Incremental Updates of Configuration Space Representations for Non-Circular Mobile Robots with 2D, 2.5D, or 3D Obstacle Models

Boris Lau Christoph Sprunk Wolfram Burgard

***Abstract*— This paper presents techniques to incrementally update collision maps, distance maps, and Voronoi diagrams in the configuration space of non-circular mobile robots. Compared to previous work, our approach only updates the cells affected by changes in the environment. Thus, it is applicable in large workspaces and in the presence of unknown or moving obstacles. The c-space collision maps allow for performing highly efficient collision checks in dynamically changing environments, for 2D, 2.5D, and 3D representations of robots and obstacles. By using the proposed c-space distance maps, long trajectories can efficiently be checked for collisions. Finally, our c-space Voronoi diagrams allow effective and complete path planning in narrow spaces. Our algorithms are easy to implement, benefit from parallelization on multi-core CPUs, and can be integrated with state-of-the-art path planners. Experiments demonstrate the effectiveness of our methods and show their applicability to real-world scenarios.**

I. Introduction

Efficient collision checks are a crucial ability for many online systems in autonomous mobile robotics: simulators, path planners, and trajectory optimizers alike need to check for every considered pose for collision. For plain 2D obstacle representations and circular approximations of the robot's footprint, these collision checks are easy to perform.

For non-circular robots in passages narrower than their circumcircle, however, circularity is too crude an assumption, and collisions have to be checked for in the three-dimensional configuration space (c-space) of robot poses. Also, even for robots moving on a plane as considered in this paper, 3D obstacles and collisions can be important: applications like robotic transporters, wheelchairs, or mobile manipulators can require the robot to partially move underneath or above obstacles as shown in Fig. 1. In these cases, collision checks can easily become a dominant part of the computational effort.

Naive collision checks on 2D occupancy grid maps require one lookup per grid cell in the area occupied by the robot. For 3D obstacle representations like multi-layer surface maps or full 3D maps [1], the effort depends on the robot's volume. However, by convolving a map with the discretized shape of the robot, one can precompute a collision map that marks all colliding poses. With such a map, a collision check requires just a single lookup, even for 3D obstacle representations.

From these c-space maps one can derive c-space distance maps that encode the distance to the closest colliding robot pose. Based on this, checking a trajectory for collisions can be made more efficient by extending the intervals of collision checks from one cell to the respective local collision distance. Similarly, in contrast to regular 2D Voronoi maps that can be used for complete motion planning of circular robots, c-space Voronoi maps can be employed for non-circular robots as well.

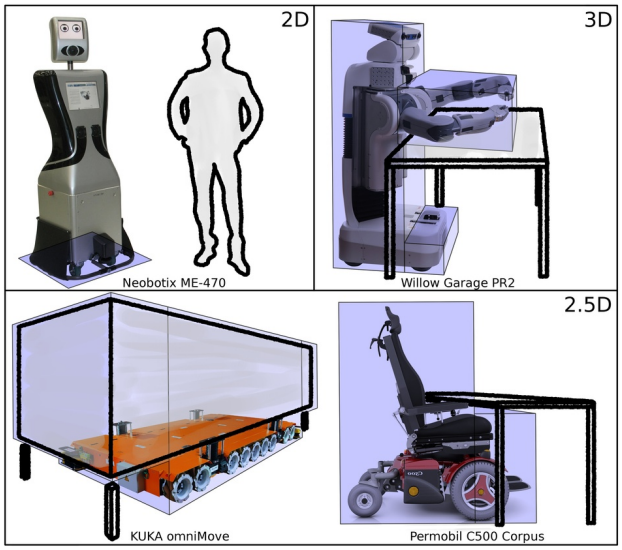

Fig. 1. For some applications, representing obstacles and robots by their 2D footprints can be sufficient (top-left). For overhanging parts of robots, their load, or obstacles, 2.5D representations are needed (bottom), whereas interaction tasks can also require actual 3D obstacle and robot models (top-right). Robot discretizations as used in our experiments are depicted in blue.

Early work on c-space obstacle representations assumes static environments [2]. More recent approaches are often motivated by dynamic obstacles, but mostly try to improve the efficiency with the goal to execute the entire computation on every sensor update. This paper presents methods to incrementally update collision, distance, and Voronoi grid maps in the configuration space of non-circular mobile robot poses. Our algorithms only update cells affected by changes, which greatly reduces the computational costs. We consider all obstacle types shown in Fig. 1 and thus provide means for efficient collision checking in real-world applications. The algorithms are easy to implement and benefit from parallelization on multiple cores.

After discussing related work, we describe dynamic c-space collision, distance, and Voronoi maps in Sects. III and IV, and their application to path planning in Sect. V. Finally, we present extensive experiments in Sect. VI.

II. Related Work

Algorithms for efficient collision checking in 3D continue to be an active area of research. For example, Tang *et al.* [3] recently proposed a connection collision query algorithm that

All authors are with the Department of Computer Science at the University of Freiburg, Germany, {lau, sprunkc, burgard}@informatik.uni-freiburg.de.

This work has partly been supported by the European Commission under FP7-248258-First-MM, FP7-248873-RADHAR, and FP7-260026-TAPAS.

detects collisions of triangle meshes moving between given states. Thus, it can be used for sampling-based path planning. For online feasibility, Pan *et al.* use multi-core GPUs for collision queries [4]. Still, the cost per collision check depends on the number of polygons used to represent the tested objects.

Schlegel precomputes collision distances for circular arcs as a function of relative obstacle location and curvature [5]. Thus, the kinematic analysis is done offline, and collision distances can be obtained with one lookup per obstacle. Precomputing c-space representations instead further reduces the online effort for collision checks to a single lookup. Since Lozano-Perez's 1983 paper on c-space planning among static polyhedric obstacles, many approaches were proposed to reduce the cost for computing c-space obstacles [2]. Because of the relevance of this problem, researchers still work on improving the efficiency [6].

Convolving a gridmap of a robot's environment with an image of its footprint yields a discrete c-space map. To always reflect the current state of previously unknown or moving obstacles, these maps need to be updated regularly. Kavraki proposed to use the fast Fourier transform (FFT) to reduce the computational cost of the convolution [7], and Therón *et al.* added parallelization as a further speed-up [8]. Later, the same group proposed a multi-resolution approach to reduce memory and computational load in large workspaces [9]. To speed up path planning for autonomous cars, Ziegler and Stiller decompose the shape of the robot into circular discs [10].

As a first dynamic approach for changing enviroments, Wu *et al.* precompute colliding robot poses for each potentially occupied cell in the work space of a manipulator [11]. Taking the union of the colliding poses for a given set of occupied cells yields the c-space obstacle map without further recomputation. For mobile robots, however, the size of the operational area can make the database storage and the online computation of the union infeasible. In contrast, our method for updating c-space collision maps is truly incremental: it executes a regular map convolution in an offline phase, and during online application only updates the cells affected by changes in the environment.

In our previous work we presented algorithms to incrementally update two-dimensional Euclidean distance maps and Voronoi diagrams [12]. For path planning with circular robots, 2D Voronoi diagrams are appealing roadmaps since they cover all topologically different paths in a map with a small number of cells. For rectangular robots however, 2D Voronoi planning looses its completeness property, and one has to repair paths in narrow areas where following the Voronoi diagram leads to collisions, e.g., by using RRTs as proposed by Foskey *et al.* [13]. In this paper we combine the dynamic distance and Voronoi maps proposed in [12] with our novel incrementally updatable c-space collision maps. In this way, we overcome the aforementioned problem and can perform complete Voronoi planning in the configuration space of non-circular robots.

Although we use A* planning as an example application, our approach can be combined with other planners, e.g., D* Lite [14], or rapidly-exploring random trees (RRT) with Voronoi-biased sampling [15].

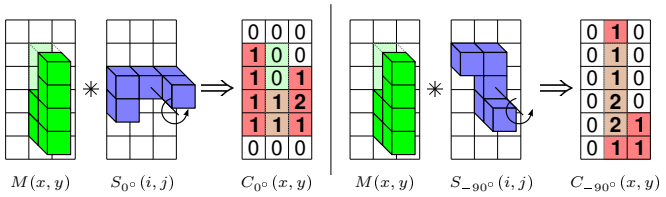

Fig. 2. Convolving a map $M(x,y)$ with a representation of the robot's shape $S_\theta(i,j)$ for a given orientation θ yields a collision map $C_\theta(x,y)$, according to Eq. (1). Each cell $\langle x,y\rangle$ in C_θ counts the cells in the robot footprint that collide with occupied cells in M, given the robot is at pose $\langle x,y,\theta\rangle$.

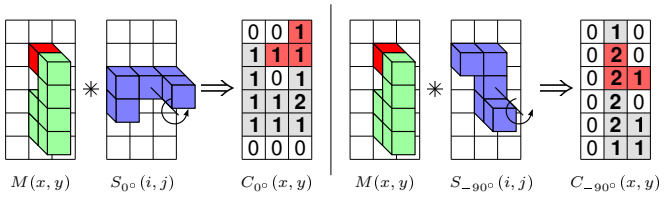

Fig. 3. A newly occupied cell in the map M (red) increments the collision count C for all robot poses that cause a collision at the location of the new obstacle. In this way, the collision map is updated (red cells) without recomputing the values for unaffected (gray) cells. Alg. 1 implements this procedure as well as the corresponding case for newly emptied cells.

III. Dynamic C-Space Collision Maps

This section presents a method to incrementally update c-space collision maps for non-circular mobile robots moving on a plane. It can be applied to all obstacle types shown in Fig. 1. For the sake of clarity we start with the 2.5D representation with overhanging obstacles shown in Fig. 1 (bottom-right), and discuss the adaptation to the other obstacle models later.

Let $M(x,y)$ be a grid map that represents the vertical clearance, i.e., the height of free space above the floor, with zeros for completely occupied cells. Consider a robot moving on the floor with continuous orientation $\tilde{\theta}$ with respect to the map coordinate system. We represent the discretized shape of the robot for a given orientation $\tilde{\theta}$ by a map $S_{\tilde{\theta}}(i,j)$, that stores the height of the robot for every cell of its footprint. $S_{\tilde{\theta}}$ has the same resolution and orientation as the grid map M, whereas its origin $S(0,0)$ is located at the robot's center.

A convolution-type conjunction of M and $S_{\tilde{\theta}}$ yields a count map $C_{\tilde{\theta}}(x,y)$ as shown in Fig. 2. Each cell $\langle x,y\rangle$ in $C_{\tilde{\theta}}$ stores the number of cells the robot collides with when located there:

$$C_{\tilde{\theta}}(x,y) = \sum_i \sum_j \text{eval}\big\{M(x+i,y+j) \leqslant S_{\tilde{\theta}}(i,j)\big\}\ , \quad (1)$$

where $\text{eval}(\text{true}) = 1$ and $\text{eval}(\text{false}) = 0$. If we discretize $\tilde{\theta}$ and stack the $C_\theta(x,y)$ for all discrete θ, we obtain the robot's c-space map $C(x,y,\theta)$ for M. Clearly, by testing $C(x,y,\theta) > 0$ we can check if the discretized pose $\langle x,y,\theta\rangle$ is colliding.

A. Discretization of Orientations

An appropriate discretization of $\tilde{\theta}$ ensures that if two adjacent poses $\langle x,y,\theta_i\rangle$ and $\langle x,y,\theta_{i+1}\rangle$ are collision-free according to C, intermediate orientations $\hat{\theta} \in [\theta_i, \theta_{i+1}]$ are collision-free as well. Under this constraint we seek to discretize $\tilde{\theta}$ as coarse as possible to keep the number of θ-layers in C small.

In occupancy grid maps, the actual location of obstacles can be anywhere in the cells they occupy. Therefore, one usually

Alg. 1 Dynamic Update of C-space Collision Map

function UpdateVerticalClearance(x, y, v_{new})

1: $v_{\text{old}} \leftarrow M(x, y)$
2: $M(x, y) \leftarrow v_{\text{new}}$
3: **for all** θ **do**
4: **for all** $\langle x', y' \rangle \in \{\langle x-i, y-j \rangle \mid S_\theta(i, j) > 0\}$ **do**
5: **if** $v_{\text{new}} \leqslant S_\theta(i, j) \wedge v_{\text{old}} > S_\theta(i, j)$ **then**
6: $C(x', y', \theta) \leftarrow C(x', y', \theta) + 1$
7: **if** $C(x', y', \theta) = 1$ **then** NewOccupied(x', y', θ)
8: **else if** $v_{\text{new}} > S_\theta(i, j) \wedge v_{\text{old}} \leqslant S_\theta(i, j)$ **then**
9: $C(x', y', \theta) \leftarrow C(x', y', \theta) - 1$
10: **if** $C(x', y', \theta) = 0$ **then** NewEmpty(x', y', θ)

assumes an additional safety margin m around the robot, e.g., of $m = 1$ pixel unit. With this, we can formulate a bound on the angular resolution for the discretization of $\tilde{\theta}$ as follows: if the robot rotates from θ_i to θ_{i+1}, each point on the robot moves along an arc. The maximum arc length occurs at the outmost point of the robot, which is the radius r (in pixels) of the circumcircle around its center. By choosing a resolution of $|\theta_i - \theta_{i+1}| = m/r$, we ensure that even in the worst case an obstacle collides only with the safety margin but not with the actual robot. Depending on the shape of the robot, less conservative bounds on the discretization can be formulated.

B. Incremental Update of the C-space Map

Unknown or moving obstacles cause changes in the environmental representation of a robot. For the 2.5D obstacle model, a change is given by an updated vertical clearance v_{new} for a cell $\langle x, y \rangle$ in M. To refresh C incrementally rather than computing it from scratch, we only update the affected parts of the sum in Eq. (1) according to Alg. 1. See Fig. 3 in comparison to Fig. 2 for an illustration.

The algorithm separately updates the θ-layers of C, and can thus be parallelized (line 3). For each cell $\langle i, j \rangle$ of the robot shape $S_\theta(i, j)$ we visit the robot position $\langle x', y' \rangle$ that lets $\langle i, j \rangle$ fall on $\langle x, y \rangle$ (line 4). These cells can be efficiently selected using standard drawing algorithms for rasterized images.

If the new vertical clearance v_{new} in $\langle x, y \rangle$ causes a collision with $S_\theta(i, j)$ while v_{old} did not, the collision counter of $\langle x', y' \rangle$ is incremented (line 5), since this represents a new collision candidate cell. Vice versa, if v_{new} is collision-free and v_{old} collides, the counter is decremented (line 8), since a collision candidate was removed. Whenever the count changes from 0 to 1 or from 1 to 0, the pose $\langle x', y', \theta \rangle$ is newly occupied (line 7) or emptied (line 10), respectively. These events can be used to trigger further computation, e.g., to update the c-space distance map and Voronoi diagram discussed in Sect. IV.

C. Adaptation to Other Kinds of Obstacles and Robots

Until here, we assumed overhanging obstacles and a robot on the floor that can move underneath obstacles as in Fig. 1 (bottom-right). By reversing the comparisons of robot height and vertical clearance in Eq. (1) and Alg. 1 (lines 5 and 8), this can easily be adapted to obstacles elevating from the floor and robots with overhanging load or parts as in Fig. 1 (bottom-left). For plain 2D robot and obstacle models, the heights v_{new} and v_{old} are binary values that encode occupied and free. The other techniques presented in this paper apply without modifications.

For some applications, the obstacles and the robot have to be represented in full 3D as in Fig. 1 (top-right). The height comparisons in Alg. 1, lines 5 and 8, then have to consider lists of obstacle heights. If the robot shape is approximated by a set of vertical columns with a given upper and lower end as in Fig. 1, one can also use a separate shape map for each column. By only considering the affected columns in line 4, the c-space collision map can be efficiently updated.

Robots with a symmetric shape with respect to their center cause a part of the θ-layers in $C(x, y, \theta)$ to be redundant. For example, a rectangular robot at orientation $180°$ causes the same c-space obstacles as at $0°$. Omitting the respective layers when iterating over θ in Alg. 1 (line 3) saves a substantial part of the computational effort and memory consumption.

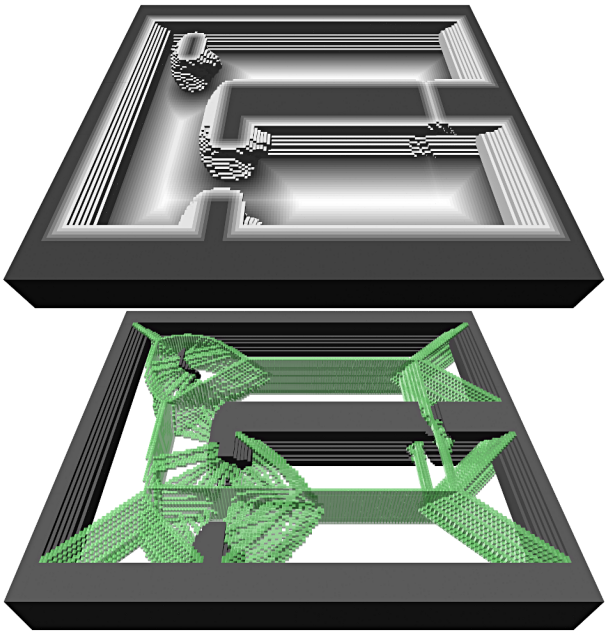

Fig. 4. C-space distance map (top) and Voronoi diagram (bottom) for a rectangular robot, both obtained by stacking layers computed in 2D for different robot orientations θ. For readability, only half of the layers are shown, the other half is identical due to the robot's symmetry. In the visualization at the top, cells above the bottom layer have a different color scaling, and were removed when exceeding a distance threshold.

IV. C-space Distance Maps and Voronoi Diagrams

The cells of a distance map D measure the distance to their closest obstacle in the map M. Given a three-dimensional c-space collision map $C(x, y, \theta)$ as defined above, one could consider a 3D distance map that uses a 3D distance measure including the angle. As discussed by Canny [16], it is however more desirable to employ 2D Euclidean distances per θ-layer. Thus, we stack Euclidean distance maps $D_\theta(x, y)$ computed for every c-space map layer $C_\theta(x, y)$, yielding the c-space distance map $D(x, y, \theta)$ as shown in Fig. 4 (top).

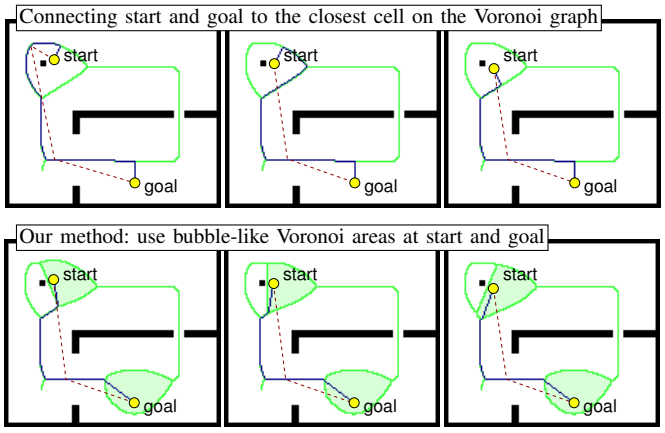

Fig. 5. Connecting start and goal to the Voronoi graph (green) during planning: using the shortest connection (top), the planned path (blue) can change abruptly for small changes of the start configuration, even for sightline-pruned paths (dashed). We create Voronoi bubbles around start and goal, and use goal-directed search therein, which yields more stable paths (bottom).

In 2D, Voronoi graphs are the union of points whose two closest obstacles are at the same distance. Similar to distance maps, we compute a Voronoi diagram $V_\theta(x, y)$ for every $C_\theta(x, y)$. Stacking these Voronoi diagrams results in the c-space Voronoi diagram $V(x, y, \theta)$ as shown in Fig. 4 (bottom). If θ is discretized according to Sect. III-A, Voronoi lines in neighboring layers connect to unbroken surfaces.

We update the layers of the c-space distance map D_θ and Voronoi diagram V_θ incrementally as proposed for 2D in our previous work [12]. The functions SetObstacle(x, y) and RemoveObstacle(x, y) described therein are used to specify newly occupied or freed cells. UpdateDistanceMap() performs the update with a dynamic brushfire algorithm. The event NewOccupied(x', y', θ) in Alg. 1 calls SetObstacle(x', y') for the dynamic distance map $D_\theta(x, y)$, and NewEmpty(x', y', θ) calls RemoveObstacle(x', y'). After the update of $C(x, y, \theta)$, we execute UpdateDistanceMap() in parallel for every D_θ, which completes the update of $D(x, y, \theta)$ and $V(x, y, \theta)$.

V. C-space Voronoi Path Planning

Given a c-space Voronoi map as described above, a goal-directed graph search on the Voronoi cells is no different from regular planning on 3D grids, except for the cyclic nature of the orientation dimension. This section details on important aspects of Voronoi planning in dynamic environments.

In general, the start and goal locations of a planning problem are not on the Voronoi graph. Trivial approaches search for the closest Voronoi cell at both locations, and connect them with straight lines to the graph [17]. This is problematic in practice, since a small change of the start pose can substantially change the planned path as shown in Fig. 5 (top row). Our approach tackles this problem by applying the following steps:

1) insert a virtual obstacle at the start and goal location,
2) update the Voronoi map for all layers,
3) use brushfire expansion to mark cells in Voronoi bubbles,
4) plan a path from start to goal,
5) undo the changes made by 1) and 2).

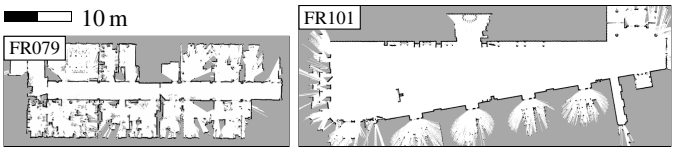

Fig. 6. Maps of the environments we used to benchmark collision checks.

Through the virtual obstacles inserted at start and goal, these locations get enclosed by the Voronoi graph which generates a "bubble"-like area as shown in Fig. 5 (bottom row). For all θ-layers in parallel, the brushfire expansion starts at the start and at the goal position, and marks all visited cells while expanding like wavefronts up to the enclosing Voronoi lines.

Then, a goal-driven A* search is initiated at the start location. It is restricted to only consider Voronoi cells and the ones in the start and goal bubble, marked by the brushfire expansion. In this way, the search expands from the start onto the Voronoi graph, follows Voronoi lines, and then connects to the goal when reaching the goal bubble. Since the whole path is the result of goal-directed graph search, the consecutive paths planned for a moving robot are very similar to each other and do not change abruptly (see Fig. 5).

The brushfire expansion has to be run for each θ-layer separately using a 4-connected neighborhood. This ensures that the expansion is contained in the start and goal bubbles. Since the update of the c-space representations is run separately for the θ layers as well, the whole procedure can be parallelized except for the actual planning routine in step 4).

VI. Experiments

This section presents application examples and benchmarks for our incrementally updatable c-space representations. We performed tests on two sequences of 2D laser range data with 200 frames each. The data was recorded by a robot moving in areas where walking people heavily affected the traversable space, namely an office building (FR079) and a large foyer (FR101). The algorithms were initialized with the gridmaps shown in Fig. 6, using a resolution of 0.05 m per grid cell. The maximum range of the laser scanner was limited to 5 m. To get 2.5D and 3D obstacles, we augmented the laser data with random height values between 0 m and the robot height.

In 2D, we assumed a medium sized rectangular robot (0.85x0.45 m) and a large one (1.75x0.85 m). In 2.5D, we modeled a wheelchair with a low front and a high rear part, as in Fig. 1 (bottom-right). In 3D, the robot was modeled like a Willow Garage PR2, with a frontal extension for the base and the fixed arms (see Fig. 1 top-right). The C++ implementation of our algorithms was executed on an Intel Core i7 2670 MHz, using OpenMP for parallelization with up to 6 threads.

A. Collision Checks for Non-Circular Robots

The c-space collision map presented in Sect. III requires computation of the incremental update in every time step, but then, each collision check for the whole robot takes only a single lookup. In the 2D model, we exploit the symmetry of the rectangular robot as described in Sect. III-C.

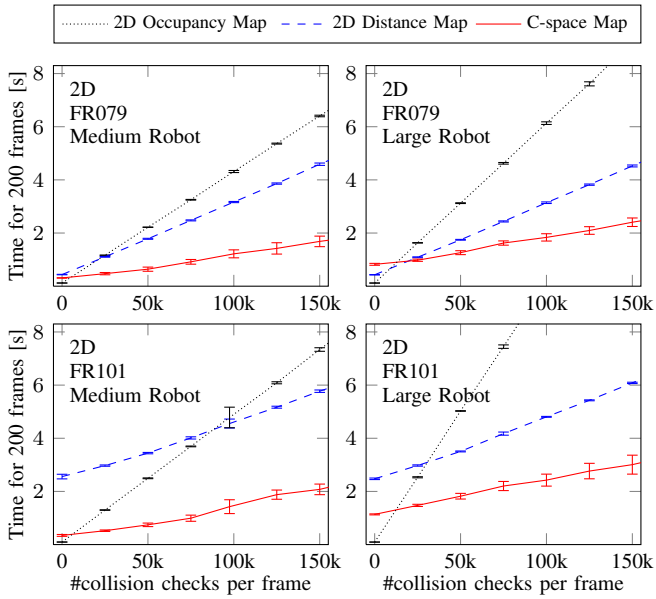

Fig. 7. Computation time for different collision checking routines for two sequences and two robot models. The update required in every frame for the c-space collision map pays off starting from 10,000 collision checks per frame. The plot shows mean and standard deviations for 10 runs.

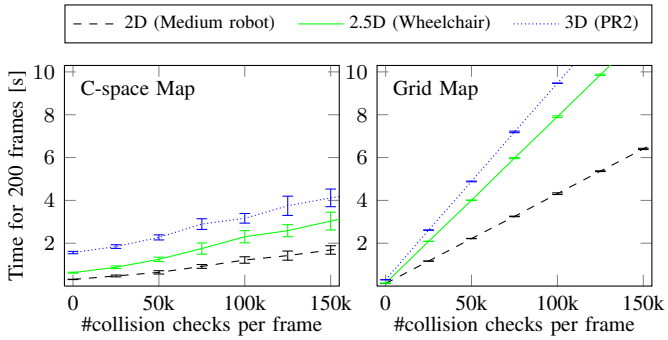

Fig. 8. Collision check performance for different robot and obstacle models, using our updatable c-space collision map (left) vs. the straight-forward occupancy gridmap approach (right). The costs for updating the c-space map are remedied by the faster collision checks for 10,000 checks or more per frame. The plot shows mean and standard deviations for 20 runs.

We compare our method to a previous collision checking approach for rectangular robots that uses incrementally updatable 2D distance maps [18]. As a baseline, we also include a straight-forward approach that checks every cell of the robot's footprint for collision using an up-to-date 2D occupancy map.

The results of this benchmark are shown in Fig. 7. The time required for updating the distance and c-space maps is shown by the first data point of each plot (zero collision checks). The slopes of the curves depend on the cost per collision check. In contrast to the distance map approach, the update time for the c-space map grows with the size of the robot (right vs. left column), but does not suffer from the open area in FR101 (bottom vs. top row). The update for the c-space collision map pays off for 10,000 or more collision checks, which can easily be required during path planning or trajectory optimization. In comparison, the break-even point for a single disc-shaped

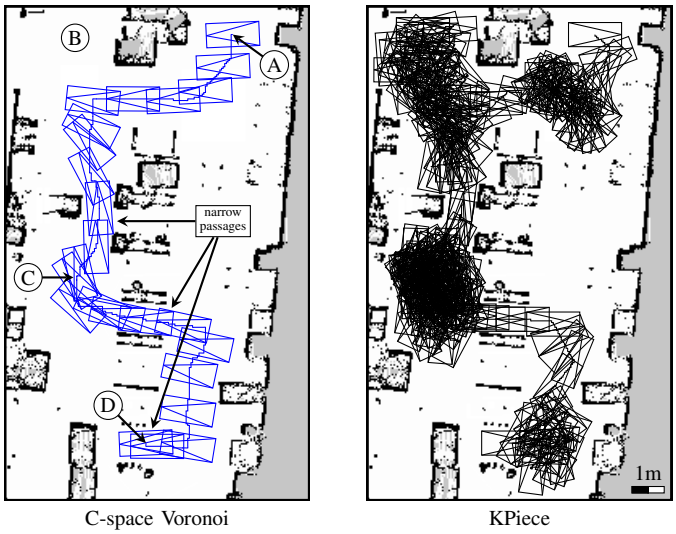

Fig. 9. Map of a factory floor (9.5x15.4 m) with start location (A) and three goals (B), (C), and (D). Example paths from (A) to (D) are shown for two different planners. The sampling-based planner (right) is challenged by narrow passages, while the performance of the Voronoi planner (left) is unaffected.

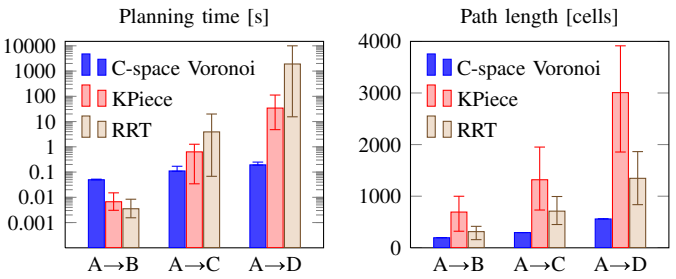

Fig. 10. Planning time and path length for three planners and the three planning tasks in Fig. 9. The plot shows mean and min/max for 20 runs. In contrast to the Voronoi planner, the sampling-based planners require several orders of magnitude more planning time for each narrow passage in the path.

object was at 22,400 for the disc-decomposition method by Ziegler *et al.*, and $5 \cdot 10^6$ for the full c-space [10].

We repeat the experiment, but with 2.5D and 3D obstacles and robots this time. Compared to the 2D rectangular robot (dashed), the costs for the c-space update with 2.5D and 3D are higher, since the robots are not symmetric anymore and consist of two and three parts, respectively, see Fig. 8 (left). However, the costs per collision check (slope of the plots) is the same, as opposed to the curves for the straight-forward occupancy grid map approach (right).

In all cases, the update of the c-space map takes less than 15 ms per frame. Performing 150,000 collision checks per frame additionally requires at most another 15 ms. This corresponds to $10 \cdot 10^6$ collision checks per second for arbitrary robot shapes, which clearly outperforms even modern GPU-based approaches with $0.5 \cdot 10^6$ collision checks per second for simple polygons [4].

B. Path Planning using C-space Voronoi Maps

The c-space Voronoi maps presented in this paper provide means for complete grid map planning for non-circular omnidirectional robots using standard graph search algorithms

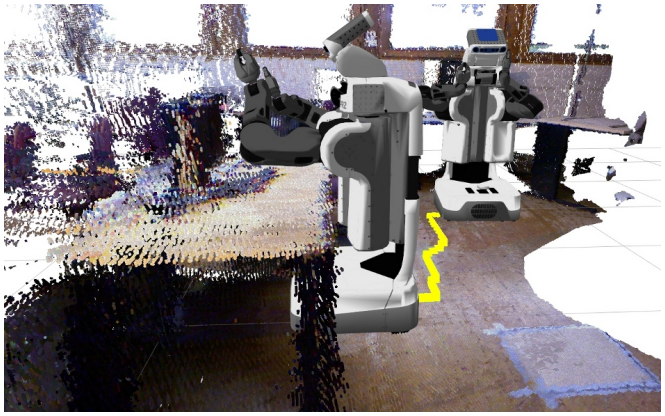

Fig. 11. Table-docking with a PR2 robot in a 3D map using Voronoi planning.

like A* or D* Lite [14]. With our algorithms for incremental updates they are applicable in dynamic environments. This experiment uses the Voronoi bubble technique proposed in Sect. V.

We use A* to plan paths for the large robot model (see above) on the grid map of the factory floor shown in Fig. 9. The start pose is given by (A), and three possible goal poses by (B), (C), and (D). Each of the consecutive goals requires traversing another narrow passage. For comparison, we test our method against the KPiece and RRT implementations available in the Open Motion Planning Library [19]. All planners use our c-space map for collision checking.

The average resulting planning times and path lengths for 20 runs per start-goal combination are shown in Fig. 10. Each additional narrow passage requires several orders of magnitude more planning time for the sampling-based planners, while the time taken by the Voronoi planner grows roughly linearly with the path length. Using per-cell collision checking rather than the c-space collision maps for the sampling based planners increases the computation times by a factor of 3.

As another application example, we plan the path of a PR2 robot using a c-space Voronoi map generated from real 3D point cloud data (see Fig. 11). After a precomputation phase of 0.5 s, planning a path on the incrementally updatable c-space Voronoi map takes less than 2.5 ms.

Clearly, Voronoi planning is of advantage in narrow areas as long as the grid resolution is fine enough. Our incrementally updatable c-space Voronoi representation allows to apply this idea to non-circular robots in dynamic environments, and could also be used in Voronoi sampling routines of other path planners [15].

VII. Conclusion

This paper presents incrementally updatable collision maps, distance maps, and Voronoi diagrams for non-circular robots. During initialization, these representations are computed using a given grid map or point cloud. During online application, our methods only update cells that are affected by changes in the environment, and can thus be used in real-world scenarios with unexpected or moving obstacles.

We consider different obstacle representations, namely a robot moving on a plane with overhanging obstacles, or vice versa, obstacles elevating from the ground, and a robot with overhanging parts. Our approach is also applicable to 2D and full 3D obstacle representations, and can exploit symmetries in the robot shape.

In our experiments we showed that the presented methods allow for a high number of collision checks and reliable path planning in frame rates required for online real-world applications. To further increase the efficiency of our algorithms, e.g., in higher-dimensional c-spaces of manipulators, one could combine them with hierarchical decomposition as proposed by Blanco *et al.* [9].

References

[1] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, 2010.

[2] K. D. Wise and A. Bowyer, "A survey of global configuration-space mapping techniques for a single robot in a static environment," *International Journal of Robotics Research*, vol. 19, no. 8, pp. 762–779, 2000.

[3] M. Tang, Y. J. Kim, and D. Manocha, "CCQ: Efficient local planning using connection collision query," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics (STAR), 2011, vol. 68.

[4] J. Pan and D. Manocha, "GPU-based parallel collision detection for real-time motion planning," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics (STAR), 2011, vol. 68.

[5] C. Schlegel, "Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot," in *Intl. Conf. on Intelligent Robots and Systems (IROS)*, Victoria, Canada, 1998.

[6] E. Behar and J.-M. Lien, "A new method for mapping the configuration space obstacles of polygons," Department of Computer Science, George Mason University, Tech. Rep. GMU-CS-TR-2011-11, 2010.

[7] L. E. Kavraki, "Computation of configuration-space obstacles using the fast Fourier transform," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 408–413, June 1995.

[8] R. Therón, V. Moreno, B. Curto, and F. J. Blanco, "Configuration space of 3D mobile robots: Parallel processing," in *11th Intl. Conf. on Advanced Robotics*, vol. 1–3, 2003.

[9] F. J. Blanco, V. Moreno, B. Curto, and R. Therón, "C-space evaluation for mobile robots at large workspaces," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[10] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *IEEE Intelligent Vehicles Symposium*, San Diego, CA, USA, June 2010.

[11] X. J. Wu, J. Tang, and K. H. Heng, "On the construction of discretized configuration space of manipulators," *Robotica*, vol. 25, pp. 1–11, 2006.

[12] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of Euclidean distance maps and Voronoi diagrams," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

[13] M. Foskey, M. Garber, M. C. Lin, and D. Manocha, "A Voronoi-based hybrid motion planner," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Maui, HI, USA, October 2001.

[14] S. Koenig and M. Likhachev, "D* lite," in *Eighteenth National Conference on Artificial Intelligence (AAAI)*, 2002, pp. 476–483.

[15] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Pasadena, 2008.

[16] J. Canny, "A Voronoi method for the piano-movers problem," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, March 1985.

[17] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*, ser. Springer Tracts in Advanced Robotics, 2004, vol. 7, pp. 43–58.

[18] C. Sprunk, B. Lau, P. Pfaff, and W. Burgard, "Online generation of kinodynamic trajectories for non-circular omnidirectional robots," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, 2011.

[19] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library (OMPL)," 2010. [Online]. Available: http://ompl.kavrakilab.org/