Efficient Navigation for Anyshape Holonomic Mobile Robots in Dynamic Environments

Marija Đakulović* Christoph Sprunk‡ Luciano Spinello‡ Ivan Petrović* Wolfram Burgard‡

***Abstract*— Platforms with holonomic drives are particularly interesting due to their maneuvering capabilities. Robots used for transportation tasks usually have a non-circular footprint. In this work, we present a navigation strategy for a holonomic mobile robot with anyshape footprint. Our technique introduces an efficient navigation method based on a strategy that makes use of discrete and continuous techniques. We introduce compact discrete intervals to represent the free space for computing fast-to-update plans. Based on these, we provide a continuous motion generation approach to generate smooth motions that are fast to compute. We evaluated our approach by running simulated experiments and by using a real holonomic L-shaped robot. Our experiments demonstrate that our technique can be carried out online and is able to smoothly drive the robot to its goal locations even in dynamic environments.**

I. Introduction

Nowadays, mobile platforms with holonomic drives are particularly appealing for industrial applications due to their high maneuverability. Especially robots designed to perform transportation tasks typically have a rectangular footprint design to accommodate a loading bay. However, overhanging payloads may modify the footprint shape of the robot. In robotics, many planning and motion execution algorithms, for the sake of efficiency, approximate the robot footprint with a circle. This assumption potentially limits the mobility of non-circular robots especially in confined environments such as corridors or narrow passages or even in dynamic environments. In this paper, we propose a navigation approach for arbitrarily shaped (anyshape) holonomic robots, that is efficient and is able to deal with dynamic environments. With respect to other existing anyshape robot navigation techniques [6, 11, 17], our approach introduces a fast, reactive navigation technique based on a discrete-continuous processing pipeline. It generates a plan that can efficiently be updated using a compact discrete representation of the free space. By using this discrete plan, it generates an efficiently computable, smooth and stable motion. The main contributions of our approach consist of: (i) a novel graph representation of the collision-free configuration space that is compact in the number of nodes, supports incremental updates to react to changes in the environment and is suitable for incremental path planners; (ii) the use of this graph for continuous motion generation that computes smooth motions in a highly reactive manner and that is proven to be Lyapunov asymptotically stable. By using this discrete-continuous strategy, both planning and motion generation can run online and thus enable the application in dynamically changing environments of an anyshape holonomic robot. In particular, we propose a novel way of compactly representing the collision free space in the environment, called orientation interval graph (OIG). Based on this graph, the computationally expensive problem of exhaustively searching for collision-free configurations is greatly reduced. The idea is to create a graph of valid orientation intervals on top of non-occupied locations of a grid map. Our motion generation is a fast reactive approach based on the receding horizon control (RHC) principle [4]. It is implemented as a closed-loop controller that guarantees that the robot will not get stuck in local minima. This controller incorporates the discrete OIG search path costs in the form of an interpolated continuous cost-to-goal function. Note that both the replanning in the OIG and the RHC optimization are computed online. We quantified the performance of our motion planning algorithm in simulations and on a real robot. Our experiments demonstrate a speedup of up to a factor of 60 in planning time.

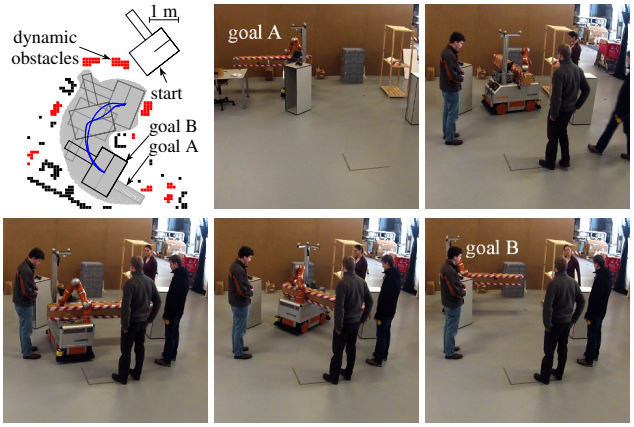

Fig. 1. An L-shaped robot navigating through opposingly oriented goals (A, B) with complex maneuvers and turns on the spot. Our method runs online on a real robot in dynamic environments.

II. Related Work

To reduce the search space, the anyshape motion planning methods presented in [6, 11] use a discrete set of motion primitives to build a state lattice graph. In [17], a reduction of the search space is obtained by fitting spheres into the environment and searching for a path composed of intersecting spheres. In contrast to these approaches, we do not sacrifice admissible configurations from the state space.

The problem of high dimensional search in motion planning for practical applications has lately been addressed by

*Department of Control and Computer Engineering, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
‡Department of Computer Science, University of Freiburg, Germany
This work has been supported by the European Commission under FP7-285939-ACROSS, FP7-260026-TAPAS, and FP7-248873-RADHAR.

"anytime" algorithms that quickly identify an initial feasible plan, and then improve this plan towards an optimal solution during the execution. A widespread representative of such methods is the rapidly-exploring random tree search (RRT). Whereas this approach finds feasible solutions and paths in high-dimensions [9], it assumes that the environment is static and tends to converge to a solution that is sub-optimal. Therefore, paths computed by RRT-based motion planning algorithms are typically post-processed to reduce the effects of randomization [3, 5]. Even though numerous methods for motion planning and trajectory optimization can be found in the literature, the trade-off between the optimality of the solution and the computational effort is still crucial. Stabilizing receding horizon controllers (RHC) are motion planners that avoid local minima [10, 13]. For practical reasons, many methods assume that the state space is collision-free. However, in real environments this assumption does not hold and as soon as the configuration space is non-convex, RHC cannot be applied [12]. This problem can be solved by splitting the configuration space into convex parts and by optimizing them separately [10]. The convergent dynamic window approach (CDW) [13] uses an interpolated continuous version of the navigation function [7] as a control Lyapunov function. The previously mentioned methods solving the stabilization problem are based on continuous analysis. However, the robot motion planning problem has a discrete-time nonlinear control loop. Our motion planning method uses [1] to prove asymptotic stability of the non-smooth discrete-time nonlinear closed loop system. Our CDW idea shares some ground with [13]. We consider anyshape robots and present a discrete stability analysis. Our search technique is in spirit similar to [14]. They propose to compress adjacent timesteps, while we instead merge adjacent collision free orientations.

III. Path Planning with Orientation Intervals

For navigating in an environment, we need a way to determine whether a certain configuration of an anyshape footprint robot is colliding with the environment. In this paper, we present a compact representation of all admissible configurations that can be used for graph-based planning in dynamic environments.

Let $(\tilde{x}, \tilde{y})$ be the center of rotation of the robot footprint in the plane and $\tilde{\theta}$ be its orientation. We define the configuration space discretization as $(x, y, \theta) = (\lfloor \tilde{x} \rfloor, \lfloor \tilde{y} \rfloor, \lfloor \tilde{\theta} \rfloor)$. In practice, this defines a 3D grid map of the environment (positions and orientations). For the orientation, we use M bins, where $M = \lceil \frac{2\pi}{1/r} \rceil$. In case of anyshape robots, a commonly used orientation bin size is $1/r$, where r is the radius in grid cells of the circumcircle of the robot around its center of rotation. This choice ensures that no point on the robot contour translates more than one grid cell if the robot changes its orientation by one increment, i.e., $1/r$ rad. In this paper, we introduce a state space representation that aims to reduce the search space for path planning. We do so by merging discrete orientations θ into *orientation intervals* $\Theta_1, \ldots, \Theta_l$, where

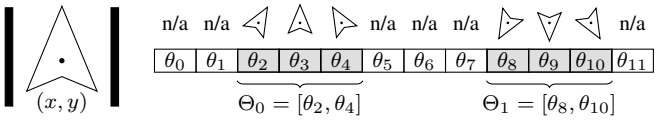

Fig. 2. A dart-shaped robot in a narrow passage (left). We collapse discretized collision-free configurations (x, y, θ_i) into collision-free orientation intervals (x, y, Θ_j) for a specific position (x, y).

$\Theta = [\theta_l, \theta_u]$ for each location (x, y). Adjacent collision-free orientations are collapsed into orientation intervals by using connected-components clustering (see Fig. 2). The set of collision free configurations $\mathcal{C}_{free}$ with elements (x, y, θ) is now represented by a (much) smaller set of collision free orientation intervals with elements (x, y, Θ) without any loss of information.

A. Orientation Interval Graph

We introduce an *orientation interval graph* $\mathcal{G}_{OI}$ that represents the set of discretized collision-free configurations $\mathcal{C}_{free}$ and its connectivity. We make use of [8] to efficiently retrieve all the valid configurations in the environment. This approach maintains incrementally updatable convolutions of the robot footprint with the environment. By storing the number of cells colliding with the footprint for each discretized configuration, collision checks can be performed with a single lookup.

We define the nodes of the orientation interval graph $\mathcal{G}_{OI}$ to be the set $\{(x, y, \Theta)\}$ containing for each location (x, y) all orientations compressed in their respective orientation intervals, Θ. An edge exists between two neighboring nodes of $\mathcal{G}_{OI}$ if they represent neighboring locations and if the respective orientation intervals intersect, such that there exists a way for the robot to transition from one to the other location. Formally, an edge $e_{p,q}$ exists between node $p := (x_p, y_p, \Theta_p)$ and node $q := (x_q, y_q, \Theta_q)$ iff the following equation holds:

$$(x_q, y_q) \in \mathcal{N}(x_p, y_p) \wedge (\Theta_p \cap \Theta_q \neq \emptyset) \;, \tag{1}$$

where $\mathcal{N}(x, y)$ is the set of 2D neighbors of location (x, y) in the discretized configuration space (typically a 4-connected or 8-connected neighborhood). Each edge $e_{p,q}$ has a weight $w_{p,q} > 0$, which defines the cost of transition between nodes p and q. For this work, we have chosen costs that depend on the cost of translation λ_t between neighboring node locations and on the size of the intersection interval, with lower cost for bigger intersection intervals, i.e.,

$$w_{p,q} := \lambda_t \cdot (M + 1 - |\Theta_p \cap \Theta_q|) \,. \tag{2}$$

In case the environment changes, we update the set of orientation intervals for the affected locations by adding or removing the appropriate orientations from the orientation intervals. In this way, the $\mathcal{G}_{OI}$ can efficiently model dynamic environments, and it can be used for incremental search techniques.

B. Searching in an Orientation Interval Graph

The graph search algorithm computes for a given graph $\mathcal{G}_{OI}$ a path $\mathcal{P}(s, g)$ from any node s to the node g with the cost $c(\mathcal{P})$ if one exists. The cost of the path $c(\mathcal{P})$ is

defined as the sum of its edge weights. In this paper, we use the D* algorithm [16] variant without heuristics for its replanning capabilities. When applied to $\mathcal{G}_{OI}$, the D* graph search computes the optimal paths and path costs for every node p to the goal node g. However, this might be a too coarse approximation for a precise and smooth execution of a path. To this end, a finer cost can be computed for each configuration represented by a node through the introduction of an edge weighting scheme during the D* search by making use of *desired orientations*. The desired orientation is the orientation that the robot needs to assume when moving from p to neighbor node q in the safest way possible. We define this to be the middle orientation $\hat{\theta}_q$ of the interval intersection $\Theta_p \cap \Theta_q$ as this typically implies the largest distance to colliding configurations. Given that nodes and edges of $\mathcal{G}_{OI}$ exist only in the collision free space, any orientation choice is guaranteed to be collision free. Now, the edge weight becomes

$$w_{p,q} := \lambda_t \cdot (M + 1 - |\Theta_p \cap \Theta_q|) + \lambda_r \cdot |\hat{\theta}_p - \hat{\theta}_q| \,, \quad (3)$$

where λ_r is the cost of rotation for one orientation increment, and $|\hat{\theta}_p - \hat{\theta}_q|$ is the minimal distance between desired orientations of nodes p and q, while taking into account that rotation is inside the intersection interval $\Theta_p \cap \Theta_q$. Note that the desired orientation for the goal node $\hat{\theta}_g$ is set to the goal orientation. If the intersection interval contains all orientations then desired orientation is set to the desired orientation of the parent node. The D* search starts from the goal node and desired orientations are computed while the search expands new nodes backwards from the goal. This means that the desired orientation of a node changes each time the search discovers a lower cost path to it. Note that D* replanning treats these updates as changes in the graph. After planning, fine costs $\phi(\cdot)$ can be retrieved for each configuration (x, y, θ) as:

$$\phi(x, y, \theta) = c(\mathcal{P}(p, g)) + \lambda_r \cdot |\hat{\theta}_p - \theta| \,, \quad (4)$$

where $(x, y, \theta) \in p$ and $|\hat{\theta}_p - \theta|$ corresponds to the cost of orienting the robot to the desired orientation, while taking into account that rotation is inside of the interval Θ_p. In case the environment changes, the orientation interval graph is updated. To allow proper processing of the graph update by D*, we interleave its replanning with the update of the graph structure.

Note that the resulting path is not guaranteed to be optimal with respect to this new cost function since we do not examine all possibilities of desired orientations for each node. Optimality can be achieved by keeping track of as many desired orientations for a node as there are edges to its neighbors, since each intersection interval with the neighbor yields a distinct middle orientation.

IV. Receding Horizon Control on Discrete Paths

Based on the discrete solution of the path planning problem we now introduce a fast and stable motion execution scheme. The working principle behind our motion control is based on receding horizon optimal control (RHC) [4]. In the following, we use only continuous states and drop the tilde notation for readability. Consider a discrete-time nonlinear time-invariant holonomic robot model:

$$s(t+1) = f(s(t), u(t)), \; t \in \mathbb{N}_0 \,, \quad (5)$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is some given time-invariant nonlinear function, $s \in \mathbb{R}^n$ is the system state vector, and $u \in \mathbb{R}^m$ is the control input. We use a representation of the robot state defined by $s = (x, y, \theta)$, the control input defined by $u = (v_x, v_y, \omega)$, and f defined by the state equation

$$\begin{aligned} x(t+1) &= x(t) + (v_x(t)\cos\theta(t) - v_y(t)\sin\theta(t))\,\Delta t \\ y(t+1) &= y(t) + (v_x(t)\sin\theta(t) + v_y(t)\cos\theta(t))\,\Delta t \\ \theta(t+1) &= \theta(t) + \omega(t)\Delta t \,, \end{aligned} \quad (6)$$

where $t \in \mathbb{N}_0$ and Δt is the duration of each sampling interval. Let $\{s_k\}_0^N$ be the evolution of the system state over a fixed horizon N (as a function of the control sequence $\{u_k\}_0^{N-1}$), according to the model (6), starting with $s_0 = s(t)$. At the current time t, and for the current state $s := s(t)$, the RHC problem is formulated as follows

$$P_N(s): \quad J^*(s) := \min_{\{u_k\}_0^{N-1}} J(\{s_k\}_0^N, \{u_k\}_0^{N-1}) \,, \quad (7)$$

subject to:

$$s_{k+1} = f(s_k, u_k), \text{ for } k = 0, \ldots, N-1 \,, \quad (8)$$

$$s_0 = s \,, \quad (9)$$

$$u_k \in \mathcal{U}, \quad \text{for } k = 0, \ldots, N-1 \,, \quad (10)$$

$$s_k \in \mathcal{C}_{free}, \quad \text{for } k = 0, \ldots, N \,, \quad (11)$$

where $\mathcal{U} \subset \mathbb{R}^m$ is the control input constraint set (kinematic and dynamic constraints), $\mathcal{C}_{free} \subset \mathbb{R}^n$ is the state constraint set, and J is the objective function given by

$$J(\{s_k\}_0^N, \{u_k\}_0^{N-1}) := F(s_N) + \sum_{k=0}^{N-1} L(s_k, u_k) \,.$$

The functions $F : \mathbb{R}^n \to \mathbb{R}$ and $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ are the *terminal state cost* and the *stage cost*, respectively. Typical choices for the weighting functions F and L are quadratic functions. We select the objective function J to be

$$J(\{s_k\}_0^N, \{u_k\}_0^{N-1}) := \sum_{k=0}^{N} \phi(x_k, y_k, \theta_k) + \rho \sum_{k=0}^{N-1} |u_k| \,,$$

where ϕ is the interpolated path cost function defined in Sec. IV-A, and ρ is a small positive scalar value. All sequences $\{u_k\}_0^{N-1}$ and $\{s_k\}_0^N$ satisfying the constraints (8)-(11) are called *feasible sequences*.

We denote the optimal control sequence at $s(t)$ by $\{u_k^*\}_0^{N-1}$. Then, the control applied to the system at time t is the first element of this sequence, defining the control law κ as

$$u(t) = \kappa(s) = u_0^* \,. \quad (12)$$

The same procedure (7)-(12) is repeated for the new state of the system at each time interval.

In order to prove the asymptotic stability of the closed loop system, the value function J^* is used as a *Lyapunov function*.

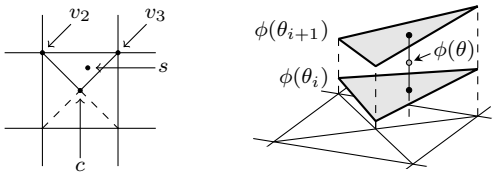

Fig. 3. Path costs are interpolated separately for two discretized orientations. A final interpolation is run between the previously interpolated costs.

According to the Lyapunov asymptotic stability analysis, the aim is to find an objective function J and a control sequence that minimizes J such that J^* decreases each time step. We introduce additional constraints on the states necessary for the stability proof, which can be interpreted as preferring those trajectories that are approaching the goal state faster:

$$\phi(x_N, y_N, \theta_N) \leq \phi(x_k, y_k, \theta_k), \text{for } k = 0, \ldots, N-1\,. \quad (13)$$

In order to make the optimization process tractable, we search only among a finite set of control sequences, denoted by $\mathcal{F}$, that satisfy the constraints (8)-(11) and (13). Similar to the dynamic window approach [2], we define each control sequence from the set $\mathcal{F}$ to be a collision free circular trajectory. In contrast to the dynamic window approach, our control sequences have zero velocities at the horizon. As stated in [4], this is a frequently used method to establish stability of the RHC implementation and can be seen as accounting for events that lie beyond the horizon.

The same principle is used by the convergent dynamic window algorithm (CDW) [13], where the horizon N is divided into two parts, $N = N_1 + N_2$, and every feasible control sequence must contain decelerations in the second part N_2 of the horizon. Therefore, every control sequence will feature a stopped robot at the horizon.

A. Path Cost Interpolation

To ensure convergent motion to the goal state the interpolated cost function ϕ must not have any local minima except for one global minimum located at the goal state. For discrete path cost values, this is fulfilled since only the goal node has null cost and there exists a path from any node to the goal with monotonically decreasing cost values. The interpolating cost function ϕ can preserve this property if it has continuous transitions between cells and discretized orientations. This is guaranteed by using the simplicial complex representation [15].

The interpolation is done as follows. Let $s = (x, y, \theta)$ be a continuous state contained in a cell with center $c = (x_1, y_1)$ and θ from the orientation bin defined by $[\theta_i, \theta_{i+1})$. This cell is divided into four triangles defined by the two diagonals of the cell as shown in the left image of Fig. 3. The state s is located in one of these four triangles. Let the two remaining vertices be $v_2 = (x_2, y_2)$ and $v_3 = (x_3, y_3)$. We interpolate the costs separately for two discretized orientations, and then again interpolate between the obtained costs (see right image of Fig. 3). We calculate the cost of the first orientation $\phi(x, y, \theta_i)$ as the interpolation of the costs at the triangle vertices where the cost $\phi(c, \theta_i)$ is the cost of the cell center at the orientation θ_i determined by the D* search. The costs

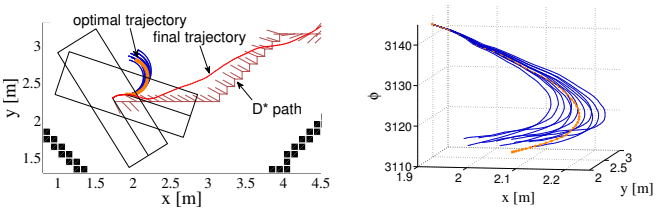

Fig. 4. An example of trajectory optimization at time t according to ϕ. Although trajectories seem to move the robot away from the goal (in blue, left), their costs are decreasing (right). The initial planned path with orientations and driven final trajectory (in red) are shown only as a reference.

of the cell vertices $\phi(v_2, \theta_i)$ and $\phi(v_3, \theta_i)$ are calculated as the minimum of the costs of the surrounding cell centers for the orientation θ_i plus the distance between the vertex and the cell center. The same is done for the orientation θ_{i+1} and the cost $\phi(x, y, \theta_{i+1})$. Finally, we obtain the cost $\phi(x, y, \theta)$ as the interpolation between $\phi(x, y, \theta_i)$ and $\phi(x, y, \theta_{i+1})$.

Our interpolation method for calculating the function ϕ at discrete orientations can be compared to the interpolated navigation function presented in the CDW algorithm [13], which calculates vertex costs using the wavefront algorithm. The interpolation is done over the triangles obtained by dividing each cell by a diagonal line drawn from the vertex with the highest cost value. Thus, the obtained navigation function has gradients only in the direction of the cell diagonals, while our cost function ϕ has quadratically more gradients due to a finer division of the cell. Another difference to the CDW algorithm is that we do not need to calculate the entire function ϕ but only its values at points on the robot trajectories. An example of evaluating the trajectories at time t is shown in Fig. 4, where 500 circular trajectories of varying lengths are being used for the horizon of length 50 steps.

B. Asymptotic Stability Proof

We follow the stability definitions given in [1] for the discontinuous discrete-time nonlinear closed loop system. Here, we shortly present how the stability of the RHC is established. A formal proof can be derived following [4].

Lemma 4.1: Let the set $\mathcal{S}$ be defined as $\mathcal{S} = \{s = (x \;\; y \;\; \theta) \in \mathcal{C}_{free} \mid \phi(x, y, \theta) < \infty\}$. Then $\forall s \in \mathcal{S}$ there exist feasible state and control sequences for the fixed horizon optimal control problem $P_N(s)$ in (7)-(12) and $\mathcal{S}$ is positively invariant for the closed loop system $s^+ = f(s, \kappa(s))$.

Proof: The control sequences are chosen from the set of control sequences $\mathcal{F}$ that satisfy constraints (8)-(11) and (13). First, we need to show that for any $s \in \mathcal{S}$ the set $\mathcal{F}$ is not empty. By construction, the set $\mathcal{F}$ is composed of collision free control sequences, and in the worst case only the zero-sequence exists, i.e., $u_k = 0$, for $k = 0, \ldots, N-1$. We denote the optimal control sequence and corresponding state sequence at time t for the state $s = s(t) \in \mathcal{S}$ as

$$U^* = \{u_0^*, \ldots, u_{N-1}^*\},\; S^* = \{s_0^*, \ldots, s_N^*\}\,. \quad (14)$$

Then, the control move applied to the system at time t is the first element of (14):

$$u(t) = \kappa(s) = u_0^*\,. \quad (15)$$

TABLE I

DISCRETE PATH AND FINAL TRAJECTORY (SIMULATION)

	rectangular		T-shape		fly-shape	
	$\mathcal{G}_{OI}$	$\mathcal{G}_{3D}$	$\mathcal{G}_{OI}$	$\mathcal{G}_{3D}$	$\mathcal{G}_{OI}$	$\mathcal{G}_{3D}$
$c(\mathcal{P})$	3,002	3,002	3,464	3,462	80,522	80,518
$c(\mathcal{T})$	3,001.6	3,001.6	3,402	3,400	80,520	80,516
$l(\mathcal{P})$ [m]	7.5	7.5	7.4	7.4	124	124
$l(\mathcal{T})$ [m]	5.903	5.903	6.285	6.285	102.71	102.9
v_{avg} [m/s]	0.294	0.294	0.259	0.259	0.387	0.388
t_G [s]	20.1	20.1	24.3	24.3	265.3	265.4
t_{src} [s]	0.003	0.063	0.007	0.278	0.078	14.594
n_{exp}[#]	1,640	28,566	6,207	136,689	97,866	5,910,471
t_{rhc}[s]	0.008	0.004	0.008	0.004	0.013	0.004

Assume that at time $t+1$ the robot moves to the state s_1^*, i.e., $s^+ = f(s,\kappa(s)) = f(s_0^*, u_0^*) = s_1^*$. There exists a control sequence $\{u_k\}_0^{N-1} \in \mathcal{F}$ with starting control move $u_0 = u_1^*$ that will produce the state trajectory $\{s_k\}_0^N$ starting from the s_1^* and repeating the last point at s_N^*:

$$U^+ = \{u_1^*, \ldots, u_{N-1}^*, 0\},\ S^+ = \{s_1^*, \ldots, s_N^*, s_N^*\}\,. \tag{16}$$

Therefore, the control sequence and the corresponding state sequence (16) are feasible for the problem $P_N(s^+)$ in (7)-(12) and hence $s^+ \in \mathcal{S}$. By this it is shown that $\mathcal{S}$ is positively invariant for the closed loop system $s^+ = f(s,\kappa(s))$. ■

Theorem 4.2: For the system $s^+ = f(s,\kappa(s))$ controlled by the RHC algorithm (7)–(12) the goal state is asymptotically stable in $\mathcal{S}$ for the closed loop system.

Proof: The value function J^* is employed as a Lyapunov function. We must check if J^* always decreases except at the goal state. At time t for the state $s \in \mathcal{S}$, and for the optimal control and state sequence given by (14) the value of the Lyapunov function is

$$J^*(s) = \sum_{k=0}^{N} \phi(s_k^*) + \rho \sum_{k=0}^{N-1} |u^*|\,. \tag{17}$$

At time $t+1$ for the next state $s^+ = f(s,\kappa(s))$ a feasible control and state sequences given by (16) are not necessarily optimal so we know that

$$J^*(s^+) \leq \sum_{k=1}^{N} \phi(s_k^*) + s_N^* + \rho \sum_{k=1}^{N-1} |u_k^*| + 0\,. \tag{18}$$

Combining (17) and (18) and substituting (13) yields:

$$J^*(s^+) - J^*(s) \leq -\rho|u_0^*|\,, \tag{19}$$

with equality only when $u_0^* = 0$. Therefore, J^* decreases for all $|u_0^*| > 0$. Finally, it must be shown that the robot will not stop at a non-goal state, i.e., $\kappa(s) \neq 0$, $s \neq G$. This is ensured by the construction of the continuous cost function ϕ with only one local minimum at the goal state G. Additionally, due to the construction of the set $\mathcal{F}$ there exists a small movement which will orient the robot to the desired orientation of the cell, and translate it to the next cell with the lowest path cost. Since we are searching the 4-connected grid, difficult controls to diagonal neighbor cells are avoided. ■

TABLE II

PATH COSTS IN $\mathcal{G}_{OI}$ IN SIMULATED ENVIRONMENTS

	cost ratio λ_r/λ_t	nodes with cost diff. from $\mathcal{G}_{3D}$ [%]	max cost diff. to $\lambda_r M$ [%]
rectangular $M = 60$	10	96.4	34.7
	1	1.2	3.3
	0.1	0	0
	0	0	0
T-shape $M = 52$	10	84.8	63.5
	1	13	26.9
	0.1	0.04	3.8
	0	0	0
fly-shape $M = 82$	10	99.7	137
	1	1	7.3
	0.1	0.0006	2.4
	0	0	0

V. EXPERIMENTS

1) Simulation Experiments: We evaluated the performance of motion generation and execution for various robot shapes by comparing the search on our orientation interval graph $\mathcal{G}_{OI}$ and on a graph containing all discrete configurations $\mathcal{G}_{3D}$, which is the standard approach to this problem. In $\mathcal{G}_{OI}$, the weights are determined such that the transition costs between two locations depend on the size of the intersection interval plus the rotation costs of their desired orientation difference, as given by (3). Accordingly, we define the weights in $\mathcal{G}_{3D}$ as

$$w_{a,b} := \lambda_t(M+1-|\Theta_{a'} \cap \Theta_{b'}|+|\hat{\theta}_{a'}-\theta_a|)+\lambda_r|\theta_a-\theta_b|\,, \tag{20}$$

where a' and b' are the nodes in $\mathcal{G}_{OI}$ containing a and b respectively. According to (20), the cost of transition between two nodes with orientation in the middle of the intersection interval is the smallest and has the same value as defined in $\mathcal{G}_{OI}$. If the intersection interval contains all orientations then it is assumed that $\hat{\theta}_{a'} = \theta_a$. The paths in $\mathcal{G}_{3D}$ are compared to the ones in $\mathcal{G}_{OI}$ in Tab. I. Three different robot shapes have been chosen in simulated static environments of different sizes, see Fig. 5. The paths have been executed by the proposed motion planning algorithm and the obtained trajectories $\mathcal{T}$ are compared according to the length $l(\mathcal{T})$ and cost $c(\mathcal{T})$ calculated from the interpolated cost function ϕ (see Sec. IV-A) as $c(\mathcal{T}) = \sum_i(\phi(\mathcal{T}_i)-\phi(\mathcal{T}_{i-1}))$. Note that this formula is valid also for calculating the cost of the path. Path cost $c(\mathcal{P})$ and trajectory cost $c(\mathcal{T})$ are given in cell numbers (cell size is 0.1 m) with $\lambda_t = 1$ and $\lambda_r = 1$. In the given simulation examples the obtained paths in $\mathcal{G}_{OI}$ have comparable values for all setups. The executed trajectories, average velocities v_{avg} and times to goal t_G are similar for both approaches (maximal velocities were set to 0.4 m/s and 50 deg/s). The costs of trajectories are lower than the costs of the discrete paths for all robots. This is expected since the generated local trajectories are defined in continuous space and give the robot the possibility to cut its way to the goal. $\mathcal{G}_{OI}$ outperforms $\mathcal{G}_{3D}$ with respect to the search time t_{src}, since the number of explored nodes n_{exp} is substantially lower, up to $60\times$ for the largest environment. In Tab. II, we analyze the influence of the ratio λ_r/λ_t on the path costs. A

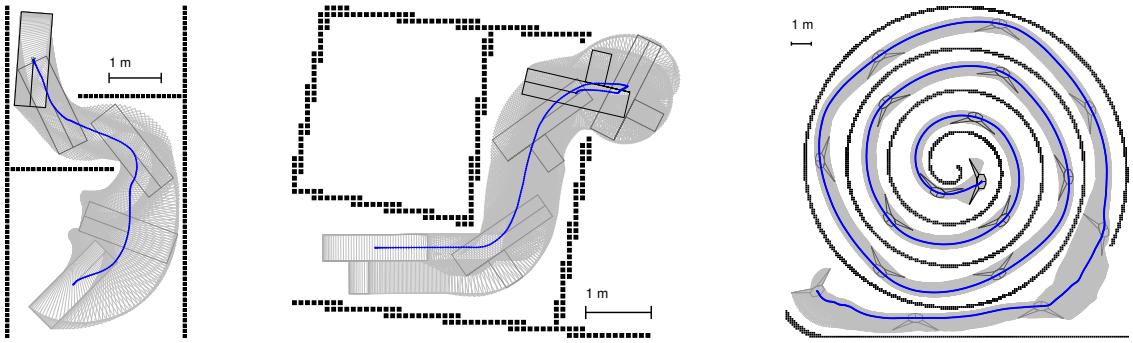

Fig. 5. Robot trajectories in a simulated environment. Note that robot motion includes complex maneuvers due to the footprint of the robot.

TABLE III
DISCRETE PATH, FINAL TRAJECTORY, AND REPLANNING

	goal A	goal B
$c(\mathcal{P})$	3,774	5,967
$c(\mathcal{T})$	3,774.6	5,952.7
$l(\mathcal{P})$ [m]	6.8	6.4
$l(\mathcal{T})$ [m]	5.38	6.86
v_{avg} [m/s]	0.294	0.236
t_G	20.2	29.4
t_{src} avg (max) [s]	0.00024 (0.012)	0.002 (0.029)
n_{exp} avg (max) [#]	60 (11087)	659 (18067)
t_{rhc} [s]	0.01	0.009

higher ratio means weighting rotational costs more strongly than translational costs. For each goal we compare the path cost value for every state (x, y, θ) obtained by searching $\mathcal{G}_{OI}$ and $\mathcal{G}_{3D}$. The path costs between approaches are equal for all states if $\lambda_r = 0$. In the extreme case of $\lambda_r/\lambda_t = 10$, the costs obtained in $\mathcal{G}_{OI}$ differs most from the ones obtained in $\mathcal{G}_{3D}$. The higher the ratio, the less good are the paths returned by our search strategy.

2) Real Robot Experiments: We tested our motion planning algorithm on the holonomic mobile robot KUKA omniRob. As the footprint of the robot is rectangular, we fixed a rectangular payload on top of the robot to make its footprint more irregular, see Fig. 1. The planned paths and obtained trajectories have been compared according to their costs in a static environment. Replanning has been tested by walking in front of the robot and blocking its path to the goal. The experiments have been repeated five times and averaged results are given in Tab. III. The trajectory costs are sometimes higher than the initial path costs, which is expected in dynamic environments. The replanning times are presented with their average and maximal values. The velocity profile during one experimental run between the two goals is given in Fig. 6.

VI. CONCLUSION

This paper presented a novel anyshape robot navigation approach that is based on a combination of discrete-continuous techniques. Our approach uses a compact graph representation of the collision-free configuration space that allows fast replanning. It applies a continuous motion generation for computing smooth motions in a highly reactive manner that we prove to be Lyapunov asymptotically stable.

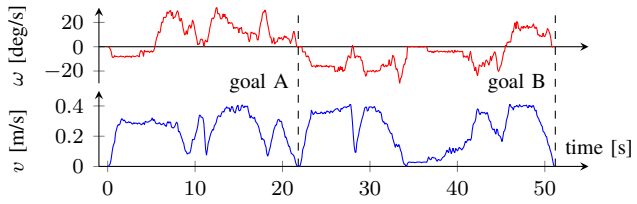

Fig. 6. Velocity profiles executed by a robot in a real world experiment.

By using this discrete-continuous strategy, both planning and motion generation run online, enabling applications in dynamically changing environments. Experiments demonstrate that our method provides fast and reactive path planning and smooth motion execution.

REFERENCES

[1] F. Christophersen. *Optimal control of constrained piecewise affine systems*. Springer Verlag, 2007.
[2] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Auton. Sys.*, 4(1):23–33, 1997.
[3] R. Geraerts and M. Overmars. Creating high-quality paths for motion planning. *Int. Journ. of Rob. Research*, 26(8):845–863, 2007.
[4] G. C. Goodwin, M. M. Seron, and J. A. De Doná. *Constrained control and estimation: an optimisation approach*. Springer Verlag, 2004.
[5] J. Kim, R. Pearce, and N. Amato. Extracting optimal paths from roadmaps for motion planning. In *IEEE Int. Conf. on Rob. & Aut.*, 2003.
[6] J. King and M. Likhachev. Efficient cost computation in cost map planning for non-circular robots. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys.*, 2009.
[7] J. Latombe. *Robot Motion Planning*. Kluwer Academic Pub., 1991.
[8] B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics & Auton. Sys.*, 2013. to appear.
[9] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *Int. Journ. of Rob. Research*, 20(5):378–400, 2001.
[10] S. Lindemann and S. LaValle. Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. *Int. Journ. of Rob. Research*, 28(5):600–621, 2009.
[11] B. MacAllister, J. Butzke, A. Kushleyev, and M. Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *IEEE Int. Conf. on Rob. & Aut.*, 2013.
[12] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36, 2000.
[13] P. Ogren and N. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Trans. on Rob.*, 21(2):188–195, 2005.
[14] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. In *IEEE Int. Conf. on Rob. & Aut.*, 2011.
[15] J. Sack and J. Urrutia. *Handbook of computational geometry*. North-Holland, 2000.
[16] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE Int. Conf. on Rob. & Aut.*, 1994.
[17] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-d path networks in unstructured environments. In *IEEE Int. Conf. on Rob. & Aut.*, 2005.