Real-Time Footstep Planning Using a Geometric Approach

Philipp Karkowski Maren Bennewitz

***Abstract*— To this date, many footstep planning systems rely on external sensors for mapping and traversability analysis or on computationally expensive algorithms that do not allow for real-time calculations. In this paper, we present an approach that analyzes the environment in the vicinity of the robot with an onboard RGBD camera while computing local footstep plans in real time. We achieve this by combining the advantages of grid-based height maps, fast planar region segmentation, and a systematic local footstep search to a local goal point. Using a single CPU core, a full mapping and planning cycle only takes 18 ms on average and thus presents an important step to autonomous humanoid robots in dynamic environments that only rely on their onboard hardware.**

I. INTRODUCTION

In the last decade, the field of humanoid robotics has continuously grown due to the capabilities of humanoids with their human-like body plan. One of the major tasks in humanoid navigation consists of determining feasible footstep paths that lead the robot to a desired destination. This task, however, proves itself to be very difficult if one needs to take into account regions that might cause instability in the walking behavior. Many well-established methods to solving this problem use optimal, anytime, or randomized path planning algorithms such as *A**, *AD**, *ARA**, or *R** [1]–[3]. Due to the calculation of the 2D cost map used for collision checking of the footsteps and the exploration of the search space as defined by a given set of possible footsteps, these algorithms are computationally expensive. Other approaches use 2.5D height maps or 3D maps and directly plan footsteps upon convenient walking regions [4], [5]. However, the amount of data to be processed leads to run times that are generally not suitable for dynamic environments. While optimized implementations of the above algorithms may find a complete footstep path to the goal location in the range of seconds [3], calculations are done on a modern external computer and expect an already provided map of the environment.

In this paper, we present a highly efficient technique to planar region segmentation and a systematic geometrical approach to 2D footstep planning. Our method first computes a local height map based on the 3D point cloud acquired with the robot's onboard sensor and performs a segmentation of the height map into different planar regions. The edges of the segmented regions are subsequently used for collision checking during path planning. Our system first finds a collision-free 2D path and computes a footstep plan that closely follows this path afterwards. For generating a valid footstep plan, we apply a fast method that systematically searches for collision-free footstep locations within the stepping range of the robot. An example path and footstep plan can be seen in Fig. 1.

P. Karkowski and M. Bennewitz are with the Humanoid Robots Lab, University of Bonn, 53113 Bonn, Germany. This work has been supported by the European Commission under contract numbers FP7-ICT-600890-ROVINA and FP7-610532-SQUIRREL and by the German Academic Exchange Service (DAAD) under contract number 57178163.

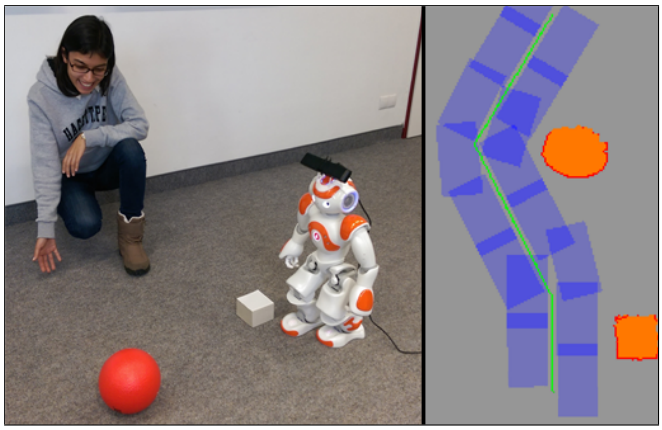

Fig. 1: Real-time footstep planning. Our framework is capable of adapting to local changes in the environment at a delay of just 18 ms on average.

As our experimental results show, our framework is capable of both analyzing the local environment using a RGBD sensor and computing a local 2D path along with a full footstep plan in 18 ms on a single CPU core on average. Since the processing power onboard a humanoid is generally limited, our approach is an important step in the direction of fully autonomous humanoids that are capable of quickly reacting to changes to the local environment.

II. RELATED WORK

In this section, we first discuss related approaches to plane segmentation and traversability estimation and then review footstep planning techniques.

Many approaches to detecting planar regions from point cloud data exist that generally provide accurate segmentations. For example, Deits *et al.* find convex regions around known obstacles using the IRIS algorithm [6]. Hulik *et al.* applied accurate methods that use second derivate estimates and find planar sections at zero curvature [7], which becomes a very expensive task if higher order polynomials are used. Other common methods use local plane normals, e.g., Fallon *et al.* who achieve fast segmentation results based on stereo-fusion of images [8], while Holz *et al.* and Trevor *et al.* use images from RGB-D cameras [9], [10]. Even though segmentations in these approaches can be obtained well below a second, run times are usually still too high to immediately react to changes in the environment and the used

point cloud based maps appear to be too complex for real-time footstep planning. Stumpf *et al.* propose 2.5D height maps and are capable of planning longer footstep paths also on inclined terrain. However, instead of detecting planar regions they use ground contact estimates to ensure stability [5]. In our approach, we combine 2.5D height maps and fast segmentation of planar regions based on plane normals and region growing.

Chestnutt *et al.* and Michel *et al.* implemented *A** and informed local searches to reach a goal [1], [11], [12], however, the methods need appropriate heuristics to achieve fast results and computations still last for a few seconds. Hornung and Maier *et al.* proposed to use *ARA** or *AD** to find paths around obstacles and replan if necessary [2], [4]. Even though faster than standard *A**, these algorithms are not real-time capable. Additionally, all approaches based on A* rely on a predefined set of possible footsteps that need to be constrained to avoid a high branching factor during the search. Deits *et al.* compute paths with an *MIQCQP* planning algorithm within convex regions [13]. While finding near optimal solutions, user-guided detection of convex regions is necessary.

Adapting footsteps to environmental changes in real time becomes progressively more difficult the closer the changes are to the robot. Baudouin *et al.* use *rapidly exploring random trees (RRT)* and precomputed dense swept volumes to find collision-free footsteps [14]. Although being able to adapt to changes without stopping the walk, adaptations can only be made two steps ahead and the framework requires an external motion capture system. The approach by Fallon *et al.* is also capable of online replanning in 3D using a high-end computer, however, calculation times of 1.5-2 s do not easily allow to react to dynamic obstacles [8].

Our technique uses systematic searches to find a path to the goal point while checking for collisions. It does not require a discrete set of footsteps, which leaves the planner with more search possibilities while still maintaining high performance.

III. FAST PLANAR REGION SEGMENTATION

We developed an approach that allows for fast footstep planning and instantaneous replanning based on estimated planar regions. We use a 2.5D height map instead of a complex 3D map as environment representation, since even modern methods are still rather slow when it comes to detecting planes from the 3D model [7]. A height map is an evenly-spaced grid map, where each grid cell contains the height information of the highest measured point that falls into the cell.

Stable foot placements can in general only be made upon planar terrain without further stability control. In addition, it is essential to retain feet from edges, e.g., corresponding to the end of steps or object boundaries. Thus, our approach generates a segmented map from the height map where each section defines a connected planar region. Furthermore, all areas that are not detected as planes or shadows, which do not contain any height information, are marked separately.

Algorithm 1: Segmenting the height map into planar regions which are safe for footstep placements.

input : Height map
output: $Segments$

$Fails \leftarrow 0$;
$Cells \leftarrow$ full set of grid cells;
$Segments \leftarrow \{\}$;
while $|Cells| > minCellsLeft$ **and** $Fails < maxFails$ **do**
  $Cell \leftarrow$ choose random Cell from Cells;
  if *cells in vicinity around* $Cell$ *have similar normal* **then**
    $Fails \leftarrow 0$;
    $Plane \leftarrow$ set of cells around $Cell$ found by region growing;
    $Cells \leftarrow Cells \setminus Plane$;
    add $Plane$ to $Segments$;
  else
    $Cells \leftarrow Cells \setminus Cell$;
    $Fails \leftarrow Fails + 1$
  end
end
return *Segments*

Alg. 1 provides a general outline of the steps that are performed during the segmentation. Our method randomly selects grid cells and finds normal vectors using triangulation between the selected cell and cells in a small vicinity. If the normal vectors for the selected cell point in the same direction, region growing then allows to find the set of all cells that belong to the plane. The region is grown by checking whether neighboring cells are not more than a predefined threshold away from the plane given by the average normal vector around the randomly chosen cell.

Right after segmentation we create a third map that contains information about detected edges. While well-established edge detection algorithms are able to accurately detect edges in 3D environments or 2.5D height maps [15], [16], the computation time can easily take several seconds and is, thus, impractical for real-time footstep planning and dynamic changes to the environment. As the segmentation map already contains all necessary information about planar regions that are possible candidates for footstep positions, we define an edge grid cell as having at least one neighbor that does not belong to the same planar region. The frontiers between planar cells and shadows are, however, not defined as edges. Since edges are used for collision checking during path planning, this allows the path planner to plan into unknown local regions, which will eventually be covered by the sensors during navigation. On the right of Fig. 2 we illustrate the segmented map as well as the different edges for an example height map.

For path planning, we use a local map that is computed from a certain number of previously calculated maps by considering the estimated transform between subsequent steps while the segmented regions are matched according to

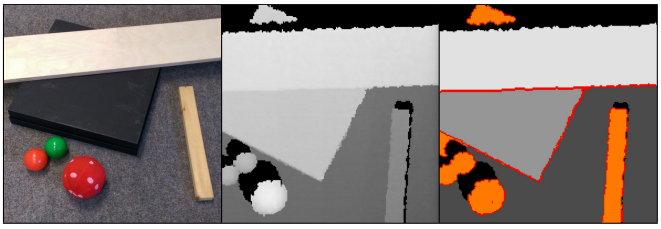

Fig. 2: Example segmentation scenario. The middle image shows the calculated height map where darker colors represent lower regions and shadows are fully black. On the right image, detected planar regions are colored in different shades of gray while uneven surfaces are orange. All edges are represented in red and non-visible shadows are again black. The small wooden stick on the right was not detected as planar, because it is too narrow for finding the normal using triangulation and the width is smaller than the humanoids foot size.

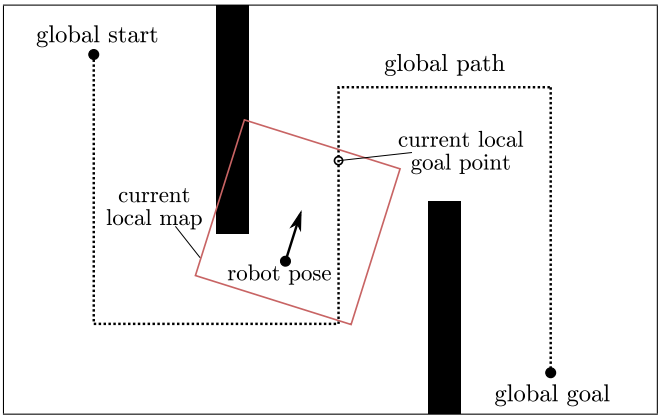

Fig. 3: Global path following. In order to follow a global path, the framework selects local goal points using the furthest reachable point of the global path that lies within the local map.

the largest overlap. We also create an artificial edge border around the local map, to retain the planning within.

IV. PATH PLANNING

We use a two-stage approach for path planning. First, we plan a 2D path that respects a safety distance d_{safe} to detected edges and follows a global path. Then, we compute a footstep plan that closely follows this local path.

A. Global Path Following

The goal point for the local path planning is calculated based on a global path computed by A* on a global 2D grid map that contains the static obstacles in the environment. We assume the position and orientation of the robot relative to the global path to be known, e.g., estimated by self-localization. Our system first finds a local goal point by determining the furthest collision-free point on the 2D path inside the local planning region[1], this is illustrated in Fig. 3. In case no solution can be found by local path planning (as described in the next subsection), the search is restarted with a new local goal point closer by a predefined distance to the current position of the robot until a successful plan is found.

[1]If the global 2D path lies fully outside the planning region, e.g., caused by avoiding a large obstacle, replanning of the global path is necessary.

While walking along the path, the robot chooses a new goal point for local planning once the distance between the current one and the robot's position falls below a threshold. Furthermore, after every step the robot takes, we check the remainder of the current 2D path for collisions. If a collision is detected, planning is restarted with a new goal point which is found as described above.

Even if the remainder of the path is collision-free, our framework checks, at every iteration, if a recalculated 2D path to the current goal point is considerably shorter than the one currently followed by the robot. This allows to immediately react to changes to the local environment such as removed obstacles or moving people.

B. Searching for a Local Path

We now introduce our method to compute a 2D path from the current robot pose to the next goal point within the local planning region. Our approach represents this path as a set of connected line segments and iteratively tries to find a direct connection to the goal point while checking for collisions with cells belonging to the safety regions around detected edges. The search consists of the following main steps:

1) The process starts with two nodes, one start node, which is initially at the robot pose, and one node at the local goal point. The line segment connecting these nodes is checked for collisions beginning from the start node. If a collision is detected, we start a search for possible alternative directions as explained in the following.
2) The line segment connecting the start node and the point of collision is rotated by an angle $\Delta\theta$ around the start node and is checked for collisions. Should a collision again be detected, the end point of the segment is changed to the new point of collision. This procedure continues until a maximum rotation angle $|\theta_{max}|$ is reached in both directions. Both situations are shown in Fig. 4a.
3) If a collision-free rotation is found, we expect there to be a discontinuity in detected edges and use the rotated line segment to create a new node as follows. We recheck the entire line segment, from the start to the border of the local map, for collisions. The distance between the new point of collision and the end point is denoted as d_c. A new node is then either created in the middle between these two points or, if $d_c > 2 \cdot d_{\text{safe}}$, at the distance d_{safe} from the end point. This is shown in Fig. 4b. The motivation behind selecting the new node in this way is that the path leads closely around obstacles.
4) The search can result in three possible outcomes, either no free rotation is found in either direction or free rotations can be found in one or both directions. If only one free rotation exists, the newly created node is used as the new starting point and the search to the goal starts over beginning with Step 1. If two nodes were created, however, the search continues using the node

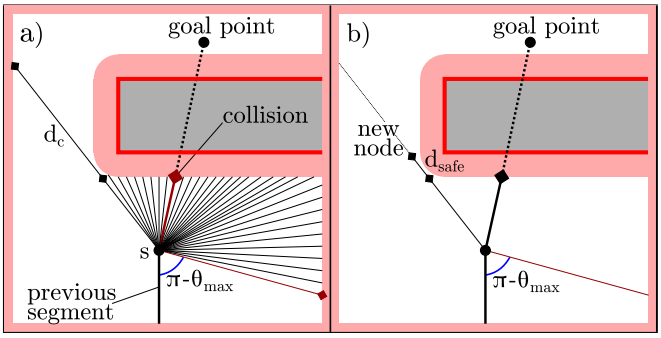

Fig. 4: Creation of a new node. Left: If a collision along the direct line from the robot's pose s to the goal point is detected the framework systematically searches for possible free rotations. Right: If a free rotation is found, a new node is created and the path search continues using the new node.

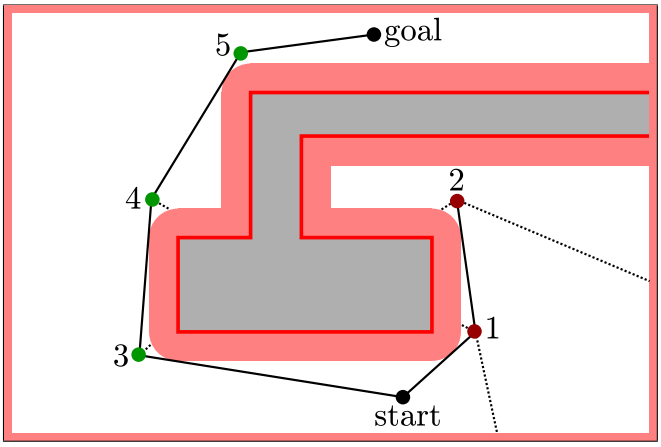

Fig. 5: Example search graph. The search first inspected the nodes 1 and 2 but did not find a free way to the goal and thus continued starting from node 3. The order of processed nodes during the search is thus: start-1-2-1-start-3-4-5-goal.

that causes the minimum change of rotation relative to the previous segment.

If no free rotation is found, the node is declared *terminal* and we return to the previous node. Should either search direction have not yet reached the maximum search angle, the search first continues on this node until a free path is found or this node is declared terminal as well. This search and backtracking procedure continues until either a path to the goal could be found, or the original start node is declared terminal, in which case the path search was unsuccessful. A possible graph is depicted in Fig. 5.

C. Footstep Planning

Once a path to the local goal has been found, our system computes a sequence of footsteps starting from the current support foot of the humanoid. We hereby use a step region for the feet defined by two ellipses centered at the zero points of the feet. The zero point is at a distance of $2d_0$ in the inward direction from the support foot. The axes depend on the maximum stepping distances of the humanoid in the forward, sideway, and backward directions, m_f, m_s, and m_b respectively. Furthermore, the ellipses are limited by inward angles α_f and α_b as depicted on the left of Fig. 6.

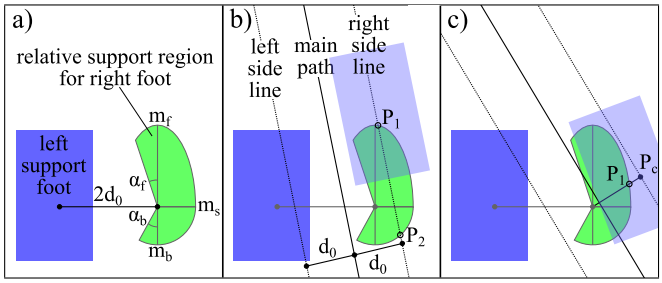

Fig. 6: Footstep calculation. Left: The reachable area of the next footstep is defined by an ellipse-shaped region with axes m_f, m_s, and m_b displaced by a distance $2d_0$ from the support foot. Middle: Intersection P_1 of the boundary of step region with side lines provides the position for the subsequent step. Right: If no intersection with the side line is found, we instead use the intersection between the boundary of step region and the line connecting the zero point and the closest point to it on the side line.

In order to calculate positions of subsequent footsteps, our system uses side lines situated at distances d_0 on both sides of the 2D path to the local goal, one for each foot. The next step position is then found by checking for intersections P_i of the stepping region with the according side line and choosing the one that has the minimum distance to the end of the current line segment. In addition, the rotation of the step is aligned with the current segment, while the maximum rotation is limited by the rotation capabilities of the humanoid and possible overlaps to the previous support foot. This is depicted in Fig. 6b. Even though the safety margin around edges leads to collision free footstep plans in most cases, it is possible that a slight overlap between steps and edges occur. We thus check every footstep for collisions and adjust it according to the stepping capabilities of the robot. Note that further constraints, e.g., a different step region depending on the foot rotation can easily be used.

If no intersection between the side line and the boundary of the step region exists, we instead use the intersection of the line connecting the zero point and the point closest to it on the side line as shown in Fig. 6c. This situation only occurs at nodes with a large angular change between the line segments.

Once either step reaches the end of the side line segment, one final footstep is added to the current segment but rotated half-way according to the direction of the following one which is then used for the subsequent steps.

V. EXPERIMENTS

We thoroughly evaluated our approach on multiple scenarios in both simulated and real-world environments. The experiments and performance tests were done using an ASUS Xtion Pro Live, mounted onto the head of a Nao robot, and an Intel Core i7 3770. The resolution of the camera was 640x480 and images were acquired at 30 fps. A single height map contained 200x200 grid cells and covered a 60x60 cm^2 area at a distance of 20 cm in front of the robot, leading to a resolution of about 3x3 mm^2 per grid cell. The combined local map contained 300x300 cells with a resolution of 4x4 mm^2 and extended 60 cm to the left and right of the robot and ranged from -30 cm behind up to 90 cm in front of the robot. For simulation, we used maps with randomly

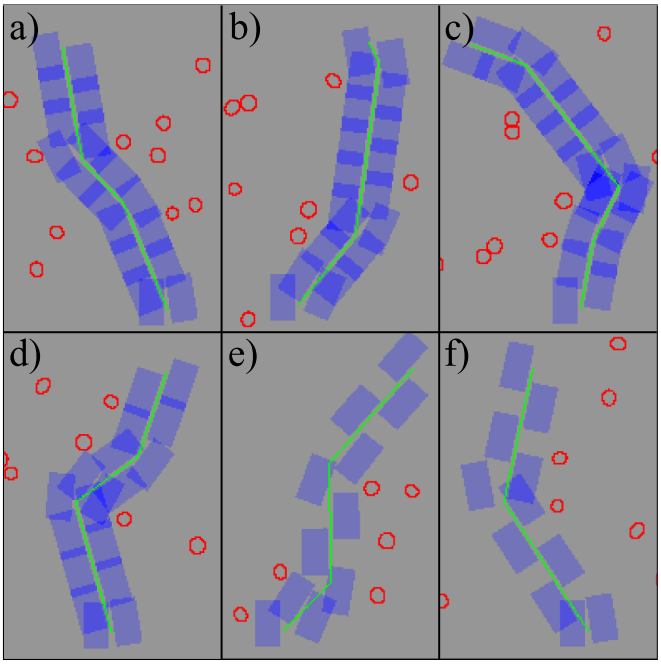

Fig. 7: Example 2D paths with footsteps calculated by our approach. We sampled a random numbers of objects in the local map. Images a-d show plans for the step distances for the Nao robot. In Figs. e and f we have increased the step distances to show resulting paths for a robot with a larger step length. All images are cropped to the regions that contain the corresponding 2D path.

sampled objects to show the performance of our footstep planning approach. The start and goal points were always located at the mid bottom and at random points at the top of the map and we omitted those maps where the goal point was unreachable. Fig. 7 shows examples of maps along with 2D paths and footsteps calculated by our system.

A. *Performance Evaluation*

We first evaluated the performance of our mapping approach. The test started after the robot had already taken seven steps such that a total of eight maps were combined for the local map. In the experiments, several objects were constantly in the field of view. We summarize the results in Table I.

Task	Avg. Time
Convert image to height map.	$6.14 \pm 2.9\ ms$
Find planar regions and create segmented map.	$2.11 \pm 1.12\ ms$
Detect edges and create edge map.	$0.20 \pm 0.12\ ms$
Combine available maps.	$4.25 \pm 1.78\ ms$

TABLE I: Performance results. The timings are averaged over 1000 iterations and the errors define one standard deviation. We combined a total of eight maps for the local map.

We then evaluated run times for footstep planning depending on the amount of segments of the 2D path and run times of the calculation and collision checks for different numbers of footsteps. All tests were done using randomly sampled maps comparable to those in Fig. 7. The results are presented in Figs. 8 and 9.

Additionally, we conducted experiments in which we randomly sampled objects and evaluated the run times with

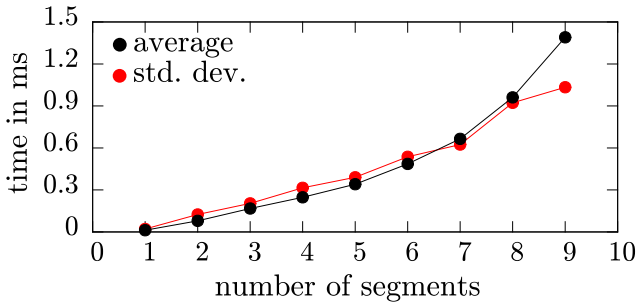

Fig. 8: Run times depending on the number of segments of the final path averaged over 1000 iterations.

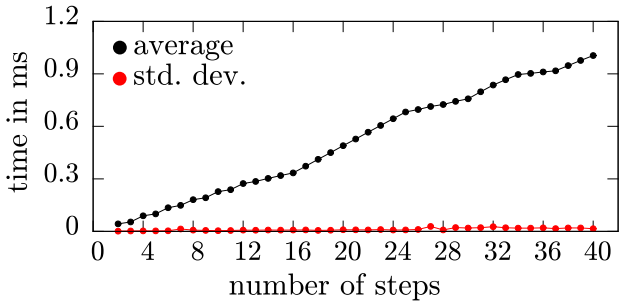

Fig. 9: Run times for footstep step calculation. We used five different sufficiently long paths and averaged over 5000 iterations per step count.

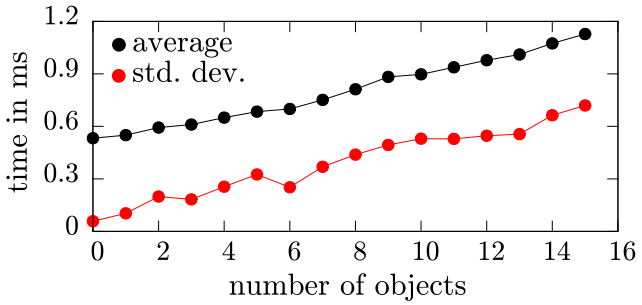

Fig. 10: Run times depending on the number of randomly sampled objects averaged over 1000 iterations.

respect to the number of objects contained in the local map. The corresponding results are presented in Fig. 10.

B. *Real-World Experiments with Dynamic Changes*

To demonstrate the real-time capabilities of our framework, we conducted an experiment by throwing a ball in the path of the robot. The resulting footstep paths at different points in time are depicted in the sequence of images in Fig. 11. As can be seen, our approach instantly generates collision-free paths.

C. *Qualitative Experiments with the HRP-4 and Nao*

In order to show the practical applicability of our approach, we have ported the system to run onboard a real Nao robot at a stable rate of 10 Hz. Furthermore, we have tested our system both on a HRP-4, as presented in Fig. 12.

VI. DISCUSSION AND FUTURE WORK

While the adaptations to dynamic changes in the environment are very fast and footstep plans are provided

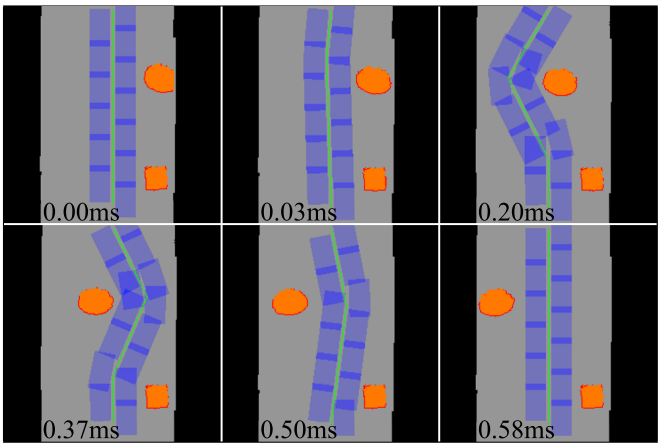

Fig. 11: Ball rolling in front of the robot. When a ball is rolled in the path of the robot, our framework allows a reaction in real time.

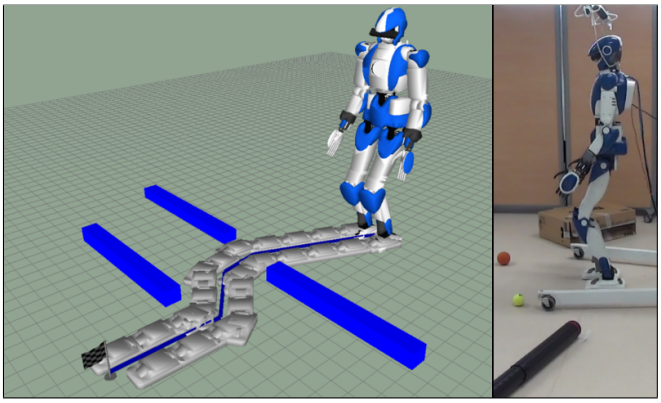

Fig. 12: Experiments using a HRP-4 both in simulation and in the real world.

near-instantaneously, we would like to address potential limitations to the system. Compared to other search algorithms, e.g., based on $A*$, our planner does not seek optimal solutions. While paths with only few line segments result in near-optimal paths, the possibility to end up with non-optimal solutions increases with the complexity of the planning problem. Thus, we favor run time over optimality, however, as shown, our approach yields reasonably short solutions also for planning problems with increased complexity.

Additionally, our search is currently limited to 2D as collisions are only checked on ground level. As a next step, we are working on extending our technique to searches on full 2.5D maps, taking into account the possibilities of stepping over obstacles and on top of further planar regions.

VII. CONCLUSION

In this paper, we have presented a fully integrated system that is capable of analyzing 3D point cloud data and recalculating a local 2D path including footstep locations in only 18 ms on average. Taking advantages of grid-based height maps and 2.5D plane segmentation, our approach can almost instantly react to changes in the vicinity of the robot while following a 2D path. Since the analysis of the local environment only takes a few milliseconds, our technique is specifically useful for dynamic walking, which requires fast adaptations to changes in the local environment, as suddenly stopping the motion of the robot is not possible.

Moreover, we ported the system to run onboard a Nao robot at a stable rate of 10 Hz and have also done first experiments with an HRP-4.

ACKNOWLEDGEMENTS

We would like to thank Don Joven Agravante and Andrea Cherubini, both working at LIRMM, France, for the help and support with the HRP-4 experiments.

REFERENCES

[1] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami, "Planning biped navigation strategies in complex environments," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2003.

[2] A. Hornung and M. Bennewitz, "Adaptive level-of-detail planning for efficient humanoid navigation," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[3] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2012.

[4] D. Maier, C. Lutz, and M. Bennewitz, "Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2013.

[5] A. Stumpf, S. Kohlbrecher, D. Conner, and O. von Stryk, "Supervised footstep planning for humanoid robots in rough terrain tasks using a black box walking controller," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2014.

[6] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*, ser. Springer Tracts in Advanced Robotics, H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, Eds. Springer International Publishing, 2015, vol. 107.

[7] R. Hulik, V. Beran, M. Spanel, P. Krsek, and P. Smrz, "Fast and accurate plane segmentation in depth maps for indoor scenes," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2012.

[8] M. F. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake, "Continuous humanoid locomotion over uneven terrain using stereo fusion," under review, 2015. [Online]. Available: http://groups.csail.mit.edu/locomotion/pubs.html

[9] D. Holz, S. Holzer, R. Rusu, and S. Behnke, "Real-time plane segmentation using RGB-D cameras," in *RoboCup 2011: Robot Soccer World Cup XV*, ser. Lecture Notes in Computer Science, T. Röfer, N. Mayer, J. Savage, and U. Saranl, Eds. Springer Berlin Heidelberg, 2012, vol. 7416.

[10] A. J. Trevor, S. Gedikli, R. B. Rusu, and H. I. Christensen, "Efficient organized point cloud segmentation with connected components," *Semantic Perception Mapping and Exploration (SPME)*, 2013.

[11] J. Chestnutt, K. Nishiwaki, J. Kuffner, and S. Kagami, "An adaptive action model for legged navigation planning," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2007.

[12] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2005.

[13] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2014.

[14] L. Baudouin, N. Perrin, T. Moulard, F. Lamiraux, O. Stasse, and E. Yoshida, "Real-time Replanning Using 3D Environment for Humanoid Robot," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, Bled, Slovenia, 2011.

[15] C. Weber, S. Hahmann, and H. Hagen, "Methods for feature detection in point clouds," in *Visualization of Large and Unstructured Data Sets-Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)*, vol. 19, 2011.

[16] C. Choi, A. J. Trevor, and H. I. Christensen, "RGB-D edge detection and edge-based registration," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2013.