Polygonal Perception for Mobile Robots

Marcell Missura Arindam Roychoudhury Maren Bennewitz

***Abstract*— Geometric primitives are a compact and versatile representation of the environment and the objects within. From a motion planning perspective, the geometric structure can be leveraged in order to implement potentially faster and smoother motion control algorithms than it has been possible with grid-based occupancy maps so far. In this paper, we introduce a novel perception pipeline that efficiently processes the point cloud obtained from an RGB-D sensor in order to produce a floor-projected 2D map in the field-of-view of the robot where obstacles are represented as polygons rather than cells. These polygons can then be processed by path planning algorithms and obstacle avoidance controllers. Our pipeline includes a ground floor plane detector that performs significantly faster than other contemporary solutions and a grid segmentation algorithm that uses image processing techniques to identify the contours of obstacles in order to convert them to polygons. We demonstrate the performance of our approach in experiments with a wheeled and a humanoid robot and show that our polygonal perception pipeline works robustly even in the presence of the disturbances caused by the shaking of a walking robot.**

I. Introduction

Robots perceive the world through sensors such as cameras, laser rangers, and RGB-D sensors that allow them to determine the location and shape of objects in their environment. Within the scope of mobile navigation, the perceived data is processed primarily to create a map that contains impassable regions the robot must not enter. Then, the map can be used to plan the motion of the robot from its current position to a given goal, without traversing any of the blocked regions. Current research has converged to represent such maps as occupancy grids where each cell of the two or three-dimensional grid represents a small spatial unit. The cells usually hold a floating point value that indicates the probability of the cell being occupied. Based solely on the grid representation, simultaneous localization and mapping (SLAM), path planning, and obstacle avoidance algorithms have matured to a state where robots are able to move about in realistic office environments [1].

The conversion of spatial information that arrives in form of a point cloud to a grid structure is a reasonable first step as it considerably reduces the amount of data and augments it with neighbourhood information. For example, it is relatively easy to identify clusters of points once they have been sorted into a grid. However, we argue that postprocessing the grid structure to polygons—a geometric representation that holds more semantic information in less amount of data—is beneficial for mobile robot navigation. In path planning, the polygonal structure can be exploited to compute a smoother path in a shorter amount of time than by using an A* search in a grid [2]. Collision avoidance approaches can also benefit from a geometric representation [3] [4], also in 3D [5] [6].

All authors are with Humanoid Robots Lab, University of Bonn, Germany

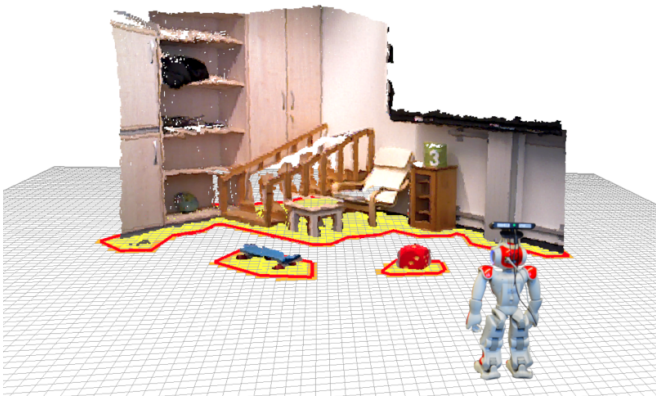


Fig. 1: Polygonal perception of a scene using the RGB-D camera mounted on a Nao's head. The point cloud observation is converted to a 2D polygonal representation (red) that marks the boundaries of blocked regions.

In a nutshell, in order to convert depth data to polygons, the first step of our pipeline is to remove the points that belong to the floor plane. This is crucial since otherwise the floor would be seen as an obstacle. The remaining points are sorted into an occupancy map, which is a 2D grid structure that stores a binary value in each cell that indicates whether the cell contains at least one point, or not. The occupancy map is then segmented using a contour detection algorithm that enumerates the border cells of each segment in counter clockwise order. These chains of border cells are already polygons, but we process them further and reduce their complexity. Figure 1 illustrates an example where the point cloud data obtained from the RGB-D camera mounted on a Nao robot's head has been converted to a polygonal representation. As we show in the experimental evaluation, the additional computation time to elevate the grid model to polygons is only a small fraction of the time needed to sort the data points into the grid to begin with. To the best of our knowledge, we are the first to present a polygonal processing pipeline working online on a walking robot with realtime computational capabilities.

II. Related Work

Demyen *et al.* [7] introduced a path finder that is based on triangulation of a polygonal map. Missura *et al.* [2] developed a shortest path search algorithm where only a necessary portion of the Visibility Graph [8] in a polygonal scene is built during the search. Kuindersma *et al.* [9], Griffin *et al.* [6], and Hildebrand *et al.* [10] implemented

footstep planning for humanoid robots where surafces are represented as polygons. [6] Missura *et al.* [4] augment polygons with a velocity vector and compute predictive collision avoidance with the Dynamic Window Approach. All of these works require a polygonal representation.

Baizid *et al.* [11] argue that occupancy maps have a prohibitively large requirement of resources when it comes to the use case of multi-agent exploration where maps have to be exchanged between robots over a network and merged on each individual robot. Bandwidth, memory, and the CPU are soon depleted when adding more robots to a team. They proposed a vector-based map representation where initially every point in the cloud is a vector in the map. Then, ϵ-close neighbours are connected with an edge, and edges are merged if they are almost colinear. The authors achieved a significant compression with respect to using an occupancy grid that was created on the same map.

Dichtl *et al.* [12] upgraded vector maps to polygonal maps where a closed polygon bounds the perimeter of the explored space. Two types of edges are distinguished—one type for edges that delimit the boundary of obstacles, and another type for edges that represent the boundary of the unexplored space. Inside the bounding polygon, smaller polygons delimit the blocked regions. The polygons are gained from an occupancy grid by outlining the borders of cells that represent the boundary of free space with vertical or horizontal vectors, and joining these vectors to polygons. Later on, the same authors realized a polygon-based SLAM system [13] where polygons are gained from a laser scan that is localized on a map using a point-to-vector ICP procedure. Here, the edges of polygons are now determined by connecting neighbouring laser points and merging nearly parallel lines. Our work includes grid processing as an intermediate step that makes our results applicable for RGB-D and Lidar (3D) sensors in addition to laser (2D). The grid processing also helps with the expansion of polygons by the radius of the robot.

Schnabel *et al.* [14] and Ochmann *et al.* [15] use the RANSAC algorithm to segment a large point cloud into geometrical templates including planes for reconstruction of buildings. Unfortunately, RANSAC is prone to report points that belong to unrelated, but coplanar surfaces. Wahrmann *et al.* [5], [16] also use RANSAC to identify planes, but split them into clusters later with the help of a 2D grid. The clusters are then converted to convex polygons with a fixed number of corners to represent walking surfaces. In our work, we use a normal-based region-growing method to first cluster points that lie on the same surface and then we fit planes into the clusters using only points that belong to the same cluster. Our polygons are non-convex and can approximate non-convex shapes such as the corners of a room. In addition to surfaces, Wahrmann *et al.* [5] process non-planar obstacles to swept-sphere volumes, which appear to be a great representation in 3D. We model obstacles as ground-projected polygons to explore the possibilities of a simpler 2D model.

Deits *et al.* [17] compute large polytopic and ellipsoid regions in a map as a representation of free space with a focus of performing a mixed-integer convex optimization for footstep planning [18]. The algorithm used is a mixture of two convex optimization programs that monotonically increase the size of a collision-free polytope around an initialization point and the inscribed ellipse within the polytope. It has been designed for offline processing with a runtime in the range of a few seconds, as stated by the authors.

As the identification of the floor plane is an essential part of our method, our work is closely related to planar segmentation of point clouds. Holz *et al.* developed the first real-time-capable algorithm that can segment a point cloud gained from an RGB-D sensor [19] [20] [21]. The method is based on the computation of surface normals from neighbouring pixels in the depth image and the identification of regions with similar normals. We adopted and improved upon this technique as explained in the following sections.

Karkowski *et al.* [22] segmented a height map into planar regions using region growing of surface normals for the purpose of footstep planning. The normals were computed from neighbouring cells in the height map instead of directly from the depth data. Here, planar regions and their boundaries are still encoded using groups of cells rather than polygons.

III. Floor Plane Detection

The identification of planar surfaces is a load-bearing concept when it comes to the segmentation of point clouds. The reason for this is that man-made environments are full of planes such as the floor, the walls, and tabletops. When the points reflected by such surfaces are removed from an RGB-D scan, the remaining points are much easier to identify as objects of interest. In our navigational scope, our only concern is the detection of the floor plane. When the floor is successfully removed, all remaining points represent obstacles the robot must not collide with. The challenge thereby is to remain computationally efficient, i. e., to be able to process the sensor data in real time. We are using an ASUS Xtion Pro Live RGB-D sensor that delivers a 640x480 depth image at a frame rate of 30 Hz to present the results of this work. Real time capability means processing these roughly 300K points in less than 33 milliseconds. Other cameras (e.g. Intel RealSense) and higher resolutions are possible.

In order to find the floor in a depth image, we compute surface normals using a mesh defined in image coordinates and add an optional pruning and vertical sorting step that helps with the recognition of the floor. We use region growing to cluster points with similar normals into contiguous surfaces and use a quick-to-compute similarity function to compare points efficiently. We apply an improved algorithm that exploits the sorted order of the points to identify the floor plane. A flow chart of the steps described in the following is presented in Figure 2.

A. Computation of Surface Normals

We assume an untransformed set of points P_c in the right hand coordinate frame of the camera where the x-axis points in the forward direction and the z-axis points up. We define

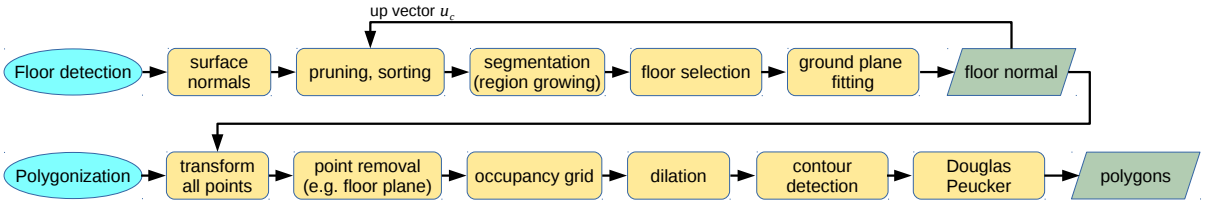


Fig. 2: The flow charts of our floor detection and polygonization pipelines.

a low-resolution grid

$$G_c = \{\boldsymbol{s}_{i,j} = (\boldsymbol{p}, \boldsymbol{n})_{i,j} \,|\boldsymbol{p}_{i,j} \in P_c, 0 \leq i, j < 32\} \quad (1)$$

of 32x32 samples $\boldsymbol{s}_{i,j}$ uniformly distributed in the pixel coordinate space of the RGB image, as shown in Figure 3(a), where $\boldsymbol{p}_{i,j} = (x, y, z)_{i,j} \in P_c$ is the 3D point corresponding to the grid coordinates (i, j) and $\boldsymbol{n}_{i,j}$ is the surface normal in this point. We compute the surface normals only for this small set of samples.

We determined the number of samples (32x32) experimentally. Using fewer samples results in the system starting to miss small floor patches. Using more samples, however, does not significantly improve the quality of the floor detection as most of the time there are more than sufficient floor samples in front of the robot. The runtime of the floor detection algorithm is linear in the number of samples and takes up only a small fraction of the total time needed to compute the entire pipeline. If more samples are needed for a certain use case, quadrupling or octuplicating the number of samples would result only in a small increase of the total runtime. Furthermore, a higher resolution of the sensor does not necessitate an increase of the number of samples. It is the opening angle of the camera that determines how many samples are needed for reliable floor detection.

In [20] and [21], neighbouring pixels are used for the computation of the surface normal of a point. This method is susceptible to the noise of the depth sensor and requires a considerable amount of smoothing. We simply compute the normals based on the neighbouring points in our low-resolution grid instead of the neighbouring pixels and do away entirely without smoothing as the increased distance between the points absorbs most of the sensor noise. Consequently, the normalized surface normal of a point $\boldsymbol{p}_{i,j}$ is given by

$$\boldsymbol{n}_{i,j} = -\frac{(\boldsymbol{p}_{i+1,j} - \boldsymbol{p}_{i-1,j}) \times (\boldsymbol{p}_{i,j+1} - \boldsymbol{p}_{i,j-1})}{|\,(\boldsymbol{p}_{i+1,j} - \boldsymbol{p}_{i-1,j}) \times (\boldsymbol{p}_{i,j+1} - \boldsymbol{p}_{i,j-1})\,|}. \quad (2)$$

B. Pruning and Sorting

We discard the samples whose normals are not approximately upright and are left with points that belong mostly to horizontal planes. This step is shown in Figure 3(b). We compute the pruned set

$$\tilde{G}_c = \{\boldsymbol{s}_{i,j} = (\boldsymbol{p}, \boldsymbol{n})_{i,j} \,|\boldsymbol{n}_{i,j} \cdot \boldsymbol{u}_c > \cos\left(\frac{\pi}{4}\right)\} \subseteq G_c \quad (3)$$

with the help of a scalar product with an up vector $\boldsymbol{u}_c$ that expresses the direction of the world z-axis in the camera frame. A rough knowledge of the vertical direction $\boldsymbol{u}_c$ is the only assumption we make in order to find the ground plane. Since ground vehicles and humanoid robots always have a cardinal upright direction, even using a constant $\boldsymbol{u}_c = (0, 0, 1)^T$ would result in a fair performance where the floor is recognized as long as the tilt angle of the sensor remains less than $\frac{\pi}{4}$ radians with respect to the world vertical. When a real robot is involved, typically a good estimate of the camera transformation M can be obtained using motor encoders and an IMU, and the up vector can be set to $\boldsymbol{u}_c = M^{-1}\,(0, 0, 1)^T$ to achieve better results. Our preferred method is to initialize the up vector with $\boldsymbol{u}_c = (0, 0, 1)^T$ and to allow a brief initialization phase where the depth camera is purposefully held in a roughly upright position. Once the floor plane has been successfully identified, i. e., at least one scan with a sufficient number of floor points has been seen, the determined floor normal $\boldsymbol{n}_F$ can be fed back as the up vector for the next frame $\boldsymbol{u}_{c_{t+1}} = \boldsymbol{n}_{F_t}$. This way, the floor detection algorithm becomes independent of the robot and can be run with a moving camera alone while the up vector is always maximally precise. We found this method superior to proprioception and this is the method we used in our experiments.

After pruning, we sort the remaining samples $\tilde{G}_c$ in vertically increasing order by the projection of their points onto the up vector, i. e.,

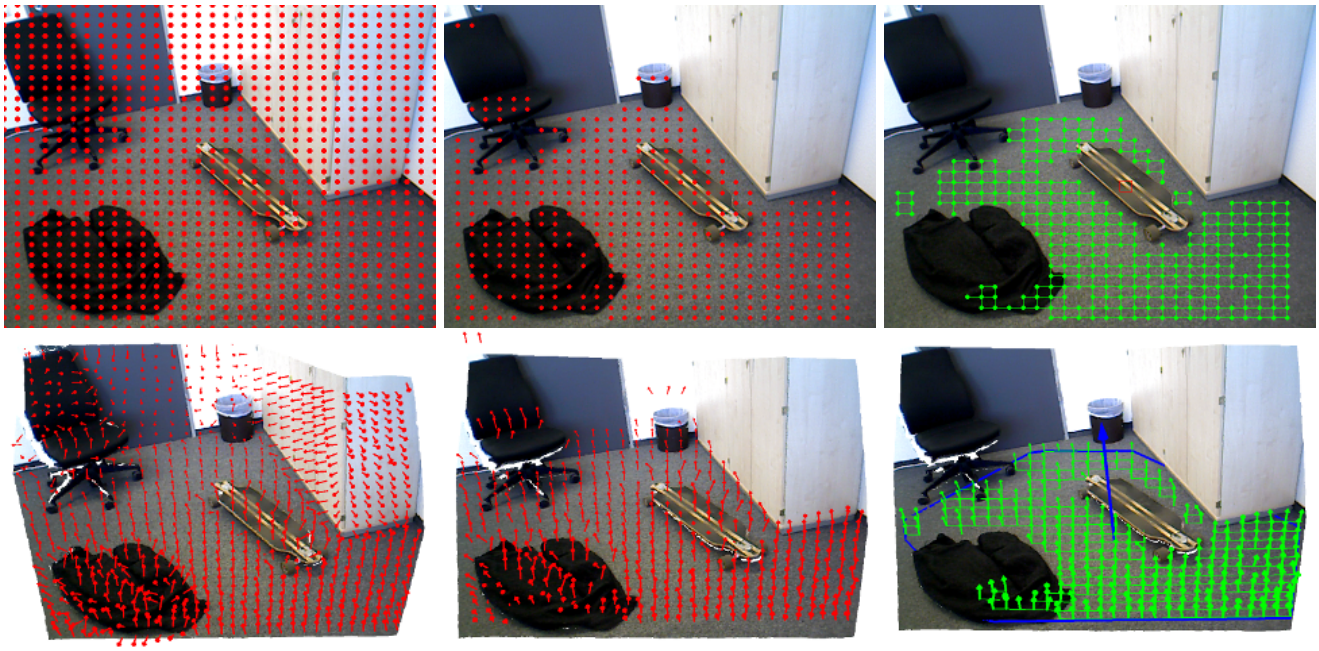
$$(\boldsymbol{s}_m > \boldsymbol{s}_n) \Leftrightarrow (\boldsymbol{p}_m \cdot \boldsymbol{u}_c > \boldsymbol{p}_n \cdot \boldsymbol{u}_c)\,. \quad (4)$$

This is equivalent to sorting the points by their z-coordinates in the world frame. Since we assume the floor plane to be the lowest plane in a scan, the sorting is helpful in a way that it moves the floor points we are looking for to the beginning of the $\tilde{G}_c$ set. Note that in order to extract all planes from a scan, the pruning and the sorting has to be omitted, and the up vector is not needed.

C. Segmentation

Using the pruned and sorted set $\tilde{G}_c$, we set the "in" flag $\sigma_{i,j} = 1$ for all samples in $\tilde{G}_c$, and initiate a recursive four-neighbour flood fill as shown in Algorithm 1 at the grid coordinates of all samples in the order they appear in $\tilde{G}_c$. One run of the flood fill algorithm collects all samples reachable from the start coordinates (i, j), neighbour by neighbour, as long as they are close to their direct neighbour according to the plane distance function

$$d(\boldsymbol{s}_m, \boldsymbol{s}_n) = |\boldsymbol{n}_m \cdot (\boldsymbol{p}_n - \boldsymbol{p}_m)\,| + |\boldsymbol{n}_n \cdot (\boldsymbol{p}_n - \boldsymbol{p}_m)\,| \quad (5)$$



(a) All Samples (b) Pruned Samples (c) Floor Samples

Fig. 3: Computation steps of our floor plane detection algorithm. The upper row shows color images, the bottom row shows depth images. a) The surface normals are computed for a low-resolution (32x32) grid of samples that have been selected in image coordinates. b) The samples are pruned such that only samples with roughly upright normals are kept in the set. c) The remaining samples are sorted vertically and clustered with a region-growing algorithm that joins direct neighbours when they seem to belong to the same plane according to our unique plane distance function (5). Then, the lowest cluster is selected as the floor set and used to fit a plane into its points. The normal of the detected floor plane and the convex hull of the floor points are indicated in blue. Note that the jacket and the skateboard appearing in the image are not seen as floor.

and adds them to a set of samples S that represents a segment of a planar surface. Neighbouring samples are considered to belong to the same plane when their distance $d(\boldsymbol{s}_m, \boldsymbol{s}_n)$ is smaller than a "low" threshold $\tau_l = 0.01$ as used in Algorithm 1 in lines 5, 7, 9, and 11. The distance function efficiently compares samples by their distance to the planes they represent and implicitly also by the angles of their normals with respect to each other. The distance is zero when two points have the same normal and lie in the same plane with respect to their normals, and monotonically increases with both kinds of errors—a mismatch of the normals or a distance between the planes. This allows us to cluster points that belong to the same surface in one sweep. Since the flood fills are started in a vertically sorted order, lower planes are prioritized to collect their samples first. The result of this procedure is shown in Figure 3(c).

Algorithm 1 FLOODFILL

Input: Grid coordinates i, j
Output: Segment of samples S

1: **if** $(\sigma_{i,j} = -1)$ **then**
2: **return**
3: $\sigma_{i,j} \leftarrow$ -1
4: $S \leftarrow S \cup \boldsymbol{s}_{i,j}$
5: **if** $(d(\boldsymbol{s}_{i+1,j}, \boldsymbol{s}_{i,j}) < \tau_l)$ **then** ▷ Eq. 5
6: FLOODFILL$(i+1, j)$
7: **if** $(d(\boldsymbol{s}_{i-1,j}, \boldsymbol{s}_{i,j}) < \tau_l)$ **then** ▷ Eq. 5
8: FLOODFILL$(i-1, j)$
9: **if** $(d(\boldsymbol{s}_{i,j+1}, \boldsymbol{s}_{i,j}) < \tau_l)$ **then** ▷ Eq. 5
10: FLOODFILL$(i, j+1)$
11: **if** $(d(\boldsymbol{s}_{i,j-1}, \boldsymbol{s}_{i,j}) < \tau_l)$ **then** ▷ Eq. 5
12: FLOODFILL$(i, j-1)$
13: **return** S

D. Floor Selection

The remaining task is to identify which of the found segments S belong to the floor, and to merge them in order to gather as many floor points as possible for a final fit. The floor plane selection procedure is described formally in Algorithm 2. We exploit the vertically sorted order and give segments that are found first priority over segments that are found later. Lower planes are in general better floor estimates than the largest planes as large planes can for example be table tops that dominate the scan. Let F denote the set of points that make up our floor hypothesis. We initialize F with the empty set and always accept the very first segment as our initial floor hypothesis, i. e., $F = S$ in line 9 of Algorithm 2. However, we attempt to merge the following segments with the floor plane hypothesis we have so far by computing the average representants $\tilde{\boldsymbol{s}}_F$ and $\tilde{\boldsymbol{s}}$—the averages of the points and the normals in the floor set F and segment S—, and comparing them with each other with our plane distance function in line 10. If the distance between the averages is lower than a "high" threshold, i. e., $d(\tilde{\boldsymbol{s}}_F, \tilde{\boldsymbol{s}}) < \tau_h = 0.1$, we

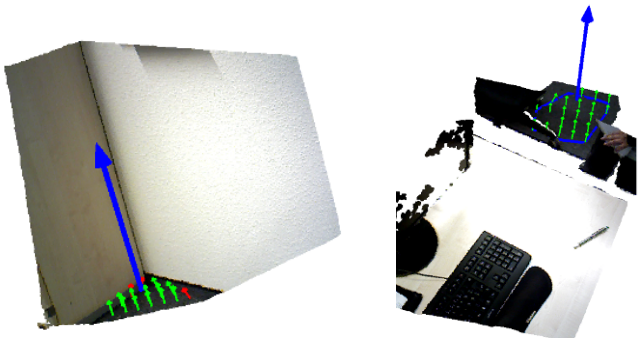
Algorithm 2 FLOORDETECTION

Input: Point cloud P_c
Output: Floor representant $\boldsymbol{s}_F = (\boldsymbol{p}_F, \boldsymbol{n}_F)$

1: Compute all normals $\boldsymbol{n}_{i,j}$ for sample set G_c ▷ Eq. 2
2: Compute pruned set $\tilde{G}_c$ ▷ Eq. 3
3: Sort the points in $\tilde{G}_c$ ▷ Eq. 4
4: $F \leftarrow \emptyset$
5: **for** $(\boldsymbol{s}_{i,j} \in \tilde{G}_c)$ **do**
6: $S \leftarrow$ FLOODFILL(i, j)
7: $\tilde{\boldsymbol{s}} = average(S)$
8: **if** $(F = \emptyset)$ **then**
9: $F \leftarrow S$, $\tilde{\boldsymbol{s}}_F \leftarrow \tilde{\boldsymbol{s}}$
10: **else if** $(d(\tilde{\boldsymbol{s}}_F, \tilde{\boldsymbol{s}}) < \tau_h)$ **then** ▷ Eq. 5
11: $F \leftarrow F \cup S$
12: $\tilde{\boldsymbol{s}}_F = average(\tilde{\boldsymbol{s}}_F, \tilde{\boldsymbol{s}})$
13: **else if** $(|F| * 20 < |S|)$ **then**
14: $F \leftarrow S$
15: $\tilde{\boldsymbol{s}}_F \leftarrow \tilde{\boldsymbol{s}}$
16: $\boldsymbol{s}_F \leftarrow planeFit(F)$
17: **return** $\boldsymbol{s}_F$

add the samples of segment S to the floor set F in line 11. If not, we ignore segment S and continue with the next one until all segments have been considered. Since our first hypothesis may well be an outlier, but then it is typically very small, we allow a segment found at a later time to replace the floor hypothesis in line 14, if its size in terms of number of contained samples is at least twenty times larger. This way, we make sure that outliers do not block the floor finding process by coming first, but we also do not simply settle for the largest plane, which could be a table top dominating a scan. When a lower segment contains at least 5% of all samples, it can no longer be replaced by a larger plane segment that appears later in the vertical order. An example of a critical case where the largest plane is a table top and only a small patch of floor is to be seen–and is correctly found–is shown in Figure 4(b).

Finally, we use the points in the floor set F for an ordinary least squares fit in line 16, and populate the floor representant $\boldsymbol{s}_F = (\boldsymbol{p}_F, \boldsymbol{n}_F)$ with the normal $\boldsymbol{n}_F$ of the fitted plane, and an arbitrary point $\boldsymbol{p}_F$ selected from the plane. Note that the surface normals are only used to select similar points, but not for the computation of the final fit of the floor plane.



(a) Close to a wall (b) Over a table top

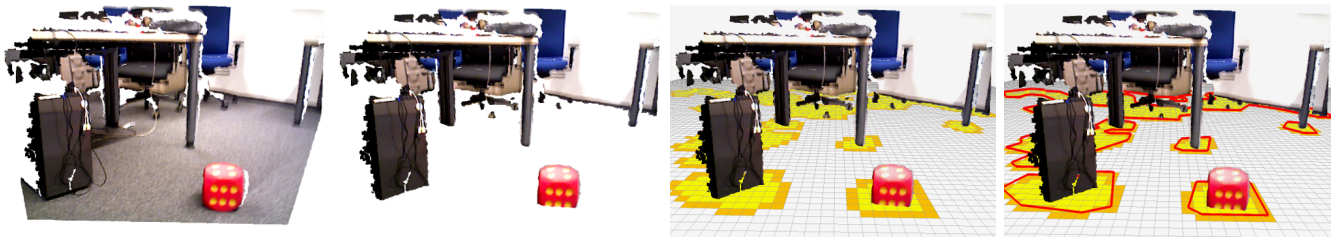
Fig. 4: Challenging situations. a) The RGB-D sensor is pointed at a wall and there are barely any floor points. The floor is still correctly detected. b) The camera is looking over a tabletop that dominates the scan. The floor is seen correctly only from a few points on the far side of the table.

Furthermore, we abstained from using the z-intercept to compare surfaces as the extrapolation of a surface segment over a large distance amplifies the noise. In [20], where the z-intercept is used to cluster surface segments to planes, a logarithmic scale was used to try and counteract this problem. Our result is robust and precise as can be seen in the accompanying video. Moreover, we did not need to assume an estimate of the position of the RGB-D sensor. Our only requirement is for the sensor to be roughly upright before the floor is found in the first scan.

IV. POLYGONIZATION

In the next step towards extracting polygons, we compute an occupancy map from the RGB-D scan. Using the output of the floor detection $\boldsymbol{s}_F = (\boldsymbol{p}_F, \boldsymbol{n}_F)$, we compute a homogeneous transformation matrix M that rectifies the point cloud such that the detected floor is aligned with the xy-plane. The axis-angle parameters of the rotation are given by $\boldsymbol{a} = \boldsymbol{n}_F \times (0,0,1)^T / |\boldsymbol{a}|$ and $\alpha = \arccos(\boldsymbol{n}_F \cdot (0,0,1)^T)$ where $\boldsymbol{a}$ and α are the axis and the angle of rotation, and $z = \boldsymbol{n}_F \cdot -\boldsymbol{p}_F$ is the vertical translation that sets the camera to the correct height relative to the plane. At this point, the position and the yaw angle (x, y, θ) of the robot with respect to a global map could also be integrated into M, if available from localization. Using M, we transform the point cloud from the camera frame into the world frame $P_w = MP_c$. Then, in a point removal step, we discard all points from P_w that have a height of less than 3 cm, most importantly including the points on the floor plane. We also discard points higher than a "ceiling" threshold, e. g., 60 cm for a Nao robot, to erase obstacles the robot could drive or walk under. The remaining points are projected to the ground and sorted into a 2D grid of 100x100 cells that extend 2.5 m to the left and to the right and 5 m to the front of the robot. Each cell that contains at least one point is marked with a value of 1. The remaining cells are marked with 0. We interpret this grid as a binary image, dilate it by the radius of the robot, and use Suzuki' algorithm [23] to detect the contours of contiguous groups of cells, which are reported in counter-clockwise order. The coordinates of these cells are already valid polygons. We reduce the complexity of the reported polygons using the Douglas Peucker algorithm [24] that joins line segments which lie almost on the same line. These computation steps are illustrated in Figures 2 and 5.

The dilation of the occupancy grid prior the contour detection is an easy way of inflating the obstacles by the radius of the robot so that the robot can be regarded as a point, which is helpful for motion planning [8]. The dilation of the grid automatically handles cases that become challenging when the inflation is done after the polygons have been computed, for example when non-convex polygons become self-intersecting or overlap each other after they have been expanded. The output of the contour detection after the



(a) Rectified Point Cloud (b) Floor Removal (c) Dilated Occupancy Grid (d) Smoothed Polygons

Fig. 5: Construction of a two-dimensional polygonal environment model. a) Raw point cloud of the RGB-D sensor. b) The floor points have been removed after floor detection. c) Projection into an occupancy grid, which has been dilated by the radius of the robot. d) Polygons after contour detection and smoothing.

dilation is guaranteed to produce non-convex polygons that do not self-intersect and do not overlap each other.

V. Experimental Results

A. Real Robot Experiments

We evaluated our method in experiments with two different mobile robots in realistic environments. The experiments can be seen in the accompanying video[1]. We used a wheeled robot "Robotino" by FESTO and recorded a few minutes of driving through an office environment. A cluttered office room, an empty corridor with straight planar surfaces, and a human walking through the sensor range can be seen, all correctly resolved to floor-projected polygons. We also mounted an RGB-D sensor on the head of a Nao robot and recorded a walk sequence in a miniature lab environment. The polygonal sensing pipeline was not modified in any way to adapt it to the humanoid. As can be seen in the video, the excessive shaking of the walking robot does not disturb our system. Despite of the shaking camera, the point cloud is always correctly aligned with the floor and the polygons are extracted the same way as on the Robotino robot. Figure 5 shows images of a cluttered office as seen by the Nao robot during these experiments. Even small details such as the legs of the table appear correctly as polygons.

[1] Video: https://youtu.be/ij3ZonX8iM8

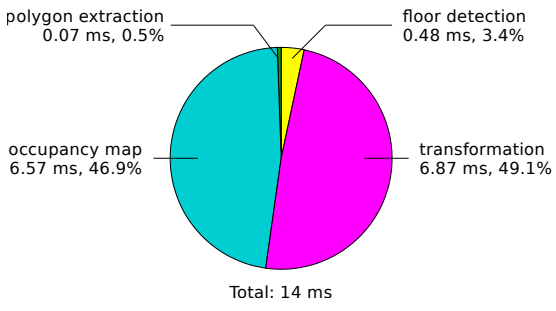


Fig. 6: Runtime analysis of our perception pipeline. The vast majority of the computation time is spent on the transformation of the depth pixels into the world frame and the sorting into the occupancy grid. The floor detection and in particular the polygon extraction take only a small fraction of the computation time.

In Figure 4, we are showing two particularly challenging situations. In situation a), the RGB-D camera is facing a wall so that only few floor points can be seen. The floor plane is fitted correctly. In situation b), the camera is looking over a tabletop that makes up the majority of the point cloud. The floor is detected correctly in a small patch on the far side of the table. Assumptions such as the floor being the largest plane, or the floor appearing in the bottom half of the image, would render the floor detection impossible in this situation.

B. Runtime Analysis

Regarding the computation time that is needed to process our perception pipeline we can say that with an average runtime of 14 ms as measured on an Intel Core i5 2.5 GHz CPU, we are well below the 33 milliseconds mark that is needed for real-time capability. We determined the runtime by averaging 1000 time measurements of single frames with the sensor in the loop. Figure 6 shows an analysis of the computation times of the processing steps. The vast majority of the computation time is spent on transforming the approximately 300K points from the camera frame into the world frame, and on sorting the transformed points into the occupancy grid. The processing times of the floor detection and the polygon extraction are only a small fraction of the total time.

C. Motion Planning

As an example of application, we used two motion planning algorithms that use polygons as inputs. The shortest path shown in Figure 7 has been computed with the Minimal Construct algorithm [2]. The motion trajectories shown in Figure 7 have been computed by a recent version of the Dynamic Window Approach [4] that has been adapted to a polygonal input.

VI. Conclusions

We introduced a perception pipeline that creates a polygonal obstacle map in the field of view of the robot from the point cloud perceived by an RGB-D sensor. Our approach is computationally efficient and robust to the disturbances of a driving or a walking robot as we demonstrated in real-world experiments with two different types of robots. The method includes a floor plane detector that reliably identifies floor

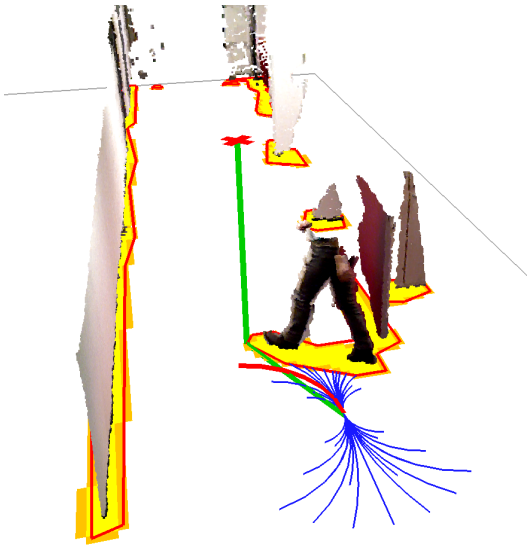


Fig. 7: Motion planning in a polygonal scene computed from a 3D point cloud. The points show a situation in a corridor where a human gets in the way of the robot. The shortest path shown in green to the target cross has been computed with the Minimal Construct algorithm [2]. The motion trajectories shown in blue have been computed by a polygonal version of the Dynamic Window Approach [4]. The cells of the occupancy grid are shown in yellow and orange color. The computed polygons are outlined in red.

points even in challenging situations so that the floor can be removed from the scan prior to the segmentation to polygons. We used the produced polygons in combination with a shortest path planner and the Dynamic Window Approach without any issues. The source code of our implementation is available [2]. In future work, we intend to expand our system to build polygonal maps and to determine the velocity of the polygons by object tracking.

References

[1] E. Marder-Eppstein, E. Berger, T. Foote, B.P. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[2] Marcell Missura, Daniel D. Lee, and Maren Bennewitz. Minimal construct: Efficient shortest path finding for mobile robots in polygonal maps. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2018.

[3] Jamie Snape, Jur van den Berg, Stephen J. Guy, and Dinesh Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, pages 4584–4589. IEEE, 2010.

[4] Marcell Missura and Maren Bennewitz. Predictive collision avoidance for the dynamic window approach. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2019.

[5] 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. *Int. Journal of Humanoid Robotics*, 2019.

[6] Robert J. Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. Footstep planning for autonomous walking over rough terrain, 2019.

[7] Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 942–947. AAAI Press, 2006.

[8] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[9] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, Mar 2016.

[10] A. C. Hildebrandt, D. Wahrmann, R. Wittmann, D. Rixen, and T. Buschmann. Real-time pattern generation among obstacles for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2015.

[11] Khelifa Baizid, Guillaume Lozenguez, Luc Fabresse, and Noury Bouraqadi. Vector maps: A lightweight and accurate map format for multi-robot systems. In *Intelligent Robotics and Applications - 9th International Conference, ICIRA 2016, Tokyo, Japan, August 22-24, 2016, Proceedings, Part I*, pages 418–429, 2016.

[12] Johann Dichtl, Luc Fabresse, Guillaume Lozenguez, and Noury Bouraqadi. Polymap: A 2d polygon-based map format for multi-robot autonomous indoor localization and mapping. In Zhiyong Chen, Alexandre Mendes, Yamin Yan, and Shifeng Chen, editors, *Intelligent Robotics and Applications*, pages 120–131, Cham, 2018. Springer International Publishing.

[13] Johann Dichtl, Xuan Sang Le, Guillaume Lozenguez, Luc Fabresse, and Noury Bouraqadi. Polyslam: A 2d polygon-based SLAM algorithm. In Luís Almeida, Luís Paulo Reis, and António Paulo Moreira, editors, *2019 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2019, Porto, Portugal, April 24-26, 2019*, pages 1–6. IEEE, 2019.

[14] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. *Computer Graphics Forum (Proc. of Eurographics)*, 28(2):503–512, March 2009.

[15] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94–103, February 2016. Special Issue on CAD/Graphics 2015.

[16] Arne-Christoph Hildebrandt, Robert Wittmann, Felix Sygulla, Daniel Wahrmann, Daniel Rixen, and Thomas Buschmann. Versatile and robust bipedal walking in unknown environments: real-time collision avoidance and disturbance rejection. *Autonomous Robots*, Feb 2019.

[17] 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.

[18] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semi-definite programming. In *in Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.

[19] Dirk Holz, Ruwen Schnabel, David Droeschel, Jörg Stückler, and Sven Behnke. Towards semantic scene analysis with time-of-flight cameras. In *RoboCup 2010: Robot Soccer World Cup XIV*, pages 121–132, 2011.

[20] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using RGB-D cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317. Springer Berlin Heidelberg, 2012.

[21] Dirk Holz and Sven Behnke. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In *Int. Conf. on Intelligent Autonomous Systems (IAS)*, 2012.

[22] P. Karkowski and M. Bennewitz. Real-time footstep planning using a geometric approach. In *ICRA*, 2016.

[23] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 1985.

[24] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 10 1973.

[2] https://github.com/MarcellMissura/polygonalperception