Prediction Maps for Real-Time 3D Footstep Planning in Dynamic Environments

Philipp Karkowski Maren Bennewitz

***Abstract*— Perception of the local environment is a precondition for mobile robots to navigate safely in dynamic environments. Most robots, i.e., humanoids and smaller wheeled robots rely on planar regions. For humanoids, a simple 2D occupancy map as environment representation on which a path is planned is hereby not sufficient since they can step over and onto objects and therefore need height information. Considering dynamic obstacles introduces another level of complexity, since they can lead to necessary replanning or collisions at later stages. In this paper, we present a framework that first extracts planar regions in height maps and detects dynamic obstacles. Our system then uses this information to create a set of *prediction maps*, in which paths can be efficiently planned in real time at low CPU cost. We show in simulation and real-world experiments that our framework keeps run times well under $10\,ms$ for one computation cycle and allows for foresighted real-time 3D footstep planning.**

I. INTRODUCTION

In recent years, service robots have become more and more capable at helping humans even with complex tasks, e.g., tasks that present a high risk to humans or when it results in a decreased long-term cost to apply robots. These tasks may include coverage searches, e.g., in collapsed buildings or moving objects between destinations in order to assist humans with their task. Most current robots are still reliant upon planar regions in order to smoothly move along, in particular, humanoid robots which are highly delicate with regards to stability.

In addition to identifying static planar regions, it is advantageous to anticipate the development of the local environment in the presence of dynamic obstacles. In this way, obstacles that move in the way of a robot can be considered during path planning, which in turn can reduce the necessity to decelerate or stop and replan at later stages in order to avoid a collision.

A common way to represent information about the structure of the environment around the robot are grid-based height maps. While 2D occupancy grid maps are sufficient for most wheeled robots, other robots, such as humanoids, also need height information to consider stepping over or onto objects. In this paper, we present a novel approach to segment such height maps robustly into planar and non-planar regions even in the presence of noise. Furthermore, we track dynamic obstacles and approximate their movement to allow for foresighted path planning. One way to accomplish this, is to provide a static map along with a list of tracked dynamic objects that can be combined to create a map for any desired point in time. However, since path planning algorithms typically have to perform a vast number of collision checks, this would require frequent mappings of the dynamic obstacles onto the static map. We, therefore, precompute a set of *prediction maps* that can be accessed directly in order to speed up path planning.

All authors are with the Humanoid Robots Lab, University of Bonn, Germany. This work has been supported by the European Commission under contract number FP7-610532-SQUIRREL.

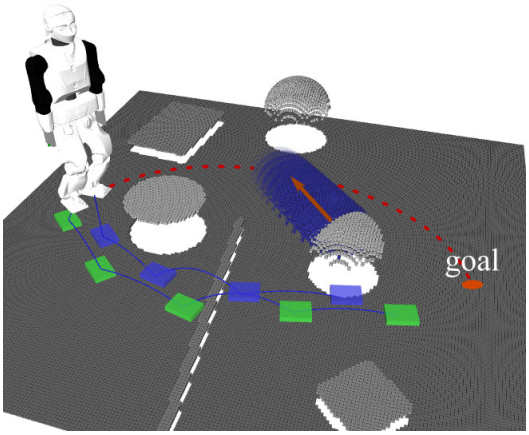

Fig. 1: We predict the future movements of dynamic objects to create a set of predicted height maps, which are segmented into planar regions. Those prediction maps can be used to find 3D footstep plans that anticipate possible collisions and avoid unnecessary replanning at later stages.

We put a high focus on the computational efficiency as it is our believe that navigation should ideally be a background task for any robot to allow it to perform other important tasks in parallel, especially for systems with constrained resources due to limited payload. Our system is capable of completing a full cycle, which includes preprocessing, segmentation, object detection, and, for example, computing a set of 20 prediction maps $10\,s$ into the future in only $6\,ms$ without the need of a graphics card.

We combined the framework with our real-time 3D footstep planner [1] and demonstrate its strengths in scenarios with dynamic objects where the prediction maps can help to avoid collisions or the necessity for replanning in the future. One such example can be seen in Fig. 1, where our footstep planner preemptively finds a higher cost path by stepping over an obstacle, because it anticipates a blocked path by the time the robot reaches the gap along the low cost path (red). To the best of our knowledge, our framework is the first that considers motion prediction of dynamic objects in 3D footstep planning.

II. RELATED WORK

Commonly used methods for image segmentation are the watershed transformation, superpixel, and plane fitting approaches. The watershed transformation operates on grey

scale images that are interpreted as height maps [2], [3]. The watershed transformation is not very efficient because of the iterative flooding step. Furthermore, this method tends to over-segmentation such that the regions have to be merged in an additional step. Superpixel algorithms follow the approach of grouping pixels into polygonal regions, which offers a compact representation of the image. Cigla and Alatan, for example, developed a method that is based on k-means clustering [4] and provides the final segmentation by applying normalized cuts to the graph generated out of the superpixels. A drawback here is that the number of regions as parameter for the clustering has to be known beforehand. Shen *et al.* provide a new clustering method using DBSCAN (density-based spatial clustering of applications with noise) [5]. DBSCAN is an iterative algorithm that clusters pixels using a distance measurement defined by color and spatial features. This approach can lead to over-segmentation so that small regions have to be merged in a post-processing step.

Other methods apply plane fitting to approximate a point cloud by a set of planes. Fallon *et al.* fuse stereo-images [6], while Jin *et al.* [7], Holz *et al.* [8], and Trevor *et al.* [9] use depth data from RGB-D cameras. Even though segmentation results in these approaches can be obtained quite fast, run times are usually still too high to allow for further processing steps and immediate reaction to dynamic changes in the environment. In our previous work [1], we developed a real-time capable plane fitting approach for 2D height maps that calculates a normal vector based on the cells in a local neighborhood and grows this estimated plane into neighboring cells. However, a drawback is the possible under-segmentation in certain cases, which is caused by the approximation of the normal vector.

For object detection and tracking, many approaches use difference images. The approach proposed by Asvadi *et al.* tracks the objects extracted from 3D LiDAR data by Kalman filters to predict their next states taking into account the velocity and trajectory [10]. Object association, however, is only performed based on gating strategies selecting eligible candidates and nearest neighbor properties, which could cause false matching. Furthermore, there exist object tracking methods that rely on stereo data. For example, Ess *et al.* developed a method based on the HOG (histogram of oriented gradients) feature descriptor to classify image regions, detect given objects of interest in street scenarios, and predict their motions [11]. Just recently, Osep *et al.* [12] presented a method for generic multi-object tracking that works for known as well as unknown objects in street scenes. Additionally, Wahrmann *et al.* presented an approach for modelling objects using swept sphere volumes and track them in unknown dynamic environments [13]. Note that in contrast to the discussed object tracking methods, the focus of our work lies on predicting their trajectories to construct prediction maps, which can then be used for grid-based path planning and collision checking in real time.

Finally, a variety of methods exist that compute the piecewise motion for RGB-D images or 3D LiDAR data to estimate the scence flow [14], [15], or semantically classify the image in static and dynamic parts [16]. The semantic segmentation is hereby rather expensive and depends on a prelearning step using labeled objects.

III. PLANAR REGION SEGMENTATION

Our segmentation method relies on a height map that can, for example, be acquired using a depth camera. The map is represented as a 2D grid with corresponding height information.

A. Normal Computation

The planar region segmentation is based on a region growing approach that considers normal vectors of neighboring grid cells. We first fit a plane $xn_x + yn_y + z + n_0 = 0$ for every grid cell using the height information of a 3×3 grid around the cell. The standard least squares fitting solution for the normal $[n_x, n_y, 1]$ is then given by solving a set of three linear equations, cf. [17]. As we are only interested in finding the normal for every cell and are not interested in the full plane equation, we shift the 3×3 grid patch such that $\sum_i z_i = 0$ and the central cell of each patch lies at $x = y = 0$. In that case the said equations simplify to

$$\begin{bmatrix} \sum_{i=0}^{N} x_i^2 & 0 & 0 \\ 0 & \sum_{i=0}^{N} y_i^2 & 0 \\ 0 & 0 & N \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_0 \end{bmatrix} = \begin{bmatrix} -\sum_{i=0}^{N} x_i z_i \\ -\sum_{i=0}^{N} y_i z_i \\ 0 \end{bmatrix}, \quad (1)$$

where the points x_i, y_i, z_i are the coordinates of the cells and N is the number of points, i.e., 9 in our case. The coordinates for x_i and y_i are given by the grid resolution c, such that $\sum_i x_i^2 = \sum_i y_i^2 = 6c^2$. Note the factor of 6 instead of 9, as three of the nine cells are 0 for both x and y, cf. Fig. 2. In addition, the sums over $x_i z_i$ and $y_i z_i$ are trivially computed, as both x_i and y_i only take the values c, 0, or $-c$. Using this information, it is possible to compute the normals very efficiently.

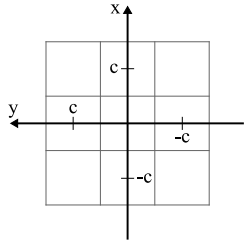

Fig. 2: Geometry of normal fitting. When we center every patch of 3×3 cells at the origin, the least squares solution to the plane fitting is highly simplified and can be computed very efficiently, cf. Eq. 1.

B. Segmentation of Planar and Non-Planar Regions

In order to find the segments, we use a region growing approach from a seed cell using the normal vectors of each cell. We first find all planar regions by examining each cell of the map and checking whether the normal vectors of its surrounding 4 cells point in a similar direction as its own normal vector. If that is the case, we use it as a seed for

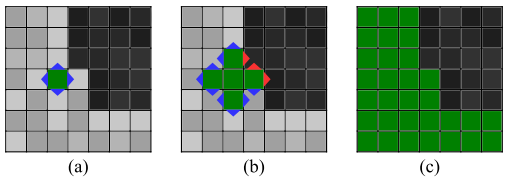

Fig. 3: Segmentation-based on region growing. The figure illustrates how a region (green) is grown into regions with normal vectors that point in similar directions, here illustrated with similar gray tones.

a region growing algorithm. This approach is more robust than our previous work [1], since the segmentation allows for slight curvatures in large regions, which are nearly planar at short distances. The algorithm starts at the seed cell and keeps adding neighboring cells if their normals point in a similar direction as the normals of their neighbors and if they have not been segmented yet (cf. Fig. 3).

We then segment all remaining connected regions and classify them as non-planar. The algorithm works similarly to the segmentation of the planar regions, but instead of checking whether neighboring normals point in similar directions, we simply add neighboring cells if they have not been segmented yet.

During the segmentation of planar regions, we concurrently perform a plane fitting algorithm and find the standard deviation of the distances to the fitted plane for the points that belong to the corresponding segment. Should that deviation be too high, we reclassify the segment as non-planar and, after completing the search for planar regions, keep growing the region as a non-planar region. This is helpful to detect large smooth objects whose normals have very similar neighboring normals, e.g., a large spherical object.

When using real data we also have to deal with shadows, which either arise due to a too low resolution of the camera's depth data or due to obstructed regions behind tall objects. Our framework uses a global map with height information for filling all shadow regions.

IV. OBJECT TRACKING

The previous section described how the height map of the local surrounding is segmented into planar and non-planar regions. These segments correspond to either stationary parts of the environment, e.g., the floor or non-moving obstacles, or also to dynamic objects. In our approach, we track those objects across multiple segmented frames and predict their future positions.

A. Object Tracking

We classify a segment as an object if it falls into a preset size range, i.e., we ignore large segments, e.g., the floor, or very small segments that may arise from high sensor noise.

For every segment that is classified as an object, we save its size, centroid, a list of its corresponding grid cells and the time stamp of its detection. In this way, we generate a list of objects for each segmented height map and maintain a maximum number f_{max} of these lists.

In order to track an object and predict its future locations, we need to associate objects from a new frame with those in the preceding frame. We associate the objects by first calculating assignment costs for any object o from the current and any object p from the previous frame. The cost matrix S is constructed by using a Gaussian distance function using the distance r_{op} between any two objects,

$$S_{op} = ae^{-kr_{op}^2}, \tag{2}$$

where a and k are constants to be determined experimentally.

We apply an algorithm similar to the *Hungarian Method* [18] to find the globally best assignment between the objects, given S. However, unlike the original Hungarian Method, we also include empty associations to account for disappearing and appearing objects, e.g., when an object enters or leaves the map area.

B. Object Motion Prediction

Since we perform data association at every time step, we maintain a sequence of positions for any tracked object with a sequence size anywhere between 1 and f_{max}. If an object has been tracked for a certain number of frames, we use the associated positions and time stamps to predict its future motion. If the position has not changed considerably over the course of the last frames, it is classified as *static*, while all other objects are classified as *dynamic*. The motion prediction of dynamic objects is split up into two parts:

1) As objects move along a straight line when no other external force act upon them, we first compute the movement direction $\hat{\boldsymbol{d}}_o$ of object o at the current time t_0 by applying a linear regression on their tracked positions.
2) The equation of kinetic friction states that a constant force acts upon a moving object opposite to the direction of motion. Additionally, if an object moves on a slope, gravity adds a constant positive or negative force along the motion direction. Combined, this results in a constant deceleration or acceleration such that the position of an object will follow a parabolic trajectory along a straight line. Thus, we perform a Newton fitting algorithm, which is well-suited for fitting a parabola to the expected parabolic trajectory.

This results in an equation of motion of the predicted future position $\boldsymbol{x}_o(t)$ at time t, given by

$$\boldsymbol{x}_o(t) = \boldsymbol{x}_o + \hat{\boldsymbol{d}}_o * (At^2 + Bt), \tag{3}$$

where $\boldsymbol{x}_o$ is the current position of object o, while A and B are the fitted parameters found by Newton's algorithm. Note that $\boldsymbol{x}_o$, $\hat{\boldsymbol{d}}_o$, A, and B, are all in principle time dependent, as the equation of motion is recomputed at every time step and the current position of any dynamic object changes over time. We have left the time dependency out for simplicity, such that Eq. 3 is valid only for the current time step.

We want to address a few issues that arise with this approach. If an object is moving with very little friction, e.g., a rolling ball, it almost follows a linear trajectory. Thus, determining the curvature of the parabola becomes error prone. Additionally, the exact position of objects cannot

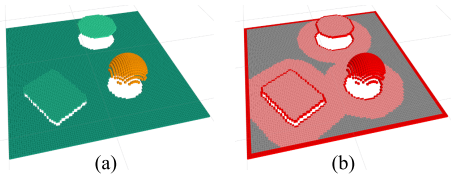

Fig. 4: Prediction maps for one time step. (a) Segmented height map into planar (turquoise) and non-planar (orange) regions. (b) Same map as (a), but displaying edges between adjacent segments (red) and an inflated region around the edges (light red).

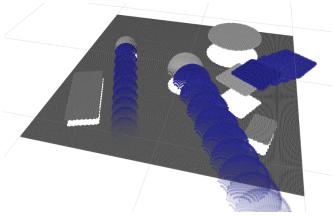

Fig. 5: Example scenario for run time evaluation. The height map has a size of 200×200 grid cells and contains 3 static and 3 dynamic objects. The predicted positions of the dynamic objects are projected onto the height map (dark blue) in $0.3\,s$ steps $5\,s$ into the future.

be determined from a grid map, especially in real world experiments, which, in turn, can lead to an inaccurate prediction, e.g., predict an acceleration instead of deceleration. Therefore, we additionally fit a linear trajectory and, if the associated error is small enough, use that predicted motion instead. This avoids too large prediction deviations due to a false prediction of the trajectory curvature.

Additionally, if an object follows a decelerating trajectory, we determine the predicted stopping point in time and replace the parabolic trajectory with the stopping position, so the motion of an object is not predicted to move backwards after stopping.

Note that our framework performs the motion prediction in the global frame, e.g., with the help of a localizer.

V. PREDICTION MAPS

While it is possible that a planning algorithm makes use of a static map along with a set of dynamic objects, many approaches to path planning, e.g., A*, perform a vast number of collision checks. Hence, to allow for efficient planning, it is of advantage to have a set of precomputed maps, each for a different future point in time.

As we need to project the dynamic objects onto their predicted future positions, this would leave the region they currently occupy with a shadow. Thus, we extract all dynamic objects from the height map and fill the region with height information from a global map. This is done similarly to filling the shadows as described in Sec. III-B.

We then compute a set of prediction maps for multiple future time steps. In particular for each time step, our system provides:

1) A grid map map that contains the height information.
2) A grid map indicating which cell has been classified as planar and non-planar
3) A grid map that contains information of edges between different neighboring segments along with an inflation radius around each edge. This map can be used for fast collision checks.

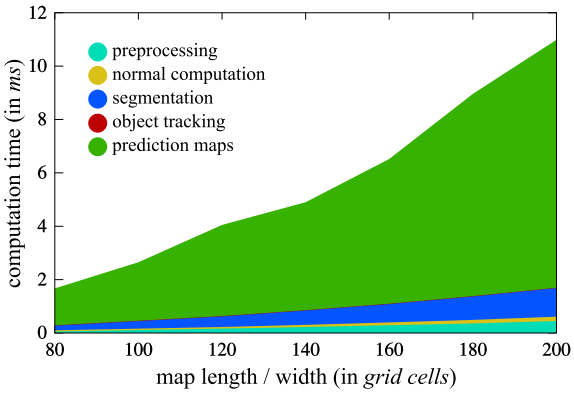

Fig. 6: Computation times for a full computation cycle. The cumulative times are shown for separate computation steps, explained in detail in Sec. III, Sec. IV, and Sec. V for the scenario seen in Fig. 5. Times are averaged over 5000 runs for each map size. The object tracking step is barely visible between computing the prediction maps and the segmentation.

An example set of maps for a single time step can be seen in Fig. 4.

While most 2D planning algorithms only need to know the planar regions and the inflation map, other algorithms, e.g., footstep planners, also need the height information to include stepping over and onto obstacles.

VI. EXPERIMENTS

We performed an extensive set of experiments to highlight the strengths and evaluate different aspects of our approach. Most experiments were performed within a simulation framework to determine more accurate estimates of computation times, object prediction accuracies, and robustness to noise. For the real-world experiment, we used the RGB-D camera *ASUS Xtion Pro Live* to record depth images. All experiments were done on an *Intel Core i7 4770* processor. In all cases, we computed 20 prediction maps for $10s$ into the future in $0.5s$ steps.

A. Preprocessing Depth Data

While the simulation framework directly creates height maps, the depth images from the RGB-D camera first need to be mapped onto the height map, given a 6D transform from the camera origin to the map origin, e.g., given by the robot pose from localization along with the transform tree of the robot. For mapping we use the highest point that falls into each height map grid cell. The computation time to convert the depth image took $2.25\,ms$ on average and is independent of the size of the height map.

An issue that arises with real-world data is noise on the height information. Therefore, we implemented an efficient median filter based on the work by Waltz *et al.* [19] for 3×3 height map grid patches. An average filter is of disadvantage for our approach as it does not conserve sharp edges which is necessary for the proper functioning of the segmentation step.

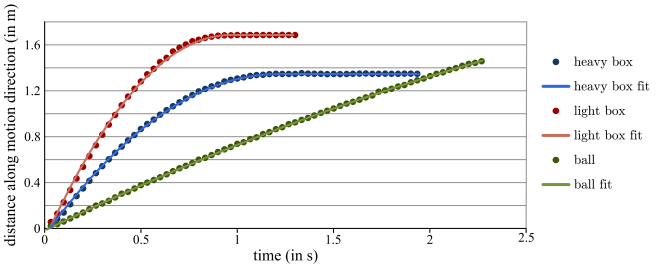

Fig. 7: Model accuracy. We recorded the trajectory of three different real objects and fit our model to the entire dataset or until the objects stopped moving. As can be seen, our model provides a highly accurate fit.

B. Computation Times

First, we show the average computation time of an example scenario that contains 6 objects of which 3 are dynamic as can be seen in Fig. 5. The run times are shown for a squared map with increasing map resolutions and are split into the following computational steps:

1) Preprocessing, which includes initializations and running the median filter, see Sec. VI-A.
2) Computation of normals, detailed in Sec. III-A.
3) Segmenting the map into planar and non-planar regions (incl. filling shadow regions), cf. Sec. III-B.
4) Detecting objects, mapping them onto objects from the previous frame, and computing the predicted trajectories of the dynamic objects, described in Sec. IV.
5) Computing the 20 prediction maps, i.e., height map, segmented map, and map with inflated edges, explained in Sec. V.

The results in Fig. 6 show that our framework is real-time capable even at high frame rates of $30\,fps$ while still not requiring the full CPU capacity.

C. Prediction Accuracy for Real Objects

In order to validate our prediction model (cf. Sec. IV), we recorded the trajectories of objects with different properties, moving on a hard carpet flooring. First, we fitted our parabolic model to the entire observed time frame. Fig. 7 demonstrates for three objects (a heavy box, a light box, and a ball, with sizes between $0.2 - 0.3m$) that our model provides a nearly perfect fit.

Furthermore, to test the prediction capabilities, we used a time frame of $0.3s$ for the Newton fitting algorithm and then predict how the motion is expected to continue. We tried to keep the time frame as short as possible, while still providing good predictions. This was done for three different sections along the trajectories of the boxes as seen in Fig. 8. One can see, that our model is capable at predicting the future motion of the objects very well. Only for the light box at the first prediction interval $[0.1s; 0.4s]$ a deviation of $0.3m$ can be observed, which is due to the difficulty of correctly computing the curvature of the parabola (cf. Sec. IV-B).

To demonstrate the issue that arises from fitting a parabolic trajectory to a near linearly moving object, Fig. 9 shows how fitting only the parabolic model to the trajectory of the ball leads to inaccurate predictions (left). If we instead use our combined model that uses a linear motion prediction for small errors in the fitting algorithm, we achieve a more accurate prediction (right).

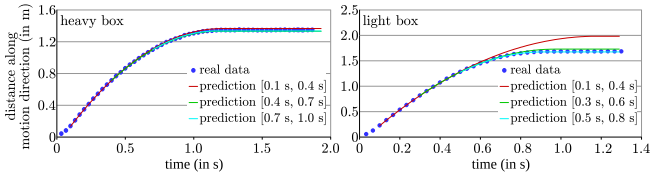

Fig. 8: Motion prediction. We use data from three subsequent $0.3s$ time frames to predict the future motion of two boxes with different properties. As can be seen, even such a short time frame can accurately determine the stopping positions and times.

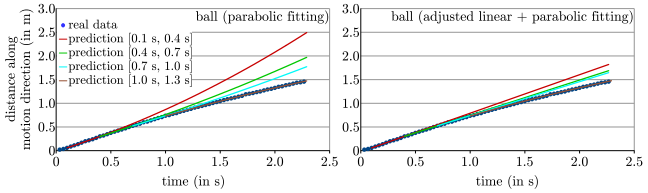

Fig. 9: Adjusted linear motion prediction. The left figure shows how only using the parabolic motion prediction can lead to and inaccurate prediction for objects that follow a near-linear path (constant speed). Our adjusted model that uses a linear motion model, if the error of a linear fit is small results in a highly improved motion prediction, as seen on the right.

D. Robustness to Noise

While simulated environments are in general noise-free, real-world cameras always introduce a certain level of noise. In order to test the segmentation robustness to noise, we added increasing levels of Gaussian noise to the same scenario as in Fig. 5. In this way, we found the maximum standard deviation of the Gaussian noise filter at $1.5cm$ that still allowed us to robustly segment 10000 height maps from multiple scenarios.

E. Planning Examples in Simulated Environments

Fig. 10 shows one example of the raw height map with the Gaussian noise filter and the filtered, segmented map.

In order to demonstrate the applicability of our framework, we combined it with a state-of-the-art real-time footstep planner [1]. We use empirically determined times for executing the footsteps to choose which predicted height map should be used for collision checking during the footstep planning. It should be noted that exact times are not necessary, as we do not consider the exact execution of the entire footstep plan, which can be recomputed in real time. Instead, we aim at foresightedly detecting areas that may become impassible by the time the robot reaches them.

As can be seen in Fig. 11a, a free path exists through the point A, which avoids high cost steps over other obstacles. However, our framework predicts a collision at point A by the time the humanoid would reach it and, thus, preemptively finds a footstep plan through point B that avoids the collision.

In Fig. 12a we can see a ball moving right toward the current position of the robot. The lowest cost path at this time would be straight ahead toward the goal by crossing

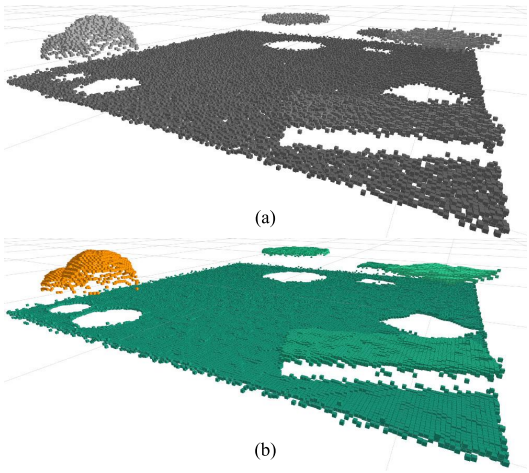

Fig. 10: Segmentation robustness against noise. The figure shows the same scenario as in Fig. 5, however, after applying a Gaussian noise filter. (a) shows the height map with the maximum level of noise that still allowed our approach to fully segment the scene into planar and non-planar regions. The filtered, segmented map is shown in (b).

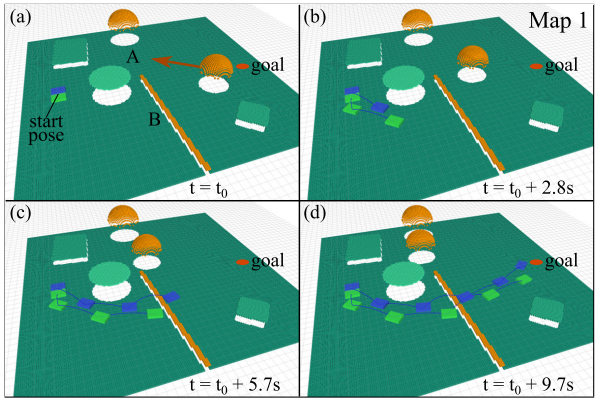

Fig. 11: Footstep planning in a dynamic environment. The motion of the tracked dynamic object is indicated with the arrow in (a). The current state of the environment at $t = t_0$ is shown in (a). (b)-(d) show different time steps of the footstep plan along with the predicted height map at the point when the robot is expected to reach the respective footsteps. As can be seen, our system anticipates the motion of the ball and the resulting footstep plan lets the robot step over an obstacle instead to reach its goal.

point A. Our framework, however, predicts the future motion of the ball and determines that a collision with the moving object would occur. Instead, a free plan to the goal is found via point B by stepping on top of the flat object, which additionally avoids a collision with the moving object.

F. Real-World Planning

Finally, we performed an experiment with real depth data from an RGB-D camera, positioned at a height of $1.65m$, equivalent to the height of *REEM-C* by *PAL Robotics*. In the experiment we have positioned a few objects in the planning scene, while still allowing the possibility of finding an almost direct footstep path to the goal; the scenario is shown in Fig. 13. Fig. 13b shows the current state of the environment, where the tracked dynamic object is displayed with blue projections that indicate its predicted future positions. While a valid footstep plan could be found straight to the goal, our framework instead finds a path around

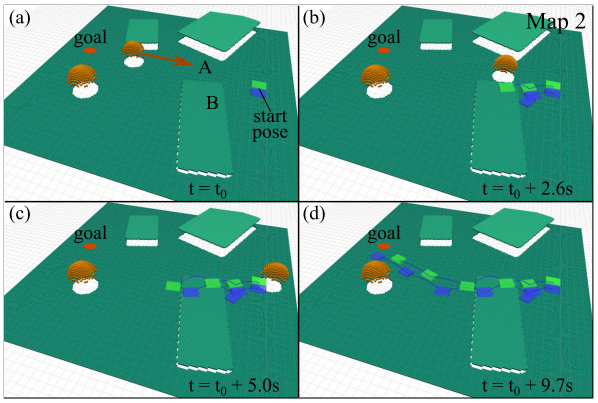

Fig. 12: Footstep planning in a dynamic environment. The motion of the tracked dynamic object is indicated with the arrow in (a). The current state of the environment at $t = t_0$ is shown in (a). (b)-(d) show different time steps of the footstep plan along with the predicted height map at the point when the robot is expected to reach the respective footsteps. As can be seen, to avoid the collision with a moving object the footstep plan lets the robot step onto an obstacle.

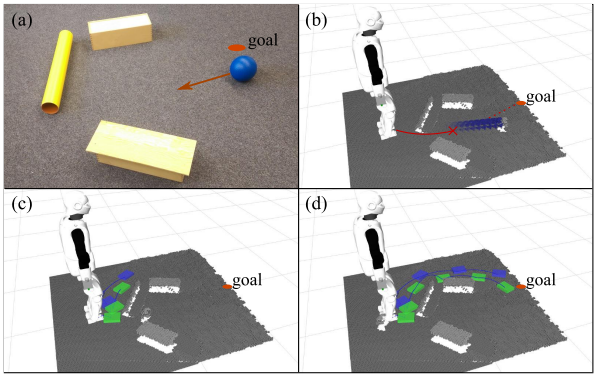

Fig. 13: Real world experiment. (a) Scenario with several objects. The movement of the dynamic object is indicated by the arrow. (b) Height map representation of the environment. The predicted future positions of the moving object are shown as blue projections. (c) The footstep plan leads around the objects in order to avoid a collision with the ball. (d) The full footstep path along with the predicted environment by the time the robot is expected to reach the goal.

the obstacles, as we predict the direct path would lead to collisions with the dynamic object after a few footsteps. The final collision free footstep plan is shown in Fig. 13d, while the possible direct path to the goal that would result by using only the static environment is indicated in red in Fig. 13b.

VII. CONCLUSION

In conclusion, we presented a framework that is capable of segmenting height maps of the local surrounding robustly into planar and non-planar regions. Moreover, our system detects dynamic objects and predicts their motion reliably into the future. This allows our framework to foresightedly detect and avoid areas that may become impassible for the humanoid by the time it reaches them.

Since our approach is not only real-time capable, but additionally keeps the CPU usage very low, even at a high update frequency of $30\,fps$, it allows the robot to perform calculations for other tasks in parallel to navigate reliably across the environment.

REFERENCES

[1] P. Karkowski, S. Oswald, and M. Bennewitz, "Real-time footstep planning in 3d environments," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2016.

[2] 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.

[3] G. Peters and J. Kerdels, "Image segmentation based on height maps," in *Computer Analysis of Images and Patterns*, ser. Lecture Notes in Computer Science, W. G. Kropatsch, M. Kampel, and A. Hanbury, Eds., 2007, vol. 4673.

[4] C. Cigla and A. A. Alatan, "Efficient graph-based image segmentation via speeded-up turbo pixels," in *IEEE International Conference on Image Processing*, 2010.

[5] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, "Real-time superpixel segmentation by dbscan clustering algorithm," *IEEE transactions on image processing*, 2016.

[6] M. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake, "Continuous humanoid locomotion over uneven terrain using stereo fusion," in *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2015.

[7] K. Jin, P. Liu, R. Sun, Z. Wei, and Z. Zhou, "Real-time plane segmentation in a ros-based navigation system for the visually impaired," in *Int. Conf. on Ubiquitous Positioning, Indoor Navigation and Location Based Services (UPINLBS)*, 2016.

[8] 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.

[9] 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.

[10] A. Asvadi, P. Peixoto, and U. Nunes, "Detection and tracking of moving objects using 2.5d motion grids," in *Int. Conf. on Intelligent Transportation Systems*, 2015.

[11] A. Ess, K. Schindler, B. Leibe, and L. van Gool, "Object detection and tracking for autonomous navigation in dynamic environments," *Int. Journal of Robotics Research (IJRR)*, vol. 29, no. 14, 2010.

[12] A. Osep, W. Mehner, P. Voigtlaender, and B. Leibe, "Track, then decide: Category-agnostic vision-based multi-object tracking," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[13] D. Wahrmann, A.-C. Hildebrandt, T. Bates, R. Wittmann, F. Sygulla, P. Seiwald, and D. Rixen, "Vision-based 3d modeling of unknown dynamic environments for real-time humanoid navigation," in *Int. Journal of Humanoid Robotics*, 2019.

[14] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers, "A primal-dual framework for real-time dense RGB-D scene flow," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[15] R. Newcombe, D. Fox, and S. Seitz., "Dynamicfusion: Re-construction and tracking of non-rigid scenes in real-time," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] A. Dewan, G. Oliveira, and W. Burgard, "Deep semantic classification for 3D LiDAR data," in *IEEE/RSJ Int. Conf. on Int. Robots and Systems (IROS)*, 2017.

[17] B. Muralikrishnan and J. Raja, *Computational Surface and Roundness Metrology*. Springer London, 2009.

[18] H. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1, pp. 83–97, 1955.

[19] F. Waltz, R. Hack, and B. Batchelor, "Fast efficient algorithms for 3x3 ranked filters using finite-state machines," in *Proc. SPIE 3521, Machine Vision Systems for Inspection and Metrology VII*, vol. 3521, 1998.