

# Off-Line and On-Line Trajectory Planning

Zvi Shiller

**Abstract** The basic problem of motion planning is to select a path, or trajectory, from a given initial state to a destination state, while avoiding collisions with known static and moving obstacles. Ideally, it is desirable that the trajectory to the goal be computed online, during motion, to allow the robot react to changes in the environment, to a moving target, and to errors encountered during motion. However, the inherent difficulty in solving this problem, which stems from the high dimensionality of the search space, the geometric and kinematic properties of the obstacles, the cost function to be optimized, and the robot's kinematic and dynamic model, may hinder a sufficiently fast solution to be computed online, given reasonable computational resources. As a result, existing work on motion planning can be classified into *off-line* and *on-line* planning. Off-line planners compute the entire path or trajectory to the goal before motion begins, whereas on-line planners generate the trajectory to the goal incrementally, during motion. This chapter reviews the main approaches to off-line and on-line planning, and presents one solution for each.

**Keywords** Motion planning · Trajectory optimization · Online planning

## 1 Introduction

One of the basic problems in robotics is that of motion planning, which attempts to move a robot from a given initial state to a destination state, while avoiding collisions with known static and moving obstacles. We distinguish between a *path* and a *trajectory*: a path *italic* represents a sequence of positions, defined in the robot's configuration space, which is the space of all positions, or configurations, that the robot can achieve [1]. A trajectory *italic* can be viewed as a path with a velocity profile along it, defined in the higher dimensional state space, where every point defines a position, or a configuration, and the velocity vector at that point. Thus, path

---

Z. Shiller (✉)

Department of Mechanical Engineering and Mechatronics, Ariel University, Ariel, Israel  
e-mail: shiller@ariel.ac.il

© Springer International Publishing Switzerland 2015

G. Carbone and F. Gomez-Bravo (eds.), *Motion and Operation Planning of Robotic Systems*, Mechanisms and Machine Science 29,  
DOI 10.1007/978-3-319-14705-5\_2

planning solves a geometric, or kinematic, problem, whereas trajectory planning solves a dynamic problem [1]. In this chapter, we will focus on trajectory planning.

Generally, it is desirable that the trajectory to the goal be computed online, during motion, to allow the robot to react to changes in the environment to a moving target, and to errors encountered during motion. However, the inherent difficulty in solving this problem, which stems from the high dimensionality of the search space, the geometric nature of the obstacles, the cost function to be optimized, and the robot's kinematic and dynamic model, prevents it from being solved sufficiently fast to be done online, given reasonable computational resources. As a result, two branches of research have emerged in the area of motion planning: off-line planning, where the trajectory to the goal is computed before motion begins, and on-line planning where the trajectory to the goal is computed incrementally during motion. We thus associate off-line with the computation of the entire trajectory to the goal, and on-line with incremental planning, regardless of the computational resources available for the planning process.

Another distinction between off-line and on-line planners is that the former may produce globally optimal solutions if the environment is fully known, whereas the latter is locally optimal at best. The challenges in off-line planning are therefore: optimality (local and global), completeness (will a solution be found if one exists), and overall computational complexity. The challenges in online planning are: completeness (is the planner guaranteed to reach the goal if a solution exists), computational complexity at each step, and optimality (how far is a solution from the optimal and is it bounded by an upper limit).

Off-line planners are most useful for repeatable tasks in static environments where optimality is essential, as is the case in many industrial applications. On-line planners are required in applications where the target states are determined on the fly, obstacles are discovered during motion, the environment is changing during motion, the computation time required for a global solution delays the task execution, or simply as an alternative to a computationally expensive off-line search [2].

The different nature of the two types of planners has resulted in distinct strategies to reaching the goal: the off-line planner takes generally a global view of the environment to select the optimal trajectory to the goal, whereas the on-line planner may select the next move based on a partial view of the environment. Both approaches were pursued at the early stages of the development of the field of motion planning in the early 80s [3–6]. The main focus then was on *path* planning, with the goal of computing the shortest path from start to goal in the presence of static obstacles [3, 4, 6].

Focusing on the shortest path resulted in geometric planners that account for the geometry of the moving object and obstacles. While the computation of an obstacle-free path may solve many important problems in industrial settings, where the robot may move slowly, it is insufficient, and almost useless, when the robot needs to move at reasonably high speeds, such as mobile robots moving through cluttered environments, and autonomous vehicles negotiating freeway traffic. Furthermore, the computed geometric path is insufficient to move the robot along the path unless

some speed profile along the path is specified. Selecting the speed profile that can be followed without the robot deviating from the path requires knowledge of the robot dynamic behavior. This brings to focus the problem of trajectory planning, which was addressed using tools rooted in optimal control theory. The two fields of research, path planning and trajectory planning, were developed in parallel over the years until recently when geometric planners were extended to searching for trajectories in the state space [7, 8]. While the focus in this chapter is off-line and on-line trajectory planning, we begin the literature review with geometric planners.

## ***1.1 Geometric Planners***

The introduction of the configuration space as the basic geometric motion planning tool [4, 9, 10] reduced the search for an obstacle-free path to computing a continuous path for a point from start to goal that avoids the forbidden regions representing the physical static obstacles. Being a geometric problem, most off-line planners are based on a geometric representation of the environment, through which a global search produces the shortest path to the goal. The geometric representations may consist of roadmaps or graphs that capture the topology of the free-space, generated e.g. by a Voronoi diagram, a visibility graph, a tangent graph [11, 12], or by cell decomposition [13]. Although each representation differs in the way it represents the free space, they all consist of a connected network of path segments that can be traversed from start to goal. The main computational effort in these planners is the representation of the free-space. This includes mapping of obstacles to the configuration space and the initial construction of the roadmap. Once the roadmap was constructed, the search for the shortest path is done using standard graph search techniques such as Dijkstra's search [14] or A\* [15]. The remaining difficulties stem from the dimensionality of the search space and the number of edges (segments) in the roadmap. The main computational effort here is the construction of the roadmap.

An alternative approach to constructing roadmaps is to overlay a uniform grid over the search space and represent the entire space by an undirected graph [2]. Assigning high costs to edges that intersect obstacles, effectively separates between inaccessible nodes and nodes in the free space. As a result, this, like all approaches that are based on a discrete representation of the search space, is resolution complete, implying that at low grid resolutions one may miss paths that pass through tight spaces between obstacles. Increasing graph resolution would severely impact the computational complexity. Compared to the roadmap-based algorithms, the number of nodes for the uniform grid representation is much greater. However, this representation, which is quite general, is applicable to problems where obstacles are not clearly defined, such as for mobile robots moving over rough terrain [16], as is demonstrated later in this chapter.

The increased interest in solving high dimensional problems, such as motion planning for humanoids or multi degrees-of-freedom arms, gave rise to a class of

sampling-based planners [17]. The most popular version of sampling-based planners is based on rapidly exploring random trees (RRT) [17–19]. They search the way to the goal by probing the configuration space (can be also done in the workspace) and incrementally expanding a collision-free tree from an initial configuration. Because the entire search is done “in the dark,” the planner attempts to reach unexplored parts of the search space, resulting eventually in a uniform coverage of the free space. The efficiency of these planners stems from the incomplete coverage of the free space and from terminating the search when the goal is first reached. The solution found is feasible but not optimal in any way. RRT based planners may not produce optimal solutions even when exploring the entire search space [7, 20]. In addition, they inherently have difficulties with tight spaces. Nevertheless, the RRT algorithms were demonstrated for solving very complex problems [21].

An extension called RRT\* was developed to asymptotically produce the optimal solution [7]. The asymptotic optimality was achieved by adding a lower bound estimate to the RRT search. Optimality is achieved iteratively at a great computational cost by running the algorithm repeatedly while refining the solution until either exhausting the available computation time or reaching a desired level of optimality [7]. Despite the great promise, the RRT\* algorithm was demonstrated in [7, 20] for the kinematic avoidance of very few planar and widely spaced obstacles, producing a smooth near-optimal solution after a large number of iterations (10,000). Solving a dynamic problem in a higher dimensional state space is expected to be much more challenging. Recently, a path generated by an RRT search was further optimized using a genetic algorithm to produce the shortest path for a hybrid manipulator with six degrees-of-freedom [22]. The idea of further optimizing paths that were generated by a geometric planner is similar in nature to the off-line planner presented here, except that the objective of the off-line planner is to produce a global optimal trajectory, not only the shortest path.

The sampling-based planners represent a paradigm shift in the motion planning community by (1) accepting probabilistic completeness, which is to say that the goal may not be reached in a finite time, (2) accepting any solution, not necessarily the optimal, and (3) abandoning the explicit geometric representation of the free configuration space in terms of roadmaps or graphs. This is a significant departure from the previous practices that evaluated motion planning algorithms for completeness and optimality.

## ***1.2 Trajectory Planning (Off-Line)***

The trajectory planning problem concerns the computation of robot motions that move the robot between two given states, while avoiding collision with obstacles, satisfying robot dynamics and actuator constraints, and usually minimizing some cost function, such as energy or time.

Early work on trajectory planning, from the 1960s to the 1980s, was rooted in the field of optimal control theory, which provides powerful tools to characterize and

generate optimal trajectories when high speed motion is desired [23]. The elegant necessary conditions, stated by the Pontryagin's maximum principle, lead to the formulation of the optimal control problem as a two-point boundary value problem, and the development of algorithms that searched for the optimal control that generates the optimal trajectories [23]. For time optimization problems, it was shown that the time-optimal control is bang-bang. This in turn reduces the optimal control problem to a parameter optimization by iterating on the switching times between the maximal controls [24, 25].

The first attempt to use these theories for robotics was by Khan and Roth [26], who computed the multi-axis time optimal trajectory for a linearized model of robot dynamics. Solving this problem for the full robot's dynamic model was computationally very difficult. The typically non-linear and coupled robot dynamics makes such solutions computationally extensive. Adding obstacles makes the computational challenge even harder.

One approach to reducing the complexity of the problem and facilitating a practical realization of time-optimal motion planning is to decouple the problem by representing robot motions by a path and a velocity profile along the path. This decoupling allows reducing the trajectory planning problem to two smaller problems: (a) computing the optimal velocity profile along a given path, and (b) searching for the optimal path in the  $n$ -dimensional configuration space.

The time optimal velocity profile along a specified path is computed using an efficient algorithm, originally developed Bobrow, Shin and McCay and Pfeiffer [27–29], and later improved by Shiller and Lu [30] and Slotine and Yang [31]. Assuming a second order system, the solution to this problem was found to be bang-bang in the acceleration, that is, applying the maximum or minimum acceleration so as to maximize the velocity along the path. The switching times are computed efficiently to avoid crossing the velocity limit curve, which reflects the actuator constraints and the robot dynamics. This approach was later extended to computing time optimal velocity profiles along specified paths for nonlinear third order systems, subject to general jerk constraints [32].

The optimal path was computed using a nonlinear parameter optimization over path parameters, such as the control points of a cubic B spline [33, 34]. In each iteration of the optimization process, the optimal velocity profile along the path is efficiently computed to produce the minimum time for that iteration. One advantage of this approach is that each iteration yields a feasible trajectory, albeit not necessarily optimal. The optimization can therefore be terminated at any time to yield an acceptable solution.

A similar approach, known as *direct optimization*, *differential inclusion* [35], and *inverse dynamics optimization* [36], was proposed by the aerospace community. Common to these methods is the direct search for the optimal trajectory as opposed to a search for the optimal control in the higher dimensional state and co-state spaces [23]. Attempts to solve the multi-axis problem using graph search techniques in the state space, solving the so called “kinodynamic” problem, did not yield practical solutions [37, 38].

### 1.3 Online Planning

Early on-line planners were developed to address the lack of apriori information about the environment. Called “sensor-based” algorithms, they navigate a point robot equipped with position and touch sensors among unknown obstacles to reach a global goal. A series of “bug” algorithms were developed, starting with the basic bug that navigates by circumventing the detected obstacle always clockwise or counterclockwise until reaching the straight line to the goal, then continuing along that line until either reaching the goal or hitting another obstacle [5]. Assuming long range vision sensors, the bug strategies were extended to the Tangent Bug algorithm, which follows the tangent line to the next obstacle that obstructs the straight line to the goal [39, 40]. It was shown that complete online navigation can be achieved with only a finite amount of memory [5, 41, 42].

Another approach to online motion planning is based on potential functions [43–45]. Representing the goal with an attractive potential, and the obstacles with repulsive potentials, the path is generated online by following the negative gradient of the potential function. While this approach is computationally efficient and is suitable for on-line feedback control, it suffers from local minima, which may cause the path to terminate at a point other than the goal. This problem was overcome using harmonic potentials [43] and navigation functions [45]. These potentials, however, address only the obstacle avoidance problem with no concern for path optimality. Furthermore, the generation of the potential function is done off-line and may be time consuming.

A similar approach generates the shortest path by following the direction of steepest descent of a discretized distance function [46]. The main computational effort is in numerically computing the distance function, which is done off-line. The computation complexity increases rapidly with the number of obstacles and with grid resolution.

The potential functions used to guide the trajectory towards the goal resemble the value function, which is the solution to the Hamilton- Jacobi-Bellman (HJB) equation [47–49]. The HJB equation states a sufficient condition of global optimality (unlike the Pontryagin Maximum Principle, which is only a necessary condition), and the value function represents the cost-to-go from any feasible state. The globally optimal trajectory is then generated by selecting the controls that minimize the time derivative of the value function. For time invariant systems, this amounts to following the negative gradient of the value function, which drives the system time-optimally to the goal from any initial state. This is similar to the potential field method, except that the value function may be regarded as the “optimal” potential function.

Although the theoretical framework exists for deriving optimal feedback controllers, it is impractical to derive a time-optimal control law, using the HJB equation, for a typical obstacle avoidance problem that accounts for robot’s dynamics.

A recently developed online algorithm navigates towards the goal by optimally avoiding one obstacle at a time [50, 51]. This transforms the multi-obstacle problem with  $m$  obstacles to  $m$  simpler sub-problems with one obstacle each, thus reducing the size of the problem from exponential to linear in the number of obstacles.

The incremental generation of the trajectories and the relatively low computational effort at each step make this algorithm an efficient on-line alternative to the computationally expensive off-line planning, thus trading optimality for efficiency. This algorithm will be later discussed in this chapter.

This chapter is organized as follows: it starts with a formal problem statement of the motion planning problem, focusing on trajectory planning rather than on path planning. It continues with the theoretical solution for the optimization problem using the Hamilton Jacobi equation. It then describes an efficient off-line planner and a very efficient online planner. Both algorithms are demonstrated for a point robot moving at high speeds over rough terrain (the off-line planner) and through very cluttered environments (the online planner).

## 2 Problem Statement

In a typical motion planning problem, we wish to solve the following optimization problem:

$$\min_u \int_0^{t_f} L(x, u) dt \quad (1)$$

subject to the system dynamics:

$$\dot{x} = f(x, u), \quad (2)$$

where  $x \in \mathbb{R}^n$  is a point in the robots state space, and  $u \in \mathbb{R}^m$  is a vector of actuator efforts, subject to the actuator constraints:

$$u_{imin} \leq u_i \leq u_{imax}, i \in \{1, \dots, m\}, \quad (3)$$

obstacle constraints:

$$g(x) \geq 0; \quad g \in \mathbb{R}^k, \quad (4)$$

and the boundary conditions:

$$x(0) = x_0; \quad x(t_f) = x_f, \quad (5)$$

where  $k$  is the number of obstacles and  $t_f$  is the final time. If the objective function is time, i.e.  $L(x, u) = 1$ , then  $t_f$  is free. We assume that the obstacles (4) do not overlap with each other and with the goal  $x_f$ .

Problem (1) is a two point boundary value (TPBV) problem: of all trajectories that satisfy the boundary conditions (5), select the one that minimizes the cost function (1) and satisfies system dynamics (2), control constraints (3) and obstacle constraints (4).

The global optimal trajectory can be computed using the Hamilton-Jacobi-Bellman (HJB) equation, which states a sufficient condition for global optimality [47–49]. Denoting the set of obstacles as  $O$ :

$$O = \{x : g(x) < 0\}, \quad (6)$$

The control  $u^*$  that is the solution to problem (1), satisfies, on  $\mathbb{R}^n - \{x_0\} - O$ , the HJB equation:

$$\min_u \{v_t(x, t) + \langle v_x(x, t), f(x, u) \rangle\} = -L(x, u) \quad (7)$$

subject to (3) and (4), where  $v(x, t)$  is a  $C^2$  scalar function, satisfying

$$v(x_0, t) = 0 \quad (8)$$

$$v(x, t) > 0, x \neq x_0 \quad (9)$$

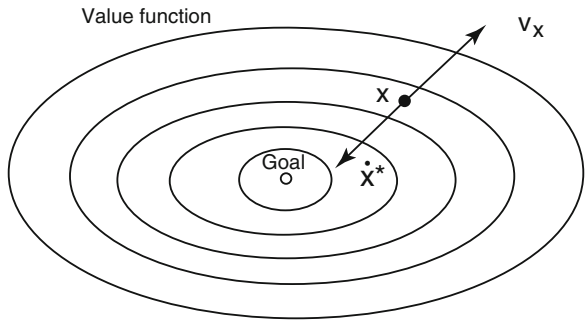
The subscripts  $x$  and  $t$  represent partial derivatives with respect to  $x$  and  $t$ , respectively, and  $\langle \cdot, \cdot \rangle$  denotes the inner product on  $\mathbb{R}^n$ .

The scalar function  $v(x, t)$  is the *value* function [47, 48, 52], representing the minimum *cost-to-go* to the origin (goal) from any given state. For an autonomous system (time-invariant) and for fixed boundary conditions,  $v_t = 0$ ; assuming in addition that the cost function to be minimized is time ( $L(x, u) = 1$ ), reduces (7) to:

$$\min_u \{\langle v_x(x), f(x, u) \rangle\} = -1 \quad (10)$$

To satisfy (10), the projection of  $\dot{x} = f(x, u)$  on  $v_x(x)$  must equal  $-1$ . It follows that the optimal control  $u^*$  that minimizes (10) drives the optimal trajectory  $\dot{x}^*(x, u)$  in the direction of the negative gradient,  $-v_x(x)$ , of the value function, as shown schematically in Fig. 1. This is similar to the trajectory generation by potential field methods [43–45, 53], except that here the potential function is the value function. Since the value function has a unique minimum at the goal, trajectories generated by following the negative gradient of the value function are globally optimal and are guaranteed to reach the goal from any initial state.

**Fig. 1** The optimal trajectory  $\dot{x}^*(x, u^*)$  slides opposite to the gradient  $v_x(x)$  of the value function





The on-line planner for the multi-obstacle avoidance problem, described later in this chapter, can be viewed in the context of the value function as following the negative gradient of an approximate value function for this problem. It generates near-optimal trajectories by avoiding obstacles *one at a time*, or equivalently, by sequentially following the negative gradient of the return function for each obstacle avoidance problem. The trajectory is generated incrementally, permitting robot motion before the entire trajectory to the goal has been computed.

Obtaining an analytical expression for the value function is practically impossible for other than for very simple cases. Computing the value function numerically would require solving the optimization problem from every point in the state space. This is essentially the approach used in [46] for solving the shortest path problem.

A discrete version of the HJB equation is the basis for the Bellman's Principle of Optimality and Dynamic Programming [54]. Dynamic programming is the optimization method used in most grid based optimizations, including the off-line optimization discussed next in this chapter.

### 3 Off-Line Planner

The off-line planner presented here computes the global time optimal trajectory between given boundary states in the presence of known static obstacles [2]. It combines a grid search in the configuration space with a continuous local optimization. In lieu of an expensive search in the  $2n$  dimensional state space for one (globally optimal) trajectory, this planner searches for many paths in the  $n$  dimensional configuration space for an  $n$  degree of freedom robot. The reduction of the search to the configuration space yields a significant (exponential) computational gain compared to a full search in the state space. The complexity of this approach is exponential in the dimension of the configuration space and linear in the number of nodes in the graph.

#### 3.1 Summary of the Approach

This planner is based on a branch-and-bound search for the global optimal trajectory between given end states in a static environment. It assumes an efficient mapping from a curve in the configuration space to the optimal traversal time along that curve. This mapping allows us to search for the optimal trajectory in the lower dimensional configuration space. We call the projection of the optimal trajectory on the configuration space the *optimal path*.

The branch-and-bound search begins by reducing the infinite set of paths between given end points to a final set by representing the configuration space by an undirected graph. The branch-and bound search then reduces this set to a small set of the most promising paths. The paths in the final set are then pruned to retain the best path in each path-neighborhood. These paths are then optimized using a nonlinear parameter

optimization to further reduce motion time. This last step significantly relaxes the grid resolution required for the initial search to ensure global optimality.

This process was proven to generate the global optimal trajectory in addition to producing a set of local minima [2]. The optimality of the solution depends on the number of paths selected in the first step, grid resolution with respect to the distance between obstacles, and the fidelity of the local optimization. This optimization was demonstrated for a six DOF manipulator moving in a cluttered environment [2] and for a mobile robot moving on general terrain [16].

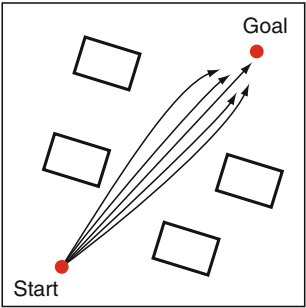
### 3.2 The Graph Search

The purpose of the graph search is to efficiently produce a set of paths that explore all regions in the configuration space and that may contain the optimal path. Obstacles are accounted for by setting high costs to edges that penetrate obstacles. For motion over rough terrain, obstacles are accounted for by considering their geometric shape and determining if the robot can safely traverse these obstacles, similarly to traversing other terrain features [16].

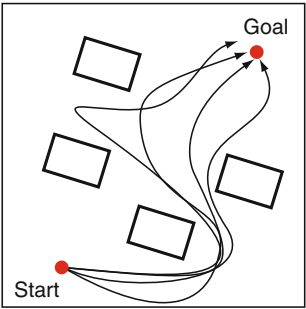
In the context of this algorithm, the optimal path is the one that can be traversed at the minimum time between given end points, subject to robot dynamics, and to control and obstacles constraints. Since metrics measured in the configuration space are not good predictors for path optimality, it is necessary to consider a large number of paths to ensure that they contain at least one path in the neighborhood of the optimal path. By representing the configuration space with a uniform grid, we reduce the infinite number of obstacle-free paths to a finite set.

One approach to generating a large set of paths, using a graph search, is to use the  $k$ -best search by Dreyfus [55] to produce a set of shortest paths. It is similar to a shortest path search except that it effectively excludes the  $k - 1$  best paths from the searched space while searching for the next  $k$ th best path. This allows us to sequentially generate the paths until some upper bound on the cost function, determined by the branch and bound search, is reached. The cost function may be path length, or some other function that produces a lower bound estimate of the optimal motion time along the path [2]. While this approach guarantees that the global and a few local minima (within grid resolution) are found, it has the drawback that it first generates a large number of paths in the neighborhood of the best path ( $k = 1$ ), usually in one homotopy class, before exploring other homotopy classes, as shown schematically in Fig. 2. A homotopy class contains all paths that can be continuously deformed into one another [1], as shown schematically in Fig. 3. Depending on grid resolution, using the  $k$ -best search may require a very large number of paths in order to cover the entire space. In addition, identifying a local minimum (that is not global optimal) is quite tedious. The difficulty arises from the regions of optimality not being easily quantified, and hence requiring that each new path be tested if it is in the neighborhood of any path generated so far. The large number of paths required by this approach thus imposes a high computational cost, first with the  $k$ -best search, which is linear in  $k$ , and then in the pruning process, which is  $O(k \log k)$ .

**Fig. 2** Near shortest paths found by the K shortest path algorithm tend to group around the shortest path



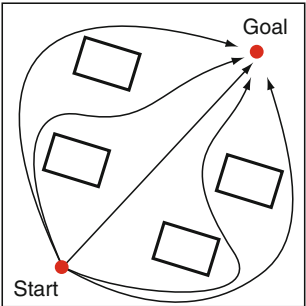
**Fig. 3** Paths in one homotopy class



The pruning process consists of selecting the best (shortest in time or distance) of all paths in the initial set of paths, then discarding all paths that are within some tube of a predefined diameter around the best path. The paths within a tube around the next best path are similarly discarded, and the process repeats until all paths in the initial set are either discarded or retained as the best path in their neighborhood. The pruning process thus reduces an initially large set of paths to a smaller set of promising paths, each is then locally optimized, as discussed later.

An alternative approach efficiently generates a large number of paths that cover the entire search space [56, 57] and can be easily reduced to the most promising path in each homotopy class, as shown schematically in Fig.4. In two steps, each consisting of a shortest path search, it generates all shortest paths that pass through

**Fig. 4** Paths of various homotopy classes



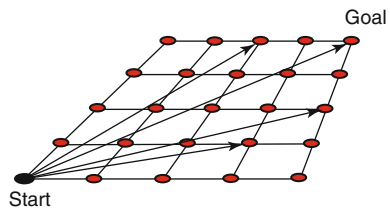
each node in the graph. This allows an efficient coverage of the entire free-space and the identification of a few promising local minima in addition to the global optimal path .

This is essentially a *single-pair* search for  $n$  constrained paths through a graph with  $n$  nodes. It starts with a single-source search, such as Dijkstra's [14], that generates the shortest path from the source  $s$  to the goal  $g$  (Fig. 5). The cost  $a_{s,i}$  stored at each node  $i$  is the cost from  $s$  to that node. Repeating this search from the goal  $g$  to  $s$  stores the cost  $b_{g,i}$  at each node  $i$  (Fig. 6). Summing the two costs  $c_{s,g,i} = a_{s,i} + b_{g,i}$  yields the optimal cost  $c_{s,g,i}$  for the path between  $s$  and  $g$  that passes through node  $i$  (Fig. 7).

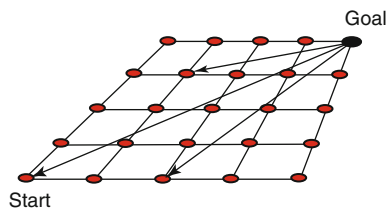
If each local minimum represents a homotopy class, the computational cost of this approach is  $O(2)$  for the initial search, and on average  $O(m/p \log m/p)$  for the pruning process, where  $m$  is the number of nodes and  $p$  is the number of homotopy classes generated by this search [57]. Compare to the  $k$ -best search,  $O(m)$  for the initial search and  $O(m \log m)$  for pruning. This efficiency is achieved at the cost of generating only a subset of all possible local minima, but at a computational cost far smaller than the alternative.

Figure 8 shows a topographic surface that is to be traversed from Start to Goal. The surface was first tessellated by a uniform grid, then the shortest paths through all nodes were computed using the algorithm discussed earlier [56, 57]. Color marking

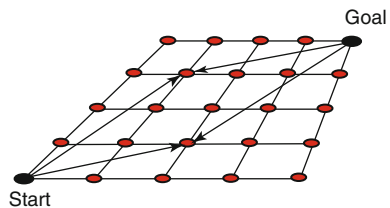
**Fig. 5** Shortest paths from start to all nodes

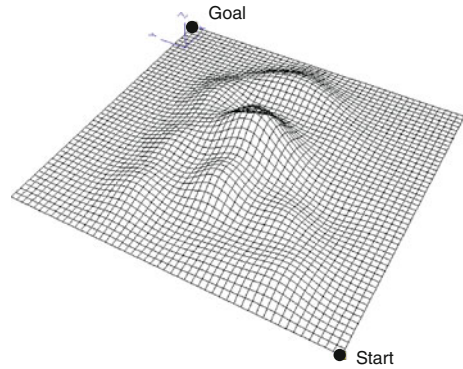
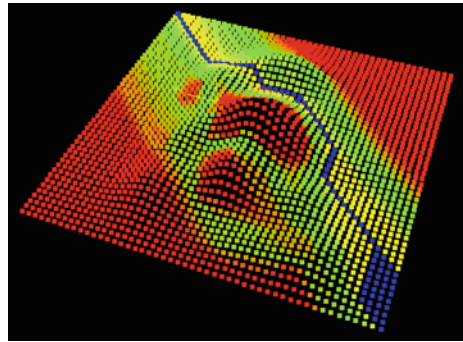


**Fig. 6** Shortest paths from goal to all nodes



**Fig. 7** Shortest paths through via points



**Fig. 8** A surface map**Fig. 9** A color coded surface map. The color at each node represents the optimal cost for passing through that node

the nodes according to the cost of passing through each node produced the *cost map* shown in Fig. 9. The cost function for this case was a traversability measure, calculated by dividing distance by the maximum safe velocity along each segment along the graph [16]. Here, blue represents the lowest cost (global minimum), then yellow, green and red represent gradually increasing costs. The *cost map* clearly shows the traversability of each region, thus offering sub-optimal alternatives to the global optimal path, which is colored blue. The blue “river”, whose nodes all have the same (optimal) cost, might be wider in regions where the neighboring edges have identical costs. In such cases, the global optimal *grid* path may not be unique, which is a common artifact of the uniform discretization of the search space.

### 3.3 Branch and Bound Search

The goal of the branch-and-bound search is to efficiently reduce the initially large set of paths in each homotopy class to a smaller set that contains the local optimal path. This is done by dividing the initial set of paths into two smaller subsets: one that contains all paths having a lower bound estimate on their cost that is higher

than the lower bound estimate of all paths in the second subset. The second subset is discarded, and the process repeats by subdividing the remaining subset using a more accurate lower bound estimate. Repeating this process, using a series of gradually increasing lower bounds, thus reduces the initial large set of paths to a much smaller set of promising paths. The search is terminated when the last subset has been shown to contain no better solution than the one already at hand. The best solution found during this search is the optimal path [15]. The fastest among the local minima found in this process is the global optimal path.

In this search, the objective function is the minimum traveling time between the two end points, whereas the initial set consists of all feasible (collision-free) paths between the given end points. It remains to determine appropriate approximations of the cost function that are guaranteed to produce lower bounds on the traveling time along a given set of paths. The computational efficiency of this approach depends on the proper selection of the lower bound estimates at each step. The most conservative but efficient approximations are used first, when the number of path candidates is large, and the more accurate but computationally expensive are used last. The last test is the exact solution, which is the optimal traveling time along the path.

We use three lower bound estimates on the optimal motion time along a given path, each represented by a different velocity profile: (1) maximum constant speed, (2) velocity limit, and (3) optimal velocity along the path. The cost estimate is computed by integrating the respective velocity profile along the path.

**Maximum Speed:** The first lower bound estimate,  $t_1$ , assumes motion everywhere at the maximum speed the robot can reach. It can be the tip velocity reached by assuming no load speeds at all joints at the most stretched configuration, or the maximum speed a mobile robot can reach on flat terrain. Dividing the distance along each edge of the graph by the maximum speed produces a lower bound estimate, obtained by the summation:

$$t_1 = \sum \frac{\Delta x_i}{v_{max}}, \quad (11)$$

where  $\Delta x_i$  is the Euclidean distance of the  $i$ th segment along the path, and  $v_{max}$  is the maximum speed.

Having assigned a fixed cost to all edges, the paths produced by the graph search are rated by a lower bound estimate on the optimal motion time along each path. Paths with lower bound estimates higher than the optimal motion time along some arbitrary path can be discarded early in the search process.

**Velocity Limit:** Once a path has been selected from the grid search, it is smoothed by cubic B splines, using the nodes of the graph along the path as control points. This eliminates the sharp corners produced by the grid segments. If the smoothed path penetrates an obstacle because of the rounded corners, it can be either discarded or kept for the next lower bound test. Eventually, the local optimization, discussed later, will divert the path away from the obstacle.

Here we assume that the speed along the path follows the velocity limit curve  $\dot{s}_{max}(s)$ ,  $s$  being the distance parameter along the path, that accounts for robot dynamics, actuator constraints, and path curvature, at every point along the path [27–29, 32].

The lower bound  $t_2$  is obtained by the integral

$$t_2 = \int_0^{s_f} \frac{ds}{\dot{s}_{max}}, \quad (12)$$

The computation of the velocity limit and the optimal velocity profile are briefly discussed later.

The value  $t_2$  is a true lower bound and greater than  $t_1$  since the velocity limit curve represents the true upper limit for the velocity profile along the path. This evaluation is computationally more demanding than the previous one but is less expensive than computing the time optimal velocity profile. This lower bound takes into account the combined effects of robot dynamics, actuator constraints, and path geometry.

**Optimal Velocity:** This is the exact solution for the optimal motion time and an upper bound to the previous lower bounds. The optimal velocity profile is always below the limit curve and at most tangent to the limit curve at a finite number of points [30]. The computation of the optimal velocity profile is briefly discussed next.

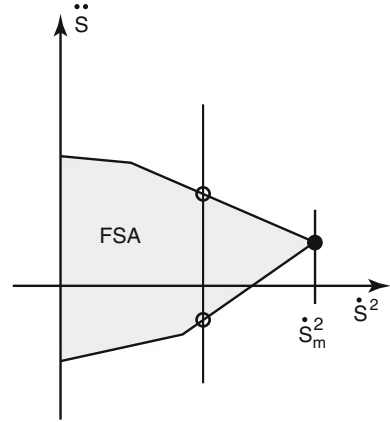
### 3.4 Time Optimal Motions Along Specified Paths

The optimal motion time along the path represents the exact cost function for the global search. It is computed using a well established algorithm [27–31, 58], which accounts for robot dynamics, actuator constraints, and path geometry. It is applicable to any fully actuated system such as industrial and mobile robots [16]. The algorithm will not be repeated here, referring the reader to the respective literature [27–31, 58].

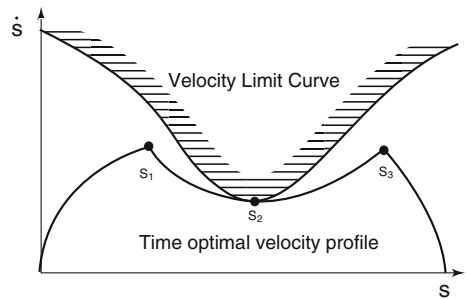
Key to this algorithm is the mapping of system dynamics to path coordinates. This reduces the multi dimensional configuration space, in which the robot operates, to a single degree-of-freedom system, where the distance and speed along the path,  $s, \dot{s}$ , are its two states, and the tangential acceleration  $\ddot{s}$  is its control input. The actuator constraints, coupled with path geometry, are mapped to constraints on  $\dot{s}$  and  $\ddot{s}$ , as shown schematically in Fig. 10 at some point  $s$  along the path. The boundary of the range of speeds and accelerations,  $FSA$ , represents states where at least one actuator reaches its limit. States outside of  $FSA$  are therefore dynamically infeasible.

At a given speed, the acceleration is bounded between its maximum and minimum values, as shown in Fig. 10. The speed  $\dot{s}_m$ , where the range of feasible accelerations reduces to a point, represents the highest speed at which the robot can still move along the prescribed path. Plotting  $\dot{s}_m$  along the path produces the velocity limit

**Fig. 10** The range of feasible speeds and accelerations (FSA)



**Fig. 11** Velocity limit curve and time optimal velocity profile



curve, as shown schematically in Fig. 11. It serves as the upper limit for any velocity profile along the path, optimal or not. Crossing the velocity limit curve implies that the robot is moving at speeds that are not sustainable by the robot's actuators or that it does not follow to prescribed path.

The time optimal velocity profile is computed using "bang-bang" control, switching between maximum acceleration and maximum deceleration along the path. The switching times are selected so that the optimal velocity profile avoids crossing the velocity limit curve [30], as shown schematically in Fig. 11. In the schematic example shown in Fig. 11, the time optimal velocity profile is integrated from the initial point at zero speed, using the maximum acceleration. At some point  $s_1$  along the path, the acceleration is switched to the maximum deceleration until point  $s_2$ , where the optimal velocity profile is tangent to the velocity limit curve. From  $s_2$ , the maximum acceleration is again integrated until some point  $s_3$ , from where the maximum deceleration is used to reach the final point at zero speed. The number of switches is usually odd for a 2nd order system, and it depends on the shape of the velocity limit curve, and the robot's dynamic properties. This algorithm is computationally very fast and can be used to efficiently assign the optimal motion time to every path in the last set of paths of the branch and bound search.



### 3.5 Local Optimization

The paths generated over the graph are forced to pass through the nodes of the graph defined by the grid used to represent the search space. To relax the demands on the grid resolution, a local optimization is used to locally alter the path to further reduce motion time [33, 34]. The optimization problem is formulated as an unconstrained parameter optimization, using the control points of cubic B splines as the optimization variables, and the optimal motion time along the path as the cost function. Obstacles are represented by penalty functions that account for the distance between the robot and the obstacles. At each iteration of the local optimization, the optimal motion time along the current path is computed using the method discussed earlier in Sect. 3.4, and the control points are modified by the optimization algorithm so as to produce paths with gradually decreasing optimal motion times. This process repeats until the optimal motion time reaches a local minimum. This optimization is obviously local since the path cannot “jump” over obstacles.

To reduce computation time and improve the convergence of the local optimization, the number of control points is reduced by retaining only a few points for each straight line segment along the grid path. It is important to note that a small number of control points may not adequately represent the true optimal path, however, a large number of parameters may be computationally costly. The true optimum can be approached asymptotically by successively increasing the number of control points and repeating the local optimization.

The local optimization is used to optimize only a small number of promising paths, selected from the paths remaining after the branch and bound search. These paths are selected as the best in each homotopy class [57] or as the best in some defined neighborhood of radius  $D_{max}$ . The classification of the paths into homotopy classes is discussed in [57] and will not be repeated here. The selection of the best path in each neighborhood is done by first discarding all paths that are contained in a tube around the best path, each satisfying the inequality:

$$D = \max |(p_i(w) - p_0(w))| < D_{max}, \quad w = [0, 1]; i = 1 \dots N, \quad (13)$$

where  $p_0(w)$  is a point along the best path in the neighborhood, with  $w$  being a normalized path distance, and  $p_i(w)$  is a point along any path in the remaining set of  $N - 1$  paths. This process is repeated for the next best path among the remaining paths until only a few paths, representing distinct regions, remain.

### 3.6 Summary of the Off-Line Planner

The off-line planner that uses the K-best search, is summarized in the following pseudo code. In the following, “best path” refers to the path along which the optimal motion time or a lower bound estimate is the smallest of all paths in the given set.

**Algorithm 1:** *Off-line planning***Step 0:** Initialize.

Receive the geometric description of the workspace, robot dynamics, actuator constraints, dynamic and state constraints, current state  $x$ , target state  $x_f$ ;

Determine the robot maximum speed  $v_{max}$  for Eq. (11);

Set an upper bound  $t_{up}$  to be used to terminate the first search;

Set diameter  $R$  for path filtering.

**Step 1:** Generate a graph over the workspace

Assign cost, usually Euclidean distance, to all edges on the graph.

Assign high cost to edges that connect unreachable nodes.

**Step 2:** Use the K-best search to generate the set  $P_0$  of shortest paths between the end points (the projections into the configuration space of the current and target states). Stop the search when  $t_1(K) \geq t_{up}$ .

**Step 3:** Smoothing.

Smooth all paths in  $P_0$  by B-splines, using the nodes along each path as control points.  $P_1$  is the set of  $K$  smoothed paths.

**Step 4:** For all paths in  $P_1$ , compute a lower bound estimate  $t_2(i)$ ,  $i = 1, \dots, K$ , using (12).

**Step 5:** Select the best path  $j$ :  $t_2(j) = \min\{t_2(i), i = 1, \dots, K\}$ . Compute the optimal motion time  $t_3(j)$ ;  $t_3(j)$  serves as the next upper bound in the branch and bound search.

**Step 6:** Move all paths in  $P_1$  that satisfy  $t_2(i) \leq t_3(j)$ ,  $i = 1, \dots, K$ , to  $P_2$ .

**Step 7:** Compute the optimal motion time  $t_3(i)$  for all paths in  $P_2$ .

**Step 8:** Pruning.

Select the best path in  $P_2$  and discard all paths that are inside a tube of radius  $R$  around that path, using (13); Move the best path to  $P_3$ ; Repeat for the next best path in  $P_2$  until  $P_2$  is empty.  $P_3$  now contains a small set of “good” paths.

**Step 9:** Local optimization.

Submit all paths in  $P_3$  to a local optimization. The resulting paths form the set of local minima  $P_4$ .

**Step 9:** Global optimum.

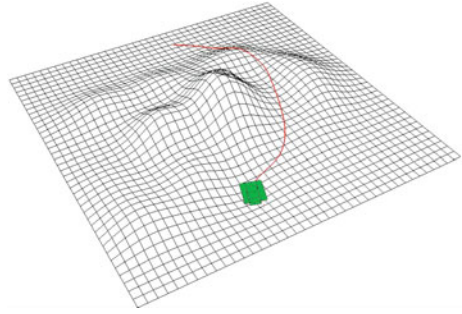
The best path in  $P_3$  is the global optimal path, along which the optimal motion time is globally optimal.

STOP.

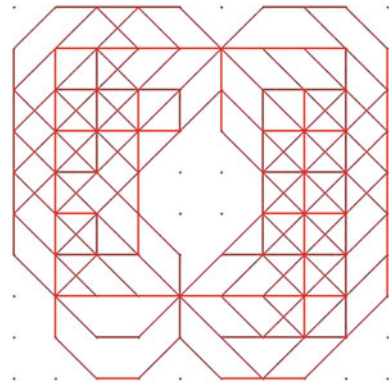
### 3.7 Example 1

Figure 12 shows the near-global time optimal trajectory, computed using the global optimization discussed here, for a vehicle moving over general terrain. For this example, the  $k$ -best search was used to generate the initial set of 500 paths, all shown in Fig. 13. The grid resolution was set low at 1 m between nodes for a  $10 \times 10$  m terrain segment. The branch and bound search retained 22 best paths, each was smoothed by

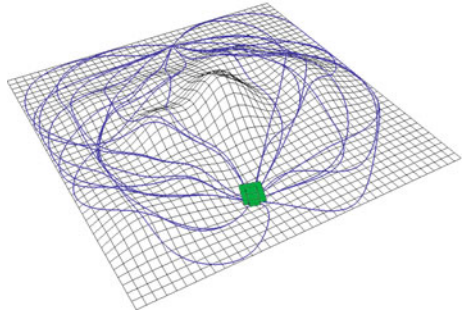
**Fig. 12** A (near) global time optimal path over general terrain, generated by the global planner



**Fig. 13** 500 shortest paths generated over the uniform grid overlayed over the terrain

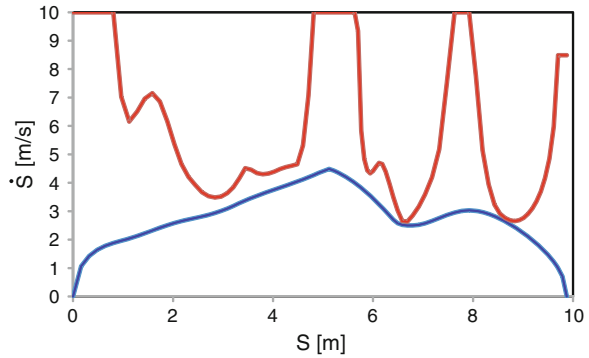


**Fig. 14** 22 best smoothed paths retained by the branch and bound search



a cubic B spline, as shown in Fig. 14. All 22 paths were locally optimized to further reduce motion time, and the best path, shown in Fig. 12, was selected as the global optimal solution. The time optimal velocity profile along the best path is shown in Fig. 15. Also shown in Fig. 15 is the velocity limit curve. Note that the vehicle slows down before accelerating again to prevent it from reaching high speeds that would cause it to airborne over the bump in the upper part of the terrain segment. The effect of the bump on the vehicle speed is reflected in the drop of the velocity limit curve.

**Fig. 15** The optimal velocity profile and the velocity limit curve along the time optimal path



The solution obtained is near global optimal due to the choices of the grid resolution and the termination condition of the local optimization.

Computation time depends on the number of paths generated in the graph search, the number of promising paths left for the local optimization, and the number of control points used to represent each path. The global planner was implemented in C and run on an Intel core-i7 3:4GHz desktop computer. For example1, the global optimal path was computed in 20 s, most of which was spent on the local optimization of 22 paths.

The global optimization presented here is inherently off-line as it produces the complete solution to the goal. It combines a search for a set of the best paths in a grid in the configuration space with a local path optimization. This combination allows to reduce the search to the lower dimensional configuration space without compromising optimality. There are only few global planners that we can compare to, especially those computing time optimal trajectories [7, 37].

The solution produced by this planner is a global optimum if the grid is sufficiently small. The requirement on grid resolution is relaxed by assuming that the region of convergence around the optimal path is large compared to the grid size. Despite this approach being presented long ago, it is still computationally efficient compared to more recent global optimizations [7, 37]. Lacking information on the use of RRT\* to solving dynamic problems, it is difficult to compare this popular approach to ours.

## 4 Online Planner

We now address the online time-optimal obstacle avoidance problem for robots moving in cluttered environments. Motivated by the observation that the effect of an obstacle on the value function (the global cost-to-go function) in (10) is local [51], we solve the multi-obstacle problem by avoiding obstacles one at a time. This

is equivalent to approximating the value function of the multi-obstacle problem by switching between the value functions of the individual problems, each avoiding a single obstacle. Computationally, this transforms the multi-obstacle problem with  $m$  obstacles to  $m$  simpler sub-problems with one obstacle each, thus reducing the size of the problem from exponential to linear in the number of obstacles. As a result, this approach produces an on-line planner, i.e. the trajectory is generated incrementally, one step at a time, requiring a low computational effort at each step relative to the original, inherently off-line, problem.

While the approach of avoiding obstacles optimally one at a time applies to any robot dynamics, and convergence can be guaranteed for any obstacle shapes, we treat here a point mass robot in the plane and convex obstacles.

We begin with the optimal avoidance of one obstacle.

## 4.1 Optimal Avoidance of a Single Obstacle

The time optimal avoidance of a single obstacle in the plane is relatively simple. It can be computed using a global optimization [2], or by running a local optimization [34] twice (one for each side of the obstacle for a planar problem).

Consider the following point mass model:

$$\begin{aligned}\ddot{x} &= u_1 \quad ; \quad |u_1| \leq 1 \\ \ddot{y} &= u_2 \quad ; \quad |u_2| \leq 1\end{aligned}\tag{14}$$

where  $(x, y)^T \in \mathbb{R}^2$  and  $(u_1, u_2)^T \in \mathbb{R}^2$  represent the configuration space variables and actuator efforts, respectively.

We first derive the *unconstrained* trajectory, for states not affected by the presence of the obstacle.

### 4.1.1 The Unconstrained Trajectory

The unconstrained trajectory for the decoupled system (14) is determined by the minimum motion time of the *slowest* axis.

Consider first a single axis, represented by the double integrator

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= u \quad ; \quad |u| \leq 1.\end{aligned}\tag{15}$$

Using optimal control theory [23], it is easy to show that the time-optimal control for system (15) is bang-bang with at most one switch [50]. In the following, we denote  $x = (x_1, x_2)$  and  $x_f = (x_{1f}, x_{2f})$ .

The minimum *time-to-go* from any state  $x$  to  $x_f$  can be computed analytically [59, 60]:

$$t_f(x, x_f) = \begin{cases} -x_2 - x_{2f} + 2\sqrt{-x_1 + x_{1f} + \frac{x_2^2}{2} + \frac{x_{2f}^2}{2}}, & \text{if } x \in \mathbf{R} \\ x_2 + x_{2f} + 2\sqrt{+x_1 - x_{1f} + \frac{x_2^2}{2} + \frac{x_{2f}^2}{2}}, & \text{otherwise} \end{cases} \quad (16)$$

where

$$\mathbf{R} = \{(x) \mid S_1(x) > 0, S_2(x) < 0\}, \quad (17)$$

and the switching curves  $S_1(x)$ ,  $S_2(x)$ , shown in Fig. 16, are:

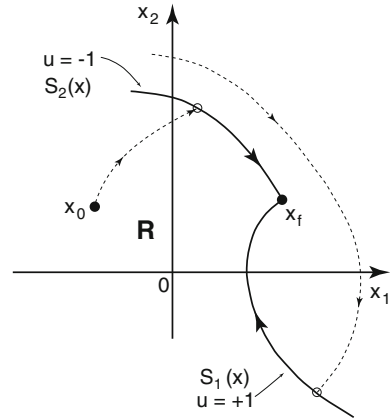
$$S_1(x) = x_2^2 - 2\left(x_1 - x_{1f} + \frac{x_{2f}^2}{2}\right) = 0, \quad (18)$$

$$S_2(x) = x_2^2 + 2\left(x_1 - x_{1f} - \frac{x_{2f}^2}{2}\right) = 0.$$

The switching time  $t_s$  is [60]:

$$t_s(x, x_f) = \begin{cases} -x_2 + \sqrt{-x_1 + x_{1f} + \frac{x_2^2}{2} + \frac{x_{2f}^2}{2}}, & \text{if } x \in \mathbf{R} \\ x_2 + \sqrt{+x_1 - x_{1f} + \frac{x_2^2}{2} + \frac{x_{2f}^2}{2}}, & \text{otherwise} \end{cases} \quad (19)$$

**Fig. 16** Switching curves in the state space of a single axis



Equation (16) computes the optimal *time-to-go* from any given state. It is used to determine the slowest axis of a multi axis system and set the motion time for the slowest axis, as discussed later.

The time-optimal trajectory thus first follows a parabola from the initial state to the switching curve, then follows the switching curve to the target state, as shown schematically in Fig. 16. Trajectories starting from initial states left of the switching curves (region **R** in Fig. 16) begin with  $u = 1$ , and right of the switching curves with  $u = -1$ . Trajectories starting from states on the switching curve follow the switching curve to the target with no switch. The switching time is determined by the initial and final states.

Since the minimum time trajectory has only one switch (excluding trajectories that emanate from initial states on the switching curves), reaching the target at a time greater than the minimum time,  $t_f$ , using bang-bang control, requires more than one switch [50].

For the two axis system (15), each axis may reach the target at a different optimal time. Obviously, the optimal time  $t_f$  to reach the target is determined by the slowest axis. Assuming, without loss of generality, that the faster axis from any initial state  $x_0 = (x_{10}, x_{20}, y_{10}, y_{20})$  to the target state  $x_f$  is the  $y$ -axis, the time-optimal trajectory is obtained by driving the  $x$ -axis optimally, and driving the  $y$ -axis so that it reaches the target at the same final time,  $t_f$ .

The trajectory of the  $x$ -axis is unique since it is optimal and hence has only one switch, whereas the trajectory of the  $y$ -axis is not optimal and hence has *at least* two switches.<sup>1</sup> It follows that the time-optimal path between the end points is not unique. The set of all time-optimal paths is bounded by two *extremal* paths, generated by the *extremal* trajectories, which are in turn generated by the extremal controls,  $u_{max}$  and  $u_{min}$  [50]:

$$u_{max}(t) = \begin{cases} 1 & \text{if } t \in [0, t_{s1}] \\ -1 & \text{if } t \in [t_{s1}, t_{s2}] \\ 1 & \text{if } t \in [t_{s2}, T] \end{cases} \quad (20)$$

$$u_{min}(t) = \begin{cases} -1 & \text{if } t \in [0, t_{s3}] \\ 1 & \text{if } t \in [t_{s3}, t_{s4}] \\ -1 & \text{if } t \in [t_{s4}, T] \end{cases} \quad (21)$$

where  $T > t_f$  is specified, and

$$\begin{aligned} t_{s1} &= \frac{1}{2\alpha} \left( x_{1f} - x_{10} + 2\alpha T - x_{20}T - \frac{T^2}{2} - \alpha^2 \right) \\ t_{s2} &= t_{s1} + \alpha \\ \alpha &= \frac{(T + x_{20} - x_{2f})}{2}, \end{aligned} \quad (22)$$

---

<sup>1</sup>The switching time of the slowest axis occurs when its trajectory reaches one of the switching curves given in (18).

$$\begin{aligned}
t_{s3} &= \frac{1}{2\beta} \left( x_{1f} - x_{10} - 2\beta T - x_{20}T + \frac{T^2}{2} + \beta^2 \right) \\
t_{s4} &= t_{s3} + \beta \\
\beta &= \frac{(T - x_{20} + x_{2f})}{2}.
\end{aligned} \tag{23}$$

We call the two-switch trajectories the *extremal* trajectories. Note that if the optimal motion times of both axes are identical, then the time-optimal trajectory is unique.

The unconstrained trajectory of system (14) from any state  $x = (x_1, x_2, y_1, y_2)$  to the target state  $x_f = (x_{1f}, x_{2f}, y_{1f}, y_{2f})$  is thus determined by the optimal motion time of the slowest axis. It can be used to drive the system as long as at least one extremal trajectory avoids the obstacle. Otherwise, the obstacle must be avoided using the *constrained* trajectory discussed next.

## 4.2 The Constrained Trajectory

The constrained trajectory is needed for points in the state-space from which *all* unconstrained time-optimal trajectories to the target intersect the obstacle. We refer to the set of such points as the Obstacle Shadow. In the kinematic case [51], the shadow corresponds to the shadow created behind the obstacle by a point light source at the target. The physical analogy for the dynamic problem is not as obvious.

The intersection of all the *extremal* time-optimal paths with the obstacle implies the intersection of all *unconstrained* optimal paths. It is therefore sufficient to check if both extremal trajectories intersect the obstacle to conclude that an avoiding trajectory, with optimal motion time greater than the optimal motion time of each axis, should be computed. Since the motion times of both axes are non-optimal, and hence greater than the unconstrained time  $t_f$ , it follows that both axes have at least two switches.

We compute the time optimal trajectory from  $x_0$  to  $x_f$  that avoids the obstacle, numerically, using a line search over the traveling time,  $t_c = t_f + \delta$ . The search terminates when the first trajectory that reaches the goal without intersecting the obstacle is found. The computation of the constrained trajectory for one obstacle is, thus, obtained by solving the following minimization problem over the single parameter,  $\delta$ :

$$t_c(x_0, x_f, OB) = \min_{\delta} t_f(x_0, x_f) + \delta \tag{24}$$

such that there exists  $j = 1, \dots, 4$  that satisfies:

$$x_{ex,j}(t) \notin OB, \tag{25}$$

where  $t_f(x_0, x_f)$  is the unconstrained optimal time (16), and  $x_{ex,j}(t)$ ,  $t \in [0, t_f + \delta]$  represents the  $j$ th extremal trajectory. The four extremal trajectories  $x_{ex,j}(t)$



correspond to the four combinations of the initial controls of both axes:  $(1, 1)$ ,  $(-1, -1)$ ,  $(1, -1)$ ,  $(-1, 1)$ . Although only two of these four trajectories are true extremals, it is simpler to test all four. It is sufficient that only one extremal satisfies (25).

### 4.3 Multi-obstacle Avoidance

The optimal avoidance of one obstacle is relatively simple, and is hence suitable for on-line computation. We use it to solve the multi-obstacle problem by avoiding obstacles *one at a time*. Key to this approach is the selection of the *current* obstacle to be avoided at any given time, as discussed next.

### 4.4 The Current Obstacle

We select the *current* obstacle as the *maximum cost* obstacle, which takes the longest time to avoid from the current state  $x$  to the goal  $x_f$ . Denoting  $t_c(x, x_f, OB(j))$ ,  $j = [1, m]$  as the minimum time it takes to avoid obstacle  $OB(j)$  from  $x$  to  $x_f$ , the *current* obstacle,  $k$ , maximizes  $t_c$ :

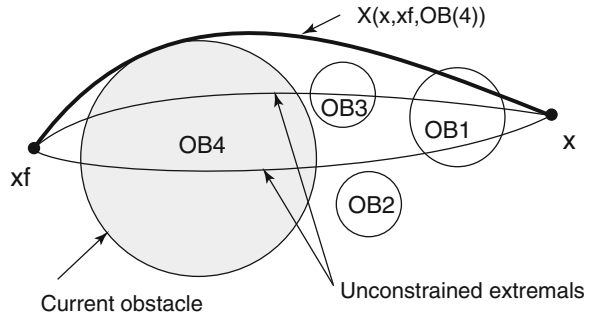
$$t_c(x, x_f, OB(k)) \geq t_c(x, x_f, OB(j)) \text{ for all } j = 1, \dots, m. \quad (26)$$

The *current* obstacle is thus selected by first determining all obstacles with shadows to the goal  $x_f$  containing the current state  $x$ , then computing the constrained trajectories avoiding each obstacle to  $x_f$ , and selecting the one with the longest motion time. If  $x$  does not lie in the shadow of any obstacle, then the cost of all obstacles equals to the unconstrained trajectory to the goal and none is selected to be avoided. One of the extremals of the set of unconstrained trajectories is then selected for navigation. The algorithm may switch between the extremals in case they collide with any obstacle, until either reaching the goal or entering the shadow of any obstacle, in which case the *current* obstacle is selected by (26).

Selecting at each step the obstacle with the highest cost to the goal produces a trajectory that is close to optimal, since the other obstacles have a smaller impact on the motion time to the goal, as is shown schematically in Fig. 17. In Fig. 17, the state  $x$  is in the shadows of obstacles 1 and 4. Of those, the trajectory avoiding  $OB(4)$ , denoted  $X(x, x_f, OB(4))$ , takes longer time than  $X(x, x_f, OB(1))$  (not shown). Hence  $OB(4)$  is selected as the *current* obstacle. Obviously, any solution to the goal must avoid obstacle 4. Hence, recognizing it early in the avoidance process increases the likelihood that the resulting trajectory will be close to optimal. The intersection of  $X(x, x_f, OB(4))$  with  $OB(1)$  will prompt a recursive process, discussed next.

While selecting the *maximum cost* obstacle is likely to result in near optimal trajectories, other selection criteria, such as the *nearest* obstacle (obstacle 1 in Fig. 17) may suffice for convergence.

**Fig. 17** Selecting the current obstacle from  $x$  to  $x_f$



## 4.5 The Avoidance Algorithm

The avoidance algorithm assumes convex and non-overlapping obstacles (in the configuration space). It selects the current obstacle to be avoided, computes the time optimal trajectory that avoids that obstacle, selects an intermediate goal along that trajectory on the boundary of that obstacle, and attempts to reach that goal. It repeats the process recursively until reaching the closest intermediate goal.

### Algorithm 2: Online Avoidance

**Step 0:** Initialize. Receive current state  $x$ , target state  $x_f$ ;

Set  $i = 0$ ,  $g(i) = x_f$ ;

**Step 1:** Determine the *current* obstacle,  $OB(k)$ , from  $x$  to  $g(i)$ .

If  $k = 0$  ( $x$  not in the shadow of any obstacle), go to Step 3.

Compute the optimal trajectory avoiding  $OB(k)$  to  $g(i)$ .

**Step 2:**  $i = i + 1$ ; Select an intermediate goal  $g(i)$  on the boundary of  $OB_k$  along the trajectory that avoids  $OB(k)$  to  $g(i - 1)$ .

Check that the velocity at  $g(i)$  is not in the *obstacle hole*<sup>2</sup> of any obstacle, consisting of *infeasible* states from which the obstacle is *unavoidable*. If it is, reduce speed at  $g(i)$  as needed.

Go to Step 1.

**Step 3:** Follow the optimal trajectory to  $g(i)$ . Set  $x = g(i)$ .

If  $i = 0$ , STOP.

$i = i - 1$

Go to Step 1.

Algorithm 2 generates a series of intermediate goals until one is reachable by a time optimal trajectory without colliding with any obstacle. Each intermediate goal  $g(i)$  ( $i \geq 1$ ) is selected along the constrained trajectory  $x_c(t)$  from the current state  $x$  to the current goal  $g(i - 1)$  at a point where  $x_c(t)$  is tangent to the current obstacle  $OB_k$ . Usually, there is just one such point. In case  $x_c(t)$  follows the obstacle for some

<sup>2</sup>The obstacle hole is a subset of the obstacle shadow.

distance, the point closest to the goal  $g(i - 1)$  is selected. When an intermediate goal is reached, a new avoidance problem is attempted from that intermediate goal to the next goal in the queue. Note that once an intermediate goal was reached, it is removed from the queue and a new goal may be assigned the same index  $i$ . The goals are added and removed from the queue while the trajectory gradually progresses to the final goal  $x_f = g(0)$ . Remembering the intermediate goals generated during the process is key to the convergence of this algorithm, as discussed later.

Step 2 of Algorithm 2 selects the speed at the intermediate goal  $g(i)$  that is both safe and feasible. A safe velocity is one that does not penetrate any obstacle hole, from which the obstacle is unavoidable. To simplify the search for the safe velocity, we choose to reduce it to the maximum velocity at which the robot can circle the current obstacle at its maximum lateral acceleration (the acceleration normal to its direction of motion). Denoting this velocity as the *curvature* velocity, it is easily proven that the *curvature* velocity does not lie in the obstacle hole of any obstacle.

**Definition 4.1** Curvature Velocity. The *curvature* velocity,  $v_c$ , is defined as:

$$v_c = \sqrt{u_{\max} R} \quad (27)$$

where  $u_{\max}$  is the maximum lateral acceleration, and  $R$  is the radius of the obstacle.

It remains to verify that the velocity at  $g(i)$  is reachable from the current state  $x$ . This is done by checking that a direct time optimal trajectory exists from  $x$  to  $g(i)$ . A direct trajectory is one that does not include loops. In case the velocity at  $g(i)$  is too high, we scale it down until it is reachable from  $x$ ; if the velocity at  $g(i)$  is too low, the current speed, which was set to the curvature velocity, can be reduced by circling the nearby obstacle at a decreasing speed. The curvature velocity (27) ensures that the obstacle can be circled to allow a safe reduction in speed when necessary. While this feature is necessary to ensure safety, it was not needed in any of the many cases tested by this algorithm.

The adjustment of speeds at the intermediate goals would ensure that any consecutive intermediate goals are connected by a feasible trajectory. This implies that a too high final velocity may be compromised for the sake of safety. Similarly, not every initial velocity is feasible for the obstacle avoidance case, even if it does not penetrate any individual obstacle hole. The speed reduction at the intermediate goals to the curvature velocity is a conservative measure to ensure safety.

## 4.6 Convergence

Convergence implies that the algorithm can reach the target state from an arbitrary feasible state, in a finite time. Since we cannot a priori determine the feasibility of arbitrary initial and target velocities, convergence of Algorithm 2 can be proven under

the assumption of zero terminal speeds (the speeds at the initial and final points), for convex obstacles that do not overlap with each other [50].

Algorithm 2 progresses incrementally towards the goal by moving through a sequence of intermediate goals. Every intermediate goal subdivides the trajectory to the goal into two smaller segments, and in fact breaks the avoidance problem into two smaller problems. Repeating this process recursively further reduces the avoidance problem until two consecutive intermediate goals are connected by an unconstrained trajectory. The motion time along each segment is finite since it is traversed at the minimum time. The number of such segments is bounded by the number of obstacles, which is assumed finite. It follows that the total travel time from start to goal is also finite, which proves convergence.

## 4.7 Optimality

The trajectory generated by Algorithm 2 is not necessarily optimal, since each step is only locally optimal. While the paths (the projection of the trajectory to the configuration space) generated by Algorithm 2 are generally close to the time optimal paths computed by a global planner [16], as demonstrated next, the motion time along the on-line trajectory is higher than the global optimal motion time due to the curvature velocity (27) imposed at the intermediate goals.

## 4.8 Numerical Examples and Experiments

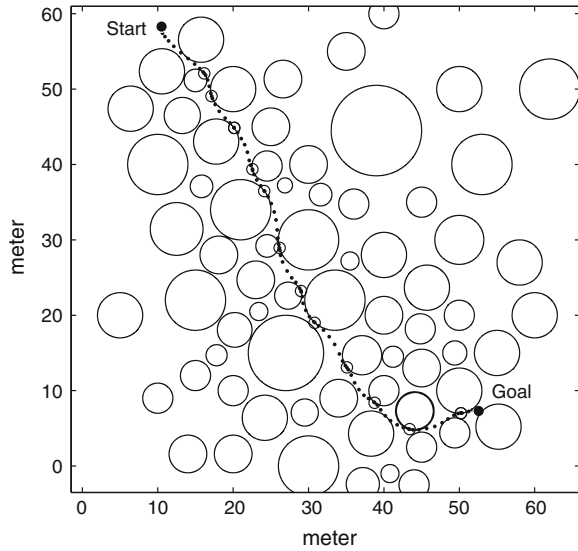
Algorithm 2 is demonstrated for a planar environment, consisting of 70 tightly spaced circular obstacles.

### 4.8.1 Example 2

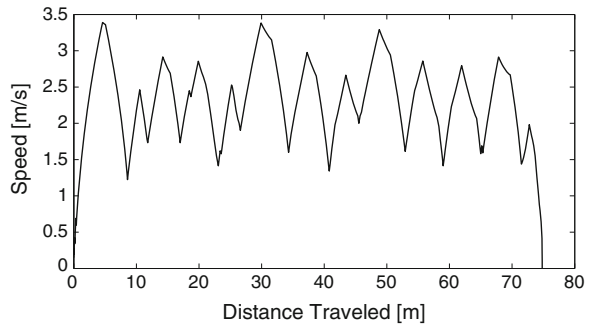
This example shows an on-line trajectory that avoids 70 obstacles, from the initial state  $(x_1, x_2, y_1, y_2) = (10.46 \text{ m}, 0.001 \text{ m/s}, 58.26 \text{ m}, 0.001 \text{ m/s})$  to the goal state  $(x_{1f}, x_{2f}, y_{1f}, y_{2f}) = (52.55 \text{ m}, 0 \text{ m/s}, 7.33 \text{ m}, 0 \text{ m/s})$ , as shown in Fig. 18. The spacing between the dots represents the speed along the path.

The motion time along this trajectory is 35.2 s, with a top speed of 3.4 m/s and an average speed of 2.1 m/s. There were 12 intermediate goals generated for this case, shown as empty circles along the trajectory. The total computation time was 4.3 s, with a time step  $\Delta t$  of 0.1 s, and an average computation time of 11 ms per-step. The speed along the trajectory, as a function of distance traveled, is shown in Fig. 19. The oscillations in the speed profile are due to the curvature velocity imposed at the intermediate goals.

**Fig. 18** Trajectory generated on-line in a tightly spaced environment with 70 circular obstacles for Example 2



**Fig. 19** Speed as a function of distance traveled along the online trajectory of Example 2

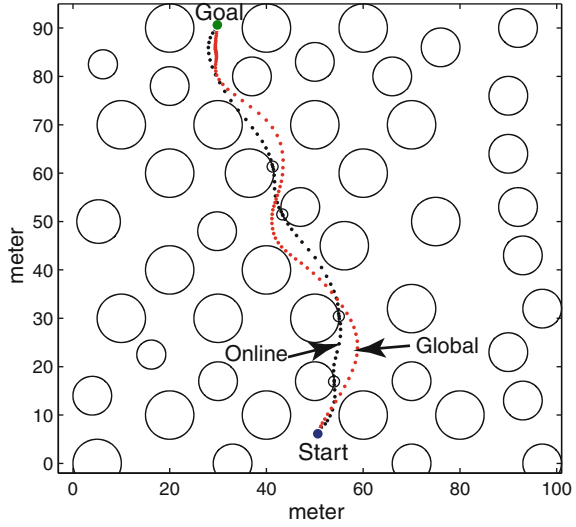


#### 4.8.2 Experiment–Global Optimality

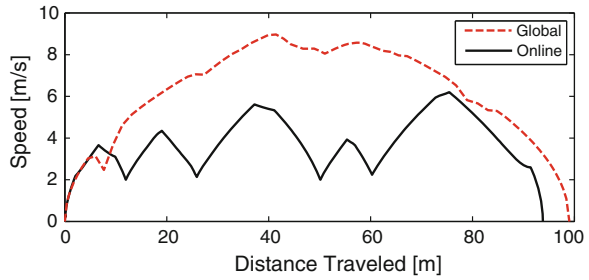
This experiment compares the online planner with the global planner [16] for the obstacle setup shown in Fig. 20 (48 obstacles).

Shown in Fig. 20 are the trajectories generated by the online planner and the global planner. The online and globally optimal paths have similar topologies as they pass between the same obstacles. The velocity profiles along both trajectories are shown in Fig. 21. The motion time along the online trajectory was 28.9s over a total distance of 93.8m, with an average speed of 3.2m/s, compared to the global optimal motion time of 20.7s over a total distance traveled of 99m, and an average speed of 4.8m/s. This difference is caused primarily by the reduction in speeds to the curvature velocities (27) at the intermediate goals.

**Fig. 20** The trajectories generated by the global and online planners, among 48 obstacles



**Fig. 21** Speed as a function of distance traveled for the online and global optimal trajectories



Repeating this test for 50 randomly selected end points yielded similar results, with the average motion time of the online trajectories being 30.52 s, compared to the average optimal time of 22.63 s. The average path length of the online trajectories was 111.51 m, compared with 115.26 m for the optimal trajectories. Here too, the increase in the motion time despite the comparable path lengths is due to the imposed curvature velocity at the intermediate goals, which is determined by the obstacle size.

#### 4.9 Computational Issues

The consideration of the obstacles one at a time reduces the original problem with  $m$  obstacles to  $m$  simpler sub-problems with one obstacle each.

The cost for this reduction is the loss of optimality, and the need to check at each time step if *all* obstacles intersect the unconstrained optimal path from the current state, and for those that do, solve the single obstacle problem. This may

seem excessive, but the alternative (solving the original exponential problem) is much worse. Our approach generates the trajectory incrementally, unlike the original problem that requires a complete solution before making the first move. In fact, for problems with many obstacles, such as in example 2 with 70 obstacles presented earlier, the on-line (heuristic) solution may be the only viable alternative.

Practically, it may not be necessary to consider all obstacles at all times, but instead consider only the obstacles within some radius of visibility around the robot. It would be then necessary to limit robot's speed to the stopping speed at the boundary of its visibility range to ensure that it does not collide with an unforeseen obstacle.

To appreciate the computational advantage of this approach, we attempted to compare it to the performance of efficient state-of-the-art algorithms. Currently, the most popular approach is the RRT planner, which rapidly explores a random tree to produce the first feasible solution to the goal [17–19]. The solution found is not optimal in any way, and this class of algorithms is known to have inherent difficulties with tight spaces. Yet, RRT is currently considered as the fastest algorithm to connect between two points through cluttered environments.

We compared a kinematic version of our online algorithm to the RRT and RRT\* planners for avoiding 70 tightly space obstacles, all running on similar computers [50]. Testing the algorithms for 100 randomly generated end points, the run time of the online algorithm was on average 0.5 ms, compared to 3.5 ms of the RRT planner, 7 times faster. However, the path lengths produced by the RRT planner were twice as long as those produced by the online planner, which were near global optimum. Attempting to optimize the paths using the RRT\* planner took 0.5 s to reach the optimality levels of the online planner; this is 1,000 times slower than the online algorithm. These results demonstrate the sound efficiency, in both computation time and optimality, of the online planner presented here. This is not surprising as the online planner consistently executes locally optimal paths at each incremental step, as opposed to the sampling-based planners which essentially search for a solution in the dark.

## 5 Summary

Motion planning is one of the basic problems in robotics as very few robotic tasks do not involve motion. The main challenge in motion planning is to produce a trajectory that safely and efficiently moves the robot from one state to another while accounting for its dynamic behavior. It is also desirable that the motion plan reflects the changing nature of the task or of the environment. While this is obviously the ultimate goal, early works on motion planning in the 80s settled for much less by focusing on geometric path planning with no account for robot dynamics. The resulting algorithms were useful for determining the shortest path from start to goal, but were useless for moving the robot at other than very low speeds. To account for robot dynamics, optimal control theory, developed in the late 60s, was applied then to robotics but failed because of insufficient computation power and the high sensitivity of the numerical

solutions to the initial guess. Like in many other endeavors, the solution emerged by solving a simpler problem.

The failure of the geometric algorithms to solve high dimensional problems gave rise to a class of sampling-based planners, with the goal of producing any feasible path in lieu of the shortest path expected by earlier work. The multi-dimensional optimal trajectory planning problem was eventually solved by first computing the optimal velocity profile along a given path. This led to a local optimization of the path and eventually to a global planner that computes “good” initial guesses for the local optimization.

In this chapter, we reviewed the main approaches to off-line and on-line motion planning, and presented one solution for each with a focus on trajectory planning. It was shown that any motion planning problem can be theoretically solved using the Hamilton Jacobi Bellman (HJB) equation. If the return function is known or approximated, this approach offers an online solution. In its discrete form, the HJB equation leads to dynamic programming, which is the basis for the combinatorial optimizations used in off-line planning.

We presented an off-line planner that takes advantage of the efficient computation of the optimal motion time along any path. The on-line planner presented converts the original problem of optimally avoiding many obstacles to many simpler problems, each avoiding optimally only one obstacle. The high correlation between the solutions of the on-line and off-line planners is not surprising since both planners are based on sound optimal control theories.

As the basic problem of trajectory planning is considered solved, and as computers are becoming more powerful, the remaining challenge rests with online planning that adapts or reacts to the changing nature of real life scenarios in the industry, in the home, and on the road.

## References

1. Choset H, Lynch KM, Hutchinson S, Kantor GA, Burgard W, Kavraki LE, Thrun S (2005) Principles of robot motion: theory, algorithms, and implementations. MIT Press, Cambridge
2. Shiller Z, Dubowsky S (1991) On computing the global time optimal motions of robotic manipulators in the presence of obstacles. *IEEE Trans Robot Autom* 7(6):785–797
3. Canny JF (1988) The complexity of robot motion planning. MIT Press, Cambridge
4. Lozano-Perez T, Wesley MA (1979) An algorithm for planning collision-free paths among polyhedral obstacles. *Commun ACM* 22(10):560–570
5. Lumelsky VJ, Stepanov A (1987) Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica* 2:403–430
6. Schwartz JT, Sharir M (1983) On the piano movers’ problem: the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun Pure Appl Math* 36:345–398
7. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *Int J Robot Res* 30(7):846–894
8. Manor G, Rimon E (2013) Vc-method: high-speed navigation of a uniformly braking mobile robot using position-velocity configuration space. *Auton Robot* 34(4):295–309
9. Lozano-Perez T (1987) A simple motion planning algorithm for robotic manipulators. *IEEE Trans Robot Autom* RA-3(3):224–238



10. Wein R, van den Berg JP, Halperin D (2005) The visibility-Voronoi complex and its applications. In: Proceedings of 21st symposium on computational geometry, pp 63–72
11. Alexopolous C, Griffin PM (1992) Path planning for a mobile robot. *IEEE Trans Syst Man Cybern* 22(2):318–322
12. Liu YH, Arimoto S (1992) Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *Int J Robot Res* 11(4):376–382
13. LaValle SM (2010) Motion planning: the essentials. *IEEE Robot Autom Mag* 110
14. Dijkstra EW (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271
15. Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization, algorithms and complexity. Prentice-Hall, Englewood Cliffs
16. Shiller Z, Gwo YR (1991) Dynamic motion planning of autonomous vehicles. *IEEE Trans Robot Autom* 7(2):241–249
17. Lavalle SM (1998) Rapidly-exploring random trees: a new tool for path planning. Technical Report 98-11, Department of CS, Iowa State University
18. Hsu D, Latombe JC, Motwani R (1999) Path planning in expansive configuration spaces. *Int J Comput Geom Appl* 4:495–512
19. Mazer E, Ahuactzin JM, Bessiere P (1998) The Ariadne's clew algorithm. *J Artif Intell* 9:295–316
20. Karaman S, Walter MR, Perez A, Frazzoli E, Teller S (2011) Anytime motion planning using the *rrt\**. In: International conference on robotics and automation
21. Amato NM, Bayazit OB, Dale LK (2000) Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans Robot Autom* 16(4):442–447
22. Gmez-Bravo F, Carbone G, Fortes JC (2012) Collision free trajectory planning for hybrid manipulators. *Mechatronics* 22(6):836–851. Special Issue on Intelligent Mechatronics
23. Bryson AE, Ho YC (1969) Applied optimal control. Blaisdell Publishing Company, Cambridge
24. Kiriazov P, Marinov P (1985) A method for time-optimal control of dynamically constrained manipulator. *Theory and practice of robotics and manipulators*. MIT Press, Cambridge, pp 169–178
25. Niv M, Auslander DM (1984) Optimal control of a robot with obstacles. In: Proceedings of American control conference (San Diego, CA), June 1984, pp 280–287
26. Khan ME, Roth B (1971) The near-minimum time control of open loop articulated kinematic chains. *J Dyn Syst Meas Control* 93(3):164–172
27. Bobrow JE, Dubowsky S, Gibson JS (1985) Time-optimal control of robotic manipulators. *IJRR* 4(3):3–17
28. Pfeiffer F, Johanni R (1987) A concept for manipulator trajectory planning. *IEEE Trans Robot Autom* RA-3(3):115–123
29. Shin KG, McKay ND (1985) Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans Autom Control* AC-30(6):531–541
30. Shiller Z, Lu HH (1992) Computation of path constrained time-optimal motions with dynamic singularities. *ASME J Dyn Syst Meas Control* 114(1):34–40
31. Slotine JE, Yang HS (1989) Improving the efficiency of time optimal path following algorithms. *IEEE Trans Robot Autom* 5(1):118–124
32. Tarkkainen M, Shiller Z (1993) Time optimal motions of manipulators with actuator dynamics. In: Proceedings of 1993 IEEE international conference on robotics and automation, vol 2, pp 725–730
33. Bobrow JE (1988) Optimal robot path planning using the minimum time criterion. *IEEE Trans Robot Autom* 4(4):443–450
34. Shiller Z, Dubowsky S (1989) Time-optimal path-planning for robotic manipulators with obstacles, actuator, gripper and payload constraints. *IJRR* 8(6):3–18
35. Seywald H (1994) Trajectory optimization based on differential inclusion. *J Guid, Control, Dyn* 17(3):480–487
36. Bryson AE (1999) Dynamic optimization. Addison Wesley, New York

37. Donald B, Xavier P (1989) A provably good approximation algorithm for optimal-time trajectory planning. In: Proceedings of IEEE conference on robotics and automation, May 1989, pp 958–963
38. Sahar G, Hollerbach JM (1985) Planning of minimum-time trajectories for robot arms. In: Proceedings of IEEE international conference on robotics and automation (St. Louis, MO), March 1985, pp 751–758
39. Kamon I, Rimon E, Rivlin E (1998) Tangentbug: a range-sensor based navigation algorithm. *Int J Robot Res* 17(9):934–953
40. Laubach S, Burdick J, Matthies L (1998) A practical autonomous path-planner for the rocky7 prototype microrover. IEEE international conference on robotics and automation
41. Choset H, Burdick JW (1995) Sensor based planning, part ii: incremental construction of the generalized Voronoi graph. In: Proceedings of IEEE international conference on robotics and automation, ICRA'95, pp 1643–1649
42. Sankaranarayanan A, Vidyasagar M (1991) Path planning for moving a point object amidst unknown obstacles in a plane: the universal lower bound on worst case path lengths and a classification of algorithms. In: Proceedings of IEEE international conference on robotics and automation, ICRA'91, pp 1734–1941
43. Connolly CI, Burns JB, Weiss R (1991) Path planning using Laplace's equation. In: IEEE conference on robotics and automation, Cincinnati, OH, vol 1, pp 102–2106
44. Khatib O (1986) Real time obstacle avoidance for manipulators and mobile robots. *Int J Robot Res* 1:65–78
45. Rimon E, Koditschek DE (1992) Exact robot navigation using artificial potential functions. *IEEE Trans Robot Autom* 8:501–518
46. Jarvis R (1985) Collision-free trajectory planning using distance transforms. *Trans Inst Eng Aust Mech Eng ME10(3)*:187–191
47. Athans M (1965) Optimal control: an introduction to the theory and its applications. Academic Press, New York
48. Cesari L (1983) Optimization—theory and applications: problems with ordinary differential equations. Springer, New York
49. Moskalenko AI (1967) Bellman equations for optimal processes with constraints on the phase coordinates. *Autom Remote Control* 4:1853–1864
50. Shiller Z, Sharma S, Stern I, Stern A (2013) On-line obstacle avoidance at high speeds. *Int J Robot Res* 32(9–10):1030–1047
51. Sundar S, Shiller Z (1997) Optimal obstacle avoidance based on sufficient conditions of optimality. *IEEE Trans Robot Autom* 13(2):305–310
52. Lee EB, Markus L (1967) Foundations of optimal control theory. Wiley, New York
53. Koditschek DE, Rimon E (1990) Robot navigation functions on manifolds with boundary. *Adv Appl Math* 11:412–442
54. Bellman R (1957) Dynamic programming. Princeton University Press, Princeton
55. Lawler EL (1976) Combinatorial optimization. Holt, Rinehart and Winston, New York
56. Fujita Y, Nakamura Y, Shiller Z (2003) Dual dijkstra search for paths with different topologies. In: ICRA, pp 3359–3364
57. Shiller Z, Fujita Y, Ophir D, Nakamura Y (2004) Computing a set of local optimal paths through cluttered environments and over open terrain. In: ICRA, pp 4759–4764
58. Pham QC (2013) Characterizing and addressing dynamic singularities in the time-optimal path parameterization algorithm. In: 2013 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 2357–2363
59. Dreyfus S (1965) Dynamic programming and the calculus of variations. Academic Press, New York
60. Sundar S (1995) Time-optimal obstacle avoidance for robotic manipulators. Doctoral Dissertation, Mechanical and Aerospace Engineering, University of California, Los Angeles, June 1995

**Motion and Operation Planning of Robotic Systems**

**Background and Practical Approaches**

Carbone, G.; Gomez-Bravo, F. (Eds.)

2015, IX, 522 p. 352 illus., 256 illus. in color., Hardcover

ISBN: 978-3-319-14704-8