# Online Generation of Homotopically Distinct Navigation Paths

Markus Kuderer    Christoph Sprunk    Henrik Kretzschmar    Wolfram Burgard

*Abstract*— **In mobile robot navigation, cost functions are a popular approach to generate feasible, safe paths that avoid obstacles and that allow the robot to get from its starting position to the goal position. Alternative ways to navigate around the obstacles typically correspond to different local minima in the cost function. In this paper we present a highly effective approach to overcome such local minima and to quickly propose a set of alternative, topologically different and optimized paths. We furthermore describe how to maintain a set of optimized trajectory alternatives to reduce optimization efforts when the robot has to adapt to changes in the environment. We demonstrate in experiments that our method outperforms a state-of-the-art approach by an order of magnitude in computation time, which allows a robot to use our method online during navigation. We furthermore demonstrate that the approach of using a set of qualitatively different trajectories is beneficial in shared autonomy settings, where a user operating a wheelchair can quickly switch between topologically different trajectories.**

## I. Introduction

Approaches to mobile robot navigation typically employ a cost function or constraints to encode desirable trajectory properties. These properties typically include penalties for increasing closeness to obstacles, which partition the space of possible paths to the goal into different ways to navigate around them. Consequently, these topological variants or so-called homotopy classes often correspond to local minima of the cost function.

If the employed cost function accounts for higher order properties such as velocities and accelerations, it is in general difficult to find the globally optimal solution due to the high dimensionality of the search space. Especially methods for online application therefore often rely on an initialization computed in a lower-dimensional space such as a 2D grid. Approaches following this principle exist in the spectrum between reactively following a guidance path [8] and initializing trajectory optimization regimes with a lower-dimensional path [16, 18]. Typically, gradient-based optimization approaches can only find solutions that are in the same homotopy class as their initializations since they are unable to “jump” over obstacles [14]. Unfortunately, methods that provide an initialization leading to the global minimum are not available. Therefore, many approaches resort to sampling techniques to increase the chance of finding the global minimum [10]. Existing methods generate a substantial number of initial paths from the same homotopy class, which in turn lead to the same local minimum after optimization.

All authors are with the Department of Computer Science, University of Freiburg, Germany.

This work has partly been supported by the German Research Foundation (DFG) under contract number SFB/TR-8 and under grant number EXC 1086, and by the Hans L. Merkle-Foundation.

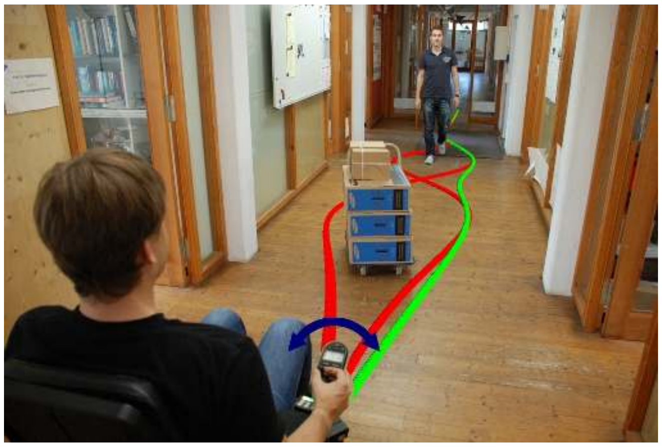

Fig. 1. Application of our approach in a shared autonomy wheelchair navigation scenario. Our method quickly provides a set of homotopically different trajectories. The user can easily choose between topological alternatives or bias the selection process by high-level direction preferences.

In this paper, we propose an online method to explicitly compute a set of homotopically different paths to the goal, i.e., a set of paths that cannot be smoothly transformed into each other without colliding with obstacles. Trajectory optimization approaches can benefit from parallel initialization with these paths from multiple homotopy classes by selecting the best result after individual optimization. In addition, instead of discarding the remaining optimized trajectories after selecting the best one, it can also be beneficial to maintain a set of optimized homotopically different trajectories during navigation. This allows robots to instantly switch to readily available fallback trajectories should the previously selected solution become more costly or even infeasible due to changes in the environment. Furthermore, it enables human-in-the-loop or shared autonomy applications that provide qualitatively different trajectories to the user as illustrated in Fig. 1.

In particular, our method computes a graph representation of the Voronoi diagram from a grid map of the environment. For any given path, the Voronoi diagram contains a corresponding path in the same homotopy class [2]. Hence, our graph representation faithfully captures the information about the different homotopy classes, enabling efficient exploration of homotopic alternatives. We employ our method to maintain during navigation a set of optimized trajectories that are homotopically distinct. To reduce optimization efforts in online applications, we only add alternatives to a set of optimized trajectories if the corresponding homotopy class is new. To this end, we propose an efficient method to capture information about the homotopy class of given trajectories.

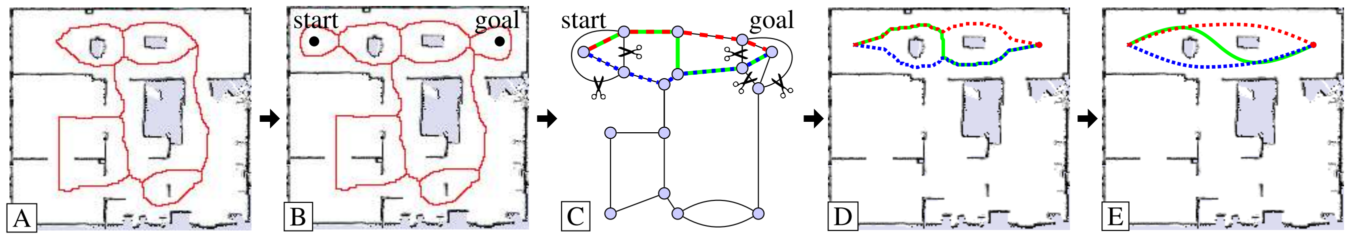

Fig. 2. Overview of the proposed method. (A) Voronoi diagram (red) of an office environment. (B) We add Voronoi cells around the start and the goal location for a robust connection to the diagram. (C) Graph representation of the Voronoi diagram with connected vertices for start and goal location. Each path in this graph corresponds to a homotopically distinct path from start to goal, the three shortest paths are shown in red (dashed), blue (dotted) and green (solid). (D) Trajectories generated from the three shortest paths. (E) Trajectories after optimization with respect to a higher order cost function.

We apply our method to shared autonomy wheelchair navigation, where the user can influence which trajectory the system follows by providing high-level control input. This enables handicapped users to control the wheelchair despite limited fine-motor skills. Our experiments furthermore suggest that for the task of extracting homotopically different paths our method outperforms a state-of-the-art technique by an order of magnitude.

## II. Related Work

Finding feasible and collision-free paths from a start to a goal location is one of the most intensively studied problems in mobile robotics. Obstacles in the environment partition the space of trajectories into homotopy classes. Trajectories are homotopic if they are smoothly transformable into each other without interacting with obstacles. Bhattacharya et al. [3] propose a method to compute paths of distinct homotopy classes. They perform an A$^*$ search on an arbitrary graph representation of the environment that they augment with the $H$-signature to capture topological information. However, their graph may contain multiple paths to the goal within the same homotopy class. To lower the computational burden, we compute a graph in which each path corresponds to a unique homotopy class. We use these paths to seed optimization of smooth trajectories, accounting for higher order properties such as velocities and accelerations.

Vernaza et al. [19] use a method similar to Bhattacharya et al. [3] to find winding constrained paths for autonomous vehicles. They point out that the vector of winding angles around each obstacle is invariant for all trajectories of a given homotopy class. We also use the vector of winding angles to compute an identifier that captures information about the homotopy class of a trajectory.

Demyen and Buro [6] propose a method for efficient triangulation-based path planning that searches an abstract graph representing the environment. Similar to our work, the resulting path in this graph is then mapped back to a trajectory in the original 2D environment. They assume a polygonal representation of the environment. In contrast, we allow arbitrary obstacles on grid cells, which enables us to incorporate online real-world sensor data.

Many authors presented approaches to find optimal trajectories with respect to a given cost function. Sprunk et al. [18] use a spline-based representation of the trajectories and optimize the corresponding control points to find time-optimized, curvature continuous trajectories with acceleration and velocity constraints. Similarly, Gulati et al. [9] optimize trajectories for an assistive mobile robot with respect to user comfort. Ratliff et al. [16] present a general framework for trajectory optimization, which they apply to high-dimensional motion planning for robots. It is well-known that such gradient-based optimization methods often fail to find globally optimal solutions since they are prone to get stuck in local minima. Kalakrishnan et al. [10] propose to use stochastic trajectory optimization to overcome these local minima. However, the large state spaces in complex settings make it infeasible to efficiently find globally optimal solutions by sampling trajectories. Our method provides an initialization for optimization strategies using trajectories in different homotopy classes which often correspond to local minima of the cost function.

Mandel and Frese [15] compare different interfaces that enable handicapped people to steer automated wheelchairs. Their first method allows the user to select a route from a discrete set of paths on a Voronoi graph using speech. Their second method directly maps head poses to steering commands. We also apply our approach to automated wheelchair navigation. We combine the advantages of both methods, i.e., the user is in the control loop but not required to provide low-level steering commands. To this end, we let the user influence the path selection online by providing high-level commands that make the wheelchair transition smoothly between different possible paths.

## III. Homotopically Distinct Paths for Navigation

This section describes our novel approach to efficiently compute a set of homotopically different trajectories from a start location to a target location through a complex environment that contains obstacles. As illustrated in Fig. 2, our method first computes a Voronoi diagram on the current map of the environment. Our approach then builds a search graph that allows us to find the $k$ shortest simple paths on the Voronoi diagram that are homotopically different. We are only interested in simple paths, i.e., paths that contain any vertex at most once, since loops are typically not desired for robot navigation. Our method uses these paths to initialize global optimization in different homotopy classes. By selecting the best trajectory after optimization in each class our method overcomes local minima of a cost function that encodes desirable trajectory properties. Our approach efficiently recognizes homotopic properties of a given trajectory, which allows the robot to re-use previously optimized trajectories during navigation.

## *A. Discretized Voronoi diagram*

The generalized Voronoi diagram is defined as the set of points in free space to which the two closest obstacles have the same distance [5]. We compute a discretized form on an obstacle grid map bounded by occupied cells and represent it as a binary grid map $VD$ in which a cell $(x, y) \in \mathbb{N}^2$ can either belong to the Voronoi diagram or not, i.e., $VD(x, y) \in \{\text{true}, \text{false}\}$. Fig. 2 (A) shows such a discretized Voronoi diagram over the obstacle map and depicts the cells for which $VD(x, y) = \text{true}$ in red.

In this work, we build upon the approach of Lau et al. [13] for an efficient, incrementally updatable computation of the Voronoi diagram. This method employs a wavefront algorithm to calculate distance transforms. It simultaneously starts wavefronts from each obstacle cell that propagate distances to the closest obstacle over the grid map. Wherever these wavefronts meet, it considers cells as candidates for the discretized Voronoi diagram as they are approximately equidistant to at least two obstacle cells. In general, a cell in a grid map will not exactly meet the Voronoi condition of being equidistant to two obstacle cells. Therefore, if two cells are at the boundary of two wavefronts it inserts the one that least violates the condition. It also takes thresholding measures to prevent Voronoi lines from appearing between neighboring obstacle cells and performs pattern matching-based post-processing to reduce discretization artifacts.

In this work, we add a further post-processing method that removes "loose ends" of the Voronoi diagram that typically reach into concave structures like room corners. We detect such loose ends via pattern matching and process them until we reach a branching point on the Voronoi diagram. As a result, we obtain a discretized Voronoi diagram as shown in Fig. 2 (A): the Voronoi lines are sparse (no double lines) and four-connected, i.e., each cell with $VD(x, y) = \text{true}$ has up to four neighbors that are also contained in the Voronoi diagram, see also the magnified part in Fig. 3 (right).

We also employ the "bubble technique" proposed by Lau et al. [13] for Voronoi diagrams. Here, dummy obstacles at the start and the goal provide a robust way to connect to the Voronoi diagram, see Fig. 2 (B) vs. (A).

## *B. Abstract graph representation of the Voronoi diagram*

From the discretized representation of the Voronoi diagram described in the previous section, we build a graph that effectively captures the connectivity of the free space. This substantially reduces the number of states compared to the original grid map representation of the Voronoi diagram. Fig. 2 shows the Voronoi diagram in an office environment (A,B) and the corresponding abstract graph (C).

In particular, vertices $V$ in the graph $G = (V, E)$ represent the branching cells of the Voronoi diagram, also known as Voronoi vertices in the literature. In branching cells more than two lines meet. They thus correspond to locations that are equidistant to more than two obstacles. We can easily identify such branching cells since they have at least three neighbors in the Voronoi diagram. Fig. 3 (right) shows a close-up of such a branching point. The edges $E$ capture the connectivity between the branching cells, i.e., each edge represents a connected line of cells that belong to the Voronoi diagram.

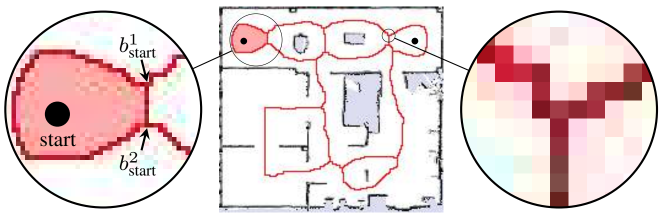

Fig. 3. *Left:* area in the Voronoi diagram that contains the start point. Highlighted are the branching cells $b^1_{\text{start}}, b^2_{\text{start}}$ connected to this area. The start vertex of the resulting graph is connected to these vertices. *Right:* branching cells are connected to at least three neighbor cells.

In addition, we need to connect the start and goal position to the graph. As described in the previous section, we insert dummy obstacles at the start and goal location such that they effectively become enclosed by "bubbles" which are obstacle free by construction of the Voronoi diagram. Using a floodfill algorithm we mark all cells inside these areas and identify the attached branching cells $\{b^i_{\text{start}}\}$ and $\{b^i_{\text{goal}}\}$, as illustrated in Fig. 3. Then, we connect the corresponding graph vertices to the start and the goal vertex, respectively. Finally, we remove the edges that connect the vertices corresponding to $\{b^i_{\text{start}}\}$ and $\{b^i_{\text{goal}}\}$, as illustrated by scissor symbols in Fig. 2 (C). This process removes edges that result from the dummy obstacles to ensure that each simple path in the graph corresponds to one unique homotopy class.

Finally, we set the weights of the edges according to the length of the lines in the Voronoi diagram. The $k$ best paths in the graph then correspond to the $k$ shortest paths in the Voronoi diagram. This graph grows linearly with the number of obstacles in the environment [1]. This follows from viewing the Voronoi graph as a planar graph where the number of faces $f$ corresponds to the number of obstacles. Since each vertex in the Voronoi graph has a minimum degree of three, the sum over the degrees of all vertices $\sum_{v \in V} \deg(v)$ is at least three times the number of vertices $|V|$. Furthermore, any undirected graph satisfies $\sum_{v \in V} \deg(v) = 2|E|$. Hence, we have $2|E| \geq 3|V|$. Combining this with the Euler relation $|V| - |E| + f = 2$ for planar graphs leads to $|E| \leq 3f$ and $|V| \leq 2f$, i.e., the number of edges and vertices is linear in the number of obstacles.

## *C. Finding the $k$ best simple paths in a graph*

In the graph introduced in the previous section different paths always correspond to different homotopy classes in the environment. Therefore, searching for the $k$ best homotopically different simple paths in this graph is equivalent to searching for the $k$ best simple paths. The best known

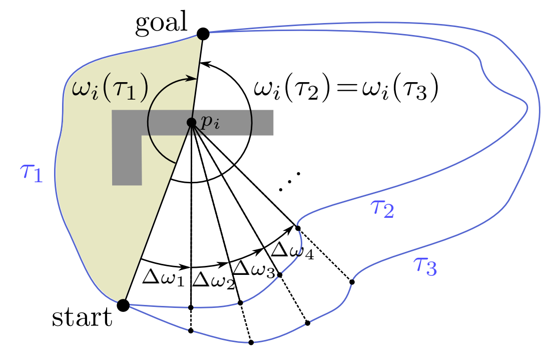

Fig. 4. Computation of the winding angles with respect to obstacles. The figure shows three paths $\tau_1, \tau_2$ and $\tau_3$ that bypass the obstacle $i$. The infinitesimal angles $\Delta\omega$ sum up to the winding angle $\omega_i$ around the representative point $p_i$ of the obstacle. The two paths on the right yield the same winding angle $\omega_i(\tau_2) = \omega_i(\tau_3)$ in contrast to the path on the left.

algorithm for this problem has a runtime complexity in $O(k(|E| + |V| \log |V|))$ [11], which follows from the complexity of $O(|E|+|V| \log |V|)$ of Dijkstra's algorithm [7]. As the number of vertices and edges in our graph depends linearly on the number of obstacles $o$, it follows that our algorithms to extract the $k$ best homotopically different simple paths has a complexity in $O(k(o \log o))$.

## D. Cost functions for navigation

For navigation, we convert the $k$ best paths obtained from the graph search to trajectories $\tau(t) : \mathbb{R} \to \mathbb{R}^2$ that map time to locations. To this end, we retrieve the paths from the grid map representation that correspond to the graph edges $E$ and augment these paths with time information. We use splines to represent the resulting trajectories and use the optimization method RPROP [17] to optimize them with respect to the cost function

$$c(\tau) = \boldsymbol{\theta}^T \mathbf{f}(\tau). \tag{1}$$

This cost function is a weighted sum of features $f_i$ that each map a trajectory $\tau$ to a real value $f_i(\tau) \in \mathbb{R}$. We can manually set the feature weights $\boldsymbol{\theta}$ according to the desired behavior, or learn them from demonstration [12]. In our experiments, we use a cost function accounting for travel time, velocities, accelerations and closeness to obstacles along the trajectories.

## E. Maintaining a set of optimized trajectories

The convergence time of optimization-based techniques typically decreases when the initial guess is already close to the optimum. Thus, during navigation, it is desirable to re-use previously optimized trajectories. In particular, from one planning cycle to the next the current position of the robot as well as the environment do not change substantially in most situations. Therefore, we propose to maintain a set $T$ of optimized, homotopically different trajectories during navigation. However, whenever a new homotopy class emerges due to changes in the environment, we need to add a corresponding optimized trajectory to the set $T$. When an obstacle vanishes, two homotopy classes fall together. Then, $T$ contains two trajectories in the same homotopy class and we can delete one of them. For inserting trajectories

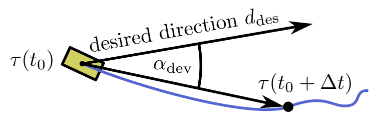

Fig. 5. The angle $\alpha_{\text{dev}}$ used in the feature modeling preferred user direction.

corresponding to new homotopy classes as well as for deleting duplicates we need to identify the homotopy class of a given trajectory, which we will describe in the following.

## F. Identifying homotopy classes

To maintain a set of optimized trajectories as described above, we need to efficiently decide whether two given trajectories belong to the same homotopy class. For efficiency reasons, we identify the homology class of trajectories instead, which is a suitable substitute for the concept of homotopy in practically relevant scenarios [3]. As Vernaza et al. [19] point out, the homology class of a path can be represented as the vector of winding angles around the regions of interest, i.e., the obstacles in the environment. The winding angle $\omega_i(\tau)$ of a trajectory $\tau$ is defined as the sum of infinitesimal angle differences $\Delta\omega$ to a representative point $p_i$ of the obstacle $i$ along the trajectory, as illustrated in Fig. 4.

To determine one representative point for each obstacle, we use a standard flood fill algorithm to determine the cells in each "bubble" in the current Voronoi diagram. In each of these regions we take an arbitrary obstacle grid cell as a representative point to compute the winding angles. Then, we evaluate the trajectory at discrete time steps and sum up the angle differences $\Delta\omega$ as shown in Fig. 4. The step size of this discretization does not change the computed winding angle as long as it still captures the "winding direction". We exploit this by adapting the step size dynamically as we walk along the trajectory to efficiently compute the winding angles. To uniquely identify the homology class of a trajectory, we compute the vector of winding angles $\boldsymbol{\omega}(\tau) = \langle \omega_1(\tau), \ldots, \omega_i(\tau), \ldots, \omega_n(\tau) \rangle$ around all obstacles $1, \ldots, n$. This vector is invariant for all trajectories of a homology (and homotopy) class . Note that we need to compute the angles each time the environment changes, since homotopy classes can emerge or fall together. This affects the set of maintained trajectories, as described in Sec. III-E.

## IV. Application to Shared Autonomy Navigation

Our method maintains a set of trajectories each of which is locally optimized with respect to a user-defined cost function. We realize shared autonomy navigation for mobile robots by adding features that incorporate user preferences online. The robot follows the trajectory that has lowest cost according to a tradeoff between these user preferences and the parts of the cost function that penalize properties such as high velocities or closeness to obstacles. The feature weights determine how strongly the robot follows the user preferences.

Let us assume a wheelchair scenario in which the handicapped user is only capable of issuing high-level commands rather than low-level controls. For example, such a user

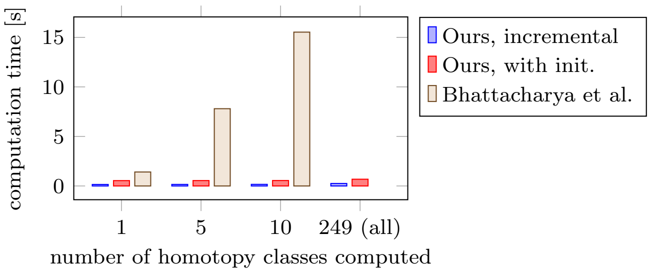

Fig. 6. Comparison of our approach to the one presented by Bhattacharya et al. [3] for the environment shown in Fig. 7. As can bee seen, our approach is substantially faster. Whereas the method proposed by Bhattacharya et al. [3] computes the 10 best paths 15.5 s, our approach finds all 249 possible simple paths in 0.7 s, including map initialization.

might want to express navigation preferences by joystick deflection, by head posture [15] or even through brain-machine interfaces [4]. To achieve this with our approach, we introduce a feature $f_{\text{dir}}$ that penalizes the deviation $\alpha_{\text{dev}}$ of a trajectory from the preferred user direction:

$$f_{\text{dir}}(\tau) = \alpha_{\text{dev}}^2 = \arccos^2 \frac{d(\tau) \cdot d_{\text{desired}}}{\|d(\tau)\| \|d_{\text{desired}}\|}, \tag{2}$$

where $d(\tau)$ is the direction of the trajectory and $d_{\text{desired}}$ is the direction selected by the user, as illustrated in Fig. 5. To compute $d(\tau)$ we use the location of trajectory $\tau$ at the time $\Delta t$ in the future, i.e., $d(\tau) = \tau(t_0 + \Delta t) - \tau(t_0)$. Note that we use this feature only for evaluating the costs of the optimized trajectories in the selection process, not for the optimization itself.

## V. Experimental Evaluation

In this section we present a set of experiments that demonstrate the performance of our method and show that it outperforms a state-of-the-art technique to compute multiple homotopically different trajectories. We furthermore show that our method is suitable for online mobile robot navigation in dynamic environments. Finally, we apply our method to shared autonomy control of a wheelchair.

### A. *Runtime evaluation*

In this section we evaluate the efficiency of our method and show that it is suitable for online computation of paths from different homotopy classes. To compare our method to the one presented by Bhattacharya et al. [3], we use the same environment as they use in their experiments, a $1{,}000 \times 1{,}000$ discretized environment with circular and rectangular obstacles, as illustrated in Fig. 7. For our method, we used an Intel Core2 Duo with 2.6 GHz, which seems comparable to the computer used by Bhattacharya et al. [3]. As Fig. 6 shows, our method outperforms their approach by an order of magnitude. Note that our method needs to initialize only once and can then answer queries for multiple start and goal positions incrementally. Our algorithm finds all 249 possible simple paths on the Voronoi diagram in 0.7 s including map initialization.

Please also note that the objectives of the compared algorithms are slightly different. Bhattacharya et al. [3] aim

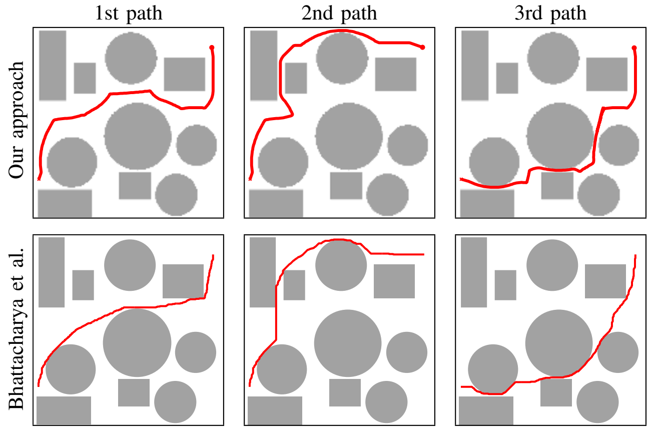

Fig. 7. The first three homotopically different paths computed by our method (top) and the corresponding paths as computed by the method proposed by Bhattacharya et al. [3] (bottom). Our method computes paths using the Voronoi diagram, therefore the paths have higher distance to obstacles.

to find the $k$ best paths that differ in their homotopy class with respect to the A$^*$ cost function whereas our algorithm explores paths that lie on the Voronoi diagram (see Fig. 7 for the differences between the resulting paths in corresponding homotopy classes). However, the 20 best paths returned by both methods have an overlap of 17 homotopy classes for the scenario shown in Fig. 7.

### B. *Trajectory alternatives in dynamic environments*

In this experiment, we evaluate the proposed scheme of maintaining a set of trajectories in distinct homotopy classes, optimized with respect to a cost function comprising velocities, accelerations, the time to reach the target, and closeness to obstacles. We implemented our method and evaluated it on an autonomous wheelchair. The wheelchair localizes itself in the environment using laser-based Monte Carlo techniques and follows the computed trajectories with an error feedback controller. Furthermore, it uses the readings of the laser scanner to constantly update the location of obstacles in the environment.

We evaluated the performance of our method in the corridor shown in Fig. 1. While the wheelchair was traveling through the otherwise empty corridor, a person traversed the corridor from right to left. Fig. 8 shows the reaction of the system: (A,B) The appearance of the person causes the system to generate and optimize a second homotopic alternative. At first, the new variant is not selected due to closeness to the obstacles; (C,D) As the human proceeds to the left the costs of the new trajectory decrease; (E) Only one homotopy class remains after the human leaves the sensor's field of view. Our method is able to evaluate the homotopy classes and optimize the set of trajectories on a standard desktop computer at 5 Hz.

### C. *Shared autonomy wheelchair control*

We also apply our method to shared autonomy control of the wheelchair. We allowed the user to bias the system by adding the feature $f_{\text{dir}}$ to the cost function. Fig. 9 shows how the wheelchair navigated the corridor in which we placed additional static obstacles: (A) Our system has computed a set

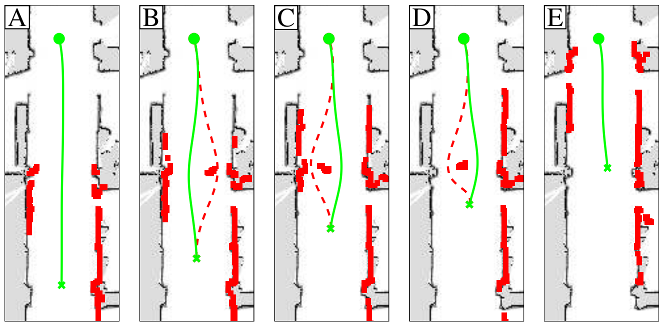

Fig. 8. Online wheelchair navigation in a dynamic environment. The figure shows obstacles detected by a laser scanner (red dots) and the trajectory selected by the wheelchair (green) as well as homotopic alternatives (red, dashed). As a person walks from the right to the left, the system selects the trajectory of the newly emerged second homotopy class.

of optimized trajectories in different homotopy classes; (B) Without user preferences, the system selects and follows the green trajectory since it has lowest costs; (C) The wheelchair discarded one of the trajectory alternatives that require turning around since its costs have exceeded a threshold; (D) In front of the second obstacle, the user turns the joystick to the left, which biases the costs of each trajectory, and the wheelchair selects the left trajectory; (E) The wheelchair follows the left trajectory since it now has the lowest costs.

## VI. Conclusion

In this paper, we presented a novel approach for online-generation of a set of trajectories from start to goal locations that are homotopically distinct. It computes these trajectories from the Voronoi graph calculated from a given map. It then optimizes these trajectories to obtain efficient alternatives that correspond to distinct homotopy classes. It thus effectively overcomes local minima of the navigation cost function. To quickly adapt to changes in the environment, it employs an efficient method to capture information about the homotopy class of a given trajectory. We demonstrated that our method is suitable for navigation in dynamic environments and shared autonomy wheelchair control. We furthermore showed that it outperforms a state-of-the-art approach by an order of magnitude in computation time.

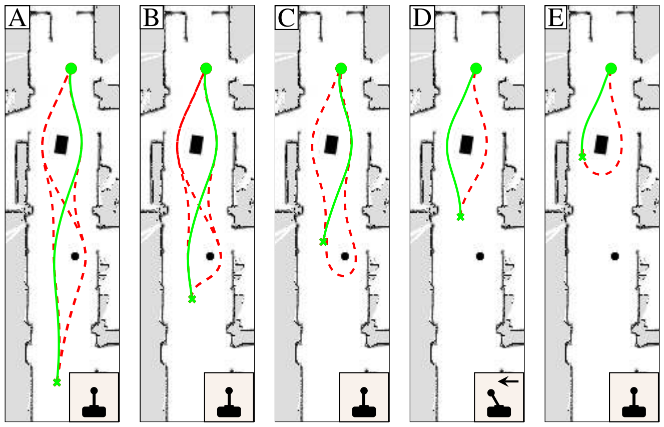

Fig. 9. Shared autonomy control of a wheelchair in a corridor with two static obstacles. The figure shows the best trajectory according to the cost function, which the wheelchair executes (green) and an additional set of optimized trajectories in different homotopy classes (red, dashed). The joystick in the lower right indicates the direction preference of the user.

## References

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.
[2] B. Banerjee and B. Chandrasekaran. A framework of voronoi diagram for planning multiple paths in free space. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4): 457–475, 2013.
[3] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012.
[4] T. E. Carlson and J. d. R. Milln. Brain-Controlled Wheelchairs: A Robotic Architecture. *IEEE Robotics and Automation Magazine*, 20(1), 2013.
[5] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *Int. Journal of Robotics Research (IJRR)*, 19(2):96–125, February 2000.
[6] D. Demyen and M. Buro. Efficient triangulation-based pathfinding. In *Proceedings of the 21st National Conf. on Artificial Intelligence (AAAI)*, 2006.
[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
[8] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine (RAM)*, 4(1):23–33, 1997.
[9] S. Gulati, C. Jhurani, B. Kuipers, and R. Longoria. A framework for planning comfortable and customizable motion of an assistive mobile robot. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.
[10] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.
[11] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12(4):411–427, 1982.
[12] M. Kuderer, H. Kretzschmar, and W. Burgard. Teaching mobile robots to cooperatively navigate in populated environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.
[13] B. Lau, C. Sprunk, and W. Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous Systems*, 61(10):1116–1130, 2013.
[14] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. Section 14.7.
[15] C. Mandel and U. Frese. Comparison of wheelchair user interfaces for the paralysed: Head-joystick vs. verbal path selection from an offered route-set. In *European Conf. on Mobile Robots (ECMR)*, 2007.
[16] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009.
[17] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Int. Conf. on Neural Networks (ICNN)*, 1993.
[18] C. Sprunk, B. Lau, P. Pfaff, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.
[19] P. Vernaza, V. Narayanan, and M. Likhachev. Efficiently finding optimal winding-constrained loops in the plane. In *Proceedings of Robotics: Science and Systems (RSS)*, 2012.