Efficient Extensible Path Planning on 3D Terrain Using Behavior Modules

Andreas Hertle Christian Dornhege

***Abstract*— We present a search-based path planning system for ground robots on three dimensional terrain. Effectively negotiating such terrain often requires to utilize dedicated robot hardware and to execute specific behaviors. Our base system is independent from the actual robot configuration, but can be customized to a robot's abilities. We explicitly plan using a full 3d representation, not requiring any projection or slicing to a 2d world. The drivable surface manifold is automatically extracted from the volumetric 3d representation and generic motions are planned on these surface cells. This is achieved with behavior modules that integrate robot skills with the search. Such a behavior module is responsible for defining traversable surfaces, computing if a motion can be executed, and its cost. We implement two such modules: One for sloped ground and ramps, and one for steps and stairs. The approach is evaluated on simulated real-world environments.**

I. INTRODUCTION

Path planning is a central task for a mobile robot system as it enables the robot to operate in various locations in its environment. Many approaches focus on generating plans that only assume robot movements on flat ground, which is usually due to the robot's limitations in mobility. However, in the field of urban search and rescue robots usually operate in unstructured terrain, and even an intact human-centered world poses many obstacles for ground robots, such as ramps, stairs and simple steps, e.g. up a side walk. How and if an obstacle can be tackled by a robot depends on its locomotion abilities. Thus to generate paths in three dimensional terrain it is necessary to take into account the terrain itself and a robot's mobility. A generic answer to this problem is full body motion planning that enables to plan for any terrain and robot (given the implementation supports that). Often this is not necessary as robot skills can be described by certain behaviors like driving up a ramp, taking a step or simply driving on flat ground.

In this paper we present an efficient and extensible approach to behavior-based path planning in three dimensional environments. To be able to adapt to different robots and skill sets we build our approach around *behavior modules*. *Behavior modules* expose robot skills to the path planner and are integrated via a generic interface. The world is represented by a generic 3d occupancy grid using the well known octomap library [1]. As a first step, we extract drivable surface cells and their connectivity from 3d volumetric data. This reduces our possible search space to two dimensional manifolds embedded in the 3d world.

Path planning is performed by using a search-based planning procedure and the search itself is carried out by the search-based planning library (SBPL) [2]. Its generic motion primitives allow us to not only consider the robot's position, but also its orientation during planning – a feature unquestionably necessary when dealing with three dimensional terrain where an obstacle cannot be traversed or even approached from any angle. We define the search space for planning on the extracted 3d surfaces and relay the actual state expansions requested by the search (i.e. specific robot movements) to the *behavior modules* that perform skill dependent tests on the 3d terrain along each motion. Demonstrating the feasibility of this approach are the implementations of two *behavior modules* that we provide:

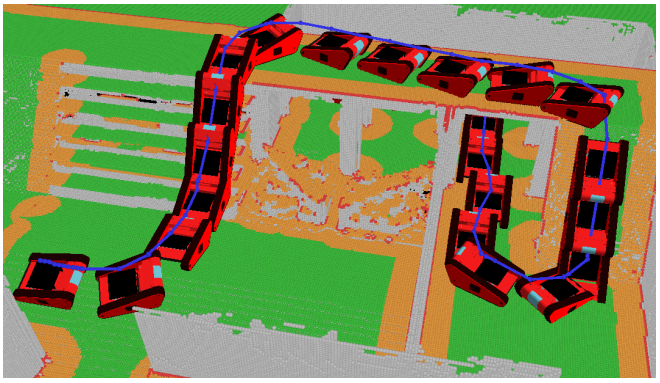


Fig. 1. This figure shows an example plan generated by our algorithm. The three dimensional terrain with ramps and stairs is handled by the specific *behavior modules*.

- The ramp behavior provides the skill for ramps (flat surfaces at arbitrary angles). A special case is the default behavior on flat ground with no ramps as a ramp with zero degrees.
- The step behavior considers steps and in the case of multiple parallel steps can also handle stairs.

Although *behavior modules* can be robot specific, both implementations are independent from the actual robot platform and only require certain skill specific configuration parameters as e.g. the maximum step height for the robot platform to be known. An example plan is shown in Figure 1.

The remainder of this paper is structured as follows: After discussing related work in the following section, we describe our path planning approach in Section III and the behavior

The authors are with the University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany.

This work was supported by the German Research Foundation (DFG) as part of the SFB/TR-8 Spatial Cognition project R7 and by the PACMAN project within the HYBRIS research group (NE 623/13-1).

modules we provide in Section IV. Section V presents an evaluation of this approach and we conclude in Section VI.

II. RELATED WORK

Motion and path planning is a widely researched area in robotics. Often applied is sampling based motion planning [3] that approximates the configuration space randomly. Many applications have been shown for articulated robots like planning footsteps for a legged robot through terrain [4].

Search-based planning uses generic search algorithms as e.g. Anytime Repairing A* [5] to explore the robot's configuration space [6] and might use a sampling-based approximation for that. More often, in this context the configuration space is built explicitly by expanding actions during the search. A lot of successful planning approaches rely on this principle and have been applied to diverse settings from planning full-body motions during navigation of a mobile manipulation robot [7] to providing plans to control autonomous cars [8]. A generic library, the Search-Based Planning Library (SBPL) [2] is available that provides search algorithms and allows to customize the search space and expansions. This library is also used in this work.

Planning on rough terrain is a challenging task that requires an expressive world model that goes beyond 2d maps [9]. Even for wheeled or tracked locomotion one must consider more than just collision-free paths such as the stability of the robot [10]. In our approach this is mainly handled by the *behavior modules*.

Many efficient approaches require a three dimensional world model at least for collision checking, but are able to plan in a two dimensional manner [11], when the environment allow this, even when a more expressive world model is available [12]. The approach by Dornhege and Kleiner [12] also uses behaviors, but is much more limited as only the robot's position is planned for neglecting the orientation.

Most approaches use 2d maps or 2.5d elevation maps, which do not generalize to all environments. Consider the scene in Figure 1, where the robot first traverses a bridge and then drives beneath it. Richer formalisms like multi-level surface maps [13] have been used, for example, for driving in a multi-level parking structure [14]. Kuemmerle et al. [14] split the graph underlying the representation into levels to enable applying a two dimensional planning approach with 3d collision checking. In our case we use an octree representation for our world model [1]. We make no assumptions that necessitate to project to a 2d planning approach and plan directly on the 3d surfaces. Due to the surface manifold extracted from the 3d data, the search space is comparable to a 2.5d representation.

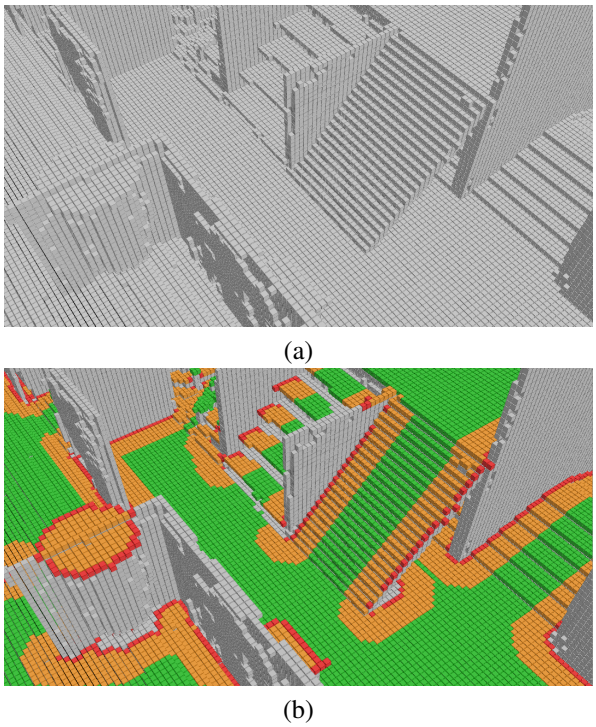
III. PATH PLANNING ON 3D GRID MAPS

We employ a search-based approach to path planning using the SBPL library. Applying this generic approach to our specific problem makes it necessary to specify: the state space of the search, the successor function defining what motions can be used in a state, and a cost function assigning a cost to those motions.

Our state space S for the search consists of tuples (x, y, z, ψ), where x, y, z define a position in 3d space and ψ gives the yaw angle of the robot. Coordinates are discretized to the map resolution and yaw angles are discretized into K bins to match similar states and close them during the search. In our experiments we use $K = 16$. A continuous 6-dof pose $(x, y, z, \phi, \theta, \psi)$ is attached to each state, where the x, y, ψ coordinates are determined by the expansion and the z coordinate, roll angle ϕ and pitch angle θ are derived given the terrain.

The successor generation is based on motion primitives – generic relative motions, e.g. move one meter forward, that are applied to the current state by the SBPL library resulting in queries in absolute coordinates. They allow to customize the path planner to the robot's locomotive capabilities. Each *behavior module* encapsulates a specific navigation skill and defines the possible motion primitives and if a motion is applicable in a state and its cost.

A. Surface extraction from 3d terrain



(a)

(b)

Fig. 2. This figure shows the surface extraction for the behavior modules introduced in Section IV. The input is a 3d occupancy map showing a steep ramp and stairs (a). Extracted surface cells are shown in (b). Ground cells are displayed in green, border cells in red and inflated cells in orange.

Grid maps provide a volumetric representation of 3d data. However, path planning for ground robots only needs to consider the surface when looking for valid paths as a robot can never be inside or levitating above the actual geometry. Other approaches split the world in multiple levels and use a 2d search space within those levels, which is a valid simplification for man-made structures like buildings. However, as we are aiming for a more generic application

scenario this is not possible and given our system also not necessary.

Surface cells are extracted as a fast preprocessing step. We use generic and module specific conditions to determine which cells are valid surface cells. Information computed in this step is stored in the cells and forms the *surface map*. One important feature for this step and also later considerations is the surface normal. Normal vectors are computed from raw sensor data by a multi-resolution plane segmentation algorithm [15] while constructing the map and attached to individual cells. This allows us to take advantage of the full sensor resolution and thus increases the precision of the surface normals.

A first generic condition is that there is a minimum of z_R free space above each cell, where z_R is the height of the robot in any configuration. The second generic condition rules out all cells with surface normals that are tilted more than the robot's safe operation limits in pitch (θ_{max}) and roll (ϕ_{max}) to the z-axis. In addition each module must declare if it can traverse to any neighboring cell. As long as any module names a cell traversable it is kept. It is important to note that this is only a relaxed condition, i.e., a module must not declare a cell untraversable, if there might be any way to do so.

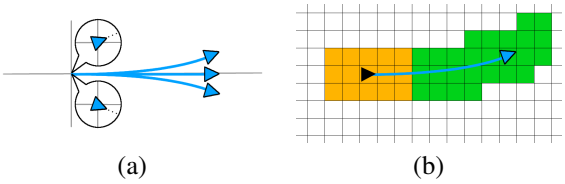
In particular this means that such tests are independent from the actual robot pose. If a cell can be passed by a specific motion is determined later during the actual search. We denote all such cells that can be traversed *ground cells* and cells that have a non-traversable neighbor by all modules *border cells*. These cells are equivalent to obstacles in traditional grid-based path planning. As a final preprocessing step we perform "obstacle growing" for all *ground cells* marking those that are closer to a *border cell* than the interior radius of the robot's footprint. This information is used later for computing an efficient heuristic. An example of the resulting cells is shown in Figure 2. The process is illustrated in Figure 4.

B. Search-based planning with behavior modules

Each behavior module must provide two things related to the actual search: a set of motion primitives and a cost function. A motion primitive defines a displacement $(\delta x, \delta y, \delta \psi)$ as shown in Figure 3(a). As we search in discretized space we precompute discretized displacements for each motion primitive and all K yaw angles and store the set of all motion primitives M. The cost function assigns a cost to each motion primitive for each state as $cost : S \times M \rightarrow \mathbb{R}^+ \cup \{\infty\}$. Returning infinite cost means that the motion is not applicable.

The successor generation performs state expansions and applies all motion primitives to the current state. Computation is then handed off to each module's cost function. All transitions that give finite cost are declared successor states and thus entered in the open queue for the search procedure.

As it is common with search-based algorithms, we use informed search with a heuristic estimate. We base our heuristics on the predicted travel distance and the robot's



(a) (b)

Fig. 3. This figure illustrates the use of motion primitives in a state expansion. Three long distance motion primitives, and two primitives for turning on the spot are shown (a). The long distance motion primitive with start pose (black) and end pose (blue) is expanded. Yellow cells visualize the robot footprint of the start pose, green cells visualize the cells intersected by the robot footprint during this motion. (b)

velocity on flat ground under the assumption that in comparison to flat ground any obstacle behavior will result in higher costs, thus achieving an admissible estimate. Besides the commonly used Euclidean distance metric, which we call h^e, we also use a shortest path search on the *ground cells* denoted as h^{SP}. This heuristic is analogous to two dimensional grid-based path planning methods with the distinction that the *ground cells* are a subset of the three dimensional space and its topology and in general cannot be projected to a 2d grid. We compute the shortest path through all connected *ground cells* that are not inflated, i.e. so that the clearance to the nearest border is greater or equal to the robot's interior radius. We can utilize the SBPL library's A* implementation for this. As no orientations or movements are considered this heuristic considers a relaxation of the actual search.

IV. GENERIC BEHAVIOR MODULES

A. Ramped Ground Module

The ramped ground behavior module provides our default behavior on ground, where no additional robot skills need to be considered. The module handles flat pieces at arbitrary angles that are connected without any notable steps, i.e. connected ramps. Normal floors without any ramp angles are just a special case with a zero degree angle.

If a robot can move on a certain ramp part is defined by a number of limits determined by the robots mobility. When negotiating obstacles robots usually move considerably slow and thus we consider static parameters to determine a stable pose. The slope expressed by the normal vectors of the ground cells must not exceed the robot's roll limit ϕ_{max_R} or pitch limit θ_{max_R}, to prevent the robot from tipping over. A robot will be able to move over small bumps without special considerations, so any height difference smaller than the limit Δz_{bump} can be treated as continuous ground. In addition we want to prevent dangerous changes in roll or pitch angle between two successive poses in a path, e.g. when traversing from a steep upward slope to a steep downward slope. Thus we introduce a maximum roll angle difference $\Delta\phi_{max_R}$ and a maximum pitch angle difference $\Delta\theta_{max_R}$.

Expanding states during path planning must be as efficient as possible. In particular, we want to avoid wasting resources on motions that end up in unfeasible poses. We can provide meaningful relaxed limits during the computation of the *surface map*. Δz_{bump} is independent from the robot pose. We

enforce the Δz_{bump} limit in the *surface map*, by checking the height difference between neighbor ground cells in the map. To enforce angle limits a full 6d robot pose is required, thus these limits can only be validated at state expansions during planning. Instead we check a relaxed limit $\alpha_{max_R} = \max(\phi_{max_R}, \theta_{max_R})$. The intuition behind α_{max_R} is that the robot cannot stand on ground more inclined than both, pitch and roll limits, independent of its particular orientation.

For this module we provide seven motion primitives. Two short distance motions, forward and backward, two turns on the spot, clockwise and counter clockwise and three long distance motions, one straight and the others slightly curved left and right.

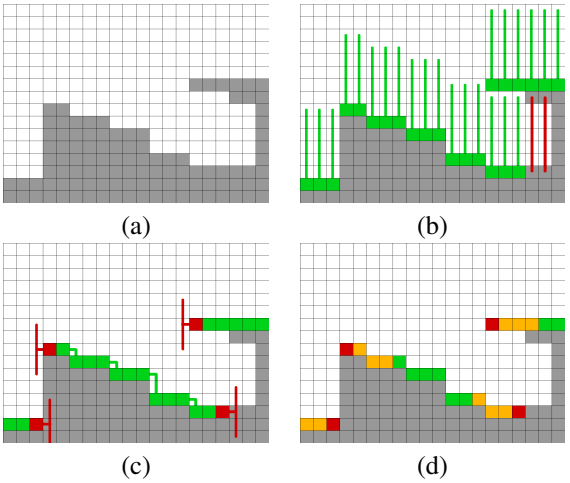


Fig. 4. Vertical cut through a map: (a) Occupied cells are shown gray, free cells white. (b) Ground cells are colored green. Green lines indicate, which cells must be unoccupied to be classified as ground. Red lines indicate, where the ceiling is too low. (c) Green lines show the neighbor relation between cells. Red lines show where cells lack neighbors. These border cells are classified as obstacles. (d) Grown obstacle cells are colored orange.

When expanding a state during search, we apply the displacement of each motion primitive to the robot pose of the current state. This gives us x, y and ψ coordinates of the newly generated state. From our *surface map* we can retrieve all ground cells with these x and y coordinates. The cell vertically closest to the expanded state determines the z coordinate of the new state and the normal vector associated with that ground cell. From the normal vector and the yaw angle ψ we can compute the missing roll and pitch angles. This full 6d pose allows to verify the safety limits.

In addition, we examine the intersecting cells associated with the motion primitive (see Figure 3(b)). Should any limit be violated or any intersecting cells not within the *surface map*, this motion primitive is not applicable. This process is equivalent to collision checking in traditional grid based path planning.

After verifying the applicability of a motion primitive, we calculate its costs. The cost of each motion c_m is determined by the linear distance d_{linear} and the angular distance $d_{angular}$ traveled according to

$$c_m = \max(d_{linear} \cdot v_{linear}, d_{angular} \cdot v_{angular})$$

where v_{linear} and $v_{angular}$ are the robot's linear and angular velocities. We penalize some primitives with a cost factor c_p. For backward motions $c_p = 5$, for all others $c_p = 1$.

In addition, getting close to safety limits is undesirable and thus included in the costs. We collect all safety limits in a vector $\mathbf{L_R}$ and the matching actual values for this motion in a vector $\mathbf{L}$. The final costs for a motion are defined as

$$c = c_m \cdot c_p \cdot \left(1 + \sum_{(l,l_R)\in(\mathbf{L},\mathbf{L_R})} \frac{l}{l_R} \cdot w_l\right)$$

where w_l are weights for each limit that allow to adjust the importance of the limits for the cost computation.

B. Step Module

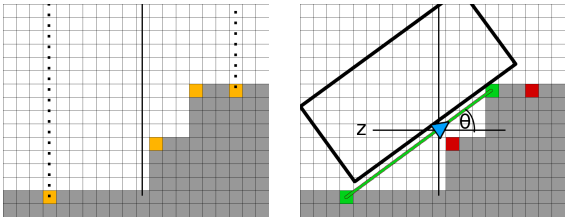
The step module is designed to plan over steps and stairs. In this context we define steps as regular structures with a height difference that forms a straight line edge. This module is not intended to plan over curved or jagged edges. Driving over steps severely constrains the robot's ability to maneuver. For instance the friction between the tracks and the ground is considerably lower, since the area of contact is reduced to the edges of the steps. Therefore, we require the robot to be aligned nearly orthogonal to a step's edge (up to a limit $\Delta\psi_{edge}$) and we do not allow rotating motions on steps. As a result, this module can handle single steps and stairs with parallel steps, but does not allow more generic cases as, e.g. spiral stairs.

To find steps in our map, we identify *ground cells*, where at least one neighbor has a certain height difference. The maximum step height Δz_{step} is derived from the robots ability to climb. The minimum step height is identical to Δz_{bump} from the ramp module, since any height difference below that threshold is considered continuous ground. Any ground cell with at least one neighbor within this height threshold is classified as a possible step cell.

Since we disallow turns on steps, we only have motion primitives for two straight forward motions – long and short – and one short backward motion. During expansions we examine the intersecting cells of motion primitives (as in Figure 3(b)) and retrieve all step cells contained therein. The goal is to find straight and parallel edges forming steps in those cells. We fit lines into clusters of connected step cells using a least squares method. With these extracted edges we can easily detect step cells producing crooked or irregular edges and do not expand over those cells. The extracted edges are attached to the current state eliminating the need to re-fit lines to the same step cells for subsequent expansions. We compare the yaw angle ψ of the robot pose against the direction of all extracted edges and do not expand this motion should any edge differ more than the limit $\Delta\psi_{edge}$.

At this stage we compute the missing coordinates of the new pose, namely z, θ and ϕ. Unlike the ramped ground module these values cannot be simply obtained form the *ground cells* beneath the new pose. Instead, we consider contact points between the robot and the extracted edges. The limits we impose on the edges and the direction of the robot greatly restrict the possible constellations between

robot and edges. Thus we can reduce the full 3d contact point calculation to a 2d approximation by projecting onto the plane spanned by the robot's orientation and the z axis in the map. Figure 5 illustrates the situation. We obtain one contact point from each edge. Two additional contact points are added, derived from the *ground cells* at the front and back of the robot. In this simplified 2d situation a naive linear time algorithm gives us the two contact points supporting the robot base. From the line through both contact points we get the z coordinate for the robot pose and the pitch angle θ. The roll angle ϕ is obtained from the edge direction of the contact point with the lowest z coordinate, as intuitively a greater part of the robot's weight pushes on the lower contact point.



(a) (b)

Fig. 5. This figure illustrates the contact point computation of a robot climbing stairs (side view). (a) The solid black line indicates where the robot pose could be located once the z coordinate is determined. Four potential contact points (yellow) are derived from steps edges and the extent of the robot base (black dots). (b) The two contact points (green) are discovered by iterating through potential contact points and comparing the slope of the connecting line. Potential contact points below that line are discarded (red). The intersection of the connecting line determines the z and θ coordinates of the new pose (blue).

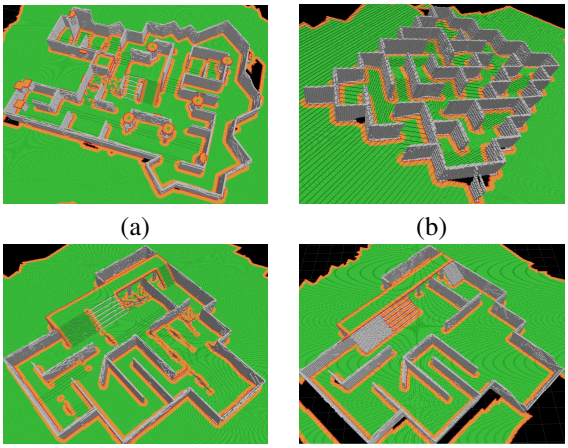
Once we know the full robot pose, we can enforce angle and angle difference limits similar to the ramped ground module. The cost is computed in the same way. However, we provide different values for the roll and pitch limits and weights for the step module since it might be intended to further reduce the allowed roll and pitch angles on stairs, depending on the mobility of the robot.

V. EVALUATION

We evaluate our algorithm on different maps recorded from the Gazebo simulator. The terrain was modeled using obstacles standardized by NIST for robot evaluations such as ramps, steps and stairs that are also used in Robocup Rescue [16]. A tool is available to create such worlds [17]. For mapping a 3d range sensor is moved through the environment and its point clouds are input into normal extraction and mapping with poses from the simulation. As an anytime search algorithm we use the Anytime Repairing A* [5]. An initial weight of $w = 5$ is used for planning and reduced when a plan is found. When $w = 1$ the solution is optimal.

We use four maps in our experiments. An overview of the maps is shown in Figure 6.

- **Robocup German Open 2011 Rescue Arena** (go 2011) is a reconstruction of the actual Robocup Rescue Arena from the competition in 2011. This map was provided by the Darmstadt Rescue Robot Team.



(a) (b) (c) (d)

Fig. 6. This figure shows an overview of the maps used in the experiments: Robocup German Open 2011 Rescue Arena (a) Ramped Maze (b) Freiburg Virtual Arena (c) Freiburg Virtual Arena with flat ground (d).

- The **Ramped Maze** (dc maze) is a reconstruction of a test arena used during a response robot evaluation in the Disaster City test site in Texas. It features a large scale narrow maze with sloped ground. Like its real world counterpart the complete area is tilted by 15 degrees.
- **Freiburg Arena** (fr arena) is a custom built map. Its main features include criss-crossing ramps providing a strong challenge for traditional path planning approaches, a second level connected by stairs and steep ramps to the ground level and rubble blocking off parts of the arena.
- **Freiburg Flat Arena** (fr flat) is a modification of the Freiburg Arena with completely flat ground and no ramps or rubble. The second level is inaccessible.

For each of our experiments we sample 7 random poses on every map. Each pose may not be closer than 3 meters to other poses ensuring good coverage of available terrain. We produce plans for all permutations of start and end poses, resulting in 42 plan requests. This is repeated with a different random seed to get a total of 84 requests per map.

In our first experiment we compare our approach to a conventional path planner performing a 2d grid based expansion scheme that serves as a base line. This experiment is conducted on the Freiburg Flat Arena, where conventional path planning approaches are applicable. Table I shows the results of the base line and our approach using the Euclidean heuristic h^e and the shortest path heuristic h^{SP}. Our algorithm requires significantly more time to find the optimal solution compared to the base line. However, when combined with the shortest path heuristic a suboptimal first solution can be found within a time span suitable for autonomous robots.

The second experiment evaluates the performance of our algorithm on the remaining three maps with difficult terrain. The base line comparison is not available as it cannot cope with the terrain. The results can be seen in Table II. Our algorithm combined with the Euclidean distance heuristic finds the first path after about 10 to 20 seconds. Combined

	base line	h^e	h^{SP}
time [s] to first	0.03 ± 0.06	2.76 ± 4.57	0.66 ± 0.16
time [s] to optimal	0.12 ± 0.15	41.90 ± 46.54	40.75 ± 25.33

TABLE I

THIS TABLE SHOWS THE AVERAGE TIME IN SECONDS UNTIL THE FIRST SOLUTION IS FOUND AND THE TIME UNTIL THE OPTIMAL SOLUTION IS FOUND. WE COMPARE THE BASE LINE APPROACH USING A GRID BASED 2D EXPANSION TO OUR ALGORITHM THAT ADDITIONALLY CONSIDERS LOCOMOTIVE CAPABILITIES OF THE ROBOT.

with the shortest path heuristic, this time is reduced to 0.81 - 3.25 seconds, making our algorithm feasible for online use. The optimal path is found after no less than 80 seconds, which is clearly infeasible for autonomous operation.

We also investigate the quality of the first solution that is found. The suboptimality overhead computes the cost of the first plan divided by the optimal path cost and thus indicates how much more costly the first path actually is. While the first path may be up to 25 percent longer than the optimal path, it requires orders of magnitude less time and state expansions to compute and still provides a reasonable path to follow. We also observe that the shortest path heuristic outperforms the Euclidean distance heuristic in finding the first solution.

		h^e	h^{SP}
go 2011	time to first [s]	7.61 ± 8.09	0.81 ± 0.38
	time to optimal [s]	96.36 ± 111.16	79.13 ± 83.26
	1000 exp to first	21.54 ± 23.02	0.81 ± 0.97
	1000 exp to optimal	264.43 ± 294.31	198.29 ± 216.18
	suboptimality	1.22 ± 0.20	1.25 ± 0.12
dc maze	time to first[s]	11.57 ± 25.28	0.93 ± 0.40
	time to optimal[s]	102.28 ± 140.99	79.45 ± 89.76
	1000 exp to first	34.26 ± 56.51	1.47 ± 1.30
	1000 exp to optimal	295.29 ± 351.14	204.40 ± 200.70
	suboptimality	1.06 ± 0.05	1.25 ± 0.09
fr arena	time to first [s]	21.20 ± 22.44	3.25 ± 2.94
	time to optimal [s]	207.57 ± 193.89	149.93 ± 139.02
	1000 exp to first	55.65 ± 57.39	6.57 ± 6.74
	1000 exp to optimal	554.14 ± 516.19	367.43 ± 337.59
	suboptimality	1.13 ± 0.08	1.17 ± 0.10

TABLE II

THIS TABLE SHOWS THE AVERAGE PERFORMANCE FOR OUR ALGORITHM USING THE EUCLIDEAN DISTANCE HEURISTIC (h^e) AND THE SHORTEST PATH HEURISTIC (h^{SP}). WE RECORDED THE AVERAGE TIME IN SECONDS AND NUMBER OF STATE EXPANSIONS IN THOUSANDS UNTIL THE FIRST AND OPTIMAL SOLUTION WAS FOUND. ADDITIONALLY WE GIVE THE SUBOPTIMALITY OVERHEAD, I.E. THE COST FOR THE FIRST SOLUTION DIVIDED BY THE OPTIMAL PATH COSTS.

VI. CONCLUSION

We presented a path planning algorithm for difficult three dimensional terrain that utilizes generic *behavior modules* to adopt the planning process to the robot's locomotive capabilities. Our algorithm expands with motion primitives, planning a path on the drivable manifold derived from a full 3d representation of the environment. We also provided two generic *behavior modules* that allow to plan on sloped ground and over steps and stairs. For efficient search we used a more informed heuristic than the ubiquitous Euclidean distance and the evaluation showed that our algorithm is able to produce plans in a reasonable time frame and an acceptable loss of optimality.

In future work we plan to move beyond simulation and evaluate our algorithm in greater depth on real robots as our Mesa Element shown in Figure 1. Furthermore, we intent to explore the full potential of our generic module interface by integrating *behavior modules* that consider advanced robot actuators like flippers.

REFERENCES

[1] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010.

[2] M. Likhachev, http://www.sbpl.net.

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

[4] K. Hauser, T. Bretl, J. Latombe, and B. Wilcox, "Motion planning for a six-legged lunar robot," in *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2006.

[5] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Proceedings of Advances in Neural Information Processing Conference (NIPS)*. MIT Press, 2004.

[6] J. P. Gonzalez and M. Likhachev, "Search-based planning with provable suboptimality bounds for continuous state spaces," in *Fourth Annual Symposium on Combinatorial Search*, 2011.

[7] A. Hornung, M. Phillips, E. Gil Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 423–429.

[8] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," *International Journal of Robotics Research*, vol. 8, pp. 933–945, 2009.

[9] P. Vernaza, M. Likhachev, S. Bhattacharya, S. Chitta, A. Kushleyev, and D. D. Lee, "Search-based planning for a legged robot over rough terrain," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 2380–2387.

[10] M. Norouzi, F. D. Bruijn, and J. V. Mir, "Planning stable paths for urban search and rescue robots," in *RoboCup 2011: Robot Soccer World Cup XV*, 2011.

[11] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "A navigation system for robots operating in crowded urban environments," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013, to appear.

[12] C. Dornhege and A. Kleiner, "Behavior maps for online planning of obstacle negotiation and climbing on rough terrain," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, California, 2007, pp. 3005–3011.

[13] P. Pfaff and W. Burgard, "An efficient extension of elevation maps for outdoor terrain mapping," in *Field and Service Robotics*. Springer, 2006, pp. 195–206.

[14] R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun, and W. Burgard, "Autonomous driving in a multi-level parking structure," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009, pp. 3395–3400.

[15] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke, "Efficient multi-resolution plane segmentation of 3d point clouds," in *Proceedings of the 4th international conference on Intelligent Robotics and Applications*. Springer-Verlag, 2011, pp. 145–156.

[16] A. Jacoff, B. Weiss, and E. Messina, "Evolution of a performance metric for urban search and rescue robots," in *Performance Metrics for Intelligent Systems*, 2003.

[17] J. Simon and S. Kohlbrecher, http://ros.org/wiki/hector_nist_arenas_gazebo.