UNIVERSITÀ DI PARMA

DEPARTMENT OF ENGINEERING AND ARCHITECTURE

PhD in Information Technologies

XXX Cycle

PATH PLANNING FOR ROAD VEHICLES BY DYNAMIC PROGRAMMING

Coordinator:
Prof. Marco Locatelli

Advisor:
Prof. Luca Consolini

PhD student: *Piero Micelli*

Years 2014-2017

Contents

List of Figures

List of Tables

List of Algorithms

Basic notations

$\mathbb{R}$	set of real numbers.
$\mathbb{R}_+$	set of nonnegative real numbers.
$\mathbb{N}$	set of positive integers.
$\mathbb{R}^n$	the euclidean n-dimensional space.
$\text{int}(E)$	the interior of the set E.
$\text{dist}(x,y)$	given a metric space E and $x,y \in E$, $\text{dist}(x,y)$ denotes the distance between x and y.
$\|\cdot\|$	the Euclidean norm of $\mathbb{R}^n$ (i.e, $\|x\| = (x \cdot x)^{1/2}$).
$\text{diam}(E)$	the diameter of the set E (i.e., $\text{diam}\,(E) = \sup\{\|x-y\| : x,y \in E\}$).
$L^\infty(E)$	the space of limited values functions $f : E \to \mathbb{R}$.
E^i	the i-th Cartesian power of E (i.e., $E^i = E \times E \ldots \times E$, i times).

$E^\star$	denotes the set $\bigcup_{i=1}^{+\infty} E^i$.
$\sigma(A)$	the spectrum of the matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, that is, the set of its eigenvalues.
$\rho(A)$	the spectral radius of A, $\rho(A) := \max\{\lvert\lambda\rvert : \lambda \in \sigma(A)\}$.
$\mathrm{sp}(A)$	the number of nonzero elements of A.
$\langle\cdot,\cdot\rangle$	the standard inner product on $\mathbb{R}^n$.
$\lVert A\rVert$	the Euclidean norm of matrix A.
$D(u)(x)$	the gradient of the function u at x, i.e., $D(u)(x) = \left(\frac{du(x)}{dx_1},\dots,\frac{du(x)}{dx_n}\right)$, $x \in \mathbb{R}^n$.

Introduction

In robotics, motion planning has been intensively researched in the last twenty years. Dedicated algorithms have been developed for different applications, from industrial machines to human-like robots. An interesting research area within this field is motion planning for Automatic guided vehicles (AGVs). AGVs have became increasingly important in different domains. In industry and agriculture, these robots are used to save human resources, being employed in fertilizing as well as to automatically move materials. In the automotive domain, driverless cars are able to navigate in urban environments in presence of traffic lights, road signs, and traffic participants. Currently, many prototypes are evolving, but no cars permitted on public roads are fully autonomous. They all require a human driver who must be ready to take control at any time. This is because this technology is still under development or too expensive for practical use and ethical issues must be solved before autonomous cars are ready for market.

Advanced driver assistance systems (ADAS) are one of the fastest-growing segment in automotive industry. These systems are developed to automate and improve vehicle systems for safety and better driving. In particular, according to analysts, autonomous valet parking systems will play an important role in next years. The main benefits for such systems include enhancing the comfort of the driver, reducing stress related to finding a free place and performing a, possibly complex, parking maneuver. They also decrease the overall travel time. They increase safety, preventing the vehicle from collisions with fixed obstacles, other vehicles or pedestrians. They may allow a better usage of available space in parking areas, reducing the fuel needed for

the parking operation. The implementation of an automatic parking system involves several disciplines from computer vision (to perceive the environment in which the vehicle moves), to control theory (to plan the trajectory to perform the task).
The aim of this thesis is to develop a path planning algorithm which is able to find a feasible path between two arbitrary vehicle configurations in order to compute a parking maneuver. In particular, the path planning problem is addressed using two different methods: Dynamic Programming and Search-based Planning. The first method, based on the numerical solution of the Hamilton-Jacobi-Bellman equation, allows finding an optimal solution at the expense of a high computational cost. On the other hand, Search-based Planning converts the path planning problem into a minimum path problem on a graph, and allows finding a solution to a planning task rather quickly, even for large and high-dimensional operating spaces. In this thesis, Dynamic Programming is first used to find an optimal solution for a small operating space. Afterwards, this approach is integrated with Search-based Planning in order to obtain a sub-optimal solution for large environments.

The thesis is structured as follows. Chapter 1 describes the problem of motion planning and presents previous works on path planning for car-like robots. In addition, this chapter gives a brief introduction of Search-based Planning and Dynamic Programming approaches. Here, it is shown that, using Search-based Planning, a motion planning task can be converted into a path finding problem on a graph, while Dynamic Programming methods convert the same problem into a fixed-point iteration.

One problem related to the Dynamic Programming approach is that the convergence speed of the fixed-point iteration is rather poor. Chapter 2 proposes a Jacobi-like acceleration that allows improving the convergence speed. The acceleration technique is implemented to solve a minimum-time maneuver problem in absence of obstacles.

Chapters 3,4 present two path planning algorithms for computing optimal parking maneuvers for road vehicles, operating in a known environment in the presence of static obstacles. Both algorithms are based on the numerical solution of the Hamilton-

Jacobi-Bellman equation. The first algorithm allows finding the minimum length trajectory with a bounded number of direction changes. In the second one, the vehicle is described by a switched system, composed of 6 autonomous systems, such that its motion can be represented as a concatenation of lines and arc segments of constant curvature. Here, the algorithm finds the trajectory that minimizes a cost function that takes into account the length of the parking maneuver and a penalty related to the number of switchings. The choice of this penalty allows taking into account the number of changes of direction and steering angle in the selection of the parking maneuver.

Both Optimal Control and Search-based Planning are used extensively for path planning and have their own set of advantages and disadvantages. Chapter 5 presents the algorithm FOCS (Fusion of Optimal Control and Search) that combines these two approaches. FOCS finds a path exploiting the advantages of both approaches while providing a bound on the sub-optimality of its solution. The returned path is the concatenation of the path found by graph-based search and the path generated by following the negative gradient of the value function obtained as solution of the Hamilton-Jacobi-Bellman equation. This chapter analyzes the algorithm and illustrates its effectiveness in finding a minimum-time path for a car-like vehicle in different environments.

Statement of contribution

This thesis leverages some recent results on Dynamic Programming in order to develop a determinist path planning algorithm for car-like vehicles. The main novel contributions of this work are listed below.

1. The dynamic programming method is based on the solution of the HJB equation. Since the HJB equation is a nonlinear partial differential equation, a closed form solution does not exist for the general case. Some recent approaches, after some simplifications, convert the problem into a fixed-point iteration. One problem related to the iteration is that its convergence speed is rather poor. As a first contribution, this thesis describes a Jacobi-like acceleration that allows improving the convergence speed.

2. This thesis formulates the motion planning for car-like vehicles as an optimization problem for a switching system. This approach leads to the development of two new algorithms.

   (a) The first algorithm allows finding the minimum length path with a number of direction changes bounded by an assigned constant.

   (b) The second algorithm characterizes the vehicle as a switched system, composed of 6 autonomous systems, such that its motion is represented as a concatenation of lines and arc segments of constant curvature. This algorithm finds the trajectory that minimizes a cost function that takes into account the length of the parking maneuver and a penalty related the number of switchings. The choice of this penalty allows taking into account the number of changes of direction and steering angle in the selection of the parking maneuver.

3. Optimal Control and Search-based Planning are combined in a single algorithm that exploits the advantages of both approaches while providing a bound on the sub-optimality of its solution. The algorithm is called FOCS (Fusion of Optimal Control and Search). It returns a solution as a concatenation of the path found by graph-based search and the path generated by following the negative gradient of the value function obtained as the solution of the Hamilton-Jacobi-Bellman equation. This approach allows finding a solution for large environments, even when the operating space around the target configuration is tight, exploiting the performances of search-based algorithms and the accuracy of the HJB equation.

Chapter 1

Motion planning: main concepts and algorithms

A motion planning algorithm produces a continuos motion from the current configuration of the robot to its goal configuration (or to a set of goal configurations) such that:

- it satisfies the environmental (e.g., avoid the obstacles) and the kinematics constraints of the robot,
- it minimizes some cost function, such as the length of the path to reach the goal configuration (or the time, the energy, some risk, etc.).

A configuration describes the pose of the robot and the configuration space is the set of all configurations. The set of configurations that satisfies the environment constraints is called free space and its complement obstacle space. The subspace of the free space which represents the set of goal configurations is called target space.
In this thesis, the vehicle is described by the following kinematic car-like model with

rear-wheel drive (see Figure 1.1):

$$\begin{cases} \dot{z}(t) = v\cos\theta(t) \\ \dot{y}(t) = v\sin\theta(t) \\ \dot{\theta}(t) = \omega(t), \end{cases} \tag{1.1}$$

where (z, y) represents the position of the center of the real wheel axle, θ the orientation angle and parameter l is the distance between the centers of the front and rear wheel axels. In this way, a configuration of the vehicle is represented by the triplet $x = (z, y, \theta)$. The control input is given by $u = (v, \omega)$, where v and ω are, respectively, the linear and angular velocities. The angular velocity is related to the front

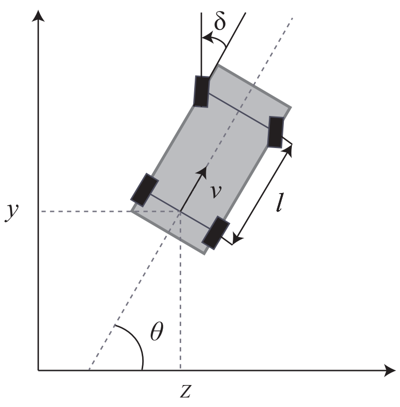


Figure 1.1: Car-like model.

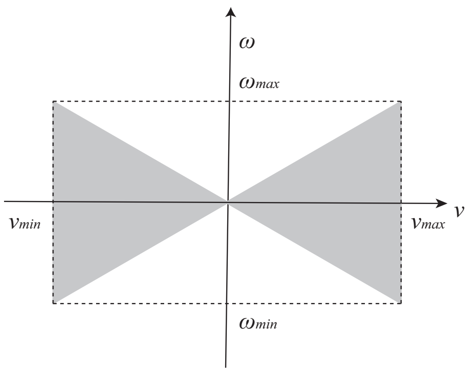


Figure 1.2: Set of admissible controls (shaded region).

wheel steering angle δ by relation $\omega = \frac{1}{l} v \tan\delta$. The input variables are constrained as follows

$$v_{min} < v < v_{max}, \qquad \omega_{min} < \omega < \omega_{max}. \tag{1.2}$$

Figure 1.2 represents the resulting set of admissible controls for v and ω. Considering the maneuvering scenario of Figure 1.3, the motion planning problem consists in finding a control signal $u = (v, \omega)$ able to lead the vehicle to a target configuration. In particular, the planned trajectory is subject to a starting and target vehicle configuration, to obstacle avoidance and kinematic constraints (1.2). The basic model

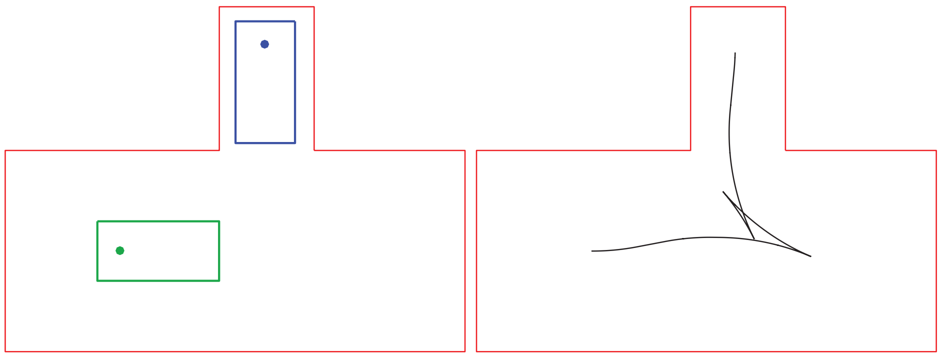


Figure 1.3: A perpendicular maneuvering scenario with obstacles whose edges are represented with red segments. On the left, starting and target state are represented with green and respectively, blue rectangles. The figure on the right shows a planned trajectory that allows reaching the target state.

(1.1) only considers the kinematic properties of the vehicle. However for a parking maneuver the speed of the vehicle is low enough such that the dynamic behavior is not relevant. For this reason, this motion planning problem can be decomposed into two subproblems using the Path-Velocity-Decomposition approach [1]: planning a collision-free path which satisfies kinematic constraints (1.2) and planning the velocity along the path. In this thesis we will focus on the path planning problem.

1.1 Relevant literature for path planning for car-like robots

Early works on path-planning for system (1.1) study the problem of connecting two arbitrary configurations by a shortest path. In reference [2], Dubin provides a method for connecting two arbitrary configurations. There, it is shown that the shortest path between two configurations is composed of a concatenation of at most three lines or arc segments of constant curvature. Note that the shortest path has a continuos tangent but not a continuos curvature. Moreover, it is assumed that the space is unbounded, without obstacles and only forward motion is considered. The results have been ex-

tended in [3], where backward motion is also allowed. In more recent years, due to the developments in automotive industry, path planning problems for system (1.1) have received great consideration in literature and have been addressed with different approaches.
Some works are based on geometrical considerations tailored to specific parking configurations. For instance, [4] describes a simple algorithm for parallel parking for the Ackerman steering configuration. Paper [5] uses a probabilistic roadmap to create a graph of randomly generated collision-free configurations that is used by a local path planner. These strategies do not provide systematic and deterministic methods to solve motion planning problems, but provide anyway practically usable maneuvers. More recently, the Defense Advanced Research Projects Agency (DARPA) has organized several DARPA Urban Challenges that originated significant new ideas. For instance, [6] and [7] use an incremental search on a multiresolution lattice state space to generate dynamically-feasible maneuvers. Work [8] uses an extended RRT algorithm to find a motion compatible with the vehicle dynamical constraints. A similar approach has been used in [9].
A different approach is represented by the numerical solution of the Hamilton-Jacobi-Bellman (HBJ) equation. This method allows finding a deterministic solution to the motion planning problem, at the expense of a greater computational cost. For instance, this approach is used in [10] that considers the problem of optimal path planning for a Dubins'car using a finite difference numerical approximation of the HJB equation.

1.2 Deterministic Path Planning in Robotics

Different approaches have been used in literature for Path Planning with their own set of advantages and disadvantages. In this chapter, we focus on the deterministic approaches of Search-based Planning and Optimal Control. In contrast to non determinist approaches (such as probabilistic algorithms) that do not always produce a feasible path when one exists, this class of algorithms either produces a solution in a finite time or correctly reports that there is none.

1.2.1 Search-based Planning

Search-based algorithms convert a motion planning task into a path finding problem on a graph. To this end, the configuration space of the robot is discretized and each vertex in the graph corresponds to one of the discretized robot configurations. The motion of a robot is decomposed into a small set of short motion primitives that constitute the edges of the graph. Heuristic search algorithms such as A* can then be used to search this graph for an optimal or close-to-optimal path from the vertex that corresponds to the current robot configuration to the vertex that corresponds to its goal configuration. This approach allows finding a solution to a planning task rather quickly, even for large and high-dimensional operating spaces by utilizing heuristic search algorithms that provide real-time performance combined with rigorous sub-optimality bounds with respect to the chosen discretization. Search-based planning can then be decomposed in two problems: turning the problem into a graph, and searching the graph for finding the optimal solution.

Environment representation

In order to properly represent the problem with a graph, we first need to discretize the configuration space in a finite set of states. The choice of the state variables varies depending on the application. For instance, for a car, one can represent a state with a position vector (x,y) and an orientation angle. For a helicopter, the state is represented by the position vector (x, y, z) and, roll, pitch, and yaw angles. In a robotic arm, it is represented by the positions of each joint in the arm. The set of all possible states is defined as the state space. For a car, if we have a 100×100 grid of positions and 6 possible angles for each position, we would have a state space of size $100 \times 100 \times 6 = 60000$.

Figure 1.4 shows a small portion of a 2D state space where the robot configurations are represented as a node with a state ID (S_1, S_2, ...). In this 2D state space, for some robot, a good strategy could be to define a path as a series of adjacent states, for example $S_2 - S_4 - S_5$. But, in general, adjacent states do not respect the kinodynamic constraints of the robot. For instance, consider the case in which the robot is a car facing north at x, y location (2,2). Using this strategy a path could contain sequence

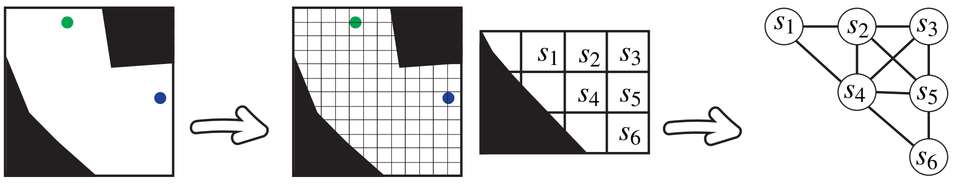


Figure 1.4: From left to right: 2D environment, small portion of the state space for the 2D environment, part of the graph constructed with adjacent states.

(2,2)-(1,2) and, while these two states are adjacent, the car cannot move sideways to achieve this transition. For this reason, Search-based Planning uses lattice-based graphs [7], [11] that are constructed using motion primitives. Figure 1.5 shows the motion primitives, for the car example with forward motion only, when the car is facing right. In this case, adjacent states (or what are usually called successors) are

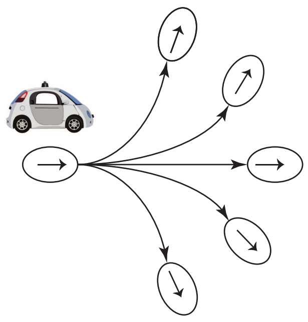


Figure 1.5: Example of motion primitives for a car with forward motion only.

not necessarily spatially adjacent. The valid successors of a given state are the states that can be reached with these motions while not colliding with obstacles (see Figure 1.6). In this way, motion primitives allow encoding the kinematic constraints of the robot into any environment. They also allow assigning different costs to each motion. For instance, if we prefer forward motions for a car, we might assign a high cost to backward motion primitives. A significant feature of a lattice-based graph is that the conversion of the motion planning problem to a minimum path problem into a graph

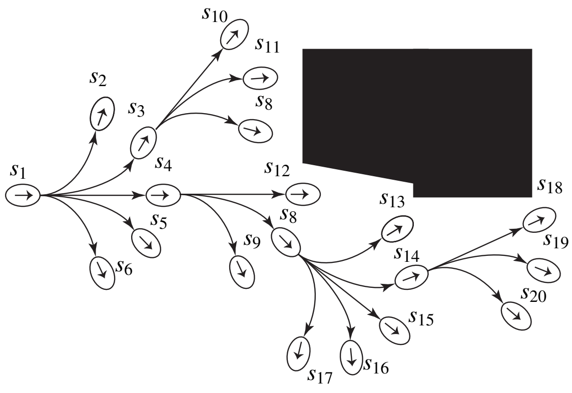


Figure 1.6: Small portion of a lattice-based graph constructed from the motion primitives of Figure 1.5.

is completely decoupled from how the problem is solved. Since each state is encoded into a single identifying number (state IDs) the graph searching algorithms do not need to know what the original domain was. This means that the algorithms can operate on any problem that can be represented as a graph (see Figure 1.7).

Planning

Suppose that the task of motion planning is converted into a path planning problem defined on an lattice-based graph. In this graph the nodes represent a finite set of states S. The edges are the connections between these nodes. The cost of the edge between nodes s and s' is denoted as $c(s,s')$, where $c(s,s') = \infty$ if there are no edges between s and s'. Therefore, $SUCCESSOR(s) := \{s' \in S \mid c(s,s') \neq \infty\}$, represents all successors of s and $c^*(s,s')$ denotes the cost of the optimal path from state s to s'. In this graph, path $\Pi : s_{start} \rightarrow s_{goal}$ is a concatenation of edges that connects starting state s_{start} to goal state s_{goal}.

The simplest approaches to search a lattice-based graph are Dijkstra's and A* algorithms, which return the cost-minimal path between a given start and goal state. However, these algorithms can be slow and memory expensive. Moreover, in most cases, we do not need to find the optimal path, but one that is good enough. One

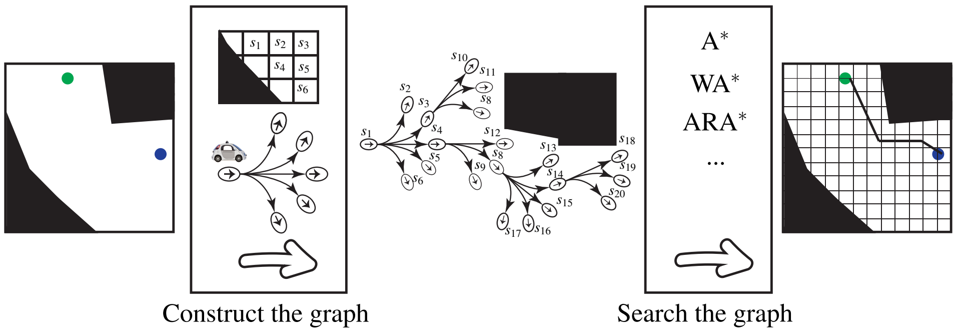


Figure 1.7: Principals steps of a search-based algorithm: construction of the graph from state space and motion primitives, searching of the graph for the optimal solution.

algorithm that does this is weighted A* (WA*).

WA* [12] is a variant of A* with inflated heuristics, meaning the heuristic values are multiplied by an inflation factor $\eta > 1$ (for $\eta = 1$ WA* is equal to A*). The inflation factor η provides a greedy flavor to the search, which finds a solution in less time at the cost of optimality ([13], [14], [15]). At each iteration of the main loop (lines 6-16 of Algorithm 1), WA* selects the path that minimizes the function $f : S \rightarrow \mathbb{R}$:

$$f(s) = g(s) + \eta h(s),$$

where s is the last node on the path, $g(s)$ denotes the current cost of the best path from s_{start} to s, and $h(s)$ is the heuristic for state s, which is an estimate of the cost of the path from s to s_{goal}. In this way, function f is an estimate of the total cost to travel from s_{start} to s_{goal} going through s. WA* maintains a priority queue, *OPEN*, of states which it plans to expand and a list, *CLOSED*, for keeping track of the expanded states. The *OPEN* queue is sorted by f, so that WA* always expands next the state which appears to be on the shortest path from s_{start} to s_{goal}. WA* initializes the *OPEN* list with the start state s_{start} (line 4). Each time it expands the state that minimizes function f (line 14), ending when goal state s_{goal} is expanded (line 6).

Usually, A* and WA* take as input a heuristic $h(s)$ that is admissible and consistent.

Algorithm 1 WA*

```
    **Input**: $s_{goal}$: goal state, $s_{start}$: starting state, $\eta > 1$: inflation factor.
    **Output**: $\Pi^*(\eta) : s_{start} \rightarrow s_{goal}$: sub-optimal path.
 1: procedure MAIN( )
 2:     $g(s_{goal}) = \infty \; g(s_{start}) = 0$
 3:     $OPEN = CLOSED = \varnothing$
 4:     insert $s_{start}$ into $OPEN$ with $f(s_{start}) = \eta h(s_{start})$
 5:     $s^* = s_{start}$
 6:     while $s^* \neq s_{goal}$ do
 7:         for each $s' \in SUCCESSOR(s^*)$ do
 8:             if $s'$ was not visited before then
 9:                 $g(s') = \infty$
10:             if $g(s') > g(s^*) + c(s^*, s')$ then
11:                 $g(s') = g(s^*) + c(s^*, s')$
12:                 if $s' \notin CLOSED$ then
13:                     insert $s'$ into $OPEN$ with $f(s')$
14:         $s^* = \underset{s \in OPEN}{\operatorname{argmin}}\{f(s)\}$
15:         remove $s^*$ from $OPEN$
16:         $CLOSED = CLOSED \cup s^*$
17:     return $\Pi^*(\eta) : s_{start} \rightarrow s_{goal}$
```

Heuristic $h(s)$ is considered admissible if it never overestimates the best path cost to s_{goal}, which means that:

$$h(s) \leq c^*(s, s_{goal}), \forall s \in S, \tag{1.3}$$

and is consistent if it satisfies:

$$\begin{cases} h(s_{goal}) = 0, \\ h(s) \leq h(s') + c(s,s'), \ \forall s, s' \mid s' \in SUCCESSOR(s), s \neq s_{goal}. \end{cases}$$

In [16], it is proven that if $h(s)$ is admissible, then the solution of WA* is bounded as follows:

$$g(s) + \eta h(s) \leq \eta(g^*(s) + h(s)), \quad \forall s \in S, \tag{1.4}$$

where $g^*(s)$ is the cost of an optimal path from s_{start} to s. In other words, the solution cost returned by Algorithm 1, $c(\Pi^*(\eta))$, is no greater than η times the cost of the optimal solution returned by A^*. Moreover, Algorithm 1 does not require re-expansions to guarantee bound (1.4) if $h(s)$ is consistent [15].
Parameter η allows trading off how good the path is versus how long it takes to compute. Different algorithms have been implemented (for instance see [15], [17], [18] and [19]) to search a lattice-based graph to produce paths with user-specified η. In some cases, using hight values of η, a suboptimal solution can be returned very quickly while for a specified amount of time a lower η-solution can be found. This is how ARA* works [15]. For instance, if a car is driving really fast, one can reduce the optimality of the path in order to reduce the computation time. If the car is driving slowly, and we would like minimal-cost paths, then we can allot more time for the algorithm to run.

1.2.2 Dynamic Programming

The Dynamic Programming approach to the motion planning problem involves the numerical solution of the Hamilton-Jacobi-Bellman (HJB) equation. The heart of the algorithm relies in finding the optimal value function (or optimal cost-to-go function) which is the solution to the first-order differential equation (the HJB equation). This value function represents the minimal cost for completing the task from the current configuration of the robot. The minimum-cost path can then be found following the negative gradient of the value function.

The dynamic programming method

Assume that the motion of a robot is modeled by an ordinary differential equation (ODE) of the form:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t)), \\ x(0) = x_0. \end{cases} \tag{1.5}$$

Here, the starting configuration $x_0 \in \mathbb{R}^n$ and the continuous function $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ are known. Function $u : [0, \infty) \to U$ is the control input and $U \in \mathbb{R}^m$ is a compact set of admissible controls. The cost functional we want to minimize, is defined as:

$$J_{x_0}(u) := \int_0^t l(x(t), u(t))dt, \tag{1.6}$$

where $x(t)$ solves (1.5) for the control $u(t)$, and $l : \mathbb{R}^n \times U \to \mathbb{R}$ is a continuous cost function.

The goal is to find an optimal control $u^*(t)$ (it can be not unique) that minimizes cost functional (1.6) and allows us to find an optimal solution for system (1.5).

Following [20], we assume that there exist positive real constants L_f, L_l, M_f, M_l such that, $\forall x_1, x_2 \in \mathbb{R}^n$, $\forall u \in U$,

$$\|f(x_1, u) - f(x_2, u)\| \leq L_f \|x_1 - x_2\|, \quad \|f(x_1, u)\| \leq M_f,$$

$$\|l(x_1, u) - l(x_2, u)\| \leq L_l \|x_1 - x_2\|, \quad \|l(x_1, u)\| \leq M_l.$$

The value function $\bar{v} : \mathbb{R}^n \to \mathbb{R}$ is defined as the best value of cost functional (1.6) for the starting state x_0:

$$\bar{v}(x_0) = \inf_{u \in U} J_{x_0}(u). \tag{1.7}$$

Since the value function (1.7) is generally unbounded, a common approach is to perform the following rescaling of $\bar{v}$ (see [21] and [22]):

$$v(x_0) = \begin{cases} \frac{1}{\lambda} & \text{if } \bar{v}(x_0) = +\infty, \\ \frac{1}{\lambda} - \frac{1}{\lambda} e^{-\lambda \bar{v}(x_0)} & \text{otherwise,} \end{cases} \tag{1.8}$$

where λ is a positive scalar. The change of variable (1.8) is called the Kružkov transformation and gives several advantages. In particular, v takes values in $\left[0, \frac{1}{\lambda}\right]$ (whereas $\bar{v}$ is generally unbounded), and this helps in both the analysis and the numerical approximation. Function $v : \mathbb{R}^n \to \mathbb{R}$ is itself a value function defined as

$$v(x_0) = \inf_{u \in U} \int_0^\infty l(x(t), u(t)) e^{-\lambda t} dt, \tag{1.9}$$

and is the unique viscosity solution of HJB equation:

$$\lambda v(x) + \sup_{u \in U} \{-Dv(x) f(x,u) - l(x,u)\} = 0, \quad x \in \mathbb{R}^n, \tag{1.10}$$

where Dv denotes the gradient of v at x. See [23] for the derivation of equation (1.10). The dynamic programming method can be decomposed as follows.

- **Step 1** Solve dynamic programming equation (1.10) to compute the value function v.
- **Step 2** Use the value function v to design an optimal feedback control $u^*(t)$ for system (1.5).

Step 1 In general, the HJB equation (1.10) is a nonlinear partial differential equation, so it is computed by numerical procedures which allow finding a linear approximation to the value function (see [24], [25], [26]). Here, we will consider the approximation scheme described in Appendix A of [20]. This scheme is based of two steps. First a time discretization of the original control problem is performed by time step $h = \Delta t$. Then a space discretization with space step $k = \Delta z$ is computed.

Thus, we first make a discretization in time of the dynamic programming equation, approximating $Dv(x)f(x,u) \simeq h^{-1}(v(x+hf(x,u)) - v(x))$ where h is a small positive real number that represents an integration time. In this way, (1.10) becomes

$$(1+\lambda h)v(x) = \min_{u \in U}\{v(x+hf(x,u)) + hl(x,u)\} = 0, \quad x \in \mathbb{R}^n,$$

by approximating $(1+\lambda h)^{-1} \simeq (1-\lambda h)$, $(1+\lambda h)^{-1}h \simeq h$ one arrives at the following HJB equation in discrete time

$$v_h(x) = \min_{u \in U}\{(1-\lambda h)v_h(x+hf(x,u)) + hl(x,u))\}, \quad x \in \mathbb{R}^n. \tag{1.11}$$

For a more rigorous derivation of (1.11), again, see [20].
In order to make a space discretization, we compute a triangulation on a finite set of vertices $S = \{x_i\} \subset \mathbb{R}^n$, $i = 1,\dots,N$. Evaluating (1.11) at $x \in S$, we have

$$\begin{aligned} v_h(x_i) = \min_{u \in U}\{(1-\lambda h)v_h(x_i+hf(x_i,u)) + hl(x_i,u))\}, \\ i = 1,\dots,N. \end{aligned} \tag{1.12}$$

Note the dependence of the value cost function on the choice of the integration step h. Using the triangulation, function v can be approximated by a linear function of the finite set of variables $v_h(x_i)$, $i = 1,\dots,N$.

Theorem 2.1 of Appendix A of [20] shows that, if $\lambda > L_f$ and $h \in (0, \frac{1}{\lambda}]$, system (1.12) has a unique solution that converges uniformly to the solution of (1.10) as $h, k, \frac{k}{h}$ tend to 0. Note that, to have convergence, one cannot choose λ arbitrarily small since it is bounded from below by L_f.

To further simplify (1.12), it is possible to discretize the control space, substituting U with a finite set of controls $\{u_1,\dots,u_M\}$, so that we can replace (1.12) with

$$\begin{aligned} v_h(x_i) = \min_{u_k}\{(1-\lambda h)v_h(x_i+hf(x_i,u_k)) + hl(x_i,u_k))\}, \\ i = 1,\dots,N \text{ and } k = 1,\dots,M. \end{aligned} \tag{1.13}$$

Figure 1.8 illustrates a step of the construction of problem (1.13). Namely, for each node of the triangulation x_i and each value of the control u_k, all end points $x_i + hf(x_i,u_k)$ of the Euler approximation of the solution of ODE (1.5) from the initial

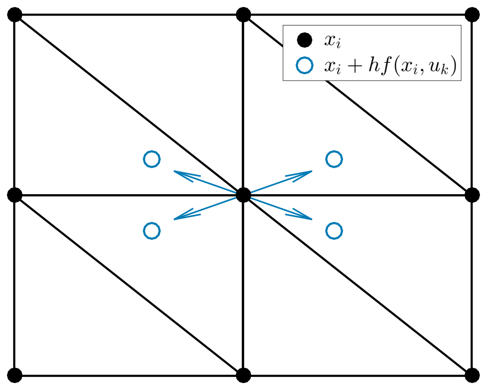


Figure 1.8: Approximation of the HJB equation on a 2-D triangulation with four controls.

state x_i are computed. The value cost function for these end points is given by a linear combination of its values on the triangulation vertices.

Set vector $z := [v_h(x_1), v_h(x_2), \dots, v_h(x_N)]$, in this way $z \in \mathbb{R}^N$ represents the value of the cost function on the grid points.

Note that, for each x_i, u_k, the right-hand side of (1.13) is affine with respect to z, so that problem (1.13) can be rewritten in form

$$z = \min_{i=1,\dots,M} \{A_i z + b_i\}, \tag{1.14}$$

where, for $i = 1, \dots, M$, $A_i \in \mathbb{R}_+^{N\times N}$ are suitable nonnegative matrices and $b_i \in \mathbb{R}_+^N$ are suitable nonnegative vectors.
Define map $T : \mathbb{R}_+^N \to \mathbb{R}_+^N$ as

$$T(x) := \min_{i=1,\dots,M} \{A_i x + b_i\}. \tag{1.15}$$

In [20] it is shown that T is a contraction, so that (1.14) can be solved as a fixed point iteration. Namely, setting

$$\begin{cases} x(k+1) = T(x(k)), \\ x(0) = x_0, \end{cases}$$

the solution z of (1.14) is obtained as $z = \lim_{k\to\infty} x(k)$, for any initial condition x_0.

Step 2 An approximated optimal control can be obtained from the numerical solution of HJB equation by setting $u^*(x) = u_h^*(x)$ such that:

$$v_h(x) = (1-\lambda h)v_h(x+hf(x,u_k^*)) + hl(x_i,u_k^*)).$$

For instance, approximating the solution of ODE (1.5) with an Euler approximation:

$$\begin{cases} z_{r+1} = z_r + hf(z_{r+1},u_r), & r\in\mathbb{N} \\ z_0 = x_0, \end{cases}$$

the feedback control law is:

$$u_r^* = u_h^*(z_m), \quad r\in\mathbb{N}.$$

In other word we design the optimal control choosing the control such that the minimum in HJB is obtained.

The minimum-time maneuvering problem

A classical path planning task is to find the minimum-time path to perform a parking maneuver. In order to employ the dynamic programming method to this problem we first describe the car-robot by an ordinary differential equation. For instance we can use the kinematic car-like model with rear-wheel drive (see Figure 1.1):

$$\begin{cases} \dot{z} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega, \end{cases}$$

where the triplet $x = (z,y,\theta)$ represents a configuration of the vehicle and the control input is given by $u = (v,\omega)$; and v and ω are, respectively, the linear and angular velocities. The angular velocity is related to the front wheel steering angle δ by relation $\omega = \frac{1}{l}v\tan\delta$. The input variables are constrained as follows

$$v_{min} < v < v_{max}, \qquad \omega_{min} < \omega < \omega_{max}.$$

Now we need to appropriately define the cost functional to formulate the car maneuvering task as a minimum-time problem. Let $x_f := (z_f, y_f, \theta_f)$ be a desired final configuration and let $\bar{R} \subset R \subset \mathbb{R}^2 \times S^1$ be closed balls of radiuses $\bar{\varepsilon} > \varepsilon > 0$, centered at x_f. Cost function $l : \mathbb{R}^2 \times S^1 \to \mathbb{R}$ is defined as

$$l(x) := \begin{cases} 0 & \text{if } x \in \bar{R}, \\ \frac{\|x - x_f\| - \varepsilon}{\bar{\varepsilon} - \varepsilon} & \text{if } x \in R \setminus \bar{R}, \\ 1 & \text{otherwise.} \end{cases}$$

Then the cost functional (1.6) begins

$$J_{x_0}(u) := \int_0^\infty l(x(t)) e^{-\lambda t} dt \,. \tag{1.16}$$

Using (1.16), value function (1.9) represents the minimum time to reach the target set $\bar{R}$ for the starting state x_0

An approximated optimal control can then be obtained from the numerical solution of HJB equation (1.10) by setting the following feedback control law $u_h^\star : \mathbb{R}^N \to U$:

$$\begin{aligned} u_h^\star(x) = \arg\min_{u_k} \{ (1 - \lambda h) v_h(x + hf(x, u_k)) + hl(x, u_k) \}, \\ k = 1, \ldots, M. \end{aligned} \tag{1.17}$$

Note that it is also possible to take into account the presence of obstacles by suitably modifying l, for instance assigning to l a very high value in subsets of the state space corresponding to forbidden regions (see [20] for details).

Chapter 2

A Jacobi-like acceleration for dynamic programming

This chapter is based on [27].

In section 1.2.2, we saw how dynamic programming provides a general method to solve optimal control problems. Dynamic programming is based on the solution of the HJB equation which is a nonlinear partial differential equation and a closed form solution does not exist for the general case. Thus, various numerical procedures have been developed for its solution. In particular, after some simplifications, the problem can be converted into a fixed-point iteration. One problem related to the iteration is that its convergence speed is rather poor. This chapter proposes a Jacobi-like acceleration to improve this convergence speed and shows the results of its application to the minimum-time maneuvering problem (see section 1.2.2).

2.1 Problem Statement

As shown in section 1.2.2, the solution of the HJB equation (1.10) is obtained by the following fixed point iteration

$$\begin{cases} x(k+1) = \min\limits_{i=1,\dots,M} \{A_i x(k) + b_i\}, \\ x(0) = x_0, \end{cases} \tag{2.1}$$

where $x_0 \in \mathbb{R}^N$ and, for $i = 1,\dots,M$, $A_i \in \mathbb{R}_+^{N\times N}$ are nonnegative matrices and $b_i \in \mathbb{R}_+^N$ are nonnegative vectors.

Note that, according to the discussion in the previous section, iteration (2.1) is convergent and $x^\star = \lim_{k\to\infty} x(k)$ satisfies

$$x^\star = \min_{i=1,\dots,M} \{A_i x^\star + b_i\}, \tag{2.2}$$

and is the solution of our optimization problem.

Remark 1. *Note that, due to the sparse structure of matrices A_i, for $i = 1,\dots,M$, problem (2.1) can be considered a consensus problem. The consensus equilibrium is represented by the fixed point $x = T(x)$.*

At this point, we make some additional assumptions on A_i that are justified by the properties of the triangulation. To this end, let Δ be the maximum degree of the graph associated to the triangulation, that is, Δ represents the maximum number of neighbours of any vertex. Moreover, let ℓ be the minimum length of an edge of the triangulation.

Let h be such that $hM_f < \ell$, then, for any $i = 1,\dots,N$, $k = 1,\dots,M$, vector $x_i + hf(x_i,u_k)$ belongs to a triangle of the triangulation of which x_i is one of the vertices. Let $x_{i_1}, x_{i_2},\dots,x_{i_l} \in S$, $l \leq \Delta$ be the other vertices. Then, there exist positive coefficients λ_j, $j = 0,\dots,l$, such that $x_i + hf(x_i,u_k) = \lambda_0 x_i + \sum_{j=1,\dots,l} \lambda_j x_{i_j}$, with $\sum_{j=0,\dots,l} \lambda_j = 1$. It is easy to prove that $\lambda_0 \geq 1 - \frac{M_f h}{\ell}$ and $\sum_{j=1,\dots,l} \lambda_j \leq \frac{M_f h}{\ell}$. Further, consider set

$$S_{j,k} := \{i = 1,\dots,N : x_i + hf(x_i,u_k) \text{ belongs to a triangle that has } x_j \text{ as a vertex}\}.$$

It is easy to verify that $|S_{j,k}| \leq \Delta$, for any $i = 1,\dots,N$, $k = 1,\dots,M$. Set $A_k = (a_{ij,k})$, then, the previous considerations imply that, for $k = 1,\dots,M$,

$$a_{ii,k} \geq (1-\lambda h)\left(1-\frac{M_f h}{\ell}\right),$$

$$\sum_{i\neq j} a_{ij,k} \leq (1-\lambda h)\frac{M_f h}{\ell},$$

$$\sum_{i\neq j} a_{ji,k} \leq (1-\lambda h)\frac{\Delta M_f h}{\ell}.$$

Define the constant $C = \frac{\Delta M_f}{\ell}$.
The speed of convergence of the fixed point iteration (2.2) can be measured by the convergence rate, defined as

$$\chi = \max_{x,y\in\mathbb{R}^N,\ x\neq y}\left\{\frac{\|T(x)-T(y)\|}{\|x-y\|}\right\},$$

with T as in (1.15). We want to find an **acceleration policy** for computing $x^\star$, namely we want to solve the following problem.

Problem 1. *Find matrices $\hat{A}_i$ and vectors $\hat{b}_i$, for $i = 1,\dots,M$, such that* $\mathrm{sp}(\hat{A}_i) \leq \mathrm{sp}(A_i)$, *for $i = 1,\dots,M$ and the solution of problem*

$$\begin{cases} \hat{x}(k+1) = \min_{i=1,\dots,M}\{\hat{A}_i x(k) + \hat{b}_i\}, \\ \hat{x}(0) = x_0 \end{cases} \tag{2.3}$$

satisfies $\lim_{k\to\infty}\hat{x}(k) - x(k) = 0$.
Further, we require that $\hat{\chi} < \chi$ where

$$\hat{\chi} = \max_{x,y\in\mathbb{R}^N,\ x\neq y}\left\{\frac{\|\hat{T}(x)-\hat{T}(y)\|}{\|x-y\|}\right\},$$

with $\hat{T}$ defined as in (1.15) *over $\hat{A}_i$ and $\hat{b}_i$.*

In other words, we want to find modified matrices $\hat{A}_i$ and vectors $\hat{b}_i$ such that the convergence of the new fixed-point iteration (2.3) is faster than the original iteration (2.1). The sparsity requirement $\mathrm{sp}(\hat{A}_i) \leq \mathrm{sp}(A_i)$, $i = 1,\dots,M$, is important to ensure that the computation of the sparse matrix product in iteration (2.3) does not require more processing time than the original problem.

2.2 Main Result

When solving linear systems one can use Jacobi's method choosing as preconditioner the diagonal of the matrix associated to the system (see [28]), but this is not applicable in general when solving nonlinear systems. However, this kind of preconditioning is computationally cheap since it requires the inversion of a diagonal matrix, so one would like to generalize this method in order to apply it to a wider class of problems. To this end, let us define a diagonal preconditioner D as

$$D := \max_{i=1,\dots,M}\{I - \operatorname{diag}(A_i)\} = I - \min_{i=1,\dots,M}\{\operatorname{diag}(A_i)\}, \tag{2.4}$$

where $I \in \mathbb{R}^{N\times N}$ is the identity matrix and $\operatorname{diag}(A) = (d_{ij})$ is a diagonal matrix defined as follows:

$$d_{ij} := \begin{cases} 0, & \text{if } i \neq j, \\ a_{ij}, & \text{if } i = j, \end{cases}$$

for $i, j \in \{1,\dots,N\}$, with $A = (a_{ij}) \in \mathbb{R}^{N\times N}$.
Given $A = (a_{ij}) \in \mathbb{R}^{N\times N}$ let us define, for $k = 1,\dots,N$

$$s_k(A) := \sum_{j=1}^{N} a_{kj}.$$

Note that $D = (d_{ij})$ defined in (2.4) is a positive diagonal matrix, since

$$\forall k \in \{1,\dots,N\}\ \forall i \in \{1,\dots,M\}\ d_{kk} > 1 - s_k(A_i) = 1 - \mu > 0,$$

where $\mu := 1 - \lambda h \in (0,1)$.
One can rewrite (2.2) as:

$$(D - D + I)x^\star = \min_{i=1,\dots,M}\{A_i x^\star + b_i\},$$

that is

$$Dx^\star = \min_{i=1,\dots,M}\{(A_i + D - I)x^\star + b_i\}.$$

Thus, it is natural to set $\hat{A}_i := D^{-1}(A_i + D - I)$ and $\hat{b}_i := D^{-1}b_i$ in (2.3). In the following theorem, which is our main result and will be proved in section 2.4, we claim that this choice provides a solution to problem (1):

Theorem 1. *If*

$$0 < h \leq \frac{\lambda}{(2C+\lambda)(C+\lambda)},$$

the choice

$$\hat{A}_i = D^{-1}(A_i + D - I), \;\; \hat{b}_i = D^{-1}b_i, \tag{2.5}$$

for $i = 1,\dots,m$, solves problem (1). In particular, $\forall x,y \in \mathbb{R}^N_+$

$$\|\hat{T}(x) - \hat{T}(y)\| < \|T(x) - T(y)\|, \tag{2.6}$$

moreover

$$\hat{\chi} \leq \frac{C(1-\lambda h)}{\lambda + C(1-\lambda h)}.$$

2.3 Numerical Experiments

Let us consider the parking problem presented in section 1.2.2. A uniform triangulation with 34848 vertices is performed over the torus $\Omega = [-0.5, 16.5] \times [-4, 9] \times S^1$, with final configuration $x_f = (8,0,0)$.
Figure 2.1 shows the value of error norm $\|x(k) - x^\star\|$ for the classic fixed-point iteration T and the Jacobi-like acceleration $\hat{T}$. In accordance to Theorem 1, $\hat{T}$ converges faster to the fixed point than T. The number of iterations required to satisfy the termination condition for the classic fixed point iteration (i.e. $\|T(x(k)) - x^\star\| \leq 10^{-4}$) is 3510 whilst for the Jacobi-like iteration (i.e. $\|\hat{T}(x(k)) - x^\star\| \leq 10^{-4}$) it is 496. Figure 2.2 shows some car maneuvers obtained with the control obtained from the numerically computed cost function, corresponding to different initial conditions.

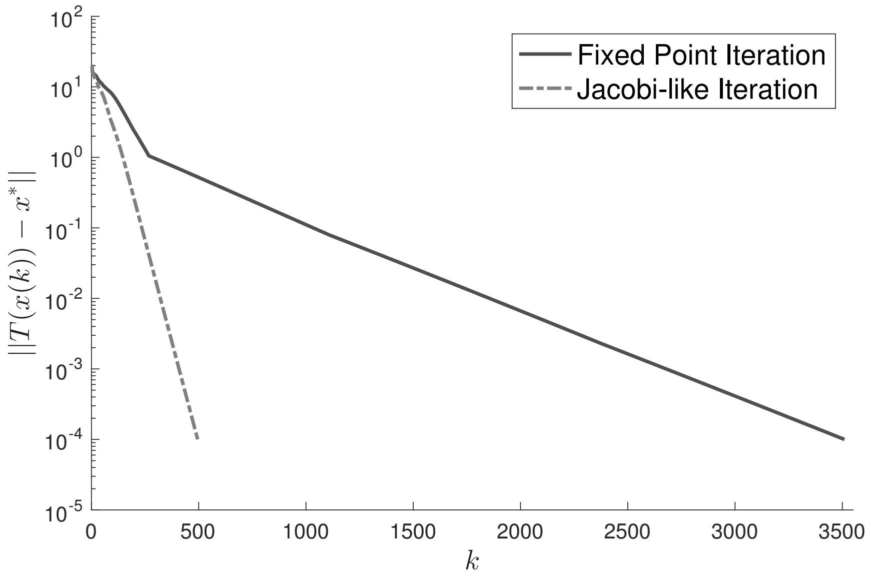


Figure 2.1: Absolute error norm of Fixed Point Iteration compared to Jacobi-like Iteration.

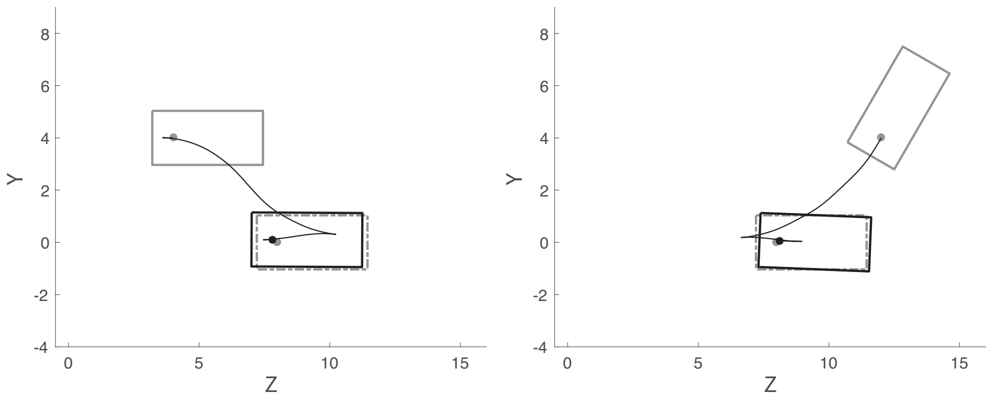


Figure 2.2: Two car maneuvers, with different initial conditions, obtained by control law (1.17). In light grey the initial position, in dash-dot grey the desired final position and in dark grey the actual final reached position.

2.4 Proofs of the Main Result

Theorem 2 (Perron-Frobenius)**.** *Let $A \in \mathbb{R}_+^{N\times N}$ be a nonnegative matrix. Then $\exists\, \lambda \in \sigma(A) : \lambda = \rho(A)$ and*

$$\min_{k=1,\dots,N} s_k(A) \le \rho(A) \le \max_{k=1,\dots,N} s_k(A). \tag{2.7}$$

In addition, if A is irreducible, (2.7) holds with equalities on both sides if and only if $\forall k,l \in \{1,\dots,N\}$ $s_k(A) = s_l(A)$.

For a proof of Theorem 2, see [29] and [30].

Theorem 3 (Gershgorin Circle)**.** *Let $A = (a_{ij}) \in \mathbb{C}^{N\times N}$ be a complex matrix and let, for $i = 1,\dots,N$,*

$$R_i := \sum_{j=1, j\neq i}^{N} |a_{ij}|, \qquad \mathscr{D}(a_{ii}, R_i) := \{z \in \mathbb{C} \,|\, |a_{ii} - z| \le R_i\}.$$

Then

$$\sigma(A) \subset \bigcup_{i=1}^{N} \mathscr{D}(a_{ii}, R_i).$$

For a proof of Theorem 3, see [31].

Proposition 1. *For all $x \in \mathbb{R}_+^N$*

$$\hat{T}(x) = (I - D^{-1})x + D^{-1}T(x),$$

where T and $\hat{T}$ are defined as in (1.15).

Proof. Rewrite $\hat{A}_i$ as

$$\hat{A}_i = D^{-1}(A_i + D - I) = A_i + (D^{-1} - I)(A_i - I)$$

then

$$\begin{aligned}\hat{T}(x) &= \min_{i=1,\dots,M}\{A_i x + (D^{-1} - I)(A_i - I)x + D^{-1}b_i\} = \\ &= -(D^{-1} - I)x + \min_{i=1,\dots,M}\{A_i x + (D^{-1} - I)A_i x + D^{-1}b_i\} =\end{aligned}$$

$$\begin{aligned}
&= -(D^{-1} - I)x + D^{-1} \min_{i=1,\dots,M} \{A_i x + b_i\} = \\
&= -(D^{-1} - I)x + D^{-1} T(x).
\end{aligned}$$

□

Given preconditioner $D = (d_{ij})$ as in (2.4), for $k = 1, \dots, N$ and $i = 1, \dots, M$ we have that

$$s_k(\hat{A}_i) = \frac{\mu + d_{kk} - 1}{d_{kk}} = 1 - \frac{1-\mu}{d_{kk}}.$$

Since $s_k(\hat{A}_i) = s_k(\hat{A}_j)$, for all $i, j \in \{1, \dots, M\}$, we can define

$$s_k := 1 - \frac{1-\mu}{d_{kk}}, \text{ for } k = 1, \dots, N,$$

and, set $\alpha = Ch$, let

$$\begin{aligned}
S := \max_{k=1,\dots,N} s_k = 1 - \frac{1-\mu}{\max\limits_{k=1,\dots,N} d_{kk}} \le \\
\le 1 - \frac{1-\mu}{1-\mu(1-\alpha)} = \frac{\mu\alpha}{1-\mu+\mu\alpha}.
\end{aligned} \tag{2.8}$$

Proposition 2. *For all $x, y \in \mathbb{R}^N_+$, it holds that*

$$\|\hat{T}(x) - \hat{T}(y)\| \le \frac{\mu\alpha}{1-\mu+\mu\alpha} \|x - y\|.$$

Proof. Let $\tilde{A} \in \mathbb{R}^{N\times N}_+$ and $\tilde{b} \in \mathbb{R}^N_+$ be such that $\tilde{A}y + \tilde{b} = \hat{T}(y)$. Then

$$\begin{aligned}
\|\hat{T}(x) - \hat{T}(y)\| &= \|\hat{T}(x) - (\tilde{A}y + \tilde{b})\| \le \\
&\le \|\tilde{A}x + \tilde{b} - (\tilde{A}y + \tilde{b})\| \le \\
&\le \|\tilde{A}(x - y)\| \le \\
&\le \rho(\tilde{A}) \|x - y\|.
\end{aligned}$$

By Frobenius Theorem 2 and (2.8) we know that

$$\rho(\tilde{A}) \leq S \leq \frac{\mu\alpha}{1-\mu+\mu\alpha},$$

that is

$$\|\hat{T}(x) - \hat{T}(y)\| \leq \frac{\mu\alpha}{1-\mu+\mu\alpha}\|x-y\|.$$

□

Now we provide an overestimation of the convergence rate χ of (2.2), that is, we give a lower bound on χ.

Proposition 3. *For all $x, y \in \mathbb{R}^N_+$, it holds that*

$$\|T(x) - T(y)\| \geq \mu(1-2\alpha)\|x-y\|.$$

Proof. For all $v = (v_i) \in \mathbb{R}^N$, $\operatorname{sgn}(v) \in \{0,1\}^N$ is the component-wise sign of v, that is, for $i = 1, \ldots, N$,

$$\operatorname{sgn}(v)_i := \begin{cases} 1, & \text{if } v_i > 0, \\ 0, & \text{if } v_i \leq 0. \end{cases}$$

Given $x, y \in \mathbb{R}^N_+$, let $A_x, A_y \in \mathbb{R}^{N\times N}_+$ and $b_x, b_y \in \mathbb{R}^N_+$ be such that

$$A_x x + b_x = T(x) \text{ and } A_y y + b_y = T(y),$$

then we can define $\bar{A} \in \mathbb{R}^{N\times N}_+$ in the following way

$$\bar{A} := UA_x + (I-U)A_y, \qquad \bar{b} := Ub_x + (I-U)b_y,$$

where $U = (u_{ij}) \in \mathbb{R}^{N\times N}_+$ is a diagonal matrix such that
$u_{ii} = \operatorname{sgn}(x-y)_i$, for $i = 1, \ldots, N$. Then it holds that

$$\langle T(x) - T(y) - \bar{A}(x-y),\, x-y\rangle \geq 0, \tag{2.9}$$

in fact, for components such that $x - y \leq 0$, we have

$$T(x) - T(y) - \bar{A}(x-y) = T(x) - (\bar{A}x + \bar{b}) \leq 0,$$

and for those such that $x-y>0$,

$$T(x)-T(y)-\bar{A}(x-y)=(\bar{A}y+\bar{b})-T(y)>0.$$

Inequality (2.9) implies that

$$\langle T(x)-T(y),\, x-y\rangle \geq \langle \bar{A}(x-y),\, x-y\rangle = (x-y)^T\bar{A}^T(x-y). \tag{2.10}$$

If we rewrite matrix $\bar{A}$ as the sum of its symmetric part $H=(\eta_{ij}):=(\bar{A}+\bar{A}^T)$ and skew-symmetric part $K:=(\bar{A}-\bar{A}^T)$ we obtain

$$\bar{A}=\frac{1}{2}H+\frac{1}{2}K.$$

Now, since $\forall v\in\mathbb{R}^N$ $v^T\bar{A}^Tv=v^T\bar{A}v$, it holds that $\forall v\in\mathbb{R}^N$ $v^TKv=0$. This means that

$$(x-y)^T\bar{A}^T(x-y)=(x-y)^T\frac{1}{2}H^T(x-y)\geq\lambda_{\min}||x-y||^2, \tag{2.11}$$

where $\lambda_{\min}$ represents the smallest eigenvalue in magnitude of $\frac{1}{2}H^T$. Our aim now is to provide a lower bound on this eigenvalue. By Gershgorin Circle Theorem 3, letting

$$\tilde{\eta}:=\min_{k=1,\dots,N}\eta_{kk},\quad \tilde{R}:=\max_{l=1,\dots,N}R_l,\quad \mathscr{D}_{\mathbb{R}}:=\mathscr{D}(\tilde{\eta},\tilde{R})\cap\mathbb{R},$$

and recalling that the eigenvalues of a symmetric matrix are real, we know that for all $\lambda_H\in\sigma\left(\frac{1}{2}H^T\right)$:

$$\begin{aligned}\lambda_H\geq\min_{z\in\mathscr{D}_{\mathbb{R}}}z\geq\min_{k=1,\dots,N}\eta_{kk}-\max_{l=1,\dots,N}\sum_{p=1,p\neq l}^{N}\eta_{lp}\geq\\ \geq\mu(1-\alpha)-\mu\alpha=\mu(1-2\alpha).\end{aligned} \tag{2.12}$$

Let us now recall Cauchy-Schwarz inequality:

$$||T(x)-T(y)||\cdot||x-y||\geq\langle T(x)-T(y),\, x-y\rangle. \tag{2.13}$$

Hence, putting together (2.10), (2.11), (2.12) and (2.13), we can conclude that

$$||T(x)-T(y)||\geq\mu(1-2\alpha)||x-y||.$$

□

We are now ready to prove Theorem 1.

Proof. By Propositions (2) and (3), statement (2.6) holds if

$$\frac{\mu\alpha}{1-\mu+\mu\alpha} < \mu(1-2\alpha),$$

Recalling that $\alpha = Ch > 0$ and $\mu = 1-\lambda h$, we obtain

$$\alpha < \frac{3\mu-3+\sqrt{\mu^2-10\mu+9}}{4\mu},$$

that is

$$Ch < \frac{-3\lambda h+\sqrt{\lambda^2h^2+8\lambda h}}{4(1-\lambda h)}.$$

Now, since $1-\lambda h < 1$, we strengthen the constraint requiring that

$$Ch \leq \frac{-3\lambda h+\sqrt{\lambda^2h^2+8\lambda h}}{4} < \frac{-3\lambda h+\sqrt{\lambda^2h^2+8\lambda h}}{4(1-\lambda h)},$$

from which follows that

$$(4C+3\lambda)h \leq \sqrt{\lambda^2h^2+8\lambda h}.$$

By squaring both members of the previous inequality we obtain

$$h[(2C+\lambda)(C+\lambda)h-\lambda] \leq 0,$$

that leads to

$$h \in \left(0, \frac{\lambda}{(2C+\lambda)(C+\lambda)}\right].$$

□

Chapter 3

Path planning with limited numbers of maneuvers

The content of this chapter is based on [32].

This chapter presents a deterministic algorithm for path planning of parking maneuvers. On the basis of a know environment in the presence of static obstacles, the algorithms finds the trajectory that minimizes the length of the parking maneuver and has a number of direction changes lower than a fixed constant. This method is proposed in contrast to the common approach of finding a minimum-time path without taking into account the numbers of direction changes. The rationale for this approach is twofold. First, in automatic parking systems each change of direction is a time-consuming operation, since it requires a deceleration to a full stop, a gear change and an acceleration. For this reason, a longer trajectory with a small number of direction changes is preferable to a shorter one with more maneuvering points. In addition, this choice better reproduces the approach of a typical human driver, that prefers a longer and simpler maneuver to a shorter but complex one, with more maneuvering points. For instance, in Figure 3.1, a human driver would prefer the trajectory on the right, due to the availability of a large maneuvering space.

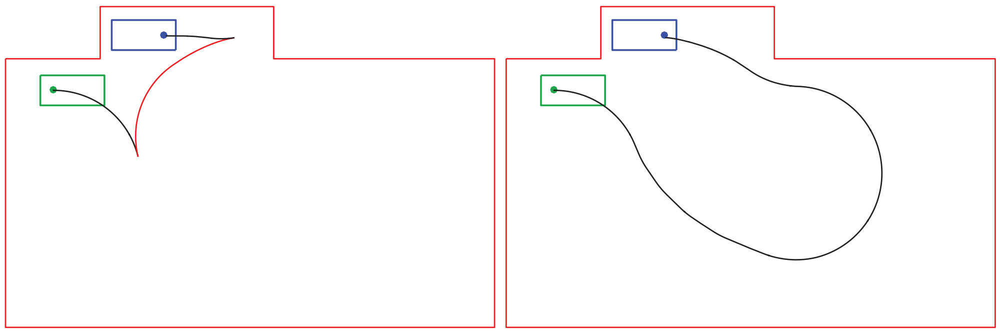


Figure 3.1: Different paths that reach the same target configuration (represented in blue).

The proposed algorithm is based on Dynamic Programming and uses a cost function that allows taking into account both the length of the path and a chosen maximum number of maneuvers. To numerically solve this problem, we leverage some recent results on the optimal control of switching systems. In particular:

- We use the results presented in [33], that show that the cost function corresponds to the viscosity solution of a system of quasi variational inequalities (QVIs).
- Using the method presented in [34], we convert the problem into the solution of a sequence of decoupled QVIs.
- Each QVI is solved with the finite element method presented in [35].

3.1 Problem Statement

Let $\Omega \subset \left(\mathbb{R}^2 \times [0, 2\pi[\right)$ be a bounded and connected domain, which represents the operating space of the vehicle. This space is partitioned as $\Omega = \Omega_{free} \cup \Omega_{obst}$, with $\Omega_{free} \cap \Omega_{obst} = \varnothing$, where Ω_{free} is the free space and Ω_{obst} is the subset of the operating space covered by obstacles. Consider the kinematic car-like model with rear-wheel drive defined by system:

$$\begin{cases} \dot{z} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega, \end{cases}$$

where the triplet $x = (z, y, \theta)$ represents a configuration of the vehicle and the control input is given by $u = (v, \omega)$; and v and ω are, respectively, the linear and angular velocities. The angular velocity is related to the front wheel steering angle δ by relation $\omega = \frac{1}{l} v \tan\delta$. The input variables are constrained as follows

$$v_{min} < v < v_{max}, \qquad \omega_{min} < \omega < \omega_{max}.$$

We model the car-like vehicle with the switched system $\dot{x}(t) = f(x(t), i, u(t))$, where $f : \Omega \times \{1,2\} \times \mathbb{R} \to \Omega$ is defined as:

$$f(x,1,\omega) = \begin{pmatrix} V_+(x)\cos\theta \\ V_+(x)\sin\theta \\ \omega \end{pmatrix}, \tag{3.1}$$

$$f(x,2,\omega) = \begin{pmatrix} V_-(x)\cos\theta \\ V_-(x)\sin\theta \\ \omega \end{pmatrix}, \tag{3.2}$$

where $V_+, V_- : \Omega \to \mathbb{R}$ are defined by

$$V_+(x) = \begin{cases} v_+ r & \text{if } x \in \Omega_{obst}, \\ v_+ & \text{otherwise ,} \end{cases}$$

$$V_-(x) = \begin{cases} v_- r & \text{if } x \in \Omega_{obst}, \\ v_- & \text{otherwise ,} \end{cases}$$

and $0 < r << 1$ is a small number. Here, $v^- < 0 < v^+$ are the speeds associated to reverse and forward gears and $i \in \{1,2\}$ denotes the configuration of the vehicle. Namely, $i = 1$ is associated to forward gear and $i = 2$ to reverse gear. Note that, if

$x \in \Omega_{obst}$, the velocity of the vehicle is reduced by factor r. Being r close to zero, the vehicle is very slow when traveling on obstacles. Since we will be interested in minimum time trajectories, for small values of r optimal trajectories will not intersect set Ω_{obst} for relevant intervals of time. Note that this is not the only way to take into account the presence of obstacles, for instance, one could add boundary conditions on the border of Ω_{obst}.
The control signal is given by the couple $\alpha = (\omega, \sigma)$, where $\omega : [0, +\infty) \to [\omega_{min}, \omega_{max}]$ is the steering control input and $\sigma \in (\mathbb{R}, \{1,2\})^\star$ is the sequence of configuration changes. In this way, the set of the control signals is defined as $A = L^\infty(\mathbb{R}, [-\omega_{min}, \omega_{max}])$ $\times (\mathbb{R}, \{1,2\})^\star$. If $(t_k, i_k) \in \sigma$, then the controller switches to subsystem i_k at time t_k. The sequence of switches satisfies:

$$\begin{aligned} &0 = \tau_1 < \tau_2 < \cdots < \tau_K, \\ &i_{k-1} \neq i_k, \quad 2 \leq k \leq K. \end{aligned}$$

Define function $I : \mathbb{R} \to \{1,2\}$ as $I(t) = i_{\bar{k}}(t)$ where $\bar{k}(t) = max\{k \in \mathbb{N} : \tau_k \leq t\}$. In this way, $\bar{k}(t)$ represents the total number of switchings occurring in interval $[0,t]$. Given $\alpha = (u, \sigma) \in A$, the trajectory of the switched system (3.1), (3.2) is defined as the solution of equation:

$$\begin{cases} \dot{x}(t) = f(x(t), I(t), u(t)) \\ x(0) = x_0, \end{cases} \tag{3.3}$$

and is denoted by $y_{x_0,\alpha}(t) = x(t)$.

Consider a target set $\Gamma \subset \Omega$, closed and such that $\text{int}(\Gamma) \neq \varnothing$. We define a cost function $t : \Omega \times \{1,2\} \times A \to \mathbb{R} \cup \{+\infty\}$ that associates to each initial state x_0, initial configuration i_0 and control $\alpha \in A$ the cost

$$t(x_0, i_0, \alpha) = \begin{cases} +\infty & \text{if } y_{x_0,\alpha}(t) \notin \Gamma, \forall t > 0, \\ \inf\limits_{t \in \mathbb{R}^+} \{t : y_{x_0,\alpha}(t) \in \Gamma\} & \text{otherwise}, \end{cases} \tag{3.4}$$

where we consider $t(x_0, i_0, \alpha) = +\infty$ if the solution never reaches set Γ for any $t > 0$. In this way, the value function $T : \Omega \times \{1,2\} \to \mathbb{R} \cup \{+\infty\}$ for problem (3.4) is given by

$$T(x_0, i_0) = \inf_{\alpha \in A} t(x_0, i_0, \alpha) \tag{3.5}$$

and represents the first time of arrival on the target Γ. It is possible to solve problem (3.5) via dynamic programming by characterizing the value function in terms of a first order Hamilton-Jacobi-Bellman (HJB) equation. We refer to [33], [34], [36], [37] and [38] for the results related to the solution of Problem (3.5). In particular, as discussed in in section 1.2.2, it is possible to obtain a boundary problem for the HJB equation performing the following rescaling of T:

$$V(x_0, i) = \begin{cases} \frac{1}{\lambda} & \text{if } T(x_0, i) = +\infty, \\ \frac{1}{\lambda} - \frac{1}{\lambda} e^{-\lambda T(x_0, i)} & \text{otherwise,} \end{cases}$$

where λ is a positive scalar. It can be proved that $V : \Omega \times \{1, 2\} \to \mathbb{R} \cup \{+\infty\}$ is itself a value function, that is

$$V(x_0, i) = \inf_{a \in A} J_{x_0, i}(t), \tag{3.6}$$

where $J(x_0, i) = \int\limits_0^{t(x_0, i_0, \alpha)} e^{-\lambda t} dt.$

3.2 Problem Resolution

3.2.1 Analytical Analysis

As already mentioned in section 3.1, problem (3.6) can be addressed via dynamic programming. In particular, it falls into the class of optimal switching control problems which was formulated and studied in [33]. There, it is proven that the value function (3.6) is uniformly bounded, Hölder continuos, and is the unique viscosity solution of the following HJB equation:

$$\max \left\{ \begin{array}{c} V(x, i) - \min\limits_{\bar{i} \neq i} \{V(x, \bar{i})\} \\ \lambda V(x, i) + \sup\limits_{\omega \in W} \{-D(V)(x, i) f(x, i, \omega) - 1\} \end{array} \right\} = 0, \tag{3.7}$$

where $V \in C(\Omega \times \{1,2\},[0,+\infty))$, $x \in \Omega \subset \mathbb{R}^3$, i $\in \{1,2\}$, $W \equiv [\omega_{min}, \omega_{max}]$ and $D(V)(x,i)$ denotes the gradient of function V at x.

Leveraging the results shown in [34], we obtain Algorithm 5 for solving system (3.7) with a sequence of decoupled QVIs.

Algorithm 2 Algorithm for solving problem (3.7)

```
    **Function** Solver $(\Gamma, i_1, \varepsilon)$
    **Input**: $\Gamma$: target state set, $i_1$: target configuration, $\varepsilon$: termination tolerance.
    **Output**: $V^0, \dots, V^K$: cost functions.
1:  *compute* $V^0$ *as the solution of*
        $\lambda V^0(x) + \sup_{\omega \in W} \{-D(V^0)(x) f(x, i_1, \omega) - 1\},$
    $k = 0$, $i = i_1$
2:  **while** $\|V^k - V^{k-1}\|_\infty > \varepsilon$ **do**
3:      $k = k+1$
4:      $i = mod(i+1,2)+1$
5:      *compute* $V^k$ *by*
            $\max \left\{ \begin{array}{c} V^k(x) - \{V^{k-1}(x)\} \\ \lambda V^k(x) + \sup_{\omega \in W} \{-D(V)(x) f(x,i,\omega) - 1\}) \end{array} \right\} = 0.$
    **return** $\{V^0, \dots, V^K\}$
```

Note that this algorithm corresponds to the algorithm presented in [34] applied to our case in which $i \in \{1,2\}$.

In line 1 of Algorithm 5, V^0 is computed as the solution of the HJB equation for a car-like vehicle that proceeds to final state set Γ in forward ($i_1 = 1$) or backward ($i_1 = 2$) direction, without direction changes. Then, in loop 2-5, V^k is computed so that, for any x such that $V^k(x) < V^{k-1}(x)$, V^k is the solution of the HJB equation applied alternatively to subsystems (3.1), (3.2). In other words, $V^0, V^1, \dots, V^K$ is a sequence of functions such that V^k represents the value function that solves the optimal maneuvering problem with a maximum of $k+1$ maneuvers. In this way, $V^k - V^{k-1}$ represents the decrease in cost function that can be obtained by passing from k to $k+1$ ma-

neuvers. This is reiterated until the stopping condition in line 2 is satisfied. Note that loop 2-5 ends when an additional maneuver does not significantly reduce the cost function. See [34] for more details on Algorithm 5.
The numerical implementation of the above algorithm relies on the solution of HJB's equations of the form:

$$\max \left\{ \begin{array}{c} V(x) - \phi(x) \\ \lambda V(x) + \sup_{\omega \in W} \{-D(V)(x) f(x,i,\omega) - 1\} \end{array} \right\} = 0, \tag{3.8}$$

with $\phi = V^{k-1}$.

3.2.2 Numerical Implementation

In order to implement a numerical solution of (3.8), we note that there exist positive real constants L_f, M_f, such that, $\forall x_1, x_2 \in \Omega$, $\forall \omega \in W$, $i \in \{1,2\}$,

$$\|f(x_1,i,\omega) - f(x_2,i,\omega)\| \leq L_f \|x_1 - x_2\|, \quad \|f(x_1,i,\omega)\| \leq M_f,$$

so that we can apply the approximation scheme presented in section 1.2.2, based on a finite approximation of state and control spaces and a discretization in time. Roughly speaking, in (3.8) one can approximate $D(V(x))f(x,i,\omega) \simeq h^{-1}(V(x+hf(x,i,\omega)) - V(x))$, where h is a small positive real number that represents an integration time. In this way, the second of (3.8) becomes

$$(1+\lambda h)V(x) = \min_{\omega \in W} \{V(x+hf(x,i,\omega)) + h\} = 0,$$

for $x \in \Omega$ and, by approximating $(1+\lambda h)^{-1} \simeq (1-\lambda h)$, $(1+\lambda h)^{-1} h \simeq h$ one ends up with the following discrete time HJB equation

$$V_h(x) = \min \left\{ \begin{array}{r} \phi(x) \\ \min\limits_{\omega \in W} \{(1-\lambda h) V_h(x+hf(x,i,\omega)) + h)\} \end{array} \right\}, \tag{3.9}$$

with $x \in \Omega$. For a more rigorous derivation of (3.9) see [20].

A grid is computed on a finite set of vertices $S = \{x_l\} \subset \Omega$, $l = 1, \dots, p$. Evaluating (3.9) at $x \in S$, we obtain

$$V_h(x_l) = \min \left\{ \begin{array}{r} \phi(x_l) \\ \min\limits_{\omega \in W} \{(1-\lambda h) V_h(x_l + hf(x_l, i, \omega)) + h)\} \end{array} \right\}, \tag{3.10}$$

with $l = 1, \dots, p$. Note the dependence of the value cost function on the choice of the integration step h. Using the grid, function V can be approximated by a linear function of the finite set of variables $V_h(x_l)$, $l = 1, \dots, p$.

By Theorem 2.1 of Appendix A of [20], if $\lambda > L_f$ and $h \in (0, \frac{1}{\lambda}]$, system (3.10) has a unique solution that converges uniformly to the solution of (3.8) as $h, d, \frac{d}{h}$ tend to 0, where $d = \max_l diam(S_l)$ and S_l is the set of all the hyper-rectangles formed by the vertices of the grid. To further simplify (3.10), it is possible to discretize the control space, substituting W with a finite set of controls $\{\omega_1, \dots, \omega_m\}$, so that we can replace (3.10) with

$$V_h(x_l) = \min \left\{ \begin{array}{r} \phi(x_l), \\ \min\limits_{\omega_j} \{(1-\lambda h) V_h(x_l + hf(x_l, i, \omega_j)) + h)\} \end{array} \right\}, \tag{3.11}$$

where $l = 1, \dots, p$ and $j = 1, \dots, m$. Figure 4.2 illustrates a step of construction of the right-hand side of problem (3.11). Namely, for each node of the grid x_l and each value of the control ω_j, all end points $x_l + hf(x_l, i, \omega_j)$ of the Euler approximation of the solution of (3.3) from the initial state x_l are computed. The value cost function for these end points is given by a multi-linear combination of its values on the grid vertices.

Set vector $z := [V_h(x_1), V_h(x_2), \dots, V_h(x_p)]$, in this way $z \in \mathbb{R}^p$ represents the value of the cost function at grid points.

Assume the cost function ϕ is non-negative, for each x_l, ω_j, the right-hand side of (3.11) is affine with respect to z, so that problem (3.11) can be rewritten in form

$$z = \min_{j=1,\dots,m} \{\phi(x_l), \quad \min\{A_j z + b_j\}\} \tag{3.12}$$

where for $j = 1, \dots, m$, $A_j \in \mathbb{R}_+^{p \times p}$ are suitable nonnegative matrices and $b_j \in \mathbb{R}_+^p$ are suitable nonnegative vectors.

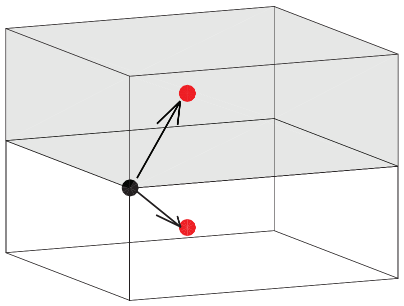


Figure 3.2: Approximation of the HJB equation on a grid with two controls. For each node x_l (depicted in black) the controls ω_j are applied to obtain the end points $x_l + hf(x_l, i, \omega_j)$ (depicted in red).

Let $w \in \mathbb{R}^n$, define map $M : \mathbb{R}^p_+ \to \mathbb{R}^p_+$ as

$$M_w(z) := \min\{w, \quad \min\{A_j z + b_j\}\}, \tag{3.13}$$

using the results of [20] it can be shown that M is a contraction, so that (3.12) can be solved as a fixed point iteration. Namely, setting

$$\begin{cases} x(s+1) = M_{\phi(x_l)}(x(s)), \\ x(0) = x_0\,, \end{cases} \tag{3.14}$$

the solution z of (3.12) is obtained as $z = \lim_{s\to\infty} x(s)$, for any initial condition x_0. Note that in (3.13) matrixes A_j and vectors b_j depend on subsystem $i \in \{1,2\}$. So for any $w \in \mathbb{R}^p$ and $i \in \{1,2\}$, we define map $M_{w,i} : \mathbb{R}^p_+ \to \mathbb{R}^p_+$ as

$$M_{w,i}(z) := \min\{w, \quad \min\{A^i_j z + b^i_j\}\}.$$

The numerical implementation for solving system (3.7) is presented in Algorithm 3. In lines 1-7 of Algorithm 3 the value function V^0 is computed by a fixed point iteration with error tolerance ε. In lines 8-15, V^k is computed by contraction (3.14). Subsystems (3),(4), alternate until the chosen maximum number of maneuvers k_{max} is reached.

Algorithm 3 Numerical algorithm for solving (3.7)

```
    Function NumericalSolver ($\Gamma, i_1, K_{max}, \varepsilon$)
    Input: $\Gamma$: target states set, $i_1$: target configuration, $K_{max}$: maximum number of maneuvers, $\varepsilon$: termination tolerance,
    Output: $V^0, \ldots, V^{K_{max}}$: cost functions.
 1: $set\ V^0(x_l) = 0 \quad for\ each\ x_l \in \Gamma$
 2: $set\ V^0(x_l) = \frac{1}{\lambda} \quad for\ each\ x_l \notin \Gamma$
 3: $s = 0$
 4: while $|V^0(s) - V^0(s-1)|_\infty > \varepsilon$ do
 5:         $V^0(s+1) = M_{V^0(0),i_1}(V^0(s))$
 6:         $s = s+1$
 7: $V^0 = V^0(s)$
    $i = i_1$
 8: for $k = 1, \ldots, K_{max}$ do
 9:     $i = mod(i+1,2)+1$
10:     $V^k(0) = V^{k-1}$
11:     $s = 0$
12:     while $|V^k(s) - V^k(s-1)|_\infty > \varepsilon$ do
13:         $V^k(s+1) = M_{V^k(0),i}(V^k(s))$
14:         $s = s+1$
15:     $V^k = V^k(s)$
    return $\{V^{0,}, \ldots, V^{K_{max}}\}$
```

3.3 Feedback controls

In the solution of the car maneuvering problem, the optimal trajectory that reaches the final state set Γ may end with either a forward maneuver ($i_1 = 1$) or with a reverse one ($i_1 = 2$). Thus, for any initial state x_0, we define the cost function $T^* : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ as

$$T^*(x_0) = \min_{i \in \{1,2\}} T(x_0, i). \tag{3.15}$$

Define the optimal final condition as $i^* = \{i \in \{1,2\} | T(x_0,i) = T^*(x_0)\}$ and let $\{V^{*,0}, \dots, V^{*,K_{max}}\}$ be the solutions of Algorithm 3. Assuming $i^* = 1$, the synthesis of feedback controls for problem (3.15) starting from state x_0 is given by Algorithm 4. If $i^* = 2$ the algorithm to solve problem (3.15) is the same except for line 4, where i is set as $mod(k+1,2)+1$.

Algorithm 4 Synthesis of feedback controls for system (3.1),(3.2)

```
   Function FeedbackControls ($\{V^{*,0},\dots,V^{*,K_{max}}\}, x_0, \varepsilon$)
   Input: $\{V^{*,0},\dots,V^{*,K_{max}}\}$: value functions returned by Algorithm 3, $x_0$: start state, $\varepsilon$: distance tolerance,
   Output: $\Omega$: vector of controls $\omega$, $I$: vector of the indexes of the subsystems.
   $x = x_0$
   $k = K_{max}$
   $l = 0$
1: while $dist(x_1, x_0) > \varepsilon$ do
2:     while $interp(V^{*,k}, x) > interp(V^{*,k-1}, x)$ do
3:         $k = k - 1$
4:     $i = mod(k,2) + 1$
5:     $j^* = \underset{j}{\operatorname{argmin}}\{interp(V^{*,k}, x + hf(x, i, \omega_j))\}$
6:     $x = x + hf(x, i, \omega_{j^*})$
7:     $\Omega[l] = \omega_{j^*}$
8:     $I[l] = i$
9:     $l = l + 1$
   return $[W, I]$
```

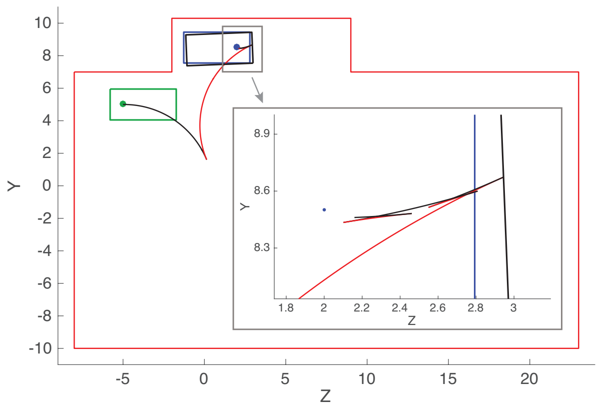
Starting from initial state x_0, Algorithm 4 performs the synthesis of feedback controls, and returns the vector Ω of controls and the vector of system configurations I that allow reaching the target state x_1. In loop 1-9, the synthesis of feedback controls is obtained by a reiterated Euler integration step until the distance between the current and target state is less than the allowed error ε. In the internal loop 2-3, the current value function is updated. Note that function $interp(V,x)$ evaluates the value cost function at x as a multi-linear interpolation of the vector V of the values of the cost function on grid vertices.

3.4 Numerical Tests

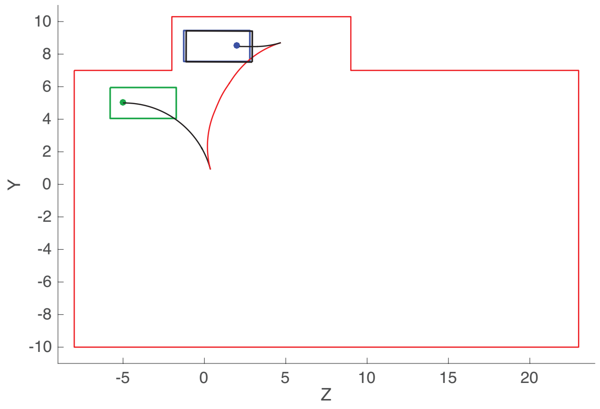
Referring to the car-like model (see Figure 1.1), set $\delta_{max} = 0.44$ *rad* and $l = 2.57$ *m*. Setting $v^- = -1$ and $v^+ = 1$ in subsystems (3.1), (3.2), the maximum angular velocity is given by $\frac{1}{l}v^+ \tan\delta_{max} = 0.183\ s^{-1}$, so that W = $\{-0.183, 0.183\}$. Let x_1 a target state and let Γ be an ellipsoid centered at x_1 with semi axes $r_z = r_y = 0.12\ m$ and $r_\theta = 0.08$.
Figure 3.3 shows the path computed by Algorithm 4 for a parallel parking maneuver scenario using a different number of maximum maneuvers K_{max}. In Algorithm 3, we used 525231 vertices to approximate the torus $\Omega = [-8,23]\times[-10,10]\times[0,2\pi)$, with target state $x_1 = (2, 8.5, \pi)$, and target configuration $i_1 = 1$. Note that, decreasing K_{max}, the number of maneuvers decreases while the shape of the path remains approximately the same (Figures 3.3a, 3.3b) until, for $K_{max} = 1$, a single forward maneuver is planned with a significant increase of path length (Figure 3.3c). A similar consideration can be made for the perpendicular parking maneuver scenario of Figure 3.4. In this case a grid with 534216 vertexes is computed over $\Omega = [-20,20]\times[-15,15]\times[0,2\pi)$, with target state $x_1 = \left(9.55, -10.4, \frac{\pi}{2}\right)$, and target configuration $i_1 = 1$.

3.4.1 Computational Time

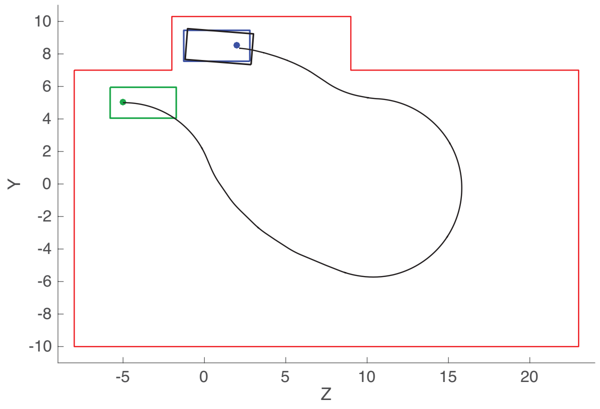
Consider a C++ implementation for Algorithms 3 and 4 running on Intel Core i7-3920XM CPU at 2.90 GHz processor with 16 GB RAM. Referring to the parallel parking scenario described in Figure 3.5a, Algorithm 3 is solved on a grid of 72971



(a) $K_{max} = 20$, M = 7, L = 17.04 *m*.

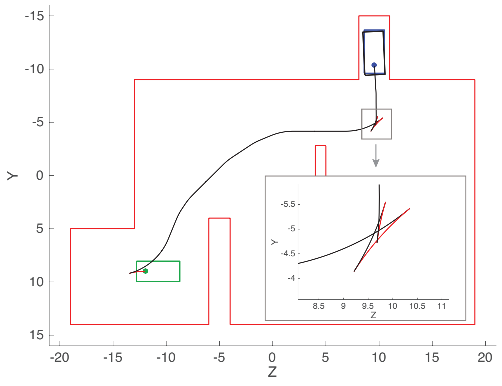


(b) $K_{max} = 10$, M = 3, L = 19.56 *m*.

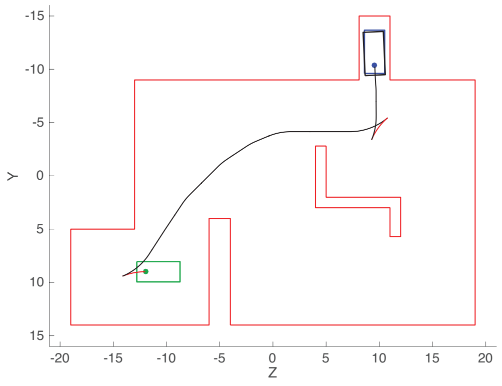


(c) $K_{max} = 1$, M = 1, L = 45.83 *m*.

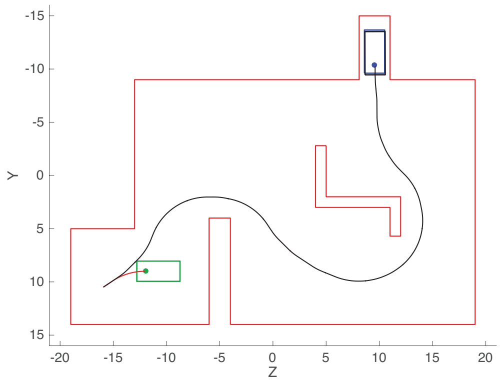
Figure 3.3: A parallel maneuvering scenario with obstacles whose edges are represented with red segments. The reverse maneuvers are depicted with red curves, while the reached final state with black polygons. Parameters: $\varepsilon = 10^{-2}$, $\lambda = 0.01$.



(a) $K_{max} = 20$, M = 6, L = 41.17 m.



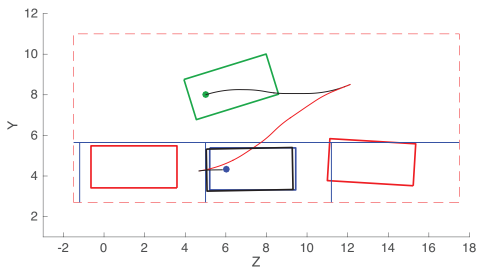
(b) $K_{max} = 10$, M = 4, L = 42.62 m.



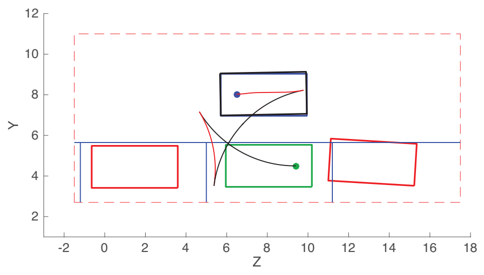
(c) $K_{max} = 2$, M = 2, L = 59.65 m.

Figure 3.4: A perpendicular parking scenario. The paths have been computed with different values of K_{max} on a grid of 534216 vertexes on $\Omega = [-20,20] \times [-15,15] \times [0,2\pi)$, with target state $x_1 = \left(9.55, -10.4, \frac{\pi}{2}\right)$ and starting state $x_0 = (-12,9,0)$. Parameters: $\varepsilon = 10^{-2}$, $\lambda = 0.01$.

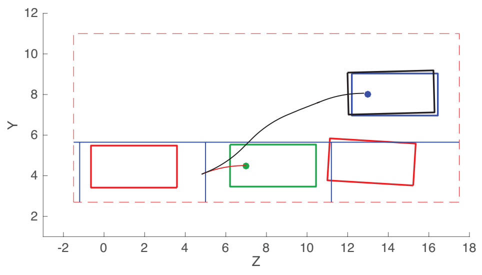
vertexes over $\Omega = [-2,18] \times [2,11] \times [0,2\pi)$, target state $x_1 = (6,4.35,0)$, target configuration $i_1 = 2$ and maximum number of maneuvers $K_{max} = 4$. The total computational time, including the resolution of Algorithm 4 with starting state $x_0 = (5,8,0.3)$, is 5.2894 s. Note that the execution time is proportional to the chosen maximum number of maneuvers K_{max}. For instance, Figure 3.5b shows a parking exit for the same scenario with target state $x_1 = (6.5,8,0)$, target configuration $i_1 = 2$ and $K_{max} = 6$. Here the resolution of the problem takes a total computational time of 7.5087 s. Moreover, the computational time is proportional to the dimension of the discrete space Ω_l. In fact, solving Algorithm 3 by a grid of 17830 vertices for the simple parking exit maneuver of Figure 3.5c the total time is reduced to 0.2584 s.



(a) $x_1 = (6,4.35,0)$, $i_1 = 2$, $x_0 = (5,8,0.3)$. Parameters: $\varepsilon = 10^{-4}$, $\lambda = 0.05$, $K_{max} = 4$.



(b) $x_1 = (6.5,8,0)$, $i_1 = 2$, $x_0 = (9.4,4.5,\pi)$. Parameters: $\varepsilon = 10^{-4}$, $\lambda = 0.05$, $K_{max} = 6$.



(c) $x_1 = (13,8,0)$, $i_1 = 1$, $x_0 = (7,4.5,0)$. Parameters: $\varepsilon = 10^{-4}$, $\lambda = 0.02$, $K_{max} = 2$.

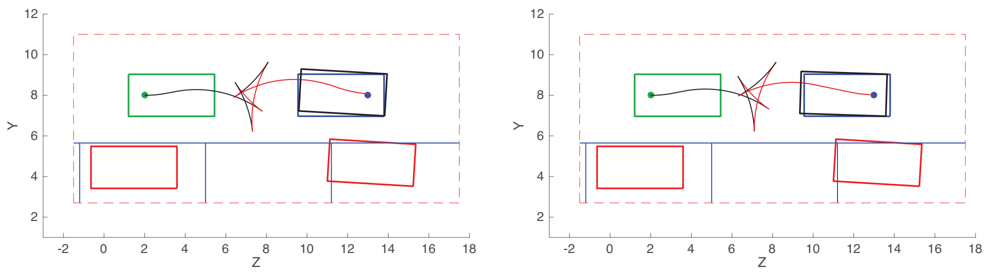
Figure 3.5: A parallel parking scenario. The red dotted lines represents the edges of the free space, the red polygons correspond to the obstacles.

Table 3.1 reports the computational time for solving the parking scenarios discussed above (Figures 3.5a-3.5c), on two different hardware architectures. Now consider to

Table 3.1: Computational time for solving the parking scenarios of Figures 3.5a-3.5c) using two different hardware architectures.

Hardware	a	b	c
2.90 GHz Core i7, 16 GB RAM	5.2894 s	7.5087 s	0.2584 s
2.60 GHz Core i5, 8 GB RAM	7.039 s	9.1622 s	0.5729 s

work on a 2.60 GHz Intel Core i5 processor with 8 GB RAM. In Figure 3.6 an inversion maneuvering scenario with 2 different values of λ is solved. With this example we highlight the fact that the computational time also depends on the choice of this parameter. In fact, using a grid of 72971 vertexes over $\Omega = [-2,18] \times [2,11] \times [0,2\pi)$, the computational time for $\lambda = 0.2$ and $\lambda = 0.07$ is 4.5406 s and 6.9819 s respectively.



(a) $\lambda = 0.2$. (b) $\lambda = 0.07$.

Figure 3.6: An inversion maneuvering scenario. Target state $x_1 = (13, 8, \pi)$, target configuration $i_1 = 2$ and starting state $x_0 = (2, 8, 0)$. Parameters: $\varepsilon = 10^{-4}$, $K_{max} = 6$.

Chapter 4

Path planning by optimal concatenation of arcs and segments

In chapter 3 we discussed the importance in autonomous parking of preferring a simplest maneuver to a shortest but more complex one. With respect to the previous chapter, this chapter presents a determinist method to find a path that is composed of a concatenation of lines and arc segments of constant curvature. This method models the vehicle with a switched system, composed of 6 autonomous systems, where each ones generates a trajectory with constant curvature and velocity. The algorithm finds the trajectory that minimizes a cost function that takes into account the length of the parking maneuver and a penalty related to the number of switchings. The choice of this penalty allows taking into account the number of changes of direction and steering angle in the selection of the parking maneuver.
As in chapter 3, to numerically solve this problem, we leverage some recent results on the optimal control of switching systems. In particular:

- We use the results presented in [33], that show that the cost function corresponds to the viscosity solution of a system of quasi variational inequalities (QVIs).

- Using the method presented in [34], we convert the problem into the solution of a sequence of decoupled QVIs.
- Each QVI is solved with the finite element method presented in [35].

4.1 Problem Statement

Let $Q \subset \left(\mathbb{R}^2 \times [0,2\pi[\right)$ be a bounded and connected domain, which represents the operating space of the vehicle. This space is partitioned as $Q = Q_{free} \cup Q_{obst}$, with $Q_{free} \cap Q_{obst} = \varnothing$, where Q_{free} is the free space and Q_{obst} is the subset of the operating space covered by obstacles. Consider the kinematic car-like model with rear-wheel drive defined by system:

$$\begin{cases} \dot{z} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega, \end{cases}$$

where the triplet $x = (z, y, \theta)$ represents a configuration of the vehicle and the control input is given by $u = (v, \omega)$; and v and ω are, respectively, the linear and angular velocities. The angular velocity is related to the front wheel steering angle δ by relation $\omega = \frac{1}{l} v \tan\delta$. The input variables are constrained as follows

$$v_{min} < v < v_{max}, \qquad \omega_{min} < \omega < \omega_{max}.$$

We model the car-like vehicle with the switched system $\dot{x}(t) = f(x(t), i)$, where $f : Q \times \{1,2,3,4,5,6\} \to Q$ is defined by the following autonomous systems:

$$f(x,1) = \begin{pmatrix} v_+(x)\cos\theta \\ v_+(x)\sin\theta \\ \Omega_+ \end{pmatrix}, \tag{4.1}$$

$$f(x,2) = \begin{pmatrix} v_+(x)\cos\theta \\ v_+(x)\sin\theta \\ 0 \end{pmatrix}, \tag{4.2}$$

$$f(x,3) = \begin{pmatrix} v_+(x)\cos\theta \\ v_+(x)\sin\theta \\ \Omega_- \end{pmatrix}, \tag{4.3}$$

$$f(x,4) = \begin{pmatrix} v_-(x)\cos\theta \\ v_-(x)\sin\theta \\ \Omega_- \end{pmatrix}, \tag{4.4}$$

$$f(x,5) = \begin{pmatrix} v_-(x)\cos\theta \\ v_-(x)\sin\theta \\ 0 \end{pmatrix}, \tag{4.5}$$

$$f(x,6) = \begin{pmatrix} v_-(x)\cos\theta \\ v_-(x)\sin\theta \\ \Omega_+ \end{pmatrix}, \tag{4.6}$$

where $v_+, v_- : Q \to \mathbb{R}$ are defined by

$$v_+(x) = \begin{cases} V_+ r & \text{if } x \in Q_{obst}, \\ V_+ & \text{otherwise }, \end{cases}$$

$$v_-(x) = \begin{cases} V_- r & \text{if } x \in Q_{obst}, \\ V_- & \text{otherwise }. \end{cases}$$

and $0 < r << 1$ is a small number. Here, $V_- < 0 < V_+$ are the speeds associated to forward and reverse gears, $\Omega_- < 0 < \Omega_+$ are the angular velocity associated to a full right or left steering, and $i \in \{1,2,3,4,5,6\} = \mathbb{I}$ denotes the configuration of the vehicle. Namely, $i = 1$ is associated to forward gear with a full right steering, $i = 2$ to forward gear with the angle of steering set to zero and so forth . Note that, if $x \in Q_{obst}$, the velocity of the vehicle is reduced by factor r. Being r close to zero, the vehicle is very slow when traveling on obstacles. Since we will be interested in minimum time trajectories, for small values of r optimal trajectories will not intersect set Q_{obst} for relevant intervals of time.

The control signal is given by $\sigma \in (\mathbb{R}, \mathbb{I})^{\star} = A$, that represents the sequence of configuration changes so that if $(t_k, i_k) \in \sigma$, then the controller switches to subsystem i_k at time t_k. The sequence of switches satisfies:

$$\begin{aligned} &0 = \tau_1 < \tau_2 < \dots < \tau_K, \\ &i_{k-1} \neq i_k, \quad i_k \neq \eta(i_{k-1}), \\ &2 \leq k \leq K. \end{aligned} \tag{4.7}$$

Here $\eta : \mathbb{I} \to \mathbb{I}$ is the function such that $\eta(i)$ is the subsystem that has opposite direction and same steering angle of subsystem i:

$$\eta(i) = \begin{cases} 6 & \text{if } i = 3, \\ mod(i+3, 6) & \text{otherwise .} \end{cases}$$

In this way we avoid that the sequence contains subsequences of the form $i, \eta(i)$. In fact, these subsequences generate a path with duplicate states (z, y, θ) and, for this reason, are not part of the optimal control signal.
Define function $I : \mathbb{R} \to \mathbb{I}$ as $I(t) = i_{\bar{k}}(t)$, where $\bar{k}(t) = max\{k \in \mathbb{N} : \tau_k \leq t\}$ represents the total number of switchings occurring in interval $[0, t]$. Let the control signal $\sigma \in A$, the trajectory of the switched system (4.1)-(4.6) is defined as the solution of equation:

$$\begin{cases} \dot{x}(t) = f(x(t), I(t)) \\ x(0) = x_0, \end{cases} \tag{4.8}$$

and is denoted by $y_{x_0,\sigma}(t) = x(t)$.

Define function $p : \mathbb{I} \times \mathbb{I} \to \mathbb{R}^+$ as $p(i, j) = P_{i,j}$, such that:

$$p(i, j) > 0, \quad p(i, i) = 0,$$

for all $i,\ j,\ \in \mathbb{I},\ i \neq j,\ j \neq \eta(i)$. The positive constant $P_{i,j}$ represents the cost for switching from subsystem i to subsystem j. For instance, Figure 4.1 shows all the possible transitions and the related costs for subsystem $i = 2$.
Consider a target set $\Gamma \subset Q$, closed and such that $int(\Gamma) \neq \varnothing$, we define a cost function $t : Q \times \mathbb{I} \times A \to \mathbb{R} \cup \{+\infty\}$ that associates to each initial state x_0, and control $\sigma \in A$

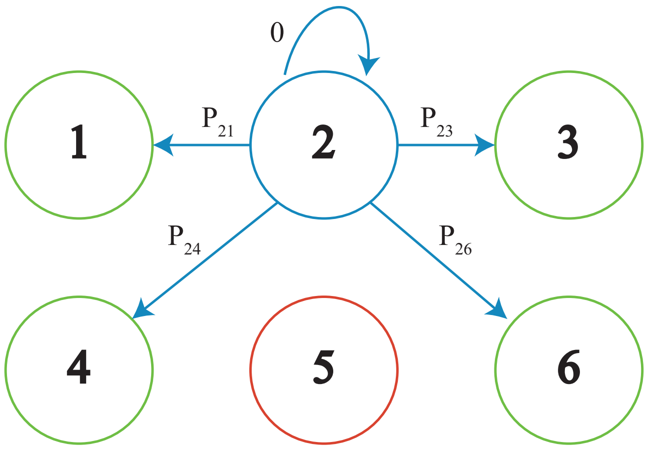


Figure 4.1: Reachable subsystems (depicted in green) starting from subsystem $i = 2$ and related costs.

the cost

$$t(x_0, i, \sigma) = \begin{cases} +\infty & \text{if } y_{x_0,\sigma}(t) \notin \Gamma, \forall t > 0, \\ \inf_{t \in \mathbb{R}^+} \left\{ t + \sum_{k=1}^{\bar{k}(t)} p(i_{k-1}, i_k) \ : y_{x_0,\sigma}(t) \in \Gamma \right\} & \text{otherwise}, \end{cases} \tag{4.9}$$

where we consider $t(x_0, i) = +\infty$ if the solution never reaches set Γ for any $t > 0$. In this way, the value function $T : Q \times \mathbb{I} \to \mathbb{R} \cup \{+\infty\}$ for problem (4.9) is given by

$$T(x_0, i) = \inf_{\sigma \in A} t(x_0, i, \sigma) \tag{4.10}$$

and represents the first time of arrival on the target Γ, incremented by the sum of all the switching costs occurred in interval $[0, t]$. We solve problem (4.10) via dynamic programming by characterizing the value function in terms of a first order Hamilton-Jacobi-Bellman (HJB) equation. To this end we use the Kružkov transformation (1.8) in order to obtain the following boundary problem for (4.10):

$$V(x_0,i) = \begin{cases} \frac{1}{\lambda} & \text{if } T(x_0,i) = +\infty, \\ \frac{1}{\lambda} - \frac{1}{\lambda} e^{-\lambda T(x_0,i)} & \text{otherwise ,} \end{cases}$$

where λ is a positive scalar. It can be proved that $V : Q \times \mathbb{I} \to \mathbb{R} \cup \{+\infty\}$ is itself a value function, that is

$$V(x_0,i) = \inf_{\sigma \in A} J_{x_0,i,\sigma}(t), \tag{4.11}$$

where

$$J(x_0,i,\sigma) = \int_0^{t(x_0,i,\sigma)} e^{-\lambda t} dt.$$

In particular, if $t = t(x_0,i,\sigma)$ it can be proved that

$$J(x_0,i,\sigma) = \int_0^t e^{-\lambda t} dt + \sum_{k=1}^{\bar{k}(t)} p(i_{k-1},i_k).$$

4.2 Problem Resolution

4.2.1 Analytical Analysis

As already mentioned in section 4.1, problem (4.11) can be addressed via dynamic programming. In particular, it falls into the class of optimal switching control problems which was formulated and studied in [33]. Using those results, it can be proved that the value function (4.11) is uniformly bounded, Hölder continuos, and is the unique viscosity solution of the following HJB equation:

$$\max \left\{ \begin{array}{c} V(x,i) - \min\limits_{\bar{i} \neq \{i,\eta(i)\}} \{p(i,\bar{i}) + V(x,\bar{i})\} \\ \lambda V(x,i) - D(V)(x,i) f(x,i) - 1 \end{array} \right\} = 0, \tag{4.12}$$

where $V \in C(Q \times \mathbb{I}, [0,+\infty))$, $x \in Q \subset \mathbb{R}^3$, $i \in \mathbb{I}$ and $D(V)(x,i)$ denotes the gradient of function V at x.

Leveraging the results shown in [34], we obtain Algorithm 5 for solving system (4.12) with a sequence of decoupled QVIs.

Algorithm 5 Algorithm for solving problem (4.12)

```
   **Function** Solver $(\Gamma, \varepsilon)$
   **Input**: $\Gamma$: target state set, $\varepsilon$: termination tolerance.
   **Output**: $V_1, \dots, V_6$: cost functions.
   *for each* $i \in \mathbb{I}$ *set boundary conditions for* $V_i^0$ *according to* $\Gamma$
   $k = 0$
1: **while** for some $i \in \mathbb{I} : \|V_i^k - V_i^{k-1}\|_\infty > \varepsilon$ **do**
2:     $k = k + 1$
3:     *for each* $i \in \mathbb{I}$ *compute* $V_i^k$ *by*
       $$\max \left\{ \begin{array}{c} V_i^k(x) - \min_{\bar{i} \neq \{i, \eta(i)\}} \{p(i, \bar{i}) + V_{\bar{i}}^{k-1}(x)\} \\ \lambda V_i^k(x) - D(V_i^k)(x) f(x, i) - 1) \end{array} \right\} = 0.$$
   *for each* $i \in \mathbb{I}$ *set* $V_i = V_i^k$
   **return** $\{V_1, \dots, V_6\}$
```

In loop 1-3 of Algorithm 5, V_i^k is computed so that, for any x such that $V_i^k(x) < \min_{\bar{i} \neq \{i, \eta(i)\}} \{P_{i,\bar{i}} + V_{\bar{i}}^{k-1}(x)\}$, V_i^k is the solution of the HJB equation applied to subsystem i. In this way, $V_1, V_2, \dots, V_6$ are the output functions such that V_i represents the value function that solves the optimal maneuvering problem with subsystem i taking into account the penalties of switching to all the subsystems $\bar{i} \neq \{i, \eta(i)\}$. In particular $V_i - (P_{i,\bar{i}} + V_{\bar{i}})$ represents the decrease in cost function that can be obtained by passing from subsystem i to $\bar{i}$. This is reiterated until the stopping condition in line 1 is satisfied (see [34] for more detail about Algorithm 5).
The numerical implementation of the above algorithm relies on the solution of HJB's equations of the form:

$$\max \left\{ \begin{array}{c} V(x,i) - \phi(x,i) \\ \lambda V(x,i) - D(V)(x,i) f(x,i) - 1 \end{array} \right\} = 0, \tag{4.13}$$

with $\phi_i = \min_{\bar{i} \neq \{i,\eta(i)\}} \{P_{i,\bar{i}} + V_{\bar{i}}^{k-1}\}$.

4.2.2 Numerical Implementation

In order to implement a numerical solution of (4.13), we apply the approximation scheme presented in section 1.2.2, based on a finite approximation of state space and a discretization in time. In (4.13) we approximate $D(V(x,i))f(x,i) \simeq h^{-1}(V(x + hf(x,i),i) - V(x,i))$, where h is a small positive real number that represents an integration time. In this way, the second of (4.13) becomes

$$(1+\lambda h)V(x,i) = V(x+hf(x,i),i) + h = 0,$$

for $x \in Q$ and, by approximating $(1+\lambda h)^{-1} \simeq (1-\lambda h)$, $(1+\lambda h)^{-1}h \simeq h$ one ends up with the following discrete time HJB's equations

$$V_h(x,i) = \min \left\{ \begin{array}{r} \phi(x,i) \\ (1-\lambda h)V_h(x+hf(x,i),i)+h \end{array} \right\}, \tag{4.14}$$

with $x \in Q$ and $i \in \mathbb{I}$. For a more rigorous derivation of (4.14) see [20].

A grid is computed on a finite set of vertices $S = \{x_l\} \subset Q$, $l = 1,\dots,L$. Evaluating (4.14) at $x \in S$, we obtain

$$V_h(x_l,i) = \min \left\{ \begin{array}{r} \phi(x_l,i) \\ (1-\lambda h)V_h(x_l+hf(x_l,i),i)+h \end{array} \right\}, \tag{4.15}$$

with $l = 1,\dots,L$ and $i \in \mathbb{I}$. Note the dependence of the value cost function on the choice of the integration step h. Using the grid, function V_i can be approximated by a linear function of the finite set of variables $V_h(x_l,i)$, $l = 1,\dots,L$. Figure 4.2 illustrates a step of construction of the right-hand side of problem (4.15). Namely, for each node of the grid x_l, all end points $x_l + hf(x_l,i)$ of the Euler approximation of the solution of (4.8) from the initial state x_l are computed. The value cost function for these end points is given by a multi-linear combination of its values on the grid vertices.

For each $i \in \mathbb{I}$ set vector $z_i := [V_h(x_1,i), V_h(x_2,i), \dots, V_h(x_L,i)]$, in this way $z_i \in \mathbb{R}^L$ represents the value of the cost function on grid points for subsystem i.

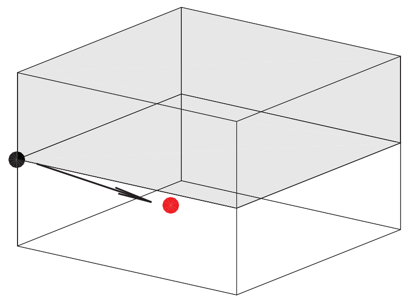


Figure 4.2: Approximation of the HJB equation on a grid for an autonomous subsystem i. For each node x_l (depicted in black) an Euler integration step is applied to obtain the end points $x_l + hf(x_l, i)$ (depicted in red).

Assume the cost function ϕ_i is non-negative, for each $x_l, i \in \mathbb{I}$, the right-hand side of (4.15) is affine with respect to z_i, so that problem (4.15) can be rewritten in form

$$z_i = \min\{\phi_i(x_l), \quad A_i z_i + b_i\} \tag{4.16}$$

where for $i \in \mathbb{I}$, $A_i \in \mathbb{R}_+^{L\times L}$ are suitable nonnegative matrices which contains the positive coefficients for subsystem i that perform the multi-linear combination on the node of the grid to obtain the value of $x_l + hf(x_l, i)$; and $b_i \in \mathbb{R}_+^L$ are suitable nonnegative vectors.
Let $w \in \mathbb{R}^L$, define map $M : \mathbb{R}_+^L \to \mathbb{R}_+^L$ as

$$M_{i,w}(u) := \min\{w, \quad A_i u + b_i\},$$

using the results of [20] it can be shown that M is a contraction, so that (4.16) can be solved as a fixed point iteration. Namely, setting

$$\begin{cases} x_i(s+1) = M_{i,\phi_i^s}(x_i(s)), \\ x_i(0) = x_0\,, \end{cases} \tag{4.17}$$

with $\phi_i^s = \min_{\bar{\imath} \neq \{i,\eta(i)\}} \{P_{i,\bar{\imath}} + x_{\bar{\imath}}(s)\}$, the solution z_i of (4.16), for each $i \in \mathbb{I}$, is obtained as $z_i = \lim_{s\to\infty} x_i(s)$, for any initial condition x_0. The numerical implementation for solving system (4.12) is presented in Algorithm 6.

Algorithm 6 Numerical algorithm for solving (4.12)

```
    **Function** NumericalSolver $(\Gamma, \varepsilon)$
    **Input**: $\Gamma$: target states set, $\varepsilon$: termination tolerance.
    **Output**: $V_1, \ldots, V_6$: cost functions associated to subsystems $i \in \mathbb{I}$.
1:  set $V_i^0(x_l) = 0$    for each $x_l \in \Gamma$, $i \in \mathbb{I}$
2:  set $V_i^0(x_l) = \frac{1}{\lambda}$    for each $x_l \notin \Gamma$, $i \in \mathbb{I}$
    $s = 0$
3:  **while** for some $i \in \mathbb{I} : |V_i^s - V_i^{s-1}|_\infty > \varepsilon$ **do**
4:      for each i $\in \mathbb{I}$ compute
5:          $\phi_i^s = \min_{\bar{i} \neq \{i, \eta(i)\}} \{P_{i,\bar{i}} + V_{\bar{i}}^s(s)\}$
6:          $V_i^{s+1} = M_{i,\phi_i^s}(V_i^s)$
    set $V_i = V_i^s$    for each i $\in \mathbb{I}$
    **return** $\{V^1, \ldots, V^6\}$
```

In lines 1,2 of the algorithm, considering the time horizon is $\left[0, \frac{1}{\lambda}\right]$, for each value function V_i the boundary conditions are set. In lines 3-6 of Algorithm 6, for each $i \in \mathbb{I}$, V_i^s is a non-increasing function associates to subsystem i that is computed by fixed point iteration (4.17) with error tolerance ε. In particular, the output V_i represents a value function that is zero for each $x_l \in \Gamma$, while for each $x_l \notin \Gamma$ represents the cost to reach the target set Γ using the subsystem i.

4.3 Feedback controls

In the solution of the car maneuvering problem, we need to find the optimal sequence of the subsystems $(4.1) - (4.6)$, occurred in time $[0,t]$, that represents the optimal control signal σ that solves problem (4.11). Note that, considering we use autonomous subsystems, the information about direction and steering of the vehicle are in the dynamics of the subsystem itself. To this end, let $\{V_1^*, \ldots, V_6^*\}$ be the solutions of Algorithm 6, the synthesis of feedback controls for problem (4.11) starting from state x_0 is given by Algorithm 7. Starting from initial state x_0, Algorithm

Algorithm 7 Synthesis of feedback controls for system (4.1)-(4.6)

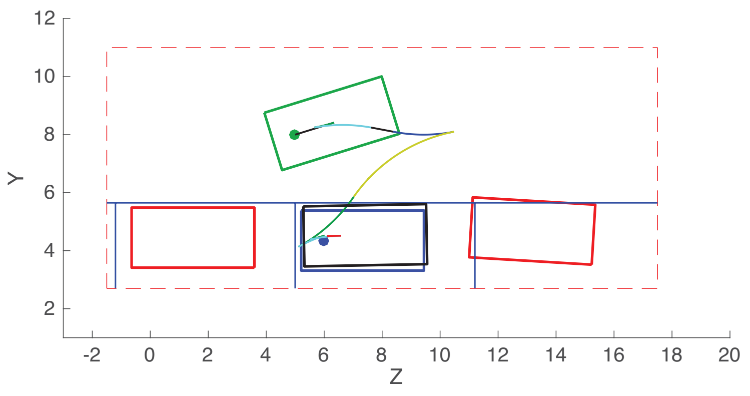
```
   Function FeedbackControls $(\{V_1^*,\dots,V_6^*\}, x_0, \varepsilon)$
   Input: $\{V_1^*,\dots,V_6^*\}$: value functions returned by Algorithm 6, $x_0$: start state $\varepsilon$: distance tolerance,
   Output: $I$: vector of the indexes of the subsystems.
   $k = 0$
   $x = x_0$
   $i^k = \underset{i \in \mathbb{I}}{\operatorname{argmin}} \{interp(V_i^*, x)\}$
   $I[k] = i^k$
1: while $dist(x_1, x_0) > \varepsilon$ do
2:     $k = k + 1$
3:     $r = \underset{i \neq \{\eta(i^{k-1}), i^{k-1}\}}{\operatorname{argmin}} \{interp(V_i^*, x) + P_{i^{k-1},i} | interp(V_i^*, x) + P_{i^{k-1},i} < interp(V_{i^{k-1}}^*, x)\}$
4:     if $r \neq NULL$ then
5:         $i^k = r$
6:     else
7:         $i^k = i^{k-1}$
8:     $x = x + hf(x, i^k)$
9:     $I[k] = i^k$
   return $I$
```

7 performs the synthesis of the feedback controls, and returns the vector of system configurations I that allow reaching the target state x_1. In loop 1-9, the synthesis of feedback controls is obtained by a reiterated Euler integration step of subsystem i^k until the distance between the current and target state is less than the allowed error ε. In line 3, the value of variable r is assigned with the index of the subsystem that allows decreasing the cost function taking into account the penalty of switching. If no one transition can reduce the cost function, r is set to NULL. In line 5 the current subsystem is updated with the subsystem r, while in line 7 the current subsystem is set with the same subsystem of the previous iteration. Note that function $interp(V,x)$ evaluates the value cost function at x as a multi-linear interpolation of the vector V of the values of the cost function on grid vertices.

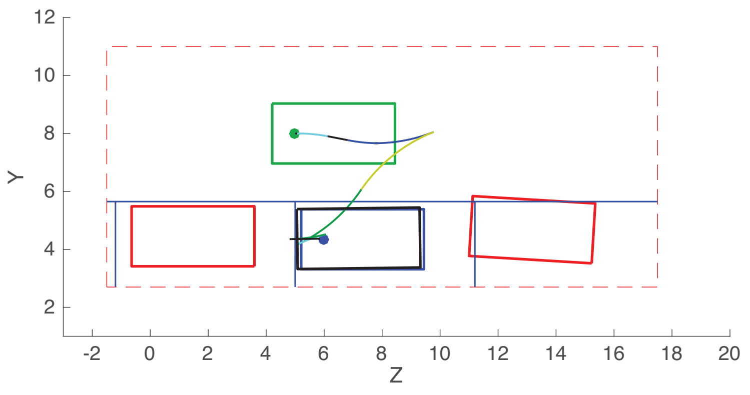
4.4 Numerical Tests

Consider a C++ implementation for Algorithms 6 and 7 running on Intel Core i7-3920XM CPU at 2.90 GHz processor with 16 GB RAM. Referring to the car-like model (see Figure 1.1), set $\delta_{max} = 0.44$ *rad* and $l = 2.57$ *m*. Setting $v^- = -1$ and $v^+ = 1$ in subsystems $(4.1),\dots,(4.6)$, the maximum angular velocity is given by $\frac{1}{l}v^+ \tan\delta_{max} = 0.183\ s^{-1}$, so that W = $\{-0.183, 0.183\}$. Let x_1 a target state and let Γ be an ellipsoid centered at x_1 with semi axes $r_z = r_y = 0.12\ m$ and $r_\theta = 0.08$.
Figures 4.3a-4.3c show some car maneuvers for different starting states for a parallel parking scenario. In these tests the penalties to switch from subsystem i to subsystem j are:

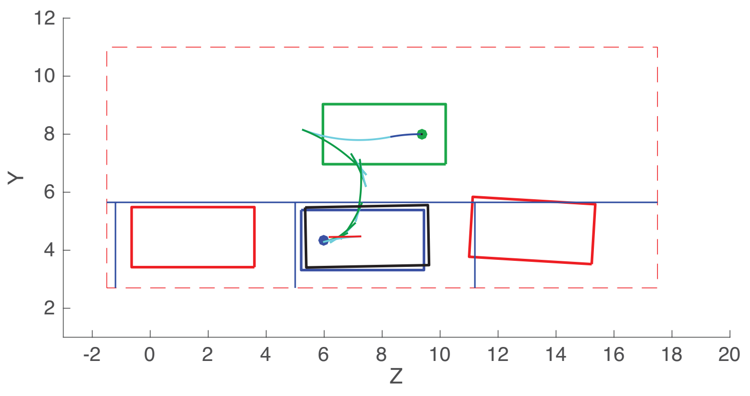
$$P_{i,j} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & - & P_{15} & P_{16} \\ P_{21} & P_{22} & P_{23} & P_{24} & - & P_{26} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} & - \\ - & P_{42} & P_{43} & P_{44} & P_{45} & P_{46} \\ P_{51} & - & P_{53} & P_{54} & P_{55} & P_{56} \\ P_{61} & P_{62} & - & P_{64} & P_{65} & P_{66} \end{bmatrix} = \begin{bmatrix} 0 & 0.1 & 0.1 & - & 0.1 & 0.1 \\ 0.1 & 0 & 0.1 & 0.1 & - & 0.1 \\ 0.1 & 0.1 & 0 & 0.1 & 0.1 & - \\ - & 0.1 & 0.1 & 0 & 0.1 & 0.1 \\ 0.1 & - & 0.1 & 0.1 & 0 & 0.1 \\ 0.1 & 0.1 & - & 0.1 & 0.1 & 0 \end{bmatrix} = A.$$



(a) $x_0 = (5, 8, 0.3)$.



(b) $x_0 = (5, 8, 0)$.



(c) $x_0 = (9.4, 8, \pi)$.

Figure 4.3: A parallel parking scenario with target state $x_1 = (6, 4.35, 0)$ and $P_{i,j} = A$. For each subsystem $(4.1) - (4.6)$ the corresponding trajectory is depicted with different colours.

Using the same and low penalty for each possible transition between subsystems (4.1)-(4.6), the result is a solution that is close to the minimum-length path. This, in some cases, produces solutions with a high number of direction changes (see Figure 4.3c). Anyway, a suitable choice of penalties $P_{i,j}$ can take into account the direction changes. For instance, Figure 4.4 shows the results for the same scenario of Figure 4.3c using the penalties:

$$P_{i,j} = \begin{bmatrix} 0 & P_{steer} & P_{steer} & - & P_{dir} & P_{dir} \\ P_{steer} & 0 & P_{steer} & P_{dir} & - & P_{dir} \\ P_{steer} & P_{steer} & 0 & P_{dir} & P_{dir} & - \\ - & P_{dir} & P_{dir} & 0 & P_{steer} & P_{steer} \\ P_{dir} & - & P_{dir} & P_{steer} & 0 & P_{steer} \\ P_{dir} & P_{dir} & - & P_{steer} & P_{steer} & \end{bmatrix} = B,$$

where $P_{steer} = 0.3$ is the penalty associated with the change of the steering angle, and $P_{dir} = 2$ represents the penalty associated with the change of direction of the vehicle. In accordance with chosen penalties $P_{i,j} = B$, the returned path is longer and presents a lower number of direction changes. The number of maneuvers is reduced to 7 from 14 of Figure 4.3c. In Figure 4.5, using $P_{dir} = 6$ the number of maneuvers is reduced to 6. In all tests Algorithm 6 is solved on a grid of 72971 vertexes over $\Omega = [-2,18] \times [2,11] \times [0,2\pi)$, target state $x_1 = (6,4.35,0)$, error tolerance $\varepsilon = 10^{-4}$ and discount factor $\lambda = 0.05$. Algorithm 6 takes a time of 3.5783 s for $P_{i,j} = A$ and 3.5231 s for $P_{i,j} = B$. Instead, Algorithms 7 takes a time in the order of milliseconds.

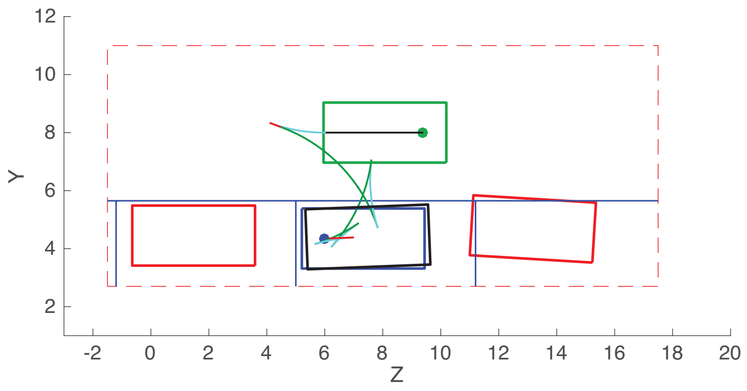


Figure 4.4: Target state $x_1 = (6, 4.35, 0)$, starting state $x_0 = (9.4, 8, \pi)$, $P_{i,j} = B$ with $P_{steer} = 0.3$ and $P_{dir} = 2$.

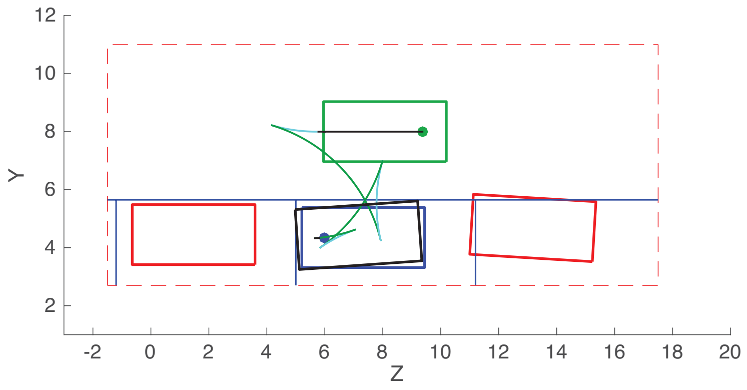


Figure 4.5: Target state $x_1 = (6, 4.35, 0)$, starting state $x_0 = (9.4, 8, \pi)$, $P_{i,j} = B$ with $P_{steer} = 0.3$ and $P_{dir} = 6$.

Chapter 5

FOCS: Fusion of Optimal Control & Search

This chapter is based on a work of Piero Micelli and Maxim Likhachev[1].

As discussed in chapter 1, many motion planning tasks in robotics can be represented as a path finding problem on a graph. To this end, the configuration space of the robot is discretized with each vertex in the graph corresponding to one of these discretized robot configurations and the motion of a robot is decomposed into a small set of short motion primitives that constitute the edges in the graph. Heuristic search algorithms such as A^* can then be used to search this graph for an optimal or close-to-optimal path from the vertex that corresponds to the current robot configuration to the vertex that corresponds to its goal configuration. This approach allows to find a solution to a planning task rather quickly, even for large and high-dimensional operating spaces by utilizing anytime heuristic search algorithms that provide real-time performance combined with rigorous sub-optimality bounds with respect to the chosen discretization. For this reason, search-based algorithms are widely used for motion planning in different domains ([39], [7] or [40]).

[1]Maxim Likhachev is with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Another approach to motion planning involves the numerical solution of the Hamilton-Jacobi-Bellman (HJB) equation. In this case, the heart of the algorithm relies in finding the optimal value function (or optimal cost-to-go function) which is the solution to the first-order differential equation (the HJB equation). This value function represents the minimal total cost for completing the task from the current configuration of the robot. The optimal solution can then be found using Dynamic Programming Principle [20]. In general, the HJB equation is a nonlinear partial differential equation, so it is computed by numerical procedures which allow to find a linear approximation to the value function ([24], [25], [26]). In contrast to search-based algorithms, optimal control methods allow finding an optimal solution to the motion planning problem but at the expense of a greater computational cost.

In this chapter, these two methods are combined in a single algorithm that we call FOCS (Fusion of Optimal Control and Search). The aim of this approach is to address the motion planning problem exploiting the advantages of Optimal Control Theory and Search-based Planning.

For instance, consider the navigation scenario shown in Figure 5.1 for a non-holonomic

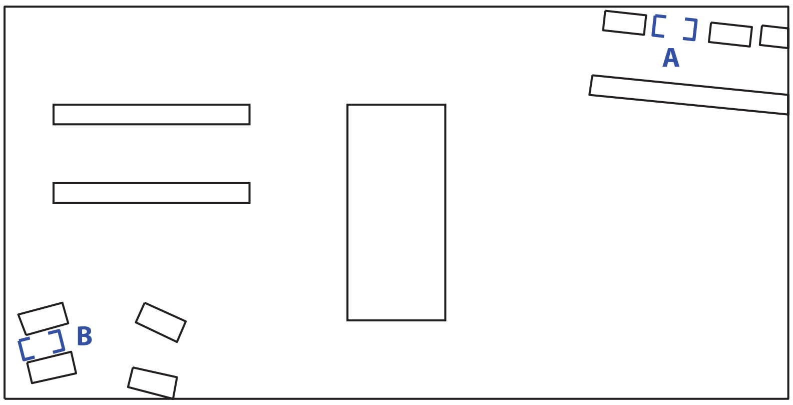


Figure 5.1: A navigation scenario with narrow space around goal configurations A and B.

robot. Here, the choice of the set of the primitives near goal configurations A or B is critical in order for the search-based algorithm to find a solution through the dense set of obstacles. On the other hand, optimal control algorithms can only find solutions in small operating spaces. This motivates us to construct an approach as follows: Search-based Planning is used to reach a small region that contains the goal configuration, the motion planning task inside the goal region is completed via Dynamic Programming-based optimal control. FOCS finds on its own how to concatenate the two portions of the path together to return a single path with guarantees on its quality. This chapter first describes algorithm FOCS and its proprieties, then shows the results of its application to navigation to find the minimum-time path for a car-like vehicle.

5.1 Problem statment

Let $\Omega \subset \mathbb{R}^n$ be a bounded and connected domain, which represents the operating space and let $\Omega^{opt} \subset \Omega$ be a region such that target state $s_{goal} \in \Omega^{opt}$. We define a lattice-based graph (see section 1.2.1) on a finite set of states in domain Ω, which we call S. In this graph the edges are the connections between these states (nodes). In this way $c(s,s')$ is the cost of the edge between nodes s and s', where $c(s,s') = \infty$ if there are no edges between s and s'. Therefore, $SUCCESSOR(s) := \{s' \in S \mid c(s,s') \neq \infty\}$, represents all successors of s and $c^*(s,s')$ denotes the cost of the optimal path from state s to s'. Moreover, we define path $\Pi : s_{start} \to s_{goal}$ as a concatenation of motion primitives of the robot that connects starting state s_{start} to goal state s_{goal}.
Now, in subset Ω^{opt}, consider to model the motion of the robot by the following ordinary differential equation:

$$\begin{cases} \dot{z}(t) = f(z(t), u(t)), \\ z(0) = z_0, \end{cases}$$

where $f : \Omega^{opt} \times U \to \Omega^{opt}$ is a continuous function, z_0 is the initial state, $u(t) \in U$ is the control input, U is a compact set of admissible controls and $y_{z_0,u}$ is a solution of the system. Here, we want to find the control function u that minimizes the cost functional

$$J_{z_0}(u) := \int_0^\infty \bar{c}(z(t), u(t))dt, \tag{5.1}$$

where $\bar{c} : \Omega^{opt} \times U \to \mathbb{R}$ is a continuous cost function. As shown in section 1.2.2, dynamic programming approach provides a method for the solution of this optimal control problem introducing the value function $\bar{v} : \Omega^{opt} \to \mathbb{R}$:

$$\bar{v}(z_0) = \inf_{u \in U} J_{z_0}(u). \tag{5.2}$$

Value function (5.2) represents the best cost value we can get from cost functional (5.1) and, as it satisfies the HJB equation, enables us to construct the optimal solution $y^*_{z_0}$ for any state in Ω^{opt}.

5.2 Algorithm FOCS

FOCS (Algorithm 8) selects the path that minimizes function $\bar{f} : S \to \mathbb{R}$:

$$\bar{f}(s) = \begin{cases} g(s) + \bar{v}(s) & \text{if } s \in \Omega^{opt} , \\ g(s) + \eta h(s) & \text{otherwise} , \end{cases}$$

where s is the last node on the path, $g(s)$ denotes the current cost of the best path from s_{start} to s, $\bar{v}(s)$ represents the optimal cost from s to s_{goal} and $h(s)$ is the heuristic function.

Given a couple of starting and goal state (s_{start}, s_{goal}) (such that $s_{goal} \in \Omega^{opt}$), inflation factor $\eta \geq 1$, value function $\bar{v}$ defined over Ω^{opt}, and heuristic function h; FOCS returns a feasible path as a concatenation of the search-based path $\Pi : s_{start} \to s$ and the optimal path $y^*_s(t)$, with $s \in \Omega^{opt}$ (line 2).

Algorithm 8 uses OPEN and CLOSED lists for keeping track of the frontier states and of the expanded states, respectively. In Loop 13-26, node s^* represents the last node on the current best path. In for loop 14-23, each successor s' of s^* is evaluated: if s' is a new node, in lines 16,17 variable g is initialized to infinity and variable *optimal* to false. Variable *optimal* is set to true if state s' is part of Ω^{opt} (line 19).

Algorithm 8 FOCS

```
    Input: $s_{goal}$: goal state, $s_{start}$: starting state, $\eta \geq 1$: inflation factor.
    Output: $[\Pi : s_{start} \to s, \quad y_s^*]$: $s \in \Omega^{opt}$.
 1: procedure PATH($s$)
 2:     return $[\Pi : s_{start} \to s, \quad y_s^*]$
 3: procedure PRIORITY($s$)
 4:     if $optimal(s)$ then
 5:         return $g(s) + \bar{v}(s)$
 6:     else
 7:         return $g(s) + \eta h(s)$
 8: procedure MAIN( )
 9:     $g(s_{goal}) = \infty$ $g(s_{start}) = 0$
10:     $OPEN = CLOSED = \varnothing$
11:     insert $s_{start}$ into $OPEN$ with PRIORITY$(s_{start})$
12:     $s^* = s_{start}$
13:     while not $optimal(s^*)$ do
14:         for each $s' \in SUCCESSOR(s^*)$ do
15:             if $s'$ was not visited before then
16:                 $g(s') = \infty$
17:                 $optimal(s') =$ false
18:             if $s' \in \Omega^{opt}$ then
19:                 $optimal(s') =$ true
20:             if $g(s') > g(s^*) + c(s^*, s')$ then
21:                 $g(s') = g(s^*) + c(s^*, s')$
22:                 if $s' \notin CLOSED$ then
23:                     insert $s'$ into $OPEN$ with PRIORITY$(s')$
24:         $s^* = \underset{s \in OPEN}{\text{argmin}}\{\text{PRIORITY}(s)\}$
25:         remove $s^*$ from $OPEN$
26:         $CLOSED = CLOSED \cup s^*$
27:     return PATH$(s^*)$
```

In line 21 value $g(s')$ is updated if the path through s^* has a lower cost and, if s' has not yet been expanded, the node is stored in OPEN. In this way, at each iteration, node s^* is updated with the node in OPEN that minimizes function $\bar{f}$ (line 24). FOCS ends when a node that is part of Ω^{opt} is expanded (line 13).

5.2.1 Theoretical Analysis

Using an admissible and consistent heuristic, all the theoretical guarantees for WA* hold for FOCS.

Theorem 4. *Let $a \in \Omega^{opt}$ a state such that the terminal condition in Algorithm 8 (line 13) is satisfied, then we have that:*

$$\operatorname*{argmin}_{s \in \text{OPEN}} \bar{f}(s) = g(a) + \bar{v}(a) \leq \eta c^*(s_{start}, s_{goal}). \tag{5.3}$$

In other words, the cost of the solution returned by FOCS is no greater than η times the cost of the optimal solution returned by A*.

Proof. To prove it by contradiction we suppose that there exists a path $\Pi^* : s_{start} \to s_{goal}$ such that:

$$g(a) + \bar{v}(a) > \eta c(\Pi^*) = \eta c^*(s_{start}, s_{goal}). \tag{5.4}$$

Case 1. *At the end of Algorithm 8,* OPEN *contains a state from* Π^* *that is part of region Ω^{opt}. Let d be such state, using (5.3) it follows that:*

$$g(d) + \bar{v}(d) \geq g(a) + \bar{v}(a),$$

and, by proprieties (1.4):

$$\eta g^*(d) + \bar{v}(d) \geq g(a) + \bar{v}(a).$$

Considering that η is a positive scalar, greater or equal to 1, one ends up with the following equation:

$$\eta\, (g^*(d) + \bar{v}(d)) \geq g(a) + \bar{v}(a).$$

Hence, by definition (5.2), we have that for any $s \in \Omega^{opt}$, $\bar{v}(s) \leq c^*(s, s_{goal})$, and so that:

$$\eta \left(g^*(d) + c^*(d, s_{goal})\right) \geq g(a) + \bar{v}(a). \tag{5.5}$$

From the initial hypothesis (d is part of Π^*) we have that $g^*(d) + c^*(d, s_{goal}) = c(\Pi^*)$. Thus equation (5.5) contradicts (5.4).

Case 2. *At the end of Algorithm 8,* OPEN *contains state c from Π^*, that is not part of region Ω^{opt}. As for Case* 1*, by (5.3) and (1.4), we obtain the following chain of inequalities:*

$$\begin{aligned}
\operatorname*{argmin}_{s \in \text{OPEN}} \bar{f}(s) &= g(a) + \bar{v}(a) \\
&\leq g(c) + \eta h(c) \\
&\leq \eta \left(g^*(c) + h(c)\right) \\
&\overset{(i)}{\leq} \eta \left(g^*(c) + c^*(c, s_{goal})\right) = \eta c(\Pi^*).
\end{aligned}$$

Here, inequality (i) is due to the admissibility properties (1.3) of heuristic function h. Thus, hypothesis (5.4) is again contradicted.

□

Theorem 5. *Any state in the graph is expanded no more than 1 times by FOCS.*

Proof. (Sketch) It is a consequence of the fact that Algorithm 8 works as WA*, ending when a state s, such that $optimal(s)$ is true, is expanded. □

5.3 Application of FOCS to navigation

We use FOCS to find the minimum-time path for a car-like vehicle. In order to do this, we first define the lattice-based graph and value function (5.2) for this particular application.

Lattice-based graph. We use lattice-state [7], [11] to obtain a discretization of the configuration space Ω into the finite set of states S, where every connection between these states represents a feasible path. As shown in section 1.2.1, the two key elements to build a lattice are: the choice of the representation of the states in the lattice, and the action space (or control set) used for the inter-state connections. In this case, each state in the lattice is represented by coordinates (x, y, θ), where the couple (x, y) represents the center of the real wheel axle of the vehicle and θ the orientation angle. The offline construction of the action space is based on the work [11] by Pivtoraiko and Kelly that aims to create near-minimal spanning action spaces. Moreover, to generate feasible actions for each state, we use a trajectory generation algorithm originally developed by Howard and Kelly [41]. In this way an action in the lattice represents a feasible path between two states. We assign the cost of an action to be the time it takes to the vehicle for traversing with constant velocity the path associated with the action.

Value function. Since, for our application we are interested in finding the minimum-time path for a car-like vehicle, we refer to the deterministic method presented in chapter 3. This method allows to define a value function, on domain $\hat{\Omega}^{opt} = \mathbb{R}^2 \times [0, 2\pi)$, which represents the minimum time to reach a target set $\Gamma \in \hat{\Omega}^{opt}$. In particular, the method allows finding the shortest and collision-free path, with a limited number of direction changes. Let $\{V^{*,0}, \dots, V^{*,K_{max}}\}$ be the corresponding solution of Algorithm 3 of Chapter 3, with at most k_{max} direction changes, we define function $\bar{\bar{v}} : S \to \mathbb{R} \cup \{+\infty\}$ as:

$$\bar{\bar{v}}(s) = \begin{cases} +\infty & \text{if } s \notin \hat{\Omega}^{opt}\,, \\ \underset{k}{\operatorname{argmin}} \left\{ -\frac{1}{\lambda} ln(1 - \lambda \; interp(V^{*,k}, s)) \right\} & \text{otherwise}\,, \end{cases}$$

where $interp(V, s)$ is the function that evaluates the value cost function at s as a multi-linear interpolation of the vector V of the values of the cost function on grid vertices. Function $\bar{\bar{v}}$ represents the optimal time to reach target set Γ, for any state s in the lattice, using at most K_{max} direction changes. Since for this application we are interested in finding a minimum time trajectory we will use an high value for parameter K_{max}. Note as for $K_{max} \to +\infty$ in Algorithm 3, $\forall s \in \hat{\Omega}^{opt}$ we have:

$$\bar{\bar{v}}(s) \leq c^*(s, s_{goal}).$$

The optimal trajectory y_s^*, $\forall s \in \hat{\Omega}^{opt}$, can then be found using Algorithm 4 of Chapter 3.

5.3.1 Numerical Tests

We validate FOCS, through the following scenarios:

1. the complex scenario of Figures 5.3,5.4,
2. a parking lot (Figure 5.5),
3. a randomly generated environment (Figure 5.6).

We used a 2.6 GHz Intel Core i5 processor with 8GB RAM.
In Algorithm 8, we compute heuristic h as the Euclidean distance in 2D (x,y), and value function $\bar{\bar{v}}$ by Algorithm 3: here we set $K_{max} = 8$ and target set Γ as an ellipsoid centered in s_{goal} with semi-axis $r_x = r_y = 0.06$ m and $r_\theta = 0.05$ rad. In our tests, the vehicle is size 4.2×2 m with a minimum turning radius of 6 m and constant velocity equal to 1 ms^{-1}.
In the lattice, we employ 16 angles, a 2D (x,y) resolution of 0.2 m for scenario 1, and 0.25 m for scenarios 2,3. Moreover, we use an action space with 8 actions for each state. Figure 5.2 illustrates the action space for a single state in the lattice when the vehicle is facing right.

1) Complex Scenario. Figures 5.3,5.4 show the path computed by FOCS for a parallel parking maneuver and a diagonal parking maneuver. Here, we depict the boundaries of the optimal control region $\hat{\Omega}^{opt}$ and the switching state in red. As per “switching state” we mean the state s^* that ends Algorithm 8, which is the one that joins paths $\Pi : s_{start} \to s^*$ (depicted in green) and $y_{s^*}^*$ (depicted in blue). In the first parking maneuver, value function $\bar{\bar{v}}$ is solved on a grid of 67126 vertexes over

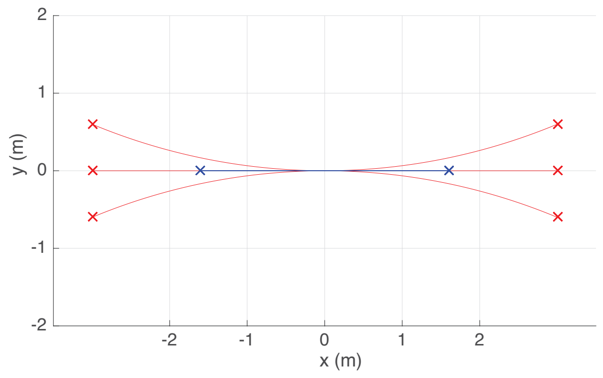


Figure 5.2: Action space for a single state in the lattice. For each action the end point is represented by the cross marker.

$\hat{\Omega}^{opt} = [60,80] \times [30,40] \times [0,2\pi)$, and takes a computational time of 10.3826 s. For the diagonal parking maneuver, the time to compute the value function on a grid of 63851 vertexes, over $\hat{\Omega}^{opt} = [0,22] \times [0,12] \times [0,2\pi)$, is 7.6821 s. In both cases we use discount factor $\lambda = 0.07$. Table 5.1 shows the performance of Algorithm 8 for these two parking maneuvers with different values of η.

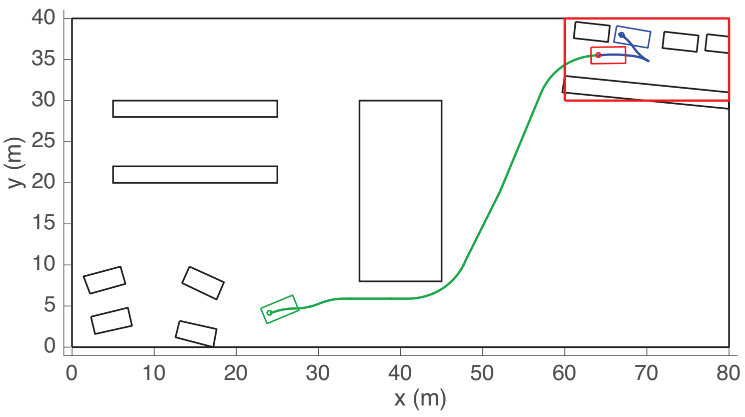
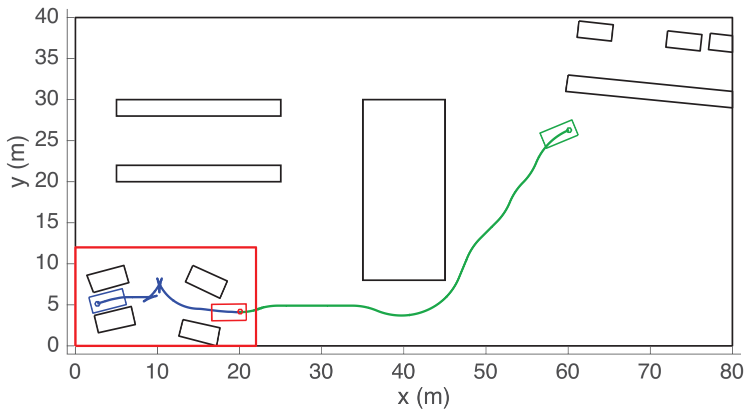


Figure 5.3: Planned path for $s_{start} = (24, 4, 0.571)$ and $s_{goal} = (67.1, 38, 6.173)$ with $\eta = 1$.



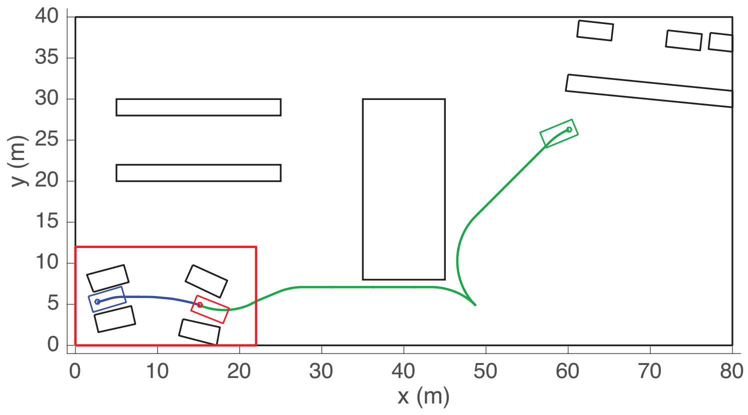


Figure 5.4: Planned path for $s_{start} = (60.1, 26.3, 3.711)$ and $s_{goal} = (2.5, 5.1, 0.25)$ with $\eta = 3$ (top picture) and $\eta = 1$ (bottom picture).

Table 5.1: Performance of FOCS for the parallel and diagonal parking with different values for inflation factor η

η	3	2	1.6	1.4	1.2	1
Parallel: time (s)	0.004	0.021	0.49	1.419	3.416	5.433
Diagonal: time (s)	0.001	0.193	0.862	2.083	4.938	6.938

2) Parking lot. In the 200×200 m parking lot environment of Figure 5.5, the horizontal and vertical lanes are wide 6 and 10 meters, respectively. Here, we test our algorithm computing 100 tests with goal state $s_{goal} = \left(13.9, 46.2, \frac{\pi}{2}\right)$ and randomly generated starting states $(x^r_{start}, y^r_{start}, \theta^r_{start})$, for different value of η. In particular for angle θ^r_{start}, we chose a random value between interval $[-\frac{\pi}{6}, \frac{\pi}{6}] \cup [-\frac{\pi}{6} + \pi, \frac{\pi}{6} + \pi]$ when the random couple $(x^r_{start}, y^r_{start})$ drops in an horizontal lane, and interval $[\frac{\pi}{4}, \frac{3\pi}{4}] \cup [\frac{\pi}{4} + \pi, \frac{3\pi}{4} + \pi]$ otherwise. For all the experiments the solution is found. Table 5.2 shows the mean time over the 100 tests for each value of η. In this case, value function $\bar{\bar{v}}$ is solved on a grid of 63676 vertexes over $\hat{\Omega}^{opt} = [7, 26] \times [44, 57] \times [0, 2\pi)$, with $\lambda = 0.05$, and takes a computational time of 6.947 s.

Table 5.2: Performance of FOCS for the parking lot environment with different values for inflation factor η

η	3	2	1.6	1.4	1.2	1
Mean time (s)	0.003	0.004	0.011	0.029	0.102	1.982

3) Random Environment. We also evaluate Algorithm 8 in the randomly generated environment of Figure 5.6. In this environment we use 100 pairs of random starting and goal states $(s^r_{goal}, s^r_{start})$. The performance of these experiments are shown in Table 5.3. In this case we solve $\bar{\bar{v}}$ over the torus $\hat{\Omega}^{opt} = [x^r_{goal} - 9, x^r_{goal} + 9] \times [y^r_{goal} - 9, y^r_{goal} + 9] \times [0, 2\pi)$, with $\lambda = 0.05$. The resolution of the value function takes a mean time of 8.1247 s for a mean number of 76480 vertexes. For all the experiments the solution is found.

Table 5.3: Performance of FOCS for the random environment with different values for inflation factor η

η	3	2	1.6	1.4	1.2	1
Mean time (s)	0.811	0.884	0.953	1.074	1.306	2.204

Note that, in general, the total time to find a solution by FOCS is given by the sum of the time to solve value function $\bar{v}$ and the time to run Algorithm 8. However, if

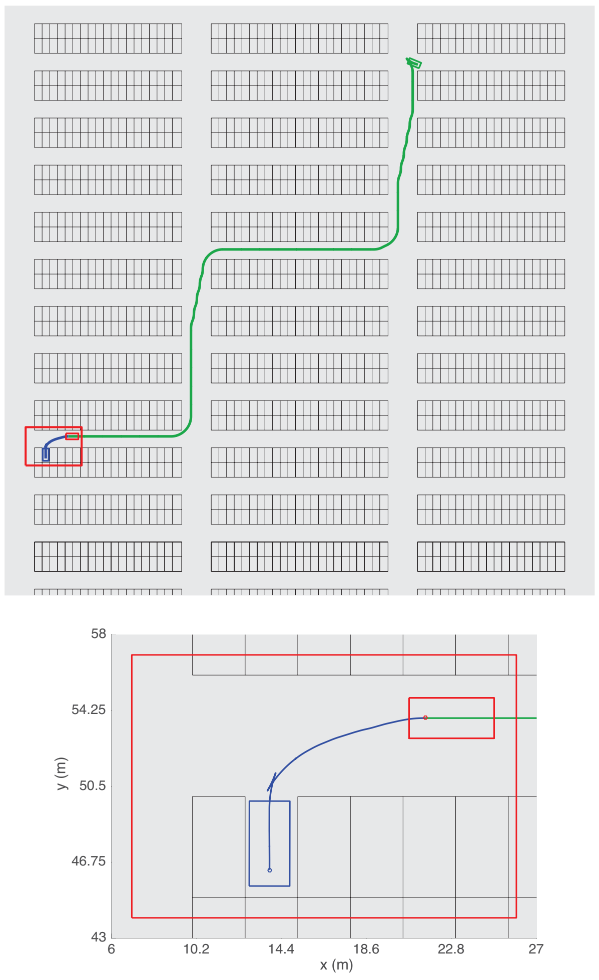


Figure 5.5: Planned path for $s_{start} = (140, 180, 2.6180)$ with $\eta = 1$ in the 200×200 meters parking lot environment.

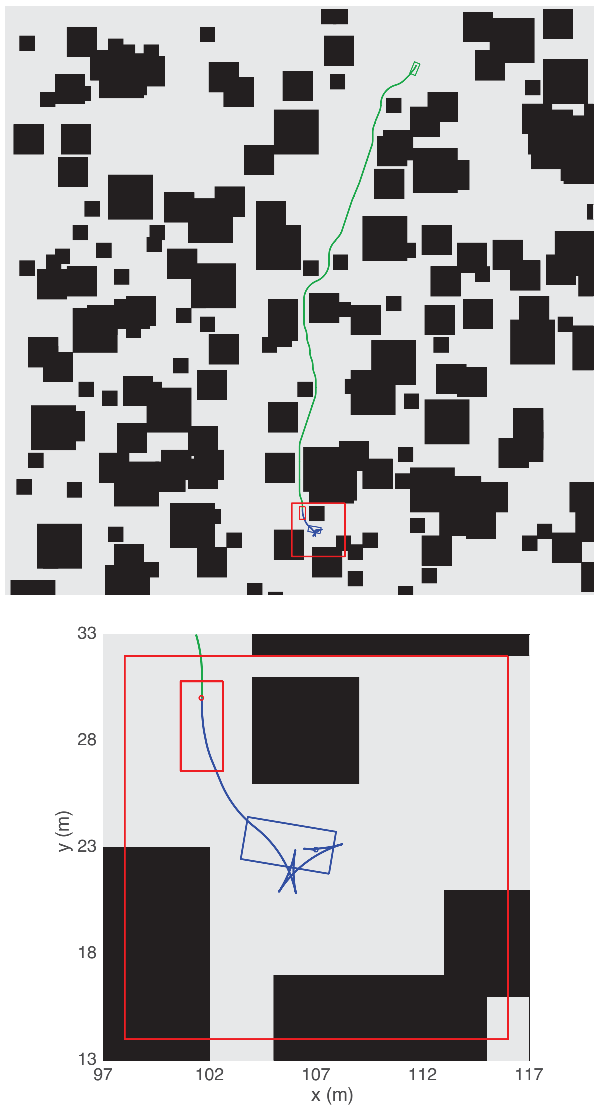


Figure 5.6: Planned path for $s_{start} = (140, 180, 4.4124)$ and $s_{goal} = (107, 23, 2.9)$, with $\eta = 1$, in the 200×200 meters randomly generated environment.

the target state is known and the obstacles in the optimal control region Ω^{opt} are static, one can compute the value function offline and consider only the time to run Algorithm 8.

Conclusions

This thesis presented a general time optimal approach for path planning of road vehicles. The proposed approach, based on the numerical solution of the Hamilton-Jacobi-Bellman equation, has been implemented in two algorithms for performing a parking maneuver for a car-like vehicle. With respect to search-based algorithms, this approach allowed finding an optimal solution at the expense of a greater computational cost. However, both these optimal control-based algorithms are not suitable for finding a solution in large environments. For this reason, this thesis presented an algorithm called FOCS (Fusion of Optimal Control and Search) that combined the advantages of these two approaches while providing a bound on the sub-optimality of its solution. This approach allowed finding a solution for large environments, even when the operating space around the target configuration is tight, exploiting the performances of search-based algorithms and the accuracy of the HJB equation.

Bibliography

[1] Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

[2] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[3] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[4] Ankit Gupta and Rohan Divekar. Autonomous parallel parking methodology for ackerman configured vehicles. In *Proc. of Int. Conf. on Control, Communication and Power Engineering*, 2010.

[5] Guang Song and Nancy M Amato. Randomized motion planning for car-like robots with C-PRM. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 37–42, 2001.

[6] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, 2005.

[7] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[8] Yoshiaki Kuwata, Gaston A Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P How. Motion planning for urban driving using RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1681–1686, 2008.

[9] Long Han, Quoc Huy Do, and Seiichi Mita. Unified path planner for parking an autonomous vehicle based on RRT. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5622–5627, 2011.

[10] Ryo Takei and Richard Tsai. Optimal trajectories of curvature constrained motion in the Hamilton–Jacobi formulation. *Journal of Scientific Computing*, 54(2-3):622–644, 2013.

[11] Mikhail Pivtoraiko and Alonzo Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, pages 3231 – 3237, Pittsburgh, PA, August 2005.

[12] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[13] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[14] Rong Zhou and Eric A Hansen. Multiple sequence alignment using anytime A*. In *Association for the Advancement of Artificial Intelligence (AAAI/IAAI)*, pages 975–977, 2002.

[15] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, 2004.

[16] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[17] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, 2005.

[18] Maxim Likhachev and Anthony Stentz. R* search. *Lab Papers (GRASP)*, page 23, 2008.

[19] Sven Koenig and Maxim Likhachev. D* lite. *AAAI/IAAI*, 15, 2002.

[20] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[21] Maurizio Falcone and Roberto Ferretti. *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*. SIAM, 2013.

[22] M Falcone. The minimum time problem and its applications to front propagation. *Motion by mean curvature and related topics*, pages 70–88, 1994.

[23] Lawrence C Evans. An introduction to mathematical optimal control theory version 0.2. *Lecture notes available at http://math. berkeley. edu/˜ evans/control. course. pdf*, 1983.

[24] S Wang, F Gao, and KL Teo. An upwind finite-difference method for the approximation of viscosity solutions to hamilton-jacobi-bellman equations. *IMA Journal of Mathematical Control and Information*, 17(2):167–178, 2000.

[25] Derong Liu and Qinglai Wei. Finite-approximation-error-based optimal control approach for discrete-time nonlinear systems. *IEEE Transactions on Cybernetics*, 43(2):779–789, 2013.

[26] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf. Discrete-time nonlinear hjb solution using approximate dynamic programming: Convergence proof. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):943–949, Aug 2008.

[27] Mattia Laurini, Piero Micelli, Luca Consolini, and Marco Locatelli. A Jacobi-like acceleration for dynamic programming. In *IEEE 55th Conference on Decision and Control (CDC)*, pages 7371–7376, 2016.

[28] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[29] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[30] H. Minc. *Nonnegative matrices*. Technion-Israel Institute of Technology, Dept. of Mathematics, 1974.

[31] Roger A. Horn and Charles R. Johnson, editors. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 1986.

[32] Piero Micelli, Luca Consolini, and Marco Locatelli. Path planning with limited numbers of maneuvers for automatic guided vehicles: An optimization-based approach. In *IEEE 25th Mediterranean Conference on Control and Automation (MED)*, pages 204–209, 2017.

[33] I Capuzzo Dolcetta and Lawrence C Evans. Optimal switching for ordinary differential equations. *SIAM Journal on Control and Optimization*, 22(1):143–161, 1984.

[34] Huan Zhang and Matthew R James. On computation of optimal switching HJB equation. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 2704–2709, 2006.

[35] M. Falcone. A numerical approach to the infinite horizon problem of deterministic control theory. *Applied Mathematics and Optimization*, 15(1):1–13, 1987.

[36] Xuping Xu and Panos J Antsaklis. Optimal control of switched systems: New results and open problems. In *Proceedings of the American Control Conference*, volume 4, pages 2683–2687. IEEE, 2000.

[37] Michael S Branicky, Vivek S Borkar, and Sanjoy K Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE transactions on Automatic Control*, 43(1):31–45, 1998.

[38] Huan Zhang, MR James, et al. Optimal control for a class of hybrid systems under mixed costs. In *IEEE Conference on Decision and Control*, volume 44, page 60. IEEE, 2005.

[39] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single-and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, 33(2):305–320, 2014.

[40] Jonathan Butzke, Kalin Gochev, Benjamin Holden, Eui-Jung Jung, and Maxim Likhachev. Planning for a ground-air robotic system with collaborative localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 284–291, 2016.

[41] Thomas M Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.