FOCS: Planning by Fusion of Optimal Control & Search and its application to navigation

Piero Micelli
Department of Engineering and Architecture
University of Parma
Parma, PR 43124 Italy
pitermicelli+@gmail.com

Maxim Likhachev
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
maxim+@cs.cmu.edu

Abstract— **Both Optimal Control and Search-based Planning are used extensively for path planning and have their own set of advantages and disadvantages. In this paper, we propose an algorithm FOCS (Fusion of Optimal Control and Search) that combines these two classes of approaches together. 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 a concatenation of the path found in the implicit graph constructed by search and the path generated by following the negative gradient of the value function obtained as a solution of the Hamilton-Jacobi-Bellman equation. We analyze the algorithm and illustrate its effectiveness in finding a minimum-time path for a car-like vehicle in different environments.**

I. Introduction

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 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 ([5], [9] or [4]).

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 [2]. 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 ([16], [11], [1]). In contrast to search-based algorithms, optimal control methods allow for finding an optimal solution to the motion planning problem but at the expense of a greater computational cost. In this paper, we propose to combine these two methods in a single algorithm that we call FOCS (Fusion of Optimal Control and Search). The aim of our 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 1 for a non-holonomic 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 while, the motion planning task inside the goal region is completed via Dynamic Programming-based optimal control. FOCS figures how to concatenate the two portions of the path together to return a single path with guarantees on its quality.

Since our approach is based on both Search-based Planning and Optimal Control Theory, in section II we provide a brief description of their individual operation and properties. The proposed algorithm FOCS and its properties are then described in section III. In section IV we show the results of its application to navigation. In particular, we use FOCS to find the minimum-time path for a car-like vehicle.

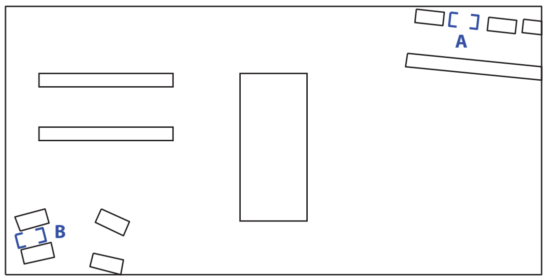


Fig. 1: A navigation scenario with narrow space around goal configurations A and B.

Statement of contribution: In this paper 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.

II. Background

Notations. 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 $s_{goal} \in \Omega^{opt}$. Given a set E, and $i \in \mathbb{N}$, $E^i = E \times E \ldots \times E$ (i times) is the i-th Cartesian power of E and $E^\star \equiv \bigcup_{i=1}^{+\infty} E^i$. Moreover, $L^\infty([0,T];E)$ is the space of essentially bounded function $f:[0,T] \to E$ and $int(E)$ is the interior of the set E.

A. Search-based Planning

Search-based Planning can be decomposed in two parts: turning the motion planning problem into a minimum path problem on a graph, and searching the graph for finding the optimal solution.

In order to properly represent the problem with a graph, the first step is to discretize the robot operating space into a finite set of states (or nodes) S. The choice of 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. The second step is to define valid connections (or edges) between these states. To this end, a common approach is to use a set of motion primitives which represent short and feasible motions of the robot. In particular, the valid successors of a given state are the states that can be reached with these motion primitives while not colliding with obstacles. Since these motions satisfy the kinematic constraints of the robot, this approach constructs a graph where each connection between its nodes represents a feasible path ([9],[14]).
In this graph we define the cost of the edge between nodes s and s' as $c(s,s')$, where $c(s,s') = +\infty$ if there exist no edge 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 an optimal path from s to s'. Moreover, the path $\Pi : s_{start} \to s_{goal}$ denotes a concatenation of edges that connects the starting state s_{start} to the goal state s_{goal}.

The simplest approaches to search a 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 an optimal path, but one that is good enough. One algorithm that does this is WA* (weighted A*). WA* [15] 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 [3], [17], [10]. At each iteration of its main loop, WA* selects the path that minimizes the function $f : S \to \mathbb{R}$:

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

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, the function f represents an estimate of the total cost to travel from s_{start} to s_{goal} going through s.

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

$$h(s) \leq c^*(s, s_{goal}), \forall s \in S, \tag{1}$$

and is consistent if it satisfies:

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

In [13] it is proven that if h is admissible, then the solution of WA* is bounded as follows:

$$g(s) + \eta h(s) \leq \eta(g^*(s) + h(s)) \quad \forall s \in S, \tag{2}$$

where $g^*(s)$ is the cost of an optimal path from s_{start} to s; and does not require re-expansions to guarantee the bound if $h(s)$ is consistent [10].

B. Optimal Control and Dynamic Programming

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

$$\begin{cases} \dot z(t) = \tilde f(z(t), u(t)), \\ z(0) = z_0, \end{cases} \tag{3}$$

where $\tilde f : \mathbb{R}^n \times U \to \mathbb{R}^n$ is a continuous function, $z_0 \in \mathbb{R}^n$ is the initial state, $u(t) \in L^\infty([0, +\infty[; U)$ is the control input and U is a compact set of admissible controls.

Let $y_{z_0,u} = z(t)$ be a solution of ODE (3) for the control $u(t)$, we define the cost functional $J : U \to \mathbb{R}$ as:

$$J_{z_0}(u) = \int_0^\infty l(z(r), u(r))dr\,, \tag{4}$$

where $l : \mathbb{R}^n \times U \to \mathbb{R}$ is a continuous cost function.

With respect to this formulation, our goal is to find an optimal pair (y^*, u^*) (it can be not unique) which minimizes cost functional (4).

Dynamic Programming approach [2] provides a method for the solution of this optimal control problem introducing the value function $\bar v : \mathbb{R}^n \to \mathbb{R}$:

$$\bar v(z_0) = \inf_{u \in U} J_{z_0}(u), \tag{5}$$

that represents the best value of cost functional (4) for the starting state z_0. Since the value function (5) is generally unbounded, a common approach is to perform the following rescaling of $\bar v$ (see [8] and [7]):

$$\breve v(z_0) = \begin{cases} \frac{1}{\lambda} & \text{if } \bar v(z_0) = +\infty, \\ \frac{1}{\lambda} - \frac{1}{\lambda} e^{-\lambda \bar v(z_0)} & \text{otherwise}, \end{cases} \tag{6}$$

where λ is a positive scalar and is called discount factor. The change of variable (6) gives several advantages. In particular, $\breve v$ 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. Once one obtains $\breve{v}$, the value function $\bar{v}$ can be recovered by the relationship:

$$\bar{v}(z_0) = -\frac{1}{\lambda}\ln\left(1 - \lambda\breve{v}(z_0)\right). \tag{7}$$

Function $\breve{v} : \mathbb{R}^n \to \mathbb{R}$ is itself a value function defined as

$$\breve{v}(z_0) = \inf_{u\in U}\int_0^\infty l(z(r), u(r))e^{-\lambda r}dr\,,$$

and is the unique viscosity solution of HJB equation:

$$\lambda\breve{v}(z) + \sup_{u\in U}\{-f(z,u)D\breve{v}(z) - l(z,u)\} = 0,\ z\in\mathbb{R}^n\,, \tag{8}$$

where $D\breve{v}$ denotes the gradient of $\breve{v}$ at z. See [6] for the derivation of equation (8).

The dynamic programming method can then be decomposed as follows. First the HJB equation (8) is solved using a numerical procedures which allows finding a linear approximation to the value function (see [16], [11], [1]). Then, the optimal feedback control $u^*(t)$ is found choosing the control such that the minimum in equation (8) is obtained.

Note that the solution of equation (8) does not depend on the choice of the starting state z_0. Once the value function is computed, it contains all the information needed to compute the optimal solution $y^*_{z_0}$ for any starting state $z_0 \in \mathbb{R}^n$.

III. Algorithm FOCS

As mentioned before, we first use Search-based Planning to find a path on a graph in order to reach the small region Ω^{opt}. Then, we use the Dynamic Programming method to compute a path that goes to s_{goal} within Ω^{opt}. Note that in general this second path does not contain the nodes of the graph.

To this end, we define the path $\Sigma_{\Pi,y}$ as a concatenation of the search-based path $\Pi : s_{start} \to s$ and the optimal path y^*_s, where the state s lies in Ω^{opt}. Let Λ be the finite set that contains all the possible paths $\Sigma_{\Pi,y}$. We introduce the cost function $\bar{l} : \Lambda \to \mathbb{R}$:

$$\bar{l}(\Sigma_{\Pi,y}) = c(\Pi : s_{start} \to s) + \bar{v}(s).$$

Here, $c(\Pi : s_{start} \to s)$ is the cost of the search-based path $\Pi : s_{start} \to s$ and, the function $\bar{v} : \Omega^{opt} \to \mathbb{R}$ is the value function obtained by applying the Dynamic Programming method, and therefore it represents the optimal cost to reach the goal state s_{goal} starting from any state in Ω^{opt}. In this way, the cost of the best path $\Sigma^*_{\Pi,y}$ is defined as follows:

$$\bar{l}(\Sigma^*_{\Pi,y}) = \min_{\Sigma_{\Pi,y}\in\Lambda} \bar{l}(\Sigma_{\Pi,y}). \tag{9}$$

We use our algorithm FOCS (Algorithm 1) to solve problem (9). In particular, for η equal to 1, FOCS returns the best solution $\Sigma_{\Pi,y}(\eta = 1) = \Sigma^*_{\Pi,y}$ (line 3), while for values of η greater than 1 it provides a sub-optimal solution to problem (9).

Algorithm 1 FOCS

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

At each iteration of its main loop, FOCS selects the path that minimizes the 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.

FOCS uses the lists *OPEN* and *CLOSED* to keep track of the frontier states and of the expanded states, respectively. Moreover, it associates with each node the variable $optimal$ to check if a node lies in Ω^{opt}: if $optimal(s)$ is true, then the node s is part of Ω^{opt}. In Loop 15-28, the node s^* represents the last node on the current best path. In for loop 16-25, the successors of s^* are evaluated: in lines 18,19, if s' is a new node, the variable g is initialized to infinity and the variable $optimal$ to false. In line 23, the value of the current best path from s_{start} to 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, as long as the terminal condition in line

15 is not satisfied, at each iteration the node s^* is updated with the node in *OPEN* that minimizes the function $\bar{f}$ (line 26).

As one can see, FOCS works in a way very similar to WA*. There are two main differences between them. One difference is that, when a node lies in Ω^{opt}, FOCS uses the value function instead of the heuristic in order to evaluate the cost from this node to the node that corresponds to the goal configuration. In addition, FOCS ends when a node that is part of Ω^{opt} is expanded while WA* only when the node s_{goal} is reached. Despite these differences, however, we prove that all the theoretical guarantees for WA* hold for FOCS.

A. Theoretical Analysis

When using an admissible and consistent heuristic, the theoretical guarantees of WA* continue to hold for FOCS.

Theorem 1. *Let $a \in \Omega^{opt}$ be a state such that the terminal condition in FOCS Algorithm is satisfied, then we have that:*

$$\min_{s \in \text{OPEN}} \bar{f}(s) = g(a) + \bar{v}(a) \leq \eta c^*(s_{start}, s_{goal}). \qquad (10)$$

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*.

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

$$g(a) + \bar{v}(a) > \eta c(\Pi^*) = \eta c^*(s_{start}, s_{goal}). \qquad (11)$$

Then there must be at least one state on Π^* that has not been expanded and is in *OPEN* at the time the algorithm terminates. Let d be such a state, and if there are multiple such states, then let d be a state that is closest to s_{start} on Π^*. Given this, the parent of d, say p, has been expanded. Consequently $g(p) \leq \eta g^*(p)$ from (2). As a result,

$$g(d) \leq g(p) + c(p,d) \leq \eta g^*(p) + c(p,d) \leq \eta g^*(d). \quad (12)$$

Case 1. $d \in \Omega^{opt}$

$$\begin{aligned}
\textit{from (10),} \quad & g(d) + \bar{v}(d) \geq g(a) + \bar{v}(a) \\
\textit{from (12),} \quad & \eta g^*(d) + \bar{v}(d) \geq g(a) + \bar{v}(a) \\
& \eta \left(g^*(d) + \bar{v}(d)\right) \geq g(a) + \bar{v}(a) \\
& \eta \left(g^*(d) + c^*(d, s_{goal})\right) \geq g(a) + \bar{v}(a) \\
& \eta c(\Pi^*) \geq g(a) + \bar{v}(a).
\end{aligned}$$

Case 2. $d \notin \Omega^{opt}$

$$\begin{aligned}
\textit{from (10),} \quad & g(d) + \eta h(d) \geq g(a) + \bar{v}(a) \\
\textit{from (12),} \quad & \eta g^*(d) + \eta h(d) \geq g(a) + \bar{v}(a) \\
& \eta \left(g^*(d) + c^*(d, s_{goal})\right) \geq g(a) + \bar{v}(a) \\
& \eta c(\Pi^*) \geq g(a) + \bar{v}(a).
\end{aligned}$$

■

Theorem 2. *Any state in the graph is expanded no more than 1 time by FOCS.*

Sketch of Proof. It is a consequence of the fact that FOCS maintains a *CLOSED* list in the same way as WA* without re-expansions does [10]. ■

IV. Application of FOCS to Navigation

We use the FOCS Algorithm to find the minimum-time path for a car-like vehicle. In order to do this, we first define the graph and the value function (5) for this specific application.

Lattice-based graph. We use the lattice-state in [9],[14] to obtain a discretization of the operating space Ω into the finite set of states S, where every connection between these states represents a feasible path. As shown in section II, 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. For this application, 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 work [14] by Pivtoraiko and Kelly that aims to create near-minimal spanning action spaces. In particular, we use an action space that is composed of 8 actions for each state. For instance, Figure 2 illustrates the action space for the state in the lattice that corresponds to the vehicle facing right. Moreover, 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.

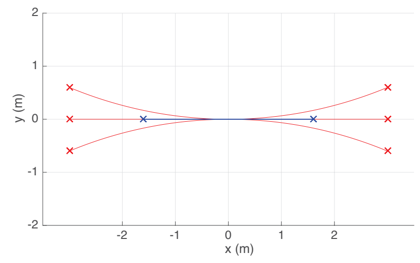


Fig. 2: Action space for a single state in the lattice. For each action the end point is represented by a cross marker.

Value function. In order to model the minimum-time problem for a car-like vehicle via Dynamic Programming, we use the algorithm presented in [12] which allows finding the shortest, collision-free path, with a limited number of direction changes.

Let $\hat{\Omega}^{opt} = \mathbb{R}^2 \times [0, 2\pi)$ be the operating space of the vehicle. Following [12] we model the car-like vehicle by switched system $\dot{z}(t) = \hat{f}(z(t), i, \omega)$, where $\hat{f} : \hat{\Omega}^{opt} \times \{1, 2\} \times \mathbb{R} \to \hat{\Omega}^{opt}$ is defined as:

$$\hat{f}(z, 1, \omega) = \begin{pmatrix} v^+ \cos\theta \\ v^+ \sin\theta \\ \omega \end{pmatrix}, \qquad (13)$$

$$\hat{f}(z, 2, \omega) = \begin{pmatrix} v^- \cos\theta \\ v^- \sin\theta \\ \omega \end{pmatrix}. \qquad (14)$$

Here, $v^- < 0 < v^+$ are the speeds associated to forward and reverse 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. 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 $\sigma = \{(t_1, i_1), (t_2, i_2), \ldots\}$ is the sequence of switches. In this way, if $(t_k, i_k) \in \sigma$, then the controller switches to subsystem i_k at time t_k. Moreover we define the cardinality of σ by $|\sigma|$ and the set of the control signals as $A = L^\infty(\mathbb{R}, [-\omega_{min}, \omega_{max}]) \times (\mathbb{R}, \{1,2\})^\star$.

Let $y_{z,\alpha}(t) = z(t)$ be the trajectory for switched system (13),(14), and $\Gamma \subset \hat{\Omega}^{opt}$ be a target set, closed and such that $int(\Gamma) \neq \varnothing$, we define a cost function $t : \hat{\Omega}^{opt} \times A \to \mathbb{R} \cup \{+\infty\}$ that associates to each initial state $z_0 \in \hat{\Omega}^{opt}$, and control α the cost:

$$t(z_0, \alpha) = \begin{cases} +\infty & \text{if } y_{z_0,\alpha}(t) \notin \Gamma, \forall t > 0\,, \\ \inf\limits_{t \in \mathbb{R}^+} \{t : y_{z_0,\alpha}(t) \in \Gamma\} & \text{otherwise}\,. \end{cases}$$

where we consider $t(z_0, \alpha) = +\infty$ if the solution never reaches set Γ for any $t > 0$. Thus, we define the function $T : \hat{\Omega}^{opt} \times \mathbb{N} \to \mathbb{R} \cup \{+\infty\}$:

$$T(z_0, i) = \inf_{|\sigma| \le i} t(z_0, \alpha)\,, \tag{15}$$

that represents the first time of arrival on the target Γ using at most i direction changes.

We solve problem (15) via dynamic programming by characterizing the function T in terms of a first order Hamilton-Jacobi-Bellman equation. To this end, using the change of variable (6) we obtain the value function:

$$V(z_0, i) = \begin{cases} \frac{1}{\lambda} & \text{if } T(z_0, i) = +\infty, \\ \frac{1}{\lambda} - \frac{1}{\lambda} e^{-\lambda T(z_0, i)} & \text{otherwise.} \end{cases} \tag{16}$$

In this work we use Algorithm 2 and 3 in [12], to solve the value function (16) and to find the optimal trajectory $y^*_{z_0}$, respectively (see [12] for more details).

Let $\{V^{*,0}, \ldots, V^{*,K_{max}}\}$ be the corresponding solution of Algorithm 2 in [12] to problem (16), with at most k_{max} direction changes. Using relationship (7) we define the function $\bar{\bar{v}} : S \to \mathbb{R} \cup \{+\infty\}$ as:

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

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 the vertices of a grid. The function $\bar{\bar{v}}$ represents the minimum time to reach target set Γ for any state s in the lattice, using at most K_{max} direction changes. Since we are interested in finding a minimum time trajectory, in this paper we consider an high value for parameter K_{max}. In fact, using an unlimited number of direction changes (which means $i \to +\infty$ in problem (15)), it can be proven that the following inequality is true:

$$\bar{\bar{v}}(s) \le c^*(s, s_{goal}) \quad \forall s \in \hat{\Omega}^{opt}.$$

A. *Numerical tests*

We validate the FOCS Algorithm through the following scenarios:

1) the complex scenario of Figures 3,4,
2) the parking lot of Figure 5,
3) the randomly generated environment of Figure 6.

We consider a C++ implementation of FOCS Algorithm running on a 2.6 GHz Intel Core i5 processor with 8GB RAM. In FOCS Algorithm, the heuristic function h is computed as the Euclidean distance in 2D (x, y), and the value function $\bar{\bar{v}}$ by a C++ implementation of Algorithm 2 in [12]: here K_{max} is set to 8 and the 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. The vehicle has size 4.2×2 m with a minimum turning radius of 6 m and constant velocity equal to 1 ms^{-1}. Moreover, Algorithm 1 runs on a lattice which employs 16 orientation angles, a 2D (x, y) resolution of 0.2 m for scenario 1, and 0.25 m for scenarios 2,3.

1) Complex scenario. Figures 3,4 show the path computed by FOCS for a parallel parking maneuver and a diagonal parking maneuver. In the figures, the boundaries of the optimal control region $\hat{\Omega}^{opt}$ and the switching state are depicted in red. As per "switching state" we mean the state s^* that ends the FOCS Algorithm, which is the one that joins paths $\Pi : s_{start} \to s^*$ (depicted in green) and $y^*_{s^*}$ (depicted in blue). Moreover, in all the figures the position of the center of the rear wheels axle of the vehicle is represented by a circle. In the first parking maneuver, the value function $\bar{\bar{v}}$ is solved on a grid of 67126 vertexes, over $\hat{\Omega}^{opt} = [60, 80] \times [30, 40] \times [0, 2\pi)$, and takes a computational time of 1.8743 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 1.3226 s. In both cases we use discount factor $\lambda = 0.07$. Table I shows the performance of FOCS for these two parking maneuvers with different values of η.

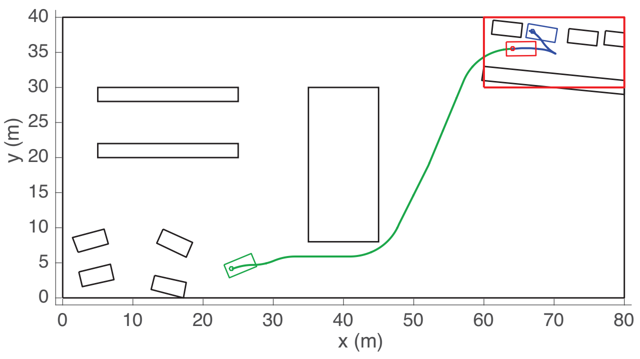
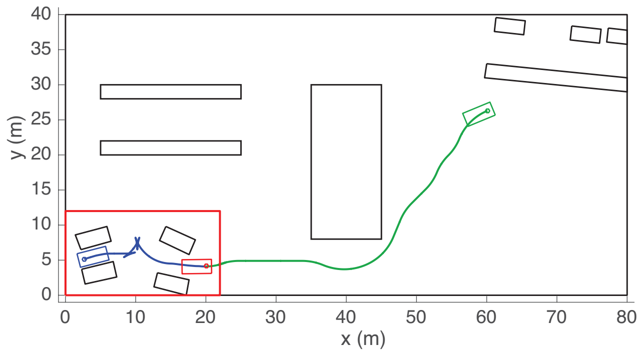


Fig. 3: Planned path for $s_{start} = [24, 4, 0.571]$ and $s_{goal} = [67.1, 38, 6.173]$ with $\eta = 1$.

2) Parking lot. In the parking lot environment of Figure 5, the horizontal and vertical lanes are wide 6 and 10 meters,



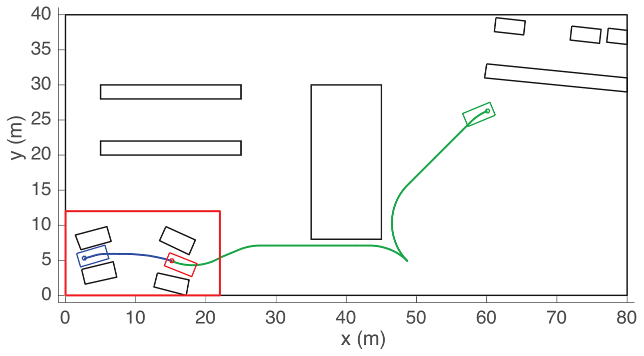


Fig. 4: Planned path for $s_{start} = [60.1, 26.3, 3.711]$ and $s_{goal} = [2.5, 5.1, 0.25]$ with $\eta = 3$ (top picture) and $\eta = 1$ (bottom picture).

TABLE I: Performance of Algorithm 1 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

TABLE II: Performance of Algorithm 1 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

TABLE III: Performance of Algorithm 1 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

respectively. Here, we run the FOCS Algorithm with goal state $s_{goal} = [13.9, 46.2, \frac{\pi}{2}]$ and 100 randomly generated starting states $(x^r_{start}, y^r_{start}, \theta^r_{start})$, using 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 II shows the mean computational time over the 100 tests for each value of η. In this case, the value function $\bar{\bar{v}}$ is solved on a grid of 63676 vertexes, over $\hat{\Omega}^{opt} = [7, 26] \times [44, 57] \times [0, 2\pi)$. Using discount factor $\lambda = 0.05$, it takes a computational time of 1.2156 s.

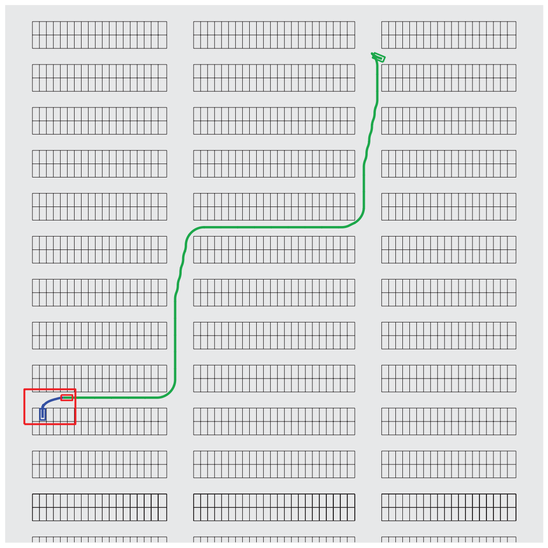
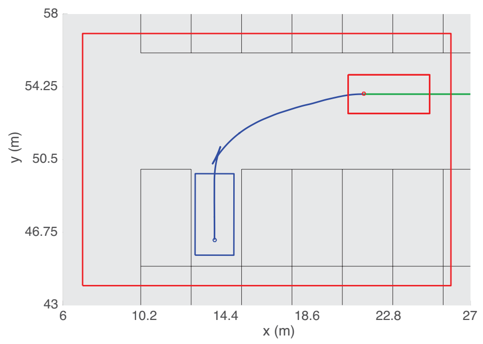



Fig. 5: Planned path for $s_{start} = [140, 180, 2.6180]$ and $s_{goal} = [13.9, 46.2, \frac{\pi}{2}]$, with $\eta = 1$ in a 200×200 meters parking lot environment.

3) Random environment. We also run the FOCS Algorithm in the randomly generated environment of Figure 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 III. 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 1.8426 s for a mean number of 76480 vertexes. For all the experiments the solution is found.

At this point it would be appropriate performing a numerical comparison with the two original algorithms since the algorithm we propose is a fusion of Optimal Control and Search-based Planning. However, if we run the search-based method keeping the same state space resolution and the same action space, the algorithm always fails in the two first scenarios. What one can do is to increase the resolution of the lattice and the number of motion primitives in order to find a solution even when the operating space over the target configuration is tight, but at the expense of a greater computational time. On the other hand, the scenarios $1-3$

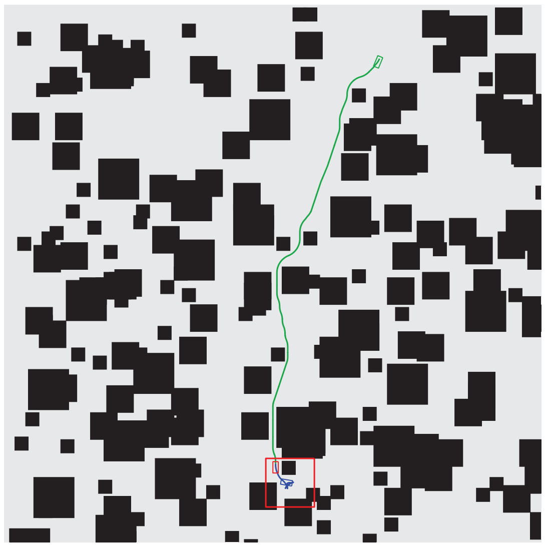
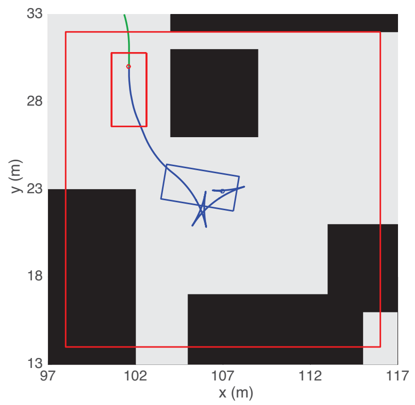



Fig. 6: Planned path for $s_{start} = [140, 180, 4.4124]$ and $s_{goal} = [107, 23, 2.9]$, with $\eta = 1$, in a 200×200 meters randomly generated environment.

cannot be addressed by the Optimal Control method due to the excessive time required for solving a value function that covers the entire scenarios. Therefore, we believe that a fair comparison can be only made with search-based methods which run on a multi-resolution lattice where, in Ω^{opt}, the state space and the action space are chosen dense enough in order to complete the task.

V. CONCLUSIONS

We presented a path planning algorithm based on the fusion of Optimal Control and Search-based Planning while providing bounds on the sub-optimality of its solution. The proposed approach allows us to find a solution for large environment, even when the operating space over the target configuration is tight, exploiting the performances of the search-based algorithms and the accuracy of the Optimal Control Theory. We have tested the algorithm to find the minimum-time path for a car-like vehicle, illustrating the effectiveness of the method with various numerical tests and different environments. Our future work involves an extension of the FOCS algorithm for using the Optimal Control approach also in the start region, since usually in motion planning problems, the planning complexity arises near the start and goal regions. Moreover, it also involves a comparison of our algorithm FOCS with search-based methods running on a multi-resolution lattice.

ACKNOWLEDGMENTS

This work was in part supported by ONR grant N00014-15-1-212.

REFERENCES

[1] 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.

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

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

[4] 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.

[5] 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.

[6] Lawrence C Evans. An introduction to mathematical optimal control theory version 0.2. 1983.

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

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

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

[10] 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.

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

[12] 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.

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

[14] Mihail Pivtoraiko and Alonzo Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3231–3237, 2005.

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

[16] S. Wang, F. Gao, and K. L. 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.

[17] Rong Zhou and Eric A Hansen. Multiple sequence alignment using anytime A*. In *AAAI/IAAI*, pages 975–977, 2002.