

Real-time motion planning in the presence of moving obstacles

Tim Mercy, Wannes Van Loock and Goele Pipeleers

Abstract—Safe operation of autonomous systems demands a collision-free motion trajectory at every time instant. This paper presents a method to calculate time-optimal motion trajectories for autonomous systems moving through an environment with both stationary and moving obstacles. To transform this motion planning problem into a small dimensional optimization problem, suitable for real-time optimization, the approach (i) uses a spline parameterization of the motion trajectory; and (ii) exploits spline properties to reduce the number of constraints. Solving this optimization problem with a receding horizon allows dealing with modeling errors and variations in the environment. In addition to extensive numerical simulations, the method is experimentally validated on a *KUKA youBot*. The average solving time of the optimization problem in the experiments is 0.05s, which is sufficiently fast for correcting deviations from the initial trajectory.

I. INTRODUCTION

Automation has a very long history, reaching from the water clock of Ktesibios [1] to the development of the Google Self-Driving Car [2]. The reasons to automate processes and machines vary widely. Automation can for instance be used to relieve humans from laborious and repetitive tasks, such as weaving clothes or harvesting fields with a tractor. In applications like robotic surgery or milling, automation is used to increase the accuracy, while in applications like inserting bolts or milking cows, its main purpose is to speed up tasks. Finally, automation can be necessary for safety reasons, e.g. to clean up nuclear waste or to explore burning areas. Due to these benefits autonomous motion systems have become very popular in industry.

A common problem in the control of autonomous systems is finding the fastest, most energy efficient, safest... way to move the system from its initial position to a certain destination, while respecting the system limitations and avoiding collisions with all obstacles. The domain of motion planning concerns the computation of such trajectories, which is typically done using numerical optimization.

In general, solving motion planning problems faces three challenges. The first challenge is that they translate into hard, non-convex optimization problems by the combination of geometric constraints (e.g. obstacle avoidance), kinematic constraints (velocity, acceleration... bounds) and dynamic

constraints (actuator limits). As handling all these constraints at once, sometimes called the coupled approach, often leads to too complicated an optimization problem, researchers have resorted to decoupled motion planning approaches [3]. Herein a so-called path planning problem is solved first to obtain a geometric path that satisfies the geometric constraints. Afterwards, a path following problem determines the optimal motion along the geometric path, taking into account all remaining constraints. In general, the subproblems of the decoupled approach are easier to solve, but the overall result is suboptimal since not all constraints are taken into account at once.

The second challenge of motion planning is that the resulting optimization problems involve constraints that must hold during the complete motion time. Such constraints are generally handled through time gridding [4], which comes down to imposing the constraints only at certain time instants, see e.g. [5]–[7]. This approach has two important drawbacks: (i) constraints may be violated between the grid points, which is intolerable in safety-critical applications; (ii) the grid spacing should be sufficiently fine to avoid such violations as much as possible. This leads to a high number of constraints.

A third challenge is that the environment in which the autonomous motion system operates is generally uncertain, since obstacle positions and movements are not fully known a priori. Therefore it is necessary to update the motion trajectory in real time, based on the most recent world information.

To tackle the three aforementioned challenges, this paper builds upon [8], which presents a method for computing optimal motion trajectories for polynomial differentially flat systems. This method already deals with the first two challenges by proposing a B-spline parameterization for the motion trajectories. Time gridding is avoided by exploiting the properties of B-splines to guarantee constraint satisfaction at all times. However, only off line motion planning is handled and no obstacles are considered in the environment. The current paper extends the work of [8] with an efficient approach to include obstacle avoidance constraints in the motion planning problem by using time-varying separating hyperplanes. Both stationary and moving obstacles can be considered. Real-time motion planning is used to intercept uncertainty and variability in the environment. In addition, it corrects for deviations from the planned trajectory due to disturbances and modeling errors. The potential of the resulting approach is experimentally demonstrated on a *KUKA youBot*.

First, Section II presents the considered motion planning problem and divides it into its three main elements: a motion system, obstacles and a border. This section also shows

Acknowledgement This work has been carried out within the framework of projects Flanders Make ICON SitControl: Control with situation information and project G0C4515N of the Research Foundation - Flanders (FWO - Flanders). This work also benefits from KU Leuven-BOF PFV/10/002 Centre of Excellence: Optimization in Engineering (OPTEC) and the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office (DYSCO).

The authors are with the Faculty of Mechanical Engineering, Division PMA, KU Leuven, BE-3001 Leuven, Belgium tim.mercy@kuleuven.be

the resulting optimization problem. Section III describes the approach used in [8] and expands it to allow for moving obstacles and a changing environment. Furthermore, this section explains how to include a safety factor in the anti-collision constraints. Finally, Section IV shows numerical simulations and experimental results.

II. MOTION PLANNING PROBLEM

The developed approach can handle motion planning problems that involve the following three elements: (i) a system for which the motion trajectory is planned¹, (ii) obstacles which are stationary or move around in the environment, (iii) a border which limits the environment. Each of these elements contribute to the optimization problem for computing the time-optimal motion trajectory. The following subsections each analyze a specific element of this motion planning problem.

A. Vehicle

The vehicle is the basis of the motion planning problem. Each vehicle is described by certain properties: a kinematic model, a shape and kinematic limits².

The kinematic model describes the possible vehicle motion. Two important classes are holonomic and nonholonomic vehicle models. A holonomic vehicle, which is a commonly used autonomous motion system, can easily move in any direction. Nonholonomic vehicles are more constrained in their motion. Although certain nonholonomic vehicles can be considered in the approach (see e.g. [9] on how to include a quadrotor) the remainder of this paper, which presents the first developments building upon [8], focuses on holonomic vehicles. A second simplification is that the vehicle has a fixed orientation, therefore the following variables describe the holonomic vehicle:

$$q(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}, \quad (1)$$

in which $x(t)$ and $y(t)$ are the position as a function of time. It is also possible to include e.g. a holonomic vehicle with a certain orientation, this requires an extra variable which expresses the vehicle orientation as a function of time.

The second vehicle property are its kinematic limits, like velocity and acceleration bounds, which are inserted into the optimization problem as constraints. For instance:

$$\dot{q}_{\min} \leq \dot{q}(t) \leq \dot{q}_{\max}, \quad \forall t \in [0, T], \quad (2)$$

in which T is the total motion time.

Finally, the vehicle also has a shape (e.g. a circle, a rectangle...), which determines the anti-collision constraints. Below, the vehicle is represented as a circle with a certain radius r_{veh} , but it could also be a rectangle with given length and width.

¹Hereafter this element is called ‘vehicle’, but it can also be another motion system, e.g. a robot or a quadrotor.

²The proposed method does not yet include vehicle dynamics.

B. Obstacles

Every autonomous motion system moving in a realistic environment will encounter obstacles on its way. Collisions can cause damage to equipment or people and must be avoided at all costs. Therefore, obstacles play an important role in the motion planning problem. In general, obstacles can be divided in different categories, based on their properties. Two important obstacle properties are its shape and its motion model.

The shape of an obstacle determines its representation. A circular obstacle is represented by a center and a radius (r). A rectangular obstacle has a center, width, height and orientation. The shape of the obstacle determines the anti-collision constraints.

The second property is the motion model of the obstacle. This motion model gives a description of the predicted position of the obstacle center as a function of time ($q_{\text{obs}}^{\text{pred}}(t)$). A general environment will include both stationary and moving obstacles. If the obstacle is stationary the motion model is a constant, equal to the current position ($q_{\text{obs}}^{\text{init}}$). If the obstacle is moving, the proposed method uses the following linear motion model:

$$q_{\text{obs}}^{\text{pred}}(t) = q_{\text{obs}}^{\text{init}} + t \cdot \dot{q}_{\text{obs}}(t), \quad (3)$$

where $\dot{q}_{\text{obs}}(t)$ is the obstacle velocity at time t . The choice for this simple model is motivated in Section III-D.

C. Border

The border will in many cases be rectangular e.g. a room or a field. For a holonomic vehicle, represented by a circle, the anti-collision constraint expresses that the vehicle has to stay within the borders, while accounting for its radius:

$$A \cdot q(t) \leq b - r_{\text{veh}}, \quad \forall t \in [0, T]. \quad (4)$$

In this equation A and b give a matrix representation of the rectangular border.

D. Optimization problem

The vehicle, obstacle and border elements of the motion planning problem determine the constraints and variables of the optimization problem. Since the goal is to compute time-optimal trajectories, the last element of the optimization problem, the objective function, will be given by the motion time T . Since the motion time is unknown beforehand, it is also a variable. For the most basic example, in which a circular vehicle has to move from an initial state q_{start} to a final state q_{end} while avoiding collision with one circular obstacle, the optimization problem is shown in equation (5). Typically, the vehicle starts from standstill and must arrive at rest, this

is expressed by the equality constraints on $\dot{q}(t)$ and $\ddot{q}(t)$.

$$\begin{aligned}
& \underset{q(\cdot), T}{\text{minimize}} && T \\
& \text{subject to} && q(0) = q_{\text{start}}, \quad q(T) = q_{\text{end}} \\
& && \dot{q}(0) = 0, \quad \dot{q}(T) = 0 \\
& && \ddot{q}(0) = 0, \quad \ddot{q}(T) = 0 \\
& && \dot{q}_{\min} \leq \dot{q}(t) \leq \dot{q}_{\max} \\
& && \ddot{q}_{\min} \leq \ddot{q}(t) \leq \ddot{q}_{\max} \\
& && \text{dist}(\text{veh}(t), \text{obs}(t)) \geq \epsilon \\
& && \forall t \in [0, T].
\end{aligned} \tag{5}$$

III. METHODOLOGY

Section II formulated the general motion planning problem for calculating time-optimal motion trajectories for autonomous systems in (5), which contains all the challenges mentioned in Section I. This section applies the method proposed in [8] to this optimization problem and elaborates on how to include moving obstacles and deal with variations in the environment.

A. Spline parameterization and B-spline relaxations

The author of [8] proposed a method to efficiently formulate and solve motion planning problems. There are two key aspects of this method: (i) a B-spline parameterization is adopted for the motion trajectory $q(\cdot)$; and (ii) the properties of B-splines are exploited to replace constraints over the entire time horizon by small finite, yet conservative, sets of constraints. This second aspect is hereafter called B-spline relaxations.

Splines are piecewise polynomial functions that can easily represent complex trajectories, using few variables. A spline is given by a linear combination of B-spline basis functions [10]:

$$s(t) = \sum_{i=1}^n c_i \cdot B_i(t). \tag{6}$$

In equation (6) $B_i(t)$ are the B-spline basis functions, c_i are the spline coefficients and n follows from the spline degree and the number of knots. Figure 1 illustrates this graphically. This figure shows the knot intervals as rectangular areas. Each knot interval of the depicted spline contains a spline coefficient, indicated by a dot. The ordinate of a dot equals the corresponding coefficient c_i . The connection of the dots forms the control polygon of the spline. Note that the spline always lies within the convex hull of the control polygon, which is called the convex hull property of splines [10]. As a consequence, semi-infinite constraints of the form

$$s(t) \leq 0.75, \quad \forall t \in [0, T] \tag{7}$$

are guaranteed to hold if

$$c_i \leq 0.75, \quad i = 1 \dots n. \tag{8}$$

Replacing infinite sets of constraints of the form (7), by the finite, yet conservative sets (8) is called a B-spline relaxation.

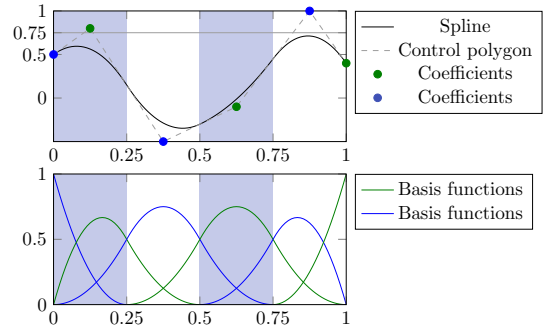


Fig. 1: Graphical illustration of a spline as a linear combination of B-spline basis functions

The major advantage is that B-spline relaxations avoid time gridding of the constraints, while they guarantee constraint satisfaction at all times. Unfortunately B-spline relaxations also introduce some conservatism. This conservatism stems from the distance between the control polygon and the spline itself (see Figure 1), since the relaxations come down to transforming constraints on the spline to constraints on the control polygon. Figure 1 shows that the relaxed constraints (8) will be violated, while in reality the spline still satisfies (7). To reduce conservatism, the spline can be represented in a higher dimensional basis that includes the original one [8]. There are two techniques to obtain a higher dimensional basis. The first technique inserts extra knots and therefore acts only locally. The second technique raises the order of the basis functions and therefore has a global effect. Although both techniques reduce the conservatism of the B-spline relaxation, they also translate into more constraints. Hence, it is necessary to make a trade-off between conservatism and computational complexity (number of constraints).

When applying the approach described above to (5), $q(t)$ will be parameterized as a spline, i.e.: $q(t) = \sum_{i=1}^n c_i^q \cdot B_i(t)$, and the variables of the optimization problem become c_i^q and T . Furthermore, all constraints on splines over the entire time horizon will be relaxed by applying B-spline relaxations. For instance, for the velocity bounds, this gives:

$$\dot{q}_{\min} \leq \dot{c}_i^q \leq \dot{q}_{\max}, \quad i = 1 \dots n-1, \tag{9}$$

where \dot{c}_i^q are the spline coefficients of $\dot{q}(t)$, they depend linearly on c_i^q . Section IV will show that the proposed B-spline formulation leads to a small scale optimization problem that is suitable for real-time optimization and guarantees constraint satisfaction at all times.

B. Collision avoidance

A first expansion to the work of [8] is the addition of obstacles. Collision avoidance translates into expressing that the shape representing the vehicle may not overlap with any obstacle. Since the vehicle is supposed to be circular, the following equation expresses collision avoidance with a circular obstacle:

$$\|q(t) - q_{\text{obs}}(t)\|_2^2 \geq \|r_{\text{obs}} - r_{\text{veh}}\|_2^2, \quad \forall t \in [0, T]. \tag{10}$$

In this equation $q_{\text{obs}}(t)$ is the obstacle center as a function of time and r_{obs} is the obstacle radius.

When the obstacle is a rectangle, there is no simple algebraic equation to express collision avoidance. In that case the proposed method uses the separating hyperplane theorem [11] to express the anti-collision constraints. This theorem states that two non-intersecting convex sets can always be separated by a hyperplane. In the 2D case this simplifies to stating that it is always possible to draw a line between two non-intersecting convex sets, which is actually a classification problem. Translated into constraints, for a rectangular obstacle, this gives:

$$\begin{aligned} a(t)^T v_i(t) - b(t) &\geq 0, \quad i = 1 \dots 4 \\ a(t)^T q(t) - b(t) &\leq -r_{\text{veh}} \\ \|a(t)\|_2 &\leq 1 \\ \forall t &\in [0, T]. \end{aligned} \quad (11)$$

In these equations $q(t)$ represents the motion trajectory. $v_i(t)$ gives the position of the vertices of the rectangular obstacle, these follow from $q_{\text{obs}}(t)$ and the obstacle shape and orientation. $a(t)$ is the normal vector to the hyperplane and $b(t)$ is hyperplane's offset. Both $a(t)$ and $b(t)$ are also parameterized as splines and describe the motion of the hyperplane as a function of time. Compared to the classical time gridding approach, which would require searching a separating hyperplane at each time sample, this implementation is more elegant and efficient. Furthermore, applying B-spline relaxations guarantees constraint satisfaction at all time instants. Figure 2 illustrates this approach graphically and shows the evolution of the separating hyperplane along the motion trajectory.

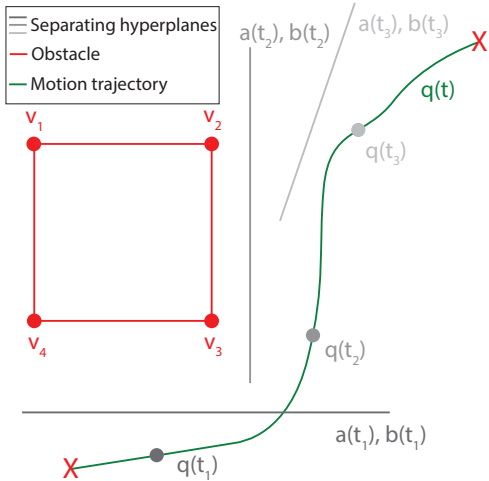


Fig. 2: Separating hyperplane theorem

Note that anti-collision constraint (10) can also be imposed using the separating hyperplane theorem. In that case v_i is the center of the circular obstacle and the 0 in the right hand side of the first inequality of (11) is replaced by r_{obs} .

C. Soft constraints

If obstacles are blocking the way to the goal position, the method will often return trajectories with near collisions, since these give a lower motion time (see solid line in Figure 3). In practice this behaviour is highly undesired, since the slightest deviation from the predicted movement of the vehicle or the obstacle may then cause a collision. Therefore the proposed method offers the possibility to include a safety factor (δ). This safety factor is set by the user, depending on the application, and expresses the desired distance between the vehicle and any obstacle. The addition of the safety factor introduces an extra variable $d(t)$ in the first inequality of (11):

$$a(t)^T v_i(t) - b(t) \geq d(t), \quad i = 1 \dots 4, \quad (12)$$

in which $d(t)$ expresses the distance between the vehicle and the corresponding obstacle as a function of time. The soft constraint adds an extra term $\|d(t) - \delta\|_2$ to the objective, which penalizes $d(t)$ if it deviates from δ . Therefore, $d(t)$ will only become smaller than δ when there is no other option (see e.g. the dashed line in Figure 3).

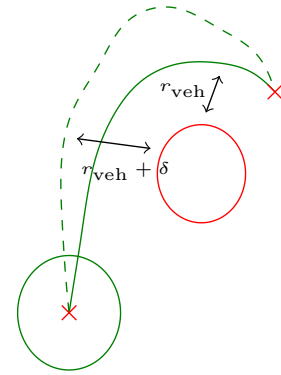


Fig. 3: Motion planning with (dashed) and without (solid) soft constraints

D. Receding horizon implementation

In practical applications there is always uncertainty e.g. in the obstacle velocity measurements or the vehicle position measurements, but also in the amount of obstacles in the environment or in the goal position. Solving the motion planning problem in real time with a receding horizon can intercept these problems, provided that the corresponding update rate is sufficiently high. A receding horizon ensures that the most recent world information (vehicle position, obstacle velocity estimation, number of obstacles...) is used while solving the optimization problem. Furthermore, the fast update rate allows applying the trajectory to the system via velocity feedforward setpoints, given by $\dot{q}(t)$. This only requires a velocity controller to obtain the desired setpoints. Position control is not necessary since there is implicit position feedback due to measurements of the vehicle position and trajectory updates which account for the latest measurements. Finally, using a receding horizon also motivates the use of a simple motion model (see Section II-B). If the

velocity of an obstacle changes, the method will account for this in the next iteration.

The receding horizon implementation looks as follows:

- 1) Solve optimization problem with current world information
- 2) Obtain complete motion trajectory
- 3) Apply the trajectory $\dot{q}(t)$, with $t \in [t, t + T_s]$, in which t is the current time and T_s is the sample time of the method
- 4) Update world information using recent measurements
- 5) Repeat

Note that this approach corresponds to model predictive control (MPC) [12] in which the vehicle's low level velocity control is assumed to be perfect.

IV. RESULTS

This section shows simulation results to illustrate the methodology. Furthermore, it makes a comparison between the proposed approach and the classical time gridding approach. Finally the section also gives an overview of the experimental results. In all results the motion trajectory consists of a degree 3 spline with 11 knots.

A. Illustrative examples

Before going to practical applications the complete method has been tested extensively by numerical simulations. The method was implemented in Python, as part of a spline-based motion planning toolbox³ which uses the CasADi framework [13] and Ipopt as a solver [14].

The first numerical example illustrates the importance of accounting for the motion of each obstacle by assigning a motion model to it. Figure 4 shows that not taking into account the movement of an obstacle can block the system and cause collisions. Suppose that there is an obstacle moving to the top left. The line starting from the obstacle center graphically represents the movement prediction. The end of the line gives the predicted obstacle center position when the vehicle arrives at its goal, the dashed line leads to the obstacle's starting position. On the left side of Figure 4 there is no motion model included, so the moving obstacle is supposed to be stationary. In this case the vehicle tries to pass in front of the obstacle, which can lead to collisions, e.g. if the obstacle moves faster than the vehicle. On the right side of Figure 4, a simple linear motion model is included. Now the motion trajectory runs through the obstacle because the method knows that the obstacle will be gone when the vehicle arrives there.

Figure 4 also gives an illustration of the receding horizon implementation, each cross in the figure denotes a new iteration. In each iteration the complete remaining motion trajectory is calculated, while only the velocity setpoints corresponding to the first part of this trajectory are actually applied to the system. This approach allows taking into account changes in the environment. Figure 5 shows the movement of a circular holonomic vehicle through two

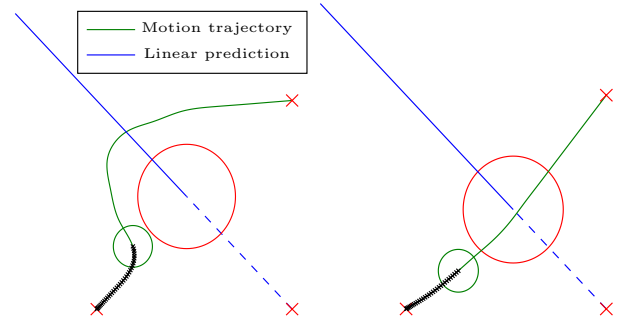


Fig. 4: Motion planning without (left) and with (right) appropriate motion model

rooms. On the left side, the figure shows how an extra obstacle influences the trajectory. On the right side, the large doorway is initially blocked by a circular obstacle. When this obstacle suddenly starts moving, the method takes this into account in the next iteration and finds a faster trajectory, denoted by the dashed line. Again, the trajectory runs through the obstacle because the method takes into account its current velocity in the obstacle's motion model.

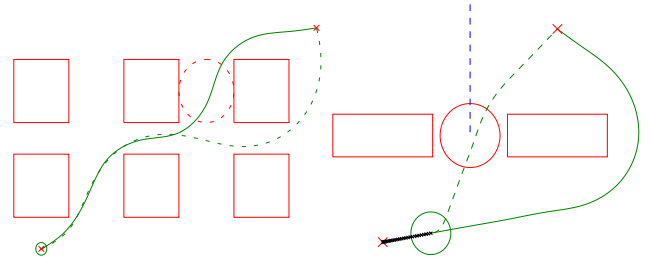


Fig. 5: Motion trajectory in case of an extra obstacle (left) or the sudden movement of an obstacle (right)

B. Comparison with time gridding

Table I gives an overview of the calculation times for three numerical examples and illustrates the advantage of using B-spline relaxations compared to classical time gridding. When using the time gridding approach the B-spline relaxations were replaced by spline evaluations at a set of grid points. For both examples the motion time $T \approx 10s$, which means $2 \frac{\text{samples}}{s}$ for 20 or $10 \frac{\text{samples}}{s}$ for 100 grid points. Note that the calculation times when using time gridding are very high, even for 20 grid points and especially for a more complex example. Furthermore, the addition of soft constraints has a much larger influence on the calculation time in that case. Finally, the problem became infeasible in some iterations while time gridding. When using B-spline relaxations the problem solved fluently and much faster.⁴

C. Experiments

Finally, the method was experimentally validated on a KUKA youBot [15]. The youBot is a small holonomic platform which can be programmed using ROS [16].

³<https://github.com/meco-group/omg-tools>

⁴All simulations were done on a notebook with Intel Core i5-4300M CPU @ 2.60GHz x 4 processor and 8GB of memory.

		relaxed	20 points	100 points
Figure 3 (solid)	t_{avg} [s]	0.038	1.23	1.29
	t_{max} [s]	0.084	28.9	4.28
Figure 3 (dashed)	t_{avg} [s]	0.044	3.28	3.38
	t_{max} [s]	0.086	50.3	5.57
Figure 5 (right)	t_{avg} [s]	0.110	5.110	94
	t_{max} [s]	0.360	11.65	75

TABLE I: Calculation times for simulations with relaxed and time gridded constraints

The experimental setup contains a ceiling camera which detects the robot via markers and then calculates its position and orientation. To get a more accurate position measurement a Kalman filter combines the camera images with odometry information of the *youBot* wheels. Furthermore the ceiling camera can also detect obstacles. The motion planning method takes the obstacle and *youBot* measurements as an input and gives a vector of velocity setpoints, representing the motion trajectory, as an output. These setpoints are applied to the vehicle as feedforward signals. If the vehicle deviates from the calculated trajectory, the method will account for this and calculate an adapted trajectory. Due to the fast update rate of both the measurements and the motion trajectory there is implicit position feedback present. Note that, for a safe operation, it is essential to include the soft constraint when calculating trajectories.

Figure 6 shows a test result for a case in which the *youBot* had to pass two stationary rectangular obstacles. Every cross denotes a new iteration of the method, every line denotes a calculated trajectory. The average calculation time per iteration was 0.05s, the maximum time was 0.45s⁵. This figure shows that feedforward velocity setpoints suffice to let the *youBot* follow the motion trajectory and that no further control methods are necessary.

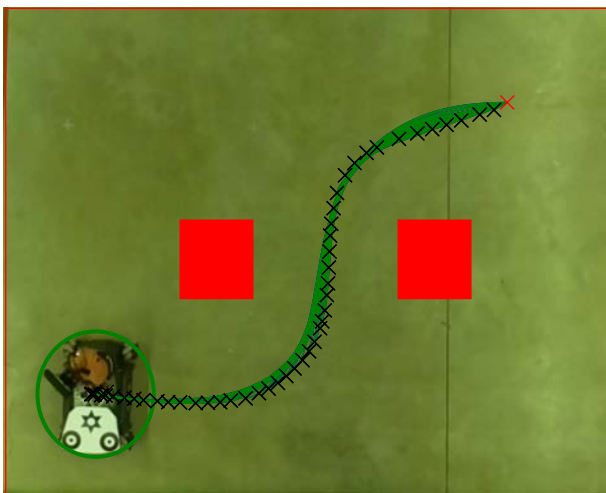


Fig. 6: Time-optimal motion of the youBot

V. CONCLUSION AND FUTURE WORK

This paper has presented a method to calculate time-optimal collision-free motion trajectories. The proposed

method can handle a holonomic vehicle moving through an environment with circular or rectangular stationary or moving obstacles. After extensive numerical simulations the method has been experimentally validated on a *KUKA youBot*. On account of an efficient problem formulation, the motion trajectory can be calculated and updated in real time. The proposed receding horizon implementation ensures that the motion planning problem is always solved with the latest world information. In the experiments, this has been shown to provide adequate position control for the *youBot*.

A first expansion of the current work is to include non-holonomic vehicles and vehicles with a non-fixed orientation. Another path which will be explored is the expansion of the proposed method to 3D-space. Possible applications are the calculation of a quadrotor flight path or the generation of picking movements for a robotic arm. Finally, the inclusion of the vehicle's dynamics into the problem formulation will be investigated.

REFERENCES

- [1] O. Mayr, *The origins of feedback control*. Cambridge, MA, USA: MIT Press, 1970.
- [2] E. Guizzo, "How google's self-driving car works," Oct. 2011.
- [3] F. Debruyere, *Optimal Robot Path Following Fast Solution Methods for Practical Non-convex Applications*. PhD thesis, Arenberg Doctoral School, KU Leuven, October 2015.
- [4] S. Lengagne, P. Mathieu, A. Kheddar, and E. Yoshida, "Generation of dynamic motions under continuous constraints: Efficient computation using B-splines and Taylor polynomials," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Taipei, Taiwan), 2010.
- [5] M. B. Milam, K. Mushambi, and R. M. Murray, "A new computational approach to real-time trajectory generation for constrained mechanical systems," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 1, (Sydney, Australia), 2000.
- [6] A. El Khoury, F. Lamiraux, and M. Taix, "Optimal motion planning for humanoid robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Karlsruhe, Germany), 2013.
- [7] B. Li and Z. Shao, "A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles," *Knowledge-Based Systems*, vol. 86, pp. 11 – 20, 2015.
- [8] W. Van Loock, G. Pipeleers, and J. Swevers, "B-spline parameterized optimal motion trajectories for robotic systems with guaranteed constraint satisfaction," *Mechanical Sciences*, vol. 6, no. 2, pp. 163–171, 2015.
- [9] W. Van Loock, G. Pipeleers, and J. Swevers, "Optimal motion planning for differentially flat systems with guaranteed constraint satisfaction," in *Proceedings of the 2015 American Control Conference (ACC)*, (Chicago), 2015.
- [10] C. de Boor, *A Practical Guide to Splines*. Springer-Verlag New York, 1978.
- [11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.
- [12] E. Camacho and C. Bordons Alba, *Model Predictive Control*. Springer-Verlag London, 2007.
- [13] J. Andersson, *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, October 2013.
- [14] A. Biegler and L. T. Wächter, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [15] R. Bischoff, U. Huggenberger, and E. Prassler, "Kuka youbot - a mobile manipulator for research and education," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.
- [16] M. Quigley *et al.*, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

⁵All tests were done on a desktop with Intel Core Xeon E5-1603 v3 CPU @ 2.8GHz x 4 processor and 16GB of memory.