Fast Staircase Detection and Estimation with Multi-View Merging for Multi-Robot Systems

Prasanna Kettavarapalyam Sriganesh

CMU-RI-TR-23-51

July 28, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Matthew Travers, *chair*
Howie Choset
Sebastian Scherer
Charles Noren

Submitted in partial fulfillment of the requirements for the degree of Masters in Robotics.

Abstract

When robotic systems are deployed in the real world, they demand advanced mobility capabilities to operate in complex, three-dimensional environments designed for human use, e.g., multi-level buildings. Staircases have been an integral part of facilitating vertical movement in these three-dimensional environments. This work presents a novel method that enables mobile robots to locate and autonomously climb a range of different staircases. We develop a staircase detection algorithm that exploits viewpoints in a point cloud, making it possible to quickly detect staircases and estimate their physical parameters. Further, the algorithm can validate the number of traversable steps in the staircase by using a simple density metric to look for obstacles or damage. This staircase perception system runs on heterogeneous platforms in real-time that can each detect staircases and merge the detections. We present results wherein a wheeled robot works with a quadrupedal system to detect different staircases quickly. The performance of this staircase detection system is compared to the current state-of-the-art detection algorithm. We show that our approach significantly increases the speed of detections by two orders of magnitude without compromising the accuracy of parameter estimation.

Acknowledgments

I sincerely thank my advisor, Dr. Matthew Travers, for their guidance and support throughout my time at Carnegie Mellon University. His expertise and knowledge have been invaluable, and I am grateful for their patience and encouragement.

I would also like to thank my committee members, Dr. Howie Choset, Dr. Sebastian Scherer, and Charles Noren, for their valuable feedback and suggestions. Their insights have been tremendous and have helped me ask the right questions and make quality improvements to my thesis work.

I am incredibly grateful for the consistent support and advice my project scientist, Dr. Bhaskar Vundurthy, provided. He has been instrumental in guiding and motivating me in the right direction whenever I needed a nudge.

I want to acknowledge all the support I received from my teammates on the MMPUG project. I especially want to thank my teammates Namya, Burhan, Adam, and Jay for helping me collect data around the campus and set up experiments every time I saw a new staircase. The results of the thesis would only be complete with their help. I am also thankful for all the technical advice that my teammate Joshua Spisak has provided over the last two years. His insights have helped me become a better coder overall.

I thank all my friends and colleagues from the Robotics Institute and the Biorobotics lab. You all have been a great group of people to interact and work with. Special thanks to one of my closest friends, Winnie; I appreciate your support over the last two years.

I want to thank my mom and family for supporting me throughout my life. I would not be in this position today without Amma's constant backing that I currently rest on today.

Lastly, I am thankful to all of our robots, without whom these algorithms would not serve any purpose.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

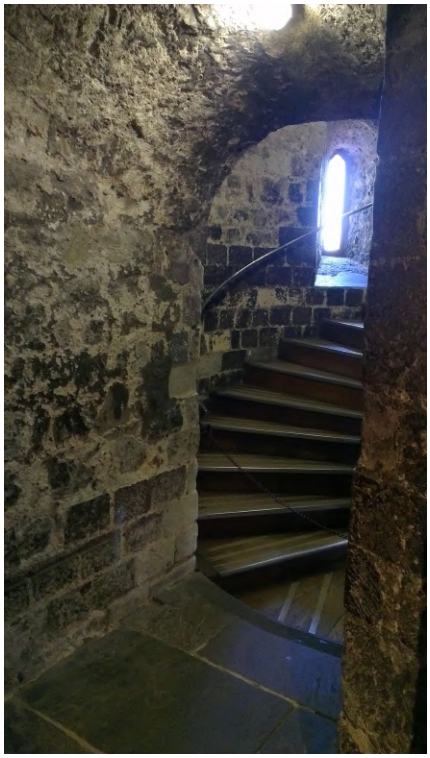
Staircases have been a prominent architectural element in almost every structure throughout human history. They have been present for at least 4,000 years[3], as seen on the Ziggurat of Ur in ancient Mesopotamia (2000s B.C.E.). They can also be seen on the pyramids in Egypt, the Mayan pyramids, and many other ancient structures (depicted in Fig. 1.1). Since then, staircases have evolved and featured sophisticated designs, as evidenced by the spiral staircases in the Tower of London (1000s C.E.). Staircases are versatile as they connect different levels of buildings and facilitate movement in the third dimension. Consequently, they continue to play a major role in today's urban landscape.

In environments with staircases or otherwise, mobile robots have been predominantly used to perform dangerous or hazardous tasks. For example, the 2011 Fukushima Daiichi nuclear disaster saw mobile robots being used to assess the damage [21]. Additionally, they help enable access to places that are especially difficult for humans to reach, aiding search and rescue operations. With mobile robots getting increasingly sophisticated, they are now being deployed in a wide variety of real-world environments [2] that were built for humans. These environments are complex and usually consist of multi-level buildings. For robotic systems to operate in such environments, they need the capability to traverse staircases autonomously.

There are various commercially available robot platforms that are reliable, ac-

(a)

(b)



(c)

Figure 1.1: **Staircases from historical structures**: (a) Staircase on the Ziggurat of Ur (Partially Restored), (b) Staircases on El Castillo Pyramid in Mexico, (c) A Spiral Staircase in the Tower of London

cessible and can climb staircases on command. This includes quadruped platforms like Boston Dynamics' Spot, the quadruped robots by Unitree Robotics, and various tracked robots like Tready by HEBI Robotics (all depicted in Fig. 1.2). However, most of these platforms need trained human operators which makes them difficult to use in emergency response operations. On the other hand, having a way to autonomously navigate staircases will facilitate reliable and easier operation of these robot platforms.

In the 2020 DARPA Subterranean Challenge, teams of heterogeneous robots comprising wheeled, legged, and aerial robots explored unknown urban spaces. Challenges like these typically operate in complex environments composed of underground spaces and multi-level buildings. In environments where remote operation is unreliable, robots need to perceive, analyze and make decisions autonomously. Navigating stair-

(a) (b) (c)

Figure 1.2: **Different Robot Platforms capable of traversing staircases:** (a) Boston Dynamics's Spot Robot (b) Unitree Robotics Go1 Platform (c) Tready tracked robot from HEBI Robotics.

cases is a significant component of handling multi-floor environments. Knowledge of staircases from wheeled robots that scout enables legged robots to navigate staircases and explore environments that are not reachable by the scout robots. Achieving such real-time coordination with advanced mobile robots requires a fast, robust method of staircase detection.

Staircase detection and characterization are particularly challenging for autonomous robot perception systems. Diversity in staircases, for instance, spiral or hollow staircases, makes autonomous detection more challenging. Existing approaches can be too slow when implemented in compute-constrained mobile robots. These approaches also do not address scenarios where information is obtained about different parts of the staircases. This is the case when a robot is climbing a long staircase and new information is obtained as the robot climbs higher. This motivates the need for a fast and robust staircase detection system that supports fusing detections from multiple viewpoints.

1.2 Thesis Statement

This thesis aims at creating a computationally efficient and fast algorithm that can detect all kinds of staircases, including ascending, descending, hollow and spiral stairs. As opposed to conventional techniques, the algorithms put forth in this thesis

exploit projections from different viewpoints in a point cloud to improve the speed of detections. Figure 1.3 shows examples of different staircases we want to detect.

Further, we aim to accurately indicate the number of steps in a staircase and estimate the size of the steps to help assess if the staircase can be traversed by the robot. In addition to detecting and estimating staircases, we also focus on predicting how many steps are traversable. We ensure the system can still run in real-time using a simple density measure to predict traversability which aids in search and rescue operations. Stairs in disaster sites often have debris on them (Fig. 1.3 (e)) or are

(a) Ascending Staircase (c) Hollow Staircase (e) Staircase with debris

(b) Descending Staircase (d) Spiral Staircase (f) Damaged Staircase

Figure 1.3: Different types of staircases we intend to perceive

broken (Fig. 1.3 (f)). These dilapidated staircases prevent the robot from being able to ascend or descend the staircase and, in worse cases, cause the robot to tumble, rendering them unusable.

One of the end goals of this work is to have a staircase perception pipeline that can be deployed on multiple robots operating together. To this end, we investigate methods that enable merging between multiple detections of a single staircase. If a new detection is spatially close to a previously detected staircase, we avoid duplicates by finding a common pivot stair between the two detections and merging them. Fusing information from different robots or viewpoints allows for a more accurate estimation of a staircase, thereby enabling a better assessment of staircase traversability. Finally, we propose an architecture for a complete staircase perception system implemented on two heterogeneous robots, including a legged robot that can traverse staircases.

1.3 Contribution

The main contributions of this work are as follows. We first present an algorithm to detect different types of staircases in real-time and estimate their geometry. This

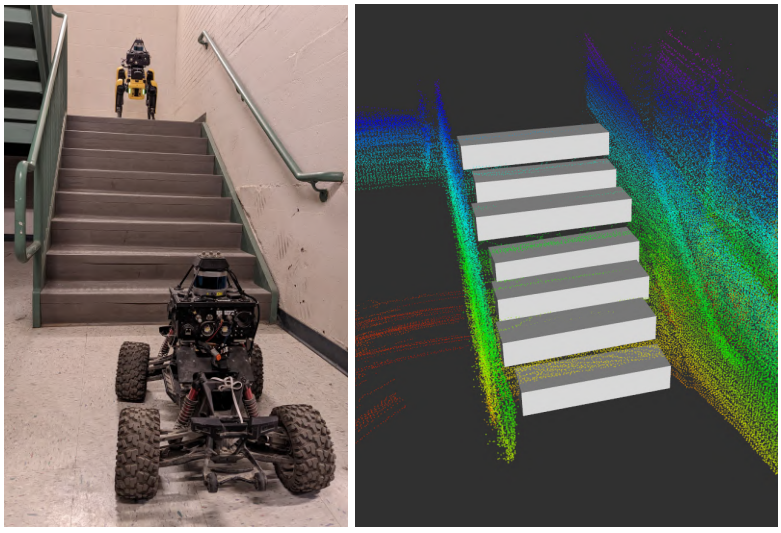


Figure 1.4: Two heterogeneous mobile robots around a staircase (left), A fully detected staircase by the algorithm as shown by the white marker (right)

algorithm uses 3D point clouds as input to identify the staircase and its specific characteristics, such as the number of steps, the step depth, width, and height. We also propose a way to predict if the steps are traversable by predicting if any of the stairs have debris or damage on it in real-time.

We next present a simple algorithm that can merge two detection instances of the same staircase obtained by different robots or different viewpoints. This algorithm allows the robots to detect long staircases that a single robot's viewpoint would not be able to capture otherwise. This also allows for better global registration of multiple staircases in an environment without having duplicate detection. Figure 1.4 shows an example of two heterogeneous robots looking at a staircase and the resulting detected staircase obtained from merging information from both robots.

Lastly, we present a complete staircase perception pipeline deployed on two heterogeneous robots. These robots can detect multiple staircases in an environment in real-time while sharing information about the staircase to fuse them. This perception pipeline can also detect and fuse long staircases that span over two stories that cannot be detected in a single time step.

Chapter 2

Staircase Detection

2.1 Background

There has been considerable research carried out in the field of staircase detection. All the methods can be classified based on the modality of the input. The two primary modalities are either images or point clouds. Although both can be used to detect staircases, estimating the parameters of the detected stairs, such as their height, depth, and slope, requires depth information.

One of the first image-based approaches to staircase detection was put forth by Cong et al. [4]. The primary idea of detecting stairs from an image is to capture the edges formed by stairways in image space. There have been multiple attempts that use different computer-vision techniques to segment the stair edges and use that to detect staircases from images [23][6][15]. Murakami et al. [8] used a combination of RGB and depth images to segment edges and successfully detected both ascending and descending staircases.

Even contemporary learning-based algorithms have been used to detect staircases from images. Ilyas et al. used a convolutional neural network (CNN) to predict a bounding box around a staircase from image inputs [7]. Patil et al. used a tiny YOLOv3 network to detect staircases and proposed a statistical image filter to enable a robot to climb up a staircase [11]. These methods can only be used to detect staircases, and they do not provide any way of estimating whether or not a staircase is traversable.

The advantage of using image-based methods is the speed of the detections. However, reliance on images makes the entire system environment dependent. The system's robustness goes down if the staircase is very reflective or in low-light conditions. Image-based methods that employ machine learning require massive datasets to work reliably, and their performance is as good as the training dataset. Moreover, the lack of depth information in RGB images prohibits accurate geometry or location estimation. The camera positioning also impacts the field of view of detection, and image-based methods can be tricked into classifying parallel line patterns as stairs.

When using pointclouds as the modality, staircases can be detected by segmenting planes that form the staircase. Using point clouds also aids in estimating the staircase's geometry and location, which can be used as an input for navigation. Point clouds are typically collected from LiDAR sensors or depth cameras. Oßwald et al. [10] first explored and experimented on two different plane segmentation methods to detect staircase risers.

Different variations of Random Sample Consensus (RANSAC) have also been extensively used to segment planes and then detect staircases [18][13]. RANSAC-based methods have been used to estimate staircase location for navigation by different robots [17][20][16]. Fourre et al. [5] even devised a way to localize industrial stairways with no risers. Even though RANSAC is a simple and efficient algorithm, it is non-deterministic. It does not guarantee a best solution or have a fixed time-bound. This is not a great way to detect stairways in scenarios that are time critical. These algorithms also have prerequisites on which part of the staircase needs to be visible to the sensor.

Westfechtel et al. [19] were the first to successfully achieve detection and estimation of staircases in all directions (360^o). They compared three different segmentation methods to detect planes in LiDAR point clouds and used a graph-based strategy to detect staircases of all types. Their estimates of the staircase location and the geometry were the best among all the previous work. Consequently, we can consider this work to be the current state-of-the-art for staircase detection. Although, their biggest drawback was the speed of the detection. The robot was expected to be static during the entire process, and plane analysis took around 4-8 seconds. This is entirely not feasible in search-and-rescue scenarios where every second is crucial.

This method also does not address scenarios when the staircases might be broken or filled with debris which is necessary to predict the traversability of a staircase.

None of the plane-based methods discuss detection speed, which is essential, especially to allow for deployment on real robots. Additionally, all previous work treats staircase detection as a one-off algorithm. They do not address cases where robots can only see part of the staircase at one time and combine multiple detections to be able to perceive the entire staircase. There has not been much research conducted on multi-robot scenarios. Two robots with different viewpoints can help achieve a better estimation of a staircase by fusing both instances. All of these aspects are important to enable autonomous navigation of staircases.

2.2 Detection Pipeline

We present an algorithm that detects staircases from 3D point clouds and accurately estimates the staircase parameters such as location, height, and depth. The input 3D point cloud can either be obtained by stacking multiple LiDAR scans into a voxel grid, or can directly use RGB-D sensors to get a dense 3D point cloud. The main intuition behind our algorithm is to segment only the edges formed by the staircase surfaces. The pipeline has three major steps: pre-processing, segmentation, and detection.

2.2.1 Pre-Processing

Let's first define the robot's frame. As shown in Fig. 2.1, the x-axis coincides with the robot's heading, the z-axis faces upwards and y-axis is to its left. The input point cloud (accumulated from multiple LiDAR scans) is converted into a 3D voxel grid with fixed voxel size (leaf size). Using the assumption that the robot has a fixed height, we also compute the location of the ground. This input point cloud is shown in Fig. 2.2.

The first step in pre-processing is to perform a top-down projection of the point cloud to get rid of vertical surfaces. This step eliminates all the points that are not visible from a top view of the point cloud. All the planes perpendicular to the ground are reduced to a line parallel to the ground while the other planes are intact. We apply this technique to the point cloud shown in Fig. 2.2 and the resulting point

Figure 2.1: Robot and its reference frame

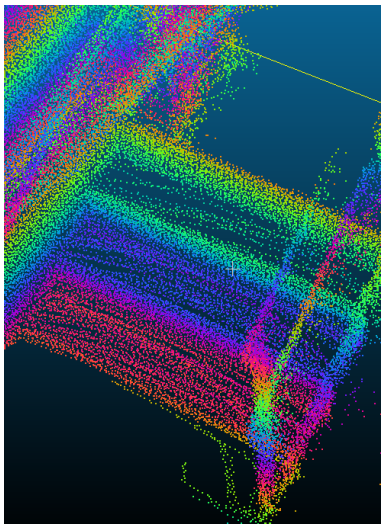


Figure 2.2: Input point cloud of the staircase

cloud is shown in Fig. 2.3.

The next step is to perform a range projection from the robot's viewpoint. To do this, we organize this point cloud into a 2D array around the robot using cylindrical coordinates. The z-axis is discretised into rows of the array, while the columns are

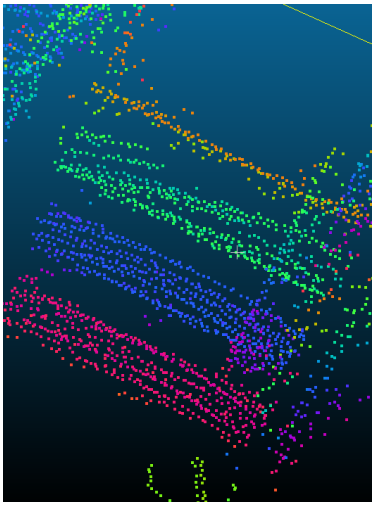


Figure 2.3: Point cloud after first step of pre-processing (Top-down projection)

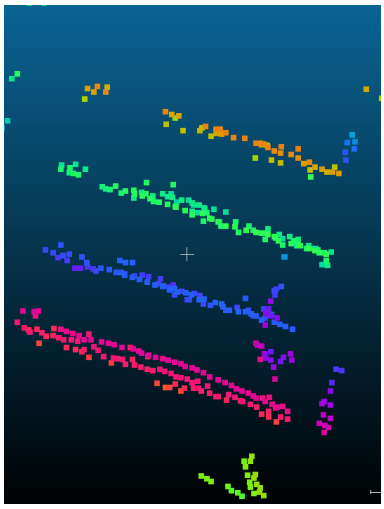
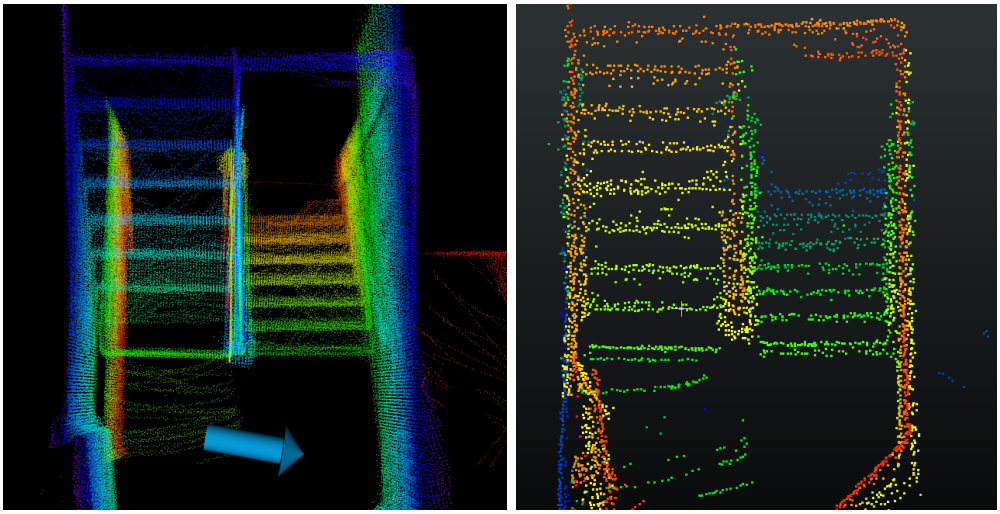


Figure 2.4: Result point cloud after the second pre-processing step (Front/Range Projection)

indexed using the azimuth angle θ which is given by $tan^{-1}(\frac{y}{x})$. We also compute the range ρ $(= \sqrt{x^2 + y^2})$ of all the points. In each row (points with similar z), we retain the proximal points (smallest range value ρ for every θ) if the row is above the ground as these points correspond to the front edge of an ascending staircase. In the case when the row is below the ground, we retain all the distal points (highest range value ρ for each θ) as it corresponds to the front edge of a descending staircase.

This pre-processing step reduces all the horizontal planes (stairs) into a single edge. The two pre-processing steps should reduce the 3D point cloud to points that belong to the stair edges. Figure 2.4 shows the final processed cloud for a regular ascending staircase. Figure 2.5 shows a top-down view of the processed cloud when there are both ascending and descending staircases in the view.



(a) Top-down view of the input point cloud with two staircases

(b) Top-down view of the output point cloud after pre-processing stage

Figure 2.5: Result of pre-processing steps when input point cloud has ascending and descending staircase. Left image shows the input point cloud. Right image shows the result after pre-processing.

2.2.2 Segmentation

As mentioned before, the point cloud is organized into a 2D array structure that can be thought of as an unwrapped cylinder with the robot at its center. Each row index

corresponds to a fixed z height, and the column indices represent the azimuth angle spanning from $-180°$ to $180°$. Consequently, a full row of this array can be treated as single 2D laser scan. Furthermore, since all points in each row have approximately equal z-values, we can fit lines in the xy space and add the height information later.

To extract lines from a 2D laser-scan, we use a modified version of Iterative-End-Point Fit [14][9]. Given N points in a scan, we fit a line between the first and last points. We then find the point with the maximum distance to this line. If the distance is more than a threshold (d_p), the points are split into two groups until the number of points in each set is greater than a limit (N_{min}) or if all the points are at a distance less than a threshold d_p.

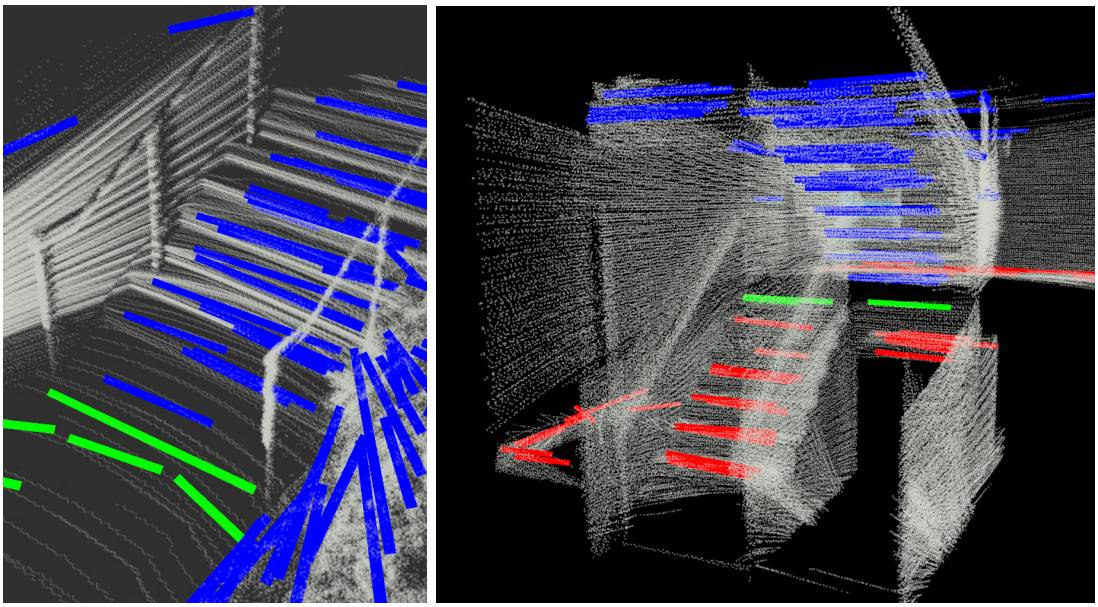
Typically, if all the points are close to the line, the loop stops and returns the line. We modify this behavior to use weighted line fitting to estimate a line with all the points. Weighted line fitting is a version of least square line fitting with the uncertainty of a point used as weights. The uncertainty models the inaccuracies from the sensor being used to obtain the point clouds. We refer to [12] for a detailed discussion of the problem.

The merging is usually done if the points forming the two lines are collinear. We modify this to use the merging criteria provided by weighted line fitting. The main advantage of weighted line fitting is that it outputs a covariance for each line. This allows us to estimate the similarity between two lines better and merge it better. We represent each line fit using this method by the parameters below:

- 3D start point of the line - $\bar{p_s}$
- 3D end point of the line - $\bar{p_e}$
- Orientation α - angle between the line with the xy plane
- Covariance Matrix P_L

This line segmentation algorithm is run separately on each row (2D scan) of the point cloud array. All the resulting lines are added to a single list $\mathcal{L}$. As a result, we get lines parallel to the ground plane. Since staircase edges also have to be parallel to the ground, it eliminates the need to check for line altitudes. If the point cloud has Z rows and T columns, the time complexity of this algorithm is $O(ZTlog(T))$.

After all the lines are segmented, we group them into three groups based on their height in the 3D space. Assuming that the robot has a fixed height, it is trivial to



(a) (b)

Figure 2.6: Segmented lines, grouped based on height. Blue lines indicate above ground, green indicate on the ground and red lines indicate below the ground. Left image shows lines segmented in a scene with ascending staircase. Right image depicts lines segmented in a scene with both ascending and descending staircases

group lines that are above the ground ($\mathcal{L}_{ag}$), below the ground ($\mathcal{L}_{bg}$), and that are part of the ground plane ($\mathcal{L}_g$). This makes it easier to search for staircases that are ascending and descending. Figure 2.6 shows the lines detected by our algorithm in that scene. Blue represents lines above the ground, the green represents lines on the ground plane and the red lines indicates lines below the ground.

2.2.3 Detection

We start searching for staircases in the segmented lines from the previous step. We first describe the model of our staircase as follows. Figure 2.7 also describes these parameters.

- Step Height, h
- Step Depth, d
- Step Width, w

- List of Lines $\mathcal{L}$, Each line l_i represents a stair and is described by 3D start ($\bar{p_s}^{(i)}$) point, 3D end ($\bar{p_e}^{(i)}$) point and it's orientation ($\alpha^{(i)}$) in the XY plane
- Stair Slope, $\phi = tan^{-1}(h/d)$

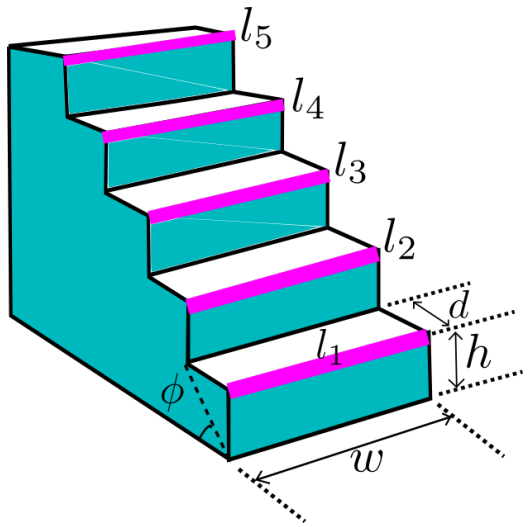


Figure 2.7: Staircase model with its parameters labelled, pink lines represent stair edges to be segmented.

In order to start grouping a set of lines that form a staircase, we define the parameter limits of a staircase based on the standards set by OSHA[1].

Definition 1 *Given two lines $l_1(\bar{p_s}^{(1)}, \bar{p_e}^{(1)}, \alpha^{(1)})$ and $l_2(\bar{p_s}^{(2)}, \bar{p_e}^{(2)}, \alpha^{(2)})$, we compute the height h_i given by the difference in z between the lines, the depth d_i given by the xy distance between the lines and the slope $\phi_i = tan^{-1}(h_i/d_i)$. Any two lines that satisfy the five conditions is defined as a* ***stair****:*

1. $0.11\ m \leq h_i \leq 0.30\ m$
2. $0.15\ m \leq d_i \leq 0.45\ m$
3. $25° \leq \phi_i \leq 60°$
4. $|\alpha^{(1)} - \alpha^{(2)}| \leq 10°$
5. *There is no other line in between l_1 and l_2 that satisfy the above conditions* ■

Depending on whether we want to detect an ascending or descending staircase, we pick the appropriate list ($\mathcal{L}_{ag}$ or $\mathcal{L}_{bg}$) of segmented lines. The algorithm to detect staircases has two parts, initialization, and extension. In the initialization stage, we create a subset of lines that are below 2.5 h_{max} in the z-direction called the

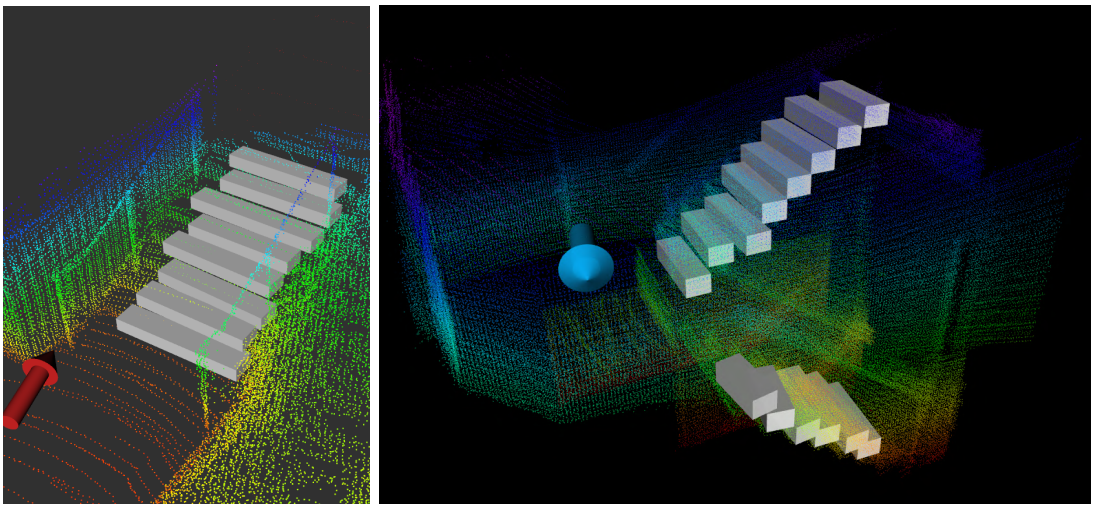
Algorithm 1 Detect ascending staircase from list of lines

Input: Set of above ground lines - $\mathcal{L}_{ag}$
Output: Staircase $\mathcal{S}$ with N lines

Initialize list $\mathcal{L}_I$ with all lines ($\mathcal{L}_{ag} \leq 2.5 h_{max}$)
while $\mathcal{L}_I$ not empty **do**
    Initialize empty staircase set $\mathcal{S}$
    $stair_{init} \leftarrow false$
    while $stair_{init}$ is $false$ **do**
        Pick 2 lines, l_1, l_2 from $\mathcal{L}_I$
        if l_1 and l_2 form a stair (Definition 1) **then**
            $stair_{init} = true$
            Add l_1 and l_2 to S
        end if
        if No valid pair exits **then**
            return $\mathcal{S}$
        end if
    end while
    Reorder lines in $\mathcal{L}_{ag}$ by ascending order of height(z)
    for Every line l_{curr} in $\mathcal{L}_{ag}$ **do**
        $l_{prev} \leftarrow$ last line in $\mathcal{S}$
        if l_{curr} and l_{prev} form a stair (Definition 1) **then**
            Add l_{curr} to Set $\mathcal{S}$
        end if
    end for
    if Total stairs in $\mathcal{S} \geq 4$ **then**
        return $\mathcal{S}$
    else
        Remove the initialized stair lines from $\mathcal{L}_I$
    end if
    Initialize empty staircase list $\mathcal{S}$
end while
return $\mathcal{S}$

initialization list. Here h_{max} is the maximum step height allowable (0.3 m). We then look for two lines in this initial list that can form a staircase as per Definition 1. If two such lines exist, we add them to a staircase list $\mathcal{S}$. Although, the limit of 2.5 h_{max} performed well on the data, it can be replaced as a tunable hyper-parameter if necessary.

Once we have the first two steps of the staircase, we perform an extension. For



(a) (b)

Figure 2.8: Detected staircase, each white marker corresponds to a stair. Left image shows detected ascending staircase with robot location shown by the red arrow. Right image shows both ascending and descending staircases with robot location depicted by the blue arrow.

every line in the increasing z-direction, we check if it forms a staircase with the previous line in the staircase set using Definition 1. If it complies, we add the new line to the set and repeat the extension until all lines are exhausted. If there are more than four lines in the staircase list, it is a successful staircase detection, and use this to estimate the staircase geometry. The minimum number of stairs can also be set by the user to detect smaller staircases.

If the initialized staircase does not extend, i.e., the set only has two stairs, we remove the lines from the initialization list and repeat the process until the initialization list is empty. The pseudo-code of this method is presented in **Algorithm 1**. To detect descending staircases, we switch the set of lines to L_{bg} and run the same algorithm, but the lines are checked in decreasing z-direction. Figure 2.8 shows the detected staircases in two scenarios. The first one only has ascending staircase and the second scenarios has two staircases, one ascending and one descending.

2.3 Staircase Estimation and Validation

Once the detection algorithm returns a positive detection, the next step in the pipeline is to predict if a robot can traverse the staircase. This can be done using two things - estimation and validation. In the estimation step, we predict the parameters of the staircase such as its height, depth and width. These parameters can serve as a check to identify if a robot can climb a staircase, as most robots have a known limit on the stepping height. In the validation step, we try to estimate the area on the stair that is actually traversable. This is necessary as stairs may have positive obstructions such as an eruption or a piece of debris resting on the surface. Additionally, negative obstructions, such as cavities or other damaged areas, may exist on the surface of the stair that cause it to be unable to support the weight of the traversing robot

2.3.1 Estimation

The detection algorithm returns a list of lines that correspond to each stair edge of the staircase. As noted before in Fig. 2.7, we can estimate the parameters of the staircase using the location of these lines. Given a list of stair lines $\mathcal{S}$ with k lines and assuming that all stairs in a stairway have similar dimensions, we compute the parameters as shown below:

$$\textit{stair height, } h = \frac{\sum_{n=1}^{k-1} ||\bar{p_s}^{(i+1)} - \bar{p_s}^{(i)}||_z + ||\bar{p_e}^{(i+1)} - \bar{p_e}^{(i)}||_z}{2k}$$

$$\textit{stair depth, } d = \frac{\sum_{n=1}^{k-1} ||\bar{p_s}^{(i+1)} - \bar{p_s}^{(i)}||_{xy} + ||\bar{p_e}^{(i+1)} - \bar{p_e}^{(i)}||_{xy}}{2k}$$

$$\textit{stair width, } w = \frac{\sum_{n=1}^{k} ||\bar{p_s}^{(i)} - \bar{p_e}^{(i)}||_{xy}}{k}$$

$$\textit{where } ||p_1 - p_2||_z \textit{ is the 1D distance in z axis}$$

$$\textit{where } ||p_1 - p_2||_{xy} \textit{ is the euclidean distance in xy plane}$$

We can also define the location of the staircase ($\bar{S_p}$) as the center of the first stair. The first line is picked relative to an ascending or descending staircase. This location is useful as it can be sent as the first target waypoint for a robot to climb that staircase. This can be computed as shown below:

$$\bar{S_p} = (\bar{p_s}^{(1)} + \bar{p_e}^{(1)})/2$$

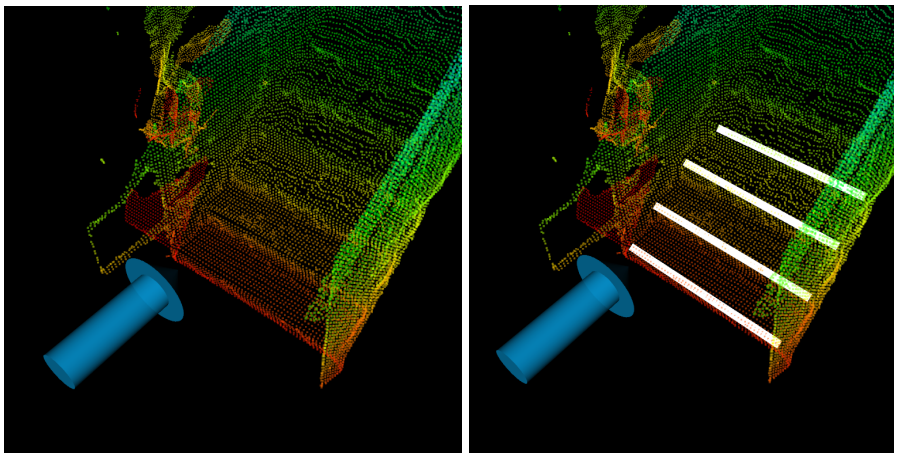
These equations for estimation as based on the assumptions that the stair parameters are equal over the entire staircase. In other scenarios, the stair edges still have sufficient information to estimate the height, depth and width of each step and a appropriate model can be in those cases.

2.3.2 Validation

The main intuition behind the validation step is to estimate the free space on a stair surface using a simple point density measure. We want to identify the number the points that reside close to the surface of the stair and compute its ratio with the number of points that a stair would have if it was healthy. This ratio provides a good indication of whether the surface of the stair is freespace.

As noted previously, our algorithm returns a set of lines that correspond to the stair edges when it detects a staircase. An example is depicted in Fig. 2.9. We also have the information about the stair depth and width as given by the estimation step. We can combine these information to define a region-of-interest(ROI) on each stair to compute the density metric. The ROI is defined as a thin cuboid whose edge overlaps with the stair edge. The depth and width of the ROI is equal to the stair's depth and width, and the depth of the ROI is set to the voxel size (5cm in our case) to account for noise. Figure 2.10 shows the computed ROI for each stair in the detected staircase.

Once the ROI has been computed, we simply count the number of points that exist within each ROI in the original input voxel grid. We can also use the top-down projection point cloud that was obtained by performing the first pre-processing step. We next also compute the number of expected points using the size of the ROI and the voxel size of the voxel grid. This is then used to compute the ratio as shown below:



(a) (b)

Figure 2.9: Staircase Detection of regular staircase: (a) The input point cloud to the algorithm (b) The stair edges that correspond to the detection as reported by the algorithm

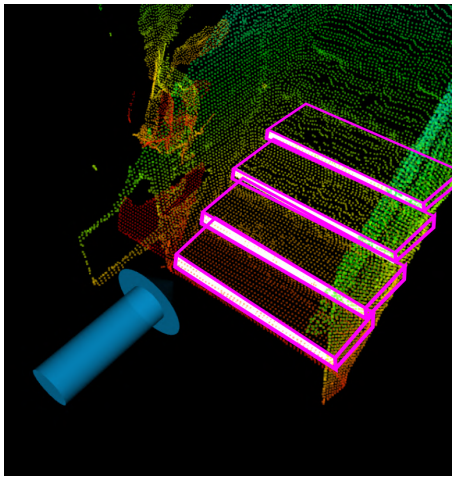


Figure 2.10: Point cloud of the scene, with pink cuboid showing the region of interest calculated using the stair edge where surface density check will be performed

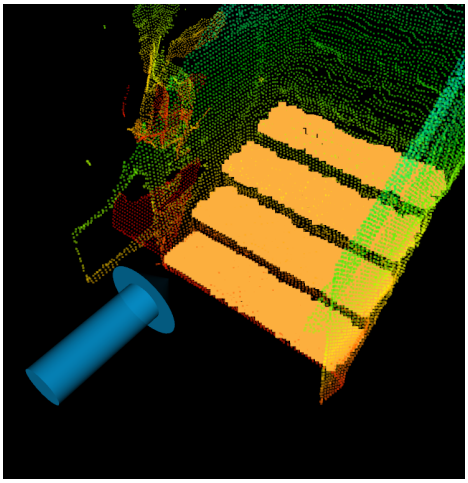


Figure 2.11: Result point cloud after validation, where orange points indicate the points on the surface that are counted to be traversable space on the stair

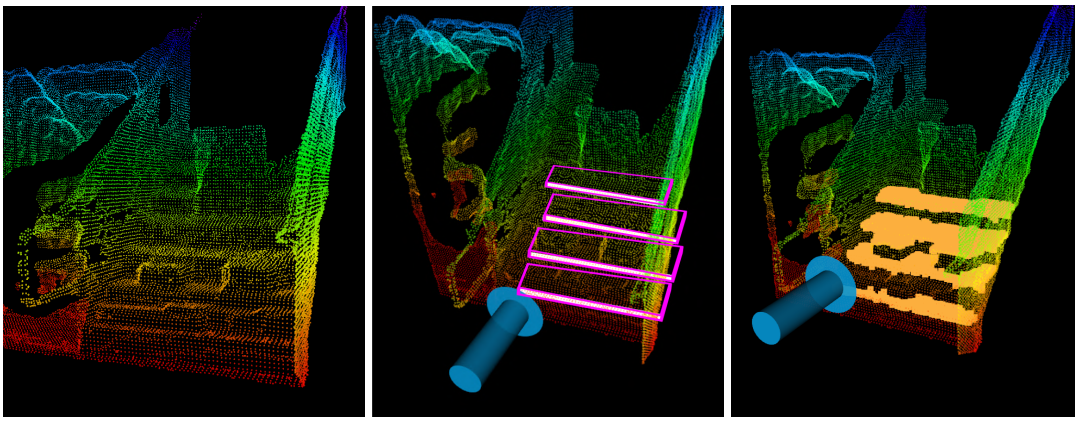
$$Stair\ surface\ density\ ratio = \frac{Number\ of\ points\ in\ ROI\ of\ stair}{Expected\ number\ of\ points\ in\ ROI}$$

where,

$$Expected\ number\ of\ points\ in\ ROI = \frac{Depth\ of\ ROI}{voxel\ size} \times \frac{Width\ of\ ROI}{voxel\ size}$$

We label a stair as healthy if the ratio is greater than 0.7 and if the ratio is less than 0.15 we infer that the stair is non existent. If the ratio is in between 0.7 and 0.15, we label it as a damaged stair as it infers that even though the stair surface exists, it either has some debris or cavities implying that the stair cannot be traversed. The output surface points after this validation step are in shown using orange points in the Fig. 2.11. Figure 2.12 provides an example for the validation pipeline when there are actually debris on the staircase. The surface points (orange points) shown on the right most image of Fig. 2.12 are incomplete as compared to Fig. 2.11.

The use of a point density metric keeps the algorithm lightweight and still achieves real-time performance. However, this step expects the point cloud to be dense and might not provide correct results if parts of the point cloud around the staircase is incomplete. This step only acts as validation step as the output is discrete on whether the stair surface can be traversed. This step does not localize or nor provide any information about where the free space on the stair exists.



(a) (b) (c)

Figure 2.12: Staircase Detection of staircase with debris: (a) The input point cloud to the algorithm (b) The detected stair edges shown as white lines that correspond to the detection as reported by the algorithm with pink cuboid shows the region of interest for surface density checks (c) Validated point cloud with orange points indicating traversable region on the staircase

Chapter 3

Multi-Detection Merging

The staircase algorithm presented previously can detect staircases locally. It will detect a staircase that is present in the input point cloud. However, it does not have any context about previously detected staircases, nor does it understand that staircases can be longer than the point cloud's field of view. In this chapter, we address these scenarios where a detected staircase can be globally registered without duplicates and also improve the estimation by using information across multiple detection instances.

3.1 Handling Multiple Staircase Detections

The first step in handling multiple staircases is transforming the locally detected staircase into a global frame. We do this using the odometry of the robot. Once we have the staircase detection information in the global frame, we next identify if this detection belongs to an already existing registered staircase or if this is a new staircase.

If the number of registered staircases is empty, we directly register the incoming detection and assign it an ID which helps query it later. If multiple staircases are registered, we match the new staircase to the existing ones using a combination of different distance metrics between two staircases. We first estimate specific parameters for each detected staircase to compute these metrics. We represent a staircase's location using the following parameters.

$$\begin{aligned}
\textit{Staircase Start}, s_{start} = [x_s, y_s, z_s] && \textit{Given by center of starting stair edge} \\
\textit{Staircase End}, s_{end} = [x_e, y_e, z_e] && \textit{Given by center of last stair edge} \\
\textit{Staircase Center}, s_{center} = [x_c, y_c, z_c] && \textit{Given by the average of start and end} \\
\textit{Staircase Direction}, \psi = tan^{-1}(\frac{y_e - y_s}{x_e - x_s})
\end{aligned}$$

We can only merge two detections if they belong to the same staircase. To predict if two detections belong to the same staircase, we use the parameters of the staircase to check if they are in the same spatial location. We define this step as staircase matching. Any two staircases with locations $A(s^{(1)}_{start}, s^{(1)}_{end}, s^{(1)}_{center}, \psi^{(1)}))$ and $B(s^{(2)}_{start}, s^{(2)}_{end}, s^{(2)}_{center}, \psi^{(2)})$ are considered to be a match if they satisfy the three conditions below.

1. $|\psi^{(1)} - \psi^{(1)}| \leq 30^{\circ}$
2. Distance from any point in B to the line formed by $s^{(1)}_{start}$ and $s^{(1)}_{end} \leq K_1$
3. Euclidean distance from any point in A to any point in $B \leq K_2$

The first condition ensures that the two staircases face the same direction. The second condition tries to estimate the closeness of the second staircase to the first one. Using the line distance provides a better estimate than the Euclidean distance, as the latter is spherical and staircases are planar in nature. The last condition ensures that both staircases are close to each other and have an overlap with at least some stairs, as the second condition cannot guarantee that. In the case of spiral staircases, only the third condition needs to be satisfied to be considered a match, as the line metric does not hold any meaning.

The thresholds K_1 and K_2 were picked after experimenting with different merging scenarios. The value of K_1 was set to $1.25m$ and K_2 was set to $1.5m$. These thresholds can also be made more granular by splitting the xy and z components for distances. This filtering is necessary to match the incoming detected staircase with an existing staircase in memory. These two staircases are then passed to the merging algorithm to perform stair matching, after which the merging is complete.

3.2 Merging Algorithm

Once the incoming detection is matched, we use the algorithm presented in this section to merge the two different detection instances of the same staircase. The detections can be either by the same robot during different time instances or by using different robots with different viewpoints. We would first like to define a criteria to classify two individual stairs as similar.

Definition 2 *Given two stairs* $l_1(\bar{p_s}^{(1)}, \bar{p_e}^{(1)}, \alpha^{(1)})$ *and* $l_2(\bar{p_s}^{(2)}, \bar{p_e}^{(2)}, \alpha^{(2)})$*, we first compute the height* h_i *given by the difference in* z *between the stairs, the depth* d_i *given by the* xy *distance between the stairs. Any two stairs that satisfy the following three conditions are considered to be the* ***same stair****:*

1. $h_i \leq 0.05\ m$
2. $d_i \leq 0.05\ m$
3. $|\alpha^{(1)} - \alpha^{(2)}| \leq 10^\circ$ ■

Algorithm 2 describes the way to combine two detection instances. The main idea is to find the location of the intersection for two detections. We do this by iteratively finding two lines, one from each detection, that are similar. This common

Algorithm 2 Merging staircases

Input: Two Staircase Detections $\mathcal{S}_a$ with k stairs and $\mathcal{S}_b$ with m stairs with $k \leq m$
Output: Fused Staircase $\mathcal{S}$ if successful, else NO_MATCH

1: $match \leftarrow false$
2: **for** Every stair l_a in $\mathcal{S}_a$ **do**
3: **for** Every stair l_b in $\mathcal{S}_b$ **do**
4: **if** l_a and l_b are similar stairs (Definition 2) **then**
5: $match \leftarrow true$ ▷ Stair match found
6: $i \leftarrow index(l_a)$ and $j \leftarrow index(l_b)$
7: **break** and **goto** 10
8: **end if**
9: **end for**
10: **end for**
11: **if** $match$ is $false$ **return** NO_MATCH
12: $\mathcal{S} \leftarrow \{l_1, \cdots, merge(l_i, l_j), merge(l_{i+1}, l_{j+1}), \cdots, l_m\}$ where 2 stairs are merged using [12]
13: Re-estimate parameters of Staircase $\mathcal{S}$ and **return** $\mathcal{S}$

line (stair) between detections will act as an anchor point to merge the two detections. Any technique can be used to merge two individual stairs once the anchor point is found. In our implementation, we average the two lines that correspond to the stair. Once merged, if any more stairs do not have a corresponding pair, it is appropriately appended to the end/start of the list. The staircase parameters are re-estimated after this merge.

This algorithm allows the merging of any two detections irrespective of the time of the detection or the robot's viewpoint as long as the point clouds are registered in the same global frame and there is at least one common stair between the detections. This merging algorithm is also agnostic to the detection algorithm and can merge detected staircases from different algorithms as long as the detection outputs use our representation of the staircase.

3.3 Staircase Perception System

All the previously discussed components can be combined to create a complete staircase perception system deployable on multiple mobile robots. Although these individual algorithms can be used in different configurations, we propose two model architectures that detects, estimates and merges staircases from different heterogeneous robots.

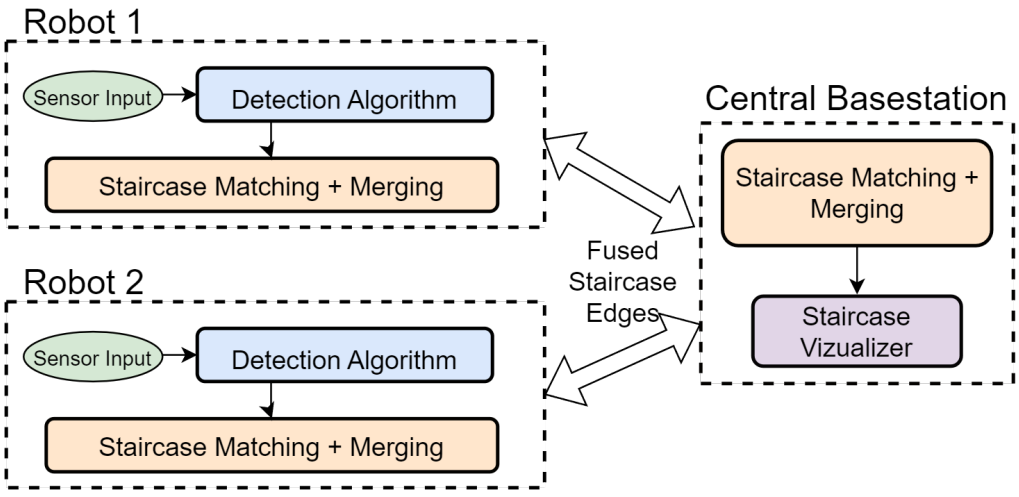


Figure 3.1: Centralized architecture to run the staircase perception pipeline on two robots

Figure 3.1 shows the block diagram for a centralized staircase perception system where a central basestation handles detected staircases from all robots. Each robot runs an instance of the detection algorithm and the merging pipeline to filter and fuse multiple detections. The robots then relay the fused staircases back to the basestation, which also runs an instance of the merging pipeline to merge staircases detected by different robots. The basestation also has a visualizer which can display all the staircases and can be used to command a robot to traverse a staircase that is fused by using information from different robots.

This system can be decentralized by eliminating the basestation and establishing communication between robots. Fig. 3.2 shows a decentralized version of the architecture. Each robot in the system sends the detected staircases to all the other robots to ensure every robot knows about every staircase detected in the environment. However, this increases the communication bandwidth required for functioning as the information must be sent to every robot, making it infeasible as the number of robots in the system goes above 3. The bandwidth required for communication is slightly lesser in a centralized system compared to a fully connected decentralized system as each robot only talks to a basestation.

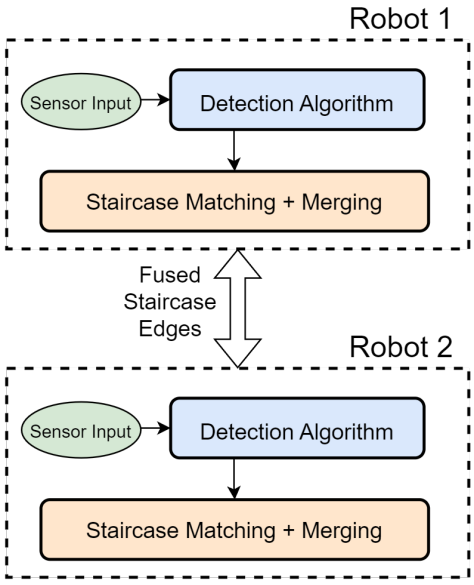


Figure 3.2: Decentralized architecture to run the staircase perception pipeline on two robots

Chapter 4

Experimentation

4.1 System Overview

To test the performance of the algorithm, we decided to test it with two different sensors that are capable of providing 3D point clouds. The first sensor payload primarily uses a LIDAR sensor to obtain three-dimensional scans of the environment. It houses a Jetson AGX Xavier as the processor that performs SLAM using Super Odometry [22] and other autonomy tasks. This payload is shown in the Fig. 4.1. The

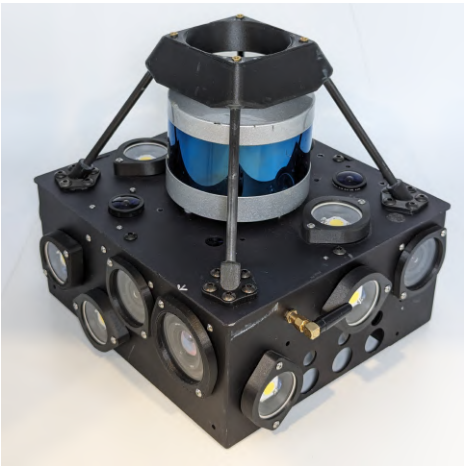


Figure 4.1: A sensor payload with LiDAR sensor and a Nvidia Xavier compute

Figure 4.2: A sensor payload with RealSense D455 RGBD sensor and a Intel NUC compute

advantage of using a LiDAR is that it does not depend on environment's lighting to have good accuracy and can support detecting staircases in all directions(360°). Although, it requires a good SLAM algorithm to achieve dense point cloud stacking and is susceptible to noise if SLAM output is noisy.

The second sensor payload we used is shown in Fig. 4.2. This payload houses a Intel NUC compute and uses a RealSense D455 RGBD sensor to produce dense point clouds. This sensor does not need any SLAM algorithm to obtain dense point clouds and hence can used to perform detections locally and fast. Although, the stereo nature of the sensor causes inaccuracies at farther distances and also is affected by the environment lighting and texture of the staircase.

To test our full staircase system, we make use of heterogeneous robots with identical perception sensor payloads. Figure 4.3 shows the Boston Dynamics Spot legged robot. This robot is capable of climbing staircases once the staircase's location has been estimated. Figure 4.4 shows the wheeled ground robot. The vehicle is capable of moving at up to 6m/s autonomously. Robots travelling at such speeds require the staircase algorithm to detect staircases really fast for it to be to deployable. These robots also have the capability of exchange messages, and share the same map thereby providing a good test-bench for the multi-stair merging pipeline.

Figure 4.3: Spot Legged Robot with the LiDAR payload

Figure 4.4: Autonomous Wheeled Robot with the LiDAR payload

4.2 Experimental Setup

We tested our detection algorithm on five different types of staircases and performed a comparison with the state-of-the-art [19]. The five different types are ascending staircases, descending staircases, hollow staircases, spiral staircases and dilapidated staircases which includes staircases with debris and damaged staircases (Fig. 1.3). We have at least two different staircase for each type, and we tested it with both the LiDAR and the RealSense payload.

As the robots move around these five staircase types, we create a dataset that spans multiple distance points and orientations in front of the staircase. To compare performance, we use the stair parameters output by each algorithm and compare it with the ground truth. An essential aspect of our comparison is the time taken to obtain a detection as we want to run our algorithm in real-time. Figure 4.5 shows examples of heterogeneous robots moving around different types of staircases.

To test the merging pipeline, we setup different experiments where a robot looks at the staircase from different locations and then run the merging algorithm. On top

(a) (b) (c)

Figure 4.5: Heterogeneous robots moving around different types of staircase to create a dataset

Figure 4.6: The Spot robot traversing a long staircase as a test of the staircase system and it's merging capabilities in real-time

of checking for a successful merge, we also compare the staircase parameters output by individual detections, and the parameters obtained after merging. We also test the same pipeline using heterogeneous robots that look at the staircase from the top and bottom simultaneously. In the last experiment, we run the Spot robot on a very long staircase that is about two stories tall to evaluate the performance of our proposed staircase perception system. An aerial snapshot of this experiment is shown in Fig. 4.6

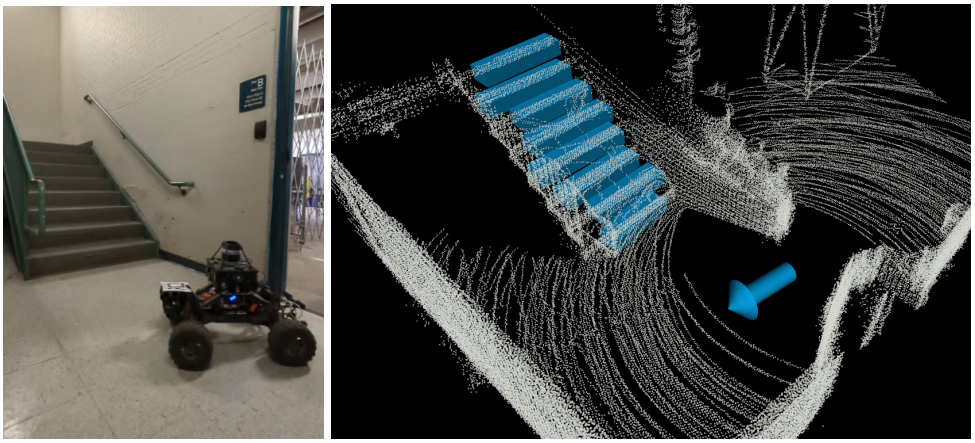
Chapter 5

Results

5.1 Detection Results

5.1.1 Ascending Staircase

We first describe the results for an ascending staircase using both the sensors. Figure 5.1 shows the wheeled robot in front of a staircase, and it's corresponding staircase

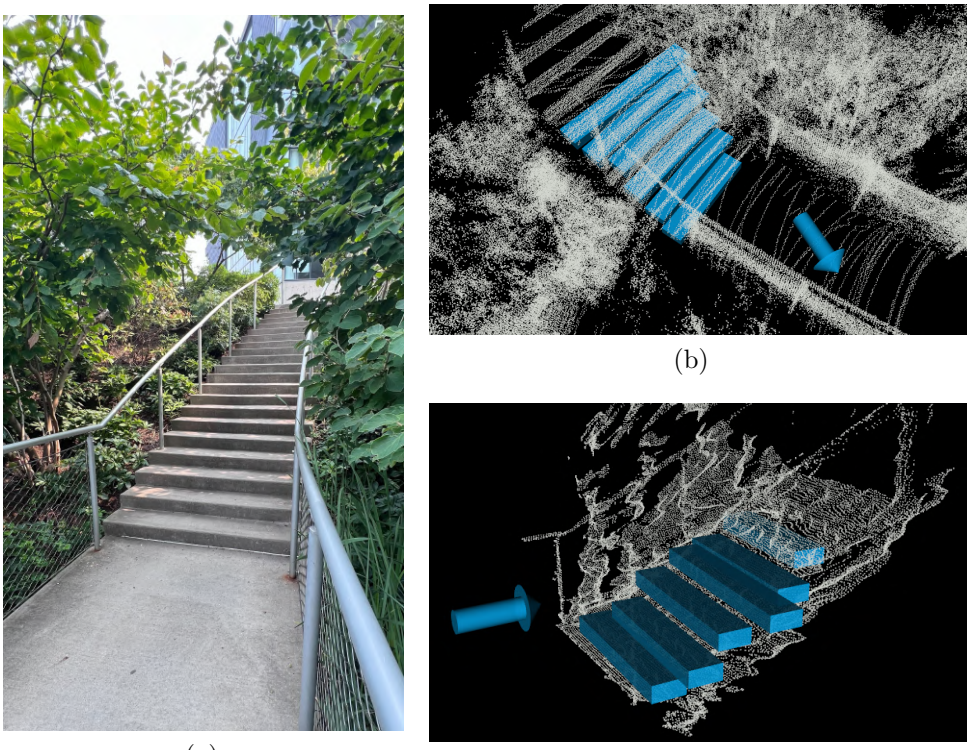


(a) (b)

Figure 5.1: Ascending staircase detected by the algorithm using LiDAR point cloud. (a) Wheeled Robot in front of a staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

detection using the point cloud on the right. The blue marker represent the detected stairs, which are resized to match the algorithm's estimation results. The blue arrow represents the location at which the staircase was detected. The LiDAR was the primary sensor in this scenario. We were also able to successfully detect staircases using the RealSense point cloud as shown in Figure 5.2 (c). From Fig. 5.2, its evident that the RealSense has much smaller field of view and is able to detect staircase in that range.

Across all the samples for ascending staircases, we were able to detect staircases within 21 milliseconds where as the SOTA [19] took around 1569 milliseconds. We measured the average error on our estimated parameters for all samples with respect to the ground truth and compared it to the SOTA. It is shown in the Table 5.1. The



(a) (b) (c)

Figure 5.2: Outdoor ascending staircase detection using both LiDAR and RealSense point clouds (a) Picture of the outdoor ascending staircase (b) Staircase detected using the LiDAR point cloud as shown using blue markers (c) Staircase detected using the RealSense point cloud as shown by the blue markers

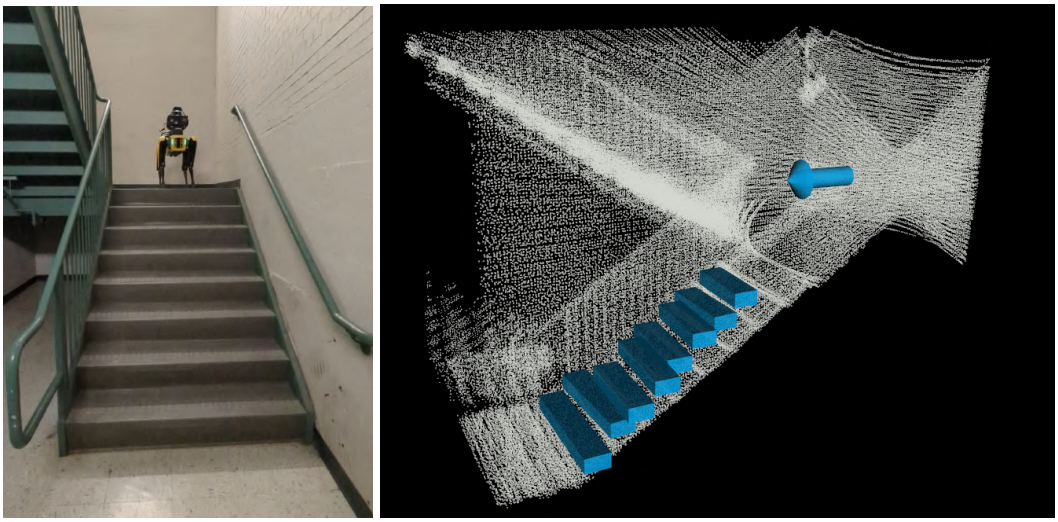
table also notes the performance difference using the number of true detections and false positives. The SOTA algorithm had a large amount of false positives especially in outdoor cluttered environments.

Ascending Staircase		
Performance Metric	**Our Method**	**Westfechtel et al.[19]**
Detection Time (ms)	21	1569
Height Error (cm)	1.101	1.527
Depth Error (cm)	2.804	2.564
Width Error (cm)	38.125	7.023
True Detections over Samples	15/15	15/15
False Positives over Samples	0/15	10/15

Table 5.1: Table showing performance metrics of our algorithm compared to the state-of-the-art for ascending staircases

5.1.2 Descending Staircase

Figure 5.3 shows our algorithm's performance on descending staircase. The average detection speeds for descending staircase was around 10 milliseconds. The reason for



(a) (b)

Figure 5.3: Descending staircase detected by the algorithm using LiDAR point cloud. (a) Spot robot on top of a staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

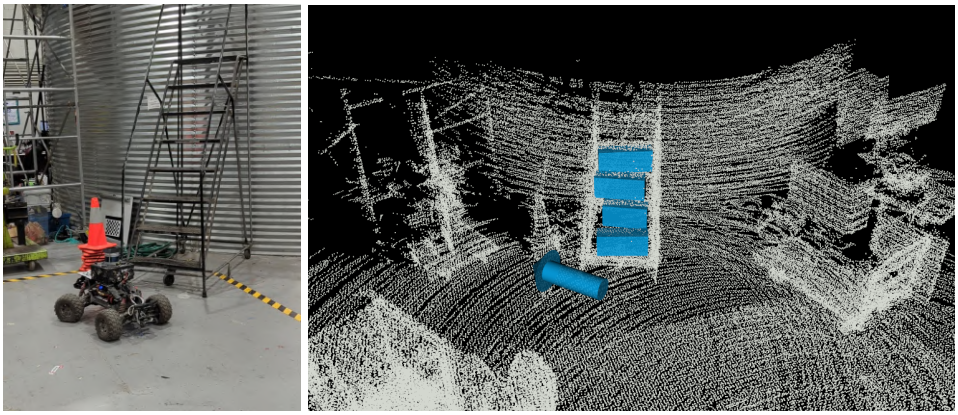
Descending Staircase		
Performance Metric	**Our Method**	**Westfechtel et al.[19]**
Detection Time (ms)	10	623
Height Error (cm)	2.367	1.517
Depth Error (cm)	2.653	4.470
Width Error (cm)	27.773	15.437
True Detections over Samples	14/14	12/14
False Positives over Samples	0/14	3/14

Table 5.2: Table showing performance metrics of our algorithm compared to the state-of-the-art for descending staircases

this is due to the point clouds being sparser when looking at descending staircase. The different errors in the parameter estimation for descending staircases are shown in the Table 5.2. We were successfully able to detect descending staircases, while trading blows in errors compared to the SOTA. The speed of detection was also significantly faster compared to the SOTA without having any false positives.

5.1.3 Hollow Staircase

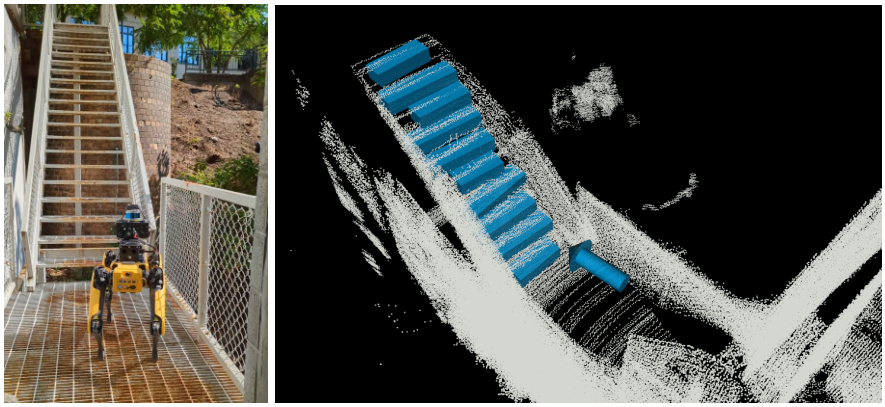
Our algorithm is able to successfully detect staircases that are slightly more sophisticated. Figure 5.4 shows the result of the detection algorithm on a staircase that has



(a) (b)

Figure 5.4: Hollow staircase detected by the algorithm using LiDAR point cloud. (a) Wheeled robot near a hollow staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

no backing, and is hollow. This is possible as we use edges to detect staircases rather than planes. Figure 5.5 shows a different example of hollow staircase that is detected by the Spot robot. In this example, the robot can detect 10 stairs from a single point cloud.



(a) (b)

Figure 5.5: Hollow staircase detected by the algorithm using LiDAR point cloud. (a) Spot near a hollow staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

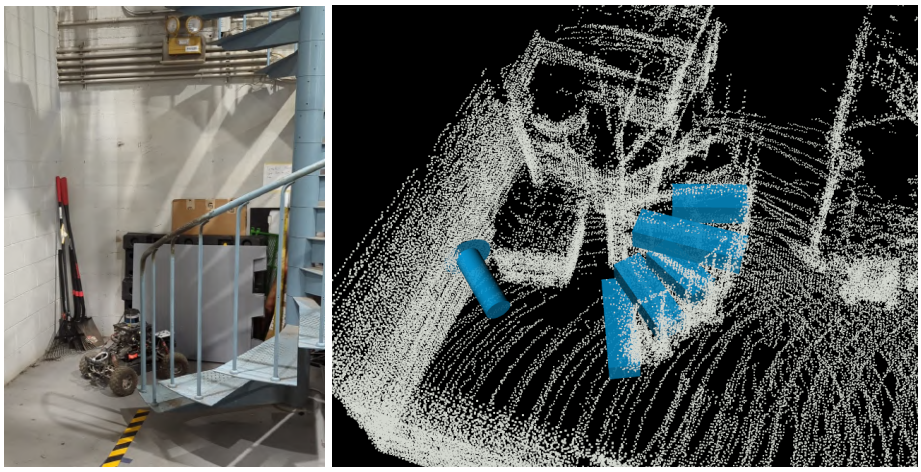
Compared to the SOTA, we perform significantly better for hollow staircases. The SOTA was not able to detect staircases in multiple scenarios as the hollow staircases lacked enough points to form planes. Even when the SOTA did detect the staircase, it took about 1009 milliseconds compared to the 18 milliseconds taken by our algorithm. We also estimate all the parameters much more accurately than the SOTA. These metrics are shown in the Table 5.3.

Hollow Staircase		
Performance Metric	**Our Method**	**Westfechtel et al.[19]**
Detection Time (ms)	18	1009
Height Error (cm)	0.982	7.673
Depth Error (cm)	0.839	13.760
Width Error (cm)	15.121	41.233
True Detections over Samples	17/17	3/17
False Positives over Samples	1/17	1/17

Table 5.3: Table showing performance metrics of our algorithm compared to the state-of-the-art for hollow staircases

5.1.4 Spiral Staircase

We can successfully detect curved staircases or those arranged in a spiral orientation. Figure 5.6 shows a spiral staircase successfully detected by the wheeled robot. This detection only took around 15 milliseconds and we could also estimate the stair height significantly more accurately than the SOTA.



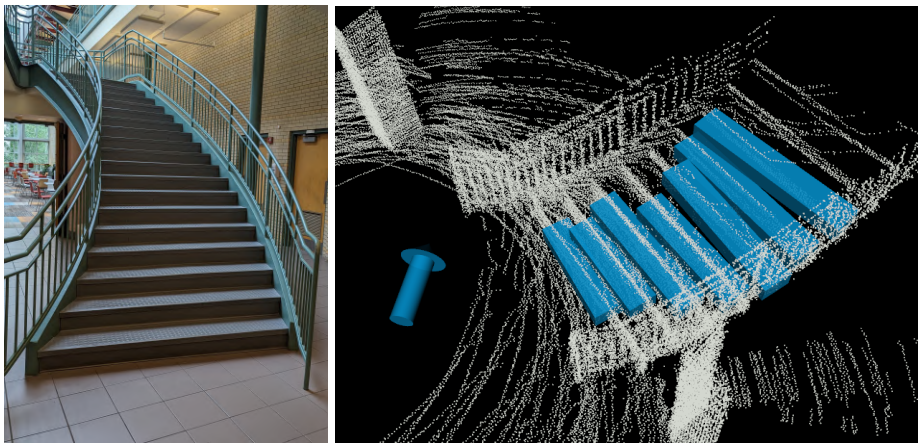
(a) (b)

Figure 5.6: Spiral staircase detected by the algorithm using LiDAR point cloud. (a) Wheeled robot near a spiral staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

Figure 5.7 shows the algorithm successfully detecting a circular staircase. Our algorithm is significantly faster than the SOTA, and we estimate the parameters of the staircase better than the SOTA as shown in Table 5.4

Spiral Staircase		
Performance Metric	**Our Method**	**Westfechtel et al.[19]**
Detection Time (ms)	15	1005
Height Error (cm)	0.490	3.357
Depth Error (cm)	3.467	2.687
Width Error (cm)	11.040	27.667
True Detections over Samples	15/15	15/15
False Positives over Samples	0/15	14/15

Table 5.4: Table showing performance metrics of our algorithm compared to the state-of-the-art for spiral staircases



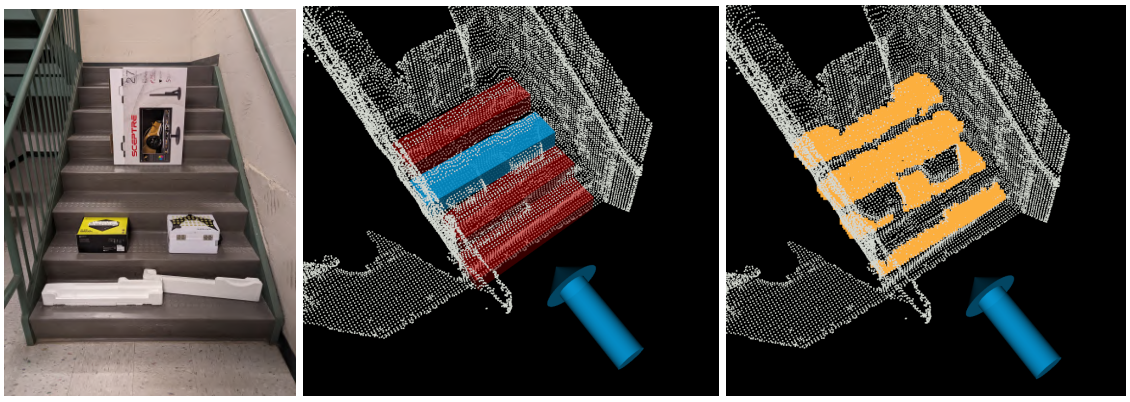
(a) (b)

Figure 5.7: Circular staircase detected by the algorithm using LiDAR point cloud. (a) Image of the circular staircase (b) Detected staircase shown using blue cuboids, with blue arrow representing robot's location

5.1.5 Dilapidated Staircase

We successfully detected and validated dilapidated staircases using both LiDAR and RealSense pointclouds. We used the RealSense payload for all the experiments involving the validation pipeline. For the first result, boxes of varying sizes were placed on different stairs as shown in Fig. 5.8(a). Figure 5.8 shows successful detection of the staircases with boxes on some of the stairs. The blue markers in the detection indicate that the stair has no obstacles and is traversable, while the red marker indicates that the surface of the stair is not entirely free. Figure 5.8(c) shows the output of the validation pipeline, where the orange points indicate the points that belong to the stair. As expected, there are no orange points in the locations of the boxes

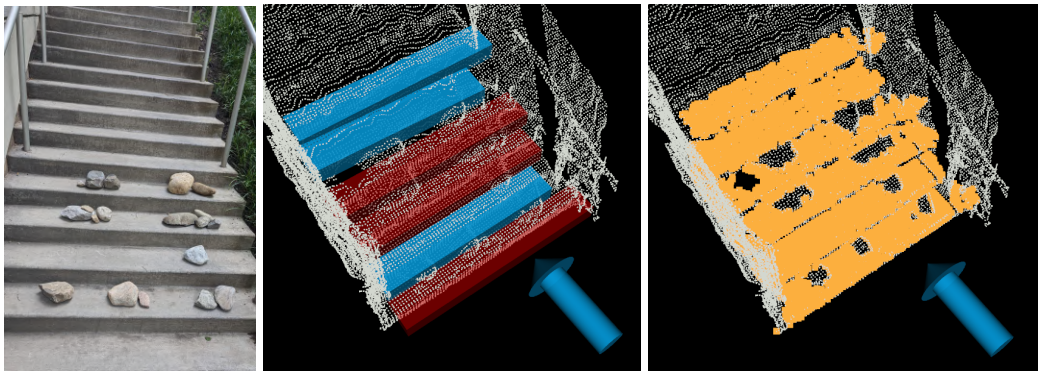
In the second result, rocks were used as debris to simulate a search and rescue environment. We successfully detected the staircase in this scenario, as shown in the Fig. 5.9. The scene presented in Fig. 5.9(a) shows only one single rock on the second stair, which in reality, should not affect the traversability of the stair. This is reflected in our validation algorithm as we can see from Fig. 5.9(b) that the second stair is classified as healthy (blue marker). In contrast, the other stairs are classified correctly as having obstacles (red marker). Fig. 5.9(c) again shows the points that



(a) (b) (c)

Figure 5.8: Positive detection and validation of a staircase with debris using RealSense pointcloud. (a) Image of the staircase that has boxes and debris on its surfaces (b) Detected staircase shown using cuboids, blue cuboid represents that the stair is free to traverse, red cuboid indicates that the stair has some obstacles on them (c) Orange points represent the points that are part of the stair surface as predicted by the validation pipeline

belong to the stair surface using the orange points.

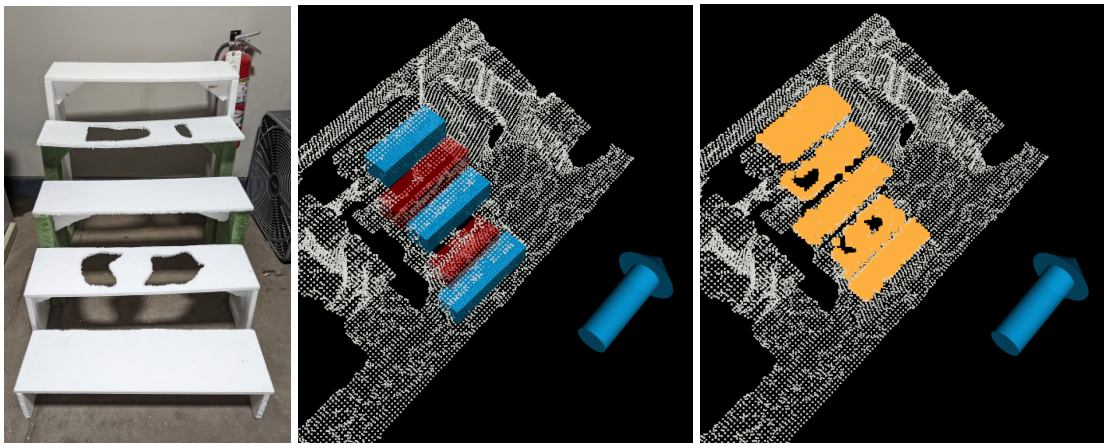


(a) (b) (c)

Figure 5.9: Positive detection and validation of a staircase with rocks on top using RealSense pointcloud as input. (a) Image of the staircase that has rocks on its surfaces (b) Detected staircase shown using cuboids, blue cuboid represents that the stair is free to traverse, red cuboid indicates that the stair has some obstacles on them (c) Orange points represent the points that are part of the stair surface as predicted by the validation pipeline

Lastly, we also tested the validation pipeline with a broken staircase. As shown in Fig. 5.10(a), the staircase has damage on the second and fourth stairs, rendering them untraversable. Our detection and validation algorithm successfully detects and classifies the stair correctly. Figure 5.10(b) shows the detected staircase with the healthy stair marked by blue, and the second and fourth stairs are marked using red to indicate damage. Figure 5.10(c) shows the output of the validation pipeline with orange points representing the points that belong to the stair surface. As expected, the orange points have a cavity in them due to the stairs' damage.

Even though the state-of-the-art algorithm detected these staircases, it does not provide any understanding of the traversability of the stair. Table 5.5 shows the error in the parameter estimation for these staircases. Our proposed method still detects significantly faster and has a slightly better estimation. The SOTA algorithm also suffers from false positives in cluttered and noisy environments. In contrast, our validation pipeline can help reject false positives by checking if the detected stair surface has no points. This is reflected in all our comparisons, where we recorded



(a) (b) (c)

Figure 5.10: Positive detection and validation of a simulated damaged staircase using RealSense pointcloud. (a) Image of the staircase with damage on its surfaces (b) Detected staircase shown using cuboids, blue cuboid represents that the stair is free to traverse, red cuboid indicates that the stair has damages on them (c) Orange points represent the points that are part of the stair surface as predicted by the validation pipeline

Dilapidated Staircase		
Performance Metric	**Our Method**	**Westfechtel et al.[19]**
Detection Time (ms)	32	2202
Height Error (cm)	2.980	3.660
Depth Error (cm)	2.362	3.736
Width Error (cm)	9.312	11.270
True Detections over Samples	10/10	10/10
False Positives over Samples	1/10	10/10

Table 5.5: Table showing performance metrics of our algorithm compared to the state-of-the-art for dilapidated staircases

only a few false positives.

To summarize our algorithm's estimation accuracy and detection speed compared to the SOTA, we estimate the height and depth better than the SOTA overall. While both algorithms trade blows with standard ascending and descending staircases, we perform much better than the SOTA in other cases. Our algorithm has significantly higher estimation accuracy when it comes to hollow or spiral staircases. We also performed better in the case of dilapidated staircases. Our algorithm detects all staircases with an average computation time of 20.64 ms as opposed to the 1281.66 ms using SOTA. These detection speeds are faster by at least two orders of magnitude. All of the results shown above only discuss local detections, i.e. detections are different at each time-step.

5.2 Multi-Detection Merging Results

We also evaluated the performance of our merging algorithm. We were successfully able to merge two different detection instances into one staircase. In this section, we present results for two types of merging - multi-robot merging and single robot merging. In single robot case, we show results of merging detections of the same staircase provided by the same robot across multiple time steps. In the multi-robot case, we present results of the merging detections from two different robots looking at a staircase from different viewpoints.

5.2.1 Multi-Robot Merging

The Spot robot first detects a descending staircase from the top. This is shown in Fig. 5.11(a). Due to its limited viewpoint, the spot successfully detects five of the eight stairs on the entire staircase. Specifically, the spot detects the bottom five stairs. Fig. 5.11(b) shows the detected staircase using the blue marker. While the Spot detects the staircase, the wheeled robot comes in from the bottom and successfully detects the same staircase as shown in Fig. 5.12(a). The successful detection is shown using pink markers corresponding to the pink arrow that depicts the wheeled robot's location as depicted by Fig. 5.12(b). The wheeled robot detects seven of the eight stairs as it misses the bottom most stair.

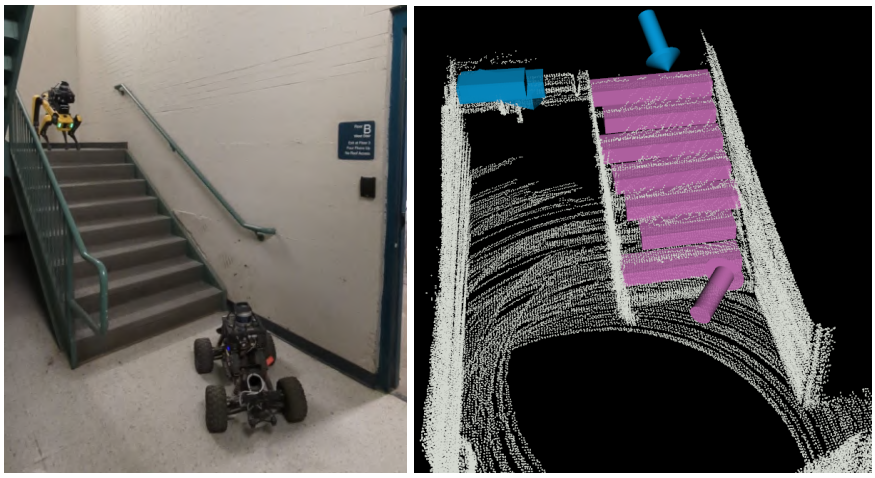
(a) (b)

Figure 5.11: Positive detection of staircase by the Spot. (a) Spot Robot looking down at a staircase (b) Detected staircase shown using the blue marker corresponding to the blue robot

The two detections are passed to the merging algorithm to combine them. Figure 5.13 shows the successfully merged staircase that is obtained by fusing both detections. The merging is successful, as the merged staircase has a total of 8 stairs which the individual detections did not have. The bottom most stair was detected by Spot and was added to the fused staircase, while the information about the top three stairs are obtained from the wheeled robot's detection.

Table 5.6 shows the estimated parameters of the staircase from the individual detections and the parameters of the staircase after merging and shows the measured

ground truth values of the staircase. It is evident that the merging algorithm gives a better estimate of the staircase as the merged parameters are much closer to the ground truth than the individual estimated parameters. Since the lines are merged by averaging, the width of the stair is equal to the average width between two stairs.



(a) (b)

Figure 5.12: Positive detection of staircase by the wheeled robot. (a) Wheeled robot in front of the staircase (b) Detected staircase shown using the pink marker corresponding to the pink robot

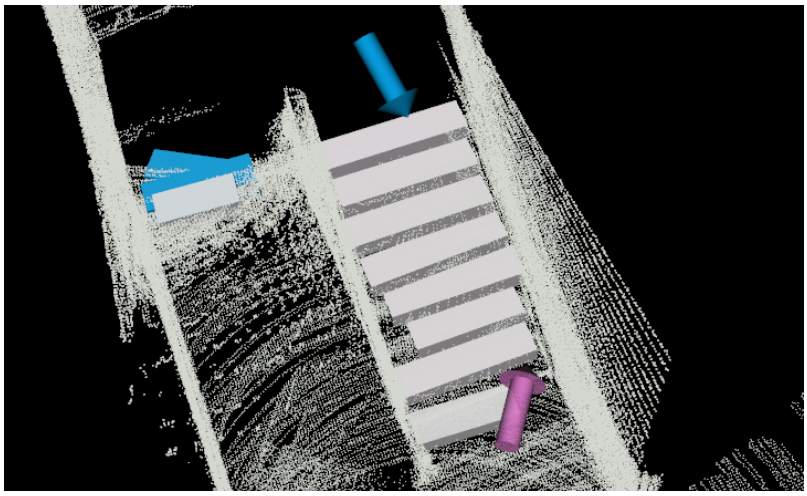


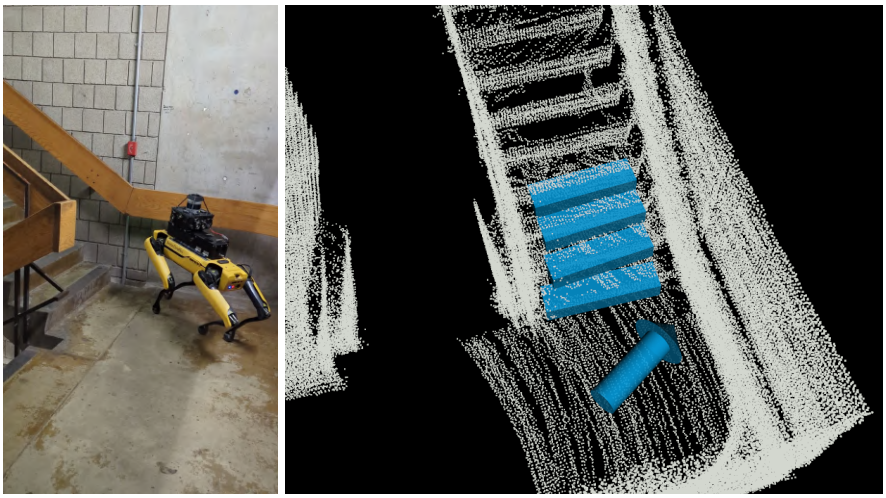
Figure 5.13: Merged staircase using two detections from both Spot and the Wheeled robot by the white markers. The colored arrows represent the robot's locations when the individual detections were reported.

Staircase Parameters	Spot's Detected Staircase	Wheeled Robot's Detected Staircase	Merged Staircase	Ground Truth
Stair Count	5	7	8	8
Stair Height (cm)	18.16	17.9	17.8	17.78
Stair Depth (cm)	27.54	29.39	29	29.21
Stair Width (cm)	91.69	120	116	129.54

Table 5.6: Table showing staircase parameters estimated by individual detections from the different robots, parameters after successful merge and the ground truth values

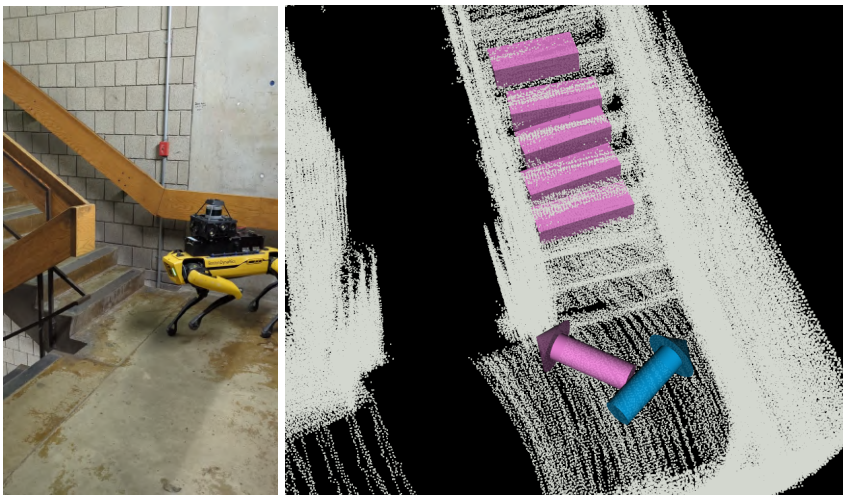
5.2.2 Single-Robot Merging

The merging algorithm can successfully merge detections from two separate time steps one second apart. At $t = 1$, the spot robot stands before the staircase, as shown in Fig. 5.14(a). At this time step, the detection algorithm can only see the bottom four stairs and reports the detection. This detection is shown using blue markers in Fig. 5.14(b). At the next time-step $t = 2$, the spot turns around and detects five stairs on



(a) (b)

Figure 5.14: Positive detection of staircase by the Spot. (a) Spot in front of the staircase at the start of the run (b) Detected staircase shown using the blue marker corresponding to the blue robot



(a) (b)

Figure 5.15: Positive detection of staircase by Spot. (a) Spot in front of the staircase in a different orientation (b) Detected staircase shown using the pink marker corresponding to the pink robot

the upper half of the staircase. Figure 5.15 shows the spot in the new orientation and its corresponding detection using the pink markers. These two detections have one stair in common, which the merging algorithm uses as a pivot to merge the two detections.

The merging algorithm successfully fuses the detections, and the merged staircase is shown using white markers in Fig. 5.16. The combined staircase has eight stairs in total, as there was only one common stair across both time steps. Table 5.7 shows the staircase parameters from the individual detections and merged detections alongside the ground truth. The update of parameters is not as significant because of only one

Staircase Parameters	**Detected Staircase t = 1**	**Detected Staircase t = 2**	**Merged Staircase**	**Ground Truth**
Stair Count	4	5	8	8
Stair Height (cm)	16.12	20.19	18.57	17.78
Stair Depth (cm)	24.09	27.23	26.06	27.94
Stair Width (cm)	89.71	79.28	83.77	106.68

Table 5.7: Table showing staircase parameters estimated by individual detections from the same robot, parameters after successful merge and the ground truth values

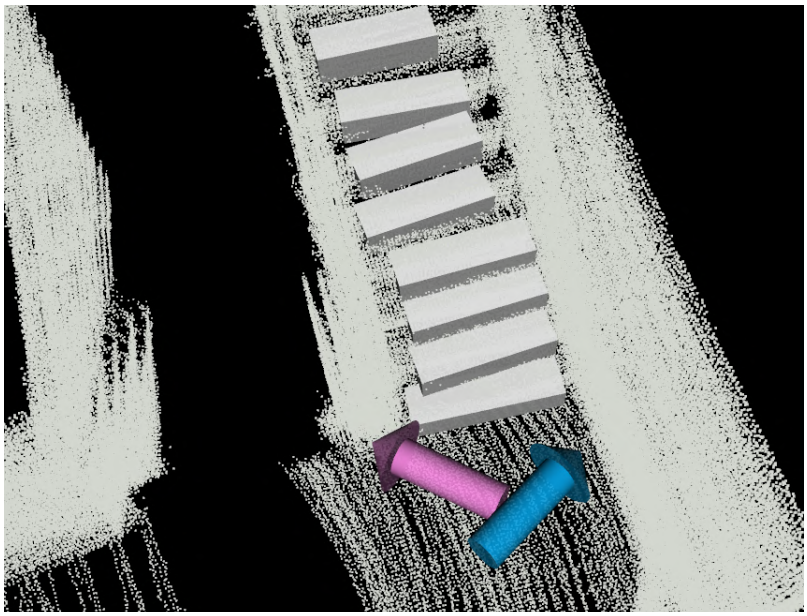


Figure 5.16: Merged staircase using two detections from the same robot across different time-steps as indicated by the white markers. The colored arrows represent the robot's locations when the individual detections were reported.

stair match. Although, the merged estimates are still closer to the ground truth than the estimated parameters by the individual detections.

5.3 Staircase System Evaluation for Large Staircases

Our proposed staircase perception system successfully detected, estimated, and merged three long staircases consisting of a total of 54 stairs that span across two stories in the real world. The system performs everything in real-time and on board the Xavier compute on the sensor payload. Figures 5.17 - 5.26 represent ten snapshots in time of the system spanning over 45 seconds for the entire run. On the left is an image of the robot on the staircase; the middle figure shows the local detection indicated by a blue marker at that time step, and the right figure shows the incrementally merged staircase using the white markers as the robot climbs up the staircase.

Figure 5.27 shows the resulting staircases at the end of the run. Our staircase system was successfully able to detect all 52 out of the 54 stairs across three staircases

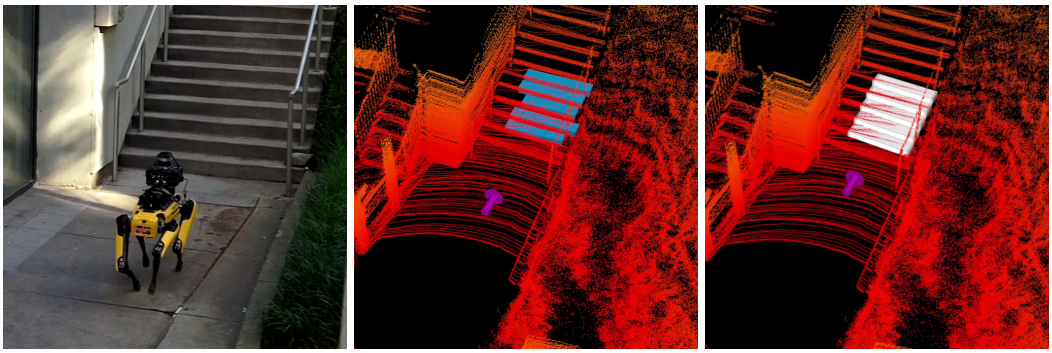


Figure 5.17: Staircase system results at $t = 1$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

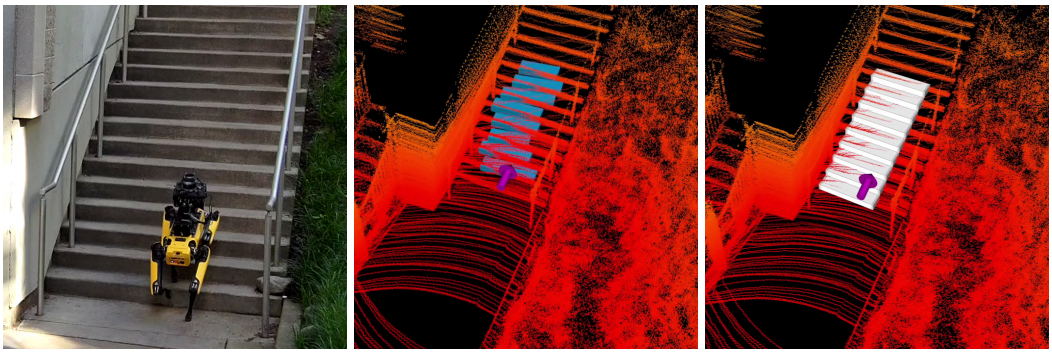


Figure 5.18: Staircase system results at $t = 2$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

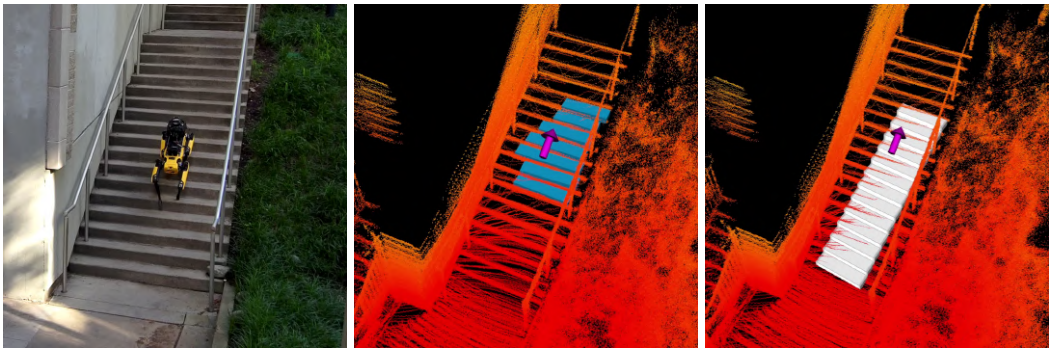


Figure 5.19: Staircase system results at $t = 3$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

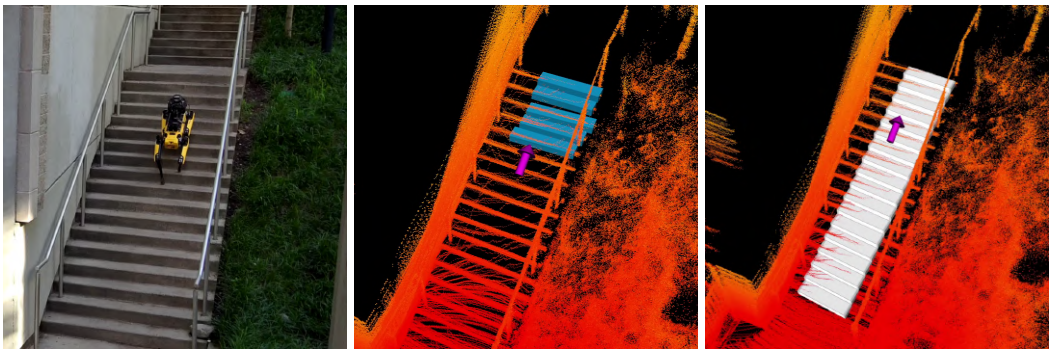


Figure 5.20: Staircase system results at $t = 4$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

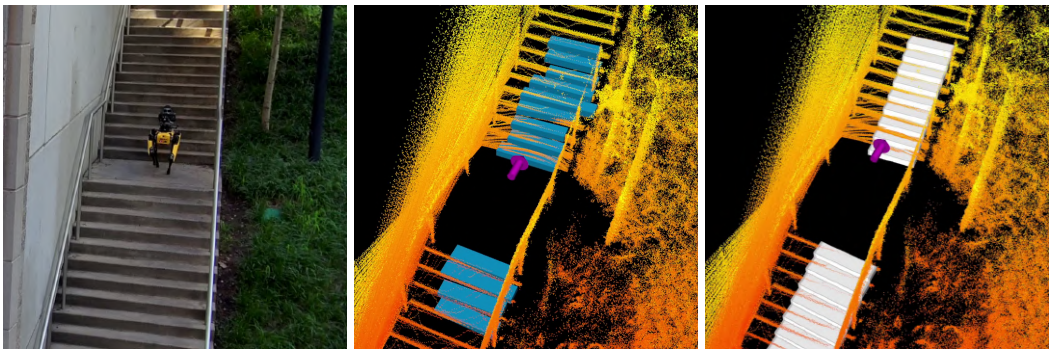


Figure 5.21: Staircase system results at $t = 5$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

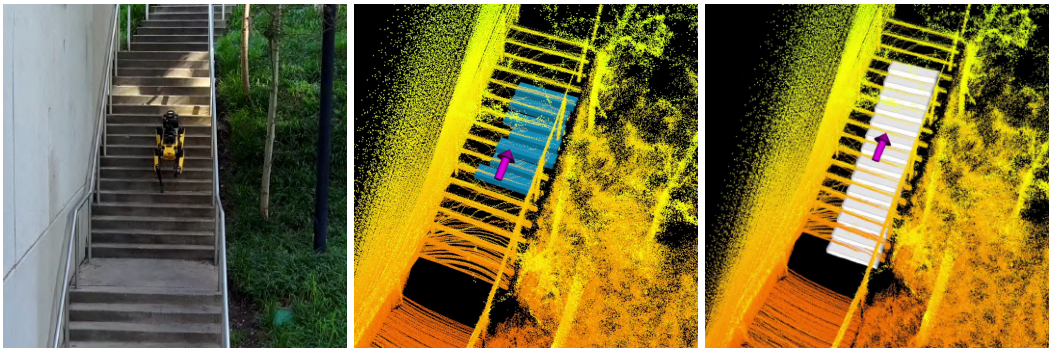


Figure 5.22: Staircase system results at $t = 6$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

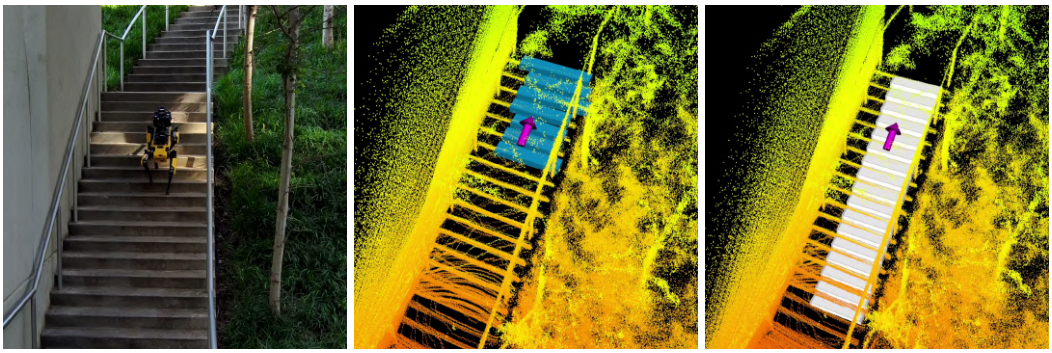


Figure 5.23: Staircase system results at $t = 7$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

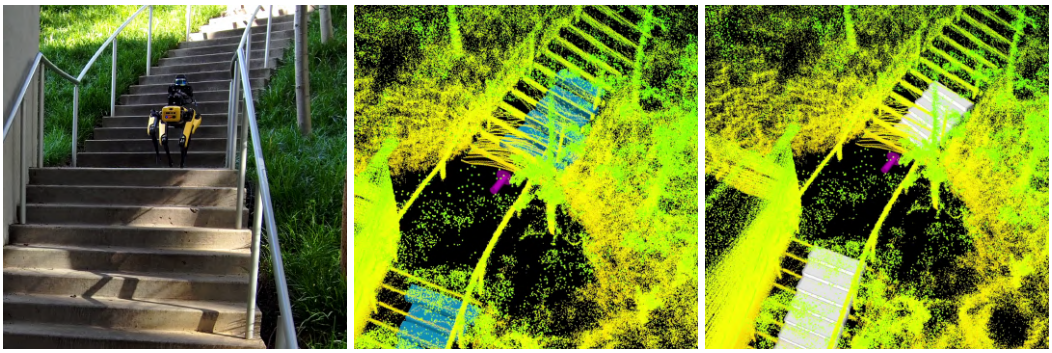


Figure 5.24: Staircase system results at $t = 8$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

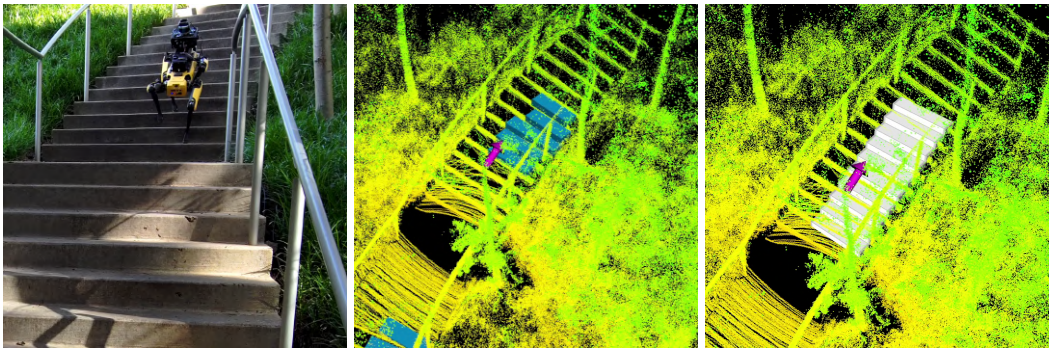


Figure 5.25: Staircase system results at $t = 9$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

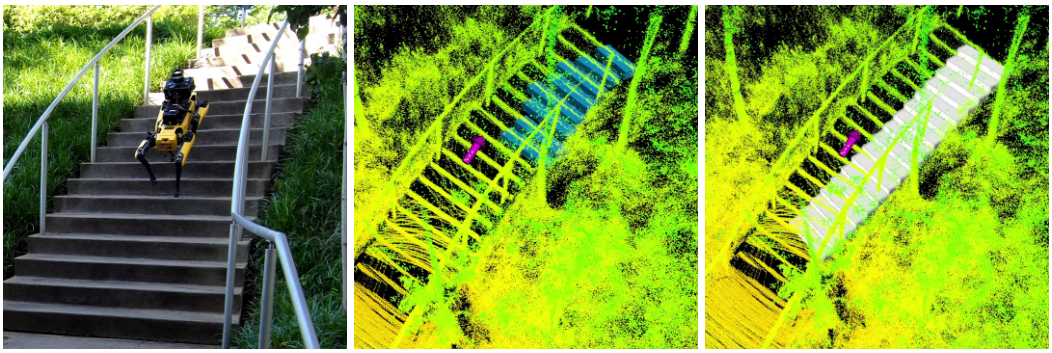


Figure 5.26: Staircase system results at $t = 10$; Spot robot climbing stair(left); Local staircase detection (middle); Merged staircase in global frame (right)

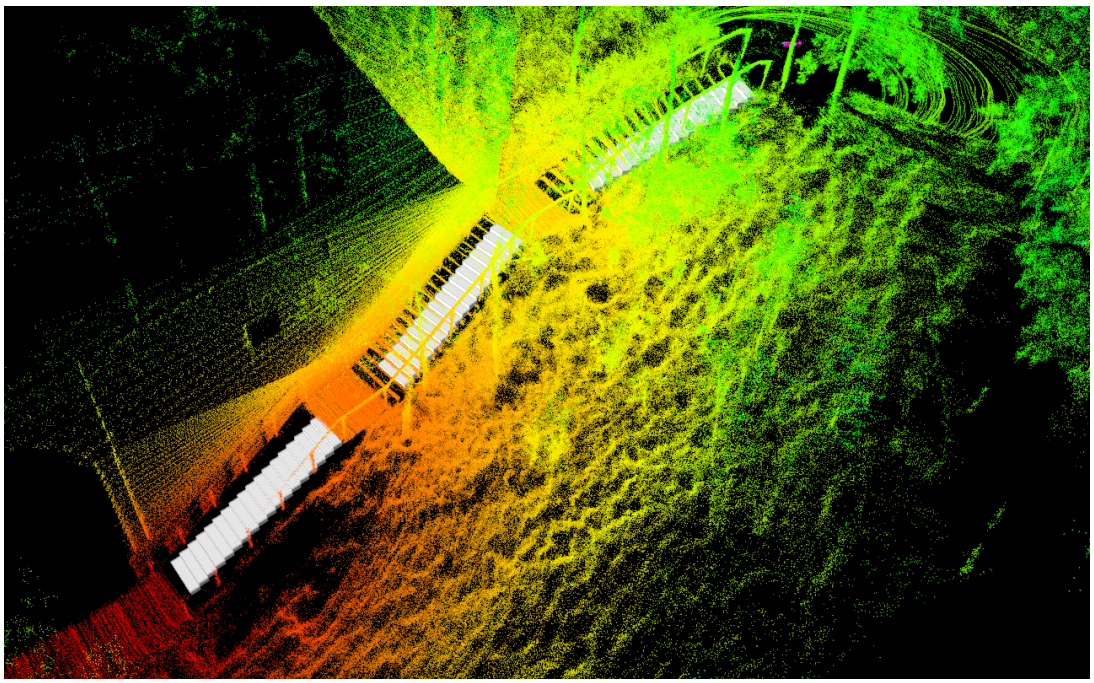


Figure 5.27: Final resulting staircases after successfully merging all detections of three long staircases spanning two stories. The white markers are sized based on the average height, depth and width of stair.

and estimate the parameters. The last 2 stairs were not visible in the field of view of the velodyne, thereby causing the algorithm to miss them. The white markers used for representing the globally merged staircases are sized based on the average parameters estimated by the algorithm, which is in contrast to the local detections

where the blue markers are sized based on the individual stair parameters. The estimated parameters at the end of run alongside the ground truth are mentioned in Table 5.8.

Staircase Parameters	**Merged Staircase**	**Ground Truth**
Stair Count	52	54
Stair Height (cm)	14.20 cm	16 cm
Stair Depth (cm)	28.13 cm	29 cm

Table 5.8: Table showing staircase parameters estimated by individual detections from the same robot, parameters after successful merge and the ground truth values

Chapter 6

Conclusions

Staircase detection and estimation is the first step in enabling robots to traverse staircases autonomously. In this thesis, we first presented an algorithm that successfully detected staircases. By exploiting the different projections of the point cloud, we reduced the segmentation process from 3D to 2D, enabling the algorithm to run in real-time. The algorithm could detect various staircases and estimate their parameters, such as their height, depth, and width. We also successfully devised a technique that predicted the traversability of each stair using a simple point density measure. This validation pipeline could correctly predict traversability on dilapidated staircases that were either broken or had obstacles on individual steps.

We also presented a pipeline that could match multiple staircases and merge two different detections of the same staircase. These detections could be from different robots or the same robot that moved around the staircases. The proposed merging algorithm was agnostic to the detection method and only depended on the representation of a staircase. We successfully deployed our algorithms on two heterogeneous robots and evaluated the accuracy of our estimation. Compared to the state-of-the-art, our algorithm is faster in detecting a staircase by order of two magnitudes while having better accuracy on sophisticated staircases such as hollow or spiral staircases. However, the state-of-the-art algorithm had better accuracy in estimating the width of the stairs.

We also proposed a framework that combined these algorithms and implemented them on heterogeneous mobile robots. This system was successfully deployed on

compute-constrained robots, and it could detect and merge long staircases in real time that a single instance of a point cloud cannot capture.

The use of projections limits the use of our algorithm in scenarios where the risers or treads of the stairs are sloped. In these cases, the segmented lines would be noisy, resulting in failed detections. We want to address these scenarios in future work by using methods to estimate ground plane orientation. Although, that might increase the detection time significantly. Additionally, the segmentation algorithm assumes that the stair edges are straight lines and the algorithm does not work with curved stairs. We would like to address this problem by using splines instead of lines. Further, the current merging algorithm uses averaging to combine two stairs, and we can improve the merging algorithm by using a Kalman filter that can help in noisy environments.

In the future, in addition to just detecting debris or damage on a stair, we would also like to investigate techniques that can quickly localize the debris or damage on the stair to provide a better traversability estimate in the validation pipeline. The validation pipeline currently assumes that the input point clouds are dense to correctly predict the traversability of the stair. This might not be true when robots quickly move past a staircase and might not be able to accumulate sufficient scans. We would like to address this in the future by using probabilistic methods to capture the uncertainty in the prediction and use that information to make better decisions.

Bibliography

[1] 1910.25 - Stairways. — Occupational Safety and Health Administration. https://www.osha.gov/laws-regs/regulations/standardnumber/1910/1910.25. Accessed: 2022-09-11. 2.2.3

[2] Automating the Way We Work - Bostondynamics. https://bostondynamics.com/webinars/automating-the-way-we-work/. Accessed: 2023-07-25. 1.1

[3] James WP Campbell and Michael Tutton. *Staircases: History, repair and conservation*. Routledge, 2013. 1.1

[4] Yang Cong, Xiaomao Li, Ji Liu, and Yandong Tang. A stairway detection algorithm based on vision for ugv stair climbing. In *2008 IEEE International Conference on Networking, Sensing and Control*, pages 1806–1811. IEEE, 2008. 2.1

[5] Jérémy Fourre, Vincent Vauchey, Yohan Dupuis, and Xavier Savatier. Autonomous rgbd-based industrial staircase localization from tracked robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10691–10696. IEEE, 2020. 2.1

[6] Hannes Harms, Eike Rehder, Tobias Schwarze, and Martin Lauer. Detection of ascending stairs using stereo vision. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2496–2502. IEEE, 2015. 2.1

[7] Muhammad Ilyas, Anirudh Krishna Lakshmanan, Anh Vu Le, and Rajesh Elara Mohan. Staircase recognition and localization using convolution neural network (cnn) for cleaning robot application. *Preprints*, 2018. 2.1

[8] Soichiro Murakami, Manabu Shimakawa, Kimivasu Kivota, and Takashi Kato. Study on stairs detection using rgb-depth images. In *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 1186–1191. IEEE, 2014. 2.1

[9] Viet Nguyen, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. In *2005 IEEE/RSJ International Conference on Intelligent*

Robots and Systems, pages 1929–1934. IEEE, 2005. 2.2.2

[10] Stefan Oßwald, Jens-Steffen Gutmann, Armin Hornung, and Maren Bennewitz. From 3d point clouds to climbing stairs: A comparison of plane segmentation approaches for humanoids. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 93–98. IEEE, 2011. 2.1

[11] Unmesh Patil, Aniket Gujarathi, Akshay Kulkarni, Aman Jain, Lokeshkumar Malke, Radhika Tekade, Kartik Paigwar, and Pradyumn Chaturvedi. Deep learning based stair detection and statistical image filtering for autonomous stair climbing. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 159–166. IEEE, 2019. 2.1

[12] Samuel T Pfister, Stergios I Roumeliotis, and Joel W Burdick. Weighted line fitting algorithms for mobile robot map building and efficient data representation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 1304–1311. IEEE, 2003. 2.2.2, 12

[13] Xiangfei Qian and Cang Ye. Ncc-ransac: A fast plane extraction method for 3-d range data segmentation. *IEEE transactions on cybernetics*, 44(12):2771–2783, 2014. 2.1

[14] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972. 2.2.2

[15] W Samakming and J Srinonchat. Development image processing technique for climbing stair of small humaniod robot. In *2008 International Conference on Computer Science and Information Technology*, pages 616–619. IEEE, 2008. 2.1

[16] José Armando Sánchez-Rojas, José Aníbal Arias-Aguilar, Hiroshi Takemura, and Alberto Elías Petrilli-Barceló. Staircase detection, characterization and approach pipeline for search and rescue robots. *Applied Sciences*, 11(22):10736, 2021. 2.1

[17] Bishwajit Sharma and Imran A Syed. Where to begin climbing? computing start-of-stair position for robotic platforms. In *2019 11th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 110–116. IEEE, 2019. 2.1

[18] Michiel Vlaminck, Ljubomir Jovanov, Peter Van Hese, Bart Goossens, Wilfried Philips, and Aleksandra Pižurica. Obstacle detection for pedestrians with a visual impairment based on 3d imaging. In *2013 International Conference on 3D Imaging*, pages 1–7. IEEE, 2013. 2.1

[19] Thomas Westfechtel, Kazunori Ohno, Bärbel Mertsching, Ryunosuke Hamada, Daniel Nickchen, Shotaro Kojima, and Satoshi Tadokoro. Robust stairway-detection and localization method for mobile robots using a graph-based model and competing initializations. *The International Journal of Robotics Research*, 37(12):1463–1483, 2018. 2.1, 4.2, 5.1.1, ??, ??, ??, ??, ??

[20] Seungjun Woo, Jinjae Shin, Yoon Haeng Lee, Young Hun Lee, Hyunyong Lee, Hansol Kang, Hyouk Ryeol Choi, and Hyungpil Moon. Stair-mapping with point-cloud data and stair-modeling for quadruped robot. In *2019 16th international conference on ubiquitous robots (UR)*, pages 81–86. IEEE, 2019. 2.1

[21] Yasuyoshi Yokokohji. The use of robots to respond to nuclear accidents: Applying the lessons of the past to the fukushima daiichi nuclear power station. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:681–710, 2021. 1.1

[22] Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, and Sebastian Scherer. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8729–8736. IEEE, 2021. 4.1

[23] Chen Zhong, Yan Zhuang, and Wei Wang. Stairway detection using gabor filter and ffpg. In *2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, pages 578–582. IEEE, 2011. 2.1