Weierstraß-Institut
für Angewandte Analysis und Stochastik
Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 0946 – 8633

A coupling of discrete and continuous optimization to solve kinodynamic motion planning problems

Chantal Landry[1], Wolfgang Welz[2], Matthias Gerdts[3]

submitted: December 17, 2013

[1] Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: chantal.landry@wias-berlin.de

[2] Institute of Mathematics
Technische Universität Berlin
Str. des 17. Juni 136
10623 Berlin
Germany
E-Mail: welz@math.tu–berlin.de

[3] Institute of Mathematics and Applied Computing (LRT)
University of the Federal Armed Forces at Munich
Werner-Heisenberg-Weg 39
85577 Neubiberg
Germany
E-Mail: matthias.gerdts@unibw.de

No. 1900
Berlin 2013

2010 *Mathematics Subject Classification.* 49J15, 49M25, 49N90, 70E60, 90C30, 90C35.

Key words and phrases. Trajectory planning, optimal control problem, collision avoidance, graph search algorithm, initialization, robotics.

This work was partially supported by the German Research Foundation MATHEON.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Abstract

A new approach to find the fastest trajectory of a robot avoiding obstacles, is presented. This optimal trajectory is the solution of an optimal control problem with kinematic and dynamics constraints. The approach involves a direct method based on the time discretization of the control variable. We mainly focus on the computation of a good initial trajectory. Our method combines discrete and continuous optimization concepts. First, a graph search algorithm is used to determine a list of via points. Then, an optimal control problem of small size is defined to find the fastest trajectory that passes through the vicinity of the via points. The resulting solution is the initial trajectory. Our approach is applied to a single body mobile robot. The numerical results show the quality of the initial trajectory and its low computational cost.

1 Introduction

Time-optimal kinodynamic motion planning refers to the computation of the fastest trajectory of a robot that must avoid obstacles (kinematic constraints) and observe the dynamic laws and the bounds on the velocity or the acceleration (dynamics constraints). This expression was first introduced by Donald et al. in [8] in 1993. However, finding the optimal collision-free trajectory is an old and still topical subject in robotics, which received several names: minimum time path planning [14, 18], optimal robot path planning using the minimum-time criterion [4], trajectory planning or modeling [9, 28].

Gilbert and Johnson were the first to formulate the kinodynamic planning as an optimal control problem [18]. In their formulation, the objective function is the travel time, the dynamics laws are a set of ordinary differential equations and the collision avoidance is a state constraint. In addition, box constraints and boundary conditions are prescribed. There exist several approaches to solve this optimal control problem. The first approach was introduced by Gilbert and Johnson in [18] and extended by other authors, such as Bobrow [4] and Dubowsky et al. [9]. The technique first discretizes the state variable with B-splines and then looks for a control variable that satisfies the dynamics constraints and minimizes the travel time. It is often assumed that the control variable has a bang-bang behaviour. The technique involves then finding the switching points [5]. This first approach does not guarantee that the resulting trajectory is the fastest one, since the control variable is not necessarily optimal.

Another approach involves path planning techniques. Path planning uses a graph search algorithm to find a collision free path. For that purpose, a graph is defined on the workspace. The technique looks for a collision-free path on the graph. The simplest method in graph algorithm is A^* algorithm or Dijkstra's algorithm. In the last years, several efficient methods have been developed such as Rapid-Exploring Tree (RRT) or Probabilistic Roadmap Planner (PRM) [16, 21]. The resulting path is collision-free, but does not take into consideration the dynamics constraints. One idea was then to build a trajectory that satisfies the dynamics constraints by smoothing the path and finding the control variables, so that the dynamics constraints are observed. Again, this method does not guarantee to find the optimal solution.

LaValle and Kuffner developed a new approach which solves the drawback of path planning [22]. The path is not searched in the workspace, but in the state space. That is, LaValle and Kuffner consider the bounds on the velocity and build a graph, so that a motion between two nodes is dynamically possible.

This new approach still does not lead to the optimal solution, but at least can handle the dynamics constraints.

We model the kinodynamic planning with an optimal control problem like Gilbert and Johnson or Bobrow did. However, we use a different technique to solve the optimal control problem. We prefer to discretize the control variable since the goal is to find the best control, so that the travel time is minimized and the kinematic and dynamics constraints are satisfied.

The weakness of gradient based methods for continuous optimization problems and optimal control problems is their dependence on a good initialization, especially if obstacles have to be avoided. Without a good initial trajectory, the chances to reach the optimal solution are low. Gilbert and Johnson [14] and other authors need a collision-free trajectory to find a solution. However, nobody explains how to obtain such a trajectory. Another option is to take the straight line that joins the initial to the final position as initial trajectory. But, this initialization may contain collision. Therefore, the straight line is not good enough for complicated trajectories.

In this article, we propose a method that automatically compute a good initial trajectory. This method combines discrete and continuous optimization methods. A path planning algorithm is first applied to get a list of via points. Then, the initial trajectory is obtained by computing the fastest trajectory that passes through the vicinity of the via points. We will not use sophisticated path planning algorithm such as RRT or PRM. Our goal is to find a rough collision-free path, since the path is then post processed through an optimal control problem.

This article is divided as follows. In the next section, we formulate the kinodynamic motion planning problem as an optimal control problem. Then, our numerical method, which is based on the time discretization of the control variable, is given. Section 3 is devoted to the computation of a good initial trajectory. In Section 4, we study more precisely the collision avoidance constraint and see which numerical difficulties this constraint can cause. Depending on the quality of the initial trajectory, the kinodynamic planning problem may be slightly modified. Numerical results are presented for a two dimensional mobile robot in Section 5. Finally, conclusions and extensions to three dimensional problems are given in Section 6.

2 Problem formulation

Let us consider a two dimensional robot that is composed of a convex compact polyhedron. Extensions to more complicated robots are given at the end of Subsection 2.1 and in Section 6. The robot evolves in a space that contains fixed obstacles. These obstacles are also convex compact polyhedra. We are interested in finding the fastest collision-free trajectory of the robot that moves between two given positions. We start this section by introducing the model to find such a trajectory.

2.1 Kinodynamic motion planning

Let $P_0 \in \mathbb{R}^2$ be the initial position and $P_f \in \mathbb{R}^2$ the goal position. To describe the motion of a robot between P_0 and P_f, we need to define several variables. Let r, v and a, respectively, denote the position, velocity and acceleration of the center of gravity of the robot in the two dimensional plane. Since the robot can rotate, let θ denote the angle of rotation of the robot and μ be the velocity of the angle of rotation. Finally, let t_f be the travel time between P_0 and P_f. The motion of the robot on the time interval $[0, t_f]$ is then given by the following dynamic laws:

$$r'(t) = v(t), \qquad v'(t) = a(t) \quad \text{and} \quad \theta'(t) = \mu(t). \tag{1}$$

In this paper, a trajectory from P_0 to P_f is defined by the pair $(t, r(t))_{t\in[0,t_f]}$ that satisfies $r(0) = P_0$ and $r(t_f) = P_f$. We assume that the robot does not have any velocity and angle at the start and goal position. Therefore, the following boundary conditions are defined

$$r(0) = P_0,\, v(0) = 0,\, \theta(0) = 0,\, r(t_f) = P_f,\, v(t_f) = 0,\, \theta(t_f) = 0. \tag{2}$$

With the robot, we associate a Cartesian coordinate system, called *body frame*, whose origin is located at the center of gravity of the robot. In this coordinate system, the vertices of the robot are stored in the matrix R_0. If n is the number of vertices in the robot, then $R_0 \in \mathbb{R}^{n\times 2}$. With the workspace, we associate another coordinate system, that is called *world frame*. The coordinates of the vertices at time t in the world frame are obtained by rotating R_0 of angle $\theta(t)$ and then applying a translation of vector $r(t)$ [21]. Let $R(t)$ be the matrix of the vertices at time t. Then, we have:

$$R(t)^\top = \begin{pmatrix} \cos(\theta(t)) & -\sin(\theta(t)) \\ \sin(\theta(t)) & \cos(\theta(t)) \end{pmatrix} R_0^\top + r(t)\cdot e_n^\top, \tag{3}$$

where e_n is the vector that contains n components, which are all equal to 1. Hence, the product $r(t)\cdot e_n^\top$ yields a $2 \times n$ matrix. In what follows, R designates the robot as a convex compact polyhedron and the matrix of vertices at the same time.

Let us assume that the workspace contains K fixed obstacles, which are convex compact polyhedra and denoted by H_k, $k = 1, \ldots, K$. A trajectory is collision-free if and only if $R(t)$ and the polyhedra H_k, $k = 1, \ldots, K$, are disjoint for all t in $[0, t_f]$. There exists several manners to express that the robot R does not intersect any obstacle. For instance, we can use arguments from linear programming. Let us describe the polyhedra with the following sets of inequalities

$$R = \{y \in \mathbb{R}^2 \,|\, Ay \le b\} \text{ and } H_k = \{y \in \mathbb{R}^2 \,|\, C_k y \le p_k\},$$

with $A \in \mathbb{R}^{n\times 2}$, $b \in \mathbb{R}^n$, $C \in R^{n_k\times 2}$ and $p_k \in \mathbb{R}^{n_k}$, where n_k is the number of vertices in H_k.

Then, Farkas's lemma [2] implies that R and H_k do not intersect if and only if there exists a vector $w_k \in \mathbb{R}^{n+n_k}$ such that

$$w_k \ge 0, \quad \begin{pmatrix} A \\ C_k \end{pmatrix}^\top w_k = 0 \quad \text{and} \quad \begin{pmatrix} b \\ p_k \end{pmatrix}^\top w_k < 0. \tag{4}$$

Hence, the polyhedra R and H_k do not collide as long as such a vector w_k exists. The collision avoidance criterion is here expressed with algebraic relations, which is very convenient. However, this criterion leads also to a kinodynamic planning problem of a huge size since a vector w_k must be found for each obstacle. Moreover, the non-uniqueness of w_k may produce numerical difficulties. Please see [13, 20] for more details on this formulation.

A second way to characterize the collision avoidance is to maintain a positive distance between the robot and the obstacles. To this end, we use the signed distance between two objects, which is negative if the objects intersect and non-negative otherwise. The signed distance between intersecting polyhedra is defined as follows [7, 17, 19]:

$$d(R, H) = -\|w\|_2, \tag{5}$$

where d is the distance function and w is the smallest translational vector, so that $\text{int}(R+w)\cap H = \emptyset$. An illustration is given in Figure 1.

Figure 1: The polyhedra H and R overlap. The vector w is the smallest vector such that $R + w$ and H come into contact, where $R + w$ is the polyhedron R translated by w.

If the polyhedra are disjoint, then d is simply the Hausdorff distance. In summary, the distance function between two convex compact polyhedra is given by

$$d(R, H_k) \;=\; \begin{cases} -\|w\|_2, & \text{if } R \cap H_k \neq \emptyset, \\ \operatorname{dist}(R, H_k), & \text{otherwise,} \end{cases}$$

where dist is the Hausdorff distance.

The collision avoidance constraint is then obtained by imposing that the minimum distance between the robot and the obstacles is larger than a safety margin, that is

$$\min_{k=1,\ldots,K} d(R, H_k) \;\geq\; \varepsilon, \tag{6}$$

where $\varepsilon > 0$ is the safety margin.

In contrast to (4), the collision avoidance criterion leads to considerably fewer constraints, since it contains only one inequality. However, the difficulty here is the discontinuity of the derivative of d, which may cause problems when solving the kinodynamic motion problem. Note that d is Lipschitz continuous only and we may use subgradients at the points of non-differentiability.

As many authors [4, 14], we prefer the second approach with its small size. Therefore, combining (1), (2) with (6), we obtain the kinodynamic motion planning problem between P_0 and P_f

$$(P) : \min\; t_f \qquad \text{subject to the constraints:}$$

• equations of motion:	$r'(t) = v(t),$	a.e. in $[0, t_f]$,
	$v'(t) = a(t),$	a.e. in $[0, t_f]$,
	$\theta'(t) = \mu(t),$	a.e. in $[0, t_f]$,
• collision avoidance:	$\min_{k=1,\ldots,K} d(R(t), H_k) \geq \varepsilon,$	a.e. in $[0, t_f]$,
• boundary conditions:	$r(0) = P_0,\; v(0) = 0,\; \theta(0) = 0,$	
	$r(t_f) = P_f,\; v(t_f) = 0,\; \theta(t_f) = 0,$	
• box constraints:	$\underline{a} \leq a(t) \leq \overline{a}$ and $\underline{\mu} \leq \mu(t) \leq \overline{\mu},$	a.e. in $[0, t_f]$,

where $\underline{a}$, $\overline{a} \in \mathbb{R}^2$ and $\underline{\mu}$, $\overline{\mu} \in \mathbb{R}$ are given lower and upper bound values.

The problem (P) is an optimal control problem where the state variables are the position, r, the velocity, v, and the angle of rotation, θ. The control variables are the acceleration of the center of gravity, a, and the velocity of the rotation angle μ. Let us store the variables in the following vectors:

- the state variable: $x = (x_1, x_2, x_3, x_4, x_5) = (r, v, \theta) \in \mathbb{R}^{n_x}$,
- the control variable: $u = (u_1, u_2, u_3) = (a, \mu) \in \mathbb{R}^{n_u}$.

with $n_x = 5$ and $n_u = 3$.

Other cost functions, such as minimizing the energy consumption, can be defined in (P). Hence, the objective function is replaced by the more general formulation: $\varphi(x(0), x(t_f), t_f)$. In other words, the objective function depends on the initial state, the final state and the travel time.

Since the matrix of vertices $R(t)$ depends on $r(t)$ and $\theta(t)$ (compare (3)), the distance function is rewritten in the form $d(x(t), H_k)$. Let us now define the following functions:

- $g : \mathbb{R}^{n_x} \to \mathbb{R}$ s.t. $g(x) = \min\limits_{k=1,\dots,K} d(x, H_k)$,
- $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ s.t. $f(x, u) = \begin{pmatrix} x_3 \\ x_4 \\ u \end{pmatrix}$,
- $c : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{2n_x}$ s.t. $c(x(0), x(t_f)) = \begin{pmatrix} r(0) - P_0 \\ v(0) \\ \theta(0) \\ r(t_f) - P_f \\ v(t_f) \\ \theta(t_f) \end{pmatrix}$.

With these new definitions and after transformation onto the fixed time interval $T := [0, 1]$, the optimal control problem (P) to find the time-optimal kinodynamic motion planning can be rewritten as follows

$$
\begin{aligned}
(OCP_c) \qquad & \min \varphi(x(0), x(1), t_f) \\
& \text{with respect to } x \in W^{n_x}_{1,\infty}(T), u \in L^{n_u}_{\infty}(T) \\
\text{s.t.} \quad & x'(t) - t_f\, f(x(t), u(t)) = 0, && \text{a.e. in } T, && (7)\\
& g(x(t)) \geq \varepsilon, && \text{a.e. in } T, && (8)\\
& c(x(0), x(1)) = 0, && && (9)\\
& u(t) \in \mathcal{U}, && \text{a.e. in } T, && (10)
\end{aligned}
$$

where $\mathcal{U} := \{u = (a, \mu) \in \mathbb{R}^{n_u} \mid \underline{a} \leq a \leq \overline{a}, \underline{\mu} \leq \mu \leq \overline{\mu}\}$.

As usual $L^{n_u}_{\infty}(T)$ denotes the Banach space of essentially bounded functions mapping from T into $\mathbb{R}^{n_u}$ and $W^{n_x}_{1,\infty}(T)$ denotes the Banach space of absolutely continuous functions with essentially bounded derivative that map from T into $\mathbb{R}^{n_x}$.

In (OCP_c), the dynamics constraints are the dynamic laws (7) and the box constraints (10). The state constraint (8) and the boundary conditions (9) define the kinematic constraints. Moreover, let us observe that the constraint (8) is not continuously differentiable because of the distance function d.

In the case of a three-dimensional robot, the kinodynamic planning problem (OCP_c) would have the same structure. The changes are the meaning of the variables and the definition of the function f in the ordinary differential equations. Hence, the development in the next sections to solve (OCP_c) can also be applied in the three-dimensional case.

2.2 Numerical method

Most of the methods which have been developed to solve (OCP_c), first discretize the state variable x and then look for a control that satisfies the dynamics constraints and minimizes the cost functional [4, 18]. Such methods will never guarantee that the final solution leads to the optimal trajectory.

Since (OCP_c) contains state constraints and the dimension of the state variable is small, we choose to use a direct method [13]. But, in contrary to [4, 18], we discretize the control variable and then utilize an explicit integration scheme to solve the ordinary differential equations. The time discretization of (OCP_c) leads to a non-linear optimization problem.

Let us consider a grid with a fixed step-size:

$$\mathbb{G}_N := \{t_k \;=\; k\,h \,|\, k = 0, 1, \ldots, N\}, \text{ with } h = t_f/N.$$

The control variable is approximated with B-splines as follows

$$u_h(t; u_0, \ldots, u_{N+r-2}) := \sum_{i=0}^{N+r-2} B_{ir}(t)\, u_i,$$

where $B_{ir}, i = 0, \ldots, N+r-2$, are elementary B-splines of order r on $\mathbb{G}_N$ and $(u_0, \ldots, u_{N+r-2})^\top \in \mathbb{R}^{n_u(N+r-1)}$ is the vector of de Boor points. The choice of B-splines is convenient since the elementary functions have a local support only.

As the elementary B-splines of order r sum up to one for all $t \in T$, the box constraints $u_h(t) \in \mathcal{U}$ are satisfied, if

$$u_i \in \mathcal{U}, \quad i = 0, \ldots, N + r - 2.$$

The ordinary differential equations (7) are integrated by an explicit one-step method, such as Runge-Kutta method. According to (7), the integration scheme depends on the initial value $x_0 = (P_0, 0, 0, 0) \in \mathbb{R}^{n_x}$, the travel time t_f and the de Boor points u_i, $i = 0, \ldots, N + r - 2$. Let us store these variables in the following vector

$$z := (x_0, u_0, \ldots, u_{N+r-2}, t_f)^\top \in \mathbb{R}^{n_z},$$

with $n_z = n_x + (N + r - 1)n_u + 1$. Then, the one-step integration method with increment function Φ leads to the state approximations

$$x_{k+1}(z) = x_k(z) + h\Phi(t_k, x_k(z), u_h(t_k; u_0, \ldots, u_{N+r-2}), t_f, h), \qquad k = 0, \ldots, N-1, \quad (11)$$

at the grid points t_k, $k = 0, \ldots, N$.

Introducing both the control and state approximations into the optimal control problem (OCP_c) leads to:

$$\begin{aligned} (NLP) \qquad & \min_z J(z) \\ \text{s.t.} \qquad & c(x_0, x_N(z)) = 0, \\ & g(x_k(z)) \geq \varepsilon, \qquad\qquad k \;=\; 0, \ldots, N, \\ & z \;\in\; \mathcal{Z}, \end{aligned}$$

where $J(z) := \varphi(x_0, x_N(z), t_f)$, $\mathcal{Z} := \{z \in \mathbb{R}^{n_z} \,|\, \underline{z} \leq z \leq \overline{z}\}$ with $\underline{z} = (x_0, \underline{a}, \underline{\mu}, \ldots, \underline{a}, \underline{\mu}, \underline{t})$, $\overline{z} = (x_0, \overline{a}, \overline{\mu}, \ldots, \overline{a}, \overline{\mu}, \overline{t})$ and $\underline{t}, \overline{t} \geq 0$ are given lower and upper bound values.

The problem (NLP) is a finite dimensional non-linear optimization problem, that we solve by using a Sequential Quadratic Programming (SQP) method. SQP methods are iterative methods [1]. At each iteration, a quadratic programming (QP) sub-problem is being solved to find a search direction. Let $\mathcal{L}$ be the Lagrange function of (NLP) and let λ, $\sigma = (\sigma_0, \ldots, \sigma_N)$ and ζ be the multipliers associated to the constraints. Then, the Lagrange function is given by

$$\mathcal{L}(z, \lambda, \sigma, \zeta) = J(z) + \lambda^\top c(x_0, x_N(z)) - \sum_{k=0}^{N} \sigma_k (g(x_k(z)) - \varepsilon) + \zeta^\top z.$$

The quadratic programming sub-problems are obtained by linearising the equality and inequality constraints as follows

$$\begin{aligned}
(QP) \qquad \min_d \ & \frac{1}{2} d^\top \nabla^2_{z,z} \mathcal{L}(z, \lambda, \sigma, \zeta) d + \nabla J(z)^\top d \\
\text{subject to} \quad & c(x_0, x_N(z)) + A(z) d = 0, \\
& g(x_k(z)) + \xi^\top d \geq \varepsilon, \qquad k = 0, \ldots, N, \\
& z + d \in \mathcal{Z},
\end{aligned}$$

where ξ is a subgradient of $g(x_k(z))$, d is the search direction to define the next iterate and $A(z)^\top := [\nabla c_1(x_0, x_N(z)) \cdots \nabla c_{2n_x}(x_0, x_N(z))]$ is the Jacobian matrix of the boundary constraint c.

The exact Hessian matrix $\nabla^2_{z,z} \mathcal{L}(z, \lambda, \sigma, \zeta)$ is not well defined at the points of non-differentiability of g. Therefore, we prefer to approximate the Hessian by using BFGS update formulas [3, 26]. Let B_ℓ denote the approximation of the Hessian at iteration ℓ. The BFGS update formulas guarantee that B_ℓ is symmetric and positive definite. Thus the quadratic sub-problems are strictly convex. The local SQP method to solve (NLP) can now be sketched:

Prototype local SQP algorithm to solve (NLP)

1 Choose $z^{(0)} \in \mathcal{Z}$, $\lambda^{(0)}$, $\zeta^{(0)}$ and $\sigma^{(0)}$. Initialize B_0 with the identity matrix. Set $\ell := 0$.

2 If $(z^{(\ell)}, \lambda^{(\ell)}, \zeta^{(\ell)}, \sigma^{(\ell)})$ is a KKT point, STOP.

3 Compute a KKT point d of the following quadratic sub-problem:

$$\begin{aligned}
(QP_\ell) \qquad \min_d \ & \frac{1}{2} d^\top B_\ell d + \nabla J(z^{(\ell)})^\top d \\
\text{subject to} \quad & c(x_0, x_N(z^{(\ell)})) + A(z^{(\ell)}) d = 0, \\
& g(x_k(z^{(\ell)})) + \xi^\top d \geq \varepsilon, \qquad k = 0, \ldots, N, \\
& z^{(\ell)} + d \in \mathcal{Z},
\end{aligned}$$

where ξ is a subgradient of $g(x_k(z^{(\ell)}))$.

4 Set $z^{(\ell+1)} := z^{(\ell)} + d$. Update $\lambda^{(\ell+1)}$, $\zeta^{(\ell+1)}$ and $\sigma^{(\ell+1)}$ with the Lagrange multipliers for the constraints in (QP_ℓ). Compute $B_{\ell+1}$ according to BFGS formulas. Set $\ell := \ell + 1$ and go to 2.

At each iteration ℓ of the SQP algorithm, an update $z^{(\ell)}$ is obtained. From this vector, a trajectory $(t_k, r_k(z^{(\ell)}))_{k=0,\ldots,N}$ is built by integrating the ordinary differential equations as explained in (11), where $x_k(z^{(\ell)}) = (r_k(z^{(\ell)}), v_k(z^{(\ell)}), \theta_k(z^{(\ell)}))$. Therefore, a sequence of trajectories is issued from the SQP method.

Without a good initialization of the starting point $z^{(0)}$, the SQP method might not converge. In order to achieve convergence from remote points, the SQP algorithm can be augmented by a globalization strategy. See [27, 29] for line-search based methods and [10, 11] for filter methods. However, for our particular problem, a good initialization of $z^{(0)}$ can be found. A method to compute such an initialization for (NLP) is developed in the next section. This method is based on the coupling of discrete and continuous optimization.

3 Initialization

In kinodynamic planning, everyone agrees with the significance of a good initial trajectory [18, 31], but nobody provides a method to find such a trajectory. This trajectory must be close to the optimal trajectory and very often collision-free. Let us consider the example depicted in Figure 2-(a) to illustrate this necessity. The workspace is a rectangle. The black quadrilateral is an obstacle and is in contact with the boundary of the workspace. We need to find a way to indicate in which direction the robot must move to reach P_f. If $z^{(0)}$ were initialized such that the associated trajectory were the segment $[P_0, P_f]$, the robot would move through the obstacle. The solver would then attempt to drive the trajectory downwards in order to eliminate any collision. But, no such trajectory can succeed since the robot cannot reach P_f by passing on the left of the obstacle without moving outside the workspace. Here, a good initial direction would be to go first upwards and then turn left.

In this section, we develop a two-step method to compute a starting point $z^{(0)}$ such that the associated initial trajectory $(t_k, r_k(z^{(0)}))_{k=0,\dots,N}$ has a favorable shape. First, a path-planning algorithm is used to find via points. These points indicate the robot the direction to reach P_f. In the second step, $z^{(0)}$ is computed, so that the initial trajectory passes through the neighborhood of the via points.

3.1 Computation of the via points

Path-planning involves searching a collision-free path in the workspace between two given positions. Several techniques exist such as graph search algorithms, cell decomposition, potential field method, probabilistic roadmap (PRM) or rapid exploring random trees (RRT). See [16, 21] for an exhaustive review. Here, we do not need to use sophisticated methods. We look for a path that is collision-free, but not necessary the shortest. In addition, the path must have a small number of turns. If many changes of direction occur (Figure 2-(a) illustrates such a case), then the second step, presented in the next subsection, could misbehave. To find such a path, we use an adaptation of classical roadmap methods such as Dijkstra's algorithm.

Let ρ be the radius of the smallest circle that contains the entire robot. Let cover the workspace by a regular grid. Here, grid nodes only exist, if they do not correspond to a coordinate lying on an obstacle or too close to an obstacle, i.e. the distance between the robot and the obstacle is smaller than ρ.

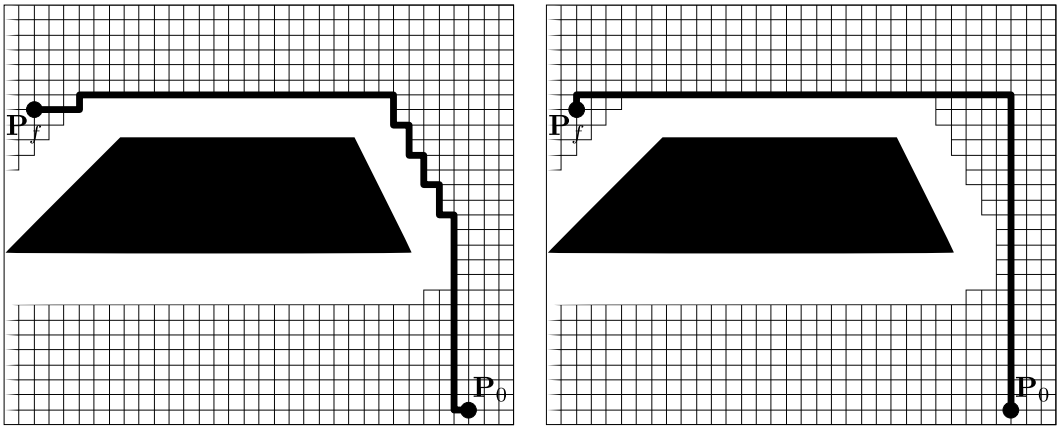
For simplicity, we only allow horizontal and vertical movements and the distance between two grid points is set to the constant δ. Further, let s and τ be the closest grid points to P_0 and P_f respectively. Since many turns may pose a problem, one solution is to calculate the shortest s-τ-path with the least amount of turns instead. This is a concept which is a common approach in many path planning problems [6, 25]. It can easily be achieved by a modification of the Dijkstra's algorithm: Each time the distance for a new node v is being calculated by extending an existing path over the edge (u, v), we add M to the path cost, if this extension introduces a turn in u.

Unfortunately, even if M is chosen larger than the number of nodes in the grid, it can still happen, that there are multiple turn-minimal shortest s-u path of which only one can be extended to a shortest turn-minimal s-v path containing u.

The easiest way to deal with such turn minimal paths is a graph modification: One original grid node is split up into four nodes, one for every possible direction. These nodes are then connected in such a way, that edges corresponding to turns have cost M and the other edges have cost 0. This blows up the size of graph by a factor of four, but Dijkstra's algorithm can still be used on this extended graph.

Another way of representing turn costs is by a so-called pseudo-dual graph. In such a graph the edges are represented by nodes and the turn costs as well as the distance penalties by arcs [32].

Figure 2-(b) shows the result of such a turn-minimal computation.



(a) (b)

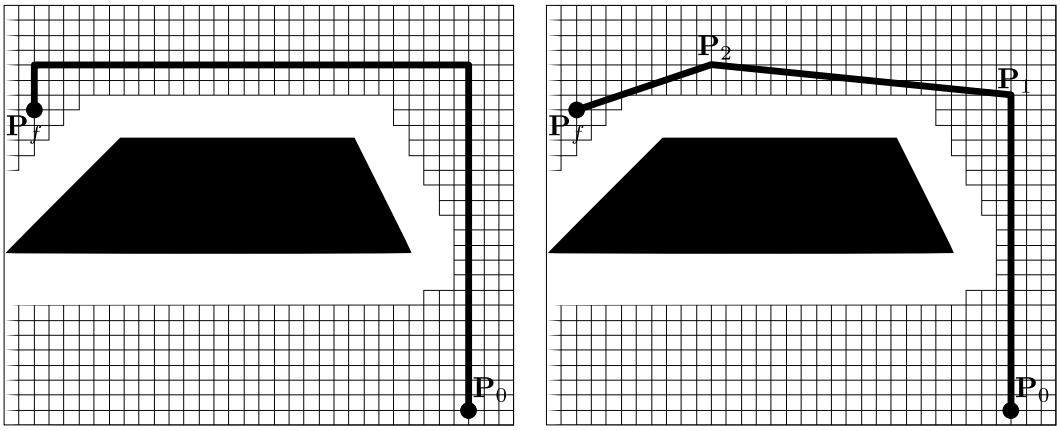
Figure 2: (a) One possible path from P_0 to P_f that uses the least amount of edges in the grid. (b) This path uses the same amount of edges, i.e. has the same length, as the path in (a), but contains only two turns.

By definition, all paths on the grid are feasible. All nodes that could possibly conflict with any obstacle are not included in the graph. So far, we only performed shortest path computations. Due to the nature of such paths, trajectories are often favoured which pass by an obstacle as close as possible. However, these parts of the path make it harder for the exact trajectory computation. In some situations it would be much better to take a path that is slightly longer, but on the other hand keeps a larger distance to the obstacles. This can be achieved by adding specific costs to the nodes. These costs should depend on the distance of node to its nearest obstacle and should drastically decrease with increasing obstacle distance.

However, in this context it makes no longer sense to find the shortest path with least turns. We want to explicitly allow some additional turns, if that helps to avoid close obstacles. Thus, we introduce the concept of *turn costs* to penalize paths with a higher amount of turns without forbidding them explicitly.

Shortest path with turn costs can be solved with the presented modifications of Dijkstra's algorithm alone, by either connecting the four newly created nodes with respect to the given turn costs for $90\,^\circ$, $180\,^\circ$ and $270\,^\circ$ turns or by using the pseudo-dual approach.

Turn costs together with the distance dependent node penalties now give us good options to alter the resulting path in such a way that it is balanced and will most probably also lead to a good starting solution.



(a) (b)

Figure 3: (a) Path that avoids close obstacles without taking to many turns. However, his path is slightly longer than the length of the shortest path in Figure 2. (b) Smoothened version of the path in (a).

So far, the resulting path only contains right angles, which is also not very well suited for further trajectory calculations. It is necessary to smoothen the path even further. One approach, that showed significant improvements in practical experiments, is the following: Starting in s we move along the so far calculated path and check for each such node v_i, whether the line segment $\overline{sv_{i-1}}$ from the point corresponding to s to the point corresponding to v_i has a distance to all obstacles of at least ρ and therefore can be used by the robot without any collisions. When we reached a point v_i where such a collision occurs, we make $\overline{sv_{i-1}}$ the beginning of our new smoothened path. We then continue along the original path further until we find a node v_j where $\overline{v_{i-1}v_j}$ is too close to an obstacle and add $\overline{v_{i-1}v_{j-1}}$ to the smoothened path. This procedure is continued until we have reached τ. See Figure 3-(b) for an example of such a smoothened path. The inner nodes v_i in the smoothened path are the via points that will be used in the next sub-section.

Applying this smoothing process we do not longer explicitly avoid trajectories which are too close to the obstacles. However, by using the nodes of a shortest path that takes this distance into account, we assure that at least the start and end nodes of the line segments are farther away. This fact leads to sufficiently good trajectories.

To speed up the path computation process the grid is created dynamically: The node obstacle penalties are only calculated when the specific node is visited for the first time in the shortest path calculation. There is also no need to generate and store the edges in advance, since they are implicitly given by the node coordinates.

3.2 Computation of the initial trajectory

In Section 3.1, the via points were computed as the nodes of a smoothened path between P_0 and P_f. Let n_I be the number of via points and P_i be the i^{th} via point.

We search for an initial value of $z^{(0)}$. The initialization is such that the associated initial trajectory is the fastest trajectory that passes through the neighborhood of the via points.

Let us start by recalling the definition of the unknown vector z:

$$z^{(0)} := (x_0, u_0^0, \ldots, u_{N+r-2}^0, t_f^0).$$

In our case, x_0 is known since $x_0 = (r(0), v(0), \theta(0)) = (P_0, 0, 0)$. The remaining unknowns, that are the de Boor points u_i^0, $i = 1, \ldots, N + r - 2$ and the initial travel time t_f^0, are initialized by solving an optimal control problem, which is very similar to (OCP_c). In this new problem, no obstacle is considered. Instead, a penalty term that forces the trajectory to pass through the neighborhood of the via points is added in the objective function. The penalty term is of the form $\|P_i - r(t)\|_2 < \varepsilon_r$, where ε_r is a positive parameter. The penalty means that the center of gravity of the robot must be, at time t, in the ball centered at P_i and of radius ε_r. This condition must not be applied at all $t \in [0, t_f^0]$, but only during a small time interval, included in $[0, t_f^0]$. Finding the time subinterval depends on the via points and the number N of grid points used in the time discretization of (OCP_c). Naturally, the control grid of (OCP_c) and of the new optimal control problem for the initialization, is the same.

In Section 2, the control grid $\mathbb{G}_N$ was defined. In practice, the number N must be determined. This number is chosen proportional to the length of the path computed in the discrete search. Let dl define the distance travelled by the robot between two time-steps of $\mathbb{G}_N$. The variable dl is a desired distance that is fixed at the beginning. Let L be the length of the path starting from P_0, passing through P_i, $i = 1, \ldots, n_I$, and ending at P_F, as illustrated in Figure 3-(a). By definition, we have:

$$L := \sum_{i=0}^{n_I} L_i = \sum_{i=0}^{n_I} \|\overrightarrow{P_i P_{i+1}}\|_2, \text{ where } P_{n_I+1} = P_F.$$

Then, the number of grid points is defined as

$$N = \begin{cases} \lfloor \frac{L}{dl} \rfloor, \text{ if } \frac{L}{dl} - \lfloor \frac{L}{dl} \rfloor \leq 0.5, \\ \lfloor \frac{L}{dl} \rfloor + 1, \text{ otherwise,} \end{cases}$$

where $\lfloor \cdot \rfloor$ is the floor function.

The initial trajectory $(t_k, r_k(z^0))_{k=0,\ldots,N}$ is such that the trajectory passes through a neighborhood of the via points P_i, $i = 1, \ldots, n_I$, at certain time steps. These time steps are chosen according to the distance travelled by the robot from the initial position to the via point. For each point P_i, we set the index of the time steps, $k(i)$, as follows:

$$k(i) := \left\lfloor \frac{\sum_{k=0}^{i} L_k}{L} N \right\rfloor + 1, \ i = 1, \ldots, n_I.$$

Finally, we check if there is always at least one time step between two via points, that is $k(i+1) - k(i) > 1$. If this is not the case, a time step is added between $k(i)$ and $k(i+1)$. In other words, the index $k(i+1)$ is moved forward from one, that is $k(i+1) := k(i+1) + 1$. Similarly, the number of grid points is changed into $N = N + 2$ to be sure that there is at least one time step between P_{n_I} and P_F. This trick yields to a better behaviour of the algorithm that computes the initial trajectory.

The initial trajectory is the fastest trajectory which joins P_0 to P_F and passes through the neighborhood of P_i at $t_{k(i)}$, $i = 1, \ldots, n_I$. To find this initial trajectory, we solve an optimal control problem where the conditions at the via points are specified as a penalty term in the objective function. Therefore, the

problem reads

$$
\begin{aligned}
(OCP_I) \qquad & \min\ c_1\, t_f + c_2 \int_0^{t_f} p(t)dt \\
\text{s.t.}\quad & x'(t) - t_f f(x(t), u(t)) = 0, \text{ a.e. in } T, \\
& c(x(0), x(1)) = 0, \\
& u(t) \in \mathcal{U}, \text{ a.e. in } T,
\end{aligned}
$$

where

$$
p(t) := \begin{cases} \max(\|P_i - r(t)\|_2 - \varepsilon_r, 0)^2, \text{ if } \exists i \in \{1, \dots, n_I\} \text{ s.t. } |t - t_{k(i)}| \leq \frac{h}{2}; \\ 0, \text{ otherwise;} \end{cases}
$$

where h is the fixed step size of $\mathbb{G}_N$ and c_1 and c_2 are positive constants.

The penalty term p is positive when the position of the center of gravity during $[t_{k(i)} - \frac{h}{2}, t_{k(i)} + \frac{h}{2}]$ is not located in the ball centered at P_i and of radius ε_r. As soon as the center of gravity of the robot is in the ball, the penalty term is equal to 0.

The constraints in (OCP_I) already appear in (OCP_c). Indeed, only the collision avoidance constraint in (OCP_c) was removed to define (OCP_I). Thus, (OCP_I) has a smaller size and does not contain non-differentiable constraints anymore.

To solve (OCP_I), the same technique as for (OCP_c) is used. If $u^*_{I,i}$, $i = 0, \dots, N + r - 2$, is the optimal control variable of (OCP_I) and if $t^*_{f,I}$ is the optimal travel time of (OCP_I), then the unknown vector $z^{(0)}$ is initialized as follows:

$$
t^0_f = t^*_{f,I} \text{ and } u^0_i = u^*_{I,i},\ i = 0, \dots, N + r - 2.
$$

Furthermore, the lower bound $\underline{t}$ and upper bound $\overline{t}$, defined in $\mathcal{Z}$, can be set as follows: $\underline{t} = \frac{1}{10} t^0_f$ and $\overline{t} = 10\, t^0_f$.

Numerical examples are given in Figure 4. The workspace is a rectangle that contains four obstacles (black quadrilateral). For each example, P_0, P_f and the via points are represented by a black square. The grey line is the initial trajectory obtained by solving (OCP_I), where the crosses indicate the center of gravity of the robot at the time steps t_k, $k = 0, \dots, N$.

We can observe that all initial trajectories are good candidates since they pass between the right obstacles to reach P_f. The optimal trajectory and the initial trajectory are actually homotopic relative to their endpoints.

Since (OCP_I) does not take into consideration the obstacles, the initial trajectory is not necessarily collision-free. However, in that case, the robot overlaps the obstacles only slightly. The trajectory in Figure 4-(b) illustrates such a situation. The slight collisions do not have any consequences on the solving algorithm of (OCP_c), as we will see in the next section.

For the examples defined in the workspace of Figure 4, the number of via points varies between 1 and 3. If there is no obstacle between P_0 and P_f, no via point is defined and the initial trajectory is a straight line. Since no via point exists and no collision can happen, (OCP_I) and (OCP_c) are equivalent: the penalty term in (OCP_I) is equal to 0 and the state constraint in (OCP_c) can be omitted. In this particular case, (OCP_I) gives the optimal solution.

Through these examples, we can see that a few number of via points suffices to compute the initial trajectory. If this number were large, or if the initial path would have unnecessary turns, then the number

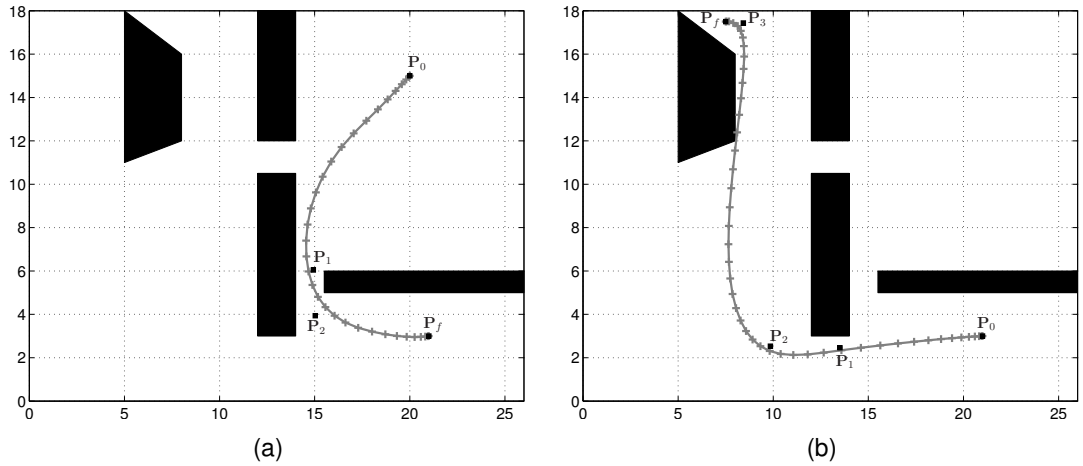


Figure 4: Examples of initial trajectories and their associated via points which come from the path planing algorithm.

of time steps N would become large, since the path is longer and we require that at least one time step exists between $t_{k(i)}$ and $t_{k(i+1)}$. It may then be more difficult to find a trajectory that passes through the neighborhood of all the via points and satisfies the dynamics constraints.

4 Fastest collision-free trajectory

In the previous section, a two-step method was introduced to find an initial value for the control variable $z^{(0)}$ in the local SQP algorithm. The initialization is such that $z^{(0)}$ satisfies the lower and upper bounds on z, i.e. $z^{(0)} \in \mathcal{Z}$. Furthermore, the associated initial trajectory $(t_k, r(z^{(0)}))_{k=0}^{N}$ is homotopic to the optimal trajectory.

According to the local SQP algorithm, the Lagrange multipliers $\lambda^{(0)}$, $\zeta^{(0)}$ and $\sigma^{(0)}$ must also be initialized. Let us remind that the constraints related to $\lambda^{(0)}$ and $\zeta^{(0)}$ appear both in (OCP_I) and (OCP_c). Consequently, these multipliers can be initialized with the solution of (OCP_I), that is:

$$\lambda^{(0)} = \lambda_I^* \text{ and } \zeta^{(0)} = \zeta_I^*,$$

where λ_I^* and ζ_I^* are the multipliers issued from (OCP_I).

The last multiplier, $\sigma = (\sigma_0, \ldots, \sigma_N)$, corresponds to the inequality constraints defined in (NLP):

$$g(x_k(z)) \geq \varepsilon,\ k = 0, \ldots, N.$$

These inequalities describe the collision avoidance between the robot and the obstacles. The complementarity condition in the Karush-Kuhn-Tucker conditions implies that σ_k is equal to 0 if the inequality constraint is inactive [26]. This means that σ_k is zero when no collision occurs between the robot and the obstacles at t_k. Because the initial trajectory is almost collision-free, we set σ to 0.

With this initialization, (OCP_c) can be solved as described in Section 2. A sequence of quadratic programming problems based on the linearization of the constraints is built. However, since the state constraint is not continuously differentiable, the solving algorithm might not converge. In this case, we observed that the state constraint, the box constraints and the boundary conditions are fulfilled, but the

Karush-Kuhn-Tucker conditions of (QP_ℓ) remain unsatisfied. This is due to the discontinuity of the gradient of g. When such a situation occurs and the number of iterations in the SQP method is larger than a threshold I, the optimal solution will be never found. However, the current trajectory is not so bad, since the robot does not intersect the obstacles (the state constraint is satisfied). Hence, our idea is to stop the SQP method and consider a new problem, where the collision avoidance is no more written as a state constraint, but expressed as a penalty term. Here is the new model

$$\begin{aligned}(OCP_p) \qquad \min\ & \alpha\, t_f + \beta \int_0^{t_f} \left(\min(g(x(t)) - \varepsilon, 0)\right)^2\, dt \\ \text{s.t.}\quad & x'(t) - t_f\, f(x(t), u(t)) = 0, \quad \text{a.e. in } T, \\ & c(x(0), x(1)) = 0, \\ & u(t) \in \mathcal{U}, \quad \text{a.e. in } T,\end{aligned}$$

where α and β are positive parameters.

This model is again an optimal control problem, but without any state constraint. The penalty term in the objective function is equal to 0 if the trajectory is collision-free. To solve (OCP_p), the numerical method introduced in Section 2 is used. No problem of discontinuity of the derivatives exists here since the term related to the distance is to the power 2 and thus it is continuously differentiable.

The initial value of $z^{(0)}$, $\lambda^{(0)}$, $\zeta^{(0)}$ and $\sigma^{(0)}$ are the last value of $z^{(\ell)}$, $\lambda^{(\ell)}$, $\zeta^{(\ell)}$ and $\sigma^{(\ell)}$ obtained in the SQP method to solve (OCP_c). The corresponding initial trajectory is collision-free. Consequently, the penalty term is equal to 0. The term becomes positive as soon as the robot approaches an obstacle, that is when the distance between them is smaller than the safety margin ε.

The goal of (OCP_p) is to minimize the travel time of a collision-free trajectory, while preventing it from collision with the penalty term. The goal of (OCP_c) was more to transform the initial trajectory into a collision-free one. If the initial trajectory is already collision-free, that is if $g(x_k(z^{(0)})) \geq \varepsilon$, $k = 0, \ldots, N$, then (OCP_p) is solved directly. A summary of this strategy is presented in Figure 5. First, the via points are computed by using a path planning algorithm. Then, (OCP_I) is solved to find a good initial trajectory. If this trajectory intersects some obstacles, then (OCP_c) is considered. If (OCP_c) does not succeed, but the constraints are satisfied, then (OCP_p) is called. On the contrary, if the initial trajectory is collision-free, then (OCP_p) is solved.

5 Numerical results

To solve (OCP_I), (OCP_c) and (OCP_c) numerically, we use the package OCPID-DAE1 developed by M. Gerdts (see www.optimal-control.de). The method introduced in Section 2 is implemented in this package. B-splines of first order ($r = 1$) are chosen to approximate the control variables. The classical Runge-Kutta method is utilized to integrate the ordinary differential equations.

Two main approaches exist to compute $d(R, H_k)$, the Hausdorff distance between the robot R and an obstacle H_k. The first approach is Gilbert, Johnson and Keerthi's algorithm published in 1988 [15] and referred as GJK. Gilbert et al. established that the distance d is equivalent to the Hausdorff distance of the Minkowski difference $R - H_k$ from the origin O. If R and H_k are separated, then O is outside $R - H_k$. In the case of a collision, O is inside $R - H_k$. Furthermore, the Hausdorff distance is equal to the norm of the vector w that characterizes the penetration depth (see (5)).

The second approach is Lin and Canny's algorithm [23, 24]. This algorithm determines the closest pair of features between the polyhedra, where the features of a polyhedron are its vertices, its edges and its

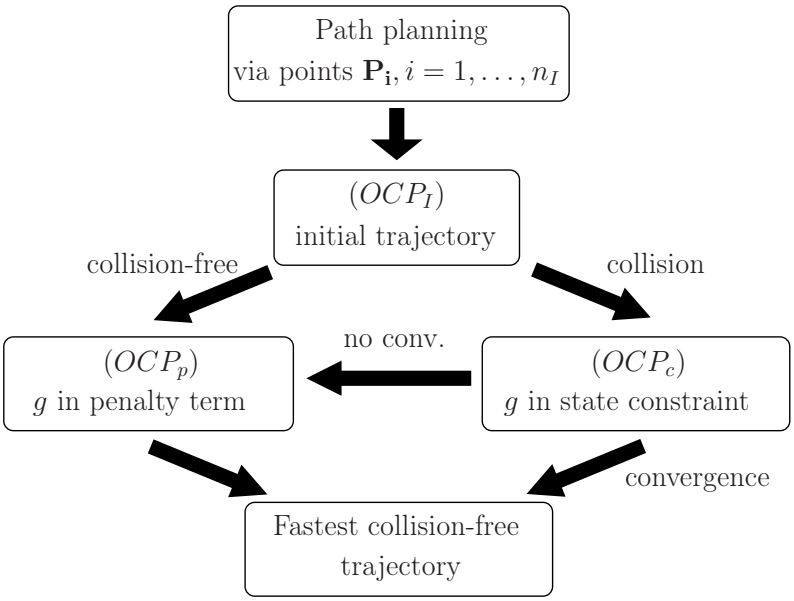


Figure 5: Scheme of the strategy to find the fastest collision-free trajectory of a robot between two given positions.

faces located on its boundary. The distance $d(R, H_k)$ is then equal to the distance between the features of the closest pair. The approach is fast, easy to implement and perfectly suited when polyhedra move slightly between two time steps. In that case, the method is of order 1. However, the approach is not adapted for penetrating polyhedra. Consequently, we apply Lin and Canny's algorithm as long as R and H_k are separated. Once $d(R, H_k)$ is smaller than ε, GJK is used in order to get a signed distance if R and H_k overlap.

Let us consider the two-dimensional example presented in Figure 6-(a). The workspace is the rectangle $[0, 26] \times [0, 18]$. The black quadrilaterals are the obstacles. The robot is a square whose vertices in the body frame are

$$R_0 = \begin{pmatrix} -0.5 & 0.5 \\ -0.5 & -0.5 \\ 0.5 & -0.5 \\ 0.5 & 0.5 \end{pmatrix}.$$

The bounds on the acceleration of the robot are $\overline{a}^\top = (1.0, 1.0)$ and $\underline{a} = -\overline{a}$. The bounds on the velocity of the rotation angle are $\overline{\mu} = \frac{\pi}{10}$ and $\underline{\mu} = -\overline{\mu}$.

The points V_i, $i = 0, \ldots, 6$ in the workspace define possible starting and end positions for the robot. We compute the fastest collision-free trajectory for any pair (P_0, P_f) with $P_0 \in \{V_0, \ldots, V_6\}$ and $P_f \in \{V_0, \ldots, V_6\} \setminus \{P_0\}$. For the numerical computations, we set the safety margin ε to 0.05, the distance dl to 0.6 and the threshold I in (OCP_c) to 30. The parameters in the objective function in (OCP_I) are chosen as follows: $\varepsilon_r = 0.5$ and $c_1 = c_2 = \frac{1}{(1+p_I)^2}$, where p_I is the maximum distance between the via points and the line passing through P_0 and P_f. Finally, α and β in (OCP_p) are respectively set to 1 and 10^4.

The numerical results are presented in Table 1. The starting and end positions of the trajectory are indicated in the first two columns. In the third column, the number of time steps for the discretization

is shown. The columns It_I, It_c and It_p give the number of iterations used in the SQP method to solve (OCP_I), (OCP_c) and (OCP_p) respectively. The columns D stand for the minimum distance between the robot and the obstacles along the trajectory, that is:

$$D = \min_{0 \le k \le N} g(x_k(z)).$$

This distance is given for the initial trajectory (fifth column) and the final trajectory (tenth column). For (OCP_c), we quantify the satisfaction of the constraints (CN) and the Karus-Kuhn-Tucker conditions (KKT). More precisely, CN and KKT are defined by

$$\begin{aligned} \text{CN} = \max \Bigg(& \max_{1 \le i \le n_z} (0, z_i^c - \overline{z}_i), \max_{1 \le i \le n_z} (0, \underline{z}_i - z_i^c), \\ & \max_{0 \le k \le N} (0, \varepsilon - g(x_k(z^c))), \|c(x_0, x_N(z^c))\|_\infty \Bigg), \\ \text{KKT} = & \|\nabla_z \mathcal{L}(z^c, \lambda^c, \sigma^c, \zeta^c)\|_\infty, \end{aligned}$$

where z^c, λ^c, σ^c and ζ^c are the outputs of (OCP_c). If (OCP_c) converges, then z^c is the optimal solution and $(t_k, r_k(z^c))_{k=0,\dots,N}$ is the fastest collision-free trajectory. Finally, the last column in Table 1 stands for the computational time, which is given in second.

The sequence of the columns follows the strategy presented in Figure 5. This means that (OCP_I) is solved first. If D is smaller than the safety margin ε, then (OCP_c) is considered. If (OCP_c) does not succeed after $I = 30$ iterations, but the constraints are satisfied (CN almost 0), then (OCP_p) is called. On the contrary, if D is larger than ε, then (OCP_p) is solved. In that case, the components for It_c, CN and KKT in Table 1 are left blank. Similarly, a blank in It_p means that we do not need to call (OCP_p) for solving the kinodynamic problem.

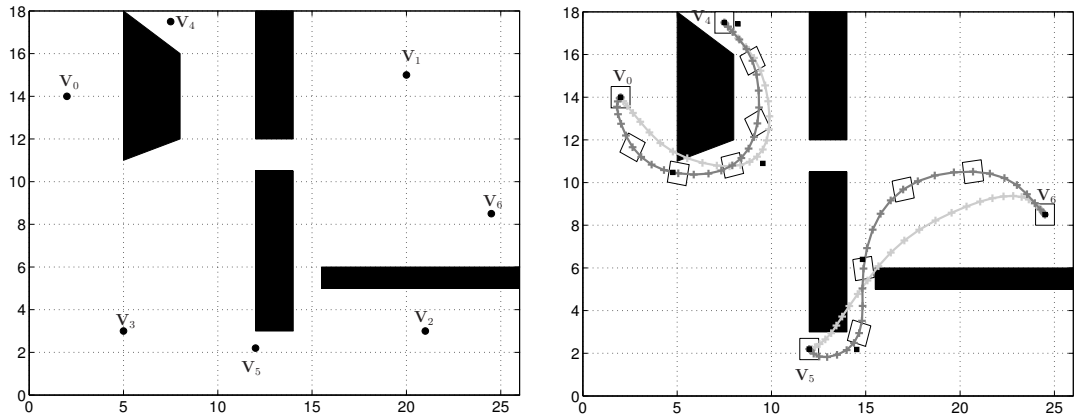


Figure 6: (a) The workspace with four obstacles (quadrilaterals) and seven points V_0 to V_6 between which the optimal trajectory is computed. (b) The optimal trajectory between V_0 and V_4 and between V_5 and V_6. The black squares are the via points, the light grey line is the initial trajectory and the dark grey line is the optimal trajectory. The robot is indicated at several time steps by a white square.

All the possible trajectories between the points V_0 to V_6 are computed successfully. Moreover, the computational time is always a matter of few seconds. The initial trajectories are mostly with collision. However, the penetration depth is not large and (OCP_c) always succeeds in finding a collision-free trajectory (CN

P_0	P_f	N	It$_I$	D	It$_c$	CN	KKT	It$_p$	D	CPU
V_0	V_1	38	182	-0.885	34	0.2E-05	0.5E-02	205	0.050	4.027
V_0	V_2	43	83	0.250				163	0.050	2.639
V_0	V_3	20	10	1.435				0	1.435	0.066
V_0	V_4	30	219	-0.597	31	0.1E-06	0.2E+00	159	0.050	2.696
V_0	V_5	28	22	-0.258	13	0.5E-14	0.2E-12		0.050	0.468
V_0	V_6	43	75	-0.763	38	0.1E-05	0.6E-02	323	0.050	6.932
V_1	V_0	37	173	-0.557	53	0.8E-06	0.5E-01	277	0.049	6.171
V_1	V_2	33	91	-0.175	31	0.3E-05	0.1E+00	124	0.057	2.520
V_1	V_3	36	125	-0.017	39	0.3E-05	0.4E+00	103	0.049	3.625
V_1	V_4	32	122	-0.339	30	0.8E-08	0.1E-02	221	0.050	3.347
V_1	V_5	30	55	-1.036	33	0.2E-05	0.4E+00	123	0.050	2.556
V_1	V_6	14	8	2.000				0	2.000	0.052
V_2	V_0	44	136	0.269				51	0.050	1.628
V_2	V_1	32	94	-0.461	32	0.5E-07	0.3E-02	59	0.050	1.760
V_2	V_3	29	21	0.240				12	0.158	0.225
V_2	V_4	47	200	-0.398	30	0.5E-06	0.1E-01	54	0.050	3.611
V_2	V_5	16	9	0.129				0	0.129	0.088
V_2	V_6	33	87	-0.543	37	0.9E-06	0.6E-02	193	0.050	2.987
V_3	V_0	20	10	1.532				0	1.532	0.066
V_3	V_1	38	164	-0.654	30	0.1E-05	0.2E-02	143	0.050	3.467
V_3	V_2	29	21	0.240				12	0.158	0.281
V_3	V_4	29	80	0.021	5	0.2E-10	0.2E-11		0.050	0.417
V_3	V_5	13	4	0.254				0	0.254	0.049
V_3	V_6	44	337	-0.515	32	0.7E-07	0.1E+00	163	0.049	5.807
V_4	V_0	31	174	0.042	33	0.1E-05	0.1E+00	140	0.050	2.577
V_4	V_1	34	204	-0.418	30	0.5E-06	0.2E-01	121	0.050	3.476
V_4	V_2	44	184	-0.092	30	0.5E-07	0.1E-01	127	0.050	3.885
V_4	V_3	29	136	0.080				18	0.102	0.518
V_4	V_5	29	25	-0.835	18	0.3E-10	0.3E-11		0.050	0.670
V_4	V_6	38	197	-0.537	46	0.1E-05	0.2E-02	113	0.050	4.659
V_5	V_0	27	7	-0.014	7	0.1E-11	0.9E-13		0.050	0.324
V_5	V_1	30	49	-1.076	31	0.4E-14	0.4E-12		0.050	1.068
V_5	V_2	16	9	0.108				0	0.108	0.104
V_5	V_3	13	4	0.254				0	0.254	0.046
V_5	V_4	30	58	-0.141	25	0.4E-14	0.5E-13		0.050	0.934
V_5	V_6	30	149	-1.088	17	0.2E-11	0.1E-12		0.050	0.882
V_6	V_0	43	257	-0.481	30	0.4E-05	0.7E-01	421	0.050	9.052
V_6	V_1	14	8	2.000				0	2.000	0.048
V_6	V_2	32	112	-0.541	43	0.9E-05	0.1E+00	177	0.050	3.098
V_6	V_3	42	153	-0.193	30	0.3E-06	0.3E-02	47	0.049	3.136
V_6	V_4	38	228	-0.558	41	0.5E-07	0.2E-01	119	0.050	4.005
V_6	V_5	30	151	-0.797	30	0.7E-05	0.3E-01	180	0.081	2.733

Table 1: Numerical results for the trajectories between the points V_0 to V_6. N is the number of time steps in the discretization. It$_I$, It$_c$ and It$_p$ are the number of iterations in the SQP method for (OCP_I), (OCP_c) and (OCP_p) respectively. D is the minimum distance between the robot and the obstacles along the trajectory. CN is the norm of the constraints and KKT is the maximum norm of the gradient of the Lagrange function for (OCP_c). CPU is the computational time (s) including the initial via point computations.

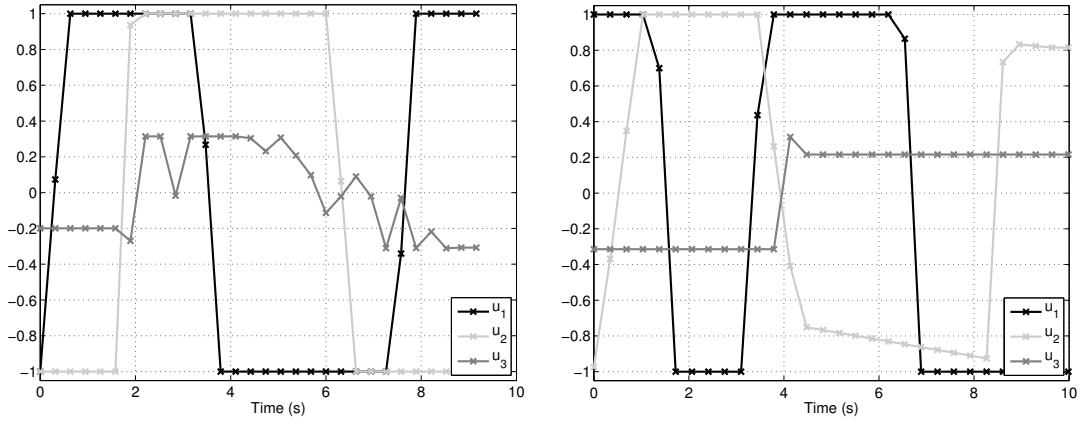
close to 0). About half of the trajectories cannot be solved by (OCP_c). This result is expected since the collision avoidance constraint is not continuously differentiable. In all these cases, the Karush-Kuhn-Tucker conditions are not satisfied due to the gradient of the collision avoidance constraint (KKT larger than 10^{-3}). But, the box constraints, the boundary condition and the collision avoidance constraint are fulfilled (CN small).

When there is no obstacle between P_0 and P_f (as between V_0 and V_3 or between V_1 and V_6), (OCP_I) finds the solution directly. Indeed, the distance D is positive and It_p is equal to 0, meaning that the initial trajectory is optimal. For all the other trajectories, (OCP_p) always succeeds in finding the optimal trajectory.

In Figure 6-(b), the optimal trajectory between V_0 and V_4 and between V_5 and V_6 are illustrated. The robot is the white square, that is shown for several time steps. The small black squares are the via points. The light grey trajectory is the initial trajectory, whereas the final trajectory is dark grey. The markers on the curve indicate the position of the center of gravity of the robot at the time steps $t_k, k = 0, \ldots, N$, of the discretization.

For both examples, the initial trajectory is good, in the sense that the trajectory indicates between which obstacles the optimal trajectory must pass. Furthermore, we can observe that the final trajectory is correct and fully satisfies the dynamics constraints. Such trajectories cannot be found by interpolating the path planning since the optimal trajectory can be removed from the points defining the path in order to satisfy the dynamics constraints.

Finally, the time evolution of the control variable $u = (u_1, u_2, u_3) = (a, \mu)$ for the trajectories in Figure 6-(b) is given in Figure 7. We can observe that the control variable has not necessarily a bang-bang behaviour. This indicates that we should use a direct method to solve the optimal control problems, and not look for the switching points as developed in Bobrow et al. [5].



(a) (b)

Figure 7: Time evolution of the control variable for the trajectory (a) between V_0 and V_4 and (b) between V_5 and V_6.

6 Conclusion

A new approach to solve the kinodynamic motion planning problem was presented. The method involves solving a sequence of optimal control problems. The same direct method was utilized to solve the optimal control problems. The main focus was on the computation of a good initial trajectory. For that purpose, a two-step approach was developed. In the first step, a set of via points was computed with a graph search algorithm. The issuing path had to follow a good balance between the number of turns and the distance to the obstacles. The second step involved finding the fastest trajectory that passes through the neighborhood of the via points. The initial trajectory was obtained by solving an optimal control problem.

This new approach was applied to a two-dimensional mobile robot. The numerical results show the quality of the computed initial trajectory and the low computational time to get the optimal trajectory. These results are promising for an application of the strategy to 3D robot. The optimal control problems (OCP_I), (OCP_c) and (OCP_p) remains valid. Therefore, the solving technique is the same direct method. The first difference is the meaning of the unknowns: in three dimensions, the unknowns are no more the centre of gravity of the robot and the rotation angle, but the joint angles that link the different bodies of the robot [13, 21]. The second difference is the definition of the ordinary differential equations.

The computation of the via points for 3D robot is very similar to the 2D-case. The underlying grid for these 3D instance has one dimension per joint and the arc weights correspond to the time needed to travel from one grid node to another. Moving multiple joints at the same time is allowed so that also diagonal edges are included in the grid. Now all the methods from 2D, such as turn costs and obstacle distance, can be applied. Only the smoothening part gets more complicated from an algorithmic point of view. It can no longer be calculated by line-polygon intersection, since in 3D intersections of moving polyhedra need to be calculated.

Eventually, we outline two possible improvements to our strategy. First, the discontinuity in the derivative of the distance function can be better handled in (OCP_c) by using some bundle methods [30]. Second, a better determination of the time steps $t_{k(i)}$ for which a condition of the form $\|P_i - r(t))\|_2$ is imposed in (OCP_I), can be established. The idea would be to consider such time steps as a free variable in (OCP_I), as it is done in Example 1.2.1 in [12].

References

[1] A. Barclay, P. E. Gill, and J. Ben Rosen. SQP methods and their application to numerical optimal control, 1997.

[2] L.D. Berkovitz. *Convexity and Optimization in* $\mathbb{R}^n$. John Wiley & Sons, New York, 2001.

[3] J. T. Betts. *Practical methods for optimal control using nonlinear programming*, volume 3 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[4] J.E. Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE Journal of Robotics and Automation*, 4:443–450, 1988.

[5] J.E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4:3–17, 1985.

[6] R. Bohlin. *Robot Path Planning*. Chalmers University of Technology, 2002.

[7] S. A. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*, pages 591–596, 1986.

[8] B. Donald, P. Xavier, J. Canny, and J. reif. Kinodynamic motion planning. *Journal of the Association for Computing Machinery.*, 40:1048–1066, 1993.

[9] S. Dubowsky, M. A. Norris, and Z. Shiller. Time optimal trajectory planning for robotic manipulators with obstacle avoidance: a CAD approach. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1906–1912, 1989.

[10] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239–269, 2002.

[11] R. Fletcher, S. Leyffer, and P. Toint. On the global convergence of a filter–SQP algorithm. *SIAM Journal on Optimization*, 13(1):44–59, 2002.

[12] M. Gerdts. *Optimal Control of ODEs and DAEs*. De Gruyter Textbook. De Gruyter, 2012.

[13] M. Gerdts, R. Henrion, D. Hömberg, and C. Landry. Path planning and collision avoidance for robots. *Numerical Algebra, Control and Optimization*, 2(3):437 – 463, 2012.

[14] E.G. Gilbert and D.W. Johnson. Distance functions and their application to robot path planning in the presence of obstacle. *IEEE Journal of Robotics and Automation*, RA-1:21–30, 1985.

[15] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. a fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[16] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[17] G. D. Hart and M. Anitescu. An O(m+n) measure of penetration depth between convex polyhedral bodies for rigid multibody dynamics, 2010.

[18] D.W. Johnson and E.G. Gilbert. Minimum time robot path planning in the presence of obstacles. In *Decision and Control, 1985 24th IEEE Conference on*, volume 24, pages 1748–1753, 1985.

[19] Y. J. Kim, M. C. Lin, and D. Manocha. DEEP: Dual-space expansion for estimating penetration depth between convex polytopes. In *IEEE Conference on Robotics and Automation*, pages 921–926, 2002.

[20] C. Landry, M. Gerdts, R. Henrion, and D. Hömberg. Path-planning with collision avoidance in automotive industry. In *System Modeling and Optimization*, 25th IFIP TC 7 Conference, Berlin, Germany, September 12-16, 2011, Revised Selected Papers IFIP AICT 391, Springer, Heidelberg; Approx. IX, 575 pp, 2013.

[21] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[22] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[23] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.

[24] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, pages 1008–1014, 1991.

[25] A. Maheshwari, J.-R. Sack, and H. N. Djidjev. Link distance problems. *Handbook of Computational Geometry*, pages 519–558, 1999.

[26] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[27] M.J.D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G.A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 144–157. Springer Berlin Heidelberg, 1978.

[28] S. F. P. Saramago and V. Steffen. Trajectory modeling of robot manipulators in the presence of obstacles. *Journal of Optimization Theory and Applications*, 110:17–34, 2001.

[29] K. Schittkowski. On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function 2. *Mathematische Operationsforschung und Statistik. Series Optimization*, 14(2):197–216, 1983.

[30] H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results. *SIAM J. Optim.*, 2(1):121–152, 1992.

[31] C. Sprunk, B. Lau, P. Pfaffz, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 72–77, 2011.

[32] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.