m/s]$.



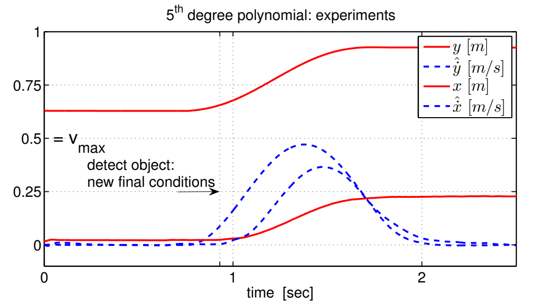


Fig. 5. Experiment for direct, online obstacle avoidance with a 5th degree polynomial (2 points, 3 constraints each). The object is smoothly avoided when detected (at $t = 0.95$ $[sec]$) with velocity constraint $v_{max} = 0.5$ $[m/s]$.

end-conditions are computed when the obstacle is detected and constraints are not violated. As the constraint is reached, time-optimality is guaranteed.

One issue that remains when designing a multi-point trajectory is the fact that, due to the addition of a via-point, the trajectory is now a 6^{th} degree polynomial, which no longer implies a minimum-jerk trajectory.

V. Conclusions

In this work, a method for direct and online trajectory generation is proposed in which constraints on via- and end-points can be taken into account at runtime. The method is suitable for point-to-point and multi-point trajectory generation, where arbitrary start- and end-states can be defined. In order to comply with predefined constraints, an optimization scheme ensures that the constraint is always reached when desired (but not violated), thus ensuring the time-optimality of the trajectory. Simulations and experiments on a 7-DOF manipulator show the effectiveness of the approach by avoiding obstacles smoothly and directly when detected.

References

[1] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[2] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer-Verlag, 1996.

[3] S. M. Lavalle, *Planning Algorithms*. Cambridge Uni. Press, 2006.

[4] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Springer-Verlag, 2008.

[5] K. Ahn, W. K. Chung, and Y. Yourn, "Arbitrary states polynomial-like trajectory (aspot) generation," in *Proc. of Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2004, pp. 123–128.

[6] S. Thompson and S. Kagami, "Continuous curvature trajectory generation with obstacle avoidance for car-like robots," in *Proc. of Int. Conf. on Computational Intelligence for Modelling, Control and Automation*, vol. 1, 2005, pp. 863–870.

[7] A. Namiki and M. Ishikawa, *Vision-Based Online Trajectory Generation and Its Application to Catching*. Control Problems in Robotics, Springer-Verlag, 2003.

[8] G. Chesi and Y. Hung, "Visual servoing: a global path-planning approach," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 2086 –2091.

[9] T. Kroger and F. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

[10] T. Kröger, *On-Line Trajectory Generation in Robotic Systems*, ser. Springer Tracts in Advanced Robotics. Springer-Verlag, 2010, vol. 58.

[11] F. Chaumette and S. Hutchinson, "Visual servo control, part I: Basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[12] S. Macfarlane and E. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, 2003.

[13] T. Flash and N. Hogan, "The coordination of arm movements: an experimentally confirmed mathematical model," *Journal of Neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

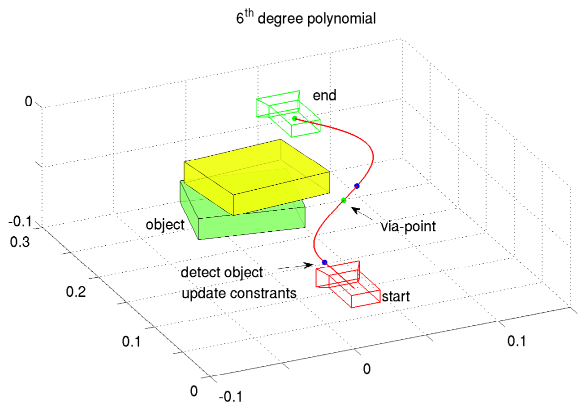
[14] H. Bay, A. Ess, T. Tuytelaars, and L. Vangool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[15] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer-Verlag, 2003.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*. New York, NY, USA: Cambridge University Press, 1992.

[17] R. Pieters, A. Alvarez-Aguirre, P. Jonker, and H. Nijmeijer, "Feed forward visual servoing for object exploration," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012 (accepted).

[18] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.



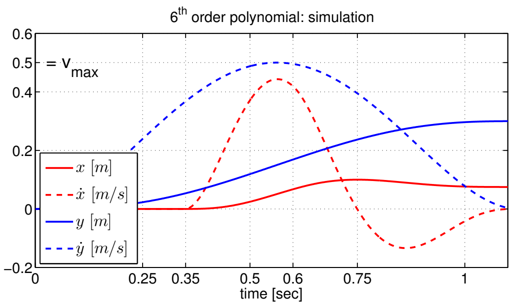
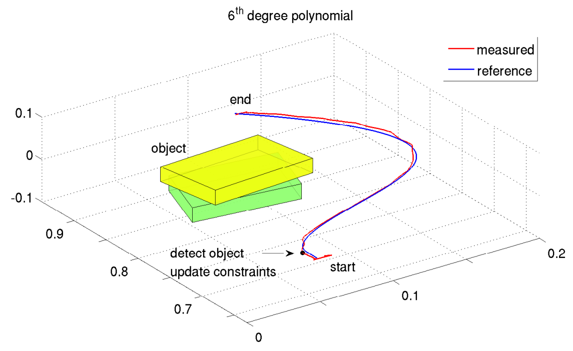


Fig. 6. Simulation for direct, online obstacle avoidance with a 6^{th} degree polynomial (3 points, 3 constraints on extremal points, only position on via-point). The object is smoothly avoided when detected (at $t = 0.35$ $[sec]$) with constrained motion $v_{max} = 0.5$ $[m/s]$. A second update is executed at $t = 0.6$ $[sec]$.



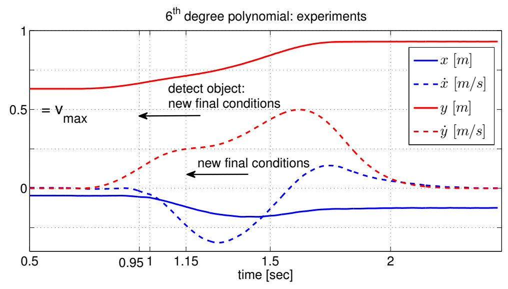


Fig. 7. Experiment for direct, online obstacle avoidance with a 6^{th} degree polynomial (3 points, 3 constraints on extremal points, only position on via-point). The object is smoothly avoided when detected (which occurs at the first blue point in the trajectory, i.e. $t = 0.95$ $[sec]$) with constrained motion $v_{max} = 0.5$ $[m/s]$. A second update is executed at $t = 1.15$ $[sec]$.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Realtime Avoidance of Fast Moving Objects: A Dynamical System-based Approach

S. Mohammad Khansari-Zadeh and Aude Billard
Ecole Polytechnique Federale de Lausanne, LASA Laboratory
{mohammad.khansari,Aude.Billard}@epfl.ch

***Abstract*—In this paper, we provide an extension to our previous approach [1] to perform obstacle avoidance in the presence of multiple fast moving and rotating obstacles. Our approach leverage on the notion of DS to generate robot motions that are inherently robust to perturbations and can instantly adapt to changes in the target and obstacles' positions in a dynamically moving environments. We validate our method in the challenging experiment of dodging a fast moving and rotating box on the 7-degrees of freedom (DoF) KUKA DLR arm.**

I. Introduction

Classical approaches to modeling robot motions rely on decomposing a task execution into two separate processes: *planning* and *control* [2]. The former is used as a means to generate a feasible path that can satisfy the task's requirements, and the latter is designed so that it follows the generated feasible path as closely as possible. These approaches consider any deviation from the desired path (due to perturbations or changes in environment) as the tracking error, and various control theories have been developed to efficiently suppress this error in terms of some objective functions. Despite the great success of these approaches in providing powerful robotic systems, particularly in factories, they are ill-suited for robotic systems that are aimed to work in the close vicinity of humans, and thus alternative techniques must be sought.

In robotics, DS-based approaches to motion generation have been proven to be interesting alternatives to classical methods as they offer a natural means to integrate planing and control into one single unit [3], [4], [5], [6], [7]. For instance when modeling robot reaching motions with DS, all possible solutions to reach the target are embedded into one single model. Such a model represents a global map which specifies *instantly* the correct direction for reaching the target, considering the current state of the robot, the target, and all the other objects in the robot's working space. Clearly such models are more similar to human movements in that they can effortlessly adapt its motion to change in environments rather than stubbornly following the previous path. In other words, the main advantage of using DS-based formulation can be summarized as: "Modeling movements with DS allows having robotic systems that have *inherent adaptivity* to changes in a dynamic environment, and that can *instantly* adopt a new path to reach the target". This advantage is particularly important in situations where there is no time to plan (or re-plan), no matter how fast the planning technique may be, and instant adaptation to a dynamically changing environment is required.

Despite the above features, most of the DS-based approaches to generating robot motions relies on a simplistic assumption that presume there is no object in the robot working space [3], [4], [7], [8]. Such assumption could be very limiting since many real world tasks require robotic systems that should work in cluttered environments where the robot may face several objects, which may appear suddenly during the task execution. Hence, it is essential to endow the existing DS-based control policy with the ability to avoid obstacles. In the face of fast moving obstacles, the devised algorithm should be computationally light so that can be used in closed-loop.

In our previous work, we have presented a novel approach to perform realtime obstacle avoidance based on dynamical systems that ensures impenetrability of multiple convex objects in quasi-static conditions [1]. This approach has a level of reactivity similar to existing local obstacle avoidance methods, while it ensures convergence to the target proper to global obstacle avoidance techniques[1]. In this paper, we extend this work for realtime obstacle avoidance in the presence of multiple fast moving and rotating objects, where the quasi-static assumption no longer holds. The presented method is free from local minima, and can ensure convergence of all trajectories to the target (as long as the target is reachable). We validate our method on the 7-DoF KUKA DLR arm in the experiment of dodging a fast moving and rotating box in 20 trials with various linear and angular velocities. Next we provide a recap of our previous work, and then present the extension for obstacle avoidance in the presence of fast moving objects.

II. DS-based obstacle avoidance

In this section we provide a brief overview of our DS-based approach for obstacle avoidance that is presented in [1]. This work assumes that the robot motion is driven by a continuous and differentiable DS in the absence of obstacle(s):

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}), \qquad \boldsymbol{f}: \mathbb{R}^d \mapsto \mathbb{R}^d \qquad \text{autonomous DS} \quad (1)$$

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t, \boldsymbol{\xi}), \qquad \boldsymbol{f}: \mathbb{R}^+ \times \mathbb{R}^d \mapsto \mathbb{R}^d \qquad \text{non-auto. DS} \quad (2)$$

where $\boldsymbol{\xi} \in \mathbb{R}^d$ is a state variable that defines the state of the robot, t corresponds to time, and $\boldsymbol{f}(.)$ could be either an autonomous or non-autonomous DS. For simplicity, we further

[1]Please refer to [1] for more discussion on differences between this work and relevant state-of-the-art obstacle avoidance algorithms.

use the notation $\boldsymbol{f}(.)$ to refer to both autonomous and non-autonomous DS. Given an initial point $\{\boldsymbol{\xi}\}^0$, the robot motion along time can be computed by integrating $\boldsymbol{f}(.)$ recursively.

In this paper we take an imitation learning approach to construct $\boldsymbol{f}(.)$. Given a set of N demonstrations $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$, an estimate of $\boldsymbol{f}(.)$ can be built using different techniques such as Stable Estimator of Dynamical Systems (SEDS) [3]. Note that throughout this paper we assume the DS $\boldsymbol{f}(.)$ is provided by the user, and henceforth we will call it the *original DS*. Next we describe our DS-based obstacle avoidance.

A. Analytical description of obstacles

Consider a d-dimensional object centered at a reference point $\boldsymbol{\xi}^o$. We denote the position of a point $\boldsymbol{\xi} \in \mathbb{R}^d$ with respect to the frame of reference centered at $\boldsymbol{\xi}^o$ with $\tilde{\boldsymbol{\xi}} = \boldsymbol{\xi} - \boldsymbol{\xi}^o$. Suppose a continuous function $\Gamma(\tilde{\boldsymbol{\xi}})$ that projects $\mathbb{R}^d$ into $\mathbb{R}$. The function $\Gamma(\tilde{\boldsymbol{\xi}})$ has continuous first order partial derivatives (i.e. $\mathcal{C}^1$ smoothness) and increases monotonically with $\|\tilde{\boldsymbol{\xi}}\|$. The level curves of Γ (i.e. $\Gamma(\tilde{\boldsymbol{\xi}}) = c$, $\forall c \in \mathbb{R}^+$) enclose a convex region. By construction, the following relation holds at the surface of the obstacle:

$$\Gamma(\tilde{\boldsymbol{\xi}}) = 1 \tag{3}$$

For example $\Gamma(\tilde{\boldsymbol{\xi}}) : \sum_{i=1}^{d}(\tilde{\boldsymbol{\xi}}_i/\boldsymbol{a}_i)^2 = 1$ corresponds to a d-dimensional ellipsoid with axis lengths $\boldsymbol{a}_i$. We can divide the space spanned by Γ into three regions $\boldsymbol{\mathcal{X}}^o$, $\boldsymbol{\mathcal{X}}^b$, and $\boldsymbol{\mathcal{X}}^f$ to distinguish between points inside the obstacle, at its boundary, and outside the obstacle respectively:

$$\text{Interior points}: \quad \boldsymbol{\mathcal{X}}^o = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) < 1\} \tag{4}$$

$$\text{Boundary points}: \quad \boldsymbol{\mathcal{X}}^b = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) = 1\} \tag{5}$$

$$\text{Free region}: \quad \boldsymbol{\mathcal{X}}^f = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) > 1\} \tag{6}$$

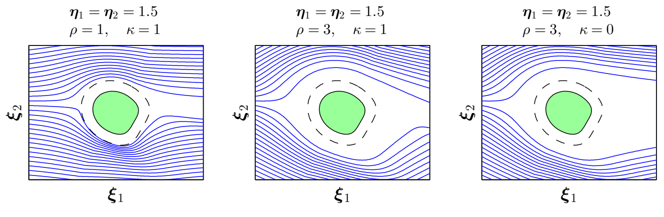
Note that in case of non-convex objects, we could consider a smooth convex envelope (also known as convex bounding volume) that fits tightly around the object. When the point cloud description of an object is available, one could also use one of the estimation techniques such as the one presented in [9] to approximate a $\mathcal{C}^1$ smoothness bounding volume around the object.

B. Modulation

Given the original DS and an analytical formulation describing the surface of K obstacles $\Gamma^1(\boldsymbol{\xi}^{o,1}) \cdots \Gamma^K(\boldsymbol{\xi}^{o,K})$, we would like to determine a combined dynamic modulation matrix $\bar{\boldsymbol{M}}$ so as to instantly modify the robot motion to avoid collision with multiple static obstacles:

$$\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\boldsymbol{\xi})\boldsymbol{f}(.) \tag{7}$$

Algorithm 1 provides the procedure on how to determine $\bar{\boldsymbol{M}}$. For brevity of this paper, we do not describe here the effect of each line in Algorithm 1 on the combined dynamic modulation matrix, and refer interested readers to [1]. However, there are three parameters that are noteworthy: the safety factors $\boldsymbol{\eta}^k$, the reactivities ρ^k, and the tail effects κ^k.



(a) Safety factor **(b)** Reactivity **(c)** Tail effect

Fig. 1: Illustration of the effect of the safety factor, reactivity, and the tail effect on the modulation. In these graphs, the original dynamics is a uniform flow moving from left to right: $\dot{\boldsymbol{\xi}} = [1; 0]$. **(a)** The area between the dashed line and the obstacle boundary is the safety margin. **(b)** By increasing the reactivity to 3, the trajectories now deflect earlier in time and go further out. **(c)** By setting $\kappa = 0$, We could remedy the tendency of the trajectories to follow the obstacle shape after passing it. Note that in this situation, the slight modulation of the trajectories after passing the obstacle is still required in order to ensure the continuity in the velocity.

For each obstacle $k \in 1..K$, the safety factor $\boldsymbol{\eta}^k \in \mathbb{R}^d$ with $\boldsymbol{\eta}_i^k \geq 1$, $\forall i \in 1..d$, controls the required safety margin around the object by virtually inflating the object along each direction $\tilde{\boldsymbol{\xi}}_i$ with the magnitude $\boldsymbol{\eta}_i^k$ (in the obstacle frame of reference). The reactivity parameter $\rho^k > 0$ controls the responsiveness of the robot to the presence of each obstacle. The larger the reactivity, the earlier the robot responds to the presence of an obstacle. This parameter is very crucial for practical purposes as one may prefer to deflect the robot trajectory earlier when it goes toward a fire flame than when it is just heading towards a soft pillow. The last parameter $\kappa \in \{0, 1\}$ controls whether the robot motion should still be modified after passing the object ($\kappa = 1$ corresponds to this situation). Figure 1 shows the effect of these three factors on the modulation.

Note that the multiplication of the combined dynamic modulation matrix guarantees the impenetrability of all the K obstacles. However in many robot experiments, not only should the robot avoid the obstacle, but it should also reach a target. In other words, we would like the modified motion to preserve the convergence property of the original dynamics while still ensuring that the motion does not collide with the object(s). As described in [1], the multiplication of $\bar{\boldsymbol{M}}$ does not change the critical points of the original dynamics. Hence, if the original DS is globally stable (i.e. all trajectories reach the target point) when there is no obstacle in the robot working space, it also remains stable in the presence of obstacles[2].

III. Extension to Multiple Moving Obstacles

In our previous work [1] we have considered situations where obstacles are static. In this section we extend our previous formulation to perform obstacle avoidance in the presence of multiple moving obstacles with linear and/or rotational velocities. In the presence of one single obstacle, this extension is straight forward and can be achieved by

[2]It should be noted that the modulation term $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ may also create other possible equilibrium points at the boundary of obstacles. As these possible equilibrium points *solely* appear on the obstacles' boundary, they can be tackled by using a contouring mechanism.

Algorithm 1 DS-Based obstacle avoidance as described in [1]

```
Input: $\boldsymbol{\xi}$, $\boldsymbol{f}(.)$, and $\{\boldsymbol{\eta}^k, \rho^k, \kappa^k\}$
1: for each obstacle $k, k \in 1..K$ do
2:     $\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k = (\boldsymbol{\xi} - \boldsymbol{\xi}^{o,k})./\boldsymbol{\eta}^k$
3:     $\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \left[\boldsymbol{n}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \quad \boldsymbol{e}^{1,k}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \quad \cdots \quad \boldsymbol{e}^{d-1,k}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\right]$
4:     $\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{cases} 1 & \text{if } K = 1 \\ \prod_{i=1, i\neq k}^{K} \frac{(\Gamma^i(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)}{(\Gamma^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)+(\Gamma^i(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)} & \text{otherwise} \end{cases}$
5:     $\begin{cases} \lambda_1^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{cases} 1 - \frac{\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)|^{\frac{1}{\rho}}} & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T\dot{\boldsymbol{\xi}} < 0 \textbf{ or } \kappa = 1 \\ 1 & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T\dot{\boldsymbol{\xi}} \geq 0 \textbf{ and } \kappa = 0 \end{cases} \\ \lambda_i^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = 1 + \frac{\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)|^{\frac{1}{\rho}}} \quad 2 \leq i \leq d \end{cases}$
6:     $\boldsymbol{D}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{bmatrix} \lambda_1^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \lambda_d^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \end{bmatrix}$
7:     $\boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\, \boldsymbol{D}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\, \left(\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\right)^{-1}$
8: end for
9: $\bar{\boldsymbol{M}}(\boldsymbol{\xi}) = \prod_{k=1}^{K} \boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)$
Output: $\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\boldsymbol{\xi})\boldsymbol{f}(.)$
```

computing the modulation in the obstacle's frame of reference. Suppose an obstacle $\Gamma(\tilde{\boldsymbol{\xi}})$ that is moving with linear and rotational velocities $\dot{\boldsymbol{\xi}}_L^o$ and $\dot{\boldsymbol{\xi}}_R^o$, respectively. The modulated dynamics for obstacle avoidance becomes:

$$\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\tilde{\boldsymbol{\xi}})\big(\boldsymbol{f}(.) - \dot{\boldsymbol{\xi}}_L^o - \dot{\boldsymbol{\xi}}_R^o \times \tilde{\boldsymbol{\xi}}\big) + \dot{\boldsymbol{\xi}}_L^o + \dot{\boldsymbol{\xi}}_R^o \times \tilde{\boldsymbol{\xi}} \quad (8)$$

where $(.) \times (.)$ denotes the cross product and $\bar{\boldsymbol{M}}(\tilde{\boldsymbol{\xi}})$ is the modulation given by Algorithm 1. In this equation, the term $\boldsymbol{f}(.) - \dot{\boldsymbol{\xi}}_L^o - \dot{\boldsymbol{\xi}}_R^o \times \tilde{\boldsymbol{\xi}}$ transforms the velocity of the robot to the obstacle's coordinates system. Then, the modulation is performed in this coordinates system where the object is static, yet the robot is moving with a different speed. After applying the modulation, the result is transformed back to the world's frame of reference through the last term.

Equation (8) ensures impenetrability of a single moving obstacle. To verify this, suppose a point $\boldsymbol{\xi}^b$ on the boundary of the moving obstacle at time t. After multiplying the modulation matrix, the radial velocity of the robot is canceled, and hence the robot can only move along the tangential hyperplane at $\boldsymbol{\xi}^b$. However, this is still not enough as the robot may hit the obstacle in the next moment t^+ since the obstacle is moving. The collision can be avoided by adding the instant velocity of the point $\boldsymbol{\xi}^b$ due to obstacle motion, which is given by $\dot{\boldsymbol{\xi}}_L^o + \dot{\boldsymbol{\xi}}_R^o \times (\boldsymbol{\xi}^b - \boldsymbol{\xi}^o)$, to the modulated velocity.

As a side effect, Eq. (8) could induce some unnecessary movements to the robot even when the robot is far from the obstacle (note that the angular velocity grows proportionally with $\|\tilde{\boldsymbol{\xi}}\|$). This can be tackled by adding an exponential term that diminishes the induced velocity due to the obstacle's movement as $\|\tilde{\boldsymbol{\xi}}\|$ increases:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{M}(\tilde{\boldsymbol{\xi}})\Big(\boldsymbol{f}(.) - e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)}\big(\dot{\boldsymbol{\xi}}_L^o + \dot{\boldsymbol{\xi}}_R^o \times \tilde{\boldsymbol{\xi}}\big)\Big) + \cdots$$
$$+ e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)}\big(\dot{\boldsymbol{\xi}}_L^o + \dot{\boldsymbol{\xi}}_R^o \times \tilde{\boldsymbol{\xi}}\big) \quad (9)$$

where σ^o is a positive scalar controlling the rate of decay of the exponential term. The higher the σ^o, the earlier the robot responds to the obstacle motion. The above change does not compromise impenetrability of the obstacle as we have $e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)} = 1$ on the boundary of the obstacle.

In the presence of multiple moving obstacles, further considerations should be taken so that the above transformation smoothly shift from one obstacle to another based on the current position of the robot. To achieve this goal, we use the weighting coefficients that are computed in the 4th line of Algorithm 1 to control the priorities of obstacles.

Let us consider K disconnected obstacles that are described by $\Gamma^k(\tilde{\boldsymbol{\xi}}^k)$, $k \in 1..K$, with associated translational and rotational velocities $\dot{\boldsymbol{\xi}}_L^{o,k}$ and $\dot{\boldsymbol{\xi}}_R^{o,k}$, respectively. We define the net shift in velocity due to the presence of these obstacles as:

$$\bar{\dot{\boldsymbol{\xi}}}^o = \sum_{k=1}^{K} \dot{\boldsymbol{\xi}}^{o,k} = \sum_{k=1}^{K} e^{-\frac{1}{\sigma^{o,k}}(\Gamma^k(\tilde{\boldsymbol{\xi}}^k)-1)}\omega^k(\tilde{\boldsymbol{\xi}}^k)\big(\dot{\boldsymbol{\xi}}_L^{o,k} + \dot{\boldsymbol{\xi}}_R^{o,k} \times \tilde{\boldsymbol{\xi}}^k\big) \quad (10)$$

where $\omega^k(\tilde{\boldsymbol{\xi}}^k)$ are computed according to Algorithm 1. In case the tail effect is not desired (i.e. $\kappa = 0$), one could remove the modulation effect by setting $\dot{\boldsymbol{\xi}}^{o,k} = 0$ for each obstacle that is moving away from the robot (i.e. when $\left(\dot{\boldsymbol{\xi}}^{o,k}\right)^T \tilde{\boldsymbol{\xi}}^{o,k} < 0$).

The combined modulation that considers the net effect of all moving/static obstacles is then given by:

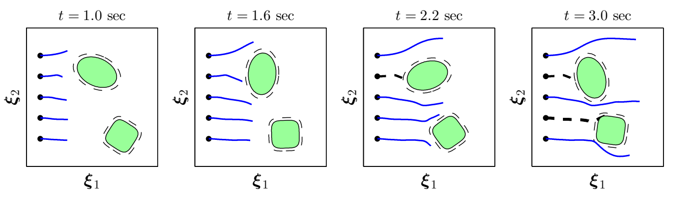
$$\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\boldsymbol{\xi})\big(\boldsymbol{f}(.) - \bar{\dot{\boldsymbol{\xi}}}^o\big) + \bar{\dot{\boldsymbol{\xi}}}^o \quad (11)$$

where $\bar{\boldsymbol{M}}(\boldsymbol{\xi})$ is given by Algorithm 1. Equation (11) ensures the impenetrability of all the K obstacles. For a point $\boldsymbol{\xi}^b$ on the boundary of the k-th obstacle, only $\omega^k = 1$ and all the other weighting coefficients are zeros. Hence $\bar{\boldsymbol{M}}(\boldsymbol{\xi}^b) = \boldsymbol{M}^k(\boldsymbol{\xi}^b)$ and $\bar{\dot{\boldsymbol{\xi}}}^o = \dot{\boldsymbol{\xi}}_L^{o,k} + \dot{\boldsymbol{\xi}}_R^{o,k} \times \tilde{\boldsymbol{\xi}}^{b,k}$, and thus the obstacle is impenetrable. Similarly to the static case, by moving from one obstacle to another, the weighting coefficients smoothly change between zero and one, and by this, impenetrability is always ensured for all the obstacles.

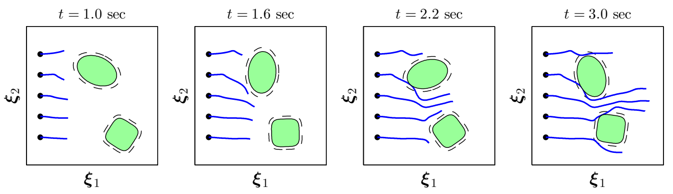
Figure 2 shows an example of obstacle avoidance in the presence of two moving obstacles. It compares two situations: 1) The quasi-static case where the obstacles' motion are neglected, and the modulation is computed at each time based on the instantaneous position and orientation of the obstacles (see Fig. 2a), and 2) The dynamic case where the obstacles' motion are taken into account (see Fig. 2b). As we can see, in the quasi-static case the impenetrability of the obstacles are no longer ensured, whereas in the dynamic case all the trajectories can safely pass the obstacles.

IV. Robot experiments

In this section we evaluate our approach in the presence of a fast moving obstacle, where the quasi static-assumption is no longer valid. The experiment consisted of having the 7-DoF KUKA DLR arm stay in a default target position while a box is slid towards the robot at high speed. Thus the robot should react quickly and change its position so that the box passes without any collision (see Fig. 3).



(a) Without considering the obstacles' motion (the quasi-static case). The dashed black lines show the failure cases where the robot actually collides with the obstacles.



(b) With considering the obstacles' motion. In this case, collision avoidance for all trajectories is ensured.

Fig. 2: Illustration of the obstacle avoidance in the presence of two moving obstacles. As we can see, solely in the dynamic case, where the obstacles' motion is considered, the trajectories can safely pass the obstacles. In this example, the trajectories move from left to right with $\dot{\boldsymbol{\xi}} = [2; 0]$ m/s. The oval-shaped object has the linear velocity $\dot{\boldsymbol{\xi}}_L^{o,1} = [-0.4; -0.2]$ m/s and the rotational velocity $\dot{\boldsymbol{\xi}}_R^{o,1} = -2$ rad/s. These values for the square-shaped obstacle are $[-0.4; -0.2]$ m/s and 1 rad/sec. Both objects have the safety factor of $\boldsymbol{\eta} = 1.2$. The variance σ^o is set to 2 and 10 for the oval and square-shaped obstacles, respectively.

The KUKA robot is controlled in the Cartesian coordinate system, and the control commands are sent at 1000Hz. We use the damped least square pseudo-inverse kinematics to compute the robot's joint angles. The torque command to the robot is computed based on the desired kinematic command using the KUKA built-in PID controller. The original DS, deriving the robot motion in the absence of obstacles, is modeled using the Stable Estimator of Dynamical Systems (SEDS) [3]. SEDS builds an estimate of a globally asymptotically stable DS from a set of demonstrations (in the absence of obstacles) provided by the user. This DS is then used to control the robot motion by generating velocity commands to keep the robot's end-effector close or, when it is feasible, at the target point.

We define the box reference point at $x^{o,B} = x^{c,B}$, $y^{o,B} = y^{c,B}$, and $z^{o,B} = 0$, and model it with the analytical formulation $\Gamma(\tilde{\boldsymbol{\xi}})^B$: $((x - x^{o,B})/0.055)^2 + ((y - y^{o,B})/0.165)^2 + ((z - z^{o,B})/0.23)^4 = 1$. Other parameters are set as follows: $\boldsymbol{\eta} = [3.5 \;\; 2.0 \;\; 1.5]^T$, $\rho = 2$, $\sigma = 30$, and $\kappa = 0$. The box's position and orientation are tracked at 240Hz using an OptiTrack vision system. We use a Kalman filter to reduce the noise effect on estimations. The working table is modeled with $x^{o,T} = y^{o,T} = 0$, $z^{o,T} = -0.01cm$ and $\Gamma(\tilde{\boldsymbol{\xi}})^T$: $((x - x^{o,T})/3)^6 + ((y - y^{o,T})/3)^6 + ((z - z^{o,T})/0.01)^4 = 1$. We set the safety factor of the table to $\boldsymbol{\eta} = 1.3$. The position of the table is set fixed in the whole experiment.

In total we ran 20 trials, lasting between 0.8 to 1.3 seconds, in which the box was slid from different initial configurations with various linear and angular velocities. In each trial, the box was set to an initial distance of about 0.5 meter away from the robot and was thrusted so as to reach a maximum linear velocity of $0.6 \sim 1.5$ m/s and/or a maximum angular velocity of $40 \sim 120$ deg/s. In 16 out of the 20 trials, the robot successfully managed to dodge the box. Figure 3 shows sequences of the motion for four of the trials. The trajectories of the robot's end-effector and the box, and the magnitude of the box's linear and angular velocities are also illustrated in Fig. 4.

The four failure cases could possibly be due to two factors that are not currently considered in our formulation: 1) The filtering of the object's position and orientation introduces a lag in determining the current linear and angular velocities of the box. In situations where the box is moving and rotating fast at a very close distance to the robot, the presence of this lag could yield collision with the obstacle. 2) The robot's joints cannot move faster than a certain value due to the hardware limitation, and hence collision with the obstacle is inevitable. Figure 5 shows the sequences of the motion for one of the failure cases. In this trial, though the avoidance seems successful at the initial stage of the motion, the box hit the end-effector from the backside due to the wrong estimation of the object's angular velocity.

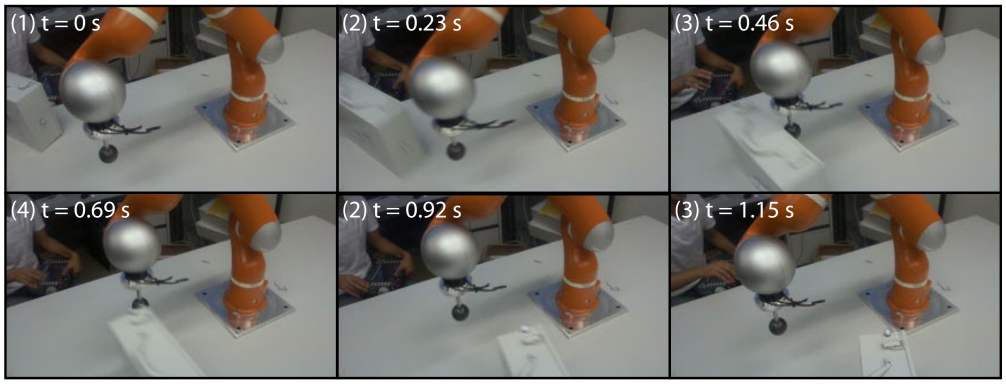
The first factor can be alleviated by using a more advanced filter or by increasing the safety factor. However, the second factor cannot be easily tackled. Some improvements might be achieved by using a planner technique that could take into account such hardware limitations during the path generation. However, as in the above failure situations the obstacle is moving fast at a very close distance to the robot, this planner should be extremely fast to provide a valid solution within an order of millisecond (recall the robot is controlled at 1000Hz).

V. Summary and Conclusion

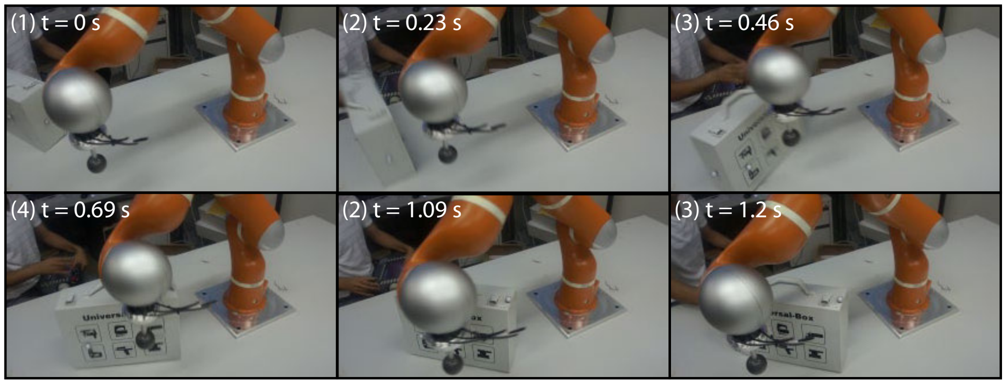
In this paper, we have extended our previous approach to perform obstacle avoidance in the presence of fast moving objects, where the quasi-static assumption no longer holds. The proposed approach requires the user to provide a DS model that governs the robot motion in the absence of obstacles and a smooth analytical formulation describing a convex bounding volume around each obstacle. Given the above information as well as the realtime position and orientation of obstacles, it instantly provides a modulation to the DS model of the motion so as the robot does not collide with the obstacles.

We have validated the applicability of our method in a real robot experiment where a fast moving and rotating box was slid towards the robot at various speeds. In most trials, the robot manages to successfully dodge the box despite its fast motion at a very close distance. Due to high speed motion of the box, the accuracy in estimating its position and orientation play an important role for the safe collision avoidance. In our implementation, there is an upper bound for the maximum amount of inaccuracies that can be handled, which is a function of the safety factor, reactivity parameter, and the object's velocity.

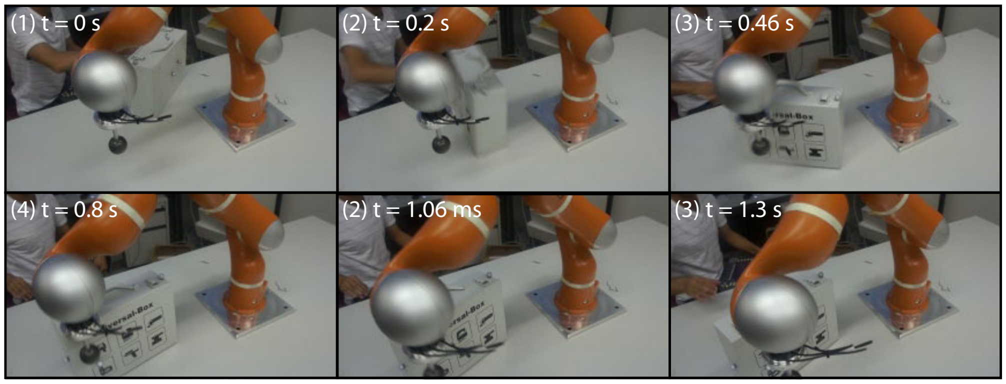
Our approach is currently limited in that it does not consider the robot's hardware limitations during the avoidance. The



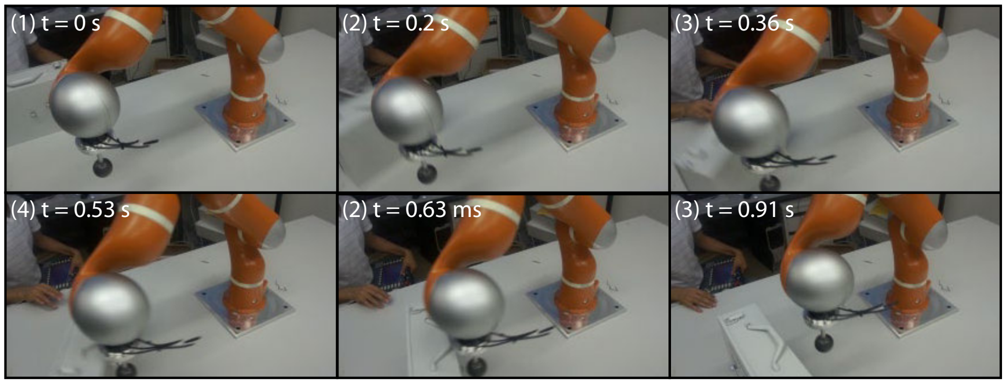
(a) First trial.



(b) Second trial.

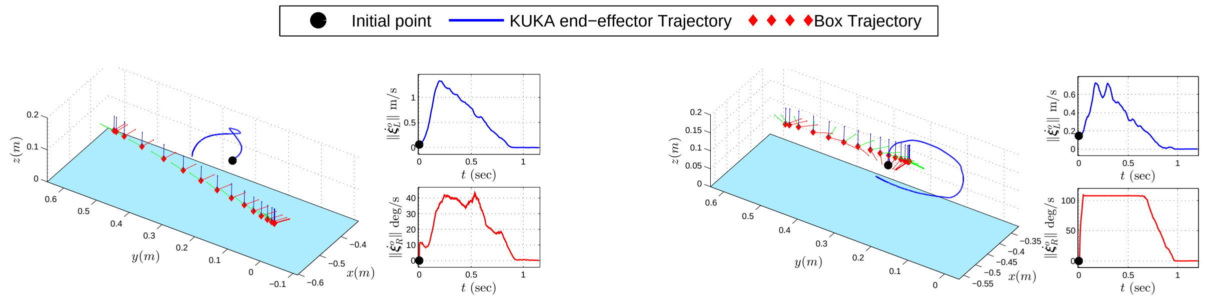


(c) Third trial.

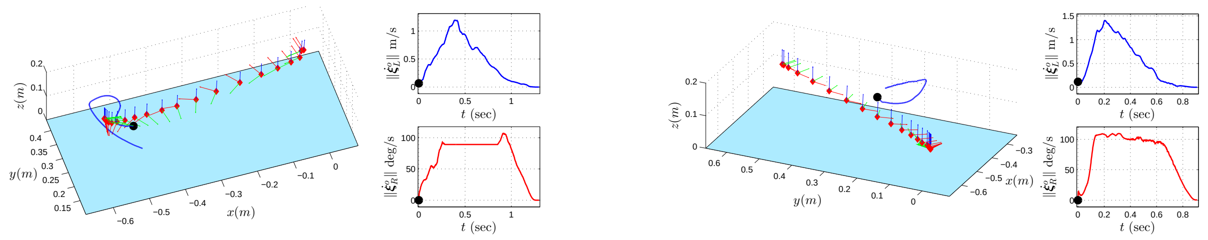


(d) Forth trial.

Fig. 3: Illustration of sequences of motion for 4 out of the 20 executed trials. In this experiment the robot was required to dodge a sliding box that was launched 20 times from different initial configurations with various linear and angular velocities. For further information please refer to Section IV.



(a) First trial. **(b)** Second trial.



(c) Third trial. **(d)** Forth trial.

Fig. 4: Illustration of trajectories of the robot's end-effector and the box, and the magnitude of the box's linear and angular velocities for the four trials shown in Fig. 3. In these graphs, the x, y, and z axes of the box's frame of reference are shown with red, green, and blue vectors, respectively. For further information please refer to Section IV.

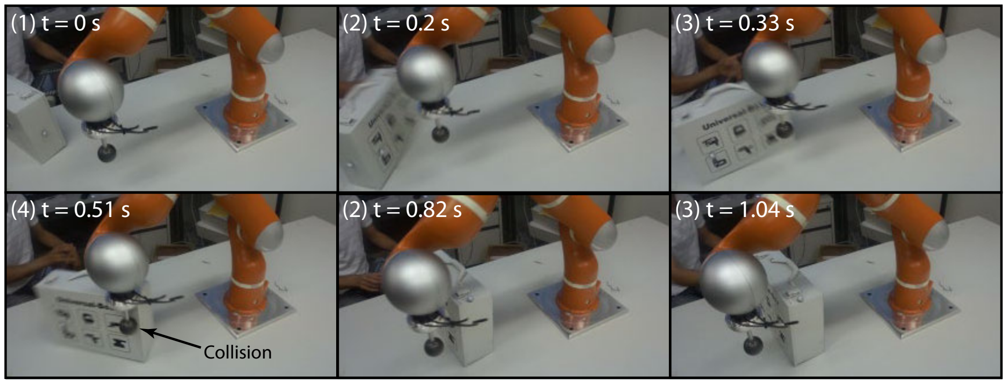


Fig. 5: Illustration of sequences of motion for one of the four cases in which the robot failed to successfully dodge the box.

DS modeling can compensate for deviations (due to hardware limitations) from the desired trajectory, by instantly adapting a new trajectory for the new position of the robot. However, an inevitable outcome of such compensation is that the robot executes the motion at a slowest pace than what is expected, which may yield to collision.

Acknowledgment

This work was supported by the European Commission through the EU Project AMARSI (FP7-ICT-248311).

References

[1] S.-M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, pp. 433–454, 2012.

[2] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[3] S. M. Khansari-Zadeh and A. Billard, "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models," *IEEE Trans. on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[4] S. Calinon, A. Pistillo, and D. G. Caldwell, "Encoding the time and space constraints of a task in explicit-duration hidden Markov model," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[5] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, pp. 1–19, 2012.

[6] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, "Learning the skill of archery by a humanoid robot iCub," in *Proc. IEEE Int. Conf. on Humanoid Robots (Humanoids)*, 2010, pp. 417–423.

[7] S. M. Khansari-Zadeh, K. Kronander, and A. Billard, "Learning to Play Minigolf: A Dynamical System-based Approach," *Advanced Robotics*, 2012.

[8] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009, pp. 1293–1298.

[9] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar, "Fast C1 Proximity Queries using Support Mapping of Sphere-Torus-Patches Bounding Volumes," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2009, pp. 483–488.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Online task space trajectory generation

Daniel Sidobre[1] and Wuwei He[2]

Abstract— **As the kinematic of the robots becomes complex and the task to realize are more and more demanding, we need tools to better define and manipulate the movements of the robots. To cope with this problem, we propose a family of trajectory, which we name the Soft Motion trajectories, defined by polynomial functions of degree three. Based on these trajectories we propose a set of tools to generate trajectories and control the robots. We present some experimental results showing the interest of the approach that unify the data exchanged from the planning level to the control level of the robot.**

I. Introduction

As machines become more and more complex and precise, they can not settle for following a path, they need better defined moves to design and think motions. The concept of trajectories that defines the position as a function of times allows building more powerful tools to animate machines. Some systems already use trajectories, but not in an integrated way from planning to control. This paper focuses on trajectories defined as series of segments of polynomial function of degree three and proposes a set of tools to generate and manipulate them.

The classical approach utilized by most of machines consists in defining a path and expect that the system can follow it. Unfortunately, many of these paths cannot be followed efficiently and precisely. For instance some paths are defined as polygonal lines that require the system stops at each vertices or approximates the lines around the vertices. It is well known that the path must be at least of class C^2 to be feasible. But this smoothness condition is practically not sufficient as the maximal speed depends on the local radius of curvature of the path.

The use of a trajectory to define a move gives all the necessary elements to verify that the move is feasible. Using a dynamic simulator, all the physical characteristic of the move can be verified: collisions, maximum velocity, maximum power, maximum torque etc.

From a control point of view, trajectories are also very interesting because they allow simpler control strategies. Torsten Kroeger showed the possibility to switch very easily between different controller [1]. For robot interacting with humans, trajectories allow to express easily the safety and comfort constraints as kinematic limits. Thanks to advances in computers sciences, the trajectories can now be manipulated very efficiently.

This paper is organized as follows. Trajectory generality are presented in section II. A set of trajectory generators are described in section III. The section IV details a method to approximate trajectories with polynomial third degree trajectories. A solution to generate trajectories from polygonal lines is described in section V. A trajectory controller is presented in the section VI. Experimental results are presented in section VII. Finally we give some concluding remarks in section VIII.

II. Trajectories

To clarify the subject, we first introduce trajectories and give their main properties. Then, we detail the model of trajectories based on series of cubic polynomial functions and introduce different tools to manipulate them.

Trajectories are time functions defined in geometrical spaces, like essentially Cartesian space and joint space. The rotations can be described using different coordinates system: quaternion, vector and angle etc. The books from Biagiotti [2] on one hand and the one from Kroger [1] on the other hand summarize background trajectory material.

Given a system whose position is defined by a set of coordinate X if the coordinates are in Cartesian space or Q if the coordinates are in joint space, a trajectory $\mathcal{T}$ is a function of time defined as:

$$\mathcal{T} : [t_I, t_F] \longrightarrow \mathbb{R}^n \tag{1}$$

$$t \longmapsto \mathcal{T}(t) = X(t) \tag{2}$$

The trajectory is defined from the time interval $[t_I, t_F]$ to $\mathbb{R}^n$ where n is the dimension of the motion space. The $\mathcal{T}(t)$ function can be a direct function of time or the composition $\mathcal{C}(s(t))$ of a function giving the path $\mathcal{C}(s)$ and a function $s(t)$ describing the time evolution along this path.

At first glance the latter offer more possibilities as the time evolution is independent of the geometrical path and so the two elements can be modified independently. Unfortunately, this approach is limited by the difficulty to integrate the derivative of the path to obtain the curvilinear abscissa. Without this parameterization, the function $s(t)$ doesn't give the tangential velocity and the kinematic of the motion is difficult to manipulate and interpret. So, as the former has a simpler expression, it provides simpler solutions to define and manipulate trajectories.

A trajectory $\mathcal{T}(t)$ defined from t_I to t_F can be defined by a series of trajectories defined between intermediate points. Given, t_u which satisfies $t_I < t_u < t_F$, an equivalent

This work has been supported by the European Community's Seventh Framework Program FP7/2007-2013 "SAPHARI under grant agreement no. 287513 and by the French National Research Agency project ANR-07-ROBO-0011 "ASSIST" and ANR-10-CORD-0025 "ICARO".

[1]D. Sidobre is with CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France; and Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France daniel.sidobre@laas.fr

[2]W. He is with CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France Wuwei.He@laas.fr

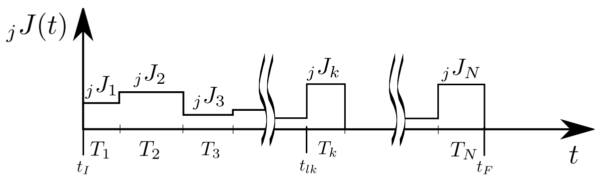


Fig. 1. Jerk profile for a series of segments of cubic polynomial trajectory

representation of $\mathcal{T}(t)$ is defined by the series of two trajectories T_1 and T_2 defined respectively by :

$$\begin{array}{ll} \mathcal{T}_1 : [t_I, t_u] \longrightarrow \mathbb{R}^n & \mathcal{T}_2 : [t_u, t_F] \longrightarrow \mathbb{R}^n \\ t \longmapsto \mathcal{T}_1(t) = \mathcal{T}(t) & t \longmapsto \mathcal{T}_2(t) = \mathcal{T}(t) \end{array} \tag{3}$$

Similarly a trajectory can be defined by a series of sub-trajectories if some continuity criteria specified for the trajectory and its derivative are verified. Generally this criterion is defined as a differentiability class C^k with $k \geq 2$.

The possible choices to define trajectory functions are very large, but as we intend to compute motions in real time, we choose a simple solution like polynomial functions. As we need C^2 functions, we choose polynomial function of third degree and name this trajectories Soft Motion trajectories. Using a long series of polynomial function, trajectories following very complex path can be defined. It is also possible to approximate or interpolate a set of points to define Soft Motions trajectories.

In the sequel, we firstly present Soft Motion trajectories and then a set of consistent trajectory generator to solve robotic problems.

Series of 3rd degree polynomial trajectories

We define Soft Motion trajectories as series of 3rd degree polynomial trajectories. Such a trajectory is composed of a vector of one-dimensional trajectories: $\mathcal{T}(t) = ({}_1Q(t), {}_2Q(t), \cdots, {}_nQ(t))^T$ for joint motions or $\mathcal{T}(t) = ({}_1X(t), {}_2X(t), \ldots, {}_nX(t))^T$ in Cartesian space. Without loss of generality, we suppose that all ${}_jX(t)$ or all ${}_jQ(t)$, $0 \leq j < n$ share the same time intervals and that $t_I = 0$. A one dimensioned trajectory ${}_jX(t)$ is defined by its initial conditions (${}_jX(0) = {}_jX_I$, ${}_jV(0) = {}_jV_I$ and ${}_jA(0) = {}_jA_I$) and K elementary trajectories ${}_jX_i(t)$ defined by the jerk ${}_jJ_i$ and the duration T_i where $1 \leq i \leq K$ and $\sum_{i=1}^{K} T_i = t_F - t_I$. By integration we can define the acceleration ${}_jA(t)$, the velocity ${}_jV(t)$ and then the position ${}_jX_i(t)$.

Assuming $k \leq K$ is such that $\sum_{i=1}^{k-1} T_i \leq t < \sum_{i=1}^{k} T_i$, the trajectory ${}_jX(t)$ and its derivative are defined by :

$${}_jJ(t) = {}_jJ_k \tag{4}$$

$${}_jA(t) = {}_jJ_k \left(t - \sum_{i=1}^{k-1} T_i \right) + \sum_{l=1}^{k-1} {}_jJ_l \, T_l + {}_jA_I \tag{5}$$

$$\begin{aligned} {}_jV(t) = {} & \frac{{}_jJ_k}{2} \left(t - \sum_{i=1}^{k-1} T_i \right)^2 + \sum_{l=1}^{k-1} {}_jJ_l \, T_l \left(t - \sum_{i=1}^{l} T_i \right) \\ & + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^2}{2} + {}_jA_I t + {}_jV_I \end{aligned} \tag{6}$$

$$\begin{aligned} {}_jX(t) = {} & \frac{{}_jJ_k}{6} \left(t - \sum_{i=1}^{k-1} T_i \right)^3 + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l}{2} \left(t - \sum_{i=1}^{l} T_i \right)^2 \\ & + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^2}{2} \left(t - \sum_{i=1}^{l} T_i \right) + \sum_{l=1}^{k-1} \frac{{}_jJ_l \, T_l^3}{6} \\ & + \frac{{}_jA_I}{2} t^2 + {}_jV_I t + {}_jX_I \end{aligned} \tag{7}$$

This general expression of the trajectories and their derivatives can be used directly to control a arm, for example, but it is not easy to obtain directly. So we will now describe different generators to build them.

III. Trajectory generators

As different motion problems exist, we need a coherent set of trajectory generator. To classify these trajectory generators we use the different types of motions we wish to define Soft Motions for. The first one is the point-to-point motion that can be done in minimum time or in an imposed time. A point-to-point move is a move where the mobile starts from rest and stops after the move. A more general problem is defined between two general situations; in this case the initial and final conditions are arbitrary. The motions can also be defined by a set of via point to approximate or interpolate. A very interesting problem is to approximate any trajectory by a Soft Motion one. Finally Soft Motions can be classified by the dimension of the motion space.

In the following sections, we present generators for each of these trajectory problems, beginning by the simpler to build the more complex.

A. One-dimensional generator

To cope with system physical limits using 3rd degree polynomial functions, we can limit the jerk, the acceleration and the velocity:

$$-J_{max} \leq J(t) \leq J_{max} \tag{8}$$

$$-A_{max} \leq A(t) \leq A_{max} \tag{9}$$

$$-V_{max} \leq V(t) \leq V_{max} \tag{10}$$

This limits define a domain for the one dimensional systems presented in the figure 2 bottom using the Acceleration-Velocity frame. In this diagram, motions with constant jerk draw parabolas.

A canonical generation of trajectory problem is defined in this domain by initial and final conditions:

$$X(t_I) = X_I \quad V(t_I) = V_I \quad A(t_I) = A_I \tag{11}$$

$$X(t_F) = X_F \quad V(t_F = V_F \quad A(t_F) = A_F \tag{12}$$

The quasi-optimal solution in minimum time to this problem is presented in [3], [4]. The well known canonical case of long point-to-point motion is depicted in figure 2. In this

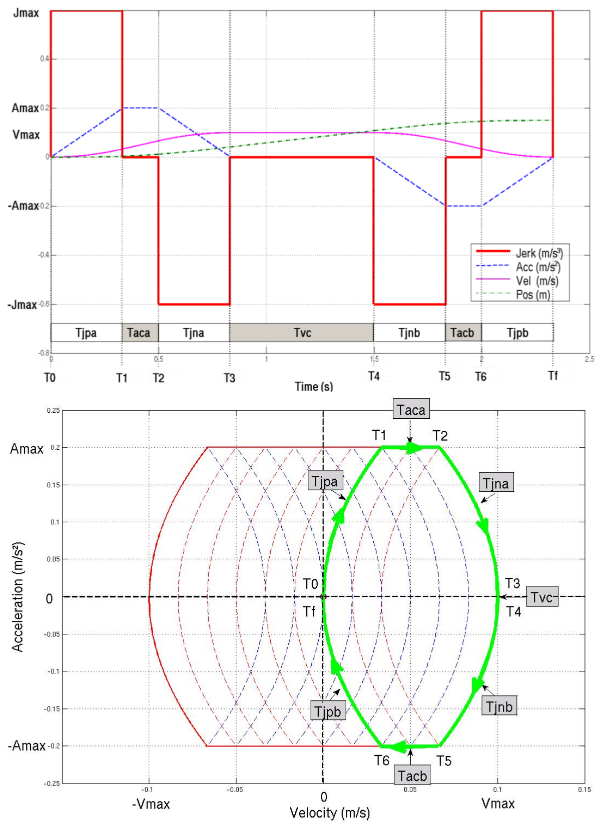


Fig. 2. Top: Position, velocity, acceleration and jerk for a point-to-point move in function of time. Bottom: the same move in the frame Acceleration-Velocity with the bounds of the validity domain.

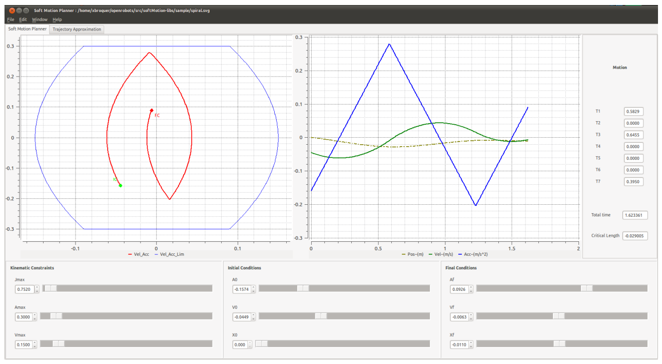


Fig. 3. Graphic user interface of the Soft Motion planner displaying a general short motion.

case the motion is composed of seven segments of 3^{rd} degree polynomial functions. This is the maximum number of segments as either the point with maximal velocity or with minimal velocity can be reached with three segments from any situation and similarly to return to any situation. From a computation point of view, the figure 3 exhibit the worse case where the intersection of three parabola has to be computed. This case generates a polynomial equation of degree 6 numerically solved by Raphson-Newton method.

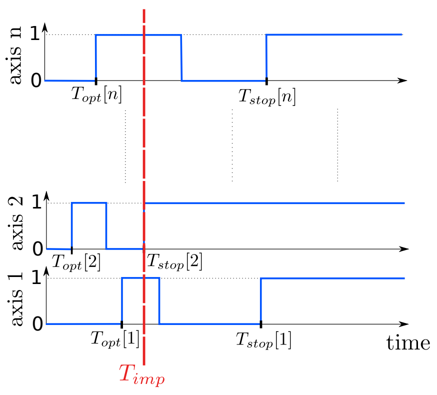


Fig. 4. Valid interval of time for a set of axes and the time T_{imp} corresponding to the move in minimum time.

B. N-dimensional point-to-point generator

The generation of trajectories in multidimensional spaces is far more complex as we will see in the next section, but the point-to-point motion is a particular case that can be bring back to a unidimensional problem. Suppose the line joining the initial and final points in $\mathbb{R}^N$ is defined relatively to a basis $\{v_i\}_{0<i\leqslant N}$ by a vector $v = \sum_{i=1}^{N} \alpha_i v_i$ with $\sum_{i=1}^{N} \alpha_i^2 = 1$. The velocity, the acceleration and the jerk are respectively limited for each axis i by V_{iM}, A_{iM} and J_{iM}.

The minimum time trajectory is directly obtained by projecting on each axis the solution of the one-dimensional problem defined on the line segment by the limits [4]:

$$J_{\max} = \min_{1\leqslant i\leqslant N} \frac{1}{\alpha_i} J_{iM} \tag{13}$$

$$A_{\max} = \min_{1\leqslant i\leqslant N} \frac{1}{\alpha_i} A_{iM} \tag{14}$$

$$V_{\max} = \min_{1\leqslant i\leqslant N} \frac{1}{\alpha_i} V_{iM} \tag{15}$$

For each segment of the trajectory, one of the velocity acceleration, or jerk functions of one of the N initial axes is saturated. The others are inside their validity domain.

C. N-dimensional general generator

In this case, initial and final velocity and acceleration are no longer zero and the problem can no longer be linearized. In a first step, we compute the minimum time movement using the method of the paragraph III-A for each axis and select the longest one $T_{min} = \max_{1\leq i\leq n} T_{opt}i$. The minimum time movement for the move cannot be lower than this time T_{min}. In some case, it is possible to compute for each of the other axes a move in this time T_{min}. Unfortunately the minimum time for the move can be larger than T_{min} as it is not always possible to increase the time of a motion for all values. For example, suppose a one axis mobile moving at V_{max} during a short time t_i so it travels a distance of X_i. Therefore, we wish to increase t_i of δ_t. For some $\delta_t = \delta_l$ we obtain a limit case where the movement is composed of two segments, the first with the jerk $-J_{max}$ and the last with

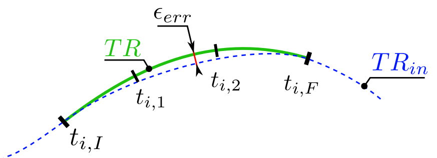


Fig. 5. The initial trajectory $\mathcal{T}_{in}$ and the approximated $\mathcal{T}$.

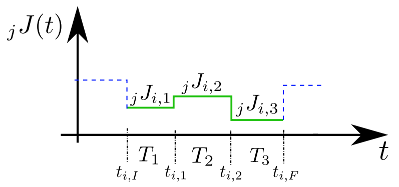


Fig. 6. Jerk profile for the axis j of the trajectory $\mathcal{T}$.

J_{max}. For a $\delta_t < \delta_l$ an infinity of solution exists to do the move in $t_i + \delta_t$, but for $\delta_l < \delta_t < \delta_u$ there is no solution. δ_u corresponds to the necessary time to the mobile to stop going beyond the final point, to go back and return. As we choose an initial motion at V_{max} it is not possible to do the move in a time less than t_i. Figure 4 shows the choice of the minimum valid time T_{imp} for a set of axes. In the case of the picture, the minimum time for each axis is T_{opt} of the first axis and the minimal time feasible is T_{stop} of the second axis.

So we can determine the minimum time T_{imp} for an N-dimensional move. But an infinity of solutions exists. Our system proposes one solution, but an optimum criterion is still to build hoping it gives simple computations. The shape of the path defining the trajectory depends on this choice. For motion planning in presence of obstacles this choice has an important influence.

Now we have a solution to generate a trajectory to define a move between two situations. In the following we introduce generators that master the shape of the trajectory between the initial and final situations.

IV. Trajectory approximation:

We now wish to define any motion with a set of polynomial trajectories of third degree. To do this, we propose to approximate any trajectory by a Soft Motion trajectory.

Suppose $\mathcal{T}_{in}$ is an arbitrary trajectory defined, for example, by a path P and a motion law $u = u(t)$. Both the path P and the law u can be defined by a large variety of curves (Bézier, NURBS, sinusoid etc.). If the differentiability class of this trajectory is at least C^2, a good approximation can be computed. But in case of discontinuity, we must accept a higher error. This error can be balanced between a geometric error and a time error. In case of geometric errors, the initial and realized paths are different but outside these difficult zones the trajectory can be precisely realized. In case of time errors, the mobile can stop to stay on the path and ensure velocity and acceleration continuity, but such a modification introduces a delay for the remaining trajectory.

A. The three segments method

If we consider a portion of the trajectory $\mathcal{T}_{in}$ defined by an initial instant $t_{i,I}$ and a final instant $t_{i,F}$, $\mathcal{T}_{in}$ defines the initial and final situations to approximate: (X_I, V_I, A_I) and (X_F, V_F, A_F).

An interesting solution to approximate this portion of trajectories is to define a sequence of three trajectory segments with constant jerk that bring the mobile from the initial situation to the final one in the time $T_{imp} = t_F - t_I$. We choose three segments because we need a small number of segments and there is no always solution with one or two segments.

The system to solve is then defined by 13 constraints : the initial and final situations (6 constraints), the continuity in position velocity and acceleration for the two switching situations and the time. Each segment of trajectory is defined by four parameters and one time. If we fix the three duration $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$, we obtain a system with 13 parameters where only the three jerks are unknown. As the final control system is periodic with period T, the times $T_{imp}/3$ must be a multiple of the period T and T_{imp} chosen to be a multiple of $3T$.

The 3 jerks are then defined by:

$$\begin{bmatrix} J_1 \\ J_2 \\ J_3 \end{bmatrix} = A^{-1}. \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} \tag{16}$$

with

$$A^{-1} = \frac{1}{T_{imp}} \begin{bmatrix} 1 & -9 & 27 \\ -7/2 & 27 & -54 \\ 11/2 & -18 & 27 \end{bmatrix} \tag{17}$$

and

$$B_1 = A_F - A_I$$
$$B_2 = V_F - V_I - A_I T_{imp}$$
$$B_3 = X_F - X_I - V_I T_{imp} - A_I \frac{T_{imp}^2}{2}$$

More details can be found in [5] and [4].

B. Distance between trajectories:

An important characteristic of the approximation is the maximum error between the two trajectories. As several distances exist to compare trajectories, we choose the Hausdorff distance and the synchronous Euclidean distance. Another interesting measure of the difference is the synchronous Euclidean distance between the velocities.

The synchronous Euclidean distance is defined by:

$$d_{SE} = \max_{t \in [t_I, t_F]} \sqrt{\sum_{j=1}^{n} \left({}_j\mathcal{T}(t) -{}_j \mathcal{T}_{in}(t)\right)^2} \tag{18}$$

The synchronous euclidean distance between velocities is defined by:

$$d_{SEV} = \max_{t \in [t_I, t_F]} \sqrt{\sum_{j=1}^{n} \left(\frac{d_j\mathcal{T}(t)}{dt} - \frac{d_j\mathcal{T}_{in}(t)}{dt}\right)^2} \tag{19}$$

The Hausdorff distance is defined by:

$$d_{Haus} = \max(\sup_{t_{in}\in[t_I,t_F]} \inf_{t\in[t_I,t_F]} d(\mathcal{T}_{in}(t_{in}), \mathcal{T}(t)), \quad (20)$$
$$\sup_{t\in[t_I,t_F]} \inf_{t_{in}\in[t_I,t_F]} d(\mathcal{T}(t), \mathcal{T}_{in}(t_{in}))) \quad (21)$$

Depending on the type of problem, one of these distances is generally more appropriated. If the geometry of the path is important as for example for machining application, the Hausdorff distance is a good choice. For moves in free space, coordination is more important and so the synchronous Euclidean distance is suitable. The distance between velocities is more sensible to identify the variation due to the motion law.

C. *Error of approximation for a trajectory*

We suppose now that $\mathcal{T}_{in}$ is bounded respectively in jerk, acceleration and velocity by J_{max}, A_{max} and V_{max}. We show in this paragraph that a relation exists between the error of approximation, the time T_{imp} and the bound J_{max}.

Let $\mathcal{V}_{in}$ and $\mathcal{A}_{in}$ denote respectively the velocity and acceleration of $\mathcal{T}_{in}$. In a first time, we examine the case where the trajectory $\mathcal{T}_{in}$ to approximate satisfies:

$$\mathcal{T}_{in}(t_I) = \mathcal{T}_{in}(t_F) = 0 \quad (22)$$
$$\mathcal{V}_{in}(t_I) = \mathcal{V}_{in}(t_F) = 0 \quad (23)$$
$$\mathcal{A}_{in}(t_I) = \mathcal{A}_{in}(t_F) = 0 \quad (24)$$

One can verify that this initial and final conditions gives three null jerks (See eq. 16).

The trajectory to approximate $\mathcal{T}_{in}$ that gives the maximum error is symmetric. As the trajectory to approximate $\mathcal{T}_{in}$ is kinematically bounded and due to the symmetry, the maximum error between the two trajectories is at the middle of the trajectory. Likewise, the maximum error is obtained for a saturated function. For a short trajectory, the acceleration is not saturated and the more difficult function to approximate is defined by the four segments trajectory:

$$T_1 = T_4 = T_{imp} * \frac{2-\sqrt{2}}{4} \quad (25)$$
$$T_2 = T_3 = T_{imp} * \frac{\sqrt{2}}{4} \quad (26)$$

and the jerks are $J_1 = J_3 = J_{max}$ $J_2 = j_4 = -J_{max}$.

The maximum error between the two trajectories is then:

$$\epsilon = \frac{\sqrt{2}-1}{48 * \sqrt{2}} = 0.0061 \times J_{max} * T_{imp}^3 \quad (27)$$

General case: Suppose $\mathcal{T}(t)$ is the approximation by the 3 segments method of the trajectory $\mathcal{T}_{in}(t)$between t_I and t_F.

We can write the $\mathcal{T}_{in}(t)$ trajectory as $\mathcal{T}_{in}(t) = \mathcal{T}(t) + (\mathcal{T}_{in}(t) - \mathcal{T}(t))$

By design the trajectory $\mathcal{T}_0(t) = \mathcal{T}_{in}(t) - \mathcal{T}(t)$ verifies the conditions 22, 23, and 24.

So the approximation error of $\mathcal{T}_0(t)$ on $[T_I, T_F]$ by a trajectory composed of three segments of cubic polynomial trajectory is less than $0.0061 \times T_{imp}^3 * (2 \times J_{max})$.

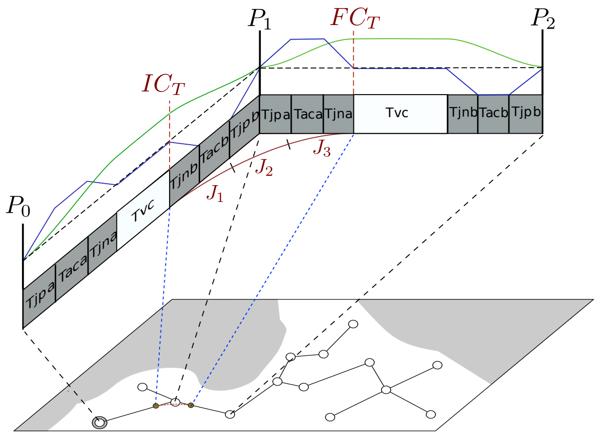


Fig. 7. From a polygonal path to a Soft Motion trajectory.

As $\mathcal{T}(t)$ is approximated without error, $\mathcal{T}_{in}(t)$ that is the sum $\mathcal{T}(t) + \mathcal{T}_0(t)$ can be approximated with an error less than: $0.0061 \times T_{imp}^3 \times (2 \times J_{max})$.

This result is extremely interesting as it gives the length of the interval to approximate a function while insuring the approximation error is smaller than a defined limit.

D. *Example of a circular trajectory:*

To approximate a trajectory following a circle of radius R at constant speed ωR, we can compute the maximum time interval T_{imp} to approximate the circle with a maximum error of ϵ.

The trajectory of the motion is defined by:

$$X_x(t) = R \times cos(\omega t) \quad (28)$$
$$X_Y(t) = R \times sin(\omega t) \quad (29)$$

and the jerk by:

$$J_X(t) = \omega^3 R \times sin(\omega t) \quad (30)$$
$$J_Y(t) = -\omega^3 R \times cos(\omega t) \quad (31)$$

So the constant jerk is $\omega^3 R$ and the maximum time interval is then

$$T = \frac{\epsilon\sqrt{3}}{0.0061 \times 2 \times J} = \frac{\epsilon\sqrt{3}}{0.0061 \times 2 \times \omega^3 R} \quad (32)$$

For a mobile completing a turn in one second about a circle of radius $R = 0.1m$ with a maximum error of $\epsilon = 10^{-6}$, T is $T = 0.0149s$ corresponding to 68 points (6 points for an error of $\epsilon = 10^{-3}$, T). This result can be used directly when radius of curvature is known and the path is traversed at constant speed.

V. Generating trajectory from polygonal path

The main advantage of the Soft Motion trajectories is to insure a continuity of data and reasoning from the high planning level to the control level. Most of the motion planners as, for example RRT planners [6], [7], produce only paths in the form of polygonal lines. The assimilation of a path to a trajectory commonly performed at planning level is not acceptable for control. So these paths should be converted in trajectories. We suppose the line segments of the path are relatively long after a path planner optimization phase and so the robot can reach the maximum velocity and stop to traverse each segment. Given a set of kinematic limits $\{J_{max}, A_{max}, V_{max}\}$, a trajectory stopping at each vertex is easily build using the linear generator of paragraph III-B.

In [5] we proposed to use a twofold strategy to smooth this trajectories. The first idea is to smooth the vertex of the polygonal line between the points where the robot must begin to decelerate (IC_T) and can stop to accelerate (FC_T) following precisely the path (see figure 7). Between the two situations FC_T and IC_T the 3 segments method gives a simple path. The second strategy take into account that the initial polygonal trajectory that stop at the vertex has no collision and can be used when the smoothed trajectory causes a collision.

The time to go from FC_T to IC_T is defined by the method presented in paragraph III-C. As we wish a continuous motion, we use the 3 segments method of the paragraph IV. The trajectory is then checked for collision.

This strategy to compute a smoothing segment of trajectory can be improved by optimizing the choice of the initial and final points defining the segment [4] as a shorter segment generating a smaller error is preferable in some situations. The trajectory computed by choosing initial and final points between IC_T and P_1 and P_1 and FC_T respectively is sometimes feasible.

VI. A TRAJECTORY CONTROLLER

We suppose each axe of the mobile is equipped with a low level controller, for example a PID controller. This low level controller can be directly fed from a trajectory definition using the expressions 7 or 6.

In general, controllers can also get feedback from complex localization systems: a mobile manipulator robot which exchange an object with a human needs to localize itself, the human and the object. To obtain its position this robot can use different sensors (cameras, lasers, odometry) and localization techniques based on different sensors, different geometric elements and different filtering techniques. To localize the object, it can use stereovision or point cloud obtained from sensors like Kinect.

So, a general control problem can be defined by:

1) a frame in which the trajectory to follow, which is generated by high level, is defined.
2) the current position, velocity and acceleration of the mobile.
3) the current position, velocity and acceleration of the target.
4) a trajectory to be executed by the controller.

A. The frame of a trajectory

Given a situation where a robot is supposed to grasp an object handed by the human. At the beginning of the task, a planner computes a trajectory for the robot. At this instant, the trajectory can be expressed in any moving or fixed frame equivalently. But after a short moment, because of the movement of humans and robots, the trajectories expressed in the alternative frames are different. For example, the end of the trajectory defines an approaching path defined to avoid collisions. This local path must be defined in a frame associated to the object, so that the gripper approaches the object along this path even though the object is moved by the human. In the same way, if the beginning of the trajectory is defined in the frame associated to the object and the human rotate a little the object at the beginning, the start of the trajectory can made a big move relatively to a fixed frame.

So each part of a trajectory must be associated to a frame in which it must be controlled. The choice of the instant to switch the controller from one frame to another must also be defined. In the previous example, the system can switch from the robot base frame to the object frame when the gripper reaches some distance from the object. Eventually, it is possible to define an intermediate segment of trajectory controlled in a third frame, for example a frame associated to the human hand.

When a robot is very close to an obstacle, controlling the robot in a frame fixed to this obstacle is generally a good solution to minimize the uncertainty and limit the risk of collision.

In conclusion, the planner must generate a trajectory and associate a control frame to each part of the trajectory.

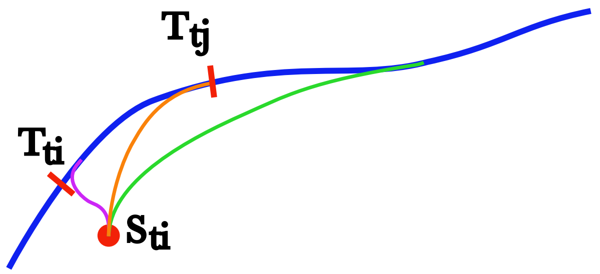


Fig. 8. Trajectory control: At instant t_i, the mobile should be at point T_{t_i} on the blue trajectory, but it is in point S_{t_i} because of the situation change. The orange trajectory reaches the blue at the instant t_j. The green trajectory reaches the blue trajectory in a longer time and the purple one in a shorter time.

B. Computing a control trajectory

Given a segment of trajectory that has to be controlled in some frame, we present now a control strategy to cope with the inherent uncertainty associated to the position. Because of the large position error associated to the base position, of the possibility to switch from a controller to another or of the possibility to switch from a sensor to another, the distance between the real position and the setting position can be large.

The control problem is illustrated in figure 8 where some mobile must follow the blue trajectory. At the instant t_i, the mobile should be in point T_{t_i} but it is in S_{t_i}. Only the position is depicted on the figure but the system take also into account velocity and acceleration.

From this initial situation, the controller must compute a control trajectory that reaches the input trajectory as soon

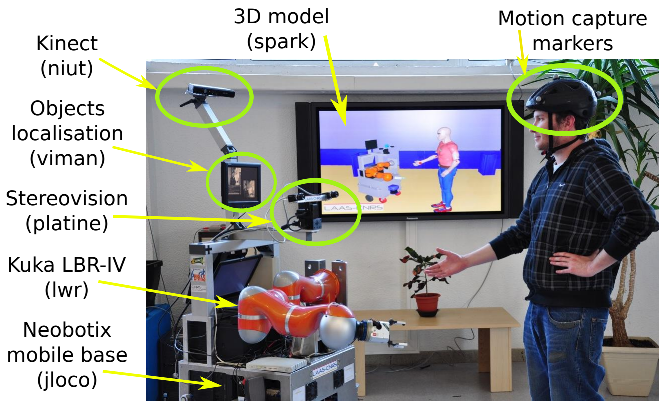


Fig. 9. A human interacting with the robot Jido and the main elements of the system.

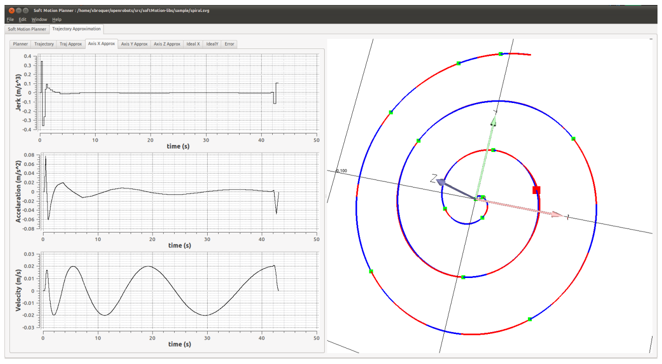


Fig. 10. A spiral trajectory approximated. Left the jerk, the acceleration and the velocity profiles.

as possible while complying with constraints. This can be done simply by computing a trajectory to reach the input trajectory for each instant t_k for $k > i$ until a valid trajectory is obtained. The computed control trajectory is drawn in orange in the figure 8. Due to real time constraints, it must be necessary to define a more efficient strategy in case of important error and k large.

The error between the real situation and the desired one at an instant can be due to many reasons. The strategy to build a control trajectory that reaches the input trajectory can be different in function of the problem. The previous solution is the solution when the objective is really the trajectory, but in some case the path is more important than the time and it can be preferable to choose a shorter path to minimize the Hausdorff error to the path. The purple trajectory of the figure 8 shows an example of such a trajectory. Of course in this case we accept a delay for the mobile. Later the mobile can or cannot catch up with this delay depending on the problem and conditions.

Similarly, a smoother trajectory could be preferable, the green trajectory of the figure 8 gives an example.

This control trajectory are computed with the three segments algorithm presented in section IV-A as the initial and final situations are precisely defined.

C. *Target tracking*

Trajectory control can also be used in the absence of input trajectory. For example for the robot reach a relative position between the robot hand and an object, the local control trajectory can be directly defined between the current state of the robot hand (position, velocity, acceleration) and the future state of the target. The future state of the target is estimated assuming a continuous and regular motion.

In this case the time to reach the target is not imposed and the trajectory generator presented in section III-A is used. If we need that all the axes move synchronously, the method presented in section III-C can be used.

This approach can also be used at the end of a controlled trajectory move to maintain the relative position of the hand.

VII. Experimental results

To illustrate the implementation of this tools based on Soft Motion trajectories, we present results carried out with our Jido robot. Jido is built up with a Neobotix mobile platform MP-L655 and a Kuka LWR-IV arm. Jido is equipped with one pair of stereo cameras and a Kinect motion sensor. The figure 9 describes the main elements of its architecture. To integrate the software, we use GENOM[1] [8], a development environment for complex real time embedded software, and robotpkg[2], a compilation framework and packaging system for installing robotics software.

To simplify the robot software, the different modules are organized around the SPARK module, which receives the data from all the software modules and builds a model of the scene. An example of this model is presented in the background of the figure 9. SPARK processes internal data from software modules that interface sensors and actuators (position of the arm and hand, position of the stereovision plate, odometer etc) and data about environment and humans (Kinect, motion capture, stereovision etc). The advantage of this centralized approach is the possibility for the module SPARK to compute accurate positions and kinetic parameters using different input data and filtering techniques. The robot is also equipped with a collision checker that verifies in line the risks of collision and stop the robot if necessary. In the actual implementation this module cannot take into account moving objects in the environment.

A human aware planner [9] computes a trajectory for the robot from the SPARK model and a description of the task to achieve. The use of position from SPARK can avoid some switch of controller input, but it introduces a delay to filter and fusion inputs.

The figure 10 shows the result of the approximation of a spiral trajectory. The original spiral was made using inkscape[3], so it is defined by a series of Bézier curves and a motion law defined as a one dimensional point-to-point trajectory. The error of approximation is depicted in figure 11.

[1]`http://www.openrobots.org/wiki/genom`
[2]`http://homepages.laas.fr/mallet/robotpkg/`
[3]http://inkscape.org/

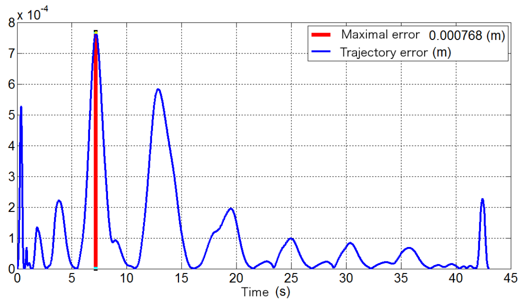


Fig. 11. The approximation error of the spiral curve.

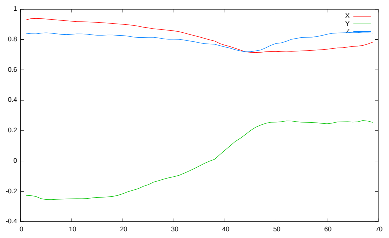


Fig. 12. A plot of the trajectory realized by the jido arm following a box handle by an human.

Figure 12 shows the trajectory executed by Jido during a task where the robot try to maintain its hand in a pre-defined position relative to a box. The trajectory was recorded when a human moved the box in front of the robot. The position of the box is obtained by stereovision tracking a tag plotted on the box.

The figure 13 shows Jido writing the word Dexmart. The trajectory was defined by a path drawn using inkscape and then approximated using the results of the section IV. The input trajectory is traveled at constant velocity. The Soft Motion approximation stops at cusp. The controller uses impedance control to maintain a constant force in the normal direction.

The figure 14 shows a trajectory build from a polygonal line to grasp an object.

A significant advantage of using trajectory controller is that the controller can work with different frequencies. For example, the position can be measured every 0.1s by vision and a low level controller that uses an impedance controller can run at 1kHz. This possibility simplifies the design of the controllers.

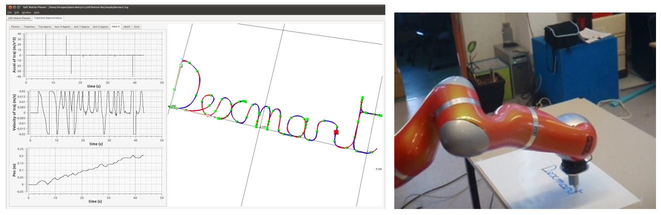


Fig. 13. Jido writing along a approximated trajectory.

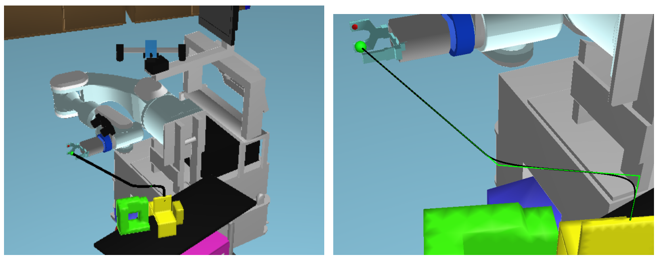


Fig. 14. Left: A simulated move to a gripper grasp an object. Right: The detail of the path defining the trajectory and the initial polygonal line.

VIII. Conclusion

In this paper we have presented a set of trajectory tools to animate machines. A first interesting point is the proposal to use polynomial third degree trajectories (Soft Motions trajectories) because they define a simple and powerful class of trajectories. We proposed a set of trajectory generators that can be used to effectively build Soft Motions trajectories.

From these tools we described how to generate trajectories at planning level and then how to control a robot in several situations.

We can conclude with the hope to build a robot based on Soft Motion trajectories. This robot will embed a trajectory planner like the one we outlined but it should associate to each generated trajectories the frames in which the move must be controlled and the condition to switch between controllers. We have proposed a set of basic controllers; this set must be enlarged with force controllers. A tool to switch between these controllers and manage the state of this meta-controller should be defined and built.

Lastly, the coordination of the motions of a full robot (base, arms, hands, head) or of two or more robots is also very challenging, trajectories could help to synchronize this motions.

References

[1] T. Kröger, *On-Line Trajectory Generation in Robotic Systems*, 1st ed., ser. Springer Tracts in Advanced Robotics. Berlin, Heidelberg, Germany: Springer, jan 2010, vol. 58.
[2] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Springer, 2008.
[3] X. Broquère, D. Sidobre, and I. Herrera-Aguilar, "Soft motion trajectory planner for service manipulator robot," in *IEEE/RSJ Int. Conf. on Intel. Rob. And Sys.*, 2008.
[4] X. Broquère, "Planification de trajectoire pour la manipulation d'objets et l'interaction homme-robot," Ph.D. dissertation, LAAS-CNRS and Université de Toulouse, Paul Sabatier, 2011.
[5] X. Broquère and D. Sidobre, "From motion planning to trajectory control with bounded jerk for service manipulator robots," in *IEEE Int. Conf. Robot. And Autom.*, 2010.
[6] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, 2010.
[7] S. M. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Workshop on the Algorithmic Foundations of Robotics*, 2001.
[8] S. Fleury, M. Herrb, and R. Chatila, "Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture," in *IEEE/RSJ Int. Conf. on Intel. Rob. And Sys.*, 1997.
[9] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Siméon, "Human aware mobile robot motion planner," *IEEE Transactions on Robotics*, 2007.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Open-loop and closed-loop filters for optimal trajectory planning under time and frequency constraints

Luigi Biagiotti, Roberto Zanasi

***Abstract*— Dynamic filters for real-time trajectory generation can be designed in different ways with quite different levels of performances and complexity. However, one can observe that the configurations of such filters are based on two main schemes: systems composed of a chain of integrators with a feedback control and systems formed of a sequence of linear filters disposed in a cascade configuration. According to these schemes, it is possible to obtain minimum-time trajectories under constraints of velocity, acceleration, jerk and even higher derivatives. In both cases, the degree of continuity of the resulting trajectory depends on the order n of the filter, which can be designed according to a modular approach.**
After a short overview of the structure of the trajectory generators, pro and cons of the two approaches are analyzed. In particular, the attention is focussed on linear filters, since their structure allows a straightforward characterization of the trajectory from a frequency point of view. As a consequence, the generator can be designed by taking into account frequency constraints, besides more standard time constraints (i.e. limits on velocity, acceleration, etc.). The proposed method combines the advantages of minimum-time trajectories with those of input shaping techniques. Moreover, it is possible to prove that, under additional hypotheses, the same chain of linear filters proposed for minimum-time trajectories generation can be used for obtaining uniform B-spline curves, that are widespread in the robotic field when the interpolation of a set of given via-points is required. In this case, the additional constraints do not allow to impose limits on the velocity or acceleration, but only to properly shape the trajectory in the frequency domain. It is therefore possible to select the trajectory/filter parameters with the purpose of suppressing residual vibrations, that may be present because elastic phenomena affecting the robotic system.

I. Introduction

Online generation of trajectories subject to kinematic constraints (on velocity, acceleration, jerk, etc.) plays a central role in all those applications where the motions cannot be planned a priori and must be optimized with respect to the time. Robotic systems are probably the most important example of such applications because their flexibility and the complexity of the required movements. For this reason, a large number of papers addressing this problem is available in the scientific literature tied to the robotic field, both for single-axis applications and for multi-axis motions. With respect to this problem, several filters based on control theory have been proposed, see e.g. [1], [2], [3], [4] among many others. These trajectory generators are based on two opposite design philosophies, i.e. closed-loop and open-loop approaches. Both allow to obtain minimum-time trajectories compliant with given limits on velocity, acceleration, jerk, etc., by specifying in runtime the desired final position. Additionally, the structure of the dynamic filters has relevant implications on the spectral content of the motion profile and can be properly modified in order to take into account frequency specifications and not only time-domain constraints. As a matter of fact, the need of high velocities often leads to the excitation of eigenfrequencies of the machines/robots caused by structural flexibilities and may produce vibrations and large tracking errors. For this reason, a number of works copes the problem of filtering preplanned trajectories in order to reduce residual vibrations. The available methods range from low-pass and notch filters to input shaping techniques, see [5] for a comparative overview, but only recently an online generator, based on a chain of linear filters, that combines the advantages of minimum-time trajectories with those of shaping techniques has been proposed [6].
With some additional constraints on the free parameters of the filters, it is possible to show that open-loop generators for minimum-time trajectories share the same structure of generators for B-splines curves which are extensively used in robotics in order to define smooth trajectories crossing a set of given via-points. Therefore, the considerations and the techniques used for properly shaping the spectral content of minimum-time motions can be extended to this class of curves.
The paper is organized as follows. In Sec. II the main concepts tied to time-optimal trajectories and dynamic filters for online trajectory generation are presented, and a general overview on closed-loop and open-loop filters is provided. Then, in Sec. III the two different types of trajectory generators are compared with respect to the different problems that they can solve (point-to-point optimal trajectory generation, smoothing of pre-planned trajectories, etc.). In Sec. IV open-loop filters are analyzed in the frequency domain and their parameters are set with the purpose of properly shaping the spectrum of the output trajectory. Similar considerations are reported in Sec. V with respect to B-spline trajectory generation. Concluding remarks are reported in the last section.

L. Biagiotti and R. Zanasi are with the Department of Engineering “Enzo Ferrari” (DIEF), University of Modena and Reggio Emilia, Via Vignolese 905, 41125 Modena, Italy, e-mail: {luigi.biagiotti, roberto.zanasi}@unimore.it.

II. Optimal trajectories and dynamic filters

The optimization process of trajectories subject to constraints on velocity, acceleration, jerk, etc., leads to the so-called multi-segment trajectories, i.e. trajectories composed by several tracts properly joined, each one characterized by a specific analytical expression, and in which the velocity, the

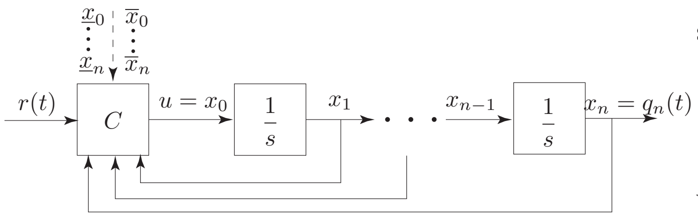


Fig. 1. Structure of a closed-loop trajectory filter of n-th order.

acceleration, or higher derivatives (depending on the required order of continuity) are saturated to the maximum allowed value. By imposing constraints on the first n derivatives, i.e.

$$q_{min}^{(i)} \leq q_n^{(i)}(t) \leq q_{max}^{(i)}, \qquad i = 1, \ldots, n \tag{1}$$

one obtains a trajectory $q_n(t)$ of class $\mathcal{C}^{n-1}$, that is with the first $n-1$ derivatives that are continuous, while the n-th derivative $q_n^{(n)}(t)$ is a piece-wise constant function whose values belong to a set $\{q_{min}^{(n)}, 0, q_{max}^{(n)}\}$. The number n is called order of the trajectory. The dynamic filters for trajectory planning generate on-line a time optimal trajectory $q_n(t)$ that tracks at best a reference signal $r(t)$, satisfying desired constraints on the first n derivatives of $q_n(t)$. The reference signal $r(t)$ is generally given by a first coarse trajectory generator providing for instance a piecewise constant profile which defines the desired final positions, or is an external input, given for example by a human operator.

The trajectory planners based on feedback regulation are composed of a chain of n integrators and a nonlinear controller able to nullify in minimum time the tracking error between the reference input $r(t)$ and the integrators output $x_n(t) = q_n(t)$, being compliant with constraints (1), see Fig. 1. In [7] a modular solution is proposed. According to this approach, the n-th order trajectory planner is designed around the filter of order $n-1$, as illustrated in Fig. 2. The

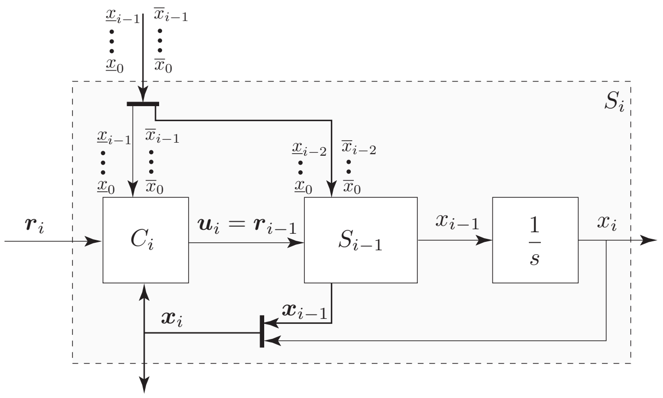


Fig. 2. Structure of the i-th control loop ($S_0 = 1$) of the modular closed-loop trajectory generator.

structure of the i-th controller is

$$C_i: \quad u_i = \begin{cases} \overline{x}_{i-1}, & \text{if } y_i \leq h_i(y_{i-1}, y_{i-2}, \ldots, y_1) \\ 0, & \text{if } y_{i-1} = y_{i-1} = \ldots = y_1 = 0 \\ \underline{x}_{i-1}, & \text{if } y_i \geq h_i(y_{i-1}, y_{i-2}, \ldots, y_1) \end{cases} \tag{2}$$

where x_i is the output of the filter, that is the position trajectory, r_i is the reference signal that denotes the target position and $y_i = x_i - r_i$ is the tracking error. The parameters $\overline{x}_{i-1}$, $\overline{x}_{i-1}$ denote the limits on the first derivative of x_i. While the structure of C_i, which is based on variable structure control, is rather simple, the expression of the function $h_i(\cdot)$ that appears in (2) is very complicated also for small values of the index i, see [7] for the detailed expression. Moreover, the computation of $h_i(\cdot)$ for $i > 3$ is critical and has not been performed yet. Therefore, at the moment, only second- or third-order trajectories may be generated according to this approach. Finally, the digital implementation of the trajectory generator starting from its continuous-time expression is not straightforward and cannot be obtained by a simple discretization of the integrators chain since the filter will be certainly affected by chattering. For this reason, ad hoc solutions are required.

Open-loop trajectory generators are characterized by a very simple structure which allows a generalization up to whatever order n with only a little increase of complexity and computational burden. As a matter of fact, a multi-segment trajectory of order n can be obtained by filtering a step input with a cascade of n dynamic filters, each one characterized by the transfer function

$$M_i(s) = \frac{1}{T_i}\frac{1 - e^{-sT_i}}{s} \tag{3}$$

where the parameter T_i (in general different for each filter composing the chain) is a time length, see Fig. 3. In mathematical terms, this means that

$$q_n(t) = h \cdot u(t) * m_1(t) * m_2(t) * \ldots * m_n(t) \tag{4}$$

where $u(t)$ denotes the unit step function, h is the desired displacement and $m_i(t) = \mathcal{L}^{-1}\{M_i(s)\}$ is the impulse response of each filter. The parameters T_i can be selected with the purpose of imposing desired bounds on velocity, acceleration, jerk and higher derivatives, i.e.

$$|q_n^{(i)}(t)| \leq q_{max}^{(i)}, \qquad i = 1, \ldots, n \tag{5}$$

by assuming

$$\begin{aligned} T_1 &= \frac{|h|}{q_{max}^{(1)}} \\ T_i &= \frac{q_{max}^{(i-1)}}{q_{max}^{(i)}}, \quad i = 2, \ldots, n \end{aligned} \tag{6}$$

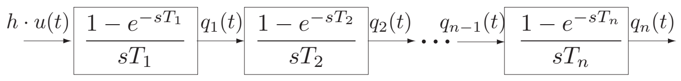

$h \cdot u(t)$
$\frac{1 - e^{-sT_1}}{sT_1}$
$q_1(t)$
$\frac{1 - e^{-sT_2}}{sT_2}$
$q_2(t)$
$q_{n-1}(t)$
$\frac{1 - e^{-sT_n}}{sT_n}$
$q_n(t)$

Fig. 3. System composed by n filters for the computation of an optimal trajectory of class $\mathcal{C}^{n-1}$.

with the additional constraints

$$T_j \geq T_{j+1} + \ldots + T_n, \qquad j = 1, \ldots, n-1. \qquad (7)$$

For more details refer to [6].
In order to evaluate the trajectory at discrete time instants kT_s, being T_s the sampling period, the system composed by n filters may be discretized by applying on each filter $M_i(s)$ the backward differences method that leads to the expression of a moving average filter

$$M_i(z) = \frac{1}{N_i}\frac{1-z^{-N_i}}{1-z^{-1}} \qquad (8)$$

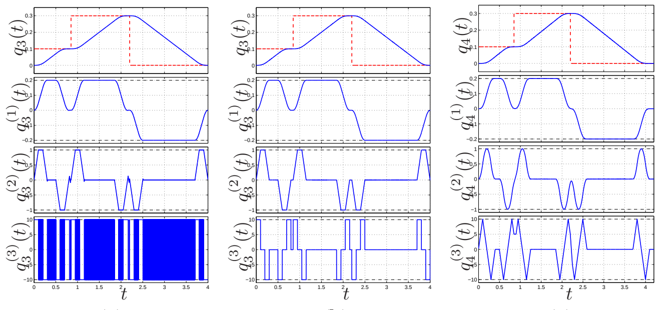
where $N_i = T_i/T_s$ is the number of samples (not null) of the filter response. Note that N_i is also equal to the number of elements composing the FIR filter (usually called *taps*) as they appear in the equivalent (nonrecursive) formulation

$$M_i(z) = \frac{1}{N_i} + \frac{1}{N_i}z^{-1} + \frac{1}{N_i}z^{-2} + \ldots + \frac{1}{N_i}z^{-N_i-1}. \quad (9)$$

The implementation of the proposed trajectory generator on a digital controller can be achieved by simply considering the function $M_i(z)$ in lieu of the corresponding function $M_i(s)$ in the block-scheme of Fig. 3. Note that the digital implementation of each filter only requires two additions and one multiplication. As a consequence, even for high values of the degree n, the trajectory generator (composed by n filters) results very efficient from a computation point of view.

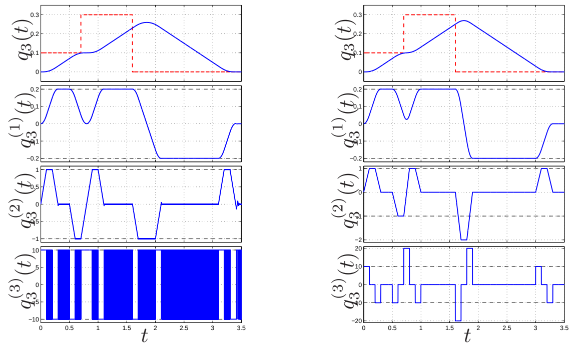
III. A COMPARISON BETWEEN CLOSED-LOOP AND OPEN-LOOP FILTERS

Besides the different structure, the two groups enjoy peculiar features that make each type of generator preferable for a specific application. Closed-loop filters are superior to analogous open-loop systems in terms of performances and flexibility, since they allow to take into account asymmetric bounds, which can even modified in runtime. Moreover, they do not require additional constraints among the filter parameters such as (7). On the other hand, they are affected by some limitations: presence of chattering superimposed to the output, high complexity of the implementation, high computational burden, maximum order $n = 3$. For these reasons the choice of a particular type of filter must be performed according to the specific application to be carried out. For instance, in standard tasks where online generation of point-to-point trajectories is required, if the desired bounds of velocity, acceleration, jerk, etc. meet the conditions (7), open-loop filters are preferable. As a matter of fact, as reported in Fig. 4(a) and Fig. 4(b) the two generators provide the same trajectory but with very different costs in terms of computational complexity. Moreover, in Fig. 4(a) one can observe the chattering in the jerk profile and the overshoots in the acceleration. Figure 4(c) shows the fourth order trajectory $q_4(t)$ obtained by applying the same stepwise input function to a chain of four running average filters. With a little increase of complexity, one can obtain the time-optimal trajectory with the desired degree of smoothness.
On the other hand, if the input function changes before



(a) (b) (c)

Fig. 4. Comparison between the output of the third order closed-loop filter (a) and those of a third order (b) and fourth order (c) open-loop filter with the same input $r(t)$ composed by step functions.

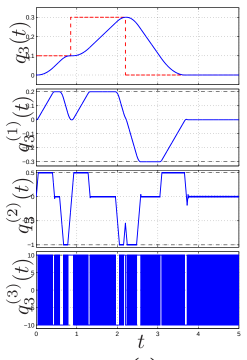


(a) (b)

Fig. 5. Comparison between the output of the third order closed-loop filter (a) and that of a cascade of three moving average filters (b) with the same input $r(t)$ composed by step functions.

that the previous tract of trajectory has ended, the open-loop generator cannot guarantee the compliance with the constraints, see Fig. 5(b). On the contrary, closed-loop filters always fulfill the prescribed constraints, as shown in Fig. 5(a). Moreover, as already mentioned, in case of feedback controlled trajectory generators, it is possible to consider asymmetric limits, i.e. $q_{max}^{(i)} \neq -q_{min}^{(i)}$, or even change the values of the bounds in runtime. In this case, the controller will act so that the new limits are satisfied in minimum time. For instance, in Fig. 6(b) the behavior of the filter is shown when the minimum value of velocity is increased from -0.2 to -0.1. As soon as this occurs, the controller modifies the velocity of the motion profile in order to meet the new limit but without violating the other constraints on acceleration and jerk. Conversely, by adopting an open-loop generator it is not possible to change the constraints during the planning of a trajectory, since this would require a structural modification of the filters composing it, leading to discontinuities of the response. In fact, the number of taps of the FIR filters is directly related to the desired bounds by means of (6).

Finally, both closed- and open-loop filters can be used to modify pre-planned trajectories with the purpose of making them compliant with the desired bounds and not only to generate point-to-point multi-segment trajectories. Also in this case the two types of dynamic systems behave quite


(a)

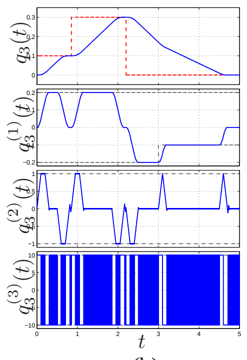
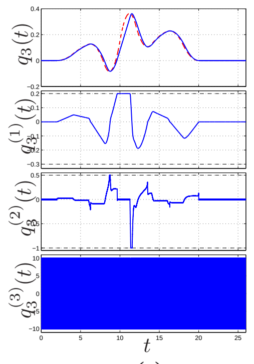

(b)

Fig. 6. Response of the third order closed-loop filter to a stepwise input function under asymmetric constraints of velocity and acceleration (a) and with a variation of the bound on the velocity (b).


(a)

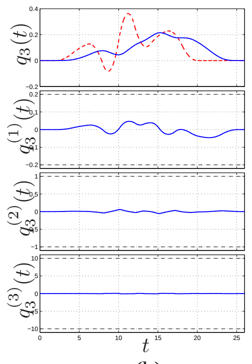
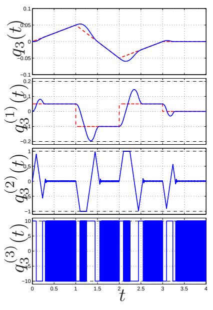
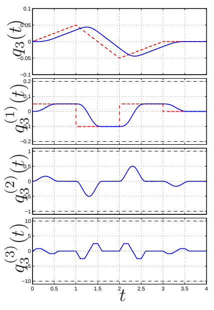

(b)

Fig. 8. Comparison between the output of the third order closed-loop filter (a) and that of a cascade of three moving average filters (b) with a generic input $r(t)$.


(a)


(b)

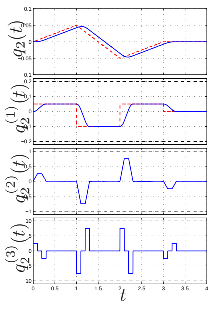

(c)

Fig. 7. Comparison between the output of the third order closed-loop filter (a) and those of a third order (b) and a second order (c) open-loop filter with the same input $r(t)$ composed by ramp functions.

differently. In Fig. 7 the responses to a reference input $r(t)$ composed by ramp functions (therefore with constant velocity and impulsive acceleration) are reported. The closed-loop filters tries to reach in minimum-time and then to track the reference $r(t)$, satisfying the given constraints. On the contrary, the third order open-loop filter produces a smoother but delayed version of the input. Note that the output signal reaches the constant velocity of the ramps but that all the other constraints are not reached. Moreover, since the input signal is continuous the resulting trajectory has continuous jerk. Therefore it is possible to consider a lower order trajectory filter (second-order), with the parameters

$$T_1 = \frac{q_{max}^{(1)}}{q_{max}^{(2)}}, \qquad T_2 = \frac{q_{max}^{(2)}}{q_{max}^{(3)}}$$

to assure that all the constraints are satisfied. In this way, the total (constant) delay between the input and the output is reduced, being $T_{tot} = T_1 + T_2$, see Fig. 7(c).
Although the design of the third order closed-loop filter is based on the hypothesis $r^{(4)} = 0$, the variable structure control allows to consider generic input signals $r(t)$ that do not fulfill this condition. In Fig. 8(a) the response of the filter to a generic input is illustrated. When the desired bounds are met the output follows exactly the input but when the reference overcomes such values the filter limits the output. In this case the reference input may represent the commands provided by an external system or a human operator, and the trajectory filter can be profitably exploited to make them compatible with the physical limits that are present in any plant.
On the contrary open-loop filters behave like a standard low-pass filter, and therefore if the input signal is "too fast" the output will result quite deformed (and delayed), see Fig. 8(b). For this reason, the chain of FIR filters is not suitable for an application to generic inputs, but on the other hand, its filtering properties makes this kind of generators quite attractive when frequency specifications must be taken into account [8].

IV. Frequency Characterization of Open-Loop Filters and Spectral Shaping of the Trajectory

The spectrum of the trajectory planned with an open-loop filter can be readily deduced by considering its expression in terms of Laplace transform (directly obtained from (4)), i.e.

$$Q_n(s) = \frac{h}{s} \cdot M_1(s) \cdot M_2(s) \cdot \ldots \cdot M_n(s). \tag{10}$$

As a matter of fact, as it is well known, the Fourier transform of $q_n(t)$ immediately descends from $Q_n(s)$, being the restriction to the imaginary axis, i.e. $Q_n(j\omega)$. Therefore, the closed-form expression of $Q_n(j\omega)$ is given by the products of the Fourier signal corresponding to the input $h\,u(t)$ and of the frequency responses of the filters composing the trajectory generator:

$$Q_n(j\omega) = \frac{h}{j\omega} \cdot M_1(j\omega) \cdot M_2(j\omega) \cdot \ldots \cdot M_n(j\omega)$$

where

$$\begin{aligned} M_i(j\omega) &= \frac{1}{T_i}\frac{1 - e^{j\omega T_i}}{j\omega} \\ &= e^{-j\frac{\omega T_i}{2}} \frac{\sin\left(\frac{\omega T_i}{2}\right)}{\frac{\omega T_i}{2}}. \end{aligned} \tag{11}$$

Since the frequency characterization of the trajectory, including its derivatives is a useful tool to predict vibratory phenomena in the systems to which the trajectory is applied

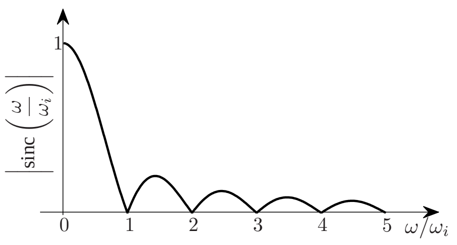


Fig. 9. Magnitude of the frequency response of the filter $M_i(s)$.

[9], it is necessary to obtain the expression of the spectrum of the generic k-th derivative of $q_n(t)$. Because of the properties of Laplace transforms, this result is straightforward. As a matter of fact, the Laplace transform of $q_n^{(k)}(t)$ is given by

$$Q_n^{(k)}(s) = s^k Q_n(s)$$

and therefore the expression of the spectrum of $q_n^{(k)}(t)$ is

$$\begin{aligned} Q_n^{(k)}(j\omega) &= (j\omega)^k \cdot Q_n(j\omega) \\ &= h\cdot (j\omega)^{k-1}\cdot M_1(j\omega)\cdot M_2(j\omega)\cdot \ldots \cdot M_n(j\omega). \end{aligned}$$

In conclusion, the amplitude spectrum of $q_n(t)$ and its derivatives, i.e. $|Q_n^{(k)}(j\omega)|$, is given by the product of two main elements:

- a power of ω, i.e. ω^{k-1}, being k the order of the derivative;
- the (magnitude of the) frequency response of the chain of n filters $M_i(s)$.

The frequency response of the cascade of filters is the product of the single frequency responses $M_i(j\omega)$, $i = 1,\ldots,n$, whose magnitude is

$$|M_i(j\omega)| = \left|\frac{\sin\left(\frac{\omega T_i}{2}\right)}{\frac{\omega T_i}{2}}\right| = \left|\text{sinc}\left(\frac{\omega}{\omega_i}\right)\right|$$

where sinc$(\cdot)$ denotes the normalized sinc function defined as sinc$(x) = \frac{\sin(\pi x)}{\pi x}$ and $\omega_i = \frac{2\pi}{T_i}$. Note that the function $|M_i(j\omega)|$, shown in Fig. 13, is equal to zero for $\omega = k\,\omega_i$, with k integer. This property can be profitably exploited to properly choose the parameters of the trajctory/filter with the purpose of nullifying the spectrum of the trajectory at critical frequencies, for instance the eigenfrequencies of the plant. For this aim, if ω_r denotes a resonant frequency, it is sufficient to assume

$$\omega_i = \frac{\omega_r}{l} \quad \Leftrightarrow \quad T_i = l\,\frac{2\pi}{\omega_r}, \quad l = 1, 2, \ldots. \qquad (12)$$

This result generalizes what has been presented in [10] where, with reference to a double S velocity trajectory, it is recognized that in order to suppress residual vibrations due to the dominating vibratory mode of an axis of motion it is necessary to assume that the duration of the "jerk period" (in which the jerk remains constant) equals a multiple of the natural period of the vibrational mode. According to (12) the reduction of residual vibrations caused by resonant frequencies of the plant can be achieved with multi-segment trajectories of any order provided that the time constant T_i

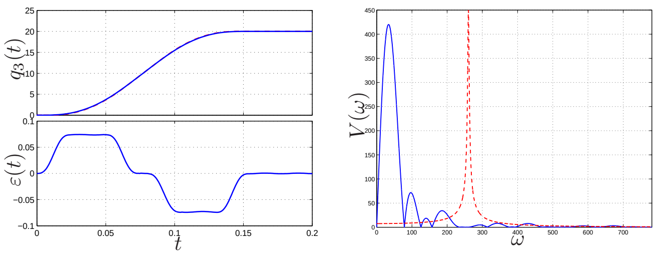


Fig. 10. Response of the elastic system $G_{ml}(s)$ to a double S velocity trajectory obtained with $T_1 = |h|/q_{max}^{(1)}$, $T_2 = q_{max}^{(1)}/q_{max}^{(2)}$ and $T_3 = T_0$: tracking error (a) and frequency spectrum of the acceleration (b).

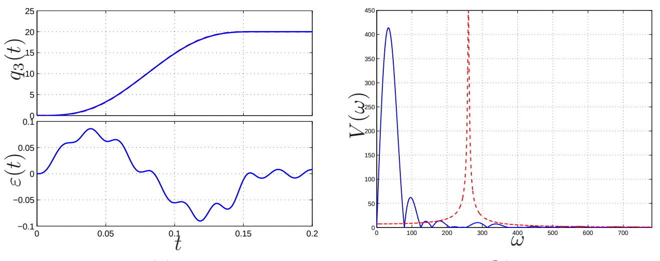


Fig. 11. Response of the elastic system $G_{ml}(s)$ to a double S velocity trajectory obtained with $T_1 = |h|/q_{max}^{(1)}$, $T_2 = q_{max}^{(1)}/q_{max}^{(2)}$ and $T_3 = T_1 - T_2$: tracking error (a) and frequency spectrum of the acceleration (b).

of a filter $M_i(s)$ is l times, l integer, the dominating natural period $T_0 = \frac{2\pi}{\omega_r}$.

For instance, if a standard motion system composed of two inertias with an elastic transmission lightly damped [11], [12], [13] is considered, some parameters of the trajectory generator can be selected with the purpose of minimizing frequency components about the resonance of the system. In Fig. 10 and Fig. 11, the responses of the system to two different third-order trajectories are reported, and in particular residual vibrations $\varepsilon(t)$ are analyzed. When the parameter T_3 of the third filter of the chain is assumed equal to T_0 vibrations at the end of motion are canceled, see Fig. 10. On the contrary, if the resonant frequency ω_r is not considered, the motion system may be affected by residual vibrations as shown in Fig. 11. Note that in this example, frequency constraints are taken into account together with time-constraints on velocity and acceleration. For more details refer to [6].

V. B-spline filters

An interesting property of the filter of Fig. 3 is the possibility of generating online B-spline trajectories. As a matter of fact, in [14], it has been shown that uniform B-splines[1] of degree p can be computed by feeding a cascade of p filters

$$M(s) = \frac{1}{T}\frac{1 - e^{-sT}}{s}, \qquad (13)$$

[1]Uniform B-spline are defined as

$$\mathbf{s}_u^p(t) = \sum_{j=0}^{n} \boldsymbol{p}_j B^p(t - jT), \qquad 0 \le t \le (m-1)T,$$

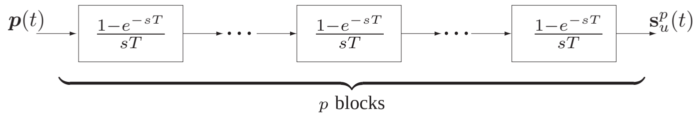


Fig. 12. System composed by p filters for the generation of trajectories based on uniform B-splines starting from the sequence of the control points $\boldsymbol{p}_j$.

shown in Fig. 12, with the function

$$\boldsymbol{p}(t) = \sum_{j=0}^{n} \boldsymbol{p}_j B^0(t - jT)$$

where $\boldsymbol{p}_j$ are the control point defining the B-spline curve (for their computation refer to [14] and [15]) and $B^0(t)$ is a rectangular function defined as

$$B^0(t) = \begin{cases} 1, & \text{if } \ 0 \le t < T \\ 0, & \text{otherwise.} \end{cases}$$

By comparing the trajectory generator of Fig. 3 with that of Fig. 12, one can immediately infer that the two systems are equivalent provided that $T_1 = T_2 = \ldots = T_n = T$. Therefore, the two filters enjoy the same properties and, in particular, both allow to precisely characterize the motion profiles from a frequency point of view.

In terms of Laplace transform, uniform B-splines based on linear filters can be written as

$$\begin{aligned} \boldsymbol{S}_u^p(s) &= \mathcal{L}\left\{\sum_{j=0}^{n} \boldsymbol{p}_j B^0(t - jT)\right\} \cdot \underbrace{M(s) \cdot M(s) \cdot \ldots \cdot M(s)}_{p \text{ filters}} \\ &= \qquad\qquad \boldsymbol{P}(s) \qquad\qquad \cdot \qquad\qquad M^p(s) \end{aligned} \tag{14}$$

Therefore, the closed form expression of B-spline spectrum, that is $\boldsymbol{S}_u^p(j\omega)$, is given by the products of the Fourier signal corresponding to the input $\boldsymbol{p}(t)$ and the frequency response of the filters composing the trajectory generator:

$$\boldsymbol{S}_u^p(j\omega) = \boldsymbol{P}(j\omega) \cdot M^p(j\omega).$$

Note that in this case, all the filters composing the chain have the same frequency response since they depend on a unique free parameter T. The magnitude of the frequency response of the cascade of p filters, shown in Fig. 13 for $p = 1, 2, 3, 4$, is given by

$$|M^p(j\omega)| = \left|\text{sinc}\left(\frac{\omega}{\omega_0}\right)\right|^p$$

where $\omega_0 = \frac{2\pi}{T}$. Like in case of multi-segment trajectories, the function $|M^p(j\omega)|$ is equal to zero for $\omega = k\,\omega_0$. Moreover, by augmenting the value of the degree p, the spectral components that follows the frequency ω_0 are considerably reduced. These features of the spectrum of $M^p(s)$ can be profitably exploited to properly choose the free parameter of the filter/B-spline trajectory, that is the time-distance T between the knots, with the purpose of decreasing or even nullifying the spectrum of the trajectory at critical frequencies, for instance the eigenfrequencies of the robotic manipulator. These features make trajectory planning based on uniform B-spline very similar to input shaping techniques, consisting in filtering the reference commands by convolving them with a train of impulses in order to form new commands that cause little or no vibrations [16], [17]. For this reason, it may be useful to compare the two kind of filters with respect to the Percent Residual Vibration (PRV) that they produce on vibratory systems. In Fig. 14, the PRV obtained by filtering the command input with $M^3(s)$ (that corresponds to a cubic B-spline) is compared with the PRV related to standard Input Shapers (IS), that is Zero Vibration (ZV) IS, Zero Vibration and Derivative (ZVD) IS and so on, whose expression for an undamped system is

$$C(s) = \left(\frac{1 + e^{-s\frac{\pi}{\omega_0}}}{2}\right)^p$$

where $p = 1$ corresponds to a ZV IS, $p = 2$ to a ZVD IS, etc. The levels of vibration in the neighborhood of $\omega_n = \omega_0$ (being ω_n the natural frequency of the system that tracks the trajectory) for $M^3(s)$ and ZVDDD are comparable, but it is worth noticing that the PRV characteristics of the cubic B-spline filter is strongly asymmetric. This considerably augments the robustness of the filter with respect to errors in the estimation of ω_n. The insensitivity of $M^p(s)$, that is the width of the frequency range where the PRV curve is below a tolerable vibration level [17], may be easily made infinite. As a matter of fact, given a desired maximum level of vibrations V_{tol}, it is sufficient to chose the degree p large enough to guarantee that PRV $\le V_{tol}$, $\forall \omega_n \ge \omega_0$. This result limits the minimum duration of the B-spline curve: given a vibratory system for which $\omega_n \ge \omega_{n,min}$, the residual vibrations may be made arbitrarily small (by acting on p) only if

$$T \ge \frac{2\pi}{\omega_{n,min}}.$$

This is very important for applications involving robots with elastic elements, since the natural frequency is not constant but it is a function of the configuration $\boldsymbol{q}$ [18], [19]. In this

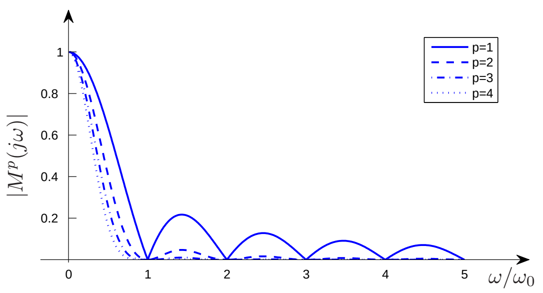


Fig. 13. Magnitude of the frequency response of the chain of filters $M^p(s)$ used for generating uniform B-spline trajectories of degree p.

where the vectorial coefficients $\boldsymbol{p}_j$, $j = 0, \ldots, m$, called *control points*, determine the shape of the curve, $B^p(t)$ are B-spline basis functions of degree p, and T denotes the (constant) time-distance between successive knots, i.e. $t_{j+1} - t_j = T,\ j = 0, \ldots m - 2$.

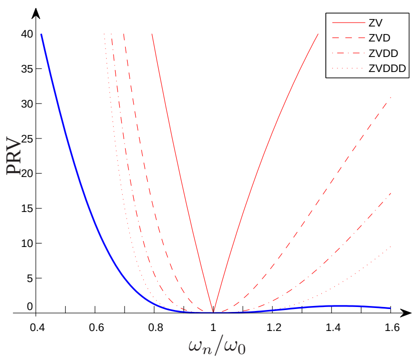


Fig. 14. Percent Residual Vibration of the cubic B-spline filter $M^3(s)$ (thick solid blue line) compared with those obtained with standard input shapers.

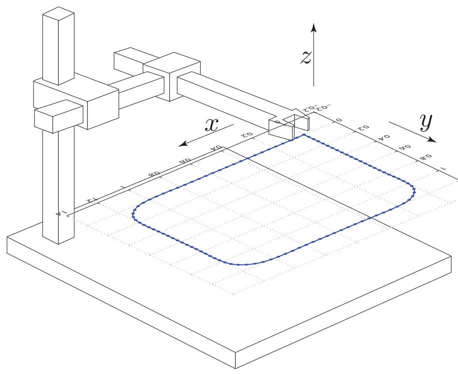


Fig. 15. Cartesian Robot tracking a planar trajectory defined by a set of via-points.

case, one must simply assume

$$T \geq \frac{2\pi}{\min_{\boldsymbol{q}}\{\omega_n(\boldsymbol{q})\}}.$$

In Fig. 16 and Fig. 17 the tracking of a trajectory obtained with a B-spline filters and Input Shapers are compared. In particular, the cartesian robot of Fig. 15 characterized by joints with elastic transmissions (that lead to natural frequencies $\omega_n = 60$ rad/s along x and z axes and $\omega_n = 40$ rad/s in the y direction) is considered. In both cases the level of residual vibrations, especially along straight-line segments, is very low but it is worth noticing that also in this ideal case IS are not able to guarantee the exact interpolation of the given via-points, while B-spline trajectories interpolate the desired points. More details about frequency characterization of B-spline trajectories/filters and their use in robotic applications can be found in [20].

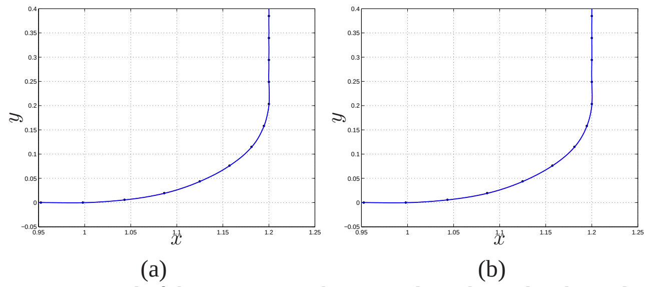


Fig. 16. Detail of the trajectory in the $x-y$ plane obtained with B-spline filters of degree $p = 3$ (a) and $p = 5$ (b) (designed with $\omega_0 = 50$ rad/s).

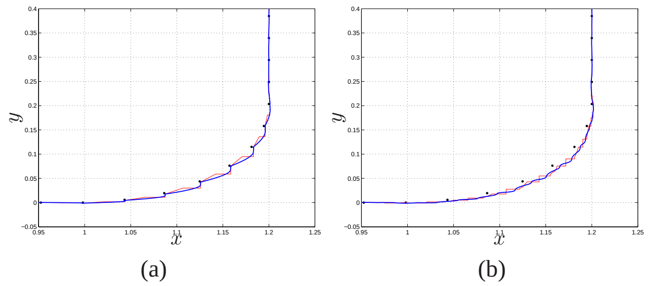


Fig. 17. Detail of the trajectory in the $x-y$ plane obtained by filtering the via-points with ZV IS (a) and ZVDD IS (b)(designed with $\omega_{0,x} = 60$ rad/s and $\omega_{0,y} = 40$ rad/s).

VI. Conclusions

Two different design philosophies for online trajectory planners (namely open-loop and closed-loop approaches) have been analyzed. Besides their different structure, the two groups enjoy peculiar features that make each type of generator preferable for a specific application. In particular, open-loop systems have a very simple structure and imply a computational cost lower than similar closed-loop generators. Because of their simple structure, they can be easily generalized in order to implement high order trajectories, with continuous jerk or even higher derivatives. On the other hand, this kind of trajectory planners suffers from some limitations that can be overcome by adopting closed-loop filters. As a matter of fact, filters with a cascade configuration only work with symmetric constraints on velocity, acceleration, etc.. Moreover, the desired bounds cannot be changed in runtime and a new trajectory cannot start before the current motion profile has executed without violating the desired limit values. On the contrary, closed-loop trajectory generators allow to modify the limits in runtime and to start a new motion at any time. However open-loop filters may be preferable in all those applications that are affected by vibrations and resonances, since the linear structure allows a precise frequency characterization of the output trajectory. In the same manner, filters for uniform B-splines generation, that are characterized by a similar structure, can be designed with the purpose of properly shaping their frequency spectrum. Therefore, by selecting the trajectory/filter parameters it is possible to suppress residual vibrations, that may be present because elastic phenomena affecting the robotic system. The effectiveness of the proposed approach is demonstrated by applying it for the generation of a 3D trajectory to be tracked by a cartesian robot with elastic joints.

REFERENCES

[1] R. Zanasi, C. G. L. Bianco, and A. Tonielli, "Nonlinear filter for the generation of smooth trajectories," *Automatica*, vol. 36, pp. 439–448, 2000.

[2] R. Zanasi and R. Morselli, "Discrete minimum time tracking problem for a chain of three integrators with bounded input," *Automatica*, vol. 39 (9), pp. 1643–1649, 2003.

[3] O. Gerelli and C. Guarino Lo Bianco, "Real-time path-tracking control of robotic manipulators with bounded torques and torque-derivatives," in *IROS*, 2008, pp. 532–537.

[4] C. Zheng, Y. Su, and P. Muller, "Simple online smooth trajectory generations for industrial systems," *Mechatronics*, vol. 19, pp. 571–576, 2009.

[5] N. Singer, W. Singhose, and W. Seering, "Comparison of filtering methods for reducing residual vibration," *European Journal of Control*, vol. 5, pp. 208–218, 1999.

[6] L. Biagiotti and C. Melchiorri, "Fir filters for online trajectory planning with time- and frequency-domain specifications," *Control Engineering Practice*, p. In Press, 2012.

[7] L. Biagiotti and R. Zanasi, "Time-optimal regulation of a chain of integrators with saturated input and internal variables: An application to trajectory planning," in *Time-Optimal Regulation of a Chain of Integrators with Saturated Input and Internal Variables: An Application to Trajectory Planning*, Bologna, Italia, 1-3 September 2010 2010.

[8] L. Biagiotti and C. Melchiorri, "Online planning of multi-segment trajectories with trigonometric blends," in *18th IFAC World Congress*, Milano, Italy, August 28 - September 2 2011. [Online]. Available: http://www.ifac-papersonline.net/Detailed/50873.html

[9] ——, *Trajectory Planning for Automatic Machines and Robots*. Springer Berlin Heidelberg, 2008.

[10] A. Olabi, R. Béarée, O. Gibaru, and M. Damak, "Feedrate planning for machining with industrial six-axis robots," *Control Engineering Practice*, vol. 18, no. 5, pp. 471–482, 2010.

[11] P. Lambrechts, M. Boerlage, and M. Steinbuch, "Trajectory planning and feedforward design for electromechanical motion systems," *Control Engineering Practice*, vol. 13, pp. 145–157, 2005.

[12] P.-J. Barre, R. Béarée, P. Borne, and E. Dumetz, "Influence of a jerk controlled movement law on the vibratory behaviour of high-dynamics systems," *Journal of Intelligent and Robotic Systems*, vol. 42, pp. 275–293, 2005.

[13] P. Meckl and P. Arestides, "Optimized s-curve motion profiles for minimum residual vibration," in *Proceedings of the American Control Conference*, Philadelphia, Pennsylvania, 1998, pp. 2627–2631.

[14] L. Biagiotti and C. Melchiorri, "B-spline based filters for multi-point trajectories planning," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 3-8 2010.

[15] ——, *Trajectory Planning for Automatic Machines and Robots*, 1st ed. Heidelberg, Germany: Springer, 2008.

[16] N. C. Singer and W. P. Seering, "Preshaping command inputs to reduce system vibration," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 112, pp. 76–82, 1990.

[17] J. Vaughan, A. Yanoa, and W. Singhose, "Comparison of robust input shapers," *Journal of Sound and Vibration*, vol. 315, no. 4-5, pp. 797–815, 2008.

[18] J. A. Somolinos, V. Feliu, and L. Snchez, "Design, dynamic modelling and experimental validation of a new three-degree-of-freedom flexible arm," *Mechatronics*, vol. 12, no. 7, pp. 919 – 948, 2002.

[19] A. D. Luca and W. Book, *Springer Handbook of Robotics*. Berlin: Springer Verlag, 2008, ch. Robots with Flexible Elements, pp. 287–317.

[20] L. Biagiotti and C. Melchiorri, "Input shaping via b-spline filters for 3-d trajectory planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, California, September 25-30 2011.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Third Order System for the Generation of Minimum-Time Trajectories with Asymmetric Bounds on Velocity, Acceleration, and Jerk

Corrado Guarino Lo Bianco and Fabio Ghilardelli

Abstract— **Reference signals used to drive feedback control loops are often evaluated on-the-fly on the basis of the systems operating conditions. As a consequence, it is very difficult to guarantee a priori properties like continuity or the existence of bounds on the signal dynamics and, therefore, control systems performances could deteriorate. Improved answers can be obtained if input signals are properly smoothed. The paper proposes a possible filtering system which output mimics at best any given input signal, compatibly with some smoothness requirements. In particular, generated signals are continuous up to the second time derivative and their first three time derivatives are constrained between assigned bounds. Differently from analogous solutions proposed in the past, it also handles asymmetric bounds.**

I. Introduction

The behavior of control systems is affected by the characteristics of their input signals. It is well known that, generally, system performances improve when smooth inputs are used. For this reason, when possible, references which are continuous up to the *n*th time derivative are adopted. Furthermore, system electro-mechanic limitations often impose bounds on the maximum allowable values of such derivatives: Tracking is lost every time bounds are violated. For these reasons reference signals that admit bounded first, second and third time derivatives are commonly used in industrial applications. Unfortunately, in many practical cases, driving signals derive from external control loops or depend on events that cannot be predicted in advance. In all these cases, smoothness cannot be guaranteed a priori and, conversely, discontinuities could easily appear. In order to avoid possible problems, rough signals are typically filtered by means of real-time planning systems, that replace them with trajectories that are characterized by the required degrees of smoothness.

Several online planners have been proposed in the literature, all of them characterized by minimum-time transients. They can be divided into two main families. In the first family, trajectories are planned by means of appropriate decision trees. In [1], in a robotic context and for continuous time frameworks, step reference signals were interpolated by means of trajectories characterized by bounded velocities, accelerations and jerks. The study was continued, for a multidimensional problem, in [2] by considering variable reference signals by fulfilling given constraints on velocities and accelerations. In [3], still for a multidimensional case, optimal online trajectories were generated for step reference signals by also managing constraints on the maximum jerk. The discrete-time solution recently devised in [4] extends previous results by also admitting generic reference signals. In the same paper an useful classification for the online trajectory planners is proposed.

In the second family, trajectories are obtained by means of dynamic filters. Such filters are constituted by a chain of integrators that are driven with variable structure sliding-mode controllers. They are able to manage generic reference signals, while output signals are still characterized by minimum-time behaviors. Early works on this approach appeared in [5], [6] and in [7], [8] respectively for continuous and discrete-time frameworks. Given solutions were based on second order filters that can guarantee the fulfillment of given bounds on the velocity and the acceleration signals. Recently, in [9], a chattering suppression method has been proposed in order to use a continuous filter within a discrete-time environment. A third-order continuous-time solution, also managing bounds on the jerk, was proposed in [10], while in [11] a dicrete-time implementation that only considers jerk bounds was synthesized. Such solution has been recently improved in [12] in order to simultaneously handle velocity, acceleration and jerk limits.

Above mentioned approaches only consider symmetric bounds. However, it is possible to cite applications, like, e.g., those proposed in [13], [14], that intrinsically admit asymmetric limits. For this reason, second-order filtering schemes, that are able to handle asymmetric constraints on the velocity and the acceleration, were proposed in [15], [14], [16]. Recently, an application that also requires the imposition of asymmetric bounds on jerk has been proposed in [17].

The new discrete-time third-order filter devised in this paper is able to generate trajectories that fulfill such requirement. Moreover, like its predecessors, it is able to manage generic reference signals, its transients are minimum-time and, finally, bounds on velocity, acceleration, and jerk can be changed in real time.

The paper is organized as follows. In §II the problem is formulated and solved by means of a novel third-order discrete-time filter. Convergence properties of the filter are analyzed in §III and in §IV. A test case is proposed in §V, while §VI reports some final conclusions.

II. The optimal trajectory scaling problem and the discrete-time filter

In the following, subscript $i \in \mathbb{Z}$ indicates sampled variables acquired at time $t = iT$, where T is the system sampling

This work was partially supported by the Ateneo Italo Tedesco in the framework of a Vigoni project.

The authors are with the Dip. di Ing. dell'Informazione, University of Parma, Italy, email: {guarino,fghilardelli}@ce.unipr.it

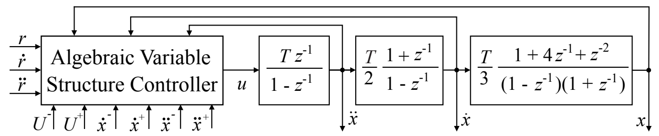

r
$\dot{r}$
$\ddot{r}$
Algebraic Variable Structure Controller
u
$\frac{T z^{-1}}{1 - z^{-1}}$
$\frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}}$
$\frac{T}{3}\frac{1+4z^{-1}+z^{-2}}{(1-z^{-1})(1+z^{-1})}$
U^-
U^+
$\dot{x}^-$
$\dot{x}^+$
$\ddot{x}^-$
$\ddot{x}^+$
$\ddot{x}$
$\dot{x}$
x

Fig. 1. The discrete-time system which solves *Problem 1*. The system is composed by a dynamic chain based on three integrators and an algebraic variable structure controller.

time. Let us consider the following problem:

Problem 1: Design a nonlinear discrete-time filter whose output x_i tracks at best a given reference signal r_i which is known together with its first and second time derivatives, while $\dddot{r}_i = 0$. The filter must guarantee that first, second, and third time derivatives of x_i are bounded between given asymmetric limits, i.e.,

$$\dot{x}^- \le \dot{x}_i \le \dot{x}^+, \quad \ddot{x}^- \le \ddot{x}_i \le \ddot{x}^+, \quad U^- \le \dddot{x}_i \le U^+ \ . \tag{1}$$

The bounds must be freely assignable and could be time-varying: They could also change during transients. If (1) are not satisfied, for example due to the filter initial conditions or to a sudden bounds change, $\dddot{x}$ must be forced in a single step within the given limits, while $\dot{x}$ and $\ddot{x}$ must reach the assigned bounds in minimum time. If a discontinuous signal r_i is applied, or r_i admits unfeasible time derivatives, its tracking is voluntarily lost. It is achieved again, still in minimum time, if r_i newly becomes feasible. In general, every time a feasible input signal r_i is applied to the filter, tracking condition $x_i = r_i$ must be obtained in minimum time and, compatibly with (1), without overshoot.

Practically, given any reference signal r_i, filter output x_i must track it at best compatibly with the constraints. According to the definition of *Problem 1*, feasibility is prior to optimality, thus r_i is voluntarily lost any time it becomes unfeasible. The problem is clearly similar to that considered in [12], but, as a novelty, the jerk constraint is supposed to be asymmetric. This apparently small improvement, that is essential in applications like that proposed in [17], has required a complete redefinition of the filter control laws. Practically, while the structure of the discrete-time filter, as shown in Fig. 1, is the same considered in [12], i.e., it is made of a chain of three integrators driven by an Algebraic Variable Structure Controller (AVSC), the AVSC equations have been completely redefined in order to fulfill the new requirements. The system dynamics is only due to the integrators' chain and can also be represented in the following state-space form

$$\mathbf{x}_{i+1} = \mathbf{A}\,\mathbf{x}_i + \mathbf{b}\,u_i \ , \tag{2}$$

where $\mathbf{x}_i := [x_i\ \dot{x}_i\ \ddot{x}_i]^T$ is the system state and

$$\mathbf{A} = \begin{bmatrix} 1 & T & \frac{T^2}{2} \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix} \ , \quad \mathbf{b} = \begin{bmatrix} \frac{T^3}{6} \\ \frac{T^2}{2} \\ T \end{bmatrix} \ . \tag{3}$$

Reference signal r_i is evaluated as follows

$$\mathbf{r}_{i+1} := \mathbf{A}\,\mathbf{r}_i \ , \tag{4}$$

where $\mathbf{r}_i := [r_i\ \dot{r}_i\ \ddot{r}_i]^T$. A step, a ramp or a parabola are generated depending on the initial values chosen for $\dot{r}_i$ and $\ddot{r}_i$. According to the hypothesis, $\dddot{r}_i = 0$.

In order to formulate the control law for the AVSC, let us first consider the following change of coordinates $y_i := x_i - r_i$, $\dot{y}_i := \dot{x}_i - \dot{r}_i$, $\ddot{y}_i := \ddot{x}_i - \ddot{r}_i$, which places the system origin on the trajectory to be tracked. Due to (4), system (2) becomes

$$\mathbf{y}_{i+1} = \mathbf{A}\,\mathbf{y}_i + \mathbf{b}\,u_i \ , \tag{5}$$

where $\mathbf{A}$ and $\mathbf{b}$ coincide with (3), while $\mathbf{y}_i := [y_i\ \dot{y}_i\ \ddot{y}_i]^T$.

A further change of coordinates is required to eliminate sampling time T from matrices $\mathbf{A}$ and $\mathbf{b}$. Because of the asymmetry of the jerk constraint, required transformation $\mathbf{y}_i = \mathbf{W}\,\mathbf{z}_i$ differs from the one proposed in [12]. Indeed, the transformation matrix, that is defined as following

$$\mathbf{W} := \begin{bmatrix} T^3 & -T^3 & \frac{T^3}{6} \\ 0 & T^2 & -\frac{T^2}{2} \\ 0 & 0 & T \end{bmatrix} \ , \tag{6}$$

does not depend on the jerk bounds. System (5) becomes

$$\mathbf{z}_{i+1} = \mathbf{A}_d\,\mathbf{z}_i + \mathbf{b}_d\,u_i \ , \tag{7}$$

where $\mathbf{z}_i := [z_{1,i}\ z_{2,i}\ z_{3,i}]^T$ and

$$\mathbf{A}_d = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \ , \quad \mathbf{b}_d = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \ . \tag{8}$$

Matrix $\mathbf{W}$ is non singular, so that the inverse transformation $\mathbf{z}_i = \mathbf{W}^{-1}\,\mathbf{y}_i$ exists with

$$\mathbf{W}^{-1} = \begin{bmatrix} \frac{1}{T^3} & \frac{1}{T^2} & \frac{1}{3T} \\ 0 & \frac{1}{T^2} & \frac{1}{2T} \\ 0 & 0 & \frac{1}{T} \end{bmatrix} \ . \tag{9}$$

The following AVSC solves *Problem 1* (in the following, subscript i has been dropped for conciseness):

$$z_2^+ := \frac{\dot{x}^+ - \dot{r}}{T^2} - \frac{\ddot{r}}{2T} \ , \tag{10}$$

$$z_2^- := \frac{\dot{x}^- - \dot{r}}{T^2} - \frac{\ddot{r}}{2T} \ , \tag{11}$$

$$z_3^+ := \frac{\ddot{x}^+}{T} \ , \tag{12}$$

$$z_3^- := \frac{\ddot{x}^-}{T} \ , \tag{13}$$

$$\overline{z}_2^+ := -\left\lceil -\frac{z_3^+}{U^-} \right\rceil \left[z_3^+ + \frac{U^-}{2}\left(\left\lceil -\frac{z_3^+}{U^-} \right\rceil - 1 \right) \right] \ , \tag{14}$$

$$\overline{z}_2^- := -\left\lceil -\frac{z_3^-}{U^+} \right\rceil \left[z_3^- + \frac{U^+}{2}\left(\left\lceil -\frac{z_3^-}{U^+} \right\rceil - 1 \right) \right] \ , \tag{15}$$

$$d_1 := z_2 - z_2^+ \ , \tag{16}$$

$$d_2 := z_2 - z_2^- \ , \tag{17}$$

for n=1,2:

$$\gamma_n := \begin{cases} \overline{z}_2^+ & \text{if } d_n < \overline{z}_2^+ \\ d_n & \text{if } \overline{z}_2^+ \le d_n \le \overline{z}_2^- \\ \overline{z}_2^- & \text{if } d_n > \overline{z}_2^- \end{cases} \ , \tag{18}$$

$$\kappa_n := \begin{cases} U^- & \text{if } d_n \le 0 \\ U^+ & \text{if } d_n > 0 \end{cases} \ , \tag{19}$$

$$m_n := \left\lfloor \frac{1}{2} + \sqrt{\frac{1}{4} + 2\left|\frac{\gamma_n}{\kappa_n}\right|} \right\rceil , \qquad (20)$$

$$\sigma_n := -\frac{\gamma_n}{m_n} - \frac{m_n - 1}{2}\kappa_n - \frac{\dot{r}}{T} , \qquad (21)$$

end for

$$[\alpha\ \beta] := \begin{cases} [U^- \ U^+] & \text{if } \eta > 0 \\ [U^+ \ U^-] & \text{if } \eta \le 0 \end{cases} , \qquad (22)$$

$$\begin{aligned} \sigma_3 \ := \ & -\frac{2}{h(h+k)}z_1 - \frac{2h+k-1}{h(h+k)}z_2 - \frac{k(1-k^2)}{6h(h+k)}\alpha \\ & -\frac{2h^3 - 3h^2 + h + 3h^2k - 3hk}{6h(h+k)}\beta , \end{aligned} \qquad (23)$$

$$\sigma := \begin{cases} \sigma_1 & \text{if} \quad \sigma_1 < \sigma_3 \\ \sigma_3 & \text{if} \quad \sigma_2 \le \sigma_3 \le \sigma_1 \\ \sigma_2 & \text{if} \quad \sigma_3 < \sigma_2 \end{cases} , \qquad (24)$$

$$\delta := z_3 - \sigma , \qquad (25)$$

$$u := \begin{cases} -U^- \text{sat}\left(\dfrac{\delta}{U^-}\right) & \text{if} \quad \delta \ge 0 \\ -U^+ \text{sat}\left(\dfrac{\delta}{U^+}\right) & \text{if} \quad \delta < 0 \end{cases} , \qquad (26)$$

where z_1, z_2 and z_3 are the three components of $\mathbf{z}$, while integers h, k, and η are functions of z_1 and z_2. Details on the definition of h, k, and η can be found [11]. The two operators $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ respectively evaluate the floor and the ceil of a real number. Function sat$(\cdot)$ saturates its argument to ± 1.

The AVSC is designed by means of sliding mode techniques in order to robustly drive the system state toward the origin, i.e., toward $\mathbf{z} = 0$, compatibly with the constraints. This as well implies that $\mathbf{y}$ is driven toward the origin, which, in turn, means that output x tracks r. The AVSC achieves this result by switching, according to (24), between three different Sliding Surfaces (SS), namely σ_1, σ_2, and σ_3, each of them surrounded by an appropriate Boundary Layer (BL). The three SSs cover the same roles of those proposed in [12]: σ_3 drives the system toward the origin in minimum time and by simultaneously fulfilling the jerk constraint, while σ_1 and σ_2 are used to satisfy the velocity and acceleration bounds. The switching criteria is the same implemented in [12], so that the reader can refer to that work for details concerning the underlying mechanisms. Since feasibility represents the prior target, σ_1 and σ_2 are primarily selected in order to accomplish this requirement in minimum time, but, as soon as feasibility does no more represent an issue, σ_3 is used in order to reach the origin in minimum time. Similarities with the filter proposed in [12] end here, since the three surfaces, due to the constraints asymmetry, have been completely redesigned.

In the next sections the filter is analyzed in detail. In particular, in §III the characteristics of σ_3 are investigated in order to prove that such surface robustly leads the system toward the origin in minimum time by simultaneously fulfilling the constraint on the maximum jerk. Then, §IV will show that, by means of the two additional surfaces σ_1 and σ_2, it is possible to robustly guarantee the convergence of the state toward an area that is feasible with respect to the velocity and the acceleration constraints.

III. The convergence properties of σ_3

In order to prove that σ_3 drives the system in minimum time toward the origin, it is first necessary to identify the set of points in the $\mathbf{z}$-space from which the origin itself can be reached in minimum time compatibly with the constraint on the maximum jerk. Such points can be found by applying the maximum admissible values of command signal u, i.e., $u = U^+$ or $u = U^-$ and by backward integrating system (7). Let us define an additional parameter $\eta = \pm 1$: If $\eta = 1$ then command signal $u = U^-$ is initially used, while, viceversa, if $\eta = -1$ then $u = U^+$. According to this procedure, the following set of points is obtained

$$\mathbf{p}(k,\eta) = \begin{bmatrix} -\frac{1}{6}k\left(k^2 - 3k + 2\right)\alpha \\ \frac{1}{2}k(k-1)\alpha \\ -k\alpha \end{bmatrix} , \qquad (27)$$

where α depends on η and it is evaluated according to (22), while $k \in \mathbb{Z}$ indicates the number of back integrations occurred.

Remark 1: Given any point defined according to (27), it is immediately possible to know the number of steps required to converge toward the origin, e.g., from point $\mathbf{p}(k,\eta)$ the origin is reached in minimum time after k sampling times.

Points $\mathbf{p}(k,\eta)$ can be used as initial conditions for a further backward integration process. If a generic point $\mathbf{p}(k,1)$, has been reached by using $u = U^-$, the new backward process will adopts $u = U^+$. Similarly, command signal $u = U^-$ is used for points $\mathbf{p}(k,-1)$. In this way, the following new set of points $\mathbf{p}(h,k,\eta)$ is found

$$\mathbf{p}(h,k,\eta) = \begin{bmatrix} -\left[\frac{k(k-1)(k-2)}{6} + \frac{hk(h+k-2)}{2}\right]\alpha - \frac{h(h-1)(h-2)}{6}\beta \\ \left[\frac{k(k-1)}{2} + hk\right]\alpha + \frac{h(h-1)}{2}\beta \\ -k\alpha - h\beta \end{bmatrix} \qquad (28)$$

where α and β, according to (22), depend on η, while k and h indicate the number of steps occurred, respectively, during the first and the second backward integration phases.

Remark 2: Given any point defined according to (28), it is immediately possible to know the number of steps required to converge toward the origin, more precisely h steps are necessary to reach points $\mathbf{p}(k,\eta)$ and, then, further k steps are required to converge to the origin. Practically, the origin is reached in minimum time with a bang-bang control.

To simplify the notation, let us define $\mathbf{p}^-_{h,k} := \mathbf{p}(h,k,-1)$ and $\mathbf{p}^+_{h,k} := \mathbf{p}(h,k,1)$. Points $\mathbf{p}^-_{h,k}$ and $\mathbf{p}^+_{h,k}$, as shown in Fig. 2, completely cover the (z_1,z_2)-space, that is partitioned into two sectors depending on η.

This premises are instrumental to demonstrate that the control law defined through equations (22), (23), (26) with

$$\delta = z_3 - \sigma_3 , \qquad (29)$$

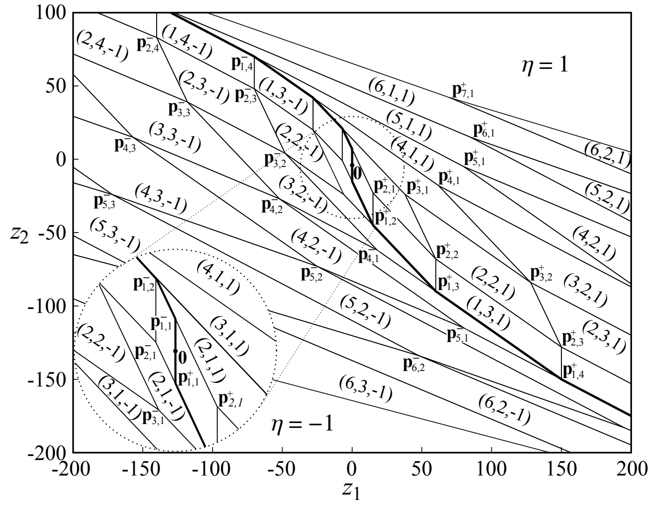


Fig. 2. Boxes (h,k,η) and points $\mathbf{p}(h,k,\eta)$ projected on the (z_1,z_2)-space.

is time-optimal and fulfills the jerk constraint. In particular, this second characteristic is immediately evident from (26): In any case $u \in [U^-, U^+]$.

As early anticipated, an AVSC, that is based on sliding mode techniques, is used to this purpose. Equation (22), depending on the current value of η, switches between two different SSs. Equation (23) expresses both of them: Because of (22), they change in function of η. Finally, (26) wraps the SS within an appropriate BL. As shown in Fig. 3, there exists a direct relationship between the SSs and points $\mathbf{p}^+_{h,k}$: Equations (22), (23), (26), and (29) associate to each point $\mathbf{p}^+_{h,k}$ a box, that is identified in the following as $(h,k,1)$, which upper/lower surfaces coincide with the upper/lower BLs of the SS.

To our purposes, it is essential to prove that the adopted control law is time-optimal. Evidently, Fig. 2 shows that σ_3 covers the whole (z_1,z_2)-space, so that, by applying $u = U^+$, the BL is certainly reached in minimum time from points placed below the SS. The same happens for points placed above the SS if command $u = U^-$ is used. In the following it will be proved that once the state enters inside any generic box (h,k,η), it slides toward the origin in minimum time. In particular, if the starting point is $\mathbf{p}(h,k,\eta)$ the convergence is achieved, as expected, in $h+k$ steps, while for any other point lying in (h,k,η) the origin is reached in $h+k+1$ steps.

Property 1: Consider system (7) and an initial state located inside box (h,k,η), with $h,k>1$. By applying control law (22), (23), (26), and (29) the system evolves, in a single step, to a new state located in box $(h-1,k,\eta)$.

Proof: Let us assume that at step i system state $\mathbf{z}_i$ lies inside box (h,k,η). By defining the following three independent vectors

$$\bar{\mathbf{e}}_r(h,k,\eta) := \begin{bmatrix} -\frac{1}{2}\left[k(k-1)+h(h-1)+2hk\right]\alpha \\ (h+k)\,\alpha \\ -\alpha \end{bmatrix}, \quad (30)$$

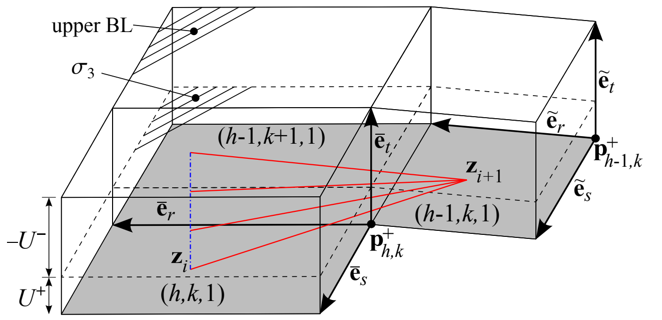


Fig. 3. Single step evolution starting from box $(h,k,1)$

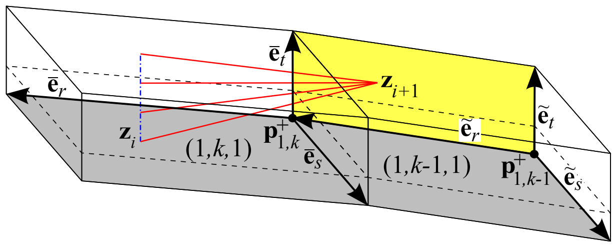


Fig. 4. Single step evolution starting from box $(1,k,1)$.

$$\bar{\mathbf{e}}_s(h,k,\eta) := \begin{bmatrix} \frac{1}{2}h(h-1) \\ -h \\ 1 \end{bmatrix}(\alpha-\beta), \quad (31)$$

$$\bar{\mathbf{e}}_t(h,k,\eta) := \begin{bmatrix} 0 \\ 0 \\ \beta-\alpha \end{bmatrix}, \quad (32)$$

which position is shown in Fig. 3 for $\eta = 1$, it is possible to describe $\mathbf{z}_i$ as follows

$$\mathbf{z}_i = \mathbf{p}(h,k,\eta) + \lambda\,\bar{\mathbf{e}}_r(h,k,\eta) + \mu\,\bar{\mathbf{e}}_s(h,k,\eta) + \nu\,\bar{\mathbf{e}}_t(h,k,\eta), \quad (33)$$

with $\lambda,\mu,\nu \in [0,1)$, $h,k>1$ and where $\mathbf{p}(h,k,\eta)$ is defined according to (28).

For $\mathbf{z} = \mathbf{z}_i$ the control law returns $u = \nu\,\alpha + (1-\nu)\,\beta$ so that the state of system (7), at step $i+1$, evolves to

$$\mathbf{z}_{i+1} = \mathbf{p}(h-1,k,\eta) + \lambda\,\tilde{\mathbf{e}}_r + \mu\,\tilde{\mathbf{e}}_s,$$

where $\tilde{\mathbf{e}}_r(h,k,\eta) = \bar{\mathbf{e}}_r(h,k,\eta)|_{h=h-1}$, and $\tilde{\mathbf{e}}_s(h,k,\eta) = \bar{\mathbf{e}}_s(h,k,\eta)|_{h=h-1}$. Point $\mathbf{z}_{i+1}$ is evidently located inside box $(h-1,k,\eta)$ and, more precisely, it lies on its lower BL if $\eta = 1$, or on its upper BL if $\eta = -1$. It is worth noticing that points $\mathbf{z}_i$, which admit the same values of λ and μ are projected in the same point $\mathbf{z}_{i+1}$, independently from ν. ■

Evidently, since *Property 1* applies for any generic point of box (h,k,η) with $h,k>1$, the system state reaches box $(1,k,\eta)$ after $h-1$ steps.

Property 2: Consider system (7) and an initial state located inside box $(1,k,\eta)$, with $k>1$. By applying control law (22), (23), (26), and (29) the system evolves, in a single step, to a new state located in box $(1,k-1,\eta)$.

Proof: Assume that at step i the system state $\mathbf{z}_i$ lies inside box $(1,k,\eta)$, so that it can be described by means of (33) by assuming $\lambda,\mu,\nu \in [0,1)$, $h=1$, $k>1$. The control

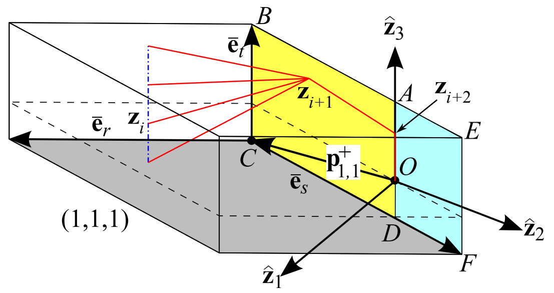


Fig. 5. Final evolution: If the system state enters into box $(1,1,1)$, it is forced toward the origin within three steps. Surface ABCD is contained into the plane individuated by unit vectors $\hat{\mathbf{z}}_2$ and $\hat{\mathbf{z}}_3$.

law still returns $u = \nu\,\alpha + (1-\nu)\,\beta$ and, consequently, the state evolves as follows

$$\mathbf{z}_{i+1} = \mathbf{p}(1,k-1,\eta) + \lambda\,\tilde{\mathbf{e}}_r + (1-\mu)\,\tilde{\mathbf{e}}_t,$$

with $\tilde{\mathbf{e}}_r(1,k,\eta) = \bar{\mathbf{e}}_r(1,k,\eta)|_{k=k-1}$, and $\tilde{\mathbf{e}}_t(1,k,\eta) = \bar{\mathbf{e}}_t(1,k,\eta)|_{k=k-1}$. State $\mathbf{z}_{i+1}$ is evidently located inside box $(1,k-1,\eta)$ and, more precisely, since component $\tilde{\mathbf{e}}_s$ is missing, it lies on a lateral face of the box. Again, as shown in Fig.4, points $\mathbf{z}_i$ that admit the same values of λ and μ are plotted in the same point $\mathbf{z}_{i+1}$ independently from ν. ■

According to *Properties 1* and *2*, starting from any generic box (h,k,η) with $h,k > 1$, the system state evolves into box $(1,1,\eta)$ after $h+k-2$ steps. The final convergence toward the origin is analyzed in the following property:

Property 3: Consider system (7) and an initial state located inside box $(1,1,\eta)$. By means of control law (22), (23), (26), and (29) the state reaches the origin with a maximum of three transitions.

Proof: Assume that at step i the system state $\mathbf{z}_i$ is located inside box $(1,1,\eta)$, so that it can be described by means of (33) by assuming $\lambda,\mu,\nu \in [0,1)$, $h,k = 1$. The command law returns $u = \nu\,\alpha + (1-\nu)\,\beta$ and the state evolves, in a single step, as follows

$$\mathbf{z}_{i+1} = \mathbf{p}(1,1,\eta) + \delta\,\bar{\mathbf{e}}_s(1,1,\eta) + \varepsilon\,\bar{\mathbf{e}}_t(1,1,\eta),$$

where $\delta \in \left[0, -\frac{\alpha}{\beta-\alpha}\right]$ and $\varepsilon \in [0,1]$. It is easy to verify that, when $\eta = 1$, point $\mathbf{z}_{i+1}$ is located on surface $ABCD$ of Fig. 5, while when $\eta = -1$ it lies on surface $AEFD$.

If a further step is executed from $\mathbf{z}_{i+1}$ the command signal becomes $u = \varepsilon\,\alpha + (1-\varepsilon)\,\beta$ and the state is forced to point $\mathbf{z}_{i+2} := [0\ \ 0\ \ (\delta-1)\,\alpha - \delta\,\beta]^T$. Bearing in mind the definition of δ, it is evident that, if $\mathbf{z}_{i+1}$ lies on $ABCD$ ($\eta = 1$), then $\mathbf{z}_{i+2}$ is located on segment OA, while, if $\mathbf{z}_{i+1}$ lies on $AEFD$ ($\eta = -1$), then $\mathbf{z}_{i+2}$ is located on OD.

Finally, for any point $\mathbf{z}_{i+2}$ located on segment AD, still with command signal $u = \varepsilon\,\alpha + (1-\varepsilon)\,\beta$, the state is forced to the origin in a single step. ■

In conclusion, if the initial state is located inside box (h,k,η), it evolves toward the origin in no more than $h+k+1$ steps. It is possible to verify that if the initial state is equal to $\mathbf{p}(h,k,\eta)$ then $h+h-2$ step are required to reach point $\mathbf{p}(1,1,\eta)$, while only two more steps are required to reach the origin: Convergence is achieved in $h+k$ steps, thus confirming the solution optimality.

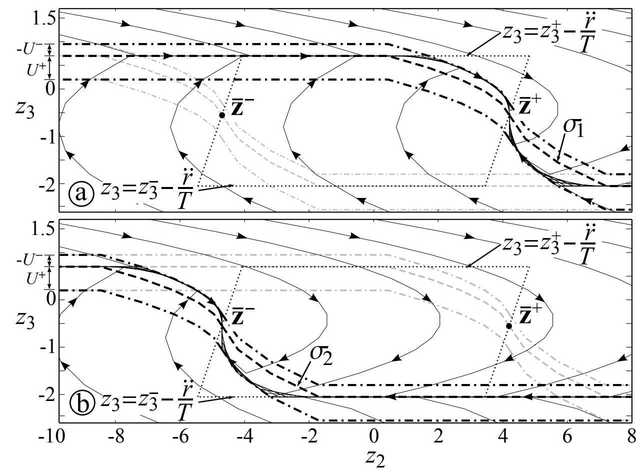


Fig. 6. System trajectories in the (z_2,z_3)-space obtained by assuming: a) $\sigma = \sigma_1$ and b) $\sigma = \sigma_2$. SSs σ_1 and σ_2 are indicated by means of dashed lines and are surrounded by their BLs (dash dotted lines). The dotted quadrangle contours the feasible area.

IV. The Convergence Properties of σ_1 and σ_2

Surfaces σ_1 and σ_2 are designed to force the system, in minimum time, within a zone where the velocity and the acceleration constraints are fulfilled. One of the two surfaces is chosen by means of (24) every time σ_3 should drive the state outside the feasible area. Such area, defined in the $(\dot{x},\ddot{x})$-space by means of a rectangle delimited by lines $\dot{x} = \dot{x}^+$, $\dot{x} = \dot{x}^-$, $\ddot{x} = \ddot{x}^+$, and $\ddot{x} = \ddot{x}^-$, can be converted into an equivalent area in the (z_2,z_3)-space. More precisely, the velocity bounds, i.e., $\dot{x} = \dot{x}^+$ and $\dot{x} = \dot{x}^-$, are respectively converted into the following limits $z_3 = 2\left(z_2 - z_2^+ + \frac{\dddot{r}}{2T}\right)$, $z_3 = 2\left(z_2 - z_2^- + \frac{\dddot{r}}{2T}\right)$, while the acceleration bounds, i.e., $\ddot{x} = \ddot{x}^+$ and $\ddot{x} = \ddot{x}^-$, become $z_3 = z_3^+ - \frac{\dddot{r}}{T}$, and $z_3 = z_3^- - \frac{\dddot{r}}{T}$. Dotted lines of Fig. 6 highlight the converted feasible area. Evidently, it is independent from z_1, and, for this reason, the following discussion will only focus on the state evolution in the (z_2,z_3)-subspace. Fig. 6 also shows σ_1 and σ_2 with the corresponding BLs and the system trajectories: The maximum command signals, i.e., $u = U^+$ or $u = U^-$, are used when the state is outside the BLs, so that the area within the two accelerations limits, i.e., within $z_3 = z_3^+ - \frac{\dddot{r}}{T}$ and $z_3 = z_3^- - \frac{\dddot{r}}{T}$, is certainly reached in minimum time. Then, the state is driven, depending on which of the two SS is used, toward $\bar{\mathbf{z}}^+$ or toward $\bar{\mathbf{z}}^-$. In any case, the desired result is achieved, since both points are evidently feasible with respect to the velocity and the acceleration constraints. States $\bar{\mathbf{z}}^+$ and $\bar{\mathbf{z}}^-$ are obtained by transforming points $(\dot{x}^+,0)$ and $(\dot{x}^-,0)$ of the $(\dot{x},\ddot{x})$-space. Practically, when σ_1 (or σ_2) is chosen, the system is forced in minimum time in $(\dot{x}^+,0)$ [or in $(\dot{x}^-,0)$]: With arguments analogous to those reported in [12], it is possible to prove that the system state, when it is locked in one of those two states, moves with zero acceleration and at the maximum speed toward σ_3. When such surface is reached, $\bar{\mathbf{z}}^+$ (or $\bar{\mathbf{z}}^-$) is abandoned and the system can finally converge to the origin with a feasible trajectory.

Surfaces σ_1 and σ_2 have been obtained by modifying the analogous surface proposed in [16]. Indeed, bearing in mind

(7) and (8), the system evolution in the $(\dot{x},\ddot{x})$-space can be expressed by the following state equation

$$\begin{bmatrix} z_{2,i+1} \\ z_{3,i+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{2,i} \\ z_{3,i} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_i, \qquad (34)$$

i.e., the model is the same of the system already considered in [16], but the role of the pair z_1 and z_2 is now played by z_2 and z_3. Thus, by using the same control law proposed in that paper, the same convergence properties are evidently maintained: The sole difference that has been introduced is that σ_1 and σ_2 are modified with respect to the original SS, so that the state does not converge to the origin but, conversely it converges to $\bar{\mathbf{z}}^+$ or to $\bar{\mathbf{z}}^-$.

V. A TEST CASE

In the test case of Fig. 7 the filter handles a discontinuous signal made of steps, ramps and parabolas. The following kinematic bounds have been initially assumed: $\dot{x}^+ = 2.5$ m s^{-1}, $\dot{x}^- = -3$ m s^{-1}, $\ddot{x}^+ = 3.5$ m s^{-2}, $\ddot{x}^- = -4.9$ m s^{-2}, $U^+ = 10$ m s^{-3}, $U^- = -15$ m s^{-3}. Fig. 7a compares the original discontinuous signal with the filter outputs: x tracks at best reference r, compatibly with the given constraints. The small undershoot, that is highlighted by the dotted circle, appears if the acceleration constraint is touched during the final transient toward r. Indeed, any time such bound is activated, the control switches to σ_1 or to σ_2, thus the state abandon σ_3 and an overshoot is produced. This problem can be eliminated by managing the acceleration constraint directly with σ_3. Studies on this topic are undergoing.

The filter bounds are changed at $t = 6.4$ s: $\dot{x}^+ = 1.5$ m s^{-1}, $\dot{x}^- = -2$ m s^{-1}, $\ddot{x}^+ = 3$ m s^{-2}, $\ddot{x}^- = -3.9$ m s^{-2}, $U^+ = 9$ m s^{-3}, $U^- = -9$ m s^{-3}. The system immediately generates trajectories that fulfill the new limits.

A new bounds change is planned in a critical situation, i.e., when the filter is in the middle of a transient toward a ramp. In particular, at $t = 12.5$ s, constraints assume the following values: $\dot{x}^+ = 1.5$ m s^{-1}, $\dot{x}^- = -1$ m s^{-1}, $\ddot{x}^+ = 5.5$ m s^{-2}, $\ddot{x}^- = -1.9$ m s^{-2}, $U^+ = 7$ m s^{-3}, $U^- = -9$ m s^{-3}. Owing to these sudden changes, constraints are instantly violated (see the dash-dotted circles in Fig. 7). As required, jerk bounds are fulfilled in a single step, while acceleration and velocity limits are satisfied in minimum time, compatibly with the jerk constraints. Thus, according to the requirements, the constraints fulfillment is considered prior with respect to the convergence toward the origin: The ramp is hanged, still in minimum time, only after the signal feasibility is guaranteed.

VI. CONCLUSIONS

The discrete-time filter proposed in the paper is able to generate, starting from rough references which continuity is not guaranteed, signals that are continuous together with their first and second time derivatives. Moreover, it is able to implicitly impose bounds on the first, the second, and the third time derivatives of the output signal. Generated transients are minimum-time. Differently from analogous filters proposed in the past, it robustly handles asymmetric constraints.

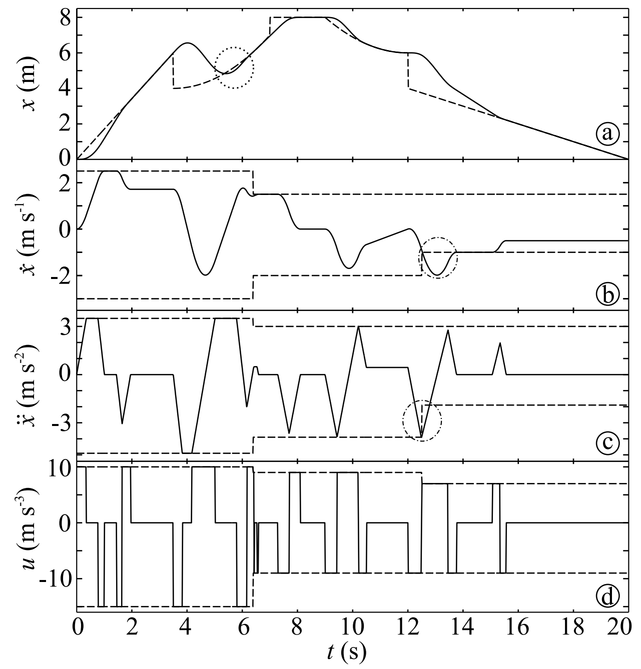


Fig. 7. The simulated test case: a) the non-smooth reference signal (dashed line) is compared with the filter output x (solid line); b) filter velocity signal $\dot{x}$ (solid line) compared with the given bounds (dashed lines) c) filter acceleration $\ddot{x}$ (solid line) compared with the given bounds (dashed lines) d) filter jerk signal $\dddot{x}$ (solid line) compared with the given bounds (dashed lines).

The proposed planner is characterized by several advantages. First of all, it has a structure that is extremely simple: The code length required for the implementation of the sliding mode controller is negligible, so that the filter can even be implemented in systems with reduced memory capabilities. Also the computational burden is particularly light: The average evaluation time detected with a PC based on an Intel Core2 Duo processor, @3GHz, and equipped with a RTAI patched operating system, is equal to 4.32e-6 s.

REFERENCES

[1] S. Liu, "An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators," in *Proc. of the seventh Int. Work. on Advanced Motion Control*, 2002, pp. 365–370.

[2] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of timeoptimal, jerk-limited trajectories," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 08*, 2008, pp. 3248–3253.

[3] X. Broquère, D. Sidobre, and I. Herrera-Aguilar, "Soft motion trajectory planner for service manipulator robot," in *Proc. of the 2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 08*, 2008, pp. 2808–2813.

[4] T. Kröger and F. M. Wahl, "On-Line Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events," *IEEE Trans. on Robotics*, vol. 26, no. 1, pp. 94–111, Feb. 2010.

[5] C. Guarino Lo Bianco, A. Tonielli, and R. Zanasi, "Nonlinear trajectory generator," in *IECON96 - 22nd Annual International Conference on the IEEE Industrial Electronics Society*, Taiwan, Taipei, August 1996, pp. 195–201.

[6] R. Zanasi, A. Tonielli, and C. Guarino Lo Bianco, "Nonlinear filter for smooth trajectory generation," in *NOLCOS98 - 4th Nonlinear Control Systems Design Symp.*, vol. 1, Enschede, the Netherlands, July 1998, pp. 245–250.

[7] R. Zanasi, C. Guarino Lo Bianco, and A. Tonielli, "Nonlinear filters for the generation of smooth trajectories," *Automatica*, vol. 36, no. 3, pp. 439–448, March 2000.

[8] C. Guarino Lo Bianco and R. Zanasi, "Smooth profile generation for a tile printing machine," *IEEE Trans. on Ind. Electronics*, vol. 50, no. 3, pp. 471–477, 2003.

[9] L. Biagiotti and R. Zanasi, "Online trajectory planner with constraints on velocity, acceleration and torque," in *Proc. of the IEEE Int. Symp. on Ind. Electr., ISIE2010*, Bari, Italy, Jul. 2010, pp. 274–279.

[10] R. Zanasi and R. Morselli, "Third order trajectory generator satisfying velocity, acceleration and jerk constraints," in *Proc. of the 2002 IEEE Int. Conf. on Control Applications*, Glasgow, Scotland, UK, Sept. 2002, pp. 1165–1170.

[11] ——, "Discrete minimum time tracking problem for a chain of three integrators with bounded input," *Automatica*, vol. 39, no. 9, pp. 1643–1649, 2003.

[12] O. Gerelli and C. Guarino Lo Bianco, "A discrete-time filter for the on-line generation of trajectories with bounded velocity, acceleration, and jerk," in *IEEE Int. Conf. on Rob. and Autom., ICRA2010*, Anchorage, AK, May 2010, pp. 3989–3994.

[13] D. Kubus, D. Inkermann, T. Vietor, and F. M. Wahl, "Joint Actuation Based on Highly Dynamic Torque Transmission Elements – Concept and Control Approaches," in *Proc. of IEEE Int. Conf. on Rob. and Autom., ICRA 2011*, Shanghai, China, May 2011, pp. 2777–2784.

[14] C. Guarino Lo Bianco and O. Gerelli, "Trajectory scaling for a manipulator inverse dynamics control subject to generalized force derivative constraints," in *2009 IEEE/RSJ Int. Conf. on Intell. Robots and Systems, IROS 2009*, St. Louis, MO, Oct. 2009, pp. 5749–5754.

[15] O. Gerelli and C. Guarino Lo Bianco, "Real-time path-tracking control of robotic manipulators with bounded torques and torque-derivatives," in *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 2008*, Nice, France, Sep. 2008, pp. 532–537.

[16] C. Guarino Lo Bianco and F. Wahl, "A novel second order filter for the real-time trajectory scaling," in *Proc. of the 2011 IEEE Int. Conf. on Robotics and Automation, ICRA 2011*, Shanghai, China, May 9–13 2011, pp. 5813–5818.

[17] C. Guarino Lo Bianco and O. Gerelli, "Online Trajectory Scaling for Manipulators Subject to High-Order Kinematic and Dynamic Constraints," *IEEE Trans. on Robotics*, vol. 27, no. 6, pp. 1144–1152, Dec. 2011.

Workshop on Robot Motion Planning:
Online, Reactive, and in Real-time
2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems, IROS 2012
Vilamoura, Algarve, Portugal, October 7-12, 2012

Online Trajectory Generation Algorithms as an Intermediate Layer between Robot Motion Planning and Control

Torsten Kroeger
Artificial Intelligence Laboratory
Department of Computer Science
Stanford University, USA

Abstract

Recently, a new class of online trajectory generation algorithms has been developed that allows robots to react instantaneously to unforeseen sensor signals and events. Trajectories are computed at the servo level, which allows the realization of highly reactive control systems, and which opens the door to new robot motion control features. Instantaneous frame changes, using hybrid switched-systems with feedback of multiple sensors, instantaneously switching from sensor-guided to trajectory following motions (and vice versa), and filtering sensor signal under consideration of dynamic constraints has been implemented in real-world applications.
This talk focusses on sensor-based online trajectory generation in robotic control systems, and in particular on an intermediate layer between low-level motion control and high-level sensor-based motion planning that contains these new algorithms. A major symbiotic effect of this use-case is that robotic systems that are guided by higher-level planning systems obtain the ability of performing immediate reflex motions as a reaction to unforeseen low-level events.
The online trajectory generation algorithms have been released in the Reflexxes Motion Libraries. Samples and use-cases showing the chain from high-level motion planning to execution will accompany the talk in order to provide a comprehensible insight into this interesting and relevant field of robotics.

References for Further Reading

T. Kroeger. *On-Line Trajectory Generation: Straight-Line Trajectories*. IEEE Trans. on Robotics, Vol. 27, No. 5, pp. 1010–1016, October 2011.

T. Kroeger and F. M. Wahl. *On-Line Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events*. IEEE Trans. on Robotics, Vol. 26, No. 1, pp. 94–111, February 2010.

T. Kroeger. *On-Line Trajectory Generation in Robotic Systems: Basic Concepts for Instantaneous Reactions to Unforeseen (Sensor) Events*. Springer Tracts in Advanced Robotics, Vol. 58, **Springer**, January 2010.

T. Kroeger. *Opening the Door to New Sensor-Based Robot Applications — The Reflexxes Motion Libraries*. In Proc. of the IEEE International Conference on Robotics and Automation, Shanghai, China, May 2011.

T. Kroeger. *On-Line Trajectory Generation: Nonconstant Motion Constraints*. In Proc. of the IEEE International Conference on Robotics and Automation, pp. 2048–2054, Saint Paul, MN, USA, May 2012.