Drivable path planning using hybrid search algorithm based on E* and Bernstein-Bézier motion primitives

Gregor Klančar, Marija Seder, *Member, IEEE,* Sašo Blažič, Igor Škrjanc and Ivan Petrović, *Member, IEEE*

***Abstract*—This work proposes a hybrid path-planning algorithm, the HE* algorithm, which combines the discrete grid-based E* search and continuous Bernstein-Bézier (BB) motion primitives. Several researchers have addressed the smooth path planning problem and the sample-based integrated path planning techniques. We believe that the main benefits of the proposed approach are: directly drivable path, no additional post-optimization tasks, reduced search branching, low computational complexity, and completeness guarantee. Several examples and comparisons with the state of the art planners are provided to illustrate and evaluate the main advantages of the HE* algorithm.**

HE* yields a collision-safe and smooth path that is close to spatially optimal (the Euclidean shortest path) with a guaranteed continuity of curvature. Therefore, the path is easily drivable for a wheeled robot without any additional post-optimization and smoothing required. HE* is a two-stage algorithm which uses a direction-guiding heuristics computed by the E* search in the first stage, which improves the quality and reduces the complexity of the hybrid search in the second stage. In each iteration, the search is expanded by a set of BBs, the parameters of which adapt continuously according to the guiding heuristics. Completeness is guaranteed by relying on a complete node mechanism, which also provides an upper bound for the calculated path cost. A remarkable feature of HE* is that it produces good results even at coarse resolutions.

***Index Terms*—Path planning, Bernstein-Bézier curve, motion primitives, hybrid planner, graph search algorithm.**

I. Introduction

Path planning is a fundamental part of any autonomous vehicle with sense-plan-act architecture. The most common scenario assumes path planning in a known environment with obstacles that has been extensively studied in survey books such as [1], [2], [3]. Commonly used methods compute collision-safe paths that are optimal in the sense of the shortest distance or safety but they lack feasibility and therefore require some post-processing to become drivable for a wheeled mobile robots. However, there is no guarantee that a post-processed path is also feasible. The environment is usually decomposed to cells or representative sample points by some algorithm like regular rectangular grid, quad trees, random sampling-based methods and the like [3]–[6]. Among those cells, an optimal collision-free path is found that connects the current robot pose and the goal location. The most commonly used are grid-based planners (e.g. A*) that usually return optimal set of cell's center points which may be connected by a sequence of straight lines to arrive from the start to the goal location. The resulting combined path does not have a continuous first and second derivative, i.e. it belongs to C^0 set of paths (continuity of position) and does not belong to C^1 (continuity of tangency) and C^2 (continuity of curvature). This means that by following the C^0 path the robot would have step changes in its orientation, while by following the C^1 path it would have step changes of its angular velocity. Therefore the calculated path needs to be smoothed to become C^2 so that a wheeled robot can follow it with smooth control actions. This is especially important in occupied and crowded environments where any tracking error could result in a collision in a worst case. Several smoothing approaches were proposed as follows. A funnel algorithm is proposed by [7] to further optimize the path inside the corridor defined by the cells comprising the optimal path. For path optimization and smoothing inside the corridor a Fast marching method [8] can be applied or smooth path generation using B-splines can be used as proposed by [9]. Several path smoothing ideas use a local nonlinear optimization around the path [10]–[13]. Sharp transitions on the path, e.g. corners, are often smoothed to enable continuous transitions by inserting parametric curves such as circular arcs [14], Bezier curves [15]–[18], clothoids [19], [20] or higher order polynomials [21], [22].

Path smoothing is often not integrated in path planning but is usually done after the optimal path is found. This, however, requires additional collision checks and can influence the path optimality [23]. Therefore a better approach in finding a smooth path is to combine motion primitives in a path planning phase. The first studies to obtain the shortest curvature-constrained smooth paths consisting of straight lines and circular arcs was performed by [24]. These paths belong to C^1 as they have a discontinuous curvature. Several local path planners were proposed to find smooth path sections between the initial and the target pose in obstacle-free space to obtain a continuous and bounded curvature path [17]–[21], [25]. However, finding a collision-safe, smooth and optimal path in complex environments with obstacles remains a challenging task.

High-dimensional motion planning for practical applications

Revised manuscript submitted August 1, 2019. This research has been supported partly by the Slovenian Research Agency (research core funding No. P2-0219) and partly by the Croatian Ministry of Science and Education through the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS).

G. Klančar, S. Blažič and I. Škrjanc are with University of Ljubljana, Faculty of Electrical Engineering, Tržaška 25, SI-1000 Ljubljana, Slovenia (e-mail: gregor.klancar@fe.uni-lj.si)

M. Seder and Ivan Petrović are with University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia (e-mail: marija.seder@fer.hr)

has lately been addressed by randomized planners such as the probabilistic roadmap (PRM), the rapidly-exploring random tree (RRT, RRT*) [26], [27], the kinodynamic RRT* [28], [29] or stable-sparse RRT* [30]. The use of PRM and RRT is beneficial in high-dimensional space since these algorithms use random sampling of the space. However, these algorithms are probabilistically complete, which means that with enough samples, the probability of finding an existing solution converges to one.

Deterministic search techniques remain attractive due to their guaranteed completeness (at fine-enough resolution) and optimality for systems which can be modeled accurately by a few dimensions such as wheeled robots [31]. Some approaches reduce the search space by using a discrete set of motion primitives to build a state lattice graph [32]. In the lattice graph motion primitives are polynomials that are regularly arranged to connect nodes (poses) of a discrete grid with smooth transitions of the curvature. Any graph search algorithm can be used to search the lattice graph such as A* [33], D* [34], or the Euclidean shortest E* planner [35], [36]. To cope with the complexity and feasibility challenges iterative search algorithms can be used to fastly obtain the suboptimal solution such as ARA* developed by [37], AD* developed by [38], and ANA* developed by [39]. [10] apply two phase planner where Hybrid A* algorithm produces C^1 path in the first phase that is locally smoothed in the second phase to obtain a feasible path. In Hybrid A* the input commands are discretized while the robot pose remains continuously defined by simulating a robot kinematic model for a small period of time corresponding to the resolution of the grid. To avoid a huge searching graph and time-consuming path planning, a discretized space is used to merge continuous states that occupy the same cell. The recent usage of Hybrid A* algorithm combines a global search at lower resolution with local search at higher resolution to react to local changes in the environment [40].

This work addresses a continuous path planning problem where we suggest the path to be composed of Bernstein-Bézier curves with continuous velocity and curvature transitions. The obtained path can therefore be directly driven by a wheeled mobile robot. Computational complexity of the proposed continuous planner is lowered by guiding the search towards directions obtained on grid-based search heuristics. Resulting paths are smooth, collision-safe, feasible and computed in reasonable time which are all obtained directly in a path planning phase. This allows faster driving velocities on the planned paths.

The main contributions of the paper are as follows. Fifth order Bézier curve sections are formulated which can easily be applied to compose a C^2 path in some path planning applications. The curve sections are not defined as a fixed set of motion primitives. Instead, they can easily adapt in terms of their length and orientation to best fit the used heuristics while overall combined path remains continuous in C^2 and thus drivable for a nonholonomic mobile robot. An application of the proposed Bézier curves to path planning using a hybrid continuous-discrete path planner is proposed. To increase its computational efficiency the heuristics from the optimal grid-based E* path planning is included. A result is the C^2 path that is in vicinity of the shortest one with a guaranteed completeness. Finally some practical directions for obtaining good path planning results at reasonable computational costs for the proposed planner are given.

II. C^2 PATH GENERATION

The resulting path in most path planning approaches is composed of path sections which are continuously joined one after another. Usually the search is done in a discrete space by discretization of all possible robot poses (e.g. grid-based representation of an environment) to a finite set. The resulting path in this case is connected by straight lines usually running through the cell centers. Another commonly used approach is to discretize input commands while the pose remains continuously defined as it is usually done in continuous path planning approaches (e.g. Hybrid A*). The latter can be applied to a differential drive robot which commands are linear velocity $v(t)$ and angular velocity $\omega(t)$. In each node (robot pose x, y, φ) the path planning algorithm expands the search in a predefined number of traveling curves obtained by setting some constant translational and angular velocities, $v(t) = v_{\text{CONST}}$ and $\omega(t) \in \{\omega_{\text{MIN}}, \cdots, 0, \cdots, \omega_{\text{MAX}}\}$, respectively. The path sections therefore have a circular shape and the final robot pose ($x(t_F)$, $y(t_F)$, $\varphi(t_F)$) at time $t_F = t_S + \Delta t$ is obtained by integration

$$\begin{aligned} x(t_F) &= \int_{t_S}^{t_F} v(t)\cos(\varphi(t))dt + x(t_S) \\ y(t_F) &= \int_{t_S}^{t_F} v(t)\sin(\varphi(t))dt + y(t_S) \\ \varphi(t_F) &= \int_{t_S}^{t_F} \omega(t)dt + \varphi(t_S) \end{aligned} \tag{1}$$

which exact solution is

$$\begin{aligned} x(t_F) &= x(t_S) + \tfrac{\Delta s}{\Delta\varphi}\left(\sin\left(\varphi(t_S)+\Delta\varphi\right) - \sin(\varphi(t_S))\right) \\ y(t_F) &= y(t_S) - \tfrac{\Delta s}{\Delta\varphi}\left(\cos\left(\varphi(t_S)+\Delta\varphi\right) - \cos(\varphi(t_S))\right) \\ \varphi(t_F) &= \varphi(t_S) + \Delta\varphi \end{aligned} \tag{2}$$

where t_S is the starting time, t_F is the final time, Δt is the time increment for the path section, $\Delta s = v\Delta t$ is the traveled distance and $\Delta\varphi = \omega\Delta t$ is the change of the robot orientation φ.

Such paths require infinite angular acceleration in the junction of two path sections as illustrated in the following example. An example of a search expansion tree using circular paths (where $x(0) = y(0) = 0$ m, $\varphi(0) = \pi/4$ rad, $v = 0.5$ m/s, $\omega \in \{-1, -0.5, 0, 0.5, 1\}$ rad/s and $\Delta t = 1$ s) is shown in Fig. 1 by dotted lines. To follow the path marked by the thick dotted line in Fig. 1 the robot controls would need to be discontinuous as shown in Fig. 2 which obviously is not C^2. Note that C^2 paths have an identical angular velocity and curvature at the junction, where the curvature is defined as $\kappa = \frac{\omega}{v}$.

To have a feasible (C^2) planned path for the robot Bernstein-Bézier (BB) fifth order curves are proposed as follows. Each BB curve finishes with final position and orientation that are calculated by (2) for the fixed Δs and $\Delta\varphi$. The fifth order BB curve $\mathbf{r}(\lambda) = [x(\lambda), y(\lambda)]^T$ which is defined by six control

points $\mathbf{P}_i = [x_{c_i}, y_{c_i}]^T$, $i \in \{0, 1, \cdots, 5\}$ is chosen as follows

$$\mathbf{r}(\lambda) = (1-\lambda)^5 \mathbf{P}_0 + 5\lambda(1-\lambda)^4 \mathbf{P}_1 + 10\lambda^2(1-\lambda)^3 \mathbf{P}_2 + 10\lambda^3(1-\lambda)^2 \mathbf{P}_3 + 5\lambda^4(1-\lambda)\mathbf{P}_4 + \lambda^5 \mathbf{P}_5, \tag{3}$$

where λ is normalized time ($0 \leq \lambda \leq 1$) within one BB curve. A combined path consists of more BB sections. The meaning of the control points of each BB is as follows. The first three control points (P_0, P_1, P_2) are needed to obtain the C^2 spline and the last two (P_4, P_5) to have a desired final position and orientation. Additionally, P_3 sets the final angular velocity and acceleration to zero which is needed to obtain the straight path without unnecessary waving in cases where the initial and final orientation are nearly the same. If the terminal angular velocity of a BB curve is nonzero, then C^2 conditions would require the same velocities of the next BB curve which could not result in a straight motion when the initial and final orientation are the same.

The C^2 spline is obtained by setting the following three conditions

$$\begin{aligned} &\lim_{\lambda\to 1} \mathbf{r}_j(\lambda) = \lim_{\lambda\to 0} \mathbf{r}_{j+1}(\lambda) \\ &\lim_{\lambda\to 1} \frac{d\mathbf{r}_j(\lambda)}{d\lambda} = \lim_{\lambda\to 0} \frac{d\mathbf{r}_{j+1}(\lambda)}{d\lambda} \\ &\lim_{\lambda\to 1} \frac{d^2\mathbf{r}_j(\lambda)}{d\lambda^2} = \lim_{\lambda\to 0} \frac{d^2\mathbf{r}_{j+1}(\lambda)}{d\lambda^2}, \end{aligned} \tag{4}$$

saying that the end of the curve j and the start of the curve $j+1$ as well as their first and second derivative are the same. This defines the first three control points of the $(j+1)$-th BB curve $\mathbf{P}_{i,j+1}$ ($i \in \{0, 1, 2\}$) related to the control points of j-th curve ($\mathbf{P}_{i,j}$)

$$\begin{aligned} \mathbf{P}_{0,j+1} &= \mathbf{P}_{5,j} \\ \mathbf{P}_{1,j+1} &= 2\mathbf{P}_{5,j} - \mathbf{P}_{4,j} \\ \mathbf{P}_{2,j+1} &= 4\mathbf{P}_{5,j} - 4\mathbf{P}_{4,j} + \mathbf{P}_{3,j}. \end{aligned} \tag{5}$$

The last two control points $\mathbf{P}_{5,j+1}$ and $\mathbf{P}_{4,j+1}$ are defined by the final position $[x_{j+1}(1), y_{j+1}(1)]$ and the final orientation $\varphi_{j+1}(1)$ calculated using (2)

$$\begin{aligned} \mathbf{P}_{4,j+1} &= \begin{bmatrix} x_{j+1}(1) \\ y_{j+1}(1) \end{bmatrix} + \tfrac{1}{5} v \begin{bmatrix} \cos(\varphi_{j+1}(1) + \pi) \\ \sin(\varphi_{j+1}(1) + \pi) \end{bmatrix} \\ \mathbf{P}_{5,j+1} &= \begin{bmatrix} x_{j+1}(1) \\ y_{j+1}(1) \end{bmatrix}, \end{aligned} \tag{6}$$

where in (2) substitute $\Delta s \leftarrow \Delta s_{j+1}$, $\Delta\varphi \leftarrow \Delta\varphi_{j+1}$, $x(t_S) \leftarrow x_{j+1}(0)$, $y(t_S) \leftarrow y_{j+1}(0)$, $\varphi(t_S) \leftarrow \varphi_{j+1}(0)$, $x(t_F) \leftarrow x_{j+1}(1)$, $y(t_F) \leftarrow y_{j+1}(1)$ and $\varphi(t_F) \leftarrow \varphi_{j+1}(1)$, respectively and v is the traveling velocity in the $(j+1)$-th BB end point.

Zero angular velocity and zero tangential acceleration requirement at the curve end are

$$\lim_{\lambda\to 1} \frac{d\varphi_{j+1}(\lambda)}{d\lambda} = 0\,, \quad \lim_{\lambda\to 1} a_{t,j+1} = 0 \tag{7}$$

which defines

$$\mathbf{P}_{3,j+1} = \begin{bmatrix} x_{j+1}(1) \\ y_{j+1}(1) \end{bmatrix} + \frac{2}{5} v \begin{bmatrix} \cos(\varphi_{j+1}(1)) + \pi) \\ \sin(\varphi_{j+1}(1)) + \pi) \end{bmatrix}. \tag{8}$$

Path expansion during the path-planning using BB curves is shown in Fig. 1. Control points of the first BB curve (the lowest curve in Fig. 1) are $\mathbf{P}_{0,j=1} = [x(0), y(0)]^T$, $\mathbf{P}_{1,j=1} = \mathbf{P}_{0,j=1} + 0.2v[\cos\varphi(0), \sin\varphi(0)]^T$ and $\mathbf{P}_{2,j=1} = 0.5\mathbf{P}_{1,j=1} + 0.5\mathbf{P}_{4,j=1}$ while $\mathbf{P}_{3,j=1}$, $\mathbf{P}_{4,j=1}$ and $\mathbf{P}_{5,j=1}$ are defined considering relations (6) and (8). The obtained graph tree of paths has a similar spread to the graph tree obtained by circular paths but has a continuous second derivative of the path which is not the case with circular paths as seen in Fig. 2. The ability to obtain straight path sections can be easily

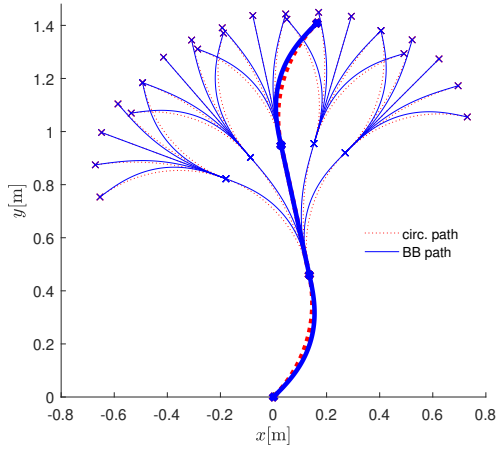

Fig. 1. Search expansion obtained by using circular paths and BB paths.

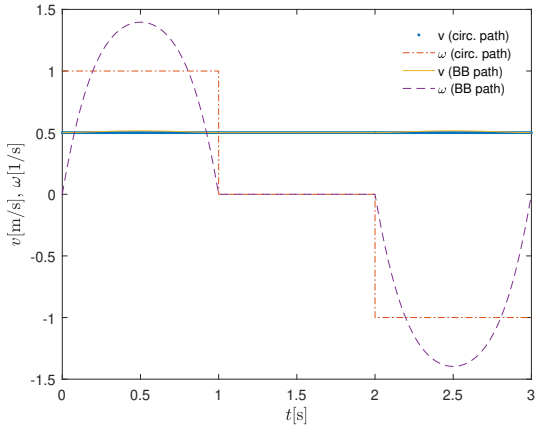

Fig. 2. Differential drive control signal to follow the thick path shown in Fig. 1 obtained by circular paths and by BB paths (x axis denotes cumulative normalized time).

seen in the second path section (the middle part of the thick curve in Fig. 1) which has the same direction in the start and in the end of the section.

The obtained combined path is therefore feasible for the robot to follow. It is smooth with continuous control velocities, a continuous path curvature and is therefore appropriate for drivable path-planning methods. The other curve-section planners, which are cited in Sec. I could also be applied to the proposed HE* planner. They would provide similar results but would be less computationally efficient and elegant. The use of BB curve sections is simple as it is completely defined by two parameters only, the distance and the orientation increment, while the initial requirements are defined by the predecessor BB section. They are suitable for describing simple shapes similar to circular arcs. However, they cannot be used for complex curves with several turns (unless more BB sections

are combined). Furthermore, they cannot be implemented as a single primitive path planner as they need to be initialized by the predecessor curve section to form a C^2 combined path.

III. Drivable path planning using discrete-continuous search

This section details the proposed hybrid path planner idea. The basic algorithm is illustrated by first using a fixed set of motion primitives for expanding the path search. Then the heuristics is introduced based on the discrete planner (E*) to obtain a more efficient search expansion and more suitable resulting paths.

A. The concept of the hybrid search algorithm

The implementation of the Hybrid E* algorithm (HE*) with the proposed C^2 path sections is illustrated in Alg. 1. It is a two stage algorithm, where in the first stage the E* algorithm on a 2D gridmap is performed for computing the heuristics for the second stage. Usually the heuristics is used in the graph search algorithms to narrow the search around the optimal path. The most basic heuristics is the Euclidean distance between the current node and the goal node. Some algorithms use better heuristics that are precomputed in advance by calling an additional lower-dimensional (2D) search and, thus, taking the obstacles into account [10], [32], or to combine different heuristics to capture the high-dimensional complexity into account [41]. Here we use E* to approximate the shortest Euclidean distance considering obstacles, and additionally to guide the creation of motion primitives. The second stage is the hybrid search with motion primitives, which are used as the edges of the graph, whose endpoints serve as the nodes of the graph. The search graph is created while searching the nodes. Each node includes parameters of a continuous BB path whose final pose defines the node pose. The search starts from the start node, initialized by the control points $\mathbf{P}_i$ (see Alg. 1, line 2). The calculation of successor nodes and their BB curves are obtained using (5)–(8). Furthermore, each node includes the total cost, which is the sum of the currently optimal cost-to-here and the estimation of cost-to-goal (heuristics). HE* keeps track of the currently expanded nodes by putting them on the so-called $OPEN$ list, where the node with the smallest total cost is expanded by its neighbors and then removed from $OPEN$ and put on the $CLOSED$ list. The node's neighborhood is defined by a set of curve sections which make constant distance Δs and orientation increments $\Delta\varphi$ according to equivalent circular arcs, e.g. (2). A fixed set of orientation increments $\{\Delta\varphi_k\}$ is usually used in hybrid search, but in HE* variable orientation increments are calculated according to guidance from E*. The search graph grows exponentially with the number of iterations where some new opened nodes can have a pose very similar to the existing ones. Even for a small number of defined continuous motion curves (e.g. 5 motion curves as in Fig. 1) and for a finite (bounded) environment the obtained graph can be infinite (e.g. circular motion where the quotient of its circumference and the curve length is irrational). Therefore nodes with a similar pose need to be eliminated to obtain a manageable graph size as done in Alg. 1, lines 15–23. A node in $OPEN$ or $CLOSED$ list is called a *twin* if its position and orientation differs (to a new node candidate for a search extension) for less than the defined thresholds. A new node which has a twin is examined in the search process only if its cost value is lower than the cost of a twin, otherwise it is ignored. The selection of the thresholds influences the search graph size, computational complexity and planning behavior. An example of appropriate threshold values for BB primitives with $\Delta s = 0.5$ m is 0.15 m and 5° for distance and orientation, respectively. Smaller thresholds result in a larger graph, an increased search complexity and a more detailed search. A higher threshold for orientation can also cause some unwanted path oscillations where orientation changes are bounded with the orientation threshold. This concept is similar to [10], where instead of ignoring a twin node similar nodes are merged into a discretized 3D pose.

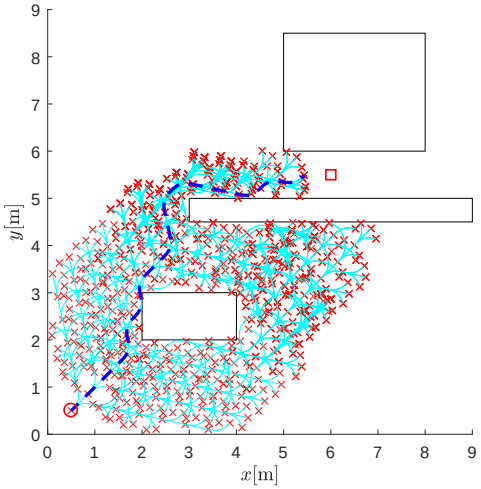

Fig. 3. A hybrid search using fixed set of three C^2 motion primitives.

In Fig. 3 an example of the hybrid search using the fixed set of motion primitives is shown. The obtained path is drivable for a wheeled robot because its path is C^2 which also follows from (4). However, there are several unnecessary turns that appear due to the use of the fixed set of BB sections (motion primitives) during the path search and the goal cannot be reached exactly. In the presented case each node has three successor nodes which are accessed through one straight path ($\Delta s = 0.5$ m, $\Delta\varphi = 0$) and two curve paths ($\Delta s = 0.5$ m, $\Delta\varphi \in \{-1, +1\}$ rad). In the following, we will explain the direction guiding heuristics from E* and a variable motion primitives determination which both further improve the hybrid search.

B. The first stage – a brief overview of E* search

E* algorithm produces the approximation of the shortest Euclidean distance costs from each node in the environment to the goal [35]. An example of the paths obtained by the E* algorithm for a discrete set of nodes by following the steepest negative cost gradient with a small step size corresponding to the grid resolution is given in Fig. 4. E* has the same complexity as the A* algorithm, but it starts the search from the goal node like the D* algorithm (the dynamic version

Algorithm 1 Hybrid E* search: C^2 path search based on E* algorithm

Require: start position $\mathbf{SP} = [x_{SP}, y_{SP}]^T$ and orientation φ_{SP}, end position $\mathbf{EP} = [x_{EP}, y_{EP}]^T$, traveling velocity v, distance increment Δs

1: E* search: planning and replanning from $\mathbf{SP}$ to $\mathbf{EP}$ computes the cost-to-goal from each grid-node in the 2D gridmap

2: Initialize the start node $\mathbf{SP}$ with its control points $\mathbf{P}_i$ for calculation of successor BB curves, cost functions CTH (cost-to-here), CTG (cost-to-goal), $COST = CTH + CTG$ (total cost) and search lists $OPEN$ and $CLOSED$:

$$\begin{aligned}
&\mathbf{P}_0 \leftarrow [x_{SP}, y_{SP}] - \Delta s[\cos\varphi_{SP}, \sin\varphi_{SP}]^T \\
&\mathbf{P}_1 \leftarrow \mathbf{P}_0 + 0.2v[\cos\varphi_{SP}, \sin\varphi_{SP}]^T \\
&\mathbf{P}_5 \leftarrow [x_{SP}, y_{SP}]^T \\
&\mathbf{P}_4 \leftarrow \mathbf{P}_5 + 0.2v[\cos(\varphi_{SP}+\pi), \sin(\varphi_{SP}+\pi)]^T \\
&\mathbf{P}_2 \leftarrow 0.5(\mathbf{P}_0 + \mathbf{P}_5) \\
&\mathbf{P}_3 \leftarrow \mathbf{2}P_4 - \mathbf{P}_5 \\
&CTH \leftarrow 0 \\
&CTG \leftarrow \text{heuristics}(\mathbf{SP}, \mathbf{EP}) \\
&COST \leftarrow CTH + CTG \\
&parent_node \leftarrow [\,] \\
&\text{Create } new_complete_node \text{ with } \mathbf{SP} \text{ and BB parameters} \\
&OPEN \leftarrow [new_complete_node],\ CLOSED \leftarrow [\,]
\end{aligned}$$

3: **while** goal $\mathbf{EP}$ is not reached or $OPEN$ list is not empty **do**

4: Take the node from $OPEN$ list which has the smallest $COST$, set it as $current\_node$ and move it to $CLOSED$ list

5: $\{\Delta\varphi_k\} \leftarrow$ compute_rotational_increments for $current\_node$ based on E* path gradients and the node orientation $\varphi(t)$

6: Extend the search from $current\_node$ by computing new_nodes positions using relation (2) considering distance increment Δs and rotational increments $\Delta\varphi_k$

7: **if** $current\_node$ is complete node **then**

8: Extend the search also by $new\_complete\_node$ using Alg. 2

9: Compute $COST$ of $new\_complete\_node$ and add it to $OPEN$:

$$\begin{aligned}
&\Delta s_C = \text{length}(new_complete_node) \\
&CTH \leftarrow current_node.CTH + \Delta s_C \\
&CTG \leftarrow \text{heuristics}(new_complete_node, \mathbf{EP}) \\
&COST \leftarrow CTH + CTG \\
&parent_node \leftarrow current_node \\
&OPEN \leftarrow OPEN \cup new_complete_node
\end{aligned}$$

10: **end if**

11: **for** all new_nodes **do**

12: Compute $COST$ of new_node:

$$\begin{aligned}
&CTH \leftarrow current_node.CTH + \Delta s \\
&CTG \leftarrow \text{heuristics}(new_node, \mathbf{EP}) \\
&COST \leftarrow CTH + CTG
\end{aligned}$$

13: **if** $current\_node$ is complete node **then**

14: Mark new_node as a valid candidate

15: **else if** new_node has twin in $CLOSED$ or $OPEN$ **then**

16: **if** $COST(new\_node) < COST(twin)$ **then**

17: Mark new_node as a valid candidate

18: **else**

19: Mark new_node as an invalid candidate

20: **end if**

21: **else**

22: Mark new_node as a valid candidate

23: **end if**

24: **if** new_node is valid **then**

25: Compute BB path for new_node using (5)–(8)

26: **if** new_node and its path is collision safe **then**

27: Add new_node to $OPEN$:

$$\begin{aligned}
&parent_node \leftarrow current_node \\
&OPEN \leftarrow OPEN \cup new_node
\end{aligned}$$

28: **end if**

29: **else**

30: Ignore new_node

31: **end if**

32: **end for**

33: **end while**

34: **if** goal $\mathbf{EP}$ is reached **then**

35: Return C^2 path by backtracking the parent nodes from the goal

36: **else**

37: C^2 path to $\mathbf{EP}$ does not exist

38: **end if**

of A*), and inherits the fast replanning capability of the D* algorithm, which is applicable in dynamic real-world environments. The basic idea is to use those path gradients for a more efficient search expansion of the HE* algorithm, where from the current node the search expands mostly in the directions of the gradient resulting from the vector field of the E* algorithm. Additionally, the cost at each node produced by the E* algorithm is used for the cost-to-goal in the HE* algorithm as admissible heuristics (lines 2, 9 and 12 in Alg. 1).

The E* algorithm uses the interpolation based on the Level Set Method [42] to produce the approximation of the shortest (Euclidean) distance from every cell in the search space to the goal. A numerical value is assigned to each cell by the so-called wavefront propagation over the free cells in the grid map. The wavefront propagation acts like a continuous contour that sweeps from the goal node outwards and at each cell record the crossing time. Therefore, the crossing time at cells can be considered as samples of continuous cost function. The path cost at each cell can be calculated by dividing the crossing time by the speed of the propagation.

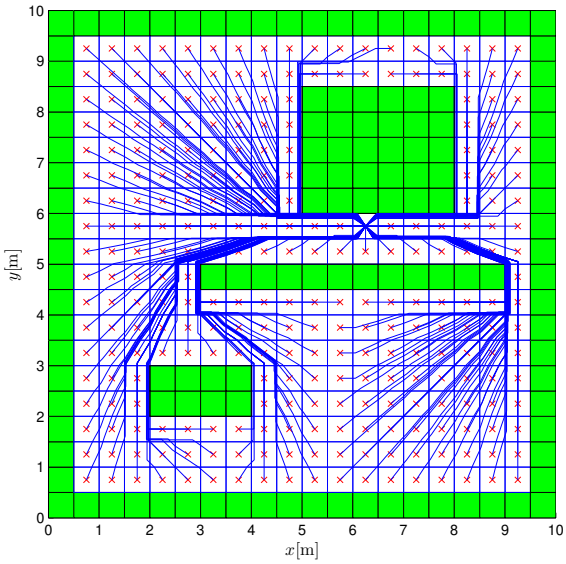

Fig. 4. The E* algorithm. Obtained path gradients from each node (x-mark) can be used to further optimize the path using the HE* algorithm. Note that path gradients are obtained from an arbitrary location inside the node.

Unlike the D* or A* algorithm, where each searched node has one parent node that determines its cost, the E* algorithm uses up to two parent nodes that are used for the calculation of the interpolation cost. Each node has four neighbors in the orthogonal (x, y) directions, and parent nodes are always orthogonal to each other (close to obstacles only one parent may exist). If we denote the cost of the node $\mathbf{N}$ as T and the costs of the parent nodes $\mathbf{A}$ and $\mathbf{B}$ as T_A and T_B, respectively, where $\mathbf{A} \equiv \mathbf{B}$ if only one parent exists, then the gradient of the path from any point within the node $\mathbf{N}$ is parallel to the vector $\overrightarrow{\mathbf{p}}$ determined as

$$\overrightarrow{\mathbf{p}} = (T - T_A) \cdot \overrightarrow{\mathbf{NA}} + (T - T_B) \cdot \overrightarrow{\mathbf{NB}}, \tag{9}$$

where $\overrightarrow{\mathbf{NA}}$, $\overrightarrow{\mathbf{NB}}$ are vectors connecting the corresponding node positions.

C. The second stage – direction guiding heuristics

Proper guiding heuristics, if available, can greatly improve a path planning performance. The simplest solution (without guiding heuristics) is to have path sections fixed by setting desired orientation increments $\Delta\varphi$ and traveled distance increment Δs (approximated by a circular arc). This may result in a suboptimal path (as seen in Fig. 3) if the needed optimal driving direction is different from those available in the fixed set. A good and simple strategy is also to append the new driving direction directly to the goal if the goal is reachable with a straight line motion. In case of an unreachable goal one

can add driving directions towards the corners of the obstacles which are in-between the current pose and the goal (as in Any-Angle path planning [43]). This, however, is not so trivial to program and could result in many new driving directions (they grow quadratically with the number of corners) which would make the search more extensive and computationally expensive.

A good choice to obtain a smooth, close-to-optimal and computationally effective path search is to consider driving directions obtained from the pre-executed straight-line-path search as informative heuristics [32]. Pre-planning can also be done by discrete grid search such as A* or Dijkstra's algorithm if the search starts from the goal and expands in the neighborhood using cost-to-hear only. The result is a set of optimal straight-line-paths from any node towards the goal node. For any node an optimal driving direction (heuristic for HE*) is then defined by a vector towards its parent node. However, due to 8-neighbor or 4-neighbor connected cells (nodes) in the graph search A* is not the most appropriate selection for HE* heuristics as its directions are limited to multiples of $45°$ or $90°$. The path gradient of E* algorithm can serve as much better heuristics since the path of E* algorithm approximates the shortest Euclidean path, as can be seen in Fig. 4. The direction of the gradient vector at the current location (in HE* search) can be determined by first locating which cell N (in E*) the current location belongs to and then applying (9). For a more extensive search, which also affects the resulting path smoothness, also directions of the gradient vector at further parent nodes of the cell N can be used. In general, it is good to choose the direction of the further parent node at the distance Δs from the current location to prevent the sharp changes of the path direction.

The desired orientation increments $\Delta\varphi_k$ ($k \in \{-1, 0, 1\}$) needed to expand the search towards the gradient of the path $\overrightarrow{\mathbf{p}}$ as follows

$$\Delta\varphi_k = \angle\overrightarrow{\mathbf{p}} + k \cdot \eta - \varphi(t) \tag{10}$$

where η is a small orientation change (e.g. $15°$ left and right from the desired direction) to allow deviations from the desired path gradient (or from the average of the desired gradient directions if also directions from the further parent nodes are used), and $\mathbf{P}(t)$ and $\varphi(t)$ are the current position and the orientation of the searched node pose, respectively (see Fig. 5 for details). Computed increments $\Delta\varphi_k$ are used in Alg. 1 (line 5). If the goal node is visible from the current node of the search, then the desired orientation increment in (10) is replaced by the direction towards the goal.

The resulting path obtained using the proposed C^2 path expansion based on the expansion directions obtained from E* algorithm is shown in Fig. 6. Due to the considered gradients from Fig. 4 the smooth C^2 path can be found with less iterations while the solution is also closer to the optimal one than in Fig. 3.

Although in each node of HE* the motion primitives are generated mostly along the gradient (see Fig. 5 and Eq. (10)) given by heuristics this does not imply that other directions are not explored. The search is directed in the direction of the gradient as well as in the directions $\pm\eta$ away from the gradient

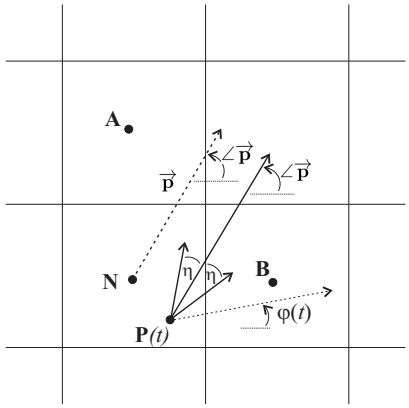

Fig. 5. Determination of orientation increments $\Delta\varphi_k$ for HE* heuristics. Desired directions $\varphi(t) + \Delta\varphi_k$ in the current node $\mathbf{P}(t)$ of HE* search are marked by solid vectors.

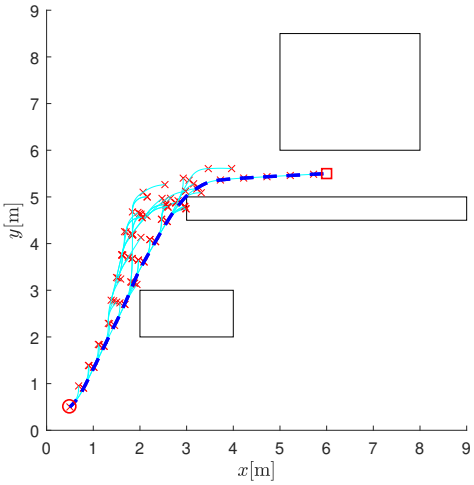

Fig. 6. C^2 path expansion based on the directions from E* algorithm.

(or even more directions e.g. $\pm\eta$ and $\pm 2\eta$). This means that also this additional directions are explored during the search. By introducing the complete motion primitives (see Sec. III-E) that pass through the cells along the gradient it is ensured that a narrow passage is never missed if the path through it is the cheapest.

To apply HE* in a partially known environment the algorithm would require modifications such as an iterative search algorithm (ARA* [37], ANA* [39]) or a reversed search from the goal to the start node as is the case with dynamic search algorithms (D*, AD* [38]). Since E* inherits dynamic properties of D*, the gradients used by HE* will change around the newly detected obstacle. This requires a new search expansion by BBs from the cells with changed gradients backwards to the current robot position, while the search tree behind the new obstacle to the goal is still valid and does not need recalculation. Note that the presented path planning can still be efficiently applied to partially known environments where the first stage with E* already replanes efficiently while the second stage of HE* needs to recompute the path from the start to the goal when the environment changes. Since the second stage has a relatively low number of nodes this is not computationally critical.

D. Final orientation

The path planning algorithm presented above arrives in a goal position with an orientation which is implicitly defined by the resulting path. When the search is sufficiently close (e.g. less than Δs) to the goal and the goal is reachable, the final node and its BB is computed to arrive in goal position exactly. If some desired goal orientation is required, it can easily be achieved by stopping the robot and rotating on the spot. When a smooth robot motion is preferred, the following approach can be used. The described planning heuristics needs to be updated to achieve the desired goal orientation. Here an idea of an intermediate direction similar to the one proposed by [44] is used. When the current search is sufficiently close to the goal ($D < D_{shift}$, D_{shift} is a design parameter), the orientation increments $\Delta\varphi_k$ in (10) need to be computed from the shifted gradient orientation for an angle γ (modify $\angle\overrightarrow{\mathbf{p}} \leftarrow \angle\overrightarrow{\mathbf{p}} + \gamma$)

$$\gamma = \varphi_r(t) - \varphi(t) + \begin{cases} \alpha, & \text{if } |\alpha| < |\beta| \\ \beta, & \text{otherwise} \end{cases} \tag{11}$$

where $\varphi(t)$ is the current orientation of the search, $\varphi_r(t)$ is the orientation of the vector from the current search position towards the goal, $\alpha = \varphi_r(t) - \varphi_{EP}$,

$$\beta = \begin{cases} \arctan\frac{r}{D}, & \text{if } \alpha > 0 \\ -\arctan\frac{r}{D}, & \text{otherwise} \end{cases} \tag{12}$$

φ_{EP} is the goal orientation, D is the distance from the current search position to the goal and $r > 0$ is a design parameter (ensure (x_r, y_r) is in free space) as illustrated in Fig. 7. The

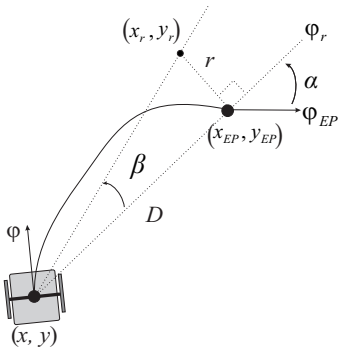

Fig. 7. Achieving a final orientation using an intermediate direction shift.

basic idea is that $\varphi_r(t)$ is shifted by the intermediate direction β in the direction away from the goal orientation φ_{EP} when the robot is far from the goal and for α when being close to the goal. This switch between β and α is smooth. This implies that the path heading is forced toward the goal point and the path arrives there with the desired goal orientation.

E. Completeness and optimality

With discrete systematic search planners which keep a list of visited nodes and operate on a finite graph, completeness and optimality are easily achieved [3]. However, a continuous search needs to prevent infinite search of state space by the use of action commands which are discretized and the obtained continuous states need to be merged to a finite set. Therefore, the completeness guarantee is usually lost because some continuous branches are pruned and optimality is not guaranteed because reachable state space is changed [10], [45].

Our approach uses a discrete E* algorithm which is complete and optimal [35] to compute the guiding heuristics for the HE* search. Therefore an existing path for a given resolution can always be found by E*. Moreover if the HE* search can follow the prescribed gradient within the cells where the gradient is defined, then HE* is also complete. To achieve this, each BB motion primitive needs to start on the cell border and exit on the other cell border of the same cell in the direction of the gradient. The entire motion primitive is always inside the cell which is guaranteed by considering a convex hull property of the BB curve. The latter states that a BB curve is always inside the convex envelope of its control points ($\mathbf{P}_i$) and thus also inside the cell provided that $\mathbf{P}_i$ are selected to be inside the cell. Let us name such a node the complete node.

During each HE* search at least one branch needs to lie entirely inside the cells along the gradient to achieve completeness. Moreover, its nodes must not be eliminated even if there exist twins in $OPEN$ or $CLOSED$ lists.

The resulting path using original nodes only results in a nice drivable path but without a completeness guarantee. In practice a reachable goal is almost always found using original nodes only if the BB length is at most equal to a half of the cell size ($\Delta s \leq \frac{1}{2R}$). However, adding the complete nodes ensures that narrow passages are not missed with the original nodes expansion. On the other hand the search consisting of only complete nodes is always complete, however, the resulting path tends to have sharp turns (e.g. in areas around obstacles where the gradient changes abruptly) because it needs to follow the gradient and stay inside the discrete cells. Therefore we propose a combination of the two to preserve the completeness property and obtain good quality drivable paths. The obtained path quality is lower bounded by the solution that would consist of complete nodes only. A brief explanation of node extension during HE* search is as follows. A start node is initialized as a complete node. During the search each complete node has original child nodes with prescribed distance increment Δs and orientation increments $\Delta\varphi_k$ according to (2) and one complete node. Original nodes have only original child nodes provided that there is no twin between the original nodes in $OPEN$ or $CLOSED$ lists. An original child node or a complete child node must not be eliminated if their parent is a complete node and therefore at least this path to the goal is always found if it exists. Due to the original nodes in HE* the optimal path is mostly found consisting of original nodes only or of combinations with complete nodes.

The details on complete node definition are explained in Alg. 2 and in Fig. 8. From a given current node the search extends to a new complete node whose BB path must lie inside its discrete E* counterpart cell ($Cell_{j+1}$). The entry point $\mathbf{S}$ in $Cell_{j+1}$ is the current node end point ($\mathbf{P}_{5,j}$). Gradient $\overrightarrow{\mathbf{p}}$ in $Cell_{j+1}$ guarantees a safe and optimal driving direction for this cell. Therefore the exit point $\mathbf{E}$ on another edge of the cell is obtained by drawing the line from $\mathbf{S}$ in the direction of $\overrightarrow{\mathbf{p}}$. This then defines the last three control points ($\mathbf{P}_{5,j+1} = \mathbf{E}$, $\mathbf{P}_{4,j+1}$ and $\mathbf{P}_{3,j+1}$) with distance $\|\mathbf{E} - \mathbf{S}\|\rho$ among them.

They always lie inside the cell if $\rho < 0.25$. The first three control points ($\mathbf{P}_{0,j+1} = \mathbf{S}$, $\mathbf{P}_{1,j+1}$ and $\mathbf{P}_{2,j+1}$) are defined by the current node final position and orientation. The newly obtained BB curve preserves C^2 continuity with the previous one because it meets the requirements (4) and (7). To ensure that the computed BB curve is entirely inside $Cell_{j+1}$, the location of $\mathbf{P}_{1,j+1}$ and $\mathbf{P}_{2,j+1}$ also needs to be verified. If $\mathbf{P}_{2,j+1}$ is outside the cell, then the relative spacing (ρ) among $\mathbf{P}_{0,j+1}$, $\mathbf{P}_{1,j+1}$ and $\mathbf{P}_{2,j+1}$ is decreased to be within the cell. To improve the BB appearance additional two parameters are defined: d_{FREE} and ε, respectively. The first extends the new complete node on the next discrete cell in case of a short BB ($||\mathbf{S}-\mathbf{E}|| < d_{FREE}$, line 9 in Alg. 2). The second one lowers the curvature in sharp turns of the curve by moving the curve away from the vertex $\mathbf{V}$ (line 12-14 in Alg. 2).

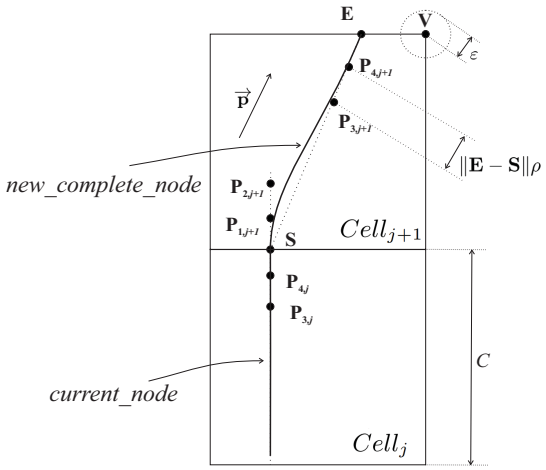

Fig. 8. Illustration of search extension with a new complete node.

As shown the proposed HE* is guaranteed complete but is not guaranteed to find a globally optimal solution for the continuous state space. It applies the discretization of the input space to explore the continuous space with a finite set of nodes and a continuous-state branch pruning where nodes which have twins and higher path cost are not explored. The obtained solution improves by increasing the number of iterations (by lowering the twin thresholds) and by increasing the number of children (with finer orientation increments in (10)) in each node expansion. It can better follow the E* gradient which is approximate for a given resolution R. Closer to optimal solution it therefore also requires finer resolution of E*. However, HE* is guaranteed to find the optimal path inside the build lattice graph obtained during its search. Nevertheless, the produced solution is inherently drivable and it lies in the neighborhood of the global optimal one.

IV. Results and Comparisons

The proposed HE* is compared to the state-of-the-art kino-dynamic planning algorithms: to the Hybrid A* (HA*), SST and lattice planner from the SBPL package that uses ARA*, AD*, and ANA* iterative search algorithms. Algorithm results are measured in terms of path length L, optimal driving time along the planned path T and the number of explored nodes N. The robot is considered as a point in its center of rotation,

Algorithm 2 Complete node determination

Require: $current\_node \leftarrow node_j$, its control points $\mathbf{P}_{i,j}$, $(i \in 0, 1, \cdots, 5)$ and its associated cell in E* $Cell_j$. Used parameters: cell size C, minimal line-of-sight distance d_{FREE}, $(\frac{C}{2} < d_{FREE} < \frac{C}{\sqrt{2}})$, minimal distance from vertex ε, $(0 < \varepsilon < \frac{C}{2})$ and relative distance ρ, ($\rho < 0.25$) between control points

1: Set entry point for $new\_complete\_node \leftarrow node_{j+1}$ as $\mathbf{S} \leftarrow \mathbf{P}_{5,j}$
2: **if** $\mathbf{S}$ is inside $Cell_j$ **then**
3: $new\_cell \leftarrow Cell_j$
4: **else**
5: $new\_cell \leftarrow Cell_{j+1}$ is neighbor sharing the same edge with $Cell_j$
6: **end if**
7: $\overrightarrow{\mathbf{p}} \leftarrow \text{gradient}(new\_cell)$
8: Compute exit point $\mathbf{E}$ on new_cell border, defined by intersection of the straight line from $\mathbf{S}$ following $\overrightarrow{\mathbf{p}}$ and appropriate edge of new_cell
9: **if** $||\mathbf{S}-\mathbf{E}|| < d_{FREE}$ **then**
10: Compute exit point $\mathbf{E}_2$ of the next cell ($Cell_{j+2}$) with gradient $\overrightarrow{p_2} \leftarrow gradient(Cell_{j+2})$. $\mathbf{E}_2$ is defined as intersection of straight line from $\mathbf{E}$ following $\overrightarrow{p_2}$ and the appropriate edge of $Cell_{j+2}$

$$\mathbf{E} \leftarrow \mathbf{E}_2$$
$$new\_cell \leftarrow new\_cell \cup Cell_{j+2}$$

11: **else**
12: **if** $\|\mathbf{E}-\mathbf{V}\| < \varepsilon$ **then**
13: move $\mathbf{E}$ for ε from $\mathbf{V}$ ($\mathbf{E} \leftarrow \mathbf{V} + \frac{\mathbf{E}-\mathbf{V}}{\|\mathbf{E}-\mathbf{V}\|}\varepsilon$)
14: **end if**
15: **end if**
16: Compute BB path for $new\_complete\_node$:

$$\mathbf{P}_{0,j+1} \leftarrow \mathbf{S}$$
$$\mathbf{P}_{1,j+1} \leftarrow \mathbf{S} + \frac{\mathbf{P}_{5,j}-\mathbf{P}_{4,j}}{\|\mathbf{P}_{5,j}-\mathbf{P}_{4,j}\|}\|\mathbf{E}-\mathbf{S}\|\rho$$
$$\mathbf{P}_{2,j+1} \leftarrow \mathbf{S} + \frac{\mathbf{P}_{5,j}-\mathbf{P}_{4,j}}{\|\mathbf{P}_{5,j}-\mathbf{P}_{4,j}\|}\|\mathbf{E}-\mathbf{S}\|2\rho$$
$$\mathbf{P}_{3,j+1} \leftarrow \mathbf{E} - (\mathbf{E}-\mathbf{S})2\rho$$
$$\mathbf{P}_{4,j+1} \leftarrow \mathbf{E} - (\mathbf{E}-\mathbf{S})\rho$$
$$\mathbf{P}_{5,j+1} \leftarrow \mathbf{E}$$

17: **if** $\mathbf{P}_{2,j+1}$ is not inside new_cell **then**
18: Find ρ_N ($\rho_N < \rho$) and recompute

$$\mathbf{P}_{1,j+1} \leftarrow \mathbf{S} + \frac{\mathbf{P}_{5,j}-\mathbf{P}_{4,j}}{\|\mathbf{P}_{5,j}-\mathbf{P}_{4,j}\|}\|\mathbf{E}-\mathbf{S}\|\rho_N$$
$$\mathbf{P}_{2,j+1} \leftarrow \mathbf{S} + \frac{\mathbf{P}_{5,j}-\mathbf{P}_{4,j}}{\|\mathbf{P}_{5,j}-\mathbf{P}_{4,j}\|}\|\mathbf{E}-\mathbf{S}\|2\rho_N$$

to become inside new_cell
19: **end if**

while obstacles are enlarged for the robot's dimensions. The optimal time to drive along the planned path is estimated by using the procedure presented in [46] considering acceleration bounds for translation $a_t \in [-2, 2]$ m/s^2 and rotation $a_r \in [-1, 1]$ m/s^2 and control bounds $v \in [0, 1.5]$ m/s, and $\omega \in [-2.5, 2.5]$ rad/s.

The HA* algorithm [10] uses a similar principle to the proposed HE* algorithm, with the precomputed heuristics to account for obstacles in the environment and additional heuristics to account for holonomic constraints of the vehicle. HA* expands the search tree by using three Reeds-Shepp (RS) curves: a straight line, and left and right turns by a specified angle. The continuous states created by this expansion are merged into a discrete state (x, y, φ). The length and turning

angle of RS curves defines the complexity of the search. If the length and the angle of RS curves are close to discretization step, i.e., the size of a grid cell and the resolution of the orientation, it may result in frequent merging of several continuous states within the same discretization step into a single discrete state. Therefore, the resulting path can have discontinuities at the merging locations. On the other hand, if the length and angle of RS curves are larger than the discretization step, the search is guided completely by the heuristics, less states are searched, and merging of similar states is avoided.

The SST algorithm [30] is a stable sparse version of the RRT algorithm, which is able to achieve asymptotic near-optimality while maintaining a sparse data structure. Motion primitives in SST are random kinodynamic trajectories generated iteratively as the tree expands according to the randomly chosen control applied during the randomly chosen time of propagation. SST has a property of probabilistic completeness. SBPL algorithm is a search algorithm for the state lattice graph. Motion primitives in SBPL are a deterministic and fixed set whose end points are snapped to discrete states. For each state there exists a function for getting predecessors (or successors), which creates them always uniquely. The whole lattice graph is not constructed in advance, but iteratively during states expansion to minimize the memory and computation requirements. SBPL is proven to be complete and optimal. Like SBPL and SST, HE* also creates a search graph during states expansion. Motion primitives are dependent on the continuous expanded state and E* heuristics. It is complete but not optimal since we do not have a unique search tree. HE* path planning algorithm is implemented in Matlab, whereas HA* [47], SST and SBPL package are implemented in C++. Comparisons are made on a personal computer with a dual core processor (i7-7600U) at base frequency of 2.9Ghz.

We conducted two tests on three obstacle configurations maps of size 10 m $\times$ 10 m (U-shape map, S-shape map, and Corridor map). The first test examines the influence of the graph resolution R (the number of cells per meter) on the path planning performance. Here, only SBPL and HE* are compared since SST and HA* are continuous algorithms without optimality guarantee. The second test compares all the algorithms for the best-tuned parameters. In the presented results the length of the HE* paths is estimated numerically where each BB curve in the path is split to several segments (e.g. 100) and the obtained segments lengths are summed.

For the experiments we used a SBPL algorithm with states represented as (x, y, φ), where (x, y) are discrete coordinates of the cells in the two dimensional grid map, and φ is one of the 16 directions of the robot's orientation as a suggested parameter for the motion primitives calculation. For such a setup three search algorithms can be used: the Anytime Repairing A* (ARA*) developed by [37], the Anytime Dynamic A* (AD*) developed by [38], and Anytime Nonparametric A* (ANA*) developed by [39]. All three algorithms are iterative, which means they compute a suboptimal solution fast and then improve it iteratively until the optimal one is found. AD* is developed for dynamic environments and searches more efficiently in case of changes that happen in the environment. Both ARA* and AD* behave identically in static known environments if suboptimality parameter ϵ is initially set to the same value. If the user sets some value $\epsilon > 1$, the heuristics are proportionally increased and a suboptimal solution is computed fast. The algorithm decreases the parameter ϵ in all further iterations until it reaches value 1. If the user sets the initial value $\epsilon = 1$ both algorithms produce the optimal solution in the first iteration. ANA* calculates the first suboptimal solution in shorter time and is fast in improving the solution without setting the initial value for ϵ, i.e., it is calculated automatically.

In the first test, we compared HE* and SBPL by varying resolution of the grid and consequently the search space of SBPL since the discrete states are built upon the 2D grid map. The results are given in Tab. I and the planned paths are presented in Fig. 9. The Corridor map has a one-cell wide passage at the lowest resolution of $R = 2$. Here we chose to present the optimal solution of SBPL, and we chose the fastest AD* executed with parameter $\epsilon = 1$. AD* is executed in the backward direction, i.e., the search starts from the goal state. The search can be started in the forward direction, i.e., from the start, and the same optimal cost of the path will be found, although the path can be slightly different since there exist more equal cost solutions. Every motion primitive in SBPL is a trajectory of 10 interpolated points. From each state there exist 6 motion primitives: long and short straight-line trajectories, circular-arc trajectories with a positive and negative orientation increment, and turn-in-place trajectories with a positive and negative orientation increment. Motion primitives mostly have the length of 8 cells, except the short straight-line trajectory, which has the length of 1 cell, and the turn-in-place trajectory of length 0. This means that by changing the resolution of the grid map the motion primitives change their length. For resolutions 2 and 4 motion primitives are too long for the given obstacle configurations and there only exists a solution with short straight-line and turn-in-place trajectories. Therefore, for $R = 2$ and $R = 4$ we changed motion primitives to have the length of 3 cells to obtain better results. On the other hand, fewer nodes are searched with 8 cells long motion primitives for higher resolution than with 3 cells long motion primitives, which is expected since fewer primitives exist in the same area. Each of 6 motion primitives has an additional gain factor so that e.g. turn-in-place trajectory can be penalized more than straight-line and/or circular-arc trajectories. Then, the optimal solution is a longer trajectory with the minimal number of turns in place. If the gain factor for a turn-in-place trajectory is set to zero, then the optimal path is the shortest possible straight-line segmented path with turns in place at the path direction changes. However, this path has a longer driving time and we chose the equal value of gain factors for a straight-line, circular-arc and turn-in-place motion primitive. Although all path lengths in Tab. I are very similar, it can be seen in Fig. 9 that for a lower resolution the path has some turn-in-place points and a slower driving time.

In this test, we fixed the length of BB of HE*, which is defined with the parameter Δs (see Sec. II) for all map resolutions to $\Delta s = 0.25$ m. It is chosen to be a half of the cell size at the lowest resolution of $R = 2$ to be sure not to overshoot the one-cell wide narrow passage in the

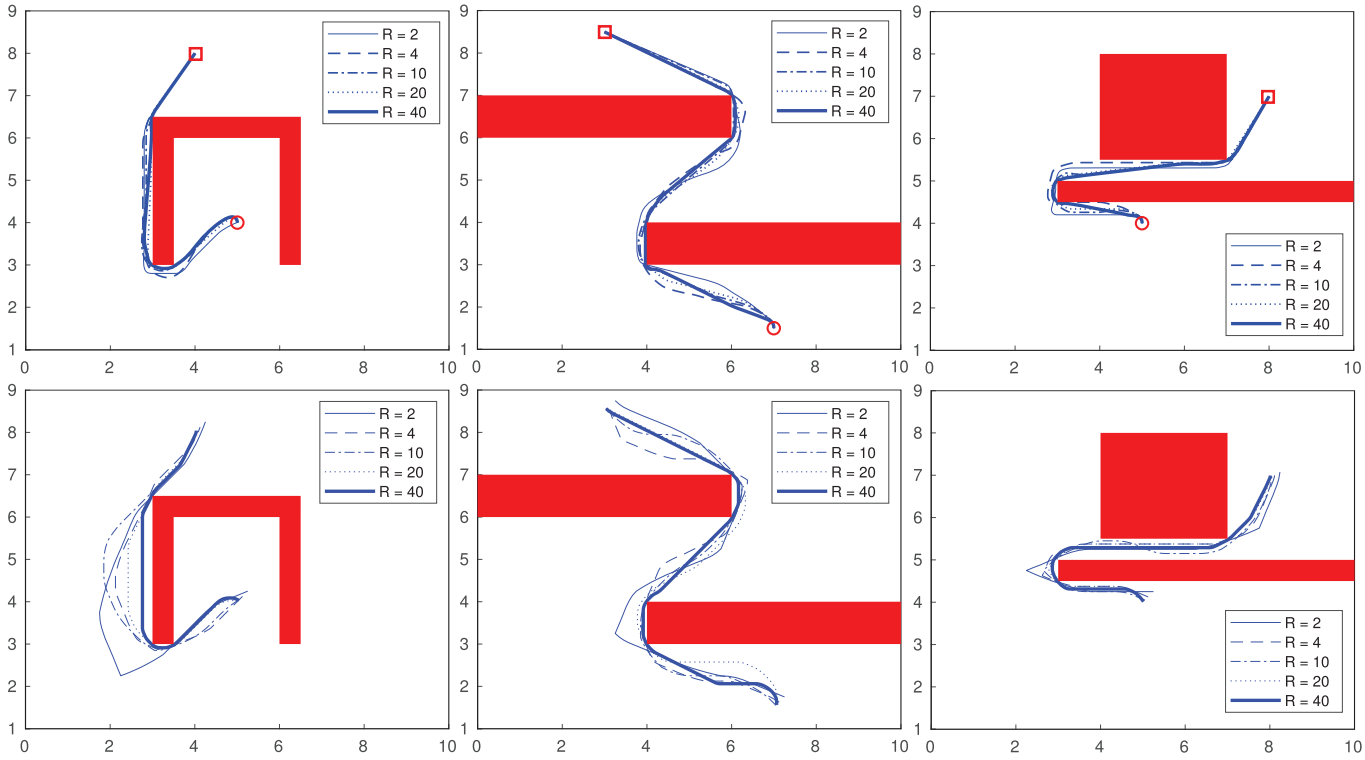

Fig. 9. Robot planned paths for the HE* (upper row) and SBPL (lower row) algorithm in three different maps of size 10 m × 10 m: the U-shape (left), the S-shape (middle), and Corridor (right), where the grid resolution varies for $R \in \{2, 4, 10, 20, 40\}$.

TABLE I
COMPARISON OF HE* AND SBPL ACCORDING TO THE MAP RESOLUTION AND OBSTACLE CONFIGURATIONS

		HE*			SBPL		
Map	R	L [m]	T [s]	N [#]	L [m]	T [s]	N [#]
U-shape	2	8.17	7.94	606	10.50	14.16	4,406
	4	8.32	8.45	561	8.68	10.03	17,779
	10	8.09	8.43	1,031	8.81	8.20	98,720
	20	7.94	7.80	1,648	8.17	8.35	451,966
	40	7.95	8.03	1,388	8.05	9.23	1,950,052
S-shape	2	12.35	11.94	696	12.92	17.22	2,977
	4	12.23	11.84	525	12.57	13.78	12,085
	10	11.87	11.07	689	12.21	12.38	69,779
	20	11.93	11.23	968	12.57	12.10	359,162
	40	11.75	11.25	1,134	11.96	12.82	2,012,626
Corridor	2	9.15	9.17	549	10.18	12.57	2,801
	4	9.13	9.26	399	9.36	10.60	10,017
	10	8.79	9.21	1,533	9.21	9.78	52,343
	20	8.70	9.00	1,795	8.87	8.93	217,607
	40	8.58	8.53	1,037	8.85	9.47	926,373

Corridor map (although the complete BB always ensures that the trajectory through the passage exists, we want to have more BB to search). In general, by decreasing the length of BB, HE* will produce a shorter path as it can better track the guiding heuristics, but this results in sharper turns (though still smooth) around obstacles and consequently takes a longer time to drive along the path. The search tree of BBs by HE* is presented in Fig. 10. It can be seen how some parts of the optimal path contain complete BBs (black curves) where rectangles denote discrete E* cells opened during the complete nodes extensions.

TABLE II
THE INFLUENCE OF THE BB LENGTH ON THE PLANNING PROCESS OF HE*

Map	R	BB length [m]	L [m]	T [s]	N [#]
S-shape	10	0.5	12.21	10.96	257
		0.25	11.87	11.07	689
		0.1	11.85	12.42	1,726
		0.05	11.79	14.50	5,590

The influence of the length of BB on the path quality for the fixed resolution $R = 10$ can be seen in Tab. II and in Fig. 11, where the smallest BB length ($\Delta s = 0.05$ m) produces the shortest path, even shorter than the path obtained with the resolution $R = 20$ (Tab. I) but the number of the explored nodes is much higher since it depends on the BB length. On the other hand, the time to travel along the path is much higher for a smaller BB length (Tab. II: $T = 14.5$ s for BB length of 0.05 m) than for the similar length path of a higher resolution R (e.g. Tab. I: $T = 11.25$ s for $R = 40$ and BB length 0.25 m), which is due to the fact that a shorter BB has higher curvature values.

Comparing the results in Tab. I and Fig. 9 for equal resolution our method produces a shorter path and less time to drive the path than SBPL algorithm. HE* creates a graph of BBs focused around the optimal path of the E* graph search algorithm on the 2D grid map and therefore searches a significantly lower number of nodes comparing to SBPL, which uses a 3D search of motion primitives. The number of the searched nodes in the HE* algorithm does not depend so

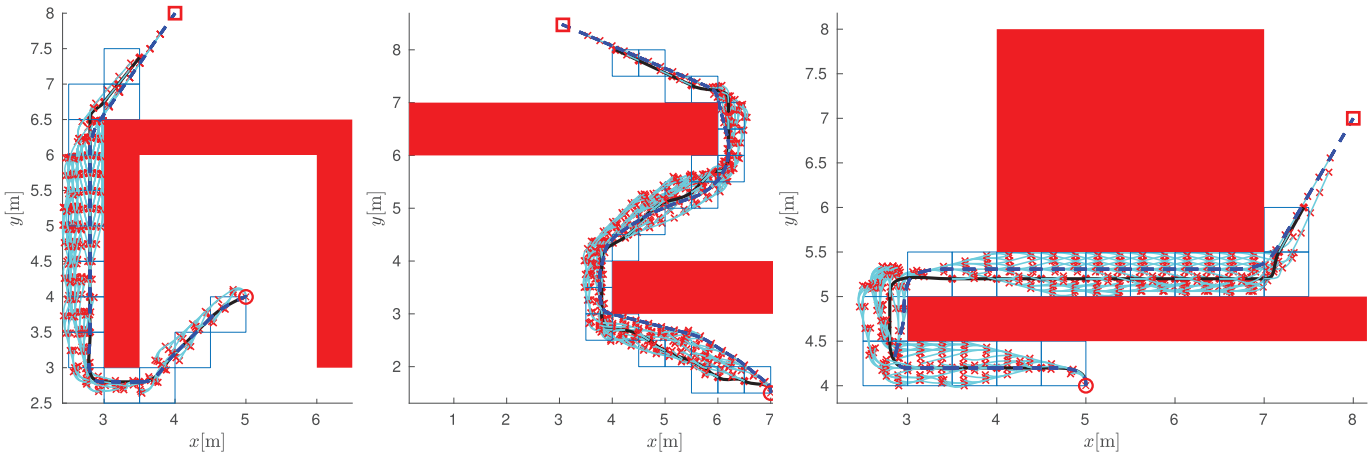

Fig. 10. BB tree searched by HE* in the U-shape (left), S-shape (middle), and Corridor (right) map, for $R = 2$ and BB length 0.25 m, with the noted grid cells (squares), searched nodes (x-mark) as end points of the regular BBs (a thin curve) and complete BBs (a thick curve), and the final path (a dashed line).

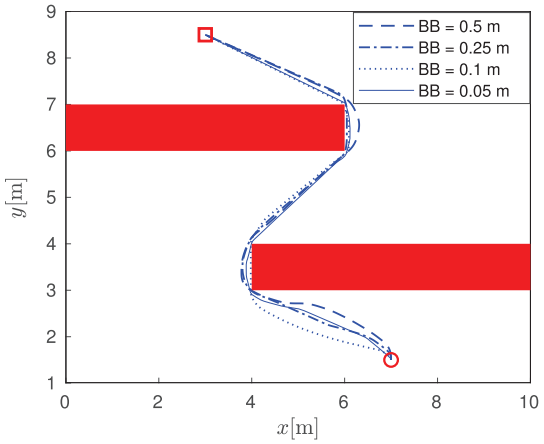

Fig. 11. The robot planned paths with HE* in the S-shape map of resolution $R = 10$ with various BB length.

much on the 2D grid map resolution but more on the node elimination criterion and the number of the used BBs in each node expansion. In the presented test we used four BBs at each node expansion according to (10) also including an additional gradient of the path from the further parent node distanced for the BB length from the current node of the search, as explained in Sec. III. On the other hand, SBPL works better at a higher resolution grid, while at a lower resolution ($R \leq 10$) the computed path has some turn-in-place points, where the robot does not move translationally but only rotates on spot. Furthermore, SBPL uses discretization of nodes according to the 2D grid and a finite number of fixed orientations, so the search will never reach the exact continuous-coordinate goal state. The proposed algorithm HE* finds the path between the continuous-coordinate start and goal, as can be seen in Fig. 9.

The comparison of HE*, HA*, SST, and SBPL algorithms is given in Tab. III and Fig. 12. HE* is executed with BB length 0.25 m and a grid map resolution 2. HA* is executed with RS length 0.25 m, RS angle 25°, grid map resolution 8, and orientation resolution 5°. In SST, we omitted the orientation at the goal state by removing the orientation from the cost function and the random tree generation stops when the goal area is reached (0.5 m around the goal is set since, in general, a dynamic system cannot reach a target state exactly). We needed to tune several parameters of the SST to obtain the best behavior in given environments: the lower and upper bound of the trajectory propagation time is set from 40 to 110 seconds; the radius for BestNear is set to 0.2 m; the radius for sparsification is set to 0.1 m, and the final solution termination is set to be after 30 seconds. SBPL is executed with the resolution 10 of the grid map so that lattice graph can be clearly seen, although a higher resolution is preferred and the lattice graph is denser. In SBPL we could not omit the orientation at the goal state since it determines the state in the lattice graph. Since we are testing a static environment scenario, both ARA* and AD* behave identically, so only AD* and ANA* are presented. Both AD* and ANA* are executed in the backward direction, i.e., the search starts from the goal state. All three iterative algorithms are tested according to the first and the final solution found. ϵ for AD* is set to a default value 3 for the suboptimal first solution and to a value 1 for the final solution.

As can be seen in Tab. III ANA* has a longer search of the final solution than AD* (more explored nodes), which is expected since we set $\epsilon = 1$ for the optimal solution of AD* so there is only one iteration for finding the solution and fewer nodes to be searched than when iteratively searching the optimal solution. Searching the optimal solution with SBPL (Fig. 12) results with a dense tree of motion primitives, while searching the first suboptimal solution with ANA* has 100 times fewer nodes than AD*. Trajectory lengths for optimal solutions of AD* and ANA* are not equal since costs of turn-in-place trajectories ware also included in the path optimization to have a better driving time T. SST has the worst first solution (the longest path) but has a close-to-optimal length of the final solution and a relatively low number of the explored nodes, although the tree propagation is very costly (the algorithm ran for 30 seconds). HA* results in a low path length, and a minimal number of the explored nodes. However, due to the constant RS angle it yields high-curvature paths. Tests with RS angle of 15° produce smoother but longer paths, while very tight spaces (e.g. Corridor map) require RS angle to be 30°. HE* has a similar path to the optimal path of SBPL

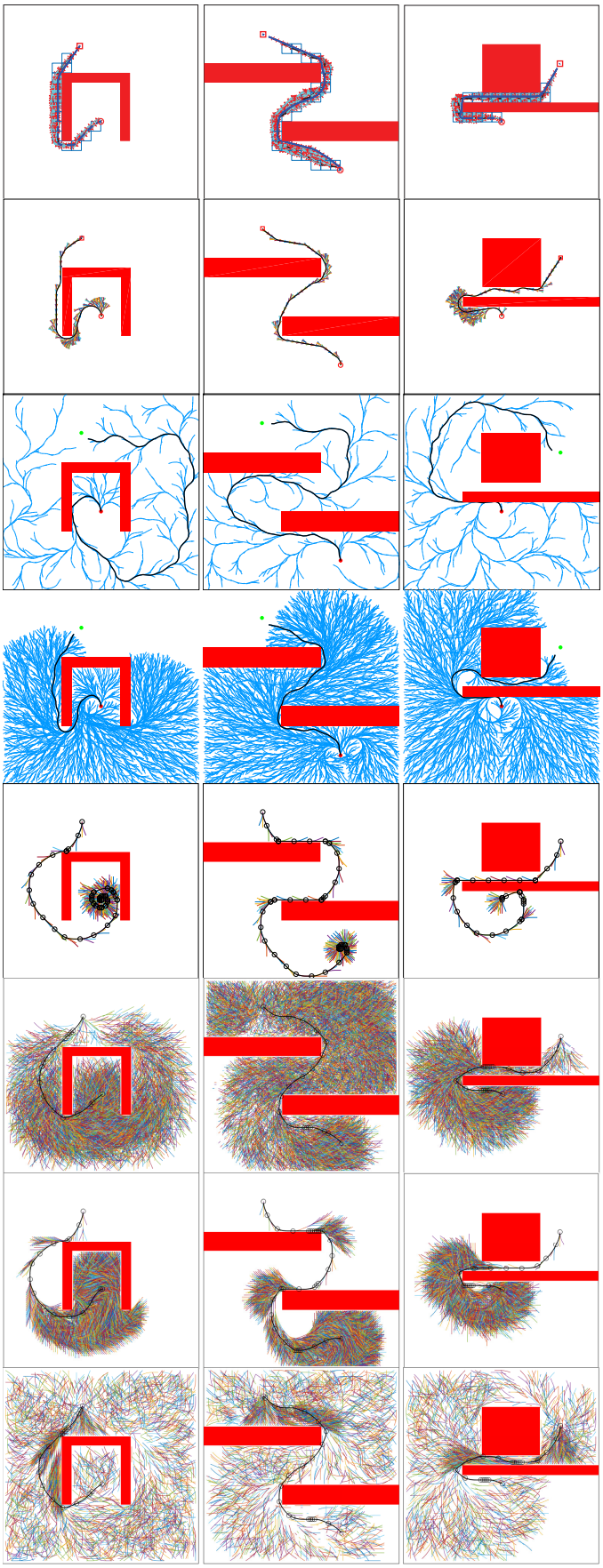

Fig. 12. Planning with: 1) HE*; 2) HA*; 3) SST (first iteration); 4) SST (final solution); 5) SBPL ANA* (first iteration); 6) SBPL ANA* (final solution); 7) SBPL AD* (first iteration); 8) SBPL AD* (final solution) in three different maps (10 m × 10 m): U-shape (left), S-shape (middle), and Corridor (right).

TABLE III
ALGORITHM COMPARISON ON THREE DIFFERENT MAPS

Map	Algorithm	L [m]	T [s]	N [#]
U-shape	HE*	8.17	7.94	606
	HA*	9.15	9.52	330
	SST* 1st sln	22.61	22.72	1,851
	SST* final sln	9.49	10.12	9,681
	SBPL ANA* 1st sln	15.65	24.08	104
	SBPL ANA* final sln	8.81	8.11	286,803
	SBPL AD* 1st sln	10.12	11.02	8,074
	SBPL AD* final sln	8.81	8.20	98,720
S-shape	HE*	12.35	11.94	696
	HA*	12.36	12.42	229
	SST* 1st sln	21.82	21.23	1,416
	SST* final sln	12.61	12.85	11,201
	SBPL ANA* 1st sln	18.82	23.72	82
	SBPL ANA* final sln	12.21	12.11	789,563
	SBPL AD* 1st sln	15.10	16.44	6,039
	SBPL AD* final sln	12.21	12.38	69,779
Corridor	HE*	9.15	9.17	549
	HA*	9.28	9.97	499
	SST* 1st sln	15.51	15.29	1,901
	SST* final sln	10.44	11.20	11,007
	SBPL ANA* 1st sln	14.64	16.72	53
	SBPL ANA* final sln	9.21	9.78	77,368
	SBPL AD* 1st sln	8.98	11.01	8,847
	SBPL AD* final sln	9.21	9.78	52,343

and a lower number of the explored nodes. It can also be used in dynamic environments since heuristics calculated by E* can be incrementally updated as environment changes, i.e. E* performed from the goal node (backward search) preserves dynamic properties inherited from the D* algorithm.

A more detailed statistical comparison is performed on a set of randomized environments of size 20 m × 20 m with randomly selected start and goal locations and different obstacles densities (20%, 30% and 50%). HE* is compared to SBPL-AD* and SST which performed the best in previous comparisons shown in Fig. 12 and Tab. III. Both SBPL-AD* and HE* run on a grid map of the resolution 10. An example of a random environment with a 50% obstacle density is shown in Fig. 13. Although HA* is presented in Fig. 13, it is not compared further since it produces longer paths than SBPL, and different obstacle configurations require different turning angle of RS curves. Results obtained in 19 randomized environments are shown in Fig. 14. Tab. IV gives statistic performance measured in terms of mean, standard deviation, minimal and maximal values of normalized results where HE* and SST are normalized with AD* result values. The performance of HE* is similar to AD* for L and T but explores a much lower number of nodes while AD* performs better in environments where obstacles do not require sharp turns. SST performs the worst and would require many more iterations to converge to the other two planners. A comparison of execution times is not provided due to the different implementation and optimization of the algorithms. Therefore, the algorithm complexity is rather measured in terms of the explored nodes which is more fair. However, it needs to be mentioned that SST has a more expensive nodes expansion calculation since

TABLE IV
A COMPARISON OF HE* AND SST NORMALIZED WITH THE VALUES OBTAINED WITH THE SBPL-AD* ALGORITHM

	HE*			SST		
	$\frac{L}{L_{\text{SBPL}}}$	$\frac{T}{T_{\text{SBPL}}}$	$\frac{N}{N_{\text{SBPL}}}$	$\frac{L}{L_{\text{SBPL}}}$	$\frac{T}{T_{\text{SBPL}}}$	$\frac{N}{N_{\text{SBPL}}}$
mean	0.951	0.970	0.098	1.118	1.422	1.913
std	0.056	0.054	0.194	0.083	0.172	3.201
min	0.757	0.808	0.001	1.006	1.061	0.034
max	1.011	1.054	0.882	1.281	1.695	14.186

randomly generated trajectories consist of 50,000 points in average (an average duration of 100 s divided by the sampling time of 0.002 s), while SBPL motion primitives consist of 10 points, and BB of 6 points.

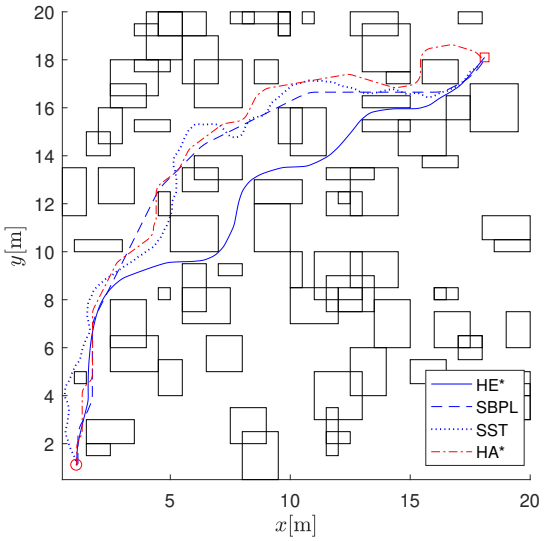

Fig. 13. An example of a random environment (experiment 16 in Fis. 14) with a 50% obstacle density and a pair of start-goal with the obtained paths by HE*, HA*, SBPL-AD* and SST.

Another test is performed on our department map of size 52 m $\times$ 15 m, recorded at resolution $R = 10$, which is a commonly used resolution for a gridmap of the real-world environment where the 0.5-m-wide robot moves, see Fig. 15. Three planned paths are calculated with the length of BB equal to 0.25 m, and with three different start positions noted by o-mark and one common goal position noted by □-mark.

V. CONCLUSION

HE* planner with a guaranteed completeness is proposed. It produces smooth paths with a continuous curvature that the wheeled robot can easily follow. The motion primitives used are based on the fifth order Bernstein-Bézier curves which are defined in a way to preserve continuity in C^2 as well as keeping a good search expansion at path planning. They are not fixed as their shape can easily adapt the length and the orientation change according to the used heuristics. This results in a shorter, smoother and easily followed path obtained at a lower computational complexity compared to the case of preset and fixed set of motion primitives. The resulting smooth path enables a faster robot motion as the

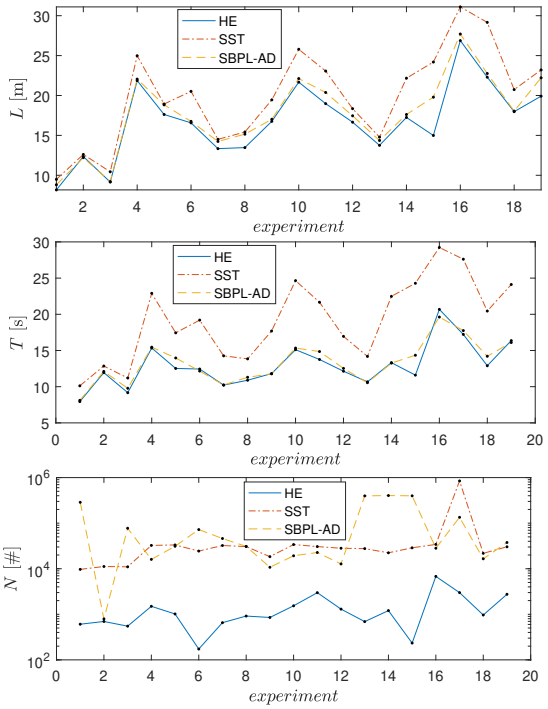

Fig. 14. A comparison of the path lengths L, the driving times T, and the numbers of searched nodes N obtained with HE*, SST and AD* on randomized environments.

Fig. 15. Three planned paths at the department map obtained by the HE* algorithm. The start of each trajectory is noted by o-mark and a common goal by □-mark.

robot does not need to slow down to such an extent or even stop before making sharper turns. To obtain a fast but still safe and feasible robot tracking of a planned path we computed an optimal velocity profile according to maximal acceleration and velocity constraints. This also enables an estimation of the minimal required time for the robot to drive along the path. The proposed approaches were tested by several experiments and compared to the state-of-the-art motion planner in terms of a path length, traveling time which implicitly depends on the curvature of the path, and of the algorithmic complexity measured by a number of the searched nodes. In most of the cases HE* produces shorter as well as faster trajectories at a much lower computational complexity. The obtained path is given parametrically and is not snapped to the discrete grid as is usually the case with most deterministic motion-primitives based planners. This means that HE* can find good quality paths even at a rougher resolution of its heuristic part. Future challenges will deal with the integration of the path following controller and the drivable path re-planning to efficiently account for changes in the environment.

REFERENCES

[1] J. Schwartz and M. Sharir, *Algorithms and Complexity*. Elsevier, 1990, ch. Algorithmic motion planning in robotics, pp. 391–430.

[2] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[3] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[4] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementations*, R. C. Arkin, Ed. MIT Press, 2005.

[5] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, *Wheeled mobile robotics - From fundamentals towards autonomous systems*. Elsevier, Butterworth-Heinemann, 2017.

[6] A. Konar, S. Member, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, 2013.

[7] M. Kallmann, "Path Planning in Triangulations," in *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland, 2005, pp. 49–54.

[8] J. Sethian, "Fast Marching Methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.

[9] T. Berglund, U. Erikson, H. Jonsson, K. Mrozek, and I. Soderkvist, "Automatic generation of smooth paths bounded by polygonal chains," in *Proc. of the International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA)*, M. Mohammadian, Ed., Las Vegas, 2011, pp. 528–535.

[10] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[11] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai, and C. Su, "Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 740–749, June 2016.

[12] M. Seder, M. Baotić, and I. Petrović, "Receding horizon control for convergent navigation of a differential drive mobile robot," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 653–660, March 2017.

[13] I. Maurovic, M. Seder, K. Lenac, and I. Petrovic, "Path planning for active slam based on the D* algorithm with negative edge weights," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 8, pp. 1321–1331, Aug 2018.

[14] X. Yang and C. Wushan, "AGV path planning based on smoothing A* algorithm," *International Journal of Software Engineering and Applications (IJSEA)*, vol. 6, no. 5, pp. 1–8, 2015.

[15] J.-W. Choi, R. Curry, and G. Elkaim, *Machine Learning and Systems Engineering*, ser. Lecture Notes in Electrical Engineering 68. Springer Science+Business Media, 2010, ch. Piecewise Bezier Curves Path Planning with Continuous Curvature Constraint for Autonomous Driving, pp. 31–45.

[16] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, USA, 2009, pp. 2427–2433.

[17] C. Chen, Y. He, C. Bu, J. Han, , and X. Zhang, "Quartic Bezier Curve based Trajectory Generation for Autonomous Vehicles with Curvature and Velocity Constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014, pp. 6108–6113.

[18] J. Yang and A. Yuen, "An analytical local corner smoothing algorithm for five-axis CNC machining," *International Journal of Machine Tools and Manufacture*, vol. 123, pp. 22–35, 2017.

[19] M. Brezak and I. Petrović, "Real-time approximation of clothoids with bounded error for path planning applications," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 507–515, April 2014.

[20] S. Gim, L. Adouane, S. Lee, and J.-P. Dérutin, "Clothoids Composition Method for Smooth Path Generation of Car-Like Vehicle Navigation," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 88, no. 1, pp. 129–146, 2017.

[21] F. Ghilardelli, L. Gabriele, and A. Piazzi, "Path Generation Using $\eta 4$-Splines for a Truck and Trailer Vehicle," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 187–203, 2014.

[22] B. Sencer, K. Ishizaki, and E. Shamoto, "A curvature optimal sharp corner smoothing algorithm for high-speed feed motion generation of NC systems along linear tool paths," *International Journal of Advanced Manufacturing Technology*, vol. 76, no. 9, pp. 1977–1992, 2015.

[23] Y. Zhou, H. Hu, Y. Liu, and Z. Ding, "Collision and deadlock avoidance in multirobot systems: A distributed approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1712–1726, July 2017.

[24] L. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *Amer. J. Math.*, vol. 79, pp. 497–516, 1957.

[25] A. Zdešar and I. Škrjanc, "Optimum velocity profile of multiple bernstein-bezier curves subject to constraints for mobile robots," *ACM Trans. Intell. Syst. Technol.*, vol. 9, no. 5, pp. 56:1–56:23, 2018.

[26] S. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[27] M. Otte and E. Frazzoli, "RRTx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[28] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5054–5061.

[29] S. Yoon, D. Lee, J. Jung, and D. H. Shim, "Spline-based RRT* using piecewise continuous collision-checking algorithm for car-like vehicles," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 3-4, pp. 537–549, 2018.

[30] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[31] M. Pivtoraiko, R. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[32] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[33] M. Montemerlo and et al., "Junior: The Stanford entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[34] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings IEEE International Conference on Robotics and Automation*, 1994, pp. 3310–3317.

[35] R. Philippsen and R. Siegwart, "An interpolated dynamic navigation function," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, 2005, pp. 3782–3789.

[36] M. Đakulović, M. Čikeš, and I. Petrović, "Efficient interpolated path planning of mobile robots based on occupancy grid maps," in *10th IFAC Symposioum on Robotic Control (SYROCO2012)*, 2012, pp. 349–354.

[37] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774.

[38] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm." in *ICAPS*, 2005, pp. 262–271.

[39] J. Van Den Berg, R. Shah, A. Huang, and K. Goldberg, "ANA*: anytime nonparametric A*," in *Proceedings of twenty-fifth AAAI conference on artificial intelligence (AAAI-11)*, vol. 2, 2011, p. 1.

[40] A. Shamsudin, K. Ohno, R. Hamada, S. Kojima, N. Mizuno, T. Westfechtel, T. Suzuki, S. Tadokoro, J. Fujita, and H. Amano, "Two-stage hybrid A* path-planning in large petrochemical complexes," in *Advanced Intelligent Mechatronics (AIM), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1619–1626.

[41] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic A*," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 224–243, 2016.

[42] J. Sethian, "A fast marching level set method for monotonically advancing fronts," in *Proceedings of the National Academy of Sciences*, vol. 93, no. 4. National Acad Sciences, 1996, pp. 1591–1595.

[43] A. Nash and S. Koenig, "Any-angle path planning," *Artificial Intelligence Magazine*, vol. 34, no. 3, pp. 85–107, 2013.

[44] M. Bowling and M. Veloso, "Motion control in dynamic multi-robot environments," in *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '99)*, Monterey, CA, 1999, pp. 168–173.

[45] J. Barraquand and J. C. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *Algorithmica*, vol. 10, no. 2-4, pp. 121–155, Oct. 1993.

[46] M. Lepetič, G. Klančar, I. Škrjanc, D. Matko, and B. Potočnik, "Time optimal path planning considering acceleration limits," *Robotics and Autonomous Systems*, vol. 45, pp. 199–210, 2003.
[47] T. Gupta and R. Kranti Kiran, "Hybrid A* path planner," https://github.com/tejus-gupta/hybrid-astar-planner, Nov. 2017.

Gregor Klančar received the B.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana, Faculty of Electrical Engineering, Slovenia, in 1999 and 2003 respectively.

He is currently an Associate Professor with the Faculty of Electrical Engineering, University of Ljubljana. He lectures autonomous mobile systems at graduate and advanced control of autonomous systems at postgraduate study. His current research interests include autonomous mobile robots, motion control, trajectory tracking, path planning, localization and agent-based behaviour systems.

Marija Seder (M'05) received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Zagreb, Croatia, in 2004 and 2010, respectively.

She was a Visiting Researcher with the Autonomous Intelligent System Group, University of Freiburg, Freiburg im Breisgau, Germany, from 2012 to 2013, under the supervision of Prof. W. Burgard. She is currently an Assistant Professor with the Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering, University of Zagreb. Her current research interests include mobile robotics, especially motion planning, path planning, coverage planning, obstacle avoidance, and environment exploration.

Sašo Blažič received the B.Sc., M. Sc., and Ph. D. degrees in 1996, 1999, and 2002, respectively, from the University of Ljubljana, Faculty of Electrical Engineering, Slovenia.

He is currently a Professor with the University of Ljubljana, Faculty of Electrical Engineering. From 2017, he holds the position of the Vice-Dean for Research at the Faculty of Electrical Engineering, University of Ljubljana. His research interests include adaptive, fuzzy and predictive control of dynamical systems, modelling of nonlinear and evolving systems, autonomous mobile systems, trajectory tracking and path planning of wheeled mobile robots.

Igor Škrjanc received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the Faculty of Electrical and Computer Engineering, University of Ljubljana, Ljubljana, Slovenia, in 1988, 1991, and 1996, respectively.

He is currently a Full Professor with the Faculty of Electrical Engineering and the Head of the Laboratory for Autonomous and Mobile Systems. His main research interests include adaptive, predictive, neuro-fuzzy, and fuzzy adaptive control systems. Dr. Skrjanc is a Humboldt Research Fellow, a Research Fellow of the Japan Society for the Promotion of Science, and the Chair of Excellence at the University Carlos III of Madrid. He also serves as an Associate Editor for multiple SCIE journals.

Ivan Petrović Ivan Petrović (M'97) is a professor and the head of the Laboratory for Autonomous Systems and Mobile Robotics at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. He is also the co-director of the Centre of Research Excellence for Data Science and Cooperative Systems. His research is focused on the advanced control and estimation techniques and their application in control and navigation of autonomous mobile robots and vehicles. He published more than 60 journal papers and 200 conference papers. Results of his research effort have been implemented in several industrial products. He is a Chair of the IFAC Technical Committee on Robotics, and a full member of the Croatian Academy of Engineering.