Robust Stairway Detection and Localization Method for Mobile Robots using a Graph-Based Model and Competing Initializations

Thomas Westfechtel¹, Kazunori Ohno²,³, Bärbel Mertsching⁴, Ryunosuke Hamada², Daniel Nickchen⁵, Shotaro Kojima¹, Satoshi Tadokoro¹

Abstract
One of the major challenges for mobile robots in human-shaped environments is stairway passage. This study presents a method for accurately detecting, localizing, and estimating the characteristics of stairways using point cloud data. The main challenge is the wide variety of different structures and shapes of stairways. This challenge is often aggravated by an unfavorable position of the sensor, which leaves large parts of the stairway occluded. This can be further aggravated by sparse point data. We overcome these difficulties by introducing a three-dimensional graph-based stairway detection method combined with competing initializations. The stairway graph characterizes the general structural design of stairways in a generic way that can be used to describe a large variety of different stairways. By using multiple ways to initialize the graph, we can robustly detect stairways even if parts of the stairway are occluded. Furthermore, by letting the initializations compete against each other, we find the best initialization that accurately describes the measured stairway. The detection algorithm utilizes a plane-based approach. We also investigate different planar segmentation algorithms and experimentally compare them in an application-orientated manner. Our system accurately detects and estimates the stairway parameters with an average error of only $2.5 \text{ mm}$ for a variety of stairways including ascending, descending and spiral stairways. Our method robustly works with different depth sensors for either small or large-scale environments and for dense and sparse point cloud data. Despite this generality, our system's accuracy is higher than most state-of-the-art stairway detection methods.

Keywords
Recognition, Range Sensing, Field Robots, Localization

Introduction
In the DARPA Robotics Challenge in 2015, one of the eight tasks for the competition was stairway climbing. The results of that competition highlighted the passage of stairways remains a major challenge for robots. This holds even more if the robot is to autonomously operate, opposed to the tele-operated way used in the challenge. A good detection method for stairways can help alleviate this problem. Knowledge about stairway parameters (i.e. height and depth of the steps) would support the tele-operator in his task and is essential for passage in an autonomous way of operation.

Stairways are everywhere in human shaped environments. Therefore, there are many examples of robots and robotic applications that could greatly benefit from the stairway detection method. Human assisted robots like semi-autonomous wheelchairs could adjust their operation mode to climb stairways. Some wheelchairs like the iBOT and SCEWO have already shown their capability to climb stairways. By detecting the exact location and parameters of the stairway, the wheelchair could autonomously position itself to the stairway and adjust the stairway parameters to not just facilitate the user’s controlling task but also allow for a more comfortable climb.

Specially in the field of rescue robotics, a robot needs to be able to explore and navigate with little to no prior knowledge. The possibility of detecting stairways allows for a safe stairway passage and enables the robot to extend the range of exploration to other floors. In other cases, the robot operates in a known environment and a pre-known map could be provided to support it. In such a case, a stairway detection method could be employed to help ease and reduce the manual task of creating and annotating large-scale maps. In addition, during operation detected stairways can be used as landmarks to support the robot in localizing itself.

This study presents a three-dimensional graph-based stairway detection and localization method for mobile robots. The main target of the method is our robot “Quince”...
Figure 1. Various types of stairways we want to detect.

(Figure 2a) which is equipped with a light detection and ranging (LIDAR) sensor that acquires a 3D scan of the environment. We also test and evaluate our algorithm for other depth sensors like the Kinect sensor (Figure 2b) and the more precise FARO Focus 3D LIDAR (Figure 2c).

Our approach utilizes planar segments of stair risers and treads to initialize a dynamic stair graph. We reviewed and experimentally compared different planar segmentation methods in terms of their ability to detect the elements of stairways (i.e. risers, treads and railing system).

Our goal is to robustly detect and localize stairways regardless of the type of stairway, the robot’s position and the robot’s viewing angle onto the stairway. These goal settings aggravate the problem. Stairways can considerably vary in the dimensions of the step’s depth and height (Figure 1). Some stairways like helical or spiral ones also exhibit an angular difference in alignment between each step (Figure 1d). Furthermore, the railing system is not strictly defined and seldom similar for two different stairways. Industrial stairways often have discontinuous stair risers, indicating an air gap between the steps (Figure 1a).

The independence of the robot’s position includes that it may be a few meters away from the stairway during the acquisition process. The point density is proportional to the inverse quadratic of the distance (∼1/d²) and as in the case of using a LIDAR the overall density is limited by the acquisition time, hence a successful detection should be possible even for sparse data.

This paper builds on our previous work (Westfechtel et al. 2016). Among other parts, we extended our graph-based model to include circular stairways, introduced the concept of competing initializations and extended the experimental evaluation (i.e. evaluation for different depth sensors).

The main contributions of this work are presented as follows:

• A method that is highly versatile due to the universality of the graph-based model. The graph-based model enables us to detect straight or spiral stairways for which only stair risers are visible because of the low height positioning of the sensor or for which only stair treads are visible (i.e. descending stairways).
• A competing initialization paradigm of the stairway model, that enables us to robustly and accurately identify stairways even in case of unfavorable conditions like sparse point cloud density for a variety of depth sensors.
• An application-oriented experimental comparison between different planar segmentation algorithms for the concrete case of detecting stairway elements.

Related Works

Stairway Detection

Stairway detection has been the focus of research for some time now. Accordingly, methods have been developed for different acquisition setups ranging from monocular cameras (Cong et al. 2008), stereo cameras (Harms et al. 2015), to depth cameras (Delmerico et al. 2013) or LIDAR sensors (Oßwald et al. 2011). The detection methods can be mainly divided into two main approaches: edge-based and plane-based. Edge-based methods try to identify the edges of the single steps of a stairway, whereas plane-based methods use the horizontal planes of stair treads and the vertical planes of stair risers to recognize stairways. A few other approaches have also been introduced; however, they will be introduced after the two main approaches because they are quite diverse.

Edge-based methods mostly utilize visual or depth cameras as sensing devices. Murakami et al. (2014) used a Sobel filter and a Hough transformation on the acquired data of a Kinect to detect the stairs’ edges. The stairway is detected if enough edges with distances comparable to a step’s size are found. An approach utilizing a Gabor
filter in combination with fuzzy fusion phase-grouping to detect the stairway in camera images was presented by Zhong et al. (2011). Delmerico et al. (2013) presented a stairway detection algorithm for the Kinect camera. The depth discontinuities at the top of the stair risers were used to detect stairways in the dense depth image. These discontinuities were present because of the occlusion of the stair treads caused by the sensor being positioned at a very low height. Harms et al. (2015) developed a stairway detection for stereo vision. A depth map from the image data was first generated. The concave and convex edges at the bottom and top of a stair riser were identified using a convolution pattern. The edges were paired to detect the stairway. Cong et al. (2008) and Samakming and Srinonchat (2008) presented more examples for the methods within this category.

Aside from the edge-based methods, the other main approach utilizes planes for the detection, mainly the horizontal planes of stair treads and the vertical planes of stair risers. These methods mostly employ a LIDAR or a depth camera. Oßwald et al. (2011) compared two different plane segmentation methods, namely the scanline grouping algorithm and the two-point random sampling algorithm, for detecting the treads and risers of steps. The intersection lines of the planes were used to estimate the stairway parameters and generate a stairway model. In their research, a humanoid robot equipped with a LIDAR was placed directly in front or on the stairway. The acquired point cloud was assumed to be mainly populated by the stairway. It is doubtful that their segmentation methods will work with a broader viewing angle. Vlaminck et al. (2013) searched for horizontal planes using a RANSAC algorithm at a predefined height offset from the ground, with the height offset presenting a predefined step height parameter. If a step is detected the process is repeated for the next successive steps. While the algorithm robustly detects the first few steps of a stairway, its accuracy declines for the later steps because no stair parameters were estimated. Qian and Ye (2014) used an NCC-RANSAC algorithm to extract planes within the surroundings acquired by a depth camera. A Gaussian mixture model (GMM) was used to classify the planes into eight categories. The planes’ features and inter-plane relationships were used as input feature for the GMM. Stairways represent only one class among others; hence no in-depth evaluation about the stairway detection, including an estimation of the stairway parameters, has been conducted. Similar to this approach, Richtsfeld et al. (2014) used relations between patch-pairs derived from perceptual grouping principles to segment objects in RGB-D data. While the publication focused on table-top scenes and did not address stairways, it is only reasonable to assume that it could easily be extended to stairways, given additional training data for the support vector machine process. Luo et al. (2013) used a vertical histogram to detect horizontal planes. Equidistant stair treads were used to initialize an adjustable stairway model. The authors showed the segmentation result for the depth images acquired by an RGB-D camera; however, that the algorithm would work for a 360° round view is unlikely.

Aside from the edge- and plane-based recognition methods, some researches have also experimented using other techniques to identify stairways. An interesting

<table>
<thead>
<tr>
<th>Method</th>
<th>Modality</th>
<th>Requirements</th>
<th>Small and Large Scale</th>
<th>Parameter Estimation</th>
<th>Ascending Stairways</th>
<th>Descending Stairways</th>
<th>Spiral Stairways</th>
</tr>
</thead>
<tbody>
<tr>
<td>Murakami et al. (2014)</td>
<td>RGB-D</td>
<td>Edges</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Zhong et al. (2011)</td>
<td>RGB image</td>
<td>Edges</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>~×</td>
<td>×</td>
</tr>
<tr>
<td>Delmerico et al. (2013)</td>
<td>Depth camera</td>
<td>Edge discontinuity</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>~×</td>
<td>×</td>
</tr>
<tr>
<td>Harms et al. (2015)</td>
<td>Stereo Vision</td>
<td>Edges</td>
<td>~×</td>
<td>✔</td>
<td>✔</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Cong et al. (2008)</td>
<td>RGB image</td>
<td>Edges</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>~×</td>
<td>×</td>
</tr>
<tr>
<td>Samakming and Srinonchat (2008)</td>
<td>RGB image</td>
<td>Edges</td>
<td>×</td>
<td>✔</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Oßwald et al. (2011)</td>
<td>LIDAR</td>
<td>Robot in front/on stairway</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>~</td>
<td>×</td>
</tr>
<tr>
<td>Vlaminck et al. (2013)</td>
<td>RGB-D</td>
<td>Treads visible</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>×</td>
</tr>
<tr>
<td>Qian and Ye (2014)</td>
<td>FLC</td>
<td>Treads or Risers visible</td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td>~</td>
<td>×</td>
</tr>
<tr>
<td>Luo et al. (2013)</td>
<td>Depth camera</td>
<td>Treads visible</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>~×</td>
<td>×</td>
</tr>
<tr>
<td>Pérez-Yus et al. (2014)</td>
<td>RGB-D</td>
<td>Treads visible</td>
<td>~×</td>
<td>✔</td>
<td>✔</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Tang et al. (2012)</td>
<td>RGB-D</td>
<td>Treads visible</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>✔</td>
<td>~×</td>
</tr>
<tr>
<td>Sinha et al. (2014)</td>
<td>Depth camera</td>
<td>Slope</td>
<td>~×</td>
<td>×</td>
<td>✔</td>
<td>~×</td>
<td>~×</td>
</tr>
<tr>
<td>Ours</td>
<td>LIDAR</td>
<td>Treads or Risers visible</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Table 1. Comparison of different stair detection methods. ✔: = supported, ×: = unsupported, ~ = unclear from the publication, ~×: = unclear, but probably supported, ~×: = unclear, but probably not supported
approach was used by Eilering et al. (2014). They utilized a Markov random field to label the points of a point cloud into different support surface classes (i.e. support plane, wall, rail). While the results of the classification seemed good, the authors did not mention how to use their results in detecting stairways as a whole. Sinha et al. (2014) utilized another approach. They developed a feature used to identify step-like objects. The feature makes use of the stepwise height incrementation of adjacent steps. Nearby detected feature points are clustered under the condition that they have a similar height increment. However, the same stairway was often separated into several different clusters, and in some cases, the algorithm mistook other objects like a wheelchair, for a stairway.

In summary, all edge-base detection methods make use of the organized array-like structure (i.e. $640 \times 480$) for edge detection. This structure is available for visual images and depth data acquired by RGB-D cameras and allows the use of convolution patterns which most edge detection methods rely on. However, this structure is not available for unorganized point cloud data acquired by a LIDAR sensor. Therefore, these methods are not usable in our case.

Some presented methods like the point feature histograms (Rusu et al. 2008), radius-based surface descriptors (Marton et al. 2010) and principle curvature of the point cloud library (Rusu and Cousins 2011) can detect edges without the need for an organized structure. We conducted first experiments using these descriptors with our acquired data. While they showed good results for dense regions, they had severe problems with regions of sparse point density. Therefore, an edge-based approach was not further pursued in this work.

Plane-based methods mostly have some prerequisites for stairway detection. The stair treads sometimes need to be visible (Luo et al. 2013), sometimes they need to be occluded (Delmerico et al. 2013). Other times, both treads and risers are required for the detection (Oßwald et al. 2011), (Harms et al. 2015). The work of Qian and Ye (2014) is the only exception here, although their goal was to classify planes correctly finding multiple planes, a single plane crossing through the two patches, which consists of more points in total, is often found. This was prevented by adding the connectivity check. The researchers divide the point cloud in non-overlapping groups. A graph is constructed to cluster these groups. Thus, the extracted planes are refined with a pixel-wise region growing. Õckermann et al. (2013) detected normal edges and depth discontinuities in a first step. Region growing was utilized on each surface until a normal edge or depth discontinuity was reached. The researcher then used coplanarity and a probabilistic composition to achieve an object-based detection out of the primitive regions acquired in the first step. A region growing based approach was also utilized by Arbeiter et al. (2014). In their work, normal-based region growing was used. The segments were then employed in conjunction with point feature descriptors to not only detect planar segments, but also curved surfaces.

The basics of these algorithms are mostly the same, with slight modifications or extensions. However, most of these modifications make use of additional information, namely the organized structure of RGB-D images. The detection of depth discontinuity as presented by Õckermann et al. (2013) is detected through the depth differences of two neighboring pixels in a two-dimensional (2D) depth image. The structure is also often exploited to speed up the segmentation process (i.e. Feng et al. (2014)). However, we employ point clouds acquired from a LIDAR sensor; hence, we cannot employ these modifications.

Aside from the region growing-based methods, another very popular method is the usage of random sample consensus (RANSAC). Schnabel et al. (2007) used the RANSAC algorithm to detect different shapes. In their work, a random set of points was used to initialize a primitive shape. The shape was extended to the other points within the point cloud, and based on the consensus, the shape was either accepted or rejected. Points belonging to a detected shape were removed from the point cloud, and the algorithm was reiterated until only a few points were left in the point cloud. Gallo et al. (2011) extended RANSAC with a component connectivity check. The motivation was that RANSAC often fails to correctly segment steps, curbs or ramps. Instead of correctly finding multiple planes, a single plane crossing through the two patches, which consists of more points in total, is often found. This was prevented by adding the connectivity check. The algorithm was extended by Qian and Ye (2013) who added a normal-coherence check to further improve the segmentation in a multistep scene.
Figure 3. Stairway detection scenario. Robot position marked as red circle, detected stair treads as blue and stair risers as red.

With the right conditions, the RANSAC algorithm can achieve quite good results and, therefore, still enjoys a large popularity. However, the runtime can be quite large, requiring lots of initializations to find a good initialization, especially in a large, mainly unstructured environment. Furthermore, using the original implementation without modifications, the RANSAC segmentation fails to correctly segment steps, curbs and ramps and fails to distinguish separate coplanar planes as different regions.

Another interesting approach, though not as popular, is the split and merge method (Wang and Tseng 2004). Like the name suggests, the algorithm splits the point cloud into increasingly finer voxels until all points within a voxel form a plane. In a second step, adjacent voxels that share the same plane coefficients are merged. This approach is interesting because it estimates the point normals in an efficient manner through the voxel structure. The connectivity of the planes is also guaranteed to a certain degree through the adjacency constrain of the voxels during the merging process. However, the minimal resolution of this technique is limited to the smallest voxel size. It also heavily depends on the point density because at least three points are required inside a voxel to calculate the plane coefficients. Nevertheless, this technique is very interesting because of its deterministic behavior and because no point normals are required. Furthermore, a segmentation based on voxel space could easily be extended to further include more sophisticated features such as pass through percentage (Shamsudin et al. 2016) to further enhance the segmentation.

Aside from these techniques, machine learning techniques have gained increased popularity in segmentation tasks. Among other techniques, a decision forest and a randomized decision forest were used by Hermans et al. (2014) for dense 3D semantic mapping. They used a 2D-3D label transfer to obtain a consistent 3D semantic reconstruction of indoor scenes. Müller and Behnke (2014) employed a conditional random field for semantic mapping of RGB-D images. A conditional random field for semantic mapping was also used by Wolf et al. (2015). In a further work, Wolf et al. (2016) extended an underlying random forest architecture with 3D entangled features to include the inter-relations between nearby objects. Machine learning techniques are often combined with the usage of superpixels (Achanta et al. 2012). These techniques are also often used to detect not primitive shapes, but regions with a specific semantic meaning. Our stair detection only relies on basic planar surfaces (of the stair treads and stair risers), and machine learning techniques require much training data; thus, we chose not to employ this approach for the segmentation task.

Stairway Detection Method

This section briefly explains our graph-based stairway detection method. The detection follows the processing pipeline shown in Figure 4. Figure 3 depicts a typical detection scenario with the parts of the detected stairway colored by their class. The input point cloud is acquired by a LiDAR sensor mounted on top of the Quince robot (Figure 2a), however, the detection is also possible for other depth sensors as will be shown later. The input point cloud is first preprocessed and segmented. For the segmentation task, we investigated and compared three different methods, which will be explained in the next section. Planes resembling stair treads or risers will be extracted after the segmentation. If two close-by stair treads or risers are found, their interrelation is used to initialize a dynamic stairway graph. The graph is extended in the stairway’s direction. The detected stairway is confirmed if it complies to certain conditions (i.e. the DIN norm (German standard norm) for stairs in buildings (DIN:18065:2015-03 2015)). For confirmed stairways the railing system to the side of the stairway is added, and the stairway is localized. The processing steps for the stairway detection will be explained in detail in the section titled “Graph-based Stairway Detection.”

Segmentation
We examined and evaluated three different segmentation methods for the segmentation task. We investigated a region growing method, a split and merge method and an own devised VoxelSAC method, which is a combination of the split and merge method and the RANSAC method. The methods are explained in more detail in this chapter.

Region Growing
Algorithm 1 depicts the region growing algorithm. The algorithm starts to grow a region based on a seed point. The points in the vicinity of the region that share the same properties are added to the region. This process is
Algorithm 1 Region Growing Algorithm

1: function REGIONGROWING(Point \( P_i \))
2: \( R_n = \{ \} \)
3: \( R_n += P_i \)
4: Plane center \( P_n = P_i \)
5: Plane coefficients \( C_n = \) point normal \( N_i \)
6: Find \( k \) nearest neighbors \( P_{1:k} \)
7: if \( P_x(x \leq k) \) not assigned to region then
8: \( \) Add \( P_x \) to comparison list \( L \)
9: for All points in the comparison list \( L \) do
10: \( \) if Plane distance of current point \( P_c \) is within a certain limit then
11: \( \) if Current normal \( N_c \) complies with \( C_n \) then
12: \( \) Add Point to region \( R_n += P_c \)
13: Update \( P_n \) with \( P_c \)
14: Update \( C_n \) with \( N_c \)
15: Find \( k \) nearest neighbors \( P_{1:k} \)
16: if \( P_x(x \leq k) \) not assigned to region then
17: \( \) Add \( P_x \) to comparison list \( L \)
18: \( \) Pop \( P_c \) from comparison list \( L \).
19: \( \) procedure MAIN
20: \( \) Sort point cloud \( PC \) by curvature
21: for All points \( P_i \in PC \) do
22: \( \) if \( P_i \) is not assigned to a region then
23: \( \) Region \( R_n = \) regionGrowing\( (P_i) \)
24: \( \) if \( P \in R_n \geq 30 \) then
25: \( \) Add \( R_n \) to output

continued until no more points in the vicinity complying to the properties can be found. For our experiments, we used the region growing implementation of the PCL with minor modifications (Rusu and Cousins 2011). We used the plane distance and the plane normal as growing properties or conditions. The plane distance was calculated as follows:

\[
d_p = (p_n - p_c) \cdot \vec{e}_n
\]

where, \( p_n \) is the center point and \( \vec{e}_n \) the normal of the plane that is grown, and \( p_c \) is the coordinate of the current point under investigation. We updated the plane coefficients with each additional point added to the region to compensate for measurement inaccuracies.

Split and Merge

Figure 5 presents the split and merge algorithm. Like the name suggests, the algorithm splits the input point cloud (Figure 5a) into increasingly finer voxels, until all points within a voxel form a plane (Figure 5b). Adjacent voxels with a similar plane coefficient are merged in a second step (Figure 5c). Algorithm 2 depicts the algorithm. We employed the principal component analysis (PCA) to calculate the plane coefficients for each voxel. The plane distance measure was used to decide whether a point complies to the plane or not. The algorithm is interesting because it evaluates the points of each voxel together as opposed to region growing algorithms. Therefore, point normals do not have to be evaluated beforehand. Although not used in our work, another advantage is that the voxel-based evaluation allows for an extension of primitive shapes like spheres and cylinders or the estimation of more sophisticated features like the pass-through percentage (Shamsudin et al. 2016). One major disadvantage of the split and merge algorithm is the limited resolution resulting from the minimal voxel size.

VoxelSAC

RANSAC is still one of the most popular segmentation methods; hence, we wanted to include a segmentation algorithm based on this algorithm. However, without modifications, the RANSAC segmentation fails to correctly segment steps, curbs and ramps and fails to separate coplanar...
planes into different regions. We wanted to segment a 360° point cloud; thus, an implementation without modification does not produce a viable result.

Accordingly, we chose a combination of the split and merge algorithm and RANSAC, and call this method the VoxelSAC. Our modification made use of the local properties of shapes, indicating the probability that the same surface being measured is higher in locally close positions. Algorithm 3 presents the algorithm. While Schnabel et al. (2007) used a random sized voxel around the first drawn sample, we chose to create subsets of the point cloud using voxels starting from large-sized voxels and iteratively decrease the voxel size. We executed RANSAC on each generated subset. Using this local limitation of the subsets, we can require a high percentage of inliers or points that match the plane hypothesis of RANSAC, which, consequently allows for an early hypothesis rejection. A sample consensus (SAC) process is initiated if a plane with sufficient inliers in a voxel is found. The algorithm searches for inliers of the plane in the adjacent voxels. The SAC process is extended to the neighboring voxels of $v_i$ if enough inliers are found within an adjacent voxel $v_i$. Thus, all points added to the region are then removed from the point cloud. The SAC process is extended by voxels that have the same size as the voxel for which the RANSAC plane was found. The initial voxel size, therefore, limits the distance of a gap with no plane-inliers. This allows us to forgo a special connectivity check, which would also have to consider the point density. However, since there is still a limited gap with no plane-inliers that the algorithm can span, this, on one hand, might lead to undersegmentation, but, on the other hand, also gives the algorithm the ability to deal with occlusion.

The process is repeated until a minimal voxel size is reached. By starting with a large voxel size, the larger planes are detected first, which rapidly reduces the remaining points and decreases the processing time. Figure 6 depicts the process. Figure 6a shows the input cloud, Figure 6b presents the results after a voxel size of 16 cm. Figure 6c illustrates the results for the next smaller voxel size of 8 cm. Lastly, Figure 6d shows the final result with a voxel size of 4 cm.

VoxelSAC shows some advantages in comparison to the region growing. First, the initialization process is done voxel-wise compared to point-wise for region growing. Therefore, the impact of noise and measurement error can be compensated. Furthermore, VoxelSAC deploys a different concept of neighborhood. For the region growing algorithm, the neighboring points for each point added to the region are searched for. In the case of a partial occlusion separating the plane into several parts, this can result in over-segmentation. For VoxelSAC, neighboring voxels with all their points are added to the SAC list. Hence, VoxelSAC can span over an occlusion of approximately 1 time of the current voxel size; however, this may also lead to overgrowing of regions in case of almost parallel close-by planes.

**Algorithm 3 VoxelSAC Algorithm**

1: function SACProcess(Plane $R_i$)
2:    Add adjacent voxel to SAC list $L_s$
3:    for All voxel in $L_s$ do
4:        for All points $p_i \in$ current voxel $v_c$ do
5:            if Point $p_i$ complies with $R_i$ then
6:                if Normal of $p_i$ complies with $R_i$ then
7:                    Add $R_i = \perp = p_i$
8:                if $k$ or more points of $v_c$ were added to $R_i$ then
9:                    Update plane coefficients of $R_i$
10:                   Add adjacent voxel of $v_c$ to SAC list $L_s$
11:                   Pop current voxel $v_c$ from $L_s$
12:    end for
13:    end for
14: end function

15: procedure MAIN
16:    while Current voxel size $s_v \geq s_{\text{min}}$ do
17:       Create octree $Oct$ of point cloud $PC$
18:       Set leaf size of $Oct$ to $s_v$
19:       for All leaves $l_i \in Oct$ do
20:          Estimate plane $R_i$ for $l_i$ using RANSAC
21:          if All points $p \in l_i$ comply to $R_i$ then
22:              SACProcess($R_i$)
23:          end if
24:       end for
25:    end while
26:    if $p \in R_i > 30$ then
27:       Add $R_i$ to output
28: end procedure

29: $s_v = s_v / 2$

**Graph-based Stairway Detection**

This section explains step-by-step the stairway detection method using the results of the segmentation results. For this, the segmented cloud is first analyzed with a plane finder to find stair tread- and riser-shaped elements. If two stair riser- or tread-shaped regions have an inter-plane relation similar to the relations between two adjacent stair steps, these regions are used to initialize a dynamic stairway graph. The graph model is extended along the direction of the stairway until no more treads or risers complying with the model are found. The detected stairway is confirmed if the grown stairway graph satisfies specific conditions. The stairway is expanded with the railing system to the sides of the stairway. In the last
step, the stairway is localized using a dynamic anchor point, depending on whether a railing system has been detected.

**Plane Finder**

The plane finder is used to detect stair tread- and riser-shaped planes in the segmented point cloud. The filter considers only the features of a single plane. Therefore, the computational complexity is linear $O(n)$ compared to the calculation of the inter-plane relations, which has a quadratic complexity of $O(n^2)$. We assume that the point cloud is aligned with respect to the robot and, hence, with respect to the floor. Therefore, we infer that the stair risers are vertical, while the stair treads are horizontal planes. In other words, the normals of the risers and treads are perpendicular to the vertical and horizontal directions, respectively. Furthermore, the planes have to be of limited size. According to the German standard norm DIN:18065:2015-03 (2015), stair risers have a maximal height of 21 cm, while stair treads have a maximal depth of 37 cm. We measure the extension of the plane along its eigenvectors to calculate the size of a plane. For this the plane is first aligned according to its eigenvectors and the minimal and maximal values for each axis (its eigenvector) are measured. The eigenvectors are determined using the PCA. In summary, the plane finder investigates two features according to the standard norm, including an additional error margin:

- The normals of the risers’ and treads’ planes are horizontal and vertical, respectively, to the orientation of the room. A deviation of up to $15^\circ$ is accepted.
- The height extension for the stair risers is less than 24 cm, and the depth extension for the stair treads (extension along the second eigenvector) is less than 40 cm.

Figure 7 shows the results of the plane finder.

**Stairway Graph**

Although stairways are categorized into various types, they mostly share the same structure. Each stairway consists of multiple steps $S_i$ (Figure 8). Each of which consists of a horizontal plane for stepping (i.e., stair tread $P_t$), and a vertical plane that connects the treads of two consecutive steps, (i.e., stair riser $P_r$). These parts are hereafter called $P_{t/r}$. The steps are lined up along the stairway’s orientation $\vec{s}$ (facing in the direction of the descent) with a vertical distance of a step’s height $h$ and a horizontal distance of the step’s depth $d$. The stairway has a total width of $w$ and is often confined by a railing system at its sides, $R_l$ and $R_r$. Our idea is to use a model described by the following four parameters:

- Step height: $h$
- Step depth: $d$
- Step width: $w$
- Stairway orientation: $\vec{s}$

**Graph Initialization**

The stairway graph is initialized by either two nearby stair tread or riser planes that have a spatial relation resembling two consecutive steps. The main idea is that the same edge (front, back, top, or bottom) for them is most likely visible. However, we do not make any assumptions about which edge is visible, but rather initialize the graph multiple times using different edge pairs. The graph-based detection requires little time; hence, this has little influence on the overall runtime. The multiple initializations allow our approach to be utilized more universally and provide a better robustness against occlusion. Furthermore, we let the different initialization compete against each other, which allows us to find the initialization that describes the measured stairway best. The different strategies for the competing initializations will be explained in the subsection titled “Competing Initialization Strategies.”

For each consecutive stair riser, we initialize the graph up to two times: one with the height difference between the top edges $h_{tt}$, and the second one with the height difference between the bottom edges $h_{bb}$ (marked with the “∨” symbol in the formula). The values for the initialization with the stair risers are as follows (Figure 9a):

- $h = h_{tt} ∨ h = h_{bb}$
- $d = \left| (\vec{c}_1 - \vec{c}_2) \cdot \vec{s} \right|$  
- $\vec{s} = (\vec{n}_1 + \vec{n}_2) / \left( |\vec{n}_1 + \vec{n}_2| \right) $

where $\vec{c}$ denotes the centroid and $\vec{n}$ denotes the normal of a plane. Therefore, the graph is initialized once with $\{h = h_{tt}, d, \vec{s}\}$ and again with $\{h = h_{bb}, d, \vec{s}\}$.

We use the same multiple initialization strategy as for the stair risers for each consecutive stair tread. We initialize the depth as the distance between the front edges $d_f$ and as the distance between the back edges $d_b$. For the stair treads, we even extend this strategy to incorporate also the stairway’s orientation $\vec{s}$ because it can be difficult to accurately estimate. We initialize the stairway graph using the second eigenvector of the first plane $\vec{u}_1$ as the stairway’s orientation and the second eigenvector of the second plane $\vec{u}_2$. The parameters are initialized as follows (Figure 9b):
The graph is initialized for each combination for a total of up to four times. For a successful initialization we verify based on the German standard norm DIN:18065:2015-03 (2015) with an additional margin of ±3 cm and ±5°, such that:

- 0.11 m ≤ h ≤ 0.24 m
- 0.18 m ≤ d ≤ 0.40 m
- 15° ≤ atan(h/d) ≤ 50°

Otherwise, the initialization is rejected.

The possibility of initializing the model with either stair treads or risers allows for a higher versatility of the detection. A stair riser-based initialization is very crucial for us, because the stair treads are often occluded owing to the low height of our robot. In contrast, the stair risers are mostly occluded for descending stairways, making a stair tread-based initialization effective.

### Spiral Stairways

We extended the stair graph and the initialization method to expand the detection method to spiral stairways (Figure 10). An additional parameter ∆ϕ was added for the stairway graph. ∆ϕ represents the difference in the angular alignment of two consecutive steps ∆ϕ = α̂(n1, n2), where α̂ denotes the inner angle between two vectors. Furthermore, the depth d of a single step is not constant over its width but grows toward the outer part. Thus, we chose to fix the depth coefficient d to the center of the step. The depth is measured along n12, which is the angle between n1 and n2. Aside from these extensions, the same rules apply as for the straight stairways. Note that for our model, we assumed a constant angle between consecutive steps, which limits the detection of stairways with variable angular differences between steps.

### Detection of the Railing System

The algorithm searches for the railing system at the side of the stairway after the stairway is detected. The railing system mainly consists of balusters, which are the vertical posts that hold up the handrail, and the handrail for handholding. Balusters usually have a much larger height than width and depth, indicating that their main extension is in the vertical direction (= first eigenvector pointing in the z-direction). The handrail runs along the stairway and, therefore, shares the stairway orientation s. Furthermore, the railing has the same height inclination as the stairway itself because of the supporting nature that a railing system provides to a human user. The railing system is detected with these characteristics. The railing system can be used to distinguish stairways from stairway-like object arrangements.

### Localization Using a Dynamic Anchor

Some parts of the stairway may be occluded depending on the robot’s position. A corner wall may hide one side of the stairway (Figure 11), or the stair is not visible in its full width. This occlusion would result in an offset toward the non-occluded side if we would localize the stairway in the center at half of its width. To prevent this, we dynamically choose the anchor point depending on the detection of the railing system. If the railing system at the side of the stairway is detected, this side can be concluded to not be occluded, but be fully visible. By choosing the anchor point according to
Algorithm 4 Graph Extension

1: Generate list $J$ of all initializations from the stair treads and risers
2: Generate point clouds $C_r$ and $C_t$ consisting of the centroids of all stair riser or tread shaped planes
3: for All initializations $J_i \in J$ do
4: Create empty stairway graph $G_i$
5: Initialize $G_i$ with $J_i$
6: Set direction to ascending
7: while Ascending and descending directions are NOT marked as finished do
8: Move one step further along the direction
9: Search within a radius $r$ for stair risers or treads $P_{r/t}$ in $C_r$ or $C_t$ respectively
10: if $P_{r/t}$ is found then
11: if $P_{r/t}$ complies with the stairway graph then
12: Add $P_{r/t}$ to stairway graph $G_i$
13: if No parts for two consecutive steps are found then
14: Mark direction as finished
15: Return to starting stairway step
16: Change direction to descending
17: if Stairway is confirmed then
18: Detect railing system $R_l, R_r$ and add to $G_i$
19: Localize stairway
20: Calculate stair score for $G_i$
21: Append $G_i$ to list of confirmed stairway graphs
22: Choose unique $G_i$ with highest score as solutions $S$

Figure 11. Detection of stairway III with ambient point cloud (black) partly occluded by a wall framed in purple.

The detected railing system, the localization process achieves a higher robustness against partial occlusion. The anchor point $P_{stair}$ is chosen at the top front position of the first step of the stairway and is located

- on the right if only the right railing system is visible,
- on the left if only the left railing system is visible, or
- in the middle of the stairway if both or no railing system is detected.

Furthermore, the first step of a stairway may not be detected. An indicator for this is if the stairway seems to begin not on a ground level but at a higher position or, receptively, at a lower position in case of descending stairways. We extend the stairway graph until it reaches the ground level to still accurately localize the stairway.

The scoring function is made up of a riser part, marked with a subscript $r$ and a tread part, marked with a subscript $t$.

\[ S = S_r \cdot n_r + S_t \cdot n_t \]  

$n_r$ and $n_t$ describe the amount of riser and tread planes, respectively.

The riser scoring function is made up of three subscores: one for scoring the depth $S_{rd}$, one for scoring the height $S_{rh}$, and one for scoring the stairway orientation $S_{rs}$.

\[ S_r = S_{rd} + S_{rh} + S_{rs} \]

We calculate for each of the subscores a value $\delta^i$ that represents the difference between the graph model coefficients and the optimal coefficients of the $i$-th riser plane. The superscript $r$ stands for riser plane, the subscript $i$ for the $i$-th plane and an additional subscript $d$, $h$ or $s$ is added depending on the specific model coefficient. We
normalize each of the subscores to have a maximal value of 1. For the riser depth subscore, this is done as follows:

\[
S_{rd} = \frac{1}{n_r} \sum_{i=0}^{n_r} \frac{(\hat{d} - \delta_{d,i})}{\hat{d}}
\]  
(4)

with

\[
\delta_{d,i} = |(P_{est,i} - c_i) \cdot \hat{s}|
\]

where \(P_{est,i}\) depicts the estimated riser coordinates of the \(i\)-th riser according to the model, and \(c_i\) depicts the center point of the measured plane (Figure 9a).

\[
P_{est,i} = \hat{P}_{stair} + j \cdot \hat{d} \cdot \hat{s}
\]

(6)

where \(j\) depicts the step position of the current riser. We also calculate the score for the height:

\[
S_{rh} = \frac{1}{n_r} \sum_{i=0}^{n_r} \frac{\hat{h} - \delta_{h,i}}{\hat{h}}
\]

(7)

In calculating \(\delta_{h,i}\), we do not directly compare the height of each riser with the height of the model because parts of the risers are often hidden. Instead, we investigate if the edges of each riser extend above the estimated graph.

\[
\delta_{h,i} = \max(h_{i}^{top} - h_{est}^{top}, 0) + \max(h_{est}^{bot} - h_{i}^{bot}, 0)
\]

(8)

with

\[
h_{est}^{top} = \hat{P}_{stair} + j \cdot \hat{h}
\]

(9)

and

\[
h_{est}^{bot} = \hat{P}_{stair} + (j - 1) \cdot \hat{h}
\]

(10)

However, for the ascending stairways the bottom part of the stair risers is normally occluded by other steps, especially for measurements using the Quince robot with its low point of view (Figure 12a). In this case, we assume that the top part of the stair riser is visible and change the function accordingly to:

\[
\delta_{h,i} = |h_{plane}^{top} - h_{est}^{top}|
\]

(11)

The scoring orientation of the stairway \(S_{rs}\) is calculated as follows:

\[
S_{rs} = \frac{1}{n_r} \sum_{i=0}^{n_r} (1 - \delta_{r,i})
\]

(12)

with

\[
\delta_{r,i} = |\arccos(\hat{s} \cdot \hat{n}_i)|
\]

(13)

where \(\hat{n}_i\) depicts the normal of the \(i\)-th riser plane.

The score functions for the stair treads consist of a score for the depth \(S_{td}\) and the height \(S_{th}\). We do not score the direction of ascend from the stair treads because it has proven to be very unreliable in our experiments, especially for high occlusion or sparse data.

\[
S_t = S_{td} + S_{th}
\]

(14)

We calculate the subscores for the tread planes in a similar fashion to the riser planes. That is, we also calculate a difference measure \(\delta_t\) between the graph model parameters and the optimal coefficients for the \(t\)-th stair tread. The superscript \(t\) stands for tread planes. Each of the subscore is normalized with an upper value of 1. For the depth subscore \(S_{td}\), it is formulated as follows:

\[
S_{td} = \frac{1}{n_t} \sum_{i=0}^{n_t} (\hat{d} - \delta_{d,i})/\hat{d}
\]

(15)

In calculating \(\delta_{d,t}\), we employ a similar strategy as for the risers’ height, in that we calculate how far each tread extends over the borders of the model estimation:

\[
\delta_{d,i} = \max(d_{i}^{back} - d_{est}^{back}, 0) + \max(d_{i}^{front} - d_{stair}^{front}, 0)
\]

(16)

The borders of each tread are calculated similar to Equation (6), (9) and (10).

Finally, the score for the height part of the stair treads is calculated as follows:

\[
S_{th} = \frac{1}{n_t} \sum_{i=0}^{n_t} \frac{\hat{h} - \delta_{h,i}}{\hat{h}}
\]

(18)

with

\[
\delta_{h,i} = (P_{est,i} - c_i) \cdot [0, 0, \hat{h}]^T
\]

(19)

In the third strategy, which is the optimizing method, the scoring function is reformulated as an error function:

\[
E(\hat{h}, \hat{d}, \hat{s}, \hat{P}_{stair}) = \sum_{i=0}^{n_r} (\delta_{d,i} + \delta_{h,i} + \delta_{r,i}) + \sum_{i=0}^{n_t} (\delta_{d,i} + \delta_{h,t,i})
\]

(20)

The stairway coefficients are optimized by minimizing the costs of the error function. The Levenberg-Marquardt algorithm is used for the optimization. In addition, the stairway extension is executed once more using the optimized stair coefficients.

As mentioned earlier, only the winners of competing initializations are output. Therefore, no stair tread nor stair riser plane can be part of multiple stairway detections.

**Segmentation Evaluation**

We evaluated the segmentation algorithms on their ability to segment the different planes of stairways. To do this, we created an optimal ground truth (GT) segmentation for each measurement. The GT segmentation was created autonomously for each position using a CAD of the respective stairway in combination with a translation and rotation matrix depending on the robot’s position during acquisition. We created the CAD model using the SolidWorks software using manual measurements of the stair’s dimension. The translation and rotation matrix was determined by placing an ultra-precise FARO-Focus 3D laser scanner directly above the robot’s LIDAR and measuring the relative position to the stairway using the FARO SCENE software.
We filled the respective plane with points of a raster size of 1 cm to check for its visibility. We enforced that points must have a margin of at least 5 mm to the edge of the plane because the corner of the plane may be visible while the plane itself is not. We casted a ray from each point to the LIDAR’s position of acquisition. We checked whether the ray crosses any other plane of the stairway model. The point was occluded by that plane if it does. The whole plane is marked as visible if at least one point of a plane is found not to be obstructed by any other plane and is, therefore, visible.

Furthermore, at least 30 points of the point cloud data are required in measuring the surface of the plane for a plane to be detectable by the segmentation algorithms. This needs to be done because this threshold was implemented in the segmentation algorithms to suppress noise. We assigned each point to its nearest plane with a maximum point-plane distance of 5 cm according to the LIDAR’s accuracy (5 cm for up to 30 m). In addition, we projected the point to its closest point on the plane and performed a visibility check.

Figure 13c depicts the result of the point assignment to the CAD model. Interestingly the second and third stair treads were not visible because of the low height of the LIDAR mounted on the Quince robot.

**Evaluation Criteria**

In evaluating the segmentation results, the extracted planes must be matched to the ground truth model. We enforced the three following conditions for a successful mapping:

- The distance between the planes must be below 5 cm (to consider the measurement noise of the LIDAR).
- The difference in the plane normal must be within 15° (in accordance to the plane finder, see subsection “Plane Finder”).
- The overlap with the GT plane must be at least 50%. This value allows for some overgrowing over the plane edges, but still enforces that most of the surface overlaps.

We checked the overlap in a grid-based fashion with a 5 cm resolution. The overlap was the ratio of the sum of grids inhabited by both extracted and GT planes divided by the sum of grids inhabited by the extracted plane.

We evaluated four different measures with this mapping of the segmented planes. The first measure was the precision of detecting the planes, which was calculated as follows:

\[
Pr_{plane} = \frac{|P_{det} \cap P_{vis}|}{|P_{vis}|} \quad (21)
\]

\(P_{vis}\) represents the set of GT planes that are visible, including in our definition that at least 30 points were assigned to the plane. \(P_{det}\) represents the set of detected planes. The intersection \(|P_{det} \cap P_{vis}|\) is the number of GT planes to which at least one plane of the segmentation was mapped. Therefore, the precision is the percentage of detected unique planes of the GT.

The second measure, which is the plane recall, measures the over-segmentation of the algorithms.

\[
Re_{plane} = \frac{|P_{det} \cap P_{vis}|}{|P_{det}|} \quad (22)
\]
A value of 1 means that each visible GT plane has at most 1 segmented plane matched to it.

The precision and recall are not only calculated plane-wise, but also point-wise, which is possible because the same point cloud is used for the segmentation and the GT. We added an unique identifier to each point of the original point cloud for the evaluation. The recall and precision were measured similarly as follows:

\[
Pr_{\text{point}} = \frac{\sum G(i)}{|C_{\text{GT}}|} \tag{23}
\]

with

\[
G(i) = \begin{cases} 
1, & \text{if } c(i)_{\text{det}} = c(i)_{\text{GT}} \\
0, & \text{otherwise} 
\end{cases} \tag{24}
\]

where, \(c(i)_{\text{det}}\) describes the assigned plane for the \(i\)-th point in the segmented point cloud and \(c(i)_{\text{GT}}\) for the \(i\)-th point of the GT. \(|C_{\text{GT}}|\) is the amount of all the points of the GT planes. The precision measures the percentage of points being mapped to the correct plane.

\[
Re_{\text{point}} = \frac{\sum G(i)}{|C_{\text{det}}|} \tag{25}
\]

\(|C_{\text{det}}|\) depicts the amount of all points that are part of a plane assigned to a GT plane.

**Experimental Segmentation Evaluation**

We used measurements of the three straight stairways (Figure 1) to evaluate the different segmentation methods. The Quince robot equipped with a Hokuyo UTM 30LX LN LIDAR sensor was used for the data acquisition (Ohno et al. 2010). The LIDAR sensor was tilted 60° and rotated around the z-axis (Ohno et al. 2009). The sensor was calibrated with respect to the robot and the floor, because the robot was standing flat on the floor. The resulting point cloud contained approximately 1.4 million points and had a 360° round view with a vertical field of view of approximately 98°. We acquired a total of 50 different point cloud data of the three stairways from various positions, varying in the distance and angle to the stairways. The robot position was measured for each position by putting the ultra-precise FARO Focus 3D in the robot’s place and measuring the distance to suitable landmarks (i.e. different walls). The acquired point cloud was down-sampled using an octree to one point per 1 cm³. The downsampling reduced the point density in the very dense region, but had little to no effect on the sparse regions, thereby reducing the computational time during the latter processing steps. The point normals for the down-sampled cloud were estimated using the NormalEstimation function included in the PCL, except for the split and merge method.

As mentioned the distance and the angle to the stairways varied widely. Therefore, the precision and recall results also varied. Normally, the closer the robot to the stairway, the better the result. As for this dependence, an absolute
value does not inherit much information. For this reason we acquired the mean value for each segmentation method and used the region growing results as a baseline. Figure 15 presents the results of the segmentation evaluation. The region growing algorithm outperformed the other methods in every category. The precision results for the VoxelsAC were close to those of region growing, with only a 2.5 percent point (pp) difference. However, the plane recall was almost 8 pp worse. The split and merge performed worst, with an almost 20 pp decrease in the point precision. However, interestingly, all algorithms achieved very similar point recall results.

We also evaluated the runtime of the different segmentation methods. The results are displayed in Figure 16. It can be seen that the split and merge segmentation method had a much shorter runtime as it did not require any normal estimation for the points. VoxelsAC had almost a 1 s faster runtime than the region growing algorithm. For further processing, we chose the region growing algorithm for the segmentation because it provides the most accurate results. The increased processing time was just slightly higher than that of VoxelsAC. The segmentation time should also be viewed with respect to the acquisition time of roughly 35 s.

Note that the segmentation was done for unorganized point clouds. When using RGB-D cameras the image-like structure could be utilized to increase the speed of normal estimation and segmentation.

### Stairway Detection Experiments and Evaluation

Our method was evaluated for different stairways at Tohoku University, Japan. Figure 1 illustrates the stairways. The same acquisition setup as in section “Experimental Segmentation Evaluation” was used. The analysis for one LIDAR-acquired point cloud lasted approximately 8 s (i.e. 4 s for the preanalysis, 3 s for the segmentation, and 1 s for the main process). The robot was not moving during the acquisition process to have a defined reference point and allow the localization results to be evaluated. The ground truth data for the robot position were acquired by manually processing a reference scan of a highly accurate FARO Focus 3D scanner placed in the same position. The implementation and used dataset will be made openly available at http://www.rm.is.tohoku.ac.jp/stairway_dataset.html.

The robot was positioned at different locations relative to the stairways, and the stairway detection method was tested. Figure 17 shows the experimental results highlighting four different positions. Figure 17a presents the result of the detection from beside stairway I, while Figure 17b illustrates the result of the detection of stairway II with only the stair risers visible. Figure 17c shows the result of the detection of descending stairway III. Figure 17d demonstrates the detection of a spiral stairway. In the case of large occlusion as in Figure 11, where a large part of the right side of the stairway was not visible because of a wall, we were able to localize the stairway with an error of 2.8 cm through the use of a dynamic anchor point. The different stairways and robot positions, including the viewing angle, made it difficult to generate results in a compact and easily presentable manner. Therefore, we conducted two measurement series to present the coefficient and localization results. For the measurement series, the robot was positioned right in front of stairways I and III and moved around 50 cm away from the stairway after each measurement. We took ten measurements for each position and employed the optimizing competing initialization strategy for the experiment. The results are shown in Figure 18. Figure 18a and Figure 18b present the results for stairway I. Figure 18c and Figure 18d depict the results for stairway III. The localization error was split into $\Delta x$ for the error in the direction of the stairway orientation $\vec{s}$ and into $\Delta y$ for the error in the perpendicular direction.

Figure 23 illustrates the detected stairways with the detected...
up to 4 m for stairway I. A detection of stairway I at the distance of 4.5 m was only successful in four out of the ten measurements. Furthermore, at this distance only the upper stair risers were correctly detected, which lead to a large height error and to a high $\Delta z$ localization error. However, the results were still remarkable considering that stair risers were almost non-existent, and the robot’s angle of view was almost parallel to the stair treads’ planes. In contrast, stairway III, for which the stair risers were continuous, was detected for distances of up to 6.5 m. Larger distances were not measured because of structural limits (opposite wall) of the experiment place. The estimation error for the depth and the height of the steps in both measurement series was within 7 mm for all cases, except for the distance of 4.5 m for stairway I. The localization error for stairway I and stairway III was within 5 cm for all measurements, with the perpendicular error $\Delta y$ increasing for farther distances.

Figure 19 shows a comparison of the three different competing initialization strategies explained in subsection “Competing Initialization Strategies” for the height and depth estimation. The greedy strategy had considerably larger estimation errors. This strategy picks the first initialization that is able to detect a stairway; hence, the deviation is understandably higher. The thorough strategy shows considerably better results and the optimizing strategy further improves the detection results in almost all cases. Figure 20 presents the runtime of the different strategies.
The greedy strategy was the fastest with an average runtime a little below 0.4 s. The thorough strategy required approximately 0.9 s, while the optimizing strategy with a runtime of approximately 0.95 s required an additional 0.05 s to the thorough strategy.

The results using the optimizing competing initialization strategy, excluding the distance of 4.5 m for stairway I, were compared to those of the other stairway detection methods in Table 2. The comparison was of limited significance because the detection methods were tested for different settings and on different stairways. However, the quality of our results can be classified with this in mind. Our obtained results are better than those of other publications except for SV (Harms et al. 2015). We also achieved significantly better results than our previous work (Westfechtel et al. 2016) with the introduction of competing initializations. In particular, we could decrease the depth estimation error by more than 50%. Our results exhibited the lowest standard deviations, indicating a higher stability of our stair detection method. In addition, one has to keep in mind that our detection method was aggravated because the measurements were performed from a single view with distances of up to 6.50 m.

Large-scale Data

We also evaluated our method for large-scale point clouds. Figure 21 shows the result for the stairway detection in a large environment. It can be used to generate the stairway models for a layout plan of a building. Another application would be that a robot is teleoperated through a large-scale environment. During the teleoperation the robot acquires and merges the scan data generating a large-scale point cloud with the goal of commuting the same route the teleoperator drove for surveillance on a regular basis. The stairway within the route can be detected using our method. The estimated stair parameters allow the robot to autonomously traverse the stairways without any additional manual initializations. The stairways could also be used as landmarks for localization purposes.

Monash Dataset

We also evaluated our method on the Monash (Tang et al. 2012) dataset. The dataset consists of 89 depth images of ascending and descending stairways and another 58 depth images of urban environments without stairways to test for false positives. The images were acquired by a Kinect sensor mounted on a helmet worn by the experimenter.

Table 3. True positive rate (TPR) and true negative rate (TNR) on the Monash stairway dataset for different methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>TPR</th>
<th>TNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Pérez-Yus et al. (2014)</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Vlaminck et al. (2013)</td>
<td>100%</td>
<td>96.55%</td>
</tr>
<tr>
<td>Tang et al. (2012)</td>
<td>94.93%</td>
<td>98.98%</td>
</tr>
</tbody>
</table>

Conclusion

This study introduced a highly versatile plane-based stairway detection method. A stairway graph was initialized using
either stair treads or risers. The graph was then extended along the stairway’s direction, and the railing system was added to the graph. The stairway’s coefficients and localization can be estimated with high accuracy even at a distance of a few meters thanks to a competing initialization strategy. The discrepancy in the results between stairways I and III suggest that stairways with discontinuous stair risers are more difficult to detect. The method can detect not only ascending stairways but also descending and spiral stairways. The detection of descending stairways was only possible if the robot was positioned directly at the start of the stairway; otherwise, all stair treads were occluded because the low height of the LIDAR sensor.

While the algorithm is not explicitly able to detect other types of stairways like L-shaped stairways or stairways consisting of multiple flights, these stairways would most likely still be detected as a combination of straight and spiral stairways. For example, an L-shaped stairway could be detected as two distinct straight stairways or two straight stairways connected by a spiral stairway, depending on whether the corner is implemented as a landing or as winding steps. However, the concrete performance has still to be examined.

Our method is also applicable to different depth sensors and we showed results for two different LIDAR sensors and RGB-D cameras. We achieved good results for dense and sparse point cloud data, for point clouds with a limited field of view, and for large-scale point cloud data.

The limitation of our method is that we assumed a stair-graph model that does not change over the course of the stairway, meaning the depth and height of each step are the same for the whole stairway. While this holds true for most stairways, a few outdoor stairways come to mind where each step has different dimensions.

Acknowledgements

This research was supported by CREST: Recognition, Summarization and Retrieval of Large-Scale Multimedia Data and the Program for Leading Graduate Schools, Inter-Graduate School Doctoral Degree Program on Global Safety, by the Ministry of Education, Culture, Sports, Science and Technology.

References


Figure 23. Stairway detection of measurement series for stairways I + III.
Figure 24. Detected stairways in the Monash dataset including the visualization of the stair graph.