pre-print: *The 43rd Intl. Symp. on Robotics (ISR2012)*, Taipei, Taiwan, 29 – 31 August 2012, pp. 1045 – 1050.

Design and evaluation of a lookup-table based collision-checking approach for fixed sets of mobile robot paths

Eric Demeester, Emmanuel Vander Poorten, Johan Philips and Alexander Hüntemann

Department of Mechanical Engineering,
University of Leuven
Heverlee, Belgium
e-mail: eric.demeester@mech.kuleuven.be

***Abstract*— Several existing mobile robot navigation systems adopt a fixed set of local paths to find traversable areas in the robot's neighborhood. In order to perform collision-checking for these paths or - more generally - to compute a path cost, it is sometimes pre-computed offline which cells in a grid representation of the environment are visited by the robot for each of the paths, and these are stored in a lookup table. Then, in the online phase, just these pre-computed cells have to be checked to efficiently compute the cost and traversability of the paths. This paper presents an alternative for such collision and cost computation of paths by constructing the lookup table the other way around, by storing for all cells which paths pass through them instead of storing for all paths the cells they visit. The algorithm has been implemented and tested on several of our robotic wheelchairs. We show that this approach is almost always faster than the existing approach, and can result in considerable gains in memory and run-time computation.**

Keywords:-path planning, motion planning, collision checking, collision avoidance.

I. INTRODUCTION

Various state-of-the-art mobile robot systems adopt a set of local paths, i.e. paths centered on the robot's current pose, to quickly find traversable areas and to avoid objects in the robot's neighborhood. For example, such sets of local paths have been successfully adopted in the Darpa Grand and Urban Challenges, see e.g. [1][2]. Some of these approaches check a set containing multiple paths, others just verify whether the motor command that will be executed will be collision-free, thereby verifying just one path, e.g. [3]. This collision-checking and cost-estimation approach is also useful for discrete motion planning in general, where a path or trajectory is searched from a start state to a goal state. It is commonly agreed upon that collision-checking and cost-estimation are the bottlenecks in most path or trajectory planning approaches. To speed up this collision-checking, some approaches compute a discretized version of the robot's configuration space (or parts of it) prior to planning, see e.g. [7]. However, these approaches only pre-compute collisions at discrete robot poses, and not for complete robot paths. With the approach presented in this paper, collision-checking for complete paths can be performed.

Such fast collision and cost-computation for a set of paths is also useful in our navigation assistance approach for robotic wheelchairs [4][5]. In our framework, the driver's navigation intention is modeled as a local path from the robot's current pose to a goal pose in the robot's neighborhood, cf. Fig. 1. This intention is estimated online by continuously updating the probability distribution regarding multiple intention hypotheses. For this, fast collision checking of the set of local paths is required.

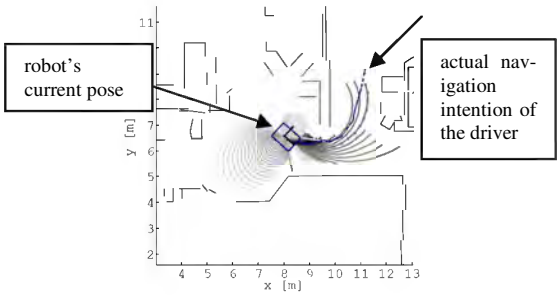

Fig. 1. Shown are a set of local paths centered on the robot's current pose in a typical home environment. The darker grey a local path is, the more probable it is to be the actual driver's navigation intention. The actual intention is shown as a blue dotted line.

The algorithm with which the approach presented in this paper will be compared is presented in Algorithm 1. This shows the online phase of the approach, i.e. it assumes that a lookup table *lookupTableLength(p,c)* has been constructed that yields for each path number p, the length of that path if cell c in the grid map is occupied. It also assumes that a lookup table *cellsP()* is available that yields for each path p which cells in the grid map are visited, ordered according to the time step at which the robot visits the cells when following path p. The algorithm then proceeds as follows. For each of the paths (line 1), the length of the paths is first initialized to a desired value (line 2), for example infinity to indicate that the path is collision-free. Then, for each of the cells c the path visits (line 3), it is checked whether the cell is occupied (line 4), and if so, the length of path p is set and the remainder of the cells in the inner for loop is skipped (line 5). For ease of reference, we will call the approach in Algorithm 1 the *path-based* approach. The algorithm proposed further in this paper will be called the *obstacle-based* approach. To our knowledge, the obsta-

ALGORITHM 1 : PATH-BASED APPROACH (ONLINE)

```
1: for p = 1 to nr_of_paths do
2:     length(p) ← ∞
3:     for c = cellsP(1) to cellsP(nrCells(p)) do
4:         if gridMap(c) == occupied do
5:             length(p) ← lookupTableLength(p,c)
6:             break and continue with for-loop at line 1
```

cle-based approach has not yet been studied in literature.

Section II explains the basic concept behind our approach to collision and cost estimation. Section III then applies this basic concept to a fixed set of paths for the local planning approach. Although not discussed in this paper, this approach can equally well be adopted for discrete motion planning algorithms. Section IV experimentally compares the performance of the path-based and obstacle-based approaches. Conclusions are presented in Section V.

II. BASIC CONCEPT

The most basic version of our collision-checking and cost estimation algorithm takes as input the following elements:

1. **a map of the environment** in the form of a grid map [6]. This grid map contains for each of its cells information about the presence of an obstacle in that cell, and possibly also other information. The cells can be of any shape, depending on the needs of the application. Some possible 2D grid maps are shown in Fig. 2. 3D maps are also possible. At this stage of our algorithm, point clouds, normal distance transforms, or feature maps (e.g. line maps) need to be transformed into a set of discrete cells in order to adopt our algorithm.
2. **a fixed set of *n* numbered paths**, each using a representation of choice, for example splines.
3. **a representation of the robot's shape**. As for the grid map, this can be of any type, for example a union of polygons.

The only requirement for the above elements is that for a given map cell and path it can be determined where along the path the robot's shape collides with the cell in case there is a collision, or, more generally speaking, what the minimum distance of the cell to the robot's shape is at each pose along the path. The relative poses of all cells w.r.t. the paths need to be known beforehand. The robot shape needs to be known as well, though it may change along the path (e.g. due to castor wheels).

The algorithm then proceeds as follows:

1. In the **offline phase**, a lookup table is generated for the given map cells and the set of paths. The lookup table contains for each cell c_i in the map which paths cross this cell c_i, and at which positions along the paths. More generally speaking, the lookup table contains for all cells in the map their contribution to the cost of the paths that come "in the neighborhood" of the cell.
2. In the **online phase**, it can then be efficiently checked for each of the occupied cells, which paths are affected by the cells and at which position along the path. This is explained in Algorithm 2.

In contrast with the path-based approach, our algorithm starts from the occupied cells and updates the paths affected by these occupied cells, rather than starting from the paths and checking which cells they affect. In case there are no obstacles at all, no paths are checked by our algorithm. In contrast, a naïve application of the path-based approach would still check all cells visited by all paths. Our obstacle-based approach first initializes all paths (lines 1-2 in Algorithm 2). Then, it searches all occupied cells (lines 3-4) and updates the paths affected by the occupied cells if necessary (lines 5-7). In order to speed up path evaluation, cells can be checked in a certain order as will be shown further.

ALGORITHM 2 : OBSTACLE-BASED APPROACH (ONLINE)

```
1: for p = 1 to nr_of_paths do
2:     cost(p) ← −∞
3: for c = 1 to nr_of_cells do
4:     if gridMap(c) == occupied do
5:         for p = 1 to nrPaths(c) do
6:             if cost(p) < lookupTableCost(c,p) do
7:                 cost(p) ← lookupTableCost(c,p)
```

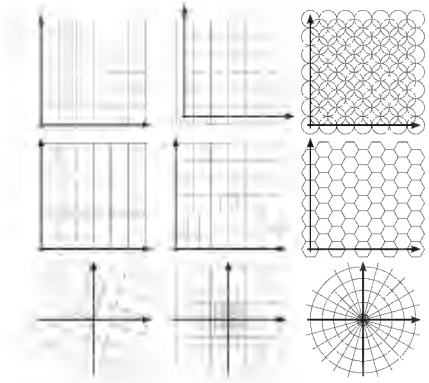

Fig. 2. Examples of grid maps compatible with our algorithm. These maps can be used to model a robot's surroundings. Cells can be of any shape (rectangular, circular, polygonal, etc.), multi-resolution and can have any orientation w.r.t. a reference coordinate frame.

III. CONCEPT APPLIED TO LOCAL PATHS

This section illustrates the cost-estimating approach for a fixed set of paths explained in Section II by applying it to *local path planning*. However, the approach described in Section II can equally well be used for collision checking of path primitives in discrete motion planning algorithms, as in [7]. Section III.A describes the algorithm for the case where only collisions are computed for local paths. In this case, the general *cost()* function in Algorithm 2 only relates to the length of the path, and does not take other cost terms into account such as closeness to obstacles or terrain properties. A long path length corresponds to a low path cost. Section III.B describes several possible extensions to this algorithm. This paper does not focus on the design of the path shapes. Hence, only simple circular path shapes will be adopted. However, the algorithm is compatible with other path shapes as well.

A. Local paths

The case of using a set of local paths can be typically adopted in a safety layer for mobile robots. The purpose is to evaluate a set of paths expressed in the robot's local coordinate frame, e.g. to search for collision-free paths. The robot continuously updates in a grid map the obsta-

cles surrounding it. Whenever it needs to be checked whether certain paths are collision-free, the local path template is placed at the robot's pose, and the lengths of all paths are computed.

1) Input to construct the lookup table

Suppose we choose to check a set of circular paths. Fig. 3 shows a set of n = 200 such paths. The paths are obtained by integrating achievable robot velocities. Therefore, these paths are inherently kinematically feasible, i.e. they respect the robot's non-holonomic constraints. We represent a path as a (v, ω, Δt) tuple, where v denotes an arbitrary but constant linear velocity in m/s, ω a constant rotational velocity in rad/s, and Δt the time during which the velocities are integrated. For car-like vehicles, turning on the spot is not possible. Instead, a minimum turning radius should be respected. Obviously, paths for car-like vehicles can be constructed in a similar way, as well as paths for omni-directional vehicles.

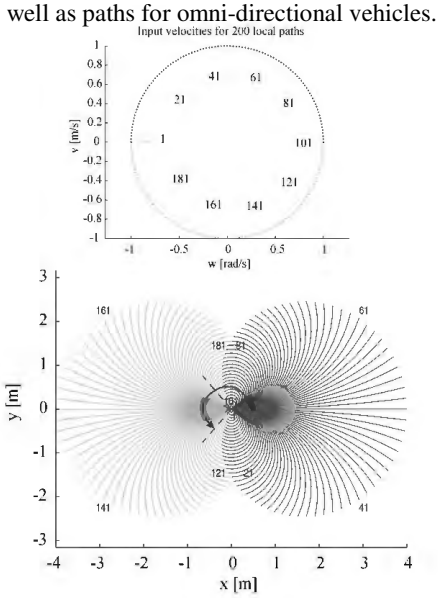

Fig. 3. The bottom figure shows a set of 200 circular paths generated by integrating each of the 200 (v, ω) pairs of the top figure during a time interval of Δt = 4s, where v denotes the linear velocity in m/s, and ω the rotational velocity in rad/s. Dark grey paths indicate forward motion, light grey paths indicate backward motion, and the two dashed black lines indicate turning on the spot over angle $\omega \cdot \Delta t$.

Suppose furthermore that the robot shape of Fig. 4 is chosen. This corresponds to the footprint of our robotic wheelchair, which is a front-wheel driven wheelchair.

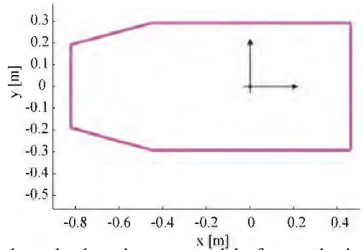

Fig. 4. Adopted robot shape, a model of our robotic wheelchair's footprint. The robot's local coordinate frame is indicated as well; it is positioned in the middle of the two driven front-wheels.

Given the robot's shape and the set of paths, we can determine the area that is covered by the robot when moving over each of the paths. This determines in turn the required area of the local grid map (around 7 m x 9.8 m for the chosen parameters). Furthermore, we choose for a grid with square cells, and resolution Δx = 2 cm. Fig. 5 shows the dimensions of the grid map. It contains around 171.000 cells.

2) Offline computation of the lookup table

For each of the cells in the local grid map, it is now computed which of the n paths pass through this cell, and at which position along the path. This can for example be performed by positioning the robot shape at a number of discrete poses along each of the paths, starting from the begin pose of the path, and to check which new cells are visited in each new step. In case new cells are visited, the path number is stored in that cell, as well as the pose along the path (modeled as a time in our representation). Fig. 6 illustrates this procedure for the circular paths. For the velocities (v_j, ω_j) of the j-th path it is determined which resolution δt will be adopted to go from the k-th pose $\boldsymbol{p_k} = [x_k \;\; y_k \;\; \theta_k]^T$ on this path to the k+1-th pose $\boldsymbol{p_{k+1}}$ by integrating (v_j, ω_j) starting from $\boldsymbol{p_k}$ during time interval δt. The time corresponding to $\boldsymbol{p_{k+1}}$ equals $t_{k+1} = t_k + \delta t$. In our implementation, we choose δt such that no point on the robot moves farther than half the grid resolution Δx. For all new cells that were visited during this time interval δt, the path number j is stored as well as the "time" to collision, t_k.

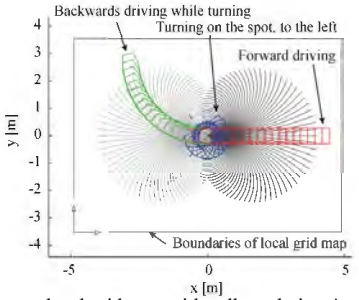

Fig. 5. Chosen local grid map with cell resolution Δx = 2 cm. Also shown are robot poses along 3 different paths: a pure rotation (v = 0 m/s, ω = 1 rad/s), a pure translation (v = 1 m/s, ω = 0 rad/s) and backwards driving while rotating (v = −0.94 m/s, ω = −0.34 rad/s).

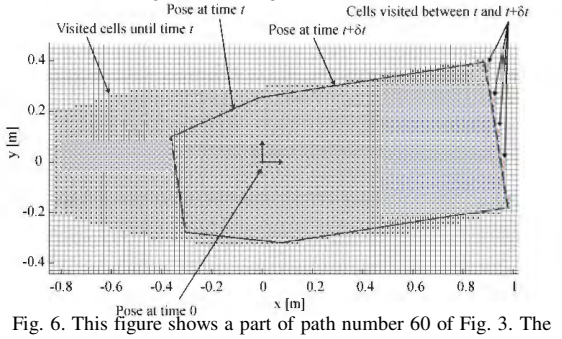

Fig. 6. This figure shows a part of path number 60 of Fig. 3. The cells visited between time 0 and time t_k are denoted with dots. The robot pose at time t_k is shown in grey full line, whereas the robot pose at time $t_k + \delta t$ is shown in dashed black lines. Since δt is chosen small (to accurately determine at which position new cells are visited), these two poses almost coincide. It is checked which new cells are visited (the crosses in the figure, to the right) when integrating the path's velocities (v_j, ω_j) during time interval δt starting from pose $\boldsymbol{p_k}$. For these cells, the path number j and time to collision t_k are stored.

After this procedure, each cell knows the identity j of the paths that pass through it, as well as the time $t_{k,j}$ at which these paths cross that cell.

3) Online verification of collisions

Given the lookup table constructed in the previous section, and given an occupancy grid map of the surroundings of the robot, the set of paths represented with $(v, \omega, \Delta t)$ tuples is updated as follows. For each occupied cell, the Δt value of the paths j that go through it is set to the Δt stored in the lookup table for that cell and that path, unless the path's Δt was already smaller. Fig. 7 illustrates this procedure in case there are two grid cells occupied. Furthermore, the obstacles in the grid map are visited in a certain order: those cells that are encountered earlier on paths are checked first, as shown in Fig. 8. A further optimization may be possible in the future. Suppose a cell is occupied that affects a certain set of paths, then all other cells that only affect a subset of these paths and affect the paths farther along the paths, do not have to be verified.

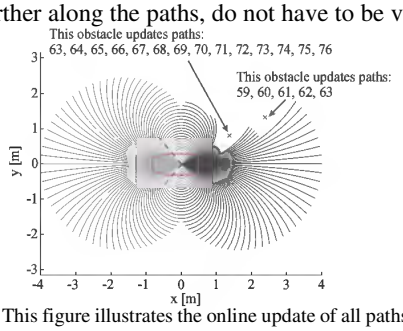

Fig. 7. This figure illustrates the online update of all paths based on two occupied cells, indicated by red crosses. The obstacles are occupied cells in the robot-centered local grid map shown in Fig. 5. In the cells, it is stored which paths should be adapted (indicated by the path numbers in the figure above), as well as their maximum length, which is in our representation determined by Δt. Both obstacles would update path 63, so path 63 is updated by the cell that gives it the smallest Δt.

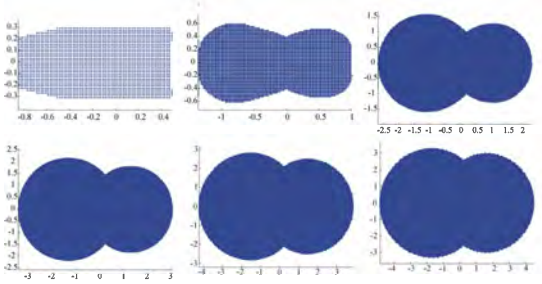

Fig. 8. Several snap shots showing the order in which cells in the occupancy grid are checked. Note that the scale of the axes changes. In total, almost 123.000 cells are checked for the paths in Fig. 3, the robot shape in Fig. 4, and a grid resolution of 2 cm (square cells).

B. Examples of extensions to the algorithm

Both path-based and obstacle-based approaches can be extended in various ways. For example, instead of using a 2D robot shape, a 3D robot shape could be adopted. This would require a certain type of (x,y,z) voxel map. Furthermore, the set of paths could be made speed-dependent, in order to account for the robot's dynamics. For example, when driving at a high speed, turning on the spot is not a feasible motion. Moreover, one could adopt a robot shape that changes along the paths, as in [3] where a wheelchair's rotating steering or castor wheels are modeled.

However, the robot shape can also change in a less predictable way, e.g. due to user-initiated actions as shown in Fig. 9. To our knowledge, this has not yet been covered much in literature. We propose two ways to tackle this changing robot shape. The first approach has a complete robot model for each possible configuration. For each configuration, a separate lookup table is constructed. This approach can be used only if there are a discrete number of robot shape configurations. The second approach we propose is more suited in case the robot shape can vary in a continuous way, as in Fig. 9. In that case we propose to construct a lookup table for a set of elements that together may constitute the complete robot. For each element, a lookup table is constructed. It is first determined online which elements best model the robot's actual shape, and then each of the chosen elements updates the set of paths. This approach is computationally more involved than the first approach. Fig. 10 illustrates the first approach. This shows a simplified 3D robot shape, which can have 2 discrete configurations (e.g. seat tilted and not tilted). Furthermore, a multi-resolution grid is adopted, with high resolution cells in the robot's neighborhood, and low resolution cells farther away. Each robot part moves in a different slice of a 3D voxel map, but updates the same set of local paths.

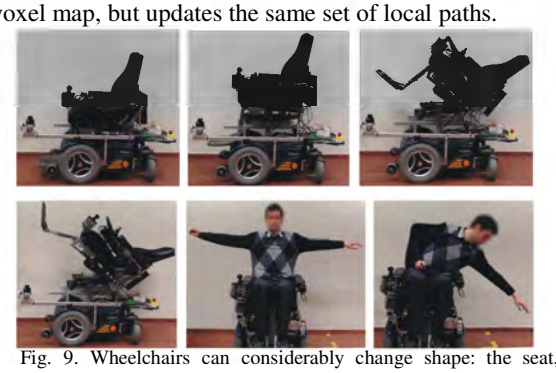

Fig. 9. Wheelchairs can considerably change shape: the seat, backrest and footrests can be lifted or tilted in various ways and degrees. Furthermore, arms and parts of the body may hang outside the wheelchair. In order to dock at a table with the driver's legs under the table, the wheelchair shape should be modeled in 3D.

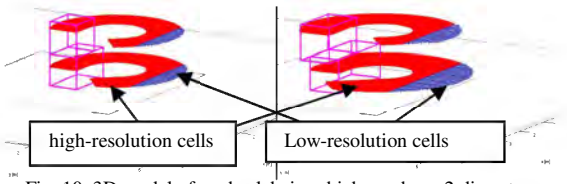

Fig. 10. 3D model of a wheelchair, which can have 2 discrete configurations, modeling a non-tilted seat (left) and a tilted seat (right). Furthermore, a multi-resolution grid is adopted: the red points show the visited cells in the high-resolution, small grid in the robot's neighborhood, the blue cells show the visited cells in the low-resolution, larger grid.

IV. Experimental Evaluation

It is clear that in open areas, the obstacle-based approach has advantages over the path-based approach: the obstacle-based approach would check all cells covered by the paths just once, whereas the path-based approach

would check many cells twice or more. It is still unclear however whether this gain in run-time performance is significant, and how the two approaches compare to each other in more cluttered areas. This section quantifies the gain in performance for a typical real-world scenario.

Fig. 11 (left) shows for all cells c_i the number of paths that pass through c_i for the concrete implementation of Section III.A (robot shape of Fig. 4 and grid cell size of 2 cm), and this for $n = 100, \cdots, 500$ circular local paths. The cells are ordered from highest number of paths passing through a cell, to lowest number. Fig. 11 (right) shows the spatial distribution of the cells and the number of paths that go through them for the case of 200 local paths. The darker a cell, the more paths pass through it. The figures show that there is a significant amount of cells through which more than one path passes. In the case of 200 paths, the implementation of Section I would result in more than 1 million double (and thus superfluous) verifications of whether a cell is occupied or not (for the map resolution and robot shape chosen in Section III.A). For 500 local paths, this is over 2.7 million superfluous checks. This does not only result in reduced run-time performance, these checks are also stored in the lookup table, thereby requiring more memory.

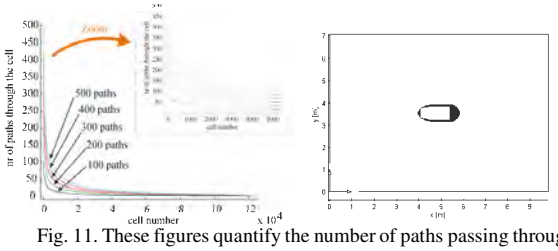

Fig. 11. These figures quantify the number of paths passing through the cells. In the left figure, the cells are ordered based on the number of paths passing through them (for fixed sets of 100 to 500 paths). The right figure shows the spatial distribution of cells that are most visited (for a set of 200 local paths): the darker a cell, the more paths pass through it.

The run-time performance of both approaches is now evaluated for a real-world scenario. The scenario corresponds to driving around in a home-like lab, which was used for testing assistive technology. In such a cluttered environment, the difference between the two approaches is smaller, since the path-based approach of Section I would not have to visit all cells for all paths, but can stop checking a path once the first obstacle on the path is found. Nevertheless, from Fig. 11 (right) it is clear that most of the superfluous checks occur close to the robot. The environment and the path followed by the robot are shown in Fig. 12.

The collision-checking algorithms are run at around 5200 poses along the robot path in Fig. 12. Both algorithms get exactly the same local occupancy grid map as input, and use the same number of local paths (Fig. 3), robot shape (Fig. 4), grid map size and cell size (2 x 2 cm^2). The tests are run on a computer with Intel® Core™ dual core 2.80 GHz CPU with 4 GB RAM. Fig. 13 shows at each time step the run-time t_r each algorithm needs, as well as the number of obstacles. It can be concluded that the obstacle-based approach varies much less than the path-based approach. The path-based approach seems to vary more with the number of obstacles in the neighborhood. Furthermore, our obstacle-based approach outperforms the existing approach at all time steps. Fig. 14 shows the gain $t_{r,p}$ / $t_{r,o}$ in run-time of the obstacle-based approach (with run-time $t_{r,o}$) as compared to the path-based approach (with run-time $t_{r,p}$). It can be seen that in this environment and for the chosen parameters, an average run-time gain of around 3 is obtained.

We will take this set of parameters (200 local paths, a grid cell size of 2 x 2 cm^2 and a grid map of 9.8 x 7 m^2) as a reference case, and will now verify the sensitivity of both algorithms to some of these parameters. For these parameters and the bottom case scenario in which there would be no obstacles at all, the run-time for the obstacle-based approach would be 0.52 ms (which is the time required to check the 123.000 cells in Fig. 8), and for the path-based approach 2.94 ms, resulting in a gain of 5.68.

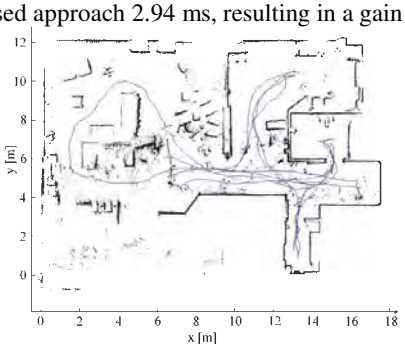

Fig. 12. This figure shows the test environment and the followed path, adopted to compare the performance of the path-based and obstacle-based algorithms.

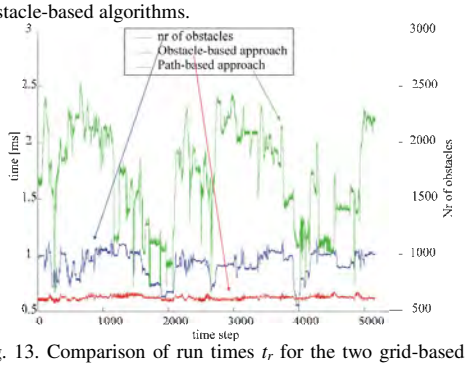

Fig. 13. Comparison of run times t_r for the two grid-based collision-checking approaches. For each time step at which a collision check is performed, the number of obstacles in the local grid map is shown (blue) as well as the computation time for both approaches (path-based approach in green, obstacle-based approach in red). The axis for the number of obstacles is shown at the right side.

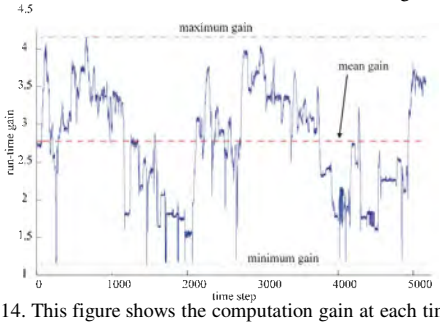

Fig. 14. This figure shows the computation gain at each time step, for the experiment in Fig. 13. The dashed line in red shows the mean run-time gain ($t_{r,p}$ / $t_{r,o}$) of the obstacle-based approach ($t_{r,o}$) as compared to the path-based approach ($t_{r,p}$). This figure also shows the minimum gain achieved (green dotted line), and the maximum gain (magenta dashed-dotted line).

We now analyze the run-times and the gains of both approaches as a function of the number of local paths (Fig. 15) and as a function of the grid cell size (Fig. 16). Fig. 15 clearly shows that the run-time of the obstacle-based approach hardly depends on the number of paths used, whereas the run-time of the path-based approach increases strongly and more or less linearly with the number of paths. Fig. 16 shows that our obstacle-based approach clearly outperforms the path-based approach for higher resolution grids, i.e. grids that allow modeling the environment with a higher precision. This is an important property for robotic wheelchairs, which are large as compared to their environment.

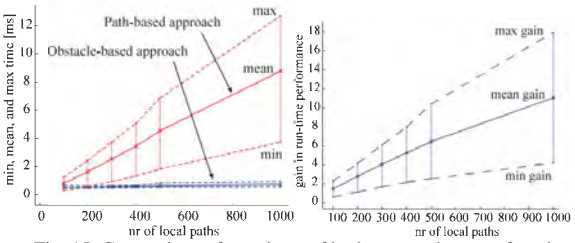

Fig. 15. Comparison of run-times of both approaches as a function of the number of local paths (left), and corresponding computation gains (right). The robot shape and cell size are the same as in the reference case (Fig. 13).

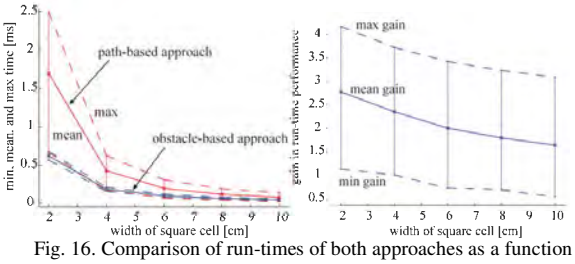

Fig. 16. Comparison of run-times of both approaches as a function of the grid cell size (left), and corresponding run-time gains (right). The robot shape and number of paths are the same as in the reference case (Fig. 13).

Fig. 17 compares the memory requirements for the two approaches as a function of the number of local paths and grid cell size. Typically, we need to store indices (path number, grid cell numbers etc.) and path lengths in the lookup table. For the former, we may e.g. use variables of type `int`, for the latter we may e.g. use variables of type `double`. Fig. 17 (left) shows the required number of `int` stored in memory for the two approaches, Fig. **17** (right) shows the required number of `double`.

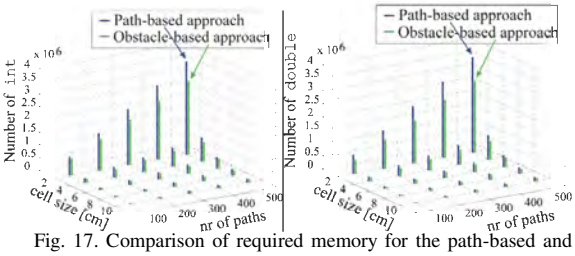

Fig. 17. Comparison of required memory for the path-based and obstacle-based approaches, as a function of grid cell size and number of paths, for the robot shape of Fig. 4.

It can be concluded that for this type of local paths, the required memory increases more or less linearly w.r.t. the number of paths, and increases exponentially the smaller the grid cell size. It can be seen that our obstacle-based approach also performs better memory-wise.

V. Conclusions

This paper proposed a novel, *obstacle-based* approach to collision-checking of sets of local paths, and compared it to a similar *path-based* approach that also uses a lookup table to efficiently check for collisions. In general, it can be concluded that the obstacle-based approach performs better relatively to the path-based approach the more cells are visited by different paths, both regarding run-time and memory. We showed that our approach is almost always faster than the path-based approach, and can result in run-time computation gains of a factor 3 in a typical house environment, for a test set of 200 circular local paths and for a grid-map cell size of 2 x 2 cm^2. We have also shown that our obstacle-based approach requires less memory, and this for an equally easy implementation. The algorithm has been successfully implemented and tested on several of our robotic wheelchairs, with satisfactory results. As the run-time gain can already be considerable for local paths and a 2D grid map, we expect that the gains are even higher for 3D robot shapes and voxel maps, and for discrete motion planning algorithms.

VI. Acknowledgment

This research was funded by the European Community's Seventh Framework Program FP7/2007-2013 - Challenge 2 - Cognitive Systems, Interaction, Robotics - under grant agreement No. 248873-RADHAR.

VII. References

[1] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, P. Mahoney, "Stanley: The Robot That Won the DARPA Grand Challenge," *Springer Tracts in Advanced Robotics*,Vol. 36, pp 1-43, 2007.

[2] F. v. Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, H.-J. Wuensche, "Driving with Tentacles - Integral Structures for Sensing and Motion," *Springer Tracts in Advanced Robotics*, Vol. 56, pp 393-440, 2009.

[3] Th. Röfer and A. Lankenau, "Architecture and applications of the Bremen Autonomous Wheelchair," *Information Sciences*, pp 1-20, July 2000.

[4] E. Demeester, A. Hüntemann, D. Vanhooydonck, G. Vanacker, H. Van Brussel and M. Nuttin, "User-adapted plan recognition and user-adapted shared control: A Bayesian approach to semi-autonomous wheelchair driving," *Autonomous Robots*, Vol. 24, No. 2, pp. 193 – 211, February 2008.

[5] A. Hüntemann, E. Demeester, M. Nuttin, and H. Van Brussel, "Online User Modeling with Gaussian Processes for Bayesian Plan Recognition during Power-wheelchair Steering," *Proceedings of IROS*, Nice, France, pp. 285-292, 2008.

[6] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, pp. 249-265, 1987.

[7] E. Demeester, M. Nuttin, D. Vanhooydonck, G. Vanacker, and H. Van Brussel, "Global dynamic window approach for holonomic and non-holonomic mobile robots with arbitrary cross-section," *Proceedings of IROS*, Edmonton, Canada, pp. 2694-2699, 2005.