HAL
open science

3D path planning and execution for search and rescue ground robots

Francis Colas, Srivatsa Mahesh, François Pomerleau, Ming Liu, Roland Siegwart

To cite this version:

Francis Colas, Srivatsa Mahesh, François Pomerleau, Ming Liu, Roland Siegwart. 3D path planning and execution for search and rescue ground robots. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, Tokyo, Japan. pp.722–727, 10.1109/IROS.2013.6696431 . hal-01143103

HAL Id: hal-01143103

https://hal.science/hal-01143103v1

Submitted on 16 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

3D Path Planning and Execution for Search and Rescue Ground Robots

Francis Colas*[1] and Srivatsa Mahesh[1] and François Pomerleau[1] and Ming Liu[1,2] and Roland Siegwart[1]
[1]Autonomous Systems Lab – ETH Zurich, Switzerland
firstname.lastname@mavt.ethz.ch
rsiegwart@ethz.ch
[2]The Hong Kong University of Science & Technology
Hong Kong, China

***Abstract*— One milestone for autonomous mobile robotics is to endow robots with the capability to compute the plans and motor commands necessary to reach a defined goal position. For indoor or car-like robots moving on flat terrain, this problem is well mastered and open-source software can be deployed to such robots. However, for many applications such as search and rescue, ground robots must handle three-dimensional terrain. In this article, we present a system that is able to plan and execute a path in a complex environment starting from noisy sensor input. In order to cope with the complexity of a high-dimensional configuration space, we separate position and configuration planning. We demonstrate our system on a search and rescue robot with flippers by climbing up and down a difficult curved staircase.**

I. Introduction

Path planning and execution are important milestones for mobile robotics towards autonomy. During missions involving robots, autonomous behaviors can free operators for other demanding tasks than just controlling the robots. Moreover, in dangerous areas for search and rescue, like in Fukushima [1], having a possibility of onboard path planning would allow for recovery procedure like homing in case of loss of communication.

Fig. 1 shows the search and rescue robot we use in this paper. It has two tracks linked by a differential and four independent flippers to help moving in difficult terrain and climb slopes up to 45°. The flippers are controlled in position but with a torque limitation which induces a behavior similar to springs with constant force (independent of the angle). In total, the configuration space of the robot has ten dimensions: three for the position, three for the orientation and one for each of the four flippers. As exteroceptive sensors, this work uses a front-mounted rolling laser scanner that can take full three-dimensional (3D) scans.

Metric path planning in 2D for indoor robots can be seen as a mostly solved problem, for which open-source software solutions are available [2], [3]. Similar algorithms also apply in outdoor scenarios like autonomous cars, but in that case, the problem mostly lies in perception and the embedding of driving rules [4]. Once the free space is identified, planning in those cases can be performed by generic graph path planners such as Dijkstra [5], A* [6] and D*-Lite [7]. Adapting to the kinematics constraints of specific robots can be done with state lattices [8] or via the specification of the cost function between search nodes [9]. In a 2D search space, there are even interpolation solutions to overcome the typical angular defects of grid-based search methods [10], [11].

This work was supported by the European FP7 project NIFTi (247870).

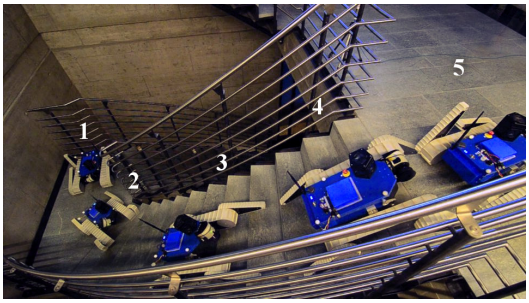

Fig. 1. NiftiBot during autonomous climbing of a curved staircase.

On the other hand, 3D path planning for aerial [12], [13], [14] or underwater [15] robots is also well studied, leading to stochastic approaches that cope with the increased dimensionality like probabilistic roadmaps [16] and rapidly exploring random trees [17], [18]. Those stochastic approaches allow for the exploration of high-dimensional search space which is unfeasible with deterministic methods. However, this is at the cost of the loss optimality.

Search and rescue robots evolve on paths which lie on a 2D manifold embedded in a three-dimensional space. Adding elevation to 2D representation (also known as 2.5D) as exemplified for space rovers [19] is insufficient in urban search and rescue scenarios. This is due to the layered structure of such environments with several floors and staircases. Therefore, there is an additional distinction between free configurations (i.e. not in collision with an obstacle) and feasible configurations that is not present for autonomous cars nor aerial vehicles.

Moreover, this distinction prevents the efficient use of stochastic planners as these assume it is relatively easy to detect whether a new sampled configuration is reachable from other configuration samples: the direct path between two sides of a gap might be free of obstacle, yet the robot might fall in the gap.

Some surface-inspection robots face a similar problem. Stumm et al. [20] consider a small robot with magnetic wheels to inspect steel pipes from the inside. Their robot is constrained to move only on the surface but this surface is a complex 3D object that cannot be accurately described with elevation maps. They propose to use tensor voting to reconstruct a dense representation of a 3D environment based on point clouds. Then, they use the A* algorithm to look for a path. For execution, they unravel the 3D path by stitching local 2D patches and applying a 2D path-following method in the neighborhood of control points in the path. This approach cannot be directly applied to our case as our robot does not magnetically stick to the surface in any orientation.

Furthermore, our robot has more degrees of freedom as the flippers also need to be controlled. Finally, they used external localization and processing and their approach was not applied online.

In the continuity of this approach, we propose a system to solve path planning and execution for ground robots evolving in 3D. We strongly link perception and planning to avoid processing useless information (Sec. II). We leverage the structure of the search space to separate position and configuration planning (Sec. III and Sec. IV). We use a fast replanning algorithm in order to compute a new plan adapted to the current state at each step of the execution (Sec. V). We evaluate this system on our robot in two different environments (Sec. VI). We conclude this paper with a discussion of the limits and choices of our system (Sec. VII).

II. Perception

A. Overall process

A fundamental problem common to all autonomous systems is the construction of an adequate representation of the environment based on the sensor data. To date, there is no generic representation as the constraint of adequacy is specific to each problem.

In our case, the sensor data are mainly 3D laser point clouds, assumed to be registered with the Iterative Closest Point algorithm [21]. This process also provides localization of the robot. What we need is a representation of the environment that can allow us to decide: 1. whether any specific pose is feasible (i.e.: free from obstacle, with enough support, and within roll and pitch angle constraints), and 2. whether we can move between any two neighboring poses. Therefore, we need to build a dense representation from sparse data with varying density. Moreover, we need onboard processing, which excludes computationally-intensive offline methods such as re-meshing of the environment [22], [23].

Stumm et al. propose tensor voting [20] as a way to get dense geometric information. Basically, tensor voting extracts geometrical primitives (plane, line, and sphere for example) and the associated saliency via the voting of points. This process is often implemented in two passes: first the points vote between themselves assuming they are spheres to get a first idea of the local structure (sparse voting), and then the extracted tensors vote according to their shape and saliency to produce a dense grid map for each tensor. Thus the result is a 3D grid, in which each cell holds the saliency and parameters of each tensor (e.g.: normal for planes, or principal direction for lines). For a complete description of the tensor voting process, please refer to [24].

In this work, we use a similar tensor voting process. Its main high-level result is the knowledge, for each cell, of whether there is a local plane and, if so, its orientation. One key parameter of tensor voting is the kernel size, which governs the scale at which the features are considered. For our application, we define the size such that the surface orientation computed by the tensor voting process corresponds to the orientation the robot would have on this surface.

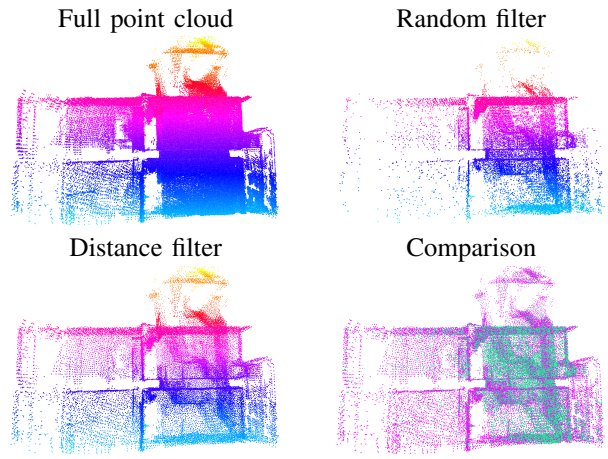

Fig. 2. Comparison of distance and random filters. Top-left: side view of the full point cloud colored by elevation (488,918 points); top-right: point cloud randomly subsampled to 36,014 points; bottom-left: point cloud filtered as exposed (36,014 points); bottom-right: comparison, in magenta the distance filter and in green the random filter.

B. Distance filter

The complexity of sparse voting is in $O(N^2)$, where N is the number of points in the point cloud, as all points should vote at each other. This can be reduced with the choice of a kernel with bounded support to $O(N \times k)$ where k is the maximum number of points in the support of the kernel, which depends on the maximum density of the point cloud. In a previous work [25], we proposed a GPU implementation in order to accelerate the calculation. However, the target robot is not equipped with GPU hardware in this study, hence the necessity to reduce the number of points. A common issue with point cloud maps is that the density of points can be arbitrarily high in some regions while low in other. This prevents random subsampling techniques to be efficient as if we do not want to lose too much information in low-density areas, we cannot subsample too aggressively.

To constrain the complexity of sparse voting, we propose to reduce both the number of points and the maximum density by filtering points that are too close. The algorithm is as follows where d is the closest acceptable distance:

for all $p \in$ point cloud **do**
  for all $n \in$ point cloud such that $\|n - p\| <= d$ **do**
    delete n
  end for
end for

This can be implemented efficiently with a fast kd-tree–based nearest-neighbor search [26], by marking points to be deleted instead of deleting them directly as this would require modifying the kd-tree.This algorithm does not generate an optimal representation minimizing the number of points. Nevertheless, it is a reasonable trade-off between time and memory complexity for subsequent processing.

Fig. 2 shows a comparison between the distance filter and random subsampling in a staircase with two explored floors. As expected, the random subsampling loses points in low-density areas whereas the distance filter keeps a more balanced representation.

TABLE I

USEFULNESS OF LAZY DENSE VOTING SCHEME.

	number of cells		average per cell	
Environment	full	lazy	# of request	computation
Stairs	5,509,350	131,586	1,805	0.474 ms
Office	1,140,700	38,070	1,016	0.577 ms

C. Lazy dense voting

The complexity of dense voting is in $O(N \times G)$ where G is the number of cells in the dense grids. G is cubic as a function of the inverse of the cell size, which can be prohibitive for even modest environments: a quarter billion 5 cm cells in a $100 \times 20 \times 16$ m building.

However, most of those cells are empty and, contrary to flying robots, even more are irrelevant for a ground vehicle (ceiling, hanging features, or higher walls...). Therefore, we propose to do dense voting only where it is needed. This can only be determined during the path-planning phase and we implement it via memoization (in a hash table) of on-demand dense voting for individual cells.

Tab. I compares the full number of cells for various environments (col. 2) with the number of cells for which the computation of dense voting was needed (col. 3). As a mean of comparison the bounding box of the robot is around 385 cells. This shows that the lazy scheme providing on-demand dense tensor voting brings a significant advantage as it reduces the number of computations by an order of magnitude. Furthermore, as we see in the table, the average number of request per cell as well as the average computation of dense voting show that the latter is still a costly process and memoization is especially advantageous to avoid computing many times the same values.

III. PATH PLANNING

A. Dimensionality reduction

As explained in the introduction, in order to reduce the complexity of the planning, we do not plan in the full 10 degrees-of-freedom configuration space at once, but we separate the position, the orientation and the angles of the flippers. This is justified by the following observations:

- the attitude (roll and pitch) of the robot depends on the position and the support surface,
- the heading of the robot depends on the change in position as the robot is not holonomic,
- the angles of the flippers can be determined by the sequence of planned positions.

We can then plan in only three dimensions but we need to ensure that there is always a feasible orientation and configuration of the flippers such that the path is executable. This is done in the specification of the connectivity of the graph (see III-B). Moreover, we need to specify cost functions in order to express the different difficulty and length of nodes and transitions (see III-C).

For the actual path planning, we use D*-Lite [7]. It is a generic least-cost planning on graphs that, contrary to A* [6], can be used for fast replanning in case of a change in costs or the motion of the robot.

B. Connectivity

2D path planning on a graph is usually done without explicit reference to obstacles by not introducing occupied regions into the graph. In the same spirit, we look for feasible (not only free from obstacle) paths by restraining the connectivity to feasible cells. Thus, we consider a base connectivity of 26 neighbors (cardinal directions plus all the diagonals in 3D) and we check, at the expansion of a cell, which of those 26 neighbors are feasible from this cell.

Severable criteria define a feasible neighbor from a cell:

- the robot is not in collision,
- there is a surface to support the robot,
- the surface is oriented so that the robot does not topple.

The not-in-collision criterion is checked by counting the number of points in the bounding box of the robot at the neighbor position, oriented as the robot would be, based on the surface normal and its direction of origin. The support criterion is checked by ensuring a minimum saliency of the support plane, computed by the tensor voting process. The orientation criterion is checked with the normal of the surface computed again by the tensor voting process.

C. Cost functions

When the cells are pushed into the open list, the cost of the path from the start of the search to a certain cell c is computed based on the cost of its neighbors as follows:

$$g(c) = \min_{n \in \mathcal{N}} \{g(n) + cost(n, c)\}$$

where $g(.)$ is the total cost, $\mathcal{N}$ the list of neighbors of c and $cost(a, b)$ is the cost of the transition from cell a to b.

For our application, we define the following cost elements:

- saliency: neighbors with higher saliency are preferred,
- orientation: neighbors with lower slopes are preferred,
- distance: closer neighbors are preferred,
- heading: neighbors for which the direction of the robot will follow the greatest slope are preferred.

These cost elements are straightforward except for the last one, which is motivated by the design of the robot. Indeed, not only do the flippers allow for crossing gaps by increasing the support polygon, but they also allow the robot to be stable on a greater slope, provided that it is in its motion direction. This is reflected by greater pitch range than roll range. In practice, our robot can climb stairs but would topple over if we tried to move in diagonal. This is common to most robots equipped with flippers, as the support polygon becomes an elongated rectangle.

In order to get the weights between those costs elements, we ran parameter optimization based on the length and curvature of the path. This places a relatively high weight on the distance cost while the other cost elements contribute to have a smoother path.

IV. CONFIGURATION PLANNING

The path generated in the preceding section needs to be completed with orientation information and flipper configuration. Orientation can be easily recovered from the path

and the geometric representation as we constrain the roll and pitch angles of the robot by the normal vector of the surface at the location of the robot, and the heading by the path direction.

It would be feasible to use the geometric representation to define the angles of the flippers, so that they lie as much as possible on the surface. This idea stems from one important role of the flippers which is to increase the stability of the robot. However, another role of the flippers is to help the transitions between surfaces of different orientations. For example, when the robot needs to climb a positive obstacle, the front flippers should stay at a fix angle, rather than fold progressively as the robot approaches the step. Similarly, when the robot needs to overcome a negative obstacle, the back flippers can ease the tipping-over by pushing behind, rather than just resting on the surface.

With these observations, we grouped the configurations of the flippers into four posturesaccording to four main situations (see Fig. 3):

- `driving`: the flippers are folded to reduce the footprint and free the field of view of the sensors; used for easy navigation.
- `flat`: the flippers are extended to increase the support surface and traction; used for slopes.
- `approach`: the front flippers are raised to handle a step while the back flippers are extended to prevent toppling backwards; used to overcome positive obstacles.
- `tip-over`: both front and back flippers are set downwards, the back for pushing and the front for landing; used to overcome negative obstacles.

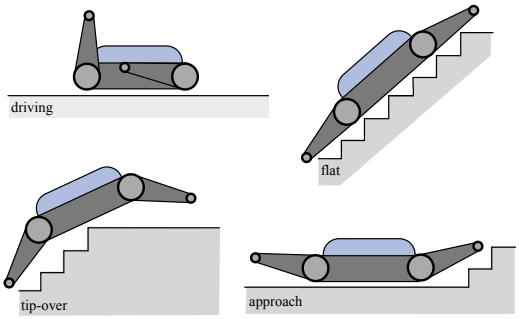

Fig. 3. Flipper configurations used to pass negative and positive obstacles.

A flipper posture is associated with each position of the path planned above. This is done in two passes over the path: first the `driving` and `flat` postures are associated to the position based on their static properties: `flat` above a given slope and `driving` otherwise. Then, the transition postures are set in a second pass according to the change of inclination of the local surface geometry, as computed by tensor voting. `approach` is used for concave edges and `tip-over` for convex edges.

V. Path Execution

Path execution is responsible for generating the control input for the robot in order to reach the goal. There are two main approaches: directly computing the best control input at each time step, or following a predefined trajectory. The latter is needed when the trajectory is imposed and not just the goal, but it is also often simpler than replanning.

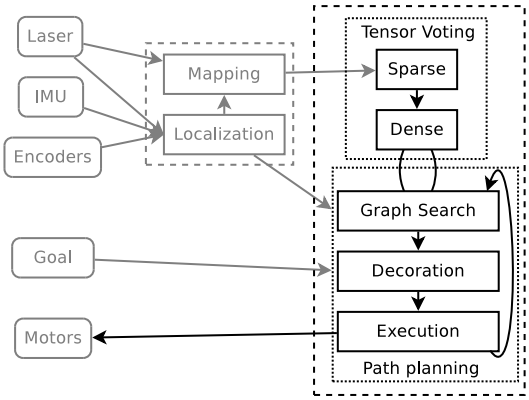

Fig. 4. Overview of the system.

In our case, we use the first approach, made possible by the ability of D*-Lite to do fast replanning both when the costs have changed but also when the starting position (the robot location) has moved. If the robot mostly follows the planned path, the re-planning step is instantaneous as the intermediary nodes have already been expanded. In the case of a deviation of the robot from the plan, the algorithm usually needs to expand a few nodes close to the original path in order to adapt it to the current state. This can necessitate collision checking and dense voting but it is usually relatively fast.

The path, being generated from a grid with neighbor and diagonal connectivity, presents a typical jagged pattern as the direction changes cannot be less than $\pi/4$. Those sharp changes in direction are not ideal to execute for mobile robots, especially on difficult grounds like stairs. However, cutting corners need to be avoided as the path might already be at the limit below which the robot risks collision. Therefore, we compute the control input aiming at the farthest point in the path such that the robot does not stray away more than a given distance boundary. As for the flipper commands, they are generated based on the configuration determined as explained in Sec. IV.

Fig. 4 shows an overall system diagram. Sensor data (laser, odometry and inertial measurement unit) are used to provide online localization of the robot based on the iterative closest point algorithm. This also allows to accumulate the laser scans into a point cloud map. This map is fed to the planner as input to the tensor voting process. The actual path planner queries the tensor map according to its expansion. Once a path is generated, it is decorated with orientation and flipper posture information. Control inputs are then computed for the robot controller. This loop comprising path-planning, path decoration, and command computation runs at a fixed frequency (5 Hz). This loop ends when the robot is close enough to the goal.

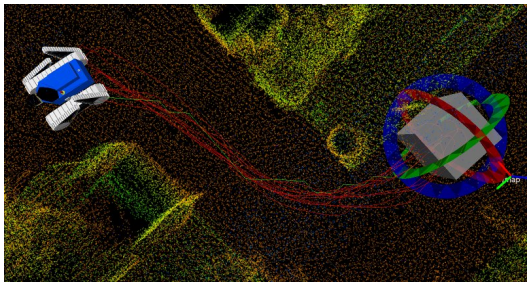

Fig. 5. Top view of the office tests. The robot is on the left and the marker indicating the goal on the right. In green is shown the path planned and in red the path travelled by the robot after 10 times. The point cloud is colored by elevation from orange for the ground to green and blue for tables and ceiling.

TABLE II

PLANNING TIME DURING OFFICE TESTS (TESTS 7-9 OMITTED)

Test	1	2	3	4	5	6	10
Initial planning (s)	29.63	0.34	0.57	0.28	0.46	0.37	1.34
Mean replanning (ms)	26.0	13.9	11.7	4.1	6.1	5.0	0.9
Total replanning (s)	6.71	5.01	3.27	1.07	1.56	1.35	0.25

VI. EXPERIMENTS

All experiments start with driving the robot around in the environment to populate the map. Then the goal is set using an interactive marker in the 3D representation [27]. The goal is validated and altered by the path-planner so that it is obstacle free, it lies on a support surface with enough saliency, and the orientation is compatible with the normal vector of the plane. This prevents setting goals that the robot could not reach as they would be too close to obstacles, or even in the air.

A. Flat terrain

We tested our system in several conditions. First, as a regression test compared to 2D path planners, we validated our planner in flat environments. These experiments also helped to assess the general performance of the system. The environment is an office with desks, chairs and tables. We set the goal alternatively from one end to the other.

Fig. 5 shows a top view of those experiments. The path computed by the planner, in green, presents the typical jagged pattern due to the grid discretization. However, the path execution is able to produce smooth trajectories, in red.

Tab. II shows the time performance of the planner on a standard laptop (core i7-2640M). This tables shows that after the very first planning, the time to compute subsequent paths is in the order of one second. This is because most of the required dense tensor voting operations are done during the first path planning in this environment (test 1, initial planning: $\approx$30 s). Similarly, the mean replanning time is well below the 200 ms control loop duration which validates our approach to do fast replanning at each time step.

B. Turning staircase

The advantage of our approach with respect to 2D or 2.5D path planning is to handle complex full 3D environments in the exact same way as a flat environment such as an office. By definition, 2D techniques are not able to climb stairs as these cannot be distinguished from walls whereas 2.5D representations are not able to properly handle multilevel environments like a building with several floors or even a complete staircase. With our approach, the only difference with respect to a 2D environment is the active need for control of the positions of the flippers. Fig. 1 shows the configuration of the flippers at different points in the trajectory: 1. rolling on the platform, 2. getting on the staircase, 3. during the actual climb, 4. tipping over, 5. rolling on the landing.

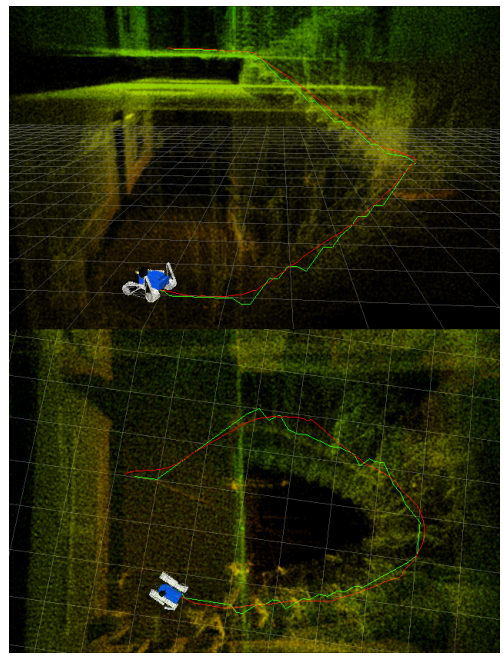

Fig. 6. Climbing down the stairs. Top row: side view, bottom row: top view. The path planned initially is shown in green. The path travelled by the robot is shown in red.

Fig. 6 presents an example of a robot trajectory in a staircase. The robot is able to successfully complete the path. This staircase is particularly challenging because, as it turns, the robot cannot simply be set in the correct orientation to climb down but must really correctly adapt its orientation.

The robot took 387 s and 260 s to climb up and down respectively. This difference is mostly accounted for the slippage, as the velocity commands in both cases are both set to 0.05 m/s. The initial planning time is 27.3 s and 20.5 s respectively, and the average replanning time is 11.0 ms and 21.0 ms. Those numbers are similar to those in Tab. II, which shows that the additional structural complexity of the environment does not directly translate into significantly longer time computation.

VII. DISCUSSION AND CONCLUSION

In this paper we presented a complete system to allow ground robots to climb stairs and, more generally, navigate in a 3D environment. This system is based on point cloud data and does not attempt to fully reconstruct the environment,

but instead uses lazy tensor voting to assess traversability. We evaluated the system in an easy 2D environment and also on a challenging 3D staircase, showing that the robot is actually able to climb stairs up and down.

So far, this system has been implemented for static environments. The map is loaded before path planning and is never updated. Moving to a dynamic environment would pose two different issues: dynamic obstacles that need to be avoided with short time delays, and significant changes in the environment such as the collapsing of part of a building or, more commonly, the closing of a door or the motion of a furniture. Both types of changes need to be properly handled by the mapping software. If mapping is able to signal the addition or subtraction of a set of points, then the result of tensor voting can be updated for the relevant cells and the cost values updated at the path planning level. The choice of D* as search algorithm allows to replan with updated cost with only the necessary computations.

In general, our system is still sensitive to the quality of the representation. Normal vectors for instance, play a central role in both the traversability assessment and the computation of the configuration of the flippers. The saliency computed by tensor voting is also directly use to know whether there is a plane of not. However, the saliency value highly depends on the density of points. Thus some parameter tuning is needed to provide a reasonable scale for this saliency.

Finally, the choice of constant replanning for path execution instead of trajectory tracking, is also not trivial. This choice avoids the complex geometrical problem of both finding a reliable closest point in a 3D path on an implicit surface given the presence of obstacles, and computing control inputs to move closer to the path. However, replanning generates a new path without regard to the past execution. More specifically, it does not take into account the fact that the robot may have just overcome an edge and should still stay in the transition posture for the flippers. We solve this issue by keeping the memory of an eventual transition in progress.

As a conclusion, we believe that the proposed approach has great potential to endow most ground robots with real 3D navigation capabilities.

Acknowledgment

The authors thank Andreas Breitenmoser, Martin Rufli and Elena Stumm for their most valuable input and discussions and Stéphane Magnenat for his thorough review of the paper.

References

[1] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[2] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the National conference on Artificial Intelligence*, 2002, pp. 593–598.

[3] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 300–307.

[4] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, "Autonomous driving in urban environments: approaches, lessons and challenges," *Philosophical Trans. of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4649–4672, 2010.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the national conference on artificial intelligence*, 2002, pp. 476–483.

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

[9] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 127–177, 2010.

[10] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," *Robotics Research*, pp. 239–253, 2007.

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

[12] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 1814–1820.

[13] S. A. Bortoff, "Path planning for UAVs," in *American Control Conference*, vol. 1, no. 6, 2000, pp. 364–368.

[14] J. Tisdale, Z. Kim, and J. Hedrick, "Autonomous UAV path planning and estimation," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, pp. 35–42, 2009.

[15] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *IEEE Trans. on Robotics*, vol. 23, no. 2, pp. 331–341, 2007.

[16] L. Kravaki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[18] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] P. Tompkins, A. Stentz, and D. Wettergreen, "Global path planning for Mars rover exploration," in *IEEE Aerospace Conference*, vol. 2, 2004, pp. 801–815.

[20] E. Stumm, A. Breitenmoser, F. Pomerleau, C. Pradalier, and R. Siegwart, "Tensor-voting-based navigation for robotic inspection of 3D surfaces using lidar point clouds," *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1465–1488, Nov. 2012.

[21] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," *Autonomous Robots*, Feb. 2013.

[22] D. Wettergreen, S. Moreland, K. Skonieczny, D. Jonak, D. Kohanbash, and J. Teza, "Design and field experimentation of a prototype lunar prospector," *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1550–1564, 2010.

[23] S. Singh and A. Kelly, "Robot planning in the space of feasible actions: Two examples," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, 1996, pp. 3309–3316.

[24] G. Medioni, C. K. Tang, and M. S. Lee, "Tensor voting: Theory and applications," in *Reconnaissance des formes et Intelligence Artificielle, 2000. Proceedings of the Conference on*, 2000.

[25] M. Liu, F. Pomerleau, F. Colas, and R. Siegwart, "Normal Estimation for Pointcloud using GPU based Sparse Tensor Voting," in *IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, 2012.

[26] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration," *Journal of Software Engineering for Robotics*, vol. 3, no. 1, pp. 2–12, Mar. 2012.

[27] D. Gossow, A. Leeper, D. Hershberger, and M. Ciocarlie, "Interactive markers: 3-D user interfaces for ROS applications," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, 2011.