Optim Eng (2016) 17:533–556
DOI 10.1007/s11081-015-9291-0

# Combining discrete and continuous optimization to solve kinodynamic motion planning problems

**Chantal Landry**[1] · **Wolfgang Welz**[2] · **Matthias Gerdts**[3]

Received: 17 December 2013 / Revised: 11 November 2014 / Accepted: 9 October 2015 /
Published online: 4 November 2015

**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 dynamic 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 intermediate points. Then, an optimal control problem of small size is defined to find the fastest trajectory that passes through the vicinity of the intermediate 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.

**Keywords** Trajectory planning · Optimal control problem · Collision avoidance · Graph search algorithm · Initialization · Robotics

**Mathematics Subject Classification** 49J15 · 49M25 · 49N90 · 70E60 · 90C30 · 90C35

✉ Chantal Landry
chantal.landry@zhaw.ch

Wolfgang Welz
welz@math.tu-berlin.de

Matthias Gerdts
matthias.gerdts@unibw.de

[1] Zurich University of Applied Sciences, Technikumstrasse 9, 8401 Winterthur, Switzerland

[2] Technische Universität Berlin, Strasse des 17. Juni 136, 10623 Berlin, Germany

[3] University of the Federal Armed Forces at Munich, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

## 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 (dynamic constraints). This expression was first introduced by Donald et al. (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 (Gilbert and Johnson 1985; Johnson and Gilbert 1985), optimal robot path planning using the minimum-time criterion (Bobrow 1988), trajectory planning or modeling (Dubowsky et al. 1989; Saramago and Steffen 2001).

Gilbert and Johnson were the first to formulate the kinodynamic planning as an optimal control problem (Johnson and Gilbert 1985). In their formulation, the objective function is the travel time, the dynamic is modelled by 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 Johnson and Gilbert (1985) and extended by other authors, such as Bobrow (1988) and Dubowsky et al. (1989). The technique first discretizes the state variable with B-splines and then looks for a control variable that satisfies the dynamic 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 (Bobrow et al. 1985). 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) (Goerzen et al. 2010; LaValle 2006). The resulting path is collision-free, but does not take into consideration the dynamic constraints. One idea was then to build a trajectory that satisfies the dynamic constraints by smoothing the path and finding the control variables, so that the dynamic 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 (LaValle and Kuffner 2011). 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 dynamic 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 dynamic 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 (1985) and other authors need a collision-free trajectory to find a solution. However, there is a lack of research into the computation of such a collision-free trajectory.

In this article, we propose a method that searches for a good initial trajectory. This method combines discrete and continuous optimization methods. A path planning algorithm is first applied to get a list of intermediate points. Then, the initial trajectory is obtained by computing the fastest trajectory that passes through the vicinity of the intermediate 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 Sect. 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 Sect. 5. Finally, conclusions and extensions to three dimensional problems are given in Section 6.

## 2 Problem formulation

Let us consider a two dimensional robot whose geometry is modelled by a convex compact polyhedron. Extensions to more complicated robots are given at the end of Sects. 2.1 and 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 $\mathbf{P}_0 \in \mathbb{R}^2$ be the initial position and $\mathbf{P}_f \in \mathbb{R}^2$ the goal position. To describe the motion of a robot between $\mathbf{P}_0$ and $\mathbf{P}_f$, we need to define several variables. Let $\mathbf{r}$, $\mathbf{v}$ and $\mathbf{a}$, 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 $\theta$ denote the angle of rotation of the robot and $\mu$ be the velocity of the angle of rotation. Finally, let $t_f$ be the travel time between $\mathbf{P}_0$ and $\mathbf{P}_f$. The motion of the robot on the time interval $[0, t_f]$ is then given by the following dynamic laws:

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

In this paper, a trajectory from $\mathbf{P}_0$ to $\mathbf{P}_f$ is defined by the pair $(t, \mathbf{r}(t))_{t\in[0,t_f]}$ that satisfies $\mathbf{r}(0) = \mathbf{P}_0$ and $\mathbf{r}(t_f) = \mathbf{P}_f$. 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

$$\mathbf{r}(0) = \mathbf{P}_0, \quad \mathbf{v}(0) = \mathbf{0}, \quad \theta(0) = 0, \quad \mathbf{r}(t_f) = \mathbf{P}_f, \quad \mathbf{v}(t_f) = 0, \quad \theta(t_f) = 0. \tag{2}$$

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 at time $t = 0$ 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 $\mathbf{r}(t)$ (LaValle 2006). Let $R(t)$ be the matrix of the vertices at time $t$. Then, we have:

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

where $\mathbf{e}_n$ is the vector that contains $n$ components, which are all equal to 1. Hence, the product $\mathbf{r}(t) \cdot \mathbf{e}_n^\top$ 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

$$R = \{\mathbf{y} \in \mathbb{R}^2 \,|\, A\mathbf{y} \leq \mathbf{b}\} \quad \text{and} \quad H_k = \{\mathbf{y} \in \mathbb{R}^2 \,|\, C_k\mathbf{y} \leq \mathbf{p}_k\},$$

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

Then, Farkas's lemma (Berkovitz 2001) implies that $R$ and $H_k$ do not intersect if and only if there exists a vector $\mathbf{w}_k \in \mathbb{R}^{n+n_k}$ such that

$$\mathbf{w}_k \geq \mathbf{0}, \quad \begin{pmatrix} A \\ C_k \end{pmatrix}^\top \mathbf{w}_k = \mathbf{0} \quad \text{and} \quad \begin{pmatrix} \mathbf{b} \\ \mathbf{p}_k \end{pmatrix}^\top \mathbf{w}_k < \mathbf{0}. \tag{4}$$

Hence, the polyhedra $R$ and $H_k$ do not collide as long as such a vector $\mathbf{w}_k$ exists. Please note that the matrix $A$ and the vector $b$ depend on the state of the moving robot and thus these constraints are in fact nonlinear and time dependent. 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 $\mathbf{w}_k$ must be found for each obstacle. Moreover, the non-uniqueness of $\mathbf{w}_k$ may produce numerical difficulties. Please see Gerdts et al. (2012), Landry et al. (2013) 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 Cameron and Culley (1986), Hart and Anitescu (2010), Kim et al. (2002):

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

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

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

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

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 Bobrow (1988), Gilbert and Johnson (1985), we prefer the second approach with its small size, taking into account the nonsmoothness issues. Therefore, combining (1), (2) with (6), we obtain the kinodynamic motion planning problem between $\mathbf{P}_0$ and $\mathbf{P}_f$

$$(P)\colon \min\ t_f \quad \text{subject to the constraints:}$$

- equations of motion: $\mathbf{r}'(t) = \mathbf{v}(t)$, a.e. in $[0,t_f]$,
  $\mathbf{v}'(t) = \mathbf{a}(t)$, 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: $\mathbf{r}(0) = \mathbf{P}_0, \mathbf{v}(0) = \mathbf{0}, \theta(0) = 0,$
  $\mathbf{r}(t_f) = \mathbf{P}_f, \mathbf{v}(t_f) = \mathbf{0}, \theta(t_f) = 0,$
- box constraints: $\underline{\mathbf{a}} \leq \mathbf{a}(t) \leq \overline{\mathbf{a}}$ and $\underline{\mu} \leq \mu(t) \leq \overline{\mu}$, a.e. in $[0,t_f]$,

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

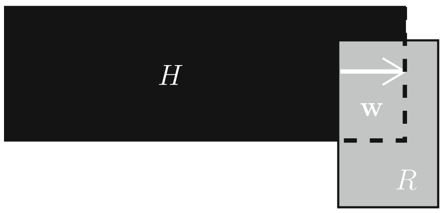

**Fig. 1** The polyhedra $H$ and $R$ overlap. The vector $\mathbf{w}$ is the smallest vector such that $R + \mathbf{w}$ and $H$ come into contact, where $R + \mathbf{w}$ is the polyhedron $R$ translated by $\mathbf{w}$

The problem ($P$) is an optimal control problem where the state variables are the position, $\mathbf{r}$, the velocity, $\mathbf{v}$, and the angle of rotation, $\theta$. The control variables are the acceleration of the center of gravity, $\mathbf{a}$, and the velocity of the rotation angle $\mu$. Let us store the variables in the following vectors:

– the state variable: $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} \mathbf{r} \\ \mathbf{v} \\ \theta \end{pmatrix} \in \mathbb{R}^{n_x}$,

– the control variable: $\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} \mathbf{a} \\ \mu \end{pmatrix} \in \mathbb{R}^{n_u}$,

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: $\varphi(\mathbf{x}(0), \mathbf{x}(t_f), t_f)$. 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 $\mathbf{r}(t)$ and $\theta(t)$ (compare (3)), the distance function is rewritten in the form $d(\mathbf{x}(t), H_k)$. Let us now define the following functions:

– $g\colon \mathbb{R}^{n_x} \to \mathbb{R}$ s.t. $g(\mathbf{x}) = \min_{k=1,\ldots,K} d(\mathbf{x}, H_k)$,

– $\mathbf{f}\colon \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ s.t. $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} x_3 \\ x_4 \\ \mathbf{u} \end{pmatrix}$,

– $\mathbf{c}\colon \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{2n_x}$ s.t. $\mathbf{c}(\mathbf{x}(0), \mathbf{x}(t_f)) = \begin{pmatrix} \mathbf{r}(0) - \mathbf{P}_0 \\ \mathbf{v}(0) \\ \theta(0) \\ \mathbf{r}(t_f) - \mathbf{P}_f \\ \mathbf{v}(t_f) \\ \theta(t_f) \end{pmatrix}$.

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

$$
\begin{aligned}
&(OCP_c) \qquad\qquad \min \varphi(\mathbf{x}(0), \mathbf{x}(1), t_f)\\
&\text{with respect to } \mathbf{x} \in W_{1,\infty}^{n_x}(T), \mathbf{u} \in L_{\infty}^{n_u}(T)\\
&\text{s.t.} \quad \mathbf{x}'(\mathrm{t}) - \mathrm{t_f}\,\mathbf{f}(\mathbf{x}(\mathrm{t}), \mathbf{u}(\mathrm{t})) = \mathbf{0}, \quad \text{a.e. in T}, \qquad (7)
\end{aligned}
$$

$$g(\mathbf{x}(t)) \geq \varepsilon, \text{ a.e. in T}, \qquad (8)$$

$$\mathbf{c}(\mathbf{x}(0), \mathbf{x}(1)) = \mathbf{0}, \qquad (9)$$

$$\mathbf{u}(t) \in \mathcal{U}, \quad \text{a.e. in T}, \qquad (10)$$

where $\mathcal{U} := \{\mathbf{u}^\top = (\mathbf{a}^\top, \mu) \in \mathbb{R}^{n_u} \,|\, \underline{\mathbf{a}} \leq \mathbf{a} \leq \bar{\mathbf{a}}, \underline{\mu} \leq \mu \leq \bar{\mu}\}$.

As usual $L_{\infty}^{n_u}(T)$ denotes the Banach space of essentially bounded functions mapping from $T$ into $\mathbb{R}^{n_u}$ and $W_{1,\infty}^{n_x}(T)$ 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 dynamic 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 $\mathbf{f}$ 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 $\mathbf{x}$ and then look for a control that satisfies the dynamic constraints and minimizes the cost functional (Bobrow 1988; Johnson and Gilbert 1985). 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 (Gerdts et al. 2012). But, in contrary to Bobrow (1988), Johnson and Gilbert (1985), 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:

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

As in Lin et al. (2014) the control variable is approximated with B-splines as follows

$$\mathbf{u}_h(t;\mathbf{u}_0,\ldots,\mathbf{u}_{N+r-2}) := \sum_{i=0}^{N+r-2} B_{ir}(t)\,\mathbf{u}_i,$$

where $B_{ir}$, $i=0,\ldots,N+r-2$, are elementary B-splines of order $r$ on $\mathbb{G}_N$ and $(\mathbf{u}_0,\ldots,\mathbf{u}_{N+r-2})^\top \in \mathbb{R}^{n_u(N+r-1)}$ is the vector of de Boor points. The choice of B-splines is convenient since the elementary functions have a local support only. For further details on B-Splines, see Quarteroni et al. (2007).

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

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

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 $\mathbf{x}_0 = (\mathbf{P}_0,0,0,0) \in \mathbb{R}^{n_x}$, the travel time $t_f$ and the de Boor points $\mathbf{u}_i$, $i=0,\ldots,N+r-2$. Let us store $t_f$ and the de Boor points in the following vector

$$\mathbf{z}^\top := (\mathbf{u}_0^\top,\ldots,\mathbf{u}_{N+r-2}^\top,t_f) \in \mathbb{R}^{n_z},$$

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

$$\begin{aligned}&\mathbf{x}_{k+1}(\mathbf{z}) = \mathbf{x}_k(\mathbf{z}) + h\Phi(t_k,\mathbf{x}_k(\mathbf{z}),\mathbf{u}_h(t_k;\mathbf{u}_0,\ldots,\mathbf{u}_{N+r-2}),t_f,h) \quad \text{with}\,\mathbf{x}_0(\mathbf{z}) = \mathbf{x}_0\\ &\quad \text{and} \quad \text{for k} = 0,\ldots,\text{N}-1,\end{aligned} \tag{11}$$

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:

$$\begin{aligned}(NLP) \qquad & \min_{\mathbf{z}} J(\mathbf{z})\\ \text{s.t.} \qquad & \mathbf{c}(\mathbf{x}_0,\mathbf{x}_\text{N}(\mathbf{z})) = \mathbf{0},\\ & g(\mathbf{x}_k(\mathbf{z})) \geq \varepsilon,\\ & \mathbf{z} \in \mathcal{Z},\end{aligned}$$

where $J(\mathbf{z}) := \varphi(\mathbf{x}_0,\mathbf{x}_N(\mathbf{z}),t_f)$, $\mathcal{Z} := \{\mathbf{z} \in \mathbb{R}^{n_z}\,|\,\underline{\mathbf{z}} \leq \mathbf{z} \leq \overline{\mathbf{z}}\}$ with $\underline{\mathbf{z}}^\top = (\underline{\mathbf{a}}^\top,\underline{\mu},\ldots,\underline{\mathbf{a}}^\top,\underline{\mu},\underline{t})$, $\overline{\mathbf{z}}^\top = (\overline{\mathbf{a}}^\top,\overline{\mu},\ldots,\overline{\mathbf{a}}^\top,\overline{\mu},\overline{t})$ and $\underline{t},\overline{t} \geq 0$ are given lower and upper bound values.

A similar approach is described in Lin et al. (2014). The problem (*NLP*) is a finite dimensional nonsmooth and non-convex optimization problem in $\mathbf{z}$. Despite the nonsmoothness of the problem we apply a sequential quadratic programming (SQP) method with BFGS updates to (*NLP*). SQP methods are iterative methods (Barclay et al. 1997; Diehl et al. 2010; Nocedal and Wright 2006). At each iteration, a quadratic programming (QP) sub-problem is being solved to find a search direction. The objective function of the (QP) sub-problems is a local quadratic approximation of the

Lagrange function. Since the Hessian matrix of the Lagrange function is not well defined at the points of non-differentiability of $g$, we use BFGS update formulas (Betts 2001; Nocedal and Wright 2006) to replace it. The nonsmooth constraint with $g$ is being linearized by using an approximate subgradient of $g$, which we obtain numerically by finite differences. More elaborate approaches to approximate a subgradient of a convex function can be found in Shor (1985). This approach is heuristic and up to now there is no mathematical proof which justifies that the BFGS SQP method will converge to stationary points in the nonsmooth case; there are even counterexamples. However, it was observed by many authors that the use of BFGS updates often works well even in the nonsmooth case. This observation was investigated in more detail in Lewis and Overton (2008, 2013). Our numerical experiments support this observation and thus we decided to apply the BFGS SQP method instead of smoothing the distance function as in Escande et al. (2014) or by using bundle type algorithms, which are specifically designed for nonsmooth problems, see, e.g., Schramm and Zowe (1992). Our numerical experience is just the opposite to what one would expect from theory. The smooth formulation with the constraints in (4) leads to a smooth but very large nonlinear nonconvex problem with many additional (artificial) controls $w$. This large dimensional problem turns out to be much harder to solve (although it is smooth) than the nonsmooth approach with the signed distance function in (NLP). A reason for that might be, that only few constraints hidden in the function $g(\mathbf{x})$ actually become active when passing an obstacle while most of the constraints are not active. This can be seen as an implicit reduction of the number of constraints and seems to lead to mild nonlinearity. Whereas in (4) there are many constraints that have to be obeyed and the corresponding controls are in general not unique. So, the degree of nonlinearity is much higher.

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

In order to achieve convergence from remote points, the SQP algorithm can be augmented by a globalization strategy. See Powell (1978), Schittkowski (1983) for line-search based methods and Fletcher and Leyffer (2002), Fletcher et al. (2002) for filter methods. However, without a good initialization of the starting point $\mathbf{z}^{(0)}$, even the globalized SQP method might not converge. The main focus of this paper is on the specific construction of a sufficiently good initialization of $\mathbf{z}^{(0)}$ for the robot motion planning problem. 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 (Johnson and Gilbert 1985; Sprunk et al. 2011), but there is a lack of research into the computation of such a trajectory. This trajectory must be close to

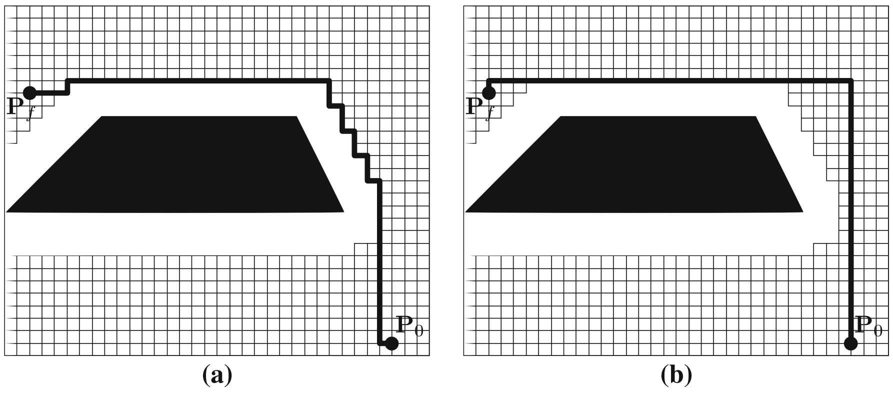

**Fig. 2** **a** One possible path from $\mathbf{P}_0$ to $\mathbf{P}_f$ that uses the least amount of edges in the grid. **b** This path uses the same amount of edges, i.e. has the same length, as the path in (**a**), but contains only two turns

the optimal trajectory and very often collision-free. Let us consider the example depicted in Fig. 2a 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 $\mathbf{P}_f$. If $\mathbf{z}^{(0)}$ were initialized such that the associated trajectory were the segment $[\mathbf{P}_0, \mathbf{P}_f]$, 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 $\mathbf{P}_f$ 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 $\mathbf{z}^{(0)}$ such that the associated initial trajectory $(t_k, \mathbf{r}_k(\mathbf{z}^{(0)}))_{k=0,\ldots,N}$ has a favorable shape. First, a path-planning algorithm is used to find intermediate points. These points indicate the robot the direction to reach $\mathbf{P}_f$. In the second step, $\mathbf{z}^{(0)}$ is computed, so that the initial trajectory passes through the neighborhood of the intermediate points.

## 3.1 Computation of the intermediate points

Path-planning involves searching for 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 Goerzen et al. (2010), LaValle (2006) for an exhaustive review. Here, we do not need to use sophisticated methods. We look for a path that is collision-free, but not necessarily the shortest. In addition, the path must have a small number of turns. If many changes of direction occur (Fig. 2a 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 $\rho$ 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 $\rho$. Hereby, it is crucial that the discretization is chosen in such a way that the grid is fine enough so that every connection in it represents a feasible collision-free motion of the robot.

For simplicity, we only allow horizontal and vertical movements and the distance between two grid points is set to the constant $\delta$. Further, let $s$ and $\tau$ be the closest grid points to $\mathbf{P}_0$ and $\mathbf{P}_f$ respectively. Since many turns may pose a problem, one solution is to calculate the shortest $s$-$\tau$-path with the least amount of turns instead. This is a concept which is a common approach in many path planning problems (Bohlin 2002; Maheshwari et al. 1999). This can be modeled by using a large enough turn cost $M$ to penalize paths containing turns.

Unfortunately, even if $M$ is chosen larger than the number of nodes in the grid, it is not sufficient to modify Dijkstra's algorithm to just add the corresponding turn cost in the extension step: Dijkstra's algorithm only keeps the shortest distance corresponding to one subpath per node. In the turn cost setting, however, there might be different subpaths and each leading to different turn costs depending on the next arc. Therefore, the optimality would no longer be guaranteed.

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 $G'$. In such a graph the edges of the original primal graph correspond to the nodes in $G'$. The arcs connecting two nodes in $G'$ now correspond to the turns in the original graph. We can now give the respective turn-costs for these arcs (Winter et al. 2002). Figure 2b shows the result of such a turn-minimal computation.

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 a 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 found using the presented graph modifications and Dijkstra's algorithm. 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.

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 node $v_i$ on this path, 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 $\rho$ and therefore can be used by the robot without any collisions. When we reached a node $v_i$ 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 $\tau$. See Fig. 3b for an example of such a smoothened path. The inner nodes $v_i$ in the smoothened path are the intermediate points that will be used in the next sub-section.

Applying this smoothening process we no 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.

The idea presented in this section represents an heuristic approach that leads to good initial trajectories. All of its aspects, such as turn cost, obstacle clearance and smoothening, are very reasonable as they represent properties of a good initial solution.

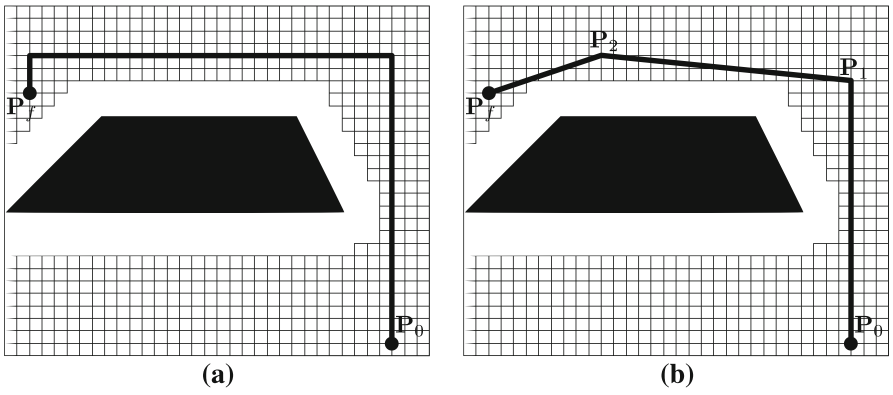

**Fig. 3** **a** Path that avoids close obstacles without taking to many turns. However, his path is slightly longer than the length of the shortest path in Fig. 2. **b** Smoothened version of the path in **a**

## 3.2 Computation of the initial trajectory

In Sect. 3.1, the intermediate points were computed as the nodes of a smoothened path between $\mathbf{P}_0$ and $\mathbf{P}_f$. Let $n_I$ be the number of intermediate points and $\mathbf{P}_i$ be the $i$th intermediate point.

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

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

$$\mathbf{z}^{(0),\top} := (\mathbf{u}_0^{0,\top}, \ldots, \mathbf{u}_{N+r-2}^{0,\top}, t_f^0).$$

The de Boor points $\mathbf{u}_i^0$, $i = 1, \ldots, N + r - 2$ 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 intermediate points is added in the objective function. The penalty term is of the form $\|\mathbf{P}_i - \mathbf{r}(t)\|_2 < \varepsilon_r$, where $\varepsilon_r$ 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 $\mathbf{P}_i$ and of radius $\varepsilon_r$. 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 intermediate 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 Sect. 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 $\mathbf{P}_0$, passing through $\mathbf{P}_i$, $i = 1, \ldots, n_I$, and ending at $\mathbf{P}_F$, as illustrated in Fig. 3a. By definition, we have:

$$L := \sum_{i=0}^{n_I} L_i = \sum_{i=0}^{n_I} \|\overrightarrow{\mathbf{P}_i\mathbf{P}_{i+1}}\|_2, \quad \text{where } \mathbf{P}_{\mathrm{n_I}+1} = \mathbf{P}_{\mathrm{F}}.$$

Then, the number of grid points is defined as

$$N = \begin{cases} \left\lfloor \dfrac{L}{dl} \right\rfloor, & \text{if } \dfrac{\mathrm{L}}{\mathrm{dl}} - \left\lfloor \dfrac{\mathrm{L}}{\mathrm{dl}} \right\rfloor \leq 0.5, \\ \left\lfloor \dfrac{L}{dl} \right\rfloor + 1, & \text{otherwise,} \end{cases}$$

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

The initial trajectory $(t_k, \mathbf{r}_k(z^0))_{k=0,\ldots,N}$ is such that the trajectory passes through a neighborhood of the intermediate points $\mathbf{P}_i$, $i = 1, \ldots, n_I$, at certain time steps. These time steps are chosen according to the distance travelled by the robot from the initial position to the intermediate point. For each point $\mathbf{P}_i$, 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 intermediate 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 $\mathbf{P}_{n_I}$ and $\mathbf{P}_F$. 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 $\mathbf{P}_i$ 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 intermediate points are specified as a penalty term in the objective function. Therefore, the problem reads

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

where

$$p(t) := \begin{cases} \max(\|\mathbf{P}_i - \mathbf{r}(t)\|_2 - \varepsilon_r, 0)^2, & \text{if } \exists \mathrm{i} \in \{1, \ldots, \mathrm{n_I}\} \text{s.t.} |\mathrm{t} - \mathrm{t_{k(i)}}| \le \dfrac{\mathrm{h}}{2}; \\ 0, & \text{otherwise;} \end{cases}$$

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 $\mathbf{P}_i$ and of radius $\varepsilon_r$. 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 $\mathbf{u}^*_{I,i}$, $i = 0, \ldots, N + r - 2$, 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 $\mathbf{z}^{(0)}$ is initialized as follows:

$$t_f^0 = t^*_{f,I} \quad \text{and} \quad \mathbf{u}^0_{\mathrm{i}} = \mathbf{u}^*_{\mathrm{I,i}},\ \mathrm{i} = 0, \ldots, \mathrm{N} + \mathrm{r} - 2.$$

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

Numerical examples are given in Figure 4. The workspace is a rectangle that contains four obstacles (black quadrilateral). For each example, $\mathbf{P}_0$, $\mathbf{P}_f$ and the intermediate 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 $t_k$, $k = 0, \ldots, N$.

We can observe that all initial trajectories are good candidates since they pass between the right obstacles to reach $\mathbf{P}_f$. 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 Fig. 4b 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 intermediate points varies between 1 and 3. If there is no obstacle between $\mathbf{P}_0$ and $\mathbf{P}_f$, no intermediate point is defined and the initial trajectory is a straight line. Since no intermediate 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 intermediate points suffices to compute the initial trajectory. If this number were large, or if the initial path would have unnecessary turns, then the number 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 intermediate points and satisfies the dynamic 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 $\mathbf{z}^{(0)}$ in the local SQP algorithm. The initialization is such that

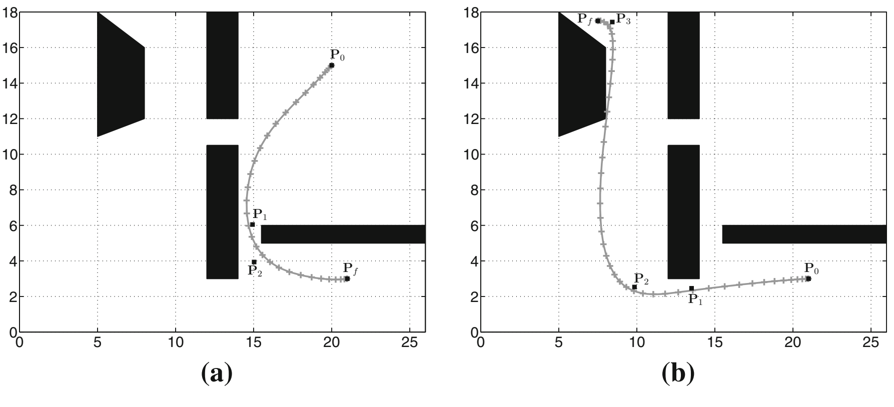

**Fig. 4** Examples of initial trajectories and their associated intermediate points which come from the path planing algorithm

$\mathbf{z}^{(0)}$ satisfies the lower and upper bounds on $\mathbf{z}$, i.e. $\mathbf{z}^{(0)} \in \mathcal{Z}$. Furthermore, the associated initial trajectory $(t_k, \mathbf{r}(\mathbf{z}^{(0)}))_{k=0}^{N}$ 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:

$$\lambda^{(0)} = \lambda_I^* \quad \text{and} \quad \zeta^{(0)} = \zeta_{\mathrm{I}}^*,$$

where $\lambda_I^*$ and $\zeta_I^*$ are the multipliers issued from $(OCP_I)$.

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

$$g(\mathbf{x}_k(\mathbf{z})) \geq \varepsilon, \quad k = 0, \ldots, N.$$

These inequalities describe the collision avoidance between the robot and the obstacles. The complementarity condition in the Karush-Kuhn-Tucker conditions implies that $\sigma_k$ is equal to 0 if the inequality constraint is inactive (Nocedal and Wright 2006). This means that $\sigma_k$ 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 $\sigma$ to 0.

With this initialization, $(OCP_c)$ can be solved as described in Sect. 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 $(QP_\ell)$ 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

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

where $\alpha$ and $\beta$ 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.

The initial value of $\mathbf{z}^{(0)}$, $\lambda^{(0)}$, $\zeta^{(0)}$ and $\sigma^{(0)}$ are the last value of $\mathbf{z}^{(\ell)}$, $\lambda^{(\ell)}$, $\zeta^{(\ell)}$ 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 $\varepsilon$.

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 $g(\mathbf{x}_k(\mathbf{z}^{(0)})) \geq \varepsilon$, $k = 0, \ldots, N$, then $(OCP_p)$ is solved directly. A summary of this strategy is presented in Fig. 5. First, the intermediate 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 $(OCP_p)$ numerically, we use the package OCPID-DAE1 (see Gerdts (2013) and http://www.optimal-control.de) developed by M. Gerdts. The method introduced in Sect. 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 Gilbert et al. (1988) 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

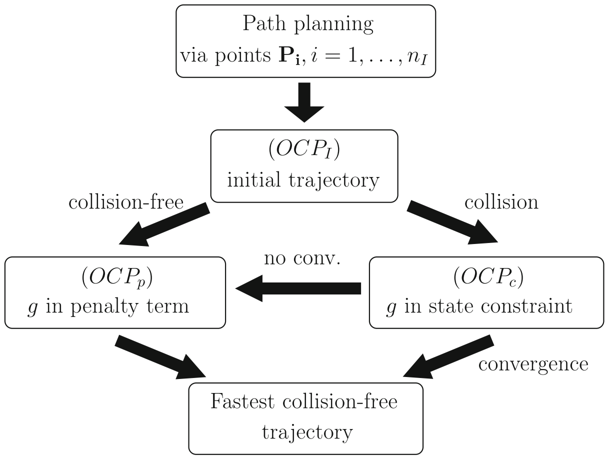

**Fig. 5** Scheme of the strategy to find the fastest collision-free trajectory of a robot between two given positions

Hausdorff distance is equal to the norm of the vector $\mathbf{w}$ that characterizes the penetration depth (see (5)).

The second approach is Lin and Canny's algorithm (Lin 1993; Lin and Canny 1991). This algorithm determines the closest pair of features between the polyhedra, where the features of a polyhedron are its vertices, its edges and its 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 $\varepsilon$, 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 $\bar{\mathbf{a}}^\top = (1.0, 1.0)$ and $\underline{\mathbf{a}} = -\bar{\mathbf{a}}$. The bounds on the velocity of the rotation angle are $\bar{\mu} = \frac{\pi}{10}$ and $\underline{\mu} = -\bar{\mu}$.

The points $\mathbf{V}_i$, $i = 0, \ldots, 6$, in the workspace define possible starting and end positions for the robot. We compute the fastest collision-free trajectory for any pair $(\mathbf{P}_0, \mathbf{P}_f)$ with $\mathbf{P}_0 \in \{\mathbf{V}_0, \ldots, \mathbf{V}_6\}$ and $\mathbf{P}_f \in \{\mathbf{V}_0, \ldots, \mathbf{V}_6\} \setminus \{\mathbf{P}_0\}$. For the numerical computations, we set the safety margin $\varepsilon$ 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

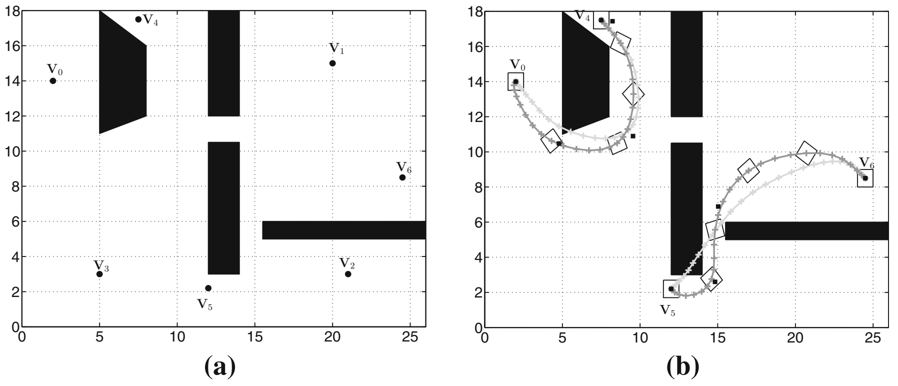

**Fig. 6** **a** The workspace with four obstacles (*quadrilaterals*) and seven points $\mathbf{V}_0$ to $\mathbf{V}_6$ between which the optimal trajectory is computed. **b** The optimal trajectory between $\mathbf{V}_0$ and $\mathbf{V}_4$ and between $\mathbf{V}_5$ and $\mathbf{V}_6$. The *black squares* are the intermediate 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*

between the intermediate points and the line passing through $\mathbf{P}_0$ and $\mathbf{P}_f$. Finally, $\alpha$ and $\beta$ 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 $\mathrm{It}_I$, $\mathrm{It}_c$ and $\mathrm{It}_p$ 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:

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

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 Karush-Kuhn-Tucker conditions (KKT). More precisely, CN and KKT are defined by

$$\begin{aligned} \mathrm{CN} = \max\Bigg( & \max_{1 \le i \le n_z} (0, z_i^c - \bar{z}_i), \max_{1 \le i \le n_z} (0, \underline{z}_i - z_i^c), \\ & \max_{0 \le k \le N} (0, \varepsilon - g(\mathbf{x}_k(\mathbf{z}^c))), \|c(\mathbf{x}_0, \mathbf{x}_N(\mathbf{z}^c))\|_\infty \Bigg), \\ \mathrm{KKT} = {} & \|\nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}^c, \lambda^c, \sigma^c, \zeta^c)\|_\infty, \end{aligned}$$

where $\mathbf{z}^c$, $\lambda^c$, $\sigma^c$ and $\zeta^c$ are the outputs of $(OCP_c)$ and $\mathcal{L}$ is the Lagrange function of $(OCP_c)$. If $(OCP_c)$ converges, then $\mathbf{z}^c$ is the optimal solution and $(t_k, \mathbf{r}_k(\mathbf{z}^c))_{k=0,\ldots,N}$ 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 $\varepsilon$, 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 $\varepsilon$, then $(OCP_p)$ is solved. In that case, the components for $\mathrm{It}_c$, CN and KKT in Table 1 are left blank. Similarly, a blank in $\mathrm{It}_p$ means that we do not need to call $(OCP_p)$ for solving the kinodynamic problem.

All the possible trajectories between the points $\mathbf{V}_0$ to $\mathbf{V}_6$ 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 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 $\mathbf{P}_0$ and $\mathbf{P}_f$ (as between $\mathbf{V}_0$ and $\mathbf{V}_3$ or between $\mathbf{V}_1$ and $\mathbf{V}_6$), $(OCP_I)$ finds the solution directly. Indeed, the distance $D$ is positive

**Table 1** Numerical results for the trajectories between the points $\mathbf{V}_0$ to $\mathbf{V}_6$. $N$ is the number of time steps in the discretization

| $\mathbf{P}_0$ | $\mathbf{P}_f$ | N | It$_I$ | D | It$_c$ | CN | KKT | It$_p$ | D | CPU |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{V}_0$ | $\mathbf{V}_1$ | 38 | 182 | −0.885 | 34 | 0.2E−05 | 0.5E−02 | 205 | 0.050 | 4.027 |
| $\mathbf{V}_0$ | $\mathbf{V}_2$ | 43 | 83 | 0.250 | | | | 163 | 0.050 | 2.639 |
| $\mathbf{V}_0$ | $\mathbf{V}_3$ | 20 | 10 | 1.435 | | | | 0 | 1.435 | 0.066 |
| $\mathbf{V}_0$ | $\mathbf{V}_4$ | 30 | 219 | −0.597 | 31 | 0.1E−06 | 0.2E+00 | 159 | 0.050 | 2.696 |
| $\mathbf{V}_0$ | $\mathbf{V}_5$ | 28 | 22 | −0.258 | 13 | 0.5E−14 | 0.2E−12 | | 0.050 | 0.468 |
| $\mathbf{V}_0$ | $\mathbf{V}_6$ | 43 | 75 | −0.763 | 38 | 0.1E−05 | 0.6E−02 | 323 | 0.050 | 6.932 |
| $\mathbf{V}_1$ | $\mathbf{V}_0$ | 37 | 173 | −0.557 | 53 | 0.8E−06 | 0.5E−01 | 277 | 0.049 | 6.171 |
| $\mathbf{V}_1$ | $\mathbf{V}_2$ | 33 | 91 | −0.175 | 31 | 0.3E−05 | 0.1E+00 | 124 | 0.057 | 2.520 |
| $\mathbf{V}_1$ | $\mathbf{V}_3$ | 36 | 125 | −0.017 | 39 | 0.3E−05 | 0.4E+00 | 103 | 0.049 | 3.625 |
| $\mathbf{V}_1$ | $\mathbf{V}_4$ | 32 | 122 | −0.339 | 30 | 0.8E−08 | 0.1E−02 | 221 | 0.050 | 3.347 |
| $\mathbf{V}_1$ | $\mathbf{V}_5$ | 30 | 55 | −1.036 | 33 | 0.2E−05 | 0.4E+00 | 123 | 0.050 | 2.556 |
| $\mathbf{V}_1$ | $\mathbf{V}_6$ | 14 | 8 | 2.000 | | | | 0 | 2.000 | 0.052 |
| $\mathbf{V}_2$ | $\mathbf{V}_0$ | 44 | 136 | 0.269 | | | | 51 | 0.050 | 1.628 |
| $\mathbf{V}_2$ | $\mathbf{V}_1$ | 32 | 94 | −0.461 | 32 | 0.5E−07 | 0.3E−02 | 59 | 0.050 | 1.760 |
| $\mathbf{V}_2$ | $\mathbf{V}_3$ | 29 | 21 | 0.240 | | | | 12 | 0.158 | 0.225 |
| $\mathbf{V}_2$ | $\mathbf{V}_4$ | 47 | 200 | −0.398 | 30 | 0.5E−06 | 0.1E−01 | 54 | 0.050 | 3.611 |
| $\mathbf{V}_2$ | $\mathbf{V}_5$ | 16 | 9 | 0.129 | | | | 0 | 0.129 | 0.088 |
| $\mathbf{V}_2$ | $\mathbf{V}_6$ | 33 | 87 | −0.543 | 37 | 0.9E−06 | 0.6E−02 | 193 | 0.050 | 2.987 |
| $\mathbf{V}_3$ | $\mathbf{V}_0$ | 20 | 10 | 1.532 | | | | 0 | 1.532 | 0.066 |
| $\mathbf{V}_3$ | $\mathbf{V}_1$ | 38 | 164 | −0.654 | 30 | 0.1E−05 | 0.2E−02 | 143 | 0.050 | 3.467 |
| $\mathbf{V}_3$ | $\mathbf{V}_2$ | 29 | 21 | 0.240 | | | | 12 | 0.158 | 0.281 |
| $\mathbf{V}_3$ | $\mathbf{V}_4$ | 29 | 80 | 0.021 | 5 | 0.2E−10 | 0.2E−11 | | 0.050 | 0.417 |
| $\mathbf{V}_3$ | $\mathbf{V}_5$ | 13 | 4 | 0.254 | | | | 0 | 0.254 | 0.049 |
| $\mathbf{V}_3$ | $\mathbf{V}_6$ | 44 | 337 | −0.515 | 32 | 0.7E−07 | 0.1E+00 | 163 | 0.049 | 5.807 |
| $\mathbf{V}_4$ | $\mathbf{V}_0$ | 31 | 174 | 0.042 | 33 | 0.1E−05 | 0.1E+00 | 140 | 0.050 | 2.577 |
| $\mathbf{V}_4$ | $\mathbf{V}_1$ | 34 | 204 | −0.418 | 30 | 0.5E−06 | 0.2E−01 | 121 | 0.050 | 3.476 |
| $\mathbf{V}_4$ | $\mathbf{V}_2$ | 44 | 184 | −0.092 | 30 | 0.5E−07 | 0.1E−01 | 127 | 0.050 | 3.885 |
| $\mathbf{V}_4$ | $\mathbf{V}_3$ | 29 | 136 | 0.080 | | | | 18 | 0.102 | 0.518 |
| $\mathbf{V}_4$ | $\mathbf{V}_5$ | 29 | 25 | −0.835 | 18 | 0.3E−10 | 0.3E−11 | | 0.050 | 0.670 |
| $\mathbf{V}_4$ | $\mathbf{V}_6$ | 38 | 197 | −0.537 | 46 | 0.1E−05 | 0.2E−02 | 113 | 0.050 | 4.659 |
| $\mathbf{V}_5$ | $\mathbf{V}_0$ | 27 | 7 | −0.014 | 7 | 0.1E−11 | 0.9E−13 | | 0.050 | 0.324 |
| $\mathbf{V}_5$ | $\mathbf{V}_1$ | 30 | 49 | −1.076 | 31 | 0.4E−14 | 0.4E−12 | | 0.050 | 1.068 |
| $\mathbf{V}_5$ | $\mathbf{V}_2$ | 16 | 9 | 0.108 | | | | 0 | 0.108 | 0.104 |
| $\mathbf{V}_5$ | $\mathbf{V}_3$ | 13 | 4 | 0.254 | | | | 0 | 0.254 | 0.046 |
| $\mathbf{V}_5$ | $\mathbf{V}_4$ | 30 | 58 | −0.141 | 25 | 0.4E−14 | 0.5E−13 | | 0.050 | 0.934 |
| $\mathbf{V}_5$ | $\mathbf{V}_6$ | 30 | 149 | −1.088 | 17 | 0.2E−11 | 0.1E−12 | | 0.050 | 0.882 |
| $\mathbf{V}_6$ | $\mathbf{V}_0$ | 43 | 257 | −0.481 | 30 | 0.4E−05 | 0.7E−01 | 421 | 0.050 | 9.052 |
| $\mathbf{V}_6$ | $\mathbf{V}_1$ | 14 | 8 | 2.000 | | | | 0 | 2.000 | 0.048 |
| $\mathbf{V}_6$ | $\mathbf{V}_2$ | 32 | 112 | −0.541 | 43 | 0.9E−05 | 0.1E+00 | 177 | 0.050 | 3.098 |
| $\mathbf{V}_6$ | $\mathbf{V}_3$ | 42 | 153 | −0.193 | 30 | 0.3E−06 | 0.3E−02 | 47 | 0.049 | 3.136 |
| $\mathbf{V}_6$ | $\mathbf{V}_4$ | 38 | 228 | −0.558 | 41 | 0.5E−07 | 0.2E−01 | 119 | 0.050 | 4.005 |

**Table 1** continued

| $\mathbf{P}_0$ | $\mathbf{P}_f$ | N | $\text{It}_I$ | D | $\text{It}_c$ | CN | KKT | $\text{It}_p$ | D | CPU |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{V}_6$ | $\mathbf{V}_5$ | 30 | 151 | −0.797 | 30 | 0.7E−05 | 0.3E−01 | 180 | 0.081 | 2.733 |

$\text{It}_I$, $\text{It}_c$ and $\text{It}_p$ 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 intermediate point computations

and $\text{It}_p$ 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 Fig. 6b, the optimal trajectory between $\mathbf{V}_0$ and $\mathbf{V}_4$ and between $\mathbf{V}_5$ and $\mathbf{V}_6$ are illustrated. The robot is the white square, that is shown for several time steps. The small black squares are the intermediate 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 dynamic 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 dynamic constraints.

Finally, the time evolution of the control variable $\mathbf{u}^\top = (u_1, u_2, u_3) = (\mathbf{a}^\top, \mu)$ for the trajectories in Fig. 6b is given in Fig. 5. 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. (1985).

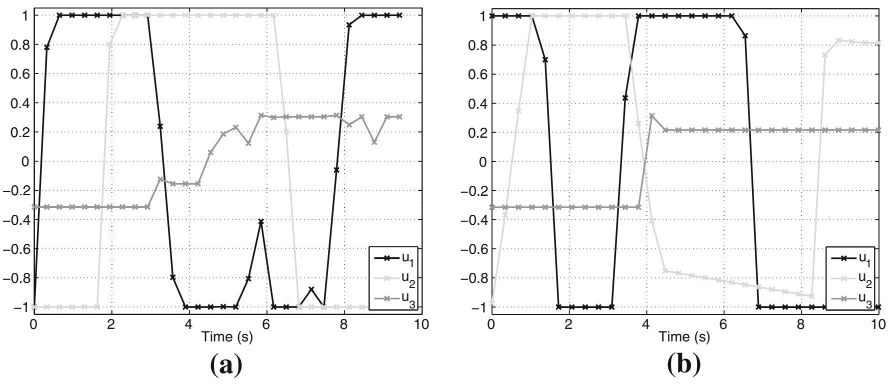

**Fig. 7** Time evolution of the control variable for the trajectory (**a**) between $\mathbf{V}_0$ and $\mathbf{V}_4$ and (**b**) between $\mathbf{V}_5$ and $\mathbf{V}_6$

## 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 intermediate 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 intermediate 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 (Gerdts et al. 2012; LaValle 2006). The second difference is the definition of the ordinary differential equations.

The computation of the intermediate 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 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 (Schramm and Zowe 1992). Second, a better determination of the time steps $t_{k(i)}$ for which a condition of the form $\|\mathbf{P}_i - \mathbf{r}(t))\|_2$ 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 Gerdts (2012) or in the paper Loxton et al. (2009).

## References

Barclay A, Gill PE, Ben Rosen J (1997) SQP methods and their application to numerical optimal control. University of California, San Diego

Berkovitz LD (2001) Convexity and optimization in $\mathbb{R}^n$. Wiley, New York

Betts JT (2001) Practical methods for optimal control using nonlinear programming. Advances in design and control. Society for Industrial and Applied Mathematics (SIAM), Philadelphia

Bobrow JE (1988) Optimal robot path planning using the minimum-time criterion. IEEE J Robot Autom 4:443–450

Bobrow JE, Dubowsky S, Gibson JS (1985) Time-optimal control of robotic manipulators along specified paths. Int J Robot Res 4:3–17

Bohlin R (2002) Robot path planning. Chalmers University of Technology, Goteborg

Cameron SA, Culley RK (1986) Determining the minimum translational distance between two convex polyhedra. In: Proceedings of international conference on robotics and automation, pp 591–596

Diehl M, Walther A, Bock HG, Kostina E (2010) An adjoint-based SQP algorithm with quasi-Newton Jacobian updates for inequality constrained optimization. Optim Methods Softw 25(4–6):531–552

Donald B, Xavier P, Canny J, Reif J (1993) Kinodynamic motion planning. J Assoc Comput Mach 40:1048–1066

Dubowsky S, Norris MA, Shiller Z (1989) Time optimal trajectory planning for robotic manipulators with obstacle avoidance: a CAD approach. In: Proceedings of IEEE international conference on robotics and automation, pp 1906–1912

Escande A, Miossec S, Benallegue M, Kheddar A (2014) A strictly convex hull for computing proximity distances with continuous gradients. IEEE Trans Robot 30(3):666–678

Fletcher R, Leyffer S (2002) Nonlinear programming without a penalty function. Math Program 91(2):239–269

Fletcher R, Leyffer S, Toint P (2002) On the global convergence of a filter-SQP algorithm. SIAM J Optim 13(1):44–59

Gerdts M (2013) OCPID-DAE1: optimal control and parameter identification with differential-algebraic equations of index 1, User Manual, Version 1.3, Department of Aerospace Engineering, Universität der Bundeswehr München. http://www.optimal-control.de

Gerdts M (2012) Optimal Control of ODEs and DAEs. De Gruyter, De Gruyter textbook

Gerdts M, Henrion R, Hömberg D, Landry C (2012) Path planning and collision avoidance for robots. Numer Algebra Control Optim 2(3):437–463

Gilbert EG, Johnson DW (1985) Distance functions and their application to robot path planning in the presence of obstacles. IEEE J Robot Autom RA–1:21–30

Gilbert EG, Johnson DW, Keerthi SS (1988) A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE J Robot Autom 4(2):193–203

Goerzen C, Kong Z, Mettler B (2010) A survey of motion planning algorithms from the perspective of autonomous UAV guidance. J Intell Robot Syst 57(1–4):65–100

Hart GD, Anitescu M (2010) An O(m + n) measure of penetration depth between convex polyhedral bodies for rigid multibody dynamics

Johnson DW, Gilbert EG (1985) Minimum time robot path planning in the presence of obstacles, vol 24. In: 24th IEEE conference on decision and control, pp 1748–1753

Kim YJ, Lin MC, Manocha D (2002) DEEP: dual-space expansion for estimating penetration depth between convex polytopes. In: IEEE conference on robotics and automation, pp 921–926

Landry C, Gerdts M, Henrion R, Hömberg D (2013) Path-planning with collision avoidance in automotive industry. In: 25th IFIP TC 7 conference system modeling and optimization, Berlin. 12–16 Sep 2011. Revised Selected Papers IFIP AICT 391, Springer, Heidelberg; Approx. IX, 575 pp 2013

LaValle SM (2006) Planning algorithms. Cambridge University Press, Cambridge

LaValle SM, Kuffner JJ (2001) Randomized kinodynamic planning. Int J Robot Res 20(5):378–400

Lewis AS, Overton ML (2008) Nonsmooth optimization via BFGS. Technical report, http://www.cs.nyu.edu/overton/ papers/pdffiles/bfgs_inexactLS.pdf

Lewis AS, Overton ML (2013) Nonsmooth optimization via quasi Newton methods. Math Progr 141(1–2):135–163

Lin MC (1993) Efficient collision detection for animation and robotics. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley

Lin MC, Canny JF (1991) A fast algorithm for incremental distance calculation. In: IEEE international conference on robotics and automation, pp 1008–1014

Lin Q, Loxton RC, Teo KL (2014) The control parameterization method for nonlinear optimal control: a survey. J Ind Manag Optim 10(1):275–309

Loxton RC, Teo KL, Rehbock V, Yiu KFC (2009) Optimal control problems with a continuous inequality constraint on the state and the control. Automatica 45(10):2250–2257

Maheshwari A, Sack JR, Djidjev HN (1999) Link distance problems. Handbook of computational geometry, pp 519–558

Nocedal J, Wright SJ (2006) Numerical optimization, 2nd edn., Springer series in operations research and financial engineering. Springer, New York

Powell MJD (1978) A fast algorithm for nonlinearly constrained optimization calculations. In: Watson GA (ed) Numerical analysis, vol 630. Lecture notes in mathematics. Springer, Berlin, pp 144–157

Quarteroni A, Sacco R, Saleri F (2007) Numerical mathematics., Texts in applied mathematics. Springer, Paris

Saramago SFP, Steffen V (2001) Trajectory modeling of robot manipulators in the presence of obstacles. J Optim Theory Appl 110:17–34

Schittkowski K (1983) 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

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

Shor NZ (1985) Minimization methods for non-differentiable functions, vol 3., Springer series in computational mathematics. Springer, Berlin

Sprunk C, Lau B, Pfaffz P, Burgard W (2011) Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In: IEEE international conference on Robotics and automation (ICRA), pp 72–77

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