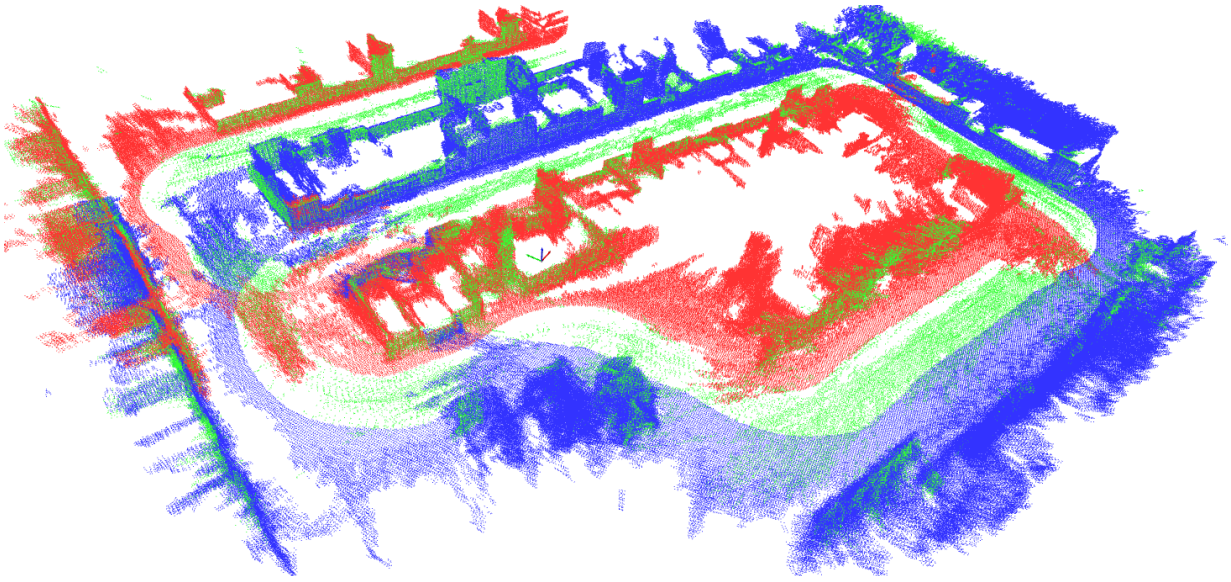


Panoramic 3D-camera SLAM for natural localization

A performance comparison with 2D-LiDAR in an industrial environment

Master's thesis in Systems, control and mechatronics

André Idoffsson & Stefan Larsson

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Panoramic 3D-camera SLAM for natural localization

A performance comparison with 2D-LiDAR in an industrial environment

André Idoffsson & Stefan Larsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Panoramic 3D-camera SLAM for natural localization

A performance comparison with 2D-LiDAR in an industrial environment

André Idoffsson & Stefan Larsson

Supervisor: Joar Manhed, FlexQube
Supervisor: Rikard Karlsson
Examiner: Petter Falkman

Master's Thesis 2021
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Point cloud obtained by a panoramic 3D-camera setup in an warehouse, each color corresponds to data captured by a separate camera.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Panoramic 3D-camera SLAM for natural localization
A performance comparison with 2D-LiDAR in an industrial environment
André Idoffsson & Stefan Larsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

A necessary component for the navigation of an Automated Guided Vehicle (AGV) is to accurately estimate where the vehicle is located in its current environment. It may also be beneficial for the AGV to have visual data of its environment to be able to perform tasks such as object identification and object avoidance. One sensor that is capable of all these tasks is the 3D-camera. However, it has been shown that the data provided by a single 3D-camera is insufficient for accurate localization.

This thesis presents a system for natural 2D-localization using multiple 3D-cameras arranged in a panoramic setup. The performance of the system is compared on both positional accuracy and repeatability against a conventional 2D-LiDAR localization system, where both systems achieve localization by Simultaneous Localization And Mapping (SLAM) using the Cartographer library. To optimize the localization performance of the panoramic system, several filters are applied to the point cloud data generated by the 3D-cameras. Evaluation of both systems are done on multiple data sets, captured in two different environments and sampled in parallel to give optimal comparability. The first environment is in a warehouse and is used to show the performance of a real world implementation. The second environment is from a lab-room equipped with a highly accurate positioning system, which is used as a reference to quantitatively evaluate the performance of both the panoramic camera system and LiDAR system.

The proposed system is shown to successfully achieve continuous localization in all tested environments.

Keywords: SLAM, Natural localization, RGBD, 3D-Camera, LiDAR, Point cloud filtering.

Acknowledgements

We would like to thank the R&D team at FlexQube and especially our supervisor Joar Manhed for welcoming us into the team and helping us out with resources used in the thesis. We would also like to thank our academic supervisor Rikard Karlsson at Chalmers for good discussion and input on the thesis work and Krister Wolf for helping us getting access to and help using the positioning reference system. A last thanks goes to our examiner Petter Falkman.

André Idoffsson & Stefan Larsson, Gothenburg, June 2021

Contents

Acronyms

AGV Automated Guided Vehicle.

CAGR Compound Annual Growth Rate.
CCL Connected Component Labeling.

EMA exponential moving average.

FLANN Fast Approximate Nearest Neighbor Search Library.
FoV Field of View.

HOG Histogram of Oriented Gradient.

ICP Iterative Closest Point.
IMU Inertial Measurement Unit.

LiDAR Light Detection and Ranging.

PCA Principal Component Analysis.

RANSAC Random sample consensus.

SIFT scale Invariant Feature Transformation.
SLAM Simultaneous Localization and Mapping.
SURF Speeded-Up Robust Features.

WMS Warehouse Management System.

1
Introduction

Autonomous transportation using an Automated Guided Vehicle (AGV) is a rapidly adopted means of transportation adopted by many industries [1], with the main motivation of reducing labor costs and increasing productivity [2]. As of 2019, the global AGV market was a 3.0 billion USD industry and have an estimated Compound Annual Growth Rate (CAGR) of 14.1% from 2020 to 2027 [2]. This indicates a significant and continuous interest in the development and improvement of AGV implementations.

From the name Automated Guided Vehicle is it apparent that this is a vehicle that is guided in an autonomous way. As described by Sabattini, Digani, Secchi, *et al.* [3] AGVs are generally controlled by some external supervisory system, usually referred to as a Warehouse Management System (WMS). The role of the WMS is to manage a fleet of AGVs inside the warehouse and assign them tasks. However, Sabattini, Digani, Secchi, *et al.* [3] also state that, even though the WMS supervise the fleet, each AGV still have to be able to independently localize itself in it's environment.

A common navigation method used for AGVs to achieve reliable behavior is to follow a physical marker on the floor [4]. This marker can, as an example, be a line of a specific color, or a magnetic tape [5], and thus defines a static path along which the AGV can travel. This static path gives a predictable behavior as the AGV is not supposed to operate outside of this defined path unless manual control is obtained. One major drawback of this static path is that the AGV is then unable to circumvent any obstacle that may appear along the defined path.

A more adaptive method of navigation would be a general 2D localization method. By letting the AGV know where in the environment it is positioned, it is given the possibility of navigating around obstacles throughout some non-obstructed path in a 2D-space. A 2D localization system requires sensors that can capture a 2D-representation of the environment. This is commonly achieved by a scanning 2D-LiDAR [5] where LiDAR is the abbreviation for Light Detection and Ranging [6], which is a laser-based range detecting sensor. However, it is only able to capture information in a 2D-plane. This makes it impossible for the sensor to detect objects outside of the sensor plane. Any overhanging or protruding obstacles can thus be a major safety risk for the AGV.

One way to solve this inherent problem of 2D-navigation would be to instead consider a 3D approach. This of course require that data can be captured in three dimensions, which is an exponential increase in data per localization area that needs to be processed in comparison to 2D. In addition, due to the increased complexity of capturing 3D data

as well as processing it, the price of a 3D sensor is often significantly higher than its 2D counterpart. An example of this is the scanning 3D-LiDAR as its technique is commonly just an extension of the 2D-LiDAR [7]. According to Tran, Becker, and Grzechca [8] is the 3D-LiDAR more expensive when compared to it's 2D counter part.

One 3D sensor technology that has shown recent improvement in both cost and performance is that of the RGB-D camera [9]. These cameras offer in addition to 3D space information also color data. By acquiring color data in addition to depth data, tasks that normally would require separate sensors for color and depth can now be achieved by one.

It may be argued that safety is one of the most important aspects when autonomously navigating a vehicle, especially when moving in the proximity of people. It is therefore important for the AGVs navigation system to have a good understanding of its environment. One way to improve safety is to use object detection as described by Bostelman, Hong, and Madhavan [10] where objects such as people are tracked live in 3D. By partially knowing the environment, specifically, that of the expected travel path of the AGV and any obstacles therein, an alternate path without collision, or if a full stop is required, can be determined, as shown by Pratama, Trong Hai, Kim, *et al.* [11].

1.1 Related work

This thesis is inspired by a previous master thesis "Investigating simultaneous localization and mapping for an automated ground vehicle" [12], which compares 2D-LiDAR to a 3D-Camera for the application of Simultaneous Localization and Mapping (SLAM) [13]. The conclusion from the thesis was that a single 3D-Camera has a hard time replicating the precision and robustness of a 2D-LiDAR. It was argued whether this was due to a narrow Field of View (FoV), insufficient detection range, or a combination of both. This theory is supported by Debeunne and Vivet [9], which also state that a lack of range is a major issue by 3D-cameras performing SLAM.

A similar problem of precision was also mentioned in a study by Dai, Yan, Liu, *et al.* [14]. In the study the authors tried to achieve SLAM with 3D-LiDAR. It was stated that for many applications the performance of the 3D-LiDAR would be insufficient without filtering of data. For this, the authors were able to show great improvements in the form of offline precision optimization. As the mentioned optimization methods were applied to generic 3D data, which should not necessarily only apply to data acquired with a 3D-LiDAR, we argue that similar methods may apply to data procured by 3D-cameras.

The authors Ji, Qin, Shan, *et al.* [15] compare the effects of different FoVs in visual SLAM. Standard camera FoV were compared to panoramic FoV and also a fisheye camera were compared. The conclusion from this report was that both the panoramic camera and fisheye camera exhibited higher robustness. One reason for this were that some algorithms did not successfully finish the data set tested with a standard field of view. This indicates that the previous thesis [12] might have been on the right track but with the limitation of not achieving a FoV large enough for the application.

1.2 Objective

The objective of this thesis is to develop and evaluate whether a panoramic multi 3D-camera system can achieve 2D positioning accuracy, trueness and repeatability comparable to existing 2D-LiDAR methods for industrial AGV localization.

1.2.1 Research question

This thesis aims to compare a panoramic 3D-camera setup to a 2D-LiDAR for localizing an AGV in industrial environments.
More precisely, it will attempt to answer the two following questions:

- Is it possible to localize an AGV in 2D with the help of a panoramic 3D-camera setup as input?
- How does the localization in 2D with a panoramic 3D-camera compare to a 2D-LiDAR setup in accuracy, trueness, and repeatability?

1.2.2 Limitations

The systems developed in this project will not be designed for navigation, only for localization. To verify a robust localization for an industrial application the proposed method will be tested in a static environment. That is, no movement of natural landmarks and no changing of lighting settings. However, multiple and diverse static environments will be tested.

The systems will be implemented on preexisting hardware in the form of an AGV. This will restrict the possibilities of sensor mounting location in a way as to not restrict the intended usage of the AGV.

The implemented localization methods for both systems must be comparable and it is regarded as of higher importance than absolute precision. The precision of both systems shall thus not be regarded as their optimal, but shall instead be evaluated in comparison to each other and when possible, towards a ground truth reference. The evaluation of both systems will be constrained to the 2D-space of the LiDAR system.

The implemented systems will not strive to achieve real-time localization as this may require sacrifices in accuracy based on the computational constraints of the implemented system.

While safety is of great importance when developing and using AGVs, it will not be of focus in the developed camera solution and will instead rely on preexisting safety systems of the provided AGV.

1.3 Thesis outline

This thesis is divided into seven chapters. It starts with this introductory chapter, followed by Chapter 2, where theory prerequisites are presented. Next, Chapter 3 presents the implementation of the systems while Chapter 4 goes over the evaluation methodology related to this thesis work. Chapter 5 presents the data from the tests and analyses the outcome of the implementations presented. Second to last in Chapter 6, are the results and choices discussed before finally in Chapter 7 ending with a summary of the project outcome based on the research questions as presented in Section 1.2.1.

2
Theory

This chapter aims to help the reader understand the concepts and methods in this thesis. It is divided into three distinct areas. First is a section about sensors and sensor data which intends to explain what sensors techniques are used and discussed. Then a section about the filters used to improve data passed to the system as well as on internal data streams. Lastly a short section about localization to give an understanding of the usage of these methods and what is required for their implementation.

2.1 Sensors and sensor data

This section focuses on a few sensor techniques for acquiring depth measurements that are relevant to this thesis and the data produced by these sensors.

2.1.1 LiDAR

LiDAR is used for detecting objects with the use of light in the form of pulsed laser beams. The distance to an object is obtained by measuring the time it takes for the beam to travel to the object, reflect, and then travel back to the sensor [16]. This is shown in Equation (2.1) where the distance d is obtained as the time t from the emission of the beam to detection times the speed of light in air c. The time is halved to only get the time of the beam traveling in one direction.

$$d = c \cdot \frac{t}{2} \tag{2.1}$$

A single static laser beam will only measure a distance in 1D [7]. These measurements can be extended into 2D by applying a known motion to the sensor [7]. A 3D-measurement can then be achieved by both applying a motion, as in 2D, and also second pivoting motion [7], or stacking multiple beams at known angles [7].

2.1.1.1 2D-LiDAR

2D-LiDAR normally only uses one laser beam to measure distances [16], but can also use multiple beams to achieve a faster sample rate by dividing the total FoV per beam instead of one beam for the entire view [7]. When using a laser beam with the sensor rotating around some defined axis, usually defined as the sensor's z-axis. For each rotation, the sensor will return several distance measurements r spaced evenly at some defined angle θ throughout the rotation, as seen in Figure 2.1. Note that some sensors have a FoV throughout the whole rotation and other does not [17]. This may be due to obstructions in the view of the sensor due to its construction.

The number of data points a 2D-LiDAR can achieve per revolution is then dependent on the sample rate, its FoV, and rotation speed [7]. In the common case where the LiDAR rotates around its z-axis the data points will span over the xy-plane of the sensor [17]. As the distance and angle for each measurement are known, the coordinate to each detection point can be represented by a polar coordinate.

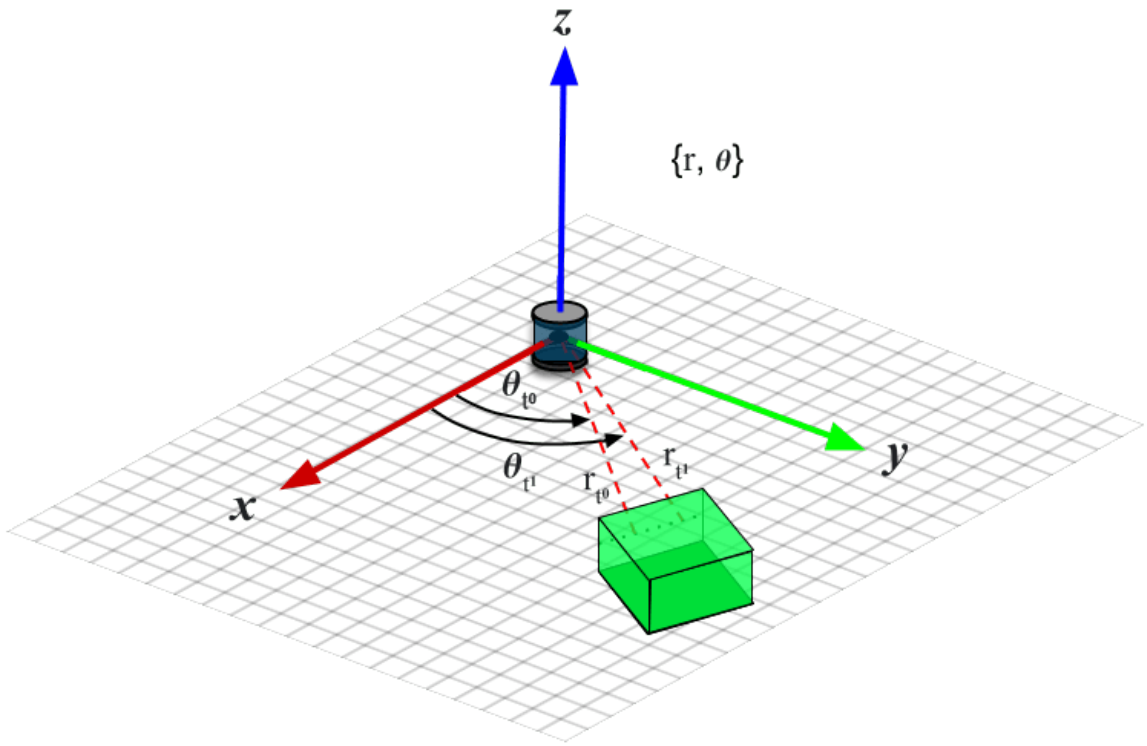


Figure 2.1: Illustration of 2D-LiDAR measurement with a single laser beam at two time steps $t_0 \rightarrow t_1$. r_t denotes the distance for which the laser beam travels at time t. θ_t denotes the angle from the x-axis to the laser beam at time t. The polar coordinate of the detection point is thus denoted as $\{r, \theta\}$.

2.1.1.2 3D-LiDAR

3D-LiDARs are similar to 2D-LiDAR in that it uses pulsing laser beams while rotating around its z-axis. In addition to rotating around one axis it will now also pivot the beam or use multiple stacked laser beams in the third dimension [7]. The pivoted or stacked laser beams are spread at some known predefined angle φ [16], as illustrated in Figure 2.2. This will lead to a collection of multiple horizontal distance measurements. These will now instead sweep cones as compared to before with 2D-LiDAR which only obtained one horizontal plane. Since each beam is sent at two known angles, φ, and θ, spherical coordinates are obtained to the objects.

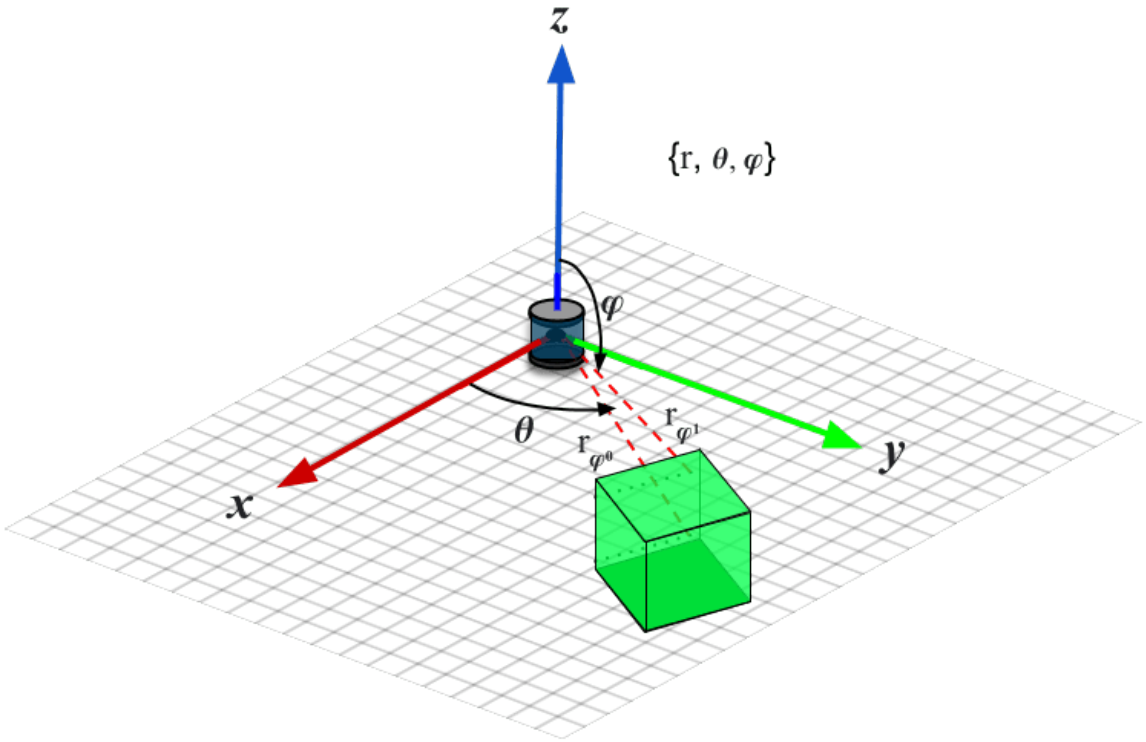


Figure 2.2: Illustration of 3D-LiDAR measurement with 2 pivoted or stacked laser beams. r_φ denotes the distance for which the laser beam travels at time t and for some angle φ. Note that φ correlates with the height of the detection point. θ denotes the angle from the x-axis to the laser beam at time t. The polar coordinate of the detection point is thus denoted as $\{r, \theta, \varphi\}$.

2.1.2 3D-camera

3D-cameras, also known as range imaging sensors, are cameras that in addition to capturing 2D images also produce depth data. There exists many techniques in which a 3D-camera can estimate depth, Stereo triangulation [18] and Structured light [19] are some of them.

2.1.2.1 Stereo triangulation

Stereo triangulation is achieved with two or more cameras that are placed in different viewing angles, in order to obtain depth from the difference of pixel position for a feature in each image [20]. Figure 2.3 illustrates the elementary form of stereo triangulation where two cameras are placed in parallel [21]. The image sensors of the cameras are placed at a known distance from each other, this distance is known as the *baseline* of the stereo camera [22].

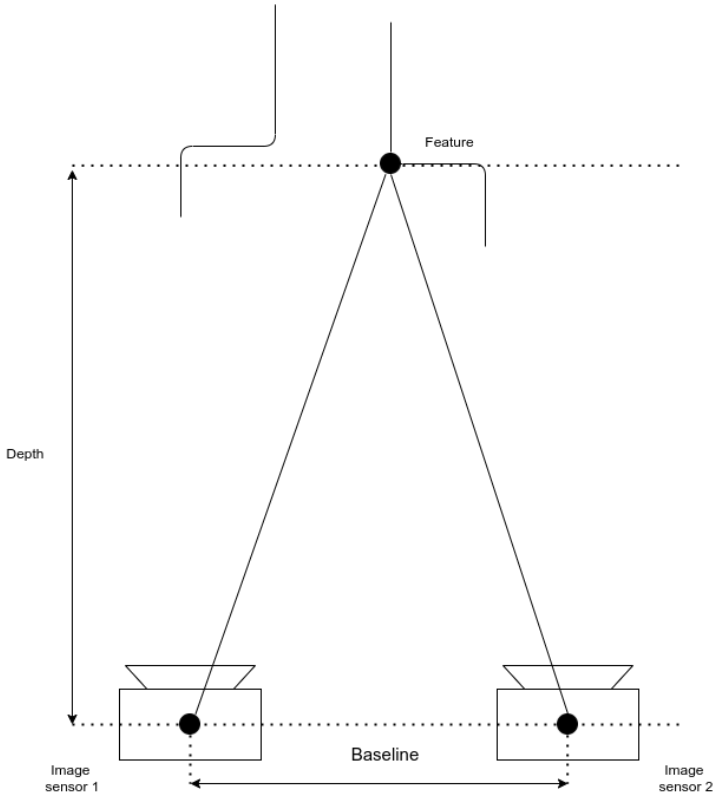


Figure 2.3: Illustration of two cameras parallel to each other detecting a feature. The depth from the cameras to the feature is obtained by triangulation.

To obtain depth from the camera images, first, unique features visible in both cameras are identified, which can be achieved by methods such as scale Invariant Feature Transformation (SIFT), Speeded-Up Robust Features (SURF) and Histogram of Oriented Gradient (HOG) [23]. Then the depth of each feature match between both views is calculated using triangulation [22].

Triangulation as seen in Figure 2.4, is done using a stereo camera with two perfectly aligned sensors separated by a known baseline distance B in meters and a known focal length f in pixels that is equal for both sensors [24]. Here Z is the depth in meters [24] to the features from the cameras calculated with

$$Z = f\frac{B}{(x - x')} \tag{2.2}$$

as described by Jain, Kasturi, and Schunck [25] were x and x' are the pixel position of the two matched features and $(x - x')$ is known as their *disparity* [21].

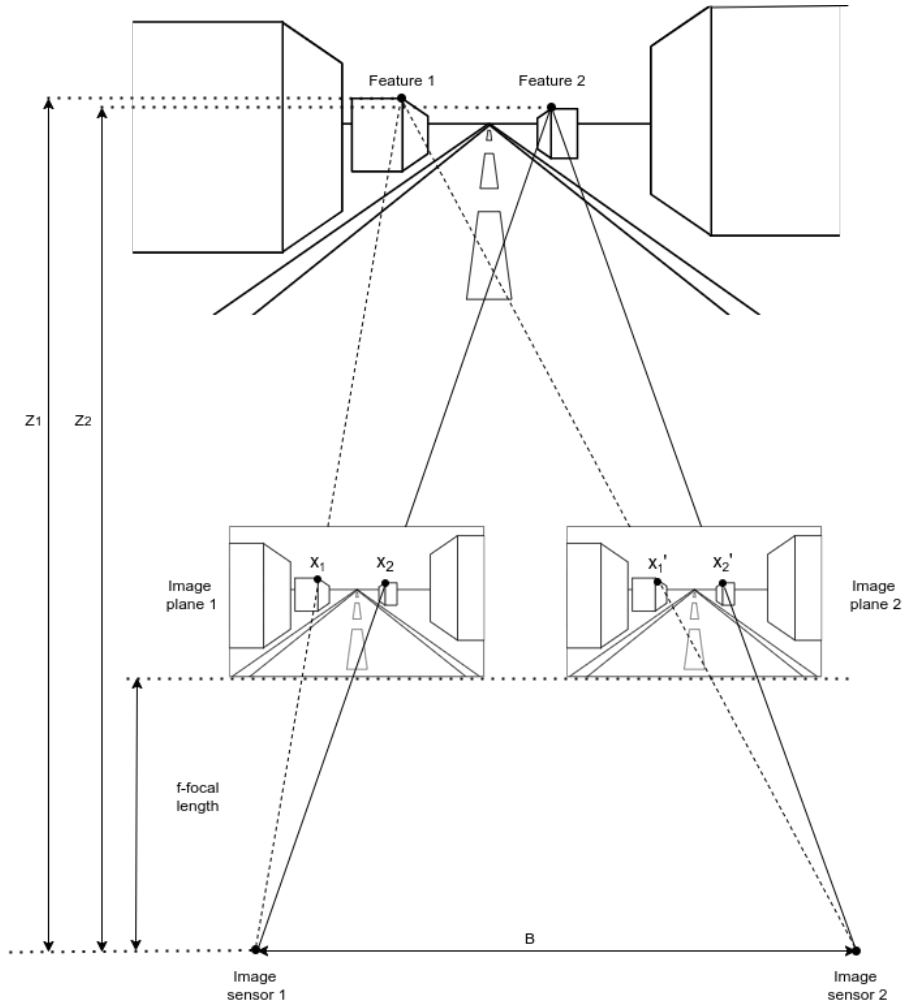


Figure 2.4: Illustration of a Stereo triangulation from two parallel image sensors capturing two features.

Due to the inherent principle of stereo triangulation relying on the known alignment between the cameras for precision is often calibration needed. There exist many different calibration schemes but from the authors, Abu Hassan, Hussain, Md Saad, *et al.*[26] it is apparent that even with calibration the depth accuracy will be far inferior to the accuracy in width and height. The authors were able to gain sub-pixel precision in width and height while the depth produced a mean error of $1.72\% \pm$ *standard deviation of* 1.6% in their test scenario. They were also not able to obtain depth data over 5.5m [26].

2.1.2.2 Structured light

Structured light is used to measure the distance to objects in three dimensions by projecting a pattern onto objects in front of a camera. Infrared light is commonly used in the projection to minimize the effects of sunlight disturbances [19]. The distance to the object in front of the camera is obtained by evaluating the distortion of the projected pattern [27]. The projected pattern can be seen as an artificial feature and is created to aid the triangulation process.

One camera is enough for extracting depth information from a scene, but it has been shown that occlusions are minimized when using cameras on opposite sides of the projector [28]. This can be seen from the illustration in Figure 2.5 where Camera 1 would only detect the left and the front face of the object. The right side is occluded by the object itself for Camera 1 but not from the projector. When instead using two cameras, both sides and the front face would be detected by the system.

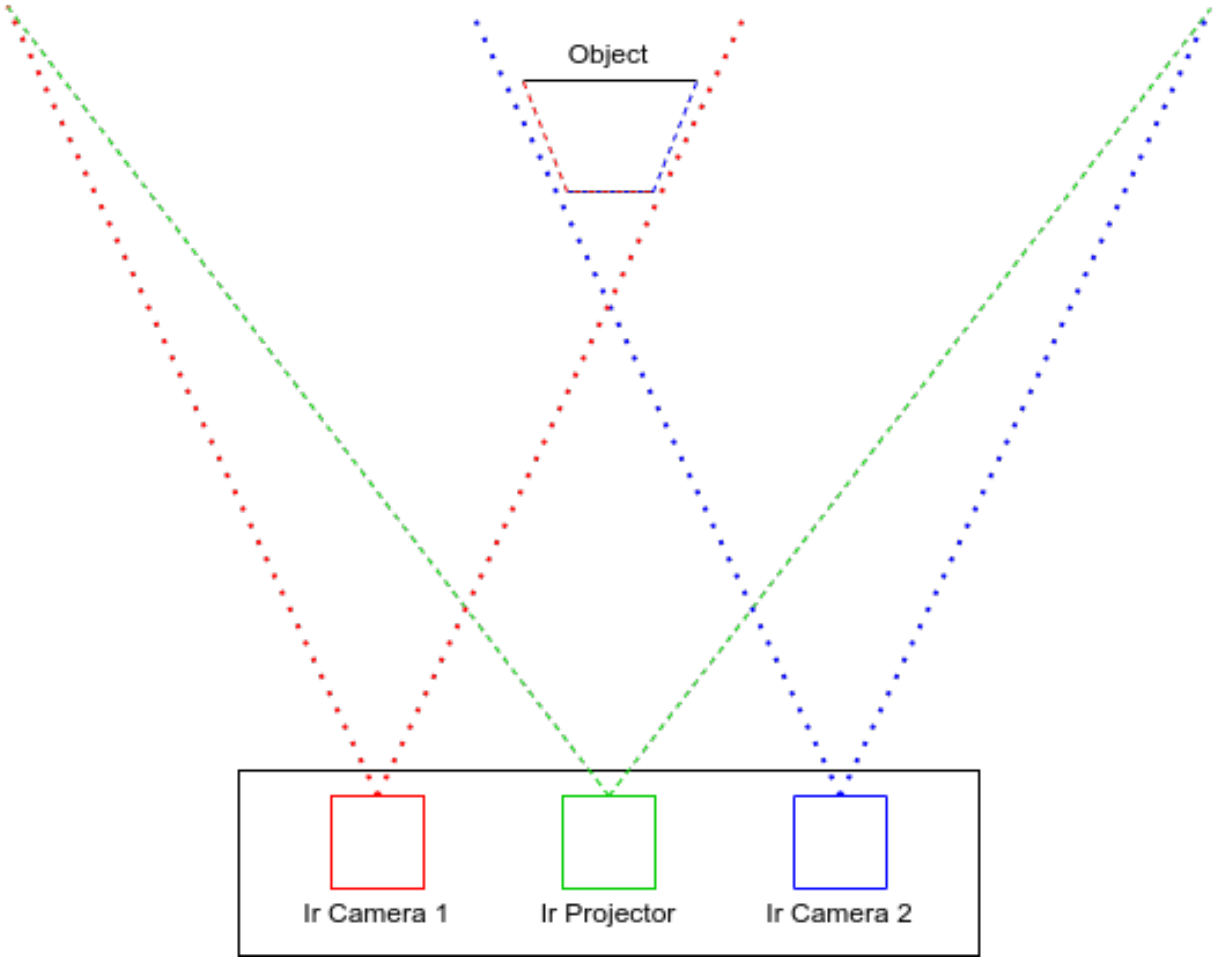


Figure 2.5: Illustration of Structured light sensor with Stereo vision. Each side is occluded from the camera in opposite direction, when combined is occlusion minimized.

2.1.3 Point cloud

A point cloud is a collection of points in 3D space [29] and is one way of describing the data acquired from LiDARs and 3D-cameras. Each point must have a coordinate, and depending on the application, also color or an illumination value [30]. The point can also be associated with the normal vector of the surface on which the point is detected [31]. The normal is useful for surface segmentation and reconstruction [32] and can be obtained by e.g. Principal Component Analysis (PCA) on a point and its closest neighbouring points [33]. A typical point cloud generated by 3D-cameras can be seen in Figure 2.6. The normal vectors of each point are illustrated by a black line.

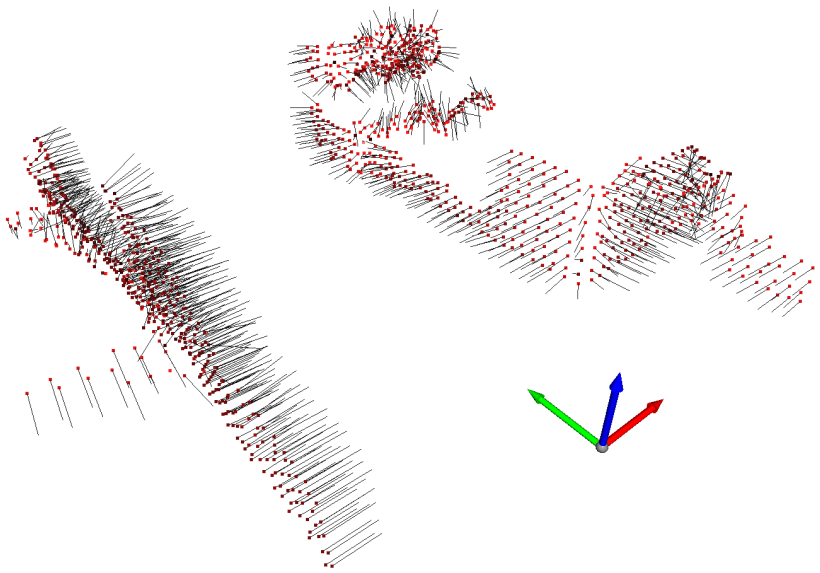


Figure 2.6: Illustration of point cloud generated by a 3D-camera (downsampled). The normal vector of each point is illustrated by a black line. The point cloud is captured while traveling down a warehouse corridor (green arrow direction).

The indexing of point cloud data can be constructed as a 1D- or 2D-vector. The 1D format is known as an unordered point cloud and is usually produced by a LiDAR [34]. The 2D format is instead known as an ordered point cloud and the data is then split into rows and columns. According to the authors, Rbrusu, Sprickerhof, Brindeiro, *et al.*[34] both stereo cameras and Time Of Flight cameras can usually produce data in this format. The ordered point cloud format resembles the structure of a 2D-cameras where the data is ordered in rows and columns based on the points x and y coordinate [34]. The advantages of ordered point clouds are the known relationship between neighboring points in operations such as nearest neighbor search which will then be much more efficient [34].

2.2 Data filtering

Data filtering is an important tool used to reduce and improve data passed through the system. By filtering raw sensor data, noise and other unwanted outlier data can be reduced. Inlier data can also be optimized to fit a selected purpose. This can be geometry identification such as planes, edges, and obstacles. The inlier data can then be modified to more accurately represent the true geometry of the real-world surface. E.g. any noise in captured data that is matched to a planar surface can be removed by projecting the point onto the plane corresponding to the planar surface.

It may not always be possible to directly match captured data to a geometry without first applying some prior filters. Captured point clouds may also be unnecessary dense and thus computationally heavy if a fast application is required. In these cases, a *down-sampling filter* may be applied. This type of filter efficiently reduces the total number of points by either combining or discarding points to a defined number or a percentage of original points. The points being combined or discarded can either be selected uniformly throughout the point cloud or be selected based on some filter-specific criteria, as explained in the following two sections.

2.2.1 Pass-through filter

A *Pass-trough filter* for the application on point clouds refers to the filtering of points that either is within or beyond some defined boundaries [35]. It can also be used to remove non-finite points [35]. The range can be either the radial distance to a point such as a camera origin, or a Cartesian distance constraint in the camera frame. For the second type, either one or multiple dimensions can be constrained, such as the height, width, or depth of the point cloud. An example of Cartesian constraints is illustrated in Figure 2.7.

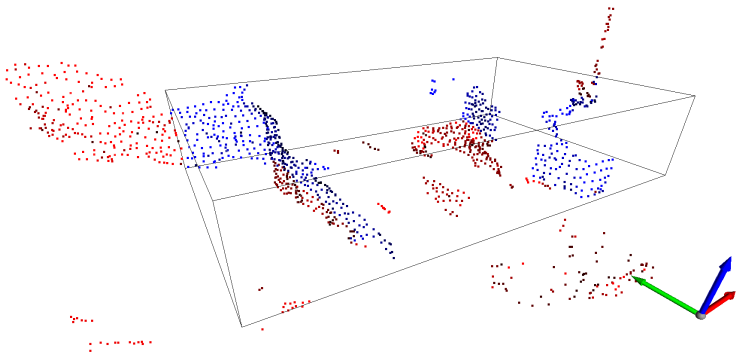


Figure 2.7: Pass-trough filter illustration. Inlier and outlier points are represented in blue and red respectively. The box shows the filter boundaries. The point cloud is captured by a 3D-camera when driving down a warehouse corridor (green arrow direction). Note that the cluster of points close to the camera (placed at origin) is due to the self-detection of the AGV. The red points in the center of the corridor are detection of the ground.

The pass-through filter can be used to efficiently reduce the number of points in a point cloud as few calculations per point are required. A simple low-pass, high-pass, band-pass, or band-stop filter is applied for the coordinate value or radius that is filtered. If one coordinate value breaks the constraint, the entire point is deemed invalid and is removed.

The filter can also be used to target a specific volume that is either to be discarded or extracted. A volume that may be of sole interest e.g. for object avoidance can be that which is: below the maximum possible height of a loaded AGV and not below the height of the ground plane. A common volume to be removed is that which corresponds to the volume of the AGV itself. This is done to efficiently remove any self-detected points that may occur if the sensor is partially obscured by the AGV. Due to the inherent principle of stereo triangulation used by some 3D-cameras, as described in the Stereo triangulation section, the precision is often far greater in width and height than in-depth. Thus a specialized pass-through filter may be applied which limits the depth of the point cloud more than width and height.

2.2.2 Decimation filter

A decimation filter is one type of downsampling filter that is applied to the 2D stereo-images produced by the cameras. The downsampling is achieved by combining the pixels of 2D-stereo images uniformly to a defined rate, and therefore results in fewer 3D-points generated by the images [36]. The filter reduces the size of the images by applying a kernel matrix which averages the data of pixels contained by the kernel. The kernel is passed throughout the entire image either column- or row-wise and in a moving window fashion [37]. The rate at which the kernel is moved is defined as the filter stride. The stride is equal to the number of pixels the kernel is moved at each time. A kernel is usually square and can be designed to different sizes depending on how much downsampling is desired. The kernel matrix can be tuned for specific applications but two simple implementations are to use the median or mean of the kernel patch [37]. Figure 2.8 illustrates an example of a decimation filter that uses an averaging kernel with the size of two by two pixels and a stride of two pixels. This will cause the original four pixels to be merged into one.

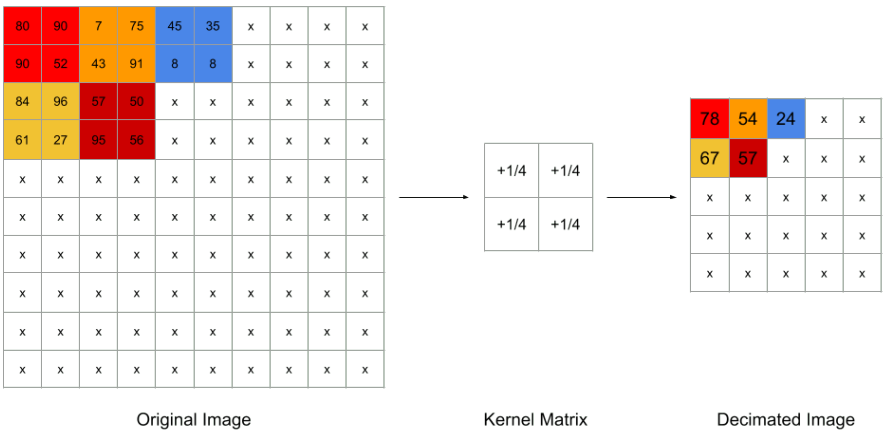


Figure 2.8: Illustration of decimation filter with an averaging kernel matrix of size two by two with a stride of two.

When using a kernel size of two by two and a stride of two the image size will be one-quarter of the original image size. Kernel size of three by three with a stride of three the image size will be one-ninth of the original. It is also possible to have a smaller stride than the size of the kernel. This will cause a smaller reduction in resolution but with a larger number of pixels being merged than with equal size and stride.

2.2.3 Voxel filter

A Voxel filter downsamples a point cloud by first sorting all points, based on their coordinate, into a 3D-grid of cubes. These cubes are called voxels and have a defined size [38]. The voxels are aligned to a 3D-grid, called voxel-grid, spanning from the lowest coordinate values of the point cloud to the highest, as to contain all points. These points define the upper and lower bounds of the voxel-grid as can be seen in Figure 2.9.

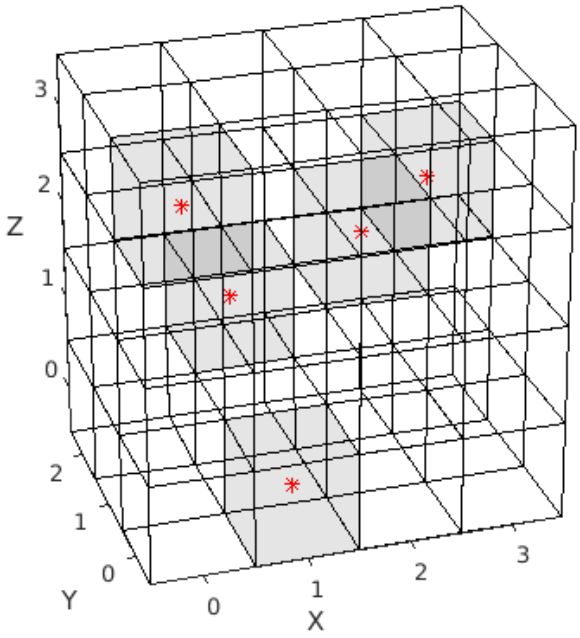


Figure 2.9: Voxel grid containing all points in the point cloud. The voxel grid origin is aligned to the lowest independent coordinate values of all points in the filtered point cloud and the grid spans to contain all points in the point cloud.

All points that are within the same voxel will be merged into one based on a defined criteria, e.g. mean coordinate value. This point is called the centroid. The points can also be merged by their median coordinate. The color and normal vectors of the points are also averaged by their mean or median value.

In Figure 2.10 are four different illustrations of a voxel. The one in the lower right corner shows a voxel that only contains one point ∗ and thus the centroid ∘ is placed at the same coordinate. The upper right voxel contains two points and a centroid in between the two points. The upper left voxel illustrates a scenario where you have a set of multiple points inside the voxel that all affect the location of the centroid. The last voxel illustrates an edge case where each corner of the voxel contains a point and thus will the centroid be in the center of the cube.

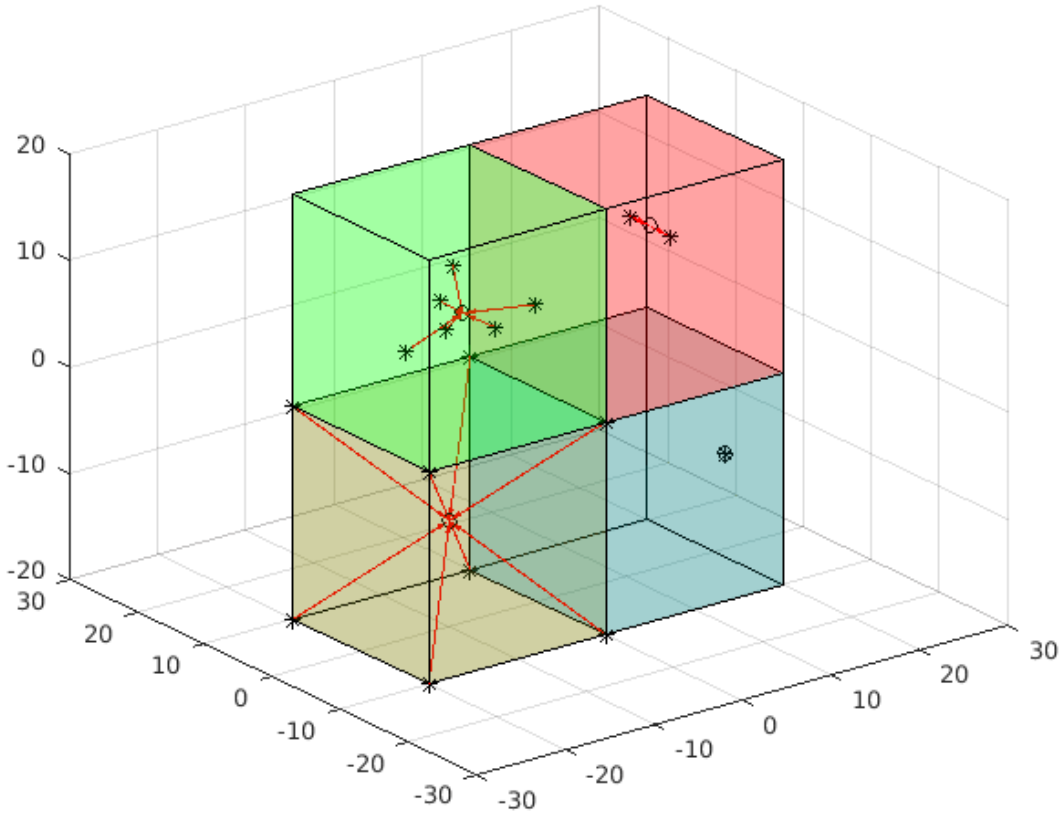


Figure 2.10: The lower right Voxel shows only one point ∗ an therefore the centroid ∘ lies in the same place. The upper right Voxel shows two points and the centroid in between. The upper left voxel shows a random set of points and the resulting centroid. The lower left voxel shows a set of points in each corner and a calculated centroid in the center of the voxel.

One voxel will not necessarily contain the same number of points at all times and will thus downsample differently in different point clouds. A point cloud with many points which are near will have a large reduction per volume of points, whereas a sparse cloud may only have a small reduction. Hence, as the voxel size is defined and constant, the maximum resolution of the cloud can be controlled. Practically this means that points unnecessarily densely packed to e.g. a basic geometry such as a plane can be significantly reduced. As the output of each voxel is one point, the highest density of a point cloud on average is that of the voxel size, or less. Because of this, the voxel size is naturally defined as the desired resolution of the filtered point cloud. A well-selected voxel size shall, for high reduction, give enough resolution to not miss important features but still remove as many points as necessary. The voxel downsampling filter as implemented in [39], is described in Algorithm 1.

Algorithm 1: Voxel downsampling filter

Input: Input point cloud P, Voxel size r
Output: Downsampled point cloud P_{ds}

$\mathbf{x}_{min} := min(P_{(\mathbf{x})}) - \frac{r}{2}$ // Voxel-grid lower bound point
$\mathcal{I}_{vox} := \emptyset$ // Voxel index list
$P_{ds} := \emptyset$
for $u \in P$ **do**
  $\mathbf{x}_{ref} := \frac{u_{(\mathbf{x})} - \mathbf{x}_{min}}{r}$
  $\mathcal{I}_{vox} \longleftarrow \mathcal{I}_{vox} +\!\!+ \left\lfloor \mathbf{x}_{ref} \right\rfloor$
for $v \in \mathcal{I}_{vox}$ **do**
  $\bar{\mathbf{x}} := \bar{v}_{\mathbf{x}}$ // Mean coordinate
  $\bar{\mathbf{n}} := \bar{v}_{\mathbf{n}}$ // Mean normal
  $\bar{c} := \bar{v}_c$ // Mean color
  $P_{ds} \longleftarrow P_{ds} +\!\!+ [\bar{\mathbf{x}},\ \bar{\mathbf{n}},\ \bar{c}]^T$

The voxel filter is given an input point cloud P and the desired voxel size r, defined in meters. Thus a r-value of 0.05 will give a voxel cube of $5 \times 5 \times 5$ cm. The lower bound of the voxel grid is then calculated as $min(P_{(\mathbf{x})}) - \frac{r}{2}$ where $min(P_{(\mathbf{x})})$ is the lowest x, y and z-value of all points independently. $\frac{r}{2}$ is subtracted to ensure that the first point is placed inside a voxel. A voxel index list, $\mathcal{I}_{vox}$ and an output point cloud P_{ds} is initiated. Then, for each point $u_{(\mathbf{x})}$ a reference coordinate $\mathbf{x}_{ref}$ is obtained as $\frac{u_{(\mathbf{x})} - x_{min}}{r}$. The reference point is constructed to place the point in its corresponding voxel in the voxel-grid coordinate system. Each voxel is indexed by its 3D-coordinate in the voxel grid. The voxel-grid coordinate system spans from zero to the number of voxels in width, height, and depth necessary to contain all points. The floor-value of the reference coordinate is then appended to the voxel-list, which means that the point is now assigned to that corresponding voxel. It is expected that multiple points are appended to the same list entry, which means that they coexist in the same voxel. Finally, for each non-empty voxel, the mean value of point coordinates, normal vectors, and colors are obtained and appended to the output point cloud P_{ds}. The number of points has then been reduced from the original number of points to the number of non-empty voxels.

2.2.4 Spatial edge-preserving filter

A spatial edge-preserving filter is used to average data while still preserving edges and corners [36]. This is done by applying a high dimension transform that will result in an image in the transform domain with a low complexity instead of the original high dimension complexity as it would be if the filter were to be applied in the original transform space [40]. The image in the transform domain will then be filtered through a space invariant filter before being turned back into the original domain. Moving the image into the transform domain will not only reduce the complexity but will also preserve the sharp lines that exist in the image while smoothing the image. For an application on 3D-cameras, specifically, cameras which obtain depth from the triangulation through disparity images, a spatial edge-preserving filter may be beneficial as it can be used to remove noise from the disparity image while still preserving edges [41].

As implemented by the Intel Realsense stereo cameras [41], one way to achieve a spatial edge-preserving filter is by first raster-scanning the depth-image produced by the sensor, in x-direction, y-direction and then back, creating four consecutive passes, resulting in the 1D representation of the depth-image, $\mathcal{Z} = [Z_1, \ldots, Z_{4\times n_{pixels}}]$. An averaging coefficient α is then applied on each depth value based on the trailing neighboring value. If an edge is detected throughout the scan, no averaging is applied. This will cause an artifact depending on the value of the trailing neighbor, and therefore each axis is scanned in both directions to compensate.

The averaging coefficient is determined by an exponential moving average (EMA) [41], where the specific recursive function is defined as:

$$S_t = \begin{cases} Z_1, & t = 1 \\ \alpha Z_t + (1-\alpha) S_{t-1}, & t > 1 \wedge \Delta = |S_t - S_{t-1}| < \delta_{thresh} \\ Z_t, & t > 1 \wedge \Delta = |S_t - S_{t-1}| > \delta_{thresh} \end{cases} \tag{2.3}$$

and where Z_t is the depth value and S_{t-1} is the EMA at some time instance t. Edge preserving is achieved by evaluating the magnitude of discontinuity $\Delta = |S_t - S_{t-1}|$. If the magnitude is larger than a defined threshold δ_{thresh} an edge is detected and the filter is temporarily inactivated, i.e. $S_t = Z_t$.

The resulting sequence of filtered depth-values $\mathcal{S} = [S_1, \ldots, S_{x\times n_{pixels}}]$ are then combined per pixel according to the reverted order of the raster-scanning and averaged. This gives the final filtered depth-image back in 2D. An illustration of a simulated case of the filter were the dept-image is projected as 3D points can be seen in Figure 2.11.

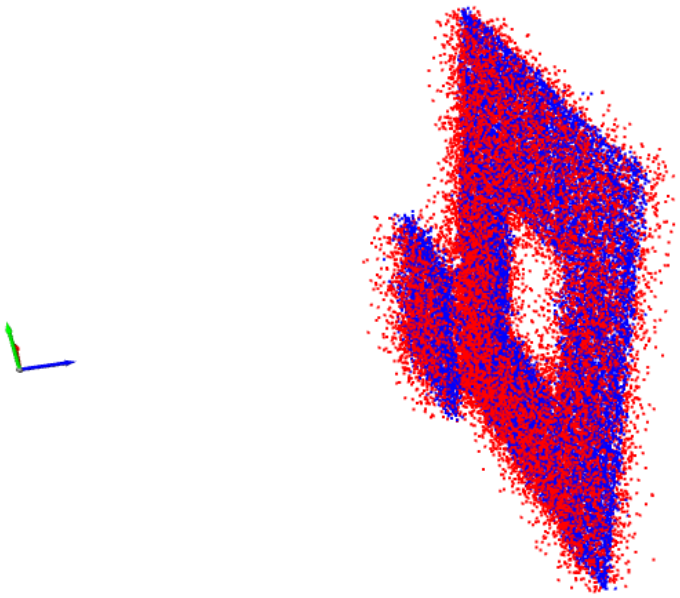


Figure 2.11: Point cloud simulation of a camera capturing a box protruding from a wall. Red illustrates noisy data and blue the data filtered by a spatial edge-preserving filter.

Figure 2.12 shows a 2D zx-slice in the center of the point cloud in Figure 2.11. In auditing to the noisy and filtered data, the true reference data as well as a Gaussian-blur [42] filtered data is also shown. The Gaussian filter is added to illustrate the downsides of using a simpler averaging filter.

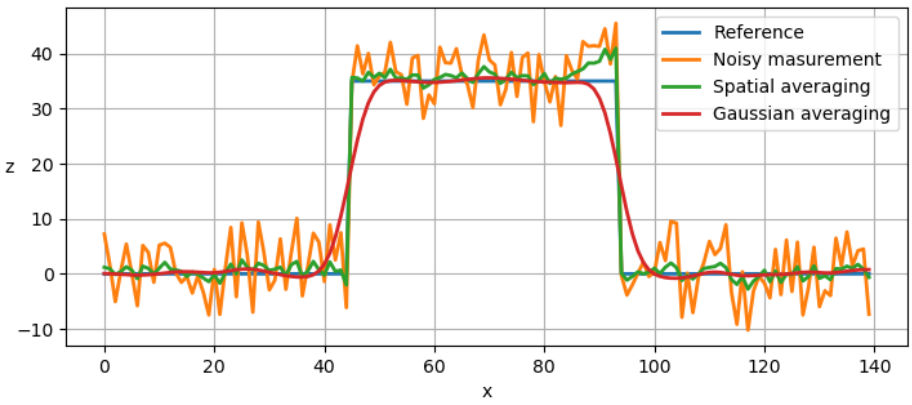


Figure 2.12: zx-slice in center of the point cloud in Figure 2.11.

The benefit of the spatial edge-preserving filter is well apparent when compered to the Gaussian-blur filter in Figure 2.12. The edges are deliberately preserved whereas the Gaussian curve is heavily rounded. Note that some smoothing, but no over-smoothing, of the edges are seen in the spatial curve as well. This is due to the y-scan still smoothing the sample as it is scanned parallel to the edge but does not pass it.

2.2.5 Temporal averaging filter

A temporal averaging filter utilizes knowledge from prior data acquisitions to improve new data [43]. This is done by averaging the position of a particle in 3D-space based on its prior positions which will reduce or filter random noise generated by the data capture [44]. However, as the filter incorporates prior data, unconsidered disturbances such as sensor motion can negatively impact the data [43]. To mitigate this effect, we present a method where the senor motion is measured. By including the knowledge of the sensor motion and transforming prior data based on this motion, new data can more accurately merge the prior data. The prior data is transformed from the frame it is captured in i.e. the center of the camera frame, back to the position where it was captured. This can be visualized as the camera moving in the world frame but the prior point cloud staying stationary. This is illustrated in Figure 2.13 as the transition of a camera frame **F** through the time step $t_k \rightarrow t_{k+1}$.

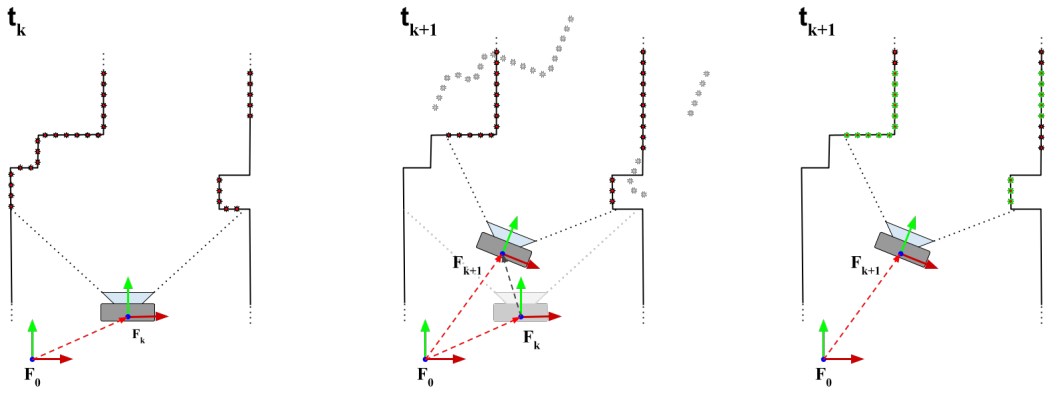


Figure 2.13: Temporal filter illustrated by merging points from camera frames $\mathbf{F}_k$ into $\mathbf{F}_{k+1}$ through time step $t_k \rightarrow t_{k+1}$. $\mathbf{F}_0$ denotes the static world frame.

It is important to note that the quality of the transformation of prior data is directly dependent on the motion estimation. Therefore the precision of the transformation can not be better than the underlying location estimation on which the motion estimation is based. To further improve on the transform estimation Iterative Closest Point (ICP) can be applied. ICP is the method of optimizing a registration between two 3D geometries as described by Besl and McKay [45].

If a prior point cloud is the result of additional filtering and thus considered as inliers, and if these points can match with new points, these may also be considered as inliers. The matched point does then not need to be filtered. However, it is important to note that, as prior points which have been evaluated as inliers may have been so within some defined range of margin, and as it may be necessary to add some margin to match new point to the prior, additional margin is now added to the previous. Thus, it is necessary to keep the criteria for assigning both filtered, as well as matched points as inliers, small enough such that drift due to accumulated error is kept to an acceptable level. Specifically, the combined error margin for inlier evaluation and temporal matching shall be less or equal

to the total acceptable error. An illustration of ill-defined error margins causing an invalid match can be seen in Figure 2.14. The potential drift can also be reduced by weighting the position of a new point higher when merging with its prior. A higher weight on the position of new points will however also give a higher emphasis on any newly introduced noise, in comparison to the already filtered position of prior points.

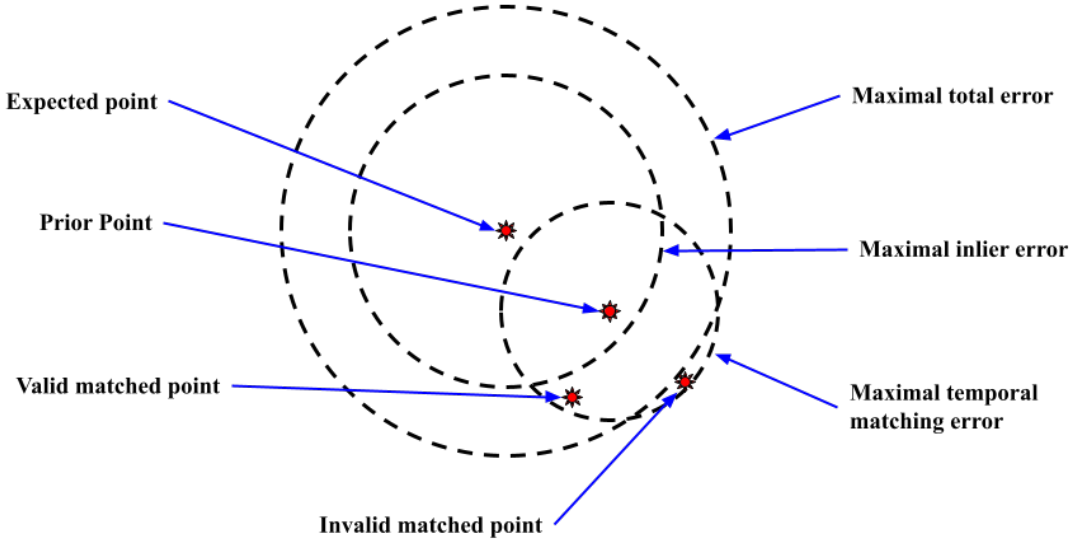


Figure 2.14: Illustration of a temporal filter with an ill-defined error constrain allowing the matching of an invalid point. The expected point is the point that equals the optimal inlier by some prior filter. The maximal inlier error is the proximity to the expected point in which the prior point must be within to still be evaluated as an inlier by the prior filter. The maximal temporal matching error is the proximity in which a new point must be to the prior point to be matched. The illustrated case shows a defined error margin in inlier- and matching-error causing a match of a point that is not within the maximal total error. Note that only inlier and matching errors are checked by the filters and that the invalid matched point exceeding total error would not be detected. Therefore inlier and matching error combined shall be defined as less or equal to the total acceptable error.

Another benefit of the temporal filer is that attributes assumed to be constant in time, such as an accurate estimation of a surface normal, can also be kept throughout iterations, instead of being recalculated. This can greatly reduce the total computational burden of each point cloud filtering. By also assuming that any optimization has been applied to the prior point, such as normal realignment of points corresponding to a known surface, then the resulting merged point is more optimal than if no temporal filtering had been applied. Computationally heavy identification operations such as point cloud segmentation [46] can also be significantly accelerated by storing identifiers, i.e. labels, of already identified geometries throughout temporal iterations.

2.2.6 Plane-fitting filter

A plane-fitting filter is used to segment and fit, points corresponding to planar surfaces into perfect planes. This is done in an sequential process as described in Algorithm 2 and which is based on the concepts by Bondemark [47], Salas-Moreno, Glocken, Kelly, *et al.* [48] and Trevor, Gedikli, Rusu, *et al.* [49], but were we present a version with the ability to handle unordered point clouds. The goal of the filter is to improve the registration of points that correspond to planar surfaces, such as panels, walls, and floors, by projecting them onto perfect planes. An illustration of the projection can be seen in Figure 2.15.

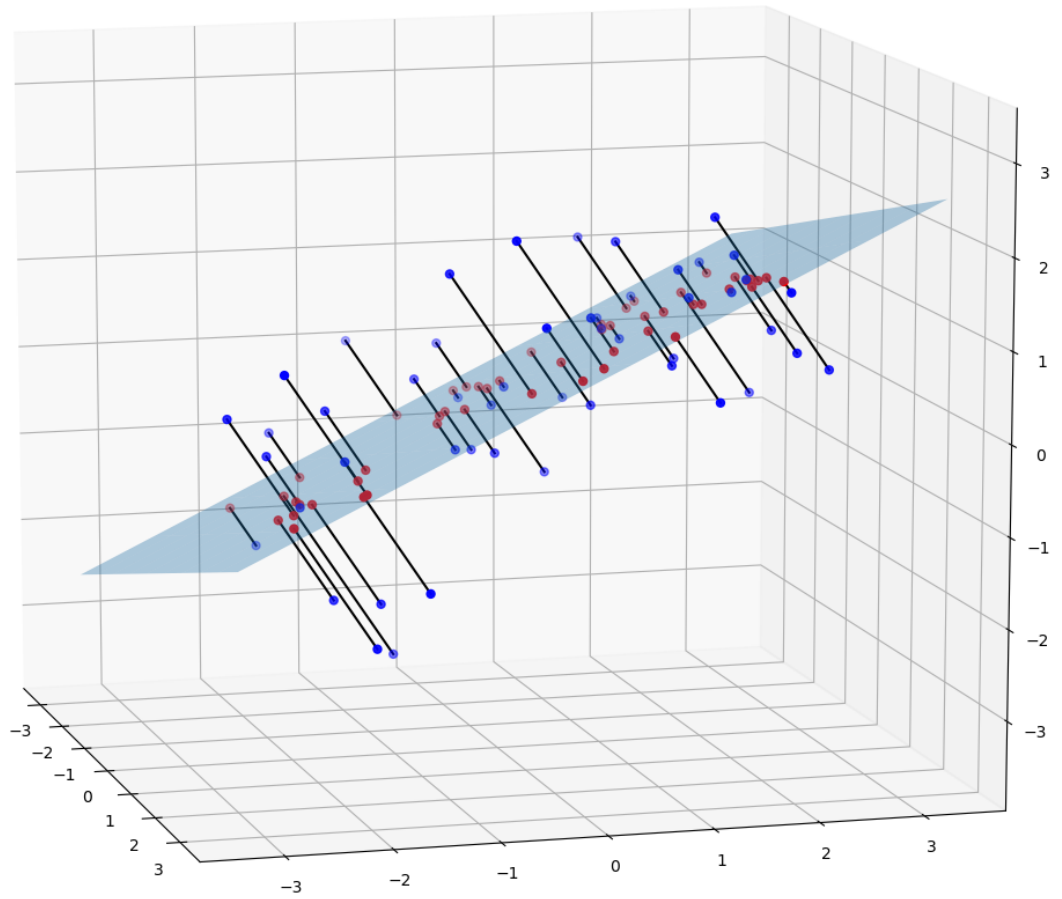


Figure 2.15: Points of a noisy measurement projected onto a perfect plane. Projection vectors are illustrated by the black lines.

Algorithm 2 illustrates the key sequential stages in a plane-fitting filter. The filter is designed to work on unordered point clouds and with no temporal knowledge. However, great improvement in throughput speed has been achieved by passing labeling information through the earlier mentioned temporal filter and is further explained in Section 2.2.6.4 Plane labeling.

Algorithm 2: Plane-fitting algorithm

Input: Input point cloud P, number of nearest neighbours k_{nn}, distance limit coefficient δ, angle limit θ_{lim} and minimum points in plane ϵ

Output: Inliers projected to planes $\bar{P}$, Inlier map $\mathcal{L}$, Average projection error $\bar{d}$

$\mathcal{I} \longleftarrow$ `search_knn`$(P,\ k_{nn})$ `// Search` k_{nn} `for each point in` P

$\mathcal{N} \longleftarrow$ `estimate_normals`$(P,\ \mathcal{I})$ `// Estimate normals for each point in` P

for $u \in P$ **do**

  $\mathcal{D} \longleftarrow \prod_{u_n \in \mathcal{I}(u)} \mathbf{S}(u, u_n)$ `// Build discontinuity map`

$\mathcal{L} \longleftarrow$ `label_plane`$(P,\ \mathcal{D},\ \mathcal{I},\ \epsilon)$

for $l \in \mathcal{L}$ **do**

  $\Pi \longleftarrow$ `estimate_plane_model`$(P[l])$

  $\bar{P}, \bar{d} \longleftarrow$ `project_points_to_plane`$(\Pi,\ P[l])$

2.2.6.1 Nearest neighbour search

To determine if a point is part of a plane, first, k_{nn} nearest neighbors for each point in point cloud P is obtained by a search algorithm such as Fast Approximate Nearest Neighbor Search Library (FLANN) [50]. The number of neighbors necessary is dependent on subsequent affected stages such as normal estimation and discontinuity mapping. For normal estimation k_{nn} depends on the desired quality. A larger number of neighbors will give a more accurate normal estimation but with the cost of a decrease in computational performance, as shown by Klasing, Althoff, Wollherr, *et al.* [31]. For discontinuity evaluation the number of neighbour evaluated is implementation specific and can be considered a tuning variable and where a larger number results in a harder constraint.

2.2.6.2 Normal estimation

For each point $u \in P$ the corresponding surface normal $n \in \mathcal{N}$ is estimated by a Principal Component Analysis (PCA) on the coordinate covariance matrix $\mathcal{C} \in \mathbb{R}^{3x3}$ of the point and its neighbours $x_k \in \mathcal{I}[u]$. $\mathcal{C}$ is defined as:

$$\mathcal{C}_u = \frac{1}{k_{nn}} \sum_{i=1}^{k_{nn}+1} (x_i - \bar{x}) \cdot (x_i - \bar{x})^T, \qquad x \in [x_u, x_{k_1}, \ldots, x_{k_{nn}}] \tag{2.4}$$

and the normal estimation is then obtained as the eigenvector corresponding to the smallest eigenvalue of $\mathcal{C}_u$ [51]. However, as there is no mathematical way to determine the correct sign of n via PCA [51], an additional evaluation is required. By assuming that all points are acquired from the same sensors and which itself is assumed to only acquire points in its view, i.e. of non-occluded geometries, all points detected must be of surfaces that normal is pointed in some parts toward the camera. That is, the component of the normal corresponding to the depth-axis of the capturing sensor must be negative. As a result, all estimated normals with a positive depth-component shall be inverted [51].

2.2.6.3 Discontinuity map

A Discontinuity map $\mathcal{D}$ is a binary list with one entry corresponding to each point in P and where a one indicates that the point is continuous. A point is continuous if it is part of local plane and is evaluated toward its k_{nn} nearest neighbours $u_n \in \mathcal{I}[u]$. The evaluation, as seen in Equation (2.5), consists of an angle criterion and a distance criterion that both must be met for all neighbors to the point.

$$\mathbf{S}(a,b) = \begin{cases} 1 & \text{if } \underbrace{||P(a)\cdot\mathcal{N}(a) - P(b)\cdot\mathcal{N}(b)|| \leq \delta P(a)^2_{(z)}}_{\text{distance criterion}} \wedge \underbrace{\mathcal{N}(a)\cdot\mathcal{N}(b) \leq \cos(\theta_{lim})}_{\text{angle criterion}} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

The distance criterion of Equation (2.5) assert that the perpendicular distance component of the two points a and b is within a defined threshold [49]. The angle criterion constrain the maximal angle deviation of the points normal in reference to its neighbors, i.e. the curvature of the surface [49]. The tuning parameter δ shall correspond to the depth precision of the input data and θ_{lim} the maximum acceptable angle deviation between the normal vectors.

2.2.6.4 Plane labeling

Plane labeling refers to the labeling of points corresponding to the same plane. For an ordered point cloud, continuous points i.e. points corresponding to a local plane [48], can be labeled by a Connected Component Labeling-algorithm (CCL) as described by Salas-Moreno, Glocken, Kelly, *et al.* [48]. However, CCL is not applicable for an unordered point cloud as no positional information can be obtained by its indexing. A proposed method for labeling unordered points by instead cluster points with all continuous neighbors is described in Algorithm 3. The algorithm is given an unordered point cloud $\mathbf{P}$, a discontinuity map $\mathcal{D}$ and a neighbor list $\mathcal{I}$. The algorithm then returns a label map $\mathcal{L}$. An optional variable ϵ may be passed which defines the minimum number of points required for a plane label to be stored, otherwise, it is discarded.

Algorithm 3: Plane labeling

Input: point cloud $\mathbf{P}$, discontinuity map $\mathcal{D}$, nearest neighbours list $\mathcal{I}$, min cluster size ϵ

Output: Cluster map $\mathcal{L}$

1 $u_{seed} := \texttt{random}(u \in P \mid \mathcal{D}[u] = 1)$ `// first point to grow label-set from`

2 $\hat{\mathcal{L}} := \left[[u_{seed}]\right]$ `// append seed to first label`

3 $\mathcal{D}[u_{seed}] := 0$ `// indicate that the seed is labeled`

4 $u_{new} := u_{seed}$

5 **while** $\mathcal{D} \neq \boldsymbol{0}$ **do**

`// get neighbours of new points`

6 $u_{neigh} := \mathcal{I}[u_{new}]$

`// get unique neighbours which are to be sorted`

7 $u_{new} := \texttt{unique}(u \in u_{neigh} | \mathcal{D}[u] = 1)$

8 **if** $u_{new} = \boldsymbol{\emptyset}$ **then**

`// no new points, make new cluster`

9 $u_{seed} := \texttt{random}(u \in P \mid \mathcal{D}[u] = 1)$

`// append seed as new cluster`

10 $\hat{\mathcal{L}} \longleftarrow \hat{\mathcal{L}} +\!\!+ [u_{seed}]$

11 $\mathcal{D}[u_{seed}] := 0$

12 $u_{new} := u_{seed}$

13 **else**

14 $\mathcal{D}[u_{new}] := 0$

`// append new points to last cluster`

15 $\hat{\mathcal{L}}_{(last)} \longleftarrow \hat{\mathcal{L}}_{(last)} +\!\!+ u_{new}$

`// only return cluster greater or equal to min cluster size`

16 $\mathcal{L} \longleftarrow \forall l \in \hat{\mathcal{L}} \mid l \in \mathbb{N}^n | n \geq \epsilon$

A significant speedup in plane labeling can be achieved by incorporating labels from prior matching. This by including the plane label of each matched point in the prior point set, used by a temporal filter, see Section 2.2.5. By assuming that a new point has been successfully matched to a prior point by a temporal filter and where the prior had a valid plane label, then the new point shall also attain the same plane label. The speedup can thus be achieved by first evaluating each pre-existing plane label from the prior points, as these are known to be part of a valid plane. Specifically by, as seen in Algorithm 3 on line 1 and 9, instead of seeding with a random point $u \in P \mid \mathcal{D}[u] = 1$, seed with all valid points from a prior plane label. That is, all points from a prior plane label that satisfies $u \in P_{prior} \mid \mathcal{D}[u] = 1,\ \forall P_{prior} \in \mathcal{L}_{prior}$. By this implementation, a new plane label is first initiated as all valid prior points to the same plane. Then, similar to the original algorithm, any valid points not labeled are evaluated. Note that, new points that are not corresponding to a prior plane label can still be matched to the same plane by the algorithm. This is correct as parts of a plane that previously may have been obstructed can now have come into view and shall thus be appended to the same plane.

2.2.6.5 Point to plane projection

For each label $l \in \mathcal{L}$ a plane model Π in general form [52] is estimated:

$$\Pi : ax + by + cz + d = 0 \tag{2.6}$$

The model is estimated by a Random sample consensus (RANSAC)-algorithm. The algorithm iteratively tries to find the plane model which minimizes the total projection error for each point $u \in l$. It does this by randomly selecting three points from the label set and then produces a plane satisfying Equation (2.6). The plane estimation is then evaluated on all points and with the goal of obtaining as many inliers as possible. A point is determined to be an inlier if its distance to the plane is within a defined margin. Only the model with an increasing number of inliers is kept till the next iteration. The algorithm iterates either until a defined number of iterations are reached, or if the mean distance error for all points to the optimal plane estimation falls below a desired threshold. Finally, all valid points are projected to planes based on their corresponding plane model:

$$n = [a,\ b,\ c]^T \tag{2.7}$$

$$p = \frac{d}{\sqrt{a^2 + b^2 + c^2}} \tag{2.8}$$

$$D = n \cdot x + p \tag{2.9}$$

$$err = Dn \tag{2.10}$$

$$\bar{x} = x - err \tag{2.11}$$

where D is the point to plane distance [52]. Additionally, the mean projection error $\bar{d}$ for each plane is obtained.

Finally, both the projected points $\bar{P}$ and mean projection errors are returned by the algorithm. The projection error can then be utilized by subsequent filters to determine if the projected plane is viable.

2.3 Localization

Localization is one of the more important features of complex AGVs and the ability to accurately sense and estimate the location of a platform, lies at the heart of almost all AGV applications according to Durrant-Whyte, Rye, and Nebot [53]. In order to perform localization in an environment must a map of the environment exist. For localization in situations where the system has no prior data of the environment or where the environment does not stay the same, the system must then simultaneously create a map of its environment and keep track of its position in that map, this is known as SLAM [54], [55]. To extend the reach of the map the position of the sensors must travel, if the sensor is mounted to a platform and this moves so will also its location in the map.

One thing in common for 2D-LiDAR, 3D-LiDAR, and 3D-Cameras used on AGVs is that these sensors can be used to simultaneously create a map of the environment since the position of the sensors relative to the platform is known, the platform's position in the environment can be estimated [56]. Many different algorithms are classed as SLAM-algorithms. Some use artificial landmarks [57], such as reflectors and AprilTags [58] to achieve higher performance. Some algorithms rely only on natural landmarks such as corners, pillars, and walls. The latter may be referred to as Natural Localization [59] and no modifications of the environment are required.

As both LiDARs and cameras use measurements based on the reflection of the surrounding surfaces are they all sensitive to effects that may distort these reflections. In situations that may cause incorrect readings, such as shiny but transparent windows, additional sensors may be used to aid in the localization. This can be an Inertial Measurement Unit (IMU) which can sense transitional and rotational accelerations [60]. Wheel odometry can also be used to calculate the relative motion of the AGV based on the rotation of the wheels.

3
Implementation

This chapter will present the hardware and software implementations of the developed localization system.

3.1 Test platform

An industrial AGV is used as a test platform to emulate a real-world application of the system. The AGV allows for a rigid mounting of sensors and also predictable travel patterns due to its differential steering. This allows for consistent testing in different environments as the physical construction of the system shall not change and therefore any change is related to the environment, not the systems and test platform itself. The AGV is supplied by FlexQube and is of their eQart line of AGVs. The supplied AGV contains its own LiDAR navigation system, this is not used in order to not disrupt its safety system. However, the AGV can supply wheel odometry data without interference. The AGV can be seen in Figure 3.1.

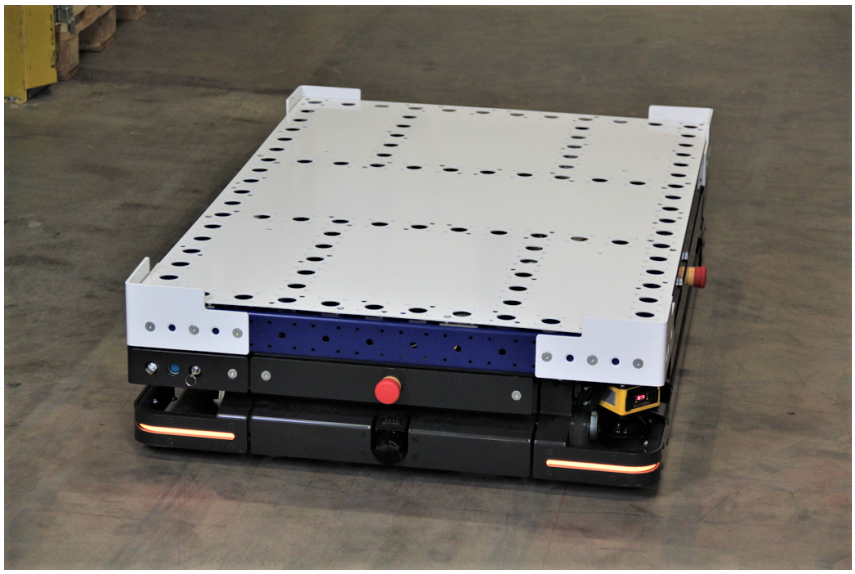


Figure 3.1: eQart supplied by FlexQube.

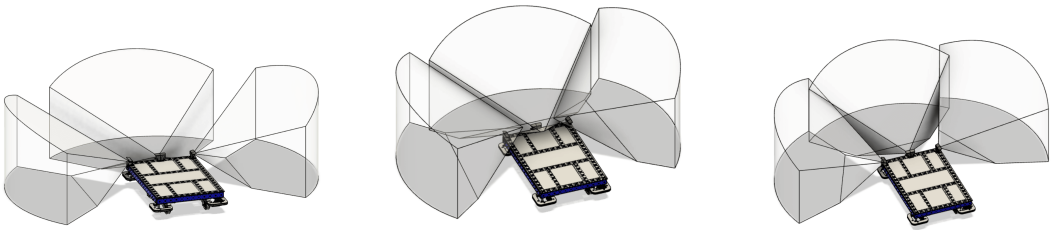
As shown by Silva Neto, Lima Silva, Figueredo, *et al.* [61] from their testing of ten different RGBD-cameras are Orbbec Astra Pro, Asus Xtion PRO live, and Realsense D435 identified as good choices for use on mobile platforms such as robots. The D435 RGBD-camera was also used by Manhed [12] and from his testing it was argued that

a longer range may have been needed. The Realsense D400 series have the ability to arrange the cameras in an outward-facing configuration and synchronize the shutters between each camera, in practice this creates a sensor arrangement that has a much wider FoV than the single camera[62]. No other cameras on the market were found to have this function. These factors combined made the Realsense D455 a great choice since it is a newer generation to the D435 sensor but which have an increased range. One of the mounted sensor can be seen in Figure 3.2.

Figure 3.2: Realsense D455 mounted on the AGV.

The hardware synchronization of the shutters in the D400 series can be done by either utilizing one camera as master or supplying an external signal via an extra cable [62]. For this thesis one of the cameras were used as a master, supplying the shutter signal to the other two cameras.

The utilization of three cameras allows for extended flexibility in the form of camera placement. A natural way for a three camera setup would be to place one camera in the driving direction, similar to Manhed [12], and then place the other cameras to extend the total FoV as were theorized to be one of the main issues. This setup will then ensure object avoidance in the driving direction as these are assumed to be detected by the forward camera. The other cameras can then be placed either in a stacked or panoramic configuration. In addition, as each camera have an oblong FoV, they can either be placed in a portrait or landscape configuration.



(a) Maximum distance **(b)** No gaps in field of view **(c)** Hybrid solution

Figure 3.3: Camera FoV alternatives solutions.

The chosen configuration uses all cameras in landscape orientation, one facing forward and the other two translated and rotated 65° outwards in the right and left direction respectively, this results in an FoV of 216°. An illustration of the camera placements can be seen in Figure 3.3c. Notice that the FoVs are not overlapping for any camera in Figure 3.3a and 3.3b. This is done to ensure maximum distance and maximum coverage respectively. The chosen solution is a hybrid of the two first in order to ensure as wide FoV as possible but still have a continuous field of view.

A Hokuyo UAM-05LP 2D-LiDAR is mounted on top of the system module. This LiDAR was chosen as it is the same model as the one used naively by the AGV and thus is shown to satisfy the criteria for navigating an AGV in an industrial environment. The sensor scans a horizontal plane 431mm over the ground. The sensor has a utilized FoV of 216° where the center of the view is directed in the forward driving direction of the AGV i.e the effective FoV. The sensor measure distance by the time of flight principle and have a specified angular resolution of 0.125° [63].

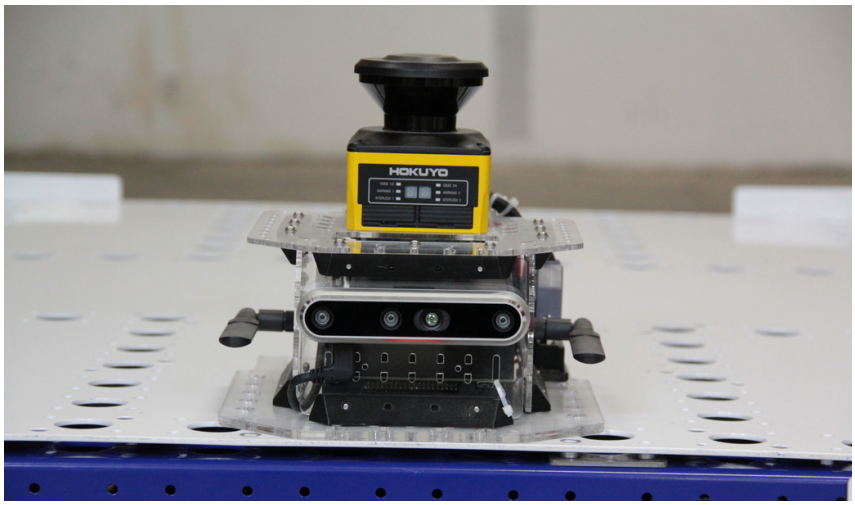


Figure 3.4: LiDAR provided by FlexQube.

The complete implementation of the systems can be seen in Figure 3.5. The implementation consists of a test platform in the form of an AGV, a system module, and external sensors and mounts. The system module contains one 2D-LiDAR, three 3D-cameras, and an integrated computer. The computer is an Nvidia Jetson AGX and with an internal extended storage of 2Tb used to record sensor data in real-time.

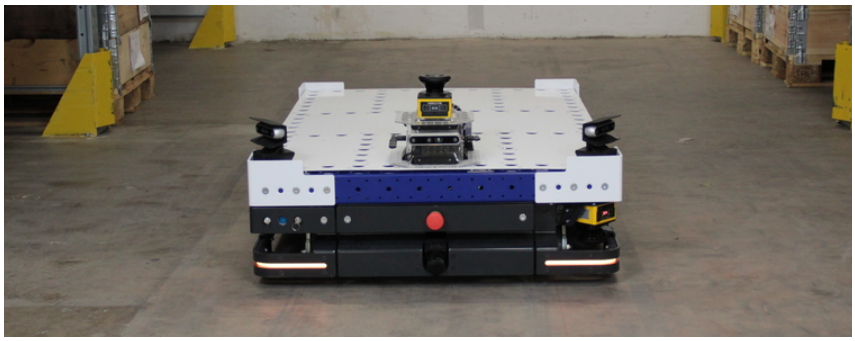


Figure 3.5: Complete test platform with one LiDAR and three 3D-cameras in a panoramic setup.

3.2 Localization

To evaluate the performance of the developed system, localization will be achieved in two ways, one is by SLAM, the other is by pure localization. Pure localization means that the map will not be updated. Instead, the map previously generated by SLAM will be used. The same algorithm for localization will be used in both cases.

To achieve a comparable result for both the 3D-camera system and 2D-LiDAR system, multiple SLAM frameworks which are capable of handle both of these data types was investigated. According to Pålsson and Smedberg [64] there are three 2D-SLAM algorithms which are the most popular, GMapping [65], Hector SLAM [66] and Google Cartographer [67]. The authors Filipenko and Afanasyev [68] also shows that Cartographer is the 2D-SLAM algorithm that have the lowest absolute error of the compared algorithms in there study, among these were the previously three most popular algorithms mentioned. This result is also shown by Pålsson and Smedberg [64]. The authors Milijas, Markovic, Ivanovic, *et al.* [69] shows that Cartographer is a good choice of algorithm, especially when there may be lack of features. Cartographer is also not a strictly a 2D algorithm and can support 3D-SLAM. According to Nuchter, Bleier, Schauer, *et al.* [70] is the 3D-SLAM of Cartographer a continuation of its 2D algorithm but with some minor tweaks for efficiency. This is good for the evaluation of the 2D-LiDAR and 3D-cameras systems as this should in theory, neglect the performance difference of running separate independent algorithms. Thus Cartographer [71] was chosen as the SLAM method used to test and verify the proposed panoramic 3D-camera setup.

Even if Cartographer is able to localize in 3D, to be comparable to the 2D-LiDAR system, only the equivalent 2D-localization of the 3D-camera SLAM is evaluated. Cartographer is configured by several codependent tuning variables. These are tuned empirically on multiple test data sets and independently for both systems. Cartographer achieves mapping in two stages, first by creating a submap describing a local environment, then aligning multiple submaps to create a global map of the environment [67]. Each submap consists of a number of data-samples, which the authors refer to as scans and is in reference to the scans acquired by a scanning LiDAR [67], but can also be used to describe each point cloud produced by the camera system. The number of scans required for a submap is one

of the parameters that is tuned. When this limit of scans is reached the submap is finalized and aligned to a larger global map and a new submap is initiated. To align the multiple scans of a submap, intra-map constraints are produced. Similarly, to align the submaps in the global map, inter-maps constraints between the submaps are generated. Figure 3.6 illustrated the multiple, densely-packed, intra-map constrains as well as some inter-maps constrains which are more sparse. Inter-maps constrains are divided into constraints of the same trajectory, shown as the yellow lines in Figure 3.6 and loop-closure constraints, shown as turquoise lines. A loop-clousure is an event where the SLAM algorithm enters a submap which it recognises as somewhere it have already mapped. If the map have some misalignment of these two matching submaps, a stronger constraints can be applied as to give a higher reward for an alignment [67].

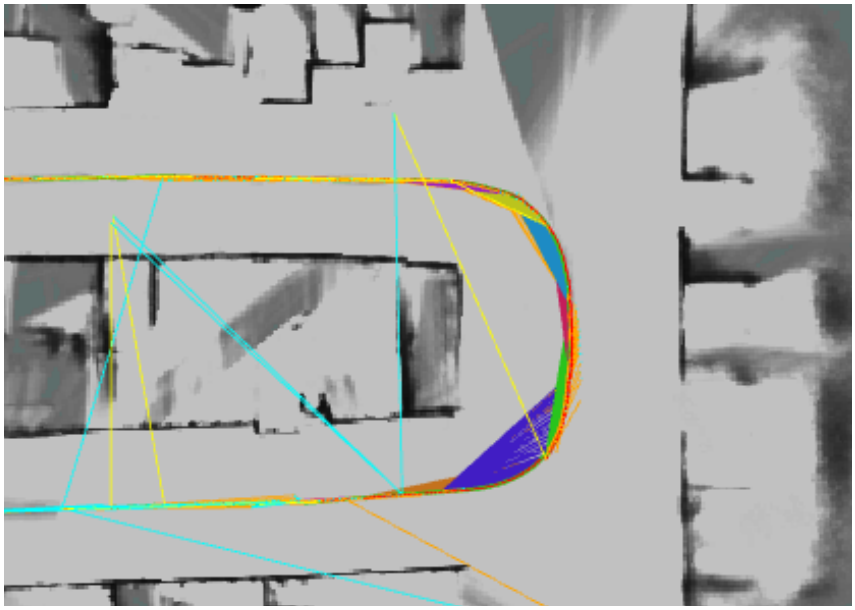


Figure 3.6: Top-down view of the map and constraints generation by Cartographer SLAM. All constraints are connecting to the current trajectory. Intra-map constraint are shown as densely packed lines of multiple colors. Inter-maps constraints are shown as yellow for constraints of the same trajectory and turquoise for loop-closure constraints, i.e. constraints between a submap from a different trajectory, to the current.

The number of scans to be used for each submap shall be enough to achieve a distinct enough submap for global alignment, but also few enough such that the drift, due to misalignment, is acceptable [72].

3.3 System architecture and data filtering

This section will explain the structure of the localization systems implemented, that is, the proposed 3D-camera system and the 2D-LiDAR system as well as the filters utilized by the camera system. To accurately quantify the performance of the systems, an additional stationary high precision localization system will be used as a ground truth reference. This system will hereinafter be referred to as the ground truth system and further explained in Chapter 4.

3.3.1 Data types

The main data types passed trough the localization systems are: point clouds $\mathbf{P}$, points u, position state vectors $\mathbf{x}$ and transformation matrices $\mathbf{T}$. All point clouds handled in this project are assumed to be unordered and to have coordinates. Throughout the systems, the point may also receive a surface normal estimation. The position state vector describes the position of an object. An identifier of the object i_{obj}, as well as the coordinate frame i_{frame} it relates to, is also included. A transformation matrix contains the rotation matrix R and translation vector t required to align one coordinate frame with another. The data types are structured as:

$$\begin{aligned} \mathbf{P} &= \{u_1, \ \ldots, \ u_{n_{max}}\} \\ u &= \{x, \ n, \ c\} \\ \mathbf{x} &= \{x, i_{obj}, i_{frame}\} \\ \mathbf{T} &= [R \mid t \ ] \end{aligned} \tag{3.1}$$

3.3.2 Systems architecture

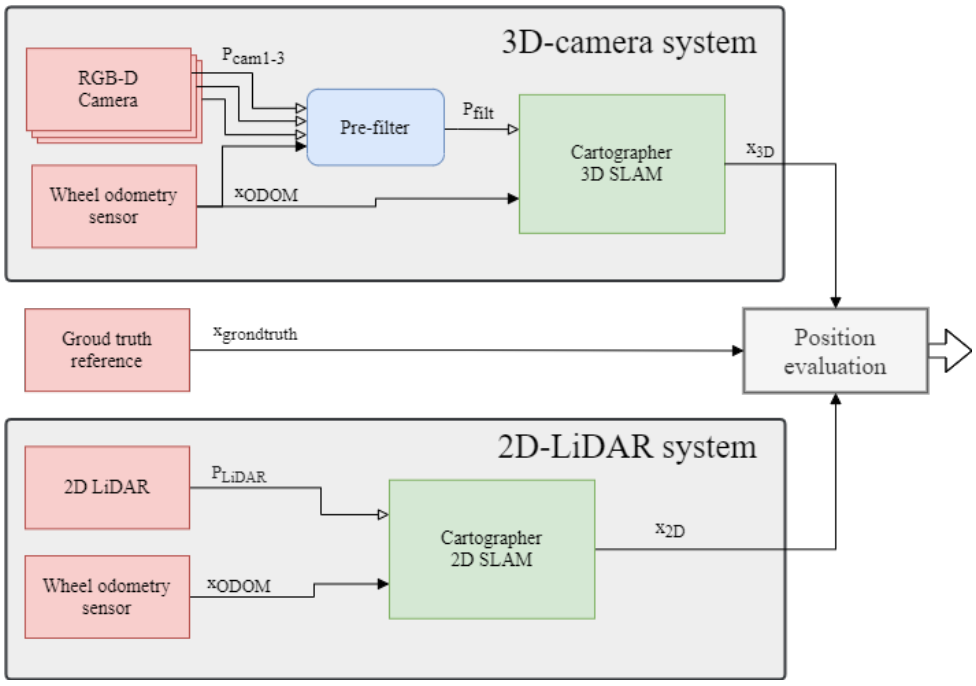


Figure 3.7: Systems overview. $\mathbf{P}$ denotes point cloud data and x denotes a state vector. The SLAM localization estimations are compiled in a Positing evaluation stage where the deviation to a ground truth reference is evaluated. If a ground truth reference is not available a relative comparison is performed.

Both filtering for the camera and LiDAR systems are sampled in parallel, and when available, combined with a ground truth source used for validation. This is done to allow for testing and validation of both systems when applied to the same environment. That

is, when tested, the input odometry data will be the same for both systems. However, to manage the increased load of parallel SLAM algorithms, the system will be tested separately with data obtained from the same sampling. For tuning, both systems are run separately on the sampled data. The ground truth system is running on an external computer and will produce a validation trajectory after the test is done.

3.3.3 Data filtering

Data filtering is a major part required to achieve good results from the captured point cloud data. The filters must achieve adequate performance in both data optimization and throughput. No additional filtering will be done one the 2D-LiDAR system, the following filters are only applied to the camera localization system.

Data filtering of the camera system is implemented in two main stages, pre-filtering and filtering done by the SLAM algorithm. This with the main goal of obtaining the optimal position estimation. The pre-filter stage consists of several internal subsequent filters. The overall task of the pre-filter is to merge, filter and optimize the multiple point clouds obtained by the cameras into one point cloud.

The filters applied by the SLAM algorithm are tasked with time aligning input data, downsample, and removing outliers. The SLAM algorithm will finally produce a trajectory estimation as well as a map of the environment. The position data is then passed back to the pre-filter stage. The usage of this data will be further explained in the plane filter Section 2.2.6.

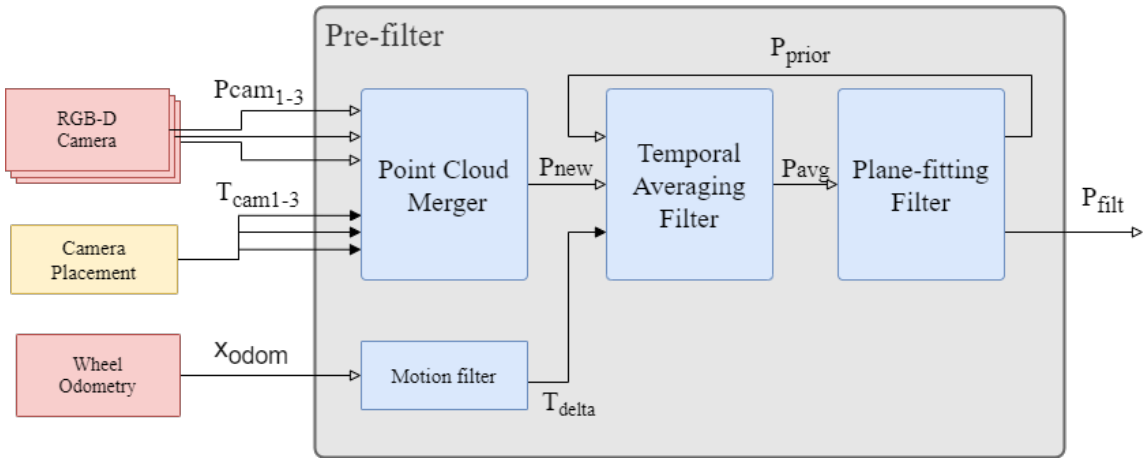


Figure 3.8: Camera data $\mathbf{P}_{cam}$ from each camera is first merged into one cloud $\mathbf{P}_{mrgd}$ then through a temporal averaging filter and finally a plane-fitting filter. $\mathbf{P}_{filt}$ is the resulting filtered data and $\mathbf{P}_{prior}$ are the same points as in $\mathbf{P}_{filt}$ but with the addition of points corresponding to the ground. The ground points are separated as these are not interesting for the SLAM algorithm.

The Camera-placement block illustrates the static camera placement parameters passed to the Pre-filter. These are parameters defining the position of each camera related to the desired unified coordinate frame. In this case, the unified coordinate frame is that of the

front-facing camera. The camera placement parameters must be of high accuracy such that the merging of all point clouds is correct.

A Motion filter is implemented with the goal of obtaining the current position of the unified frame. The last estimation is also saved. Both the current and last pose is then used to calculate the relative motion between the two iterations, $\mathbf{T}_{delta}$. This transform is then passed to the temporal filter to be used for aligning the prior point cloud with the new, as explained in Section 2.2.5.

Before the temporal filter, each point cloud passed from the cameras are merged in the Point cloud merger block. In addition to some per-cloud filtering, the merger also merge the points that are overlapping. That is, the point that have a close enough proximity where a merging of both points will not negatively effect the geometry of the cloud.

Finally, after the merging of all point clouds and temporal filtering, the unified point cloud is passed to a Plane fitting filter. This is a complex filter that contains multiple internal stages and with the goal of fitting points corresponding to planar surfaces onto perfect planes. All points which are successfully fitted to planes, in combination with the filtered but non-fitted points, are then passed to the SLAM stage.

3.3.3.1 Camera intrinsic filters

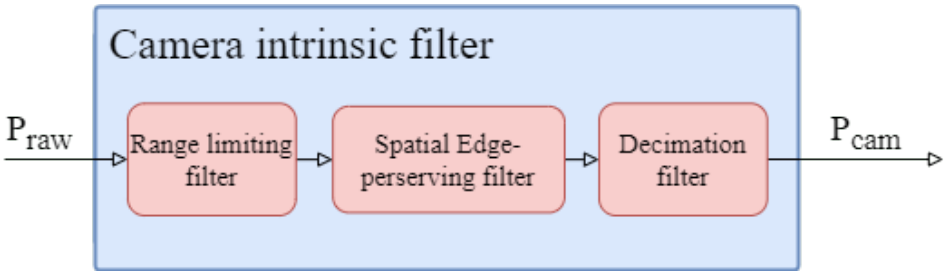


Figure 3.9: Camera intrinsic filters

The Realsense D455 camera also have several intrinsic filters. A subset of these was applied in this implementation. These are a Range limiting filter, Decimation filter and a Spatial edge-preserving filter.

The filters are tuned with the main goal of downsampling raw data and to do an initial outlier removal. i.e points that are erroneous detections or otherwise of no interest. Outliers are mainly removed by the Range and Spatial filter whereas the Decimation filter is added to uniformly reduce the output data. The Range filter was tuned to discard any point beyond the recommended range as specified by the camera manufacturer while the spatial filter was tuned empirically to smooth the data without losing the sharpness of edges and corners. The Decimation filter uses a square median averaging kernel of size k_{dec} and stride s_{dec} and was tuned to give a desired output cloud density for subsequent filters. The resulting filter parameters were:

$$
\begin{aligned}
d_{range} &\leq 8\ [m] \\
M_{spatial} &= 3 \\
\alpha_{spatial} &= 0.5 \\
\delta_{spatial} &= 20 \\
k_{dec} &= 3 \\
s_{dec} &= 3
\end{aligned}
$$

3.3.3.2 Point cloud merger

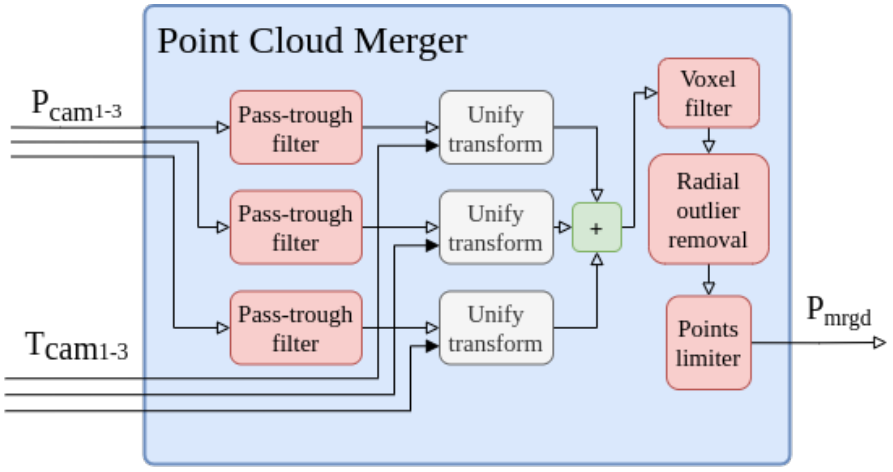


Figure 3.10: Multi point cloud merging. Point clouds are merged based on the given Transformation matrix for each camera.

The Point cloud merger is given several time-synced point clouds. It is assumed that frames from each cloud are captured very close in time as such that they describe the environment with the same position of the unified frame. To ensure good frame timing, hardware synchronization is applied, as described in Section 3.1. To merge all streams into the defined unified frame, a transformation matrix T_{cam} is given for each point cloud. This is a static transformation that must be of high accuracy as any deviation will have a constant effect on all acquired data.

Before the transformation, a Pass-trough-filter was applied to each cloud. This was done before the transformation so that the parameters of each filter can be set to optimally filter the data of the specific camera in its own frame. A pass-through filter of band-pass type was used. The depth range, corresponding to the z-limits in the camera frame, was tuned with a lower bound equal to the camera specification and an upper bound tuned to achieve maximal depth range but with acceptable depth precision for the application. A symmetric limit in x, corresponding to width in the camera frame, was selected. The selection is to suffice a range enough to fit all valid x-coordinate of a point corresponding to the cameras FoV and the defined depth limit, obtained as:

$$x_{max} = z_{max} \cdot tan\left(\frac{FoV_{(x)}}{2}\right) \tag{3.2}$$

The y-limits, corresponding to the cameras height limits, were selected with a lower bound such that ground points could be detected and with some margin as to account for small downhills. The upper bound was used to tune the total volume evaluated, to give a desired average in a total number of points. The resulting filter limit in meters was:

$$\begin{aligned} -5.6 \leq x &\leq 5.6 \\ -1.0 \leq y &\leq 3.0 \\ 0.5 \leq z &\leq 6.0 \end{aligned}$$

When all streams are transformed into the unified frame a voxel filter is applied. This is done both to reduce the total number of points but also to remove point cloud overlap. As these overlapping points will, if any, detect the same geometry, will it be because their position is close. This will cause many of these overlapping points to coexist in the same voxel and thus be merged, as mentioned in Section 2.2.3. The voxel size was tuned to achieve a desired average point cloud density sufficient for subsequent filter applications in relation to the desired execution speed. This size is also used for any subsequent voxel filter to keep a consistent point cloud density. The voxel size in meters was selected as:

$$r_{voxel} = 0.05m$$

Next, a Radial outlier removal-filter is applied. The filter evaluates whether an evaluated point of a point cloud has a sufficient number of neighbors within a specified radius. If the number of neighbors is too few, does it mean that the point is not part of any local geometry and is thus discarded. The specified number of neighbors and radius range was selected as:

$$\begin{aligned} n_{neigh} &\geq 10 \\ r &\leq 0.1m \end{aligned}$$

Finally, the number of remaining points is counted. If the number is over a defined limit, only the closest points are kept. This is based on the assumption of higher data accuracy at a closer range as mentioned in Section 2.1.2.1. The limit on the number of points highly affects the worst-case execution time, per point cloud filter iteration as it directly correlates with the highest number of combined per-point calculations done. To achieved a sufficient throughput, the maximal number of points was selected as:

$$n_{points} \leq 15000$$

3.3.3.3 Temporal averaging filter

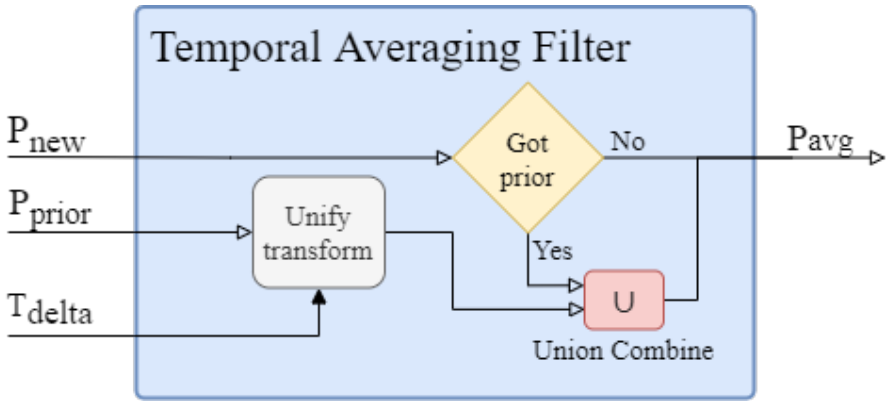


Figure 3.11: The temporal filter when give a new and prior point cloud, will optimize the new point cloud based on knowledge of the prior. Due to potential misalignment of the new and prior point cloud, an estimated relative motion transform $\mathbf{T}_{delta}$ is also given.

The Temporal averaging filter will obtain the most recent merged point cloud P_{mrgd} as well as the last filtered point cloud P_{prior}. It will then attempt to merge points from the last cloud into the new cloud. However, as described in Section 2.2.5, any motion of the sensor in the time span of capturing the last cloud and the new will cause misalignment of the two clouds. To prevent this effect, the filter will attempt to estimate and mitigate this motion. This is done by obtaining the position of the unified frame at the time of the last frame acquisition as well as the new. The position information is also given for both sources as transformation matrices. These matrices show the transformation from one, common in time and position, reference frame. The reference frame is that of the world frame at startup time t_0. As both given transformation matrices have the same reference frame, the difference in transformation will be the same as the relative transformation between the two given point cloud frames. This relative transformation matrix called T_{delta} is then used to transform the prior cloud to align with the new point cloud. This will cause the new and prior cloud to overlap at multiple points. The clouds are then passed through a Union-combine filter to remove points from the new cloud which are close to prior points. The motivation for this is that the prior points are already filtered and known to be relevant. However, any prior points that are not close to a new point, are omitted as it is not sure that they are still valid. The resulting output of the filter are: new points that are not close to prior points and, prior points close to new points as illustrated in Figure 3.12.

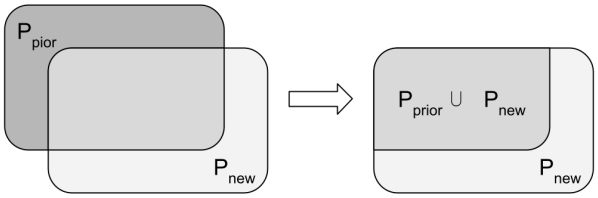


Figure 3.12: Resulting set of points from union of P_{prior} and P_{new}.

The intended effect of the temporal averaging filter is to preserve the knowledge of the already filtered and known to be valid prior geometries, i.e. planes. However, as one detected geometry may be reused for subsequent frames, and as new data of the same geometry is added, this estimation of a geometry can be improved instead of staying the same. To make this possible, the tolerance for merging new and prior points in the temporal filter shall be lower than that the overall criteria for inliers of the geometry. This means that only points fitting well to the geometry will be kept by the temporal filter and, as more points are fitted to the geometry, the filtered points in combination with new inliers will move the estimation into a more optimal average.

3.3.3.4 Plane-fitting filter

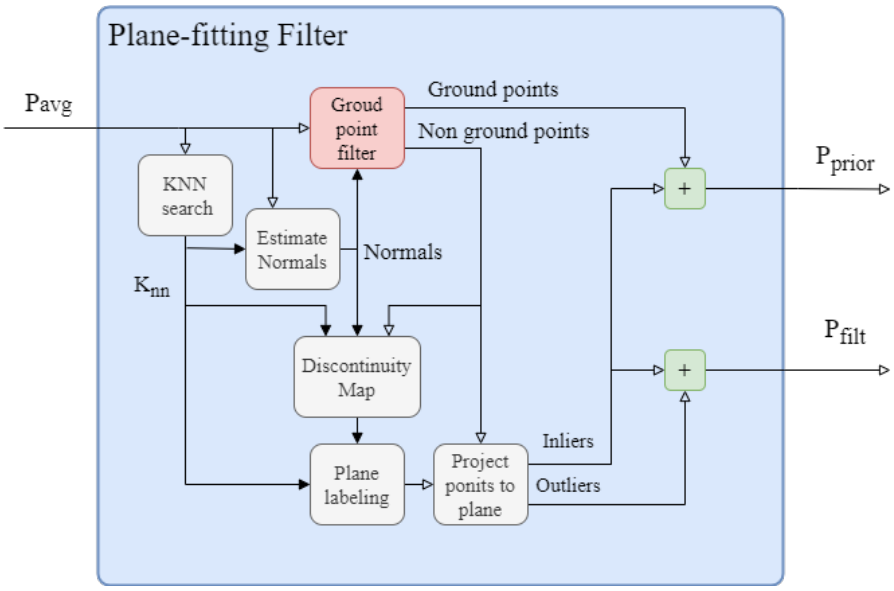


Figure 3.13: Points from the Temporal averaging filter $\mathbf{P}_{avg}$ that correspond to planar surfaces are fitted to planes. All fitted points in combination with points corresponding to the ground plane are returned as $\mathbf{P}_{prior}$. $\mathbf{P}_{filt}$ are all filtered points not corresponding to the ground.

The main benefit of applying a plane-fitting filter is that it can reduce the ambiguity of noisy point cloud data. As the measurement noise of the 3D-camera increases exponentially as the distance is increased, by applying the plane fitting filter this ambiguity can be reduced. The problem is converted into a binary decision problem where the points are either fitted to a plane or not. If the points are fitted to a plane this will be of a perfect plane without any ambiguities. However, as it may not be sufficient to only rely on the detection of planar geometries, some non-planar points may also be included.

4
Evaluation methodology

To adhere to the research questions as stated in Section 1.2.1, all evaluation methods must be developed to give both the camera system and the LiDAR system an equal opportunity. That is, the panoramic 3D-camera system and 2D-LiDAR system shall be tested on the same computer and AGV, it should also be in the same environments. Therefore, by running both systems in parallel, as proposed in Section 3.3.2, all of these criteria can be met. The implementation of mounting both sensor systems rigidly to the same platform ensures that both systems can utilize the same auxiliary data acquired in the form of wheel odometry. This ensures that any error propagated from this data is affecting both systems equally. The speed and trajectory of the AGV are also equally perceived by both systems due to the rigid mounting.

Both systems will be tested on data from the same environments, specifically a warehouse and a lab-room of approximately 6x8m in which a ground truth system is installed. The warehouse will not have a ground truth system and therefore the results will be evaluated relatively between the two systems. The warehouse will also be in use throughout the tests and shall thus be considered as an authentic industrial environment. The lab-room is instead arranged to emulate a small workstation.

Due to software limitations on the integrated computer of the test platform, the data will be evaluated on an external computer.

4.1 Ground truth evaluation

The developed 3D-camera system will be tested in parallel with the reference 2D-LiDAR system and compared to a highly accurate positioning system. The highly accurate positioning system is a Qualisys motion capture system[1] consisting of eight cameras, as can be seen in Figure 4.1. These are calibrated to achieved a self-diagnosed precision of approximately 2mm per camera. The trajectories are thus not perfect and each position measured by it contains a residual. The residual is the potential measurement error to the true position and can be seen as the radius of which the point can actually be within. The residual is based on the self diagnosed precision by each camera contributing to the measurement [73]. However, since the precision is at least one order of magnitude better than the expected precision of the systems compared, it is regarded as a ground truth for test in the lab-room.

[1] https://www.qualisys.com

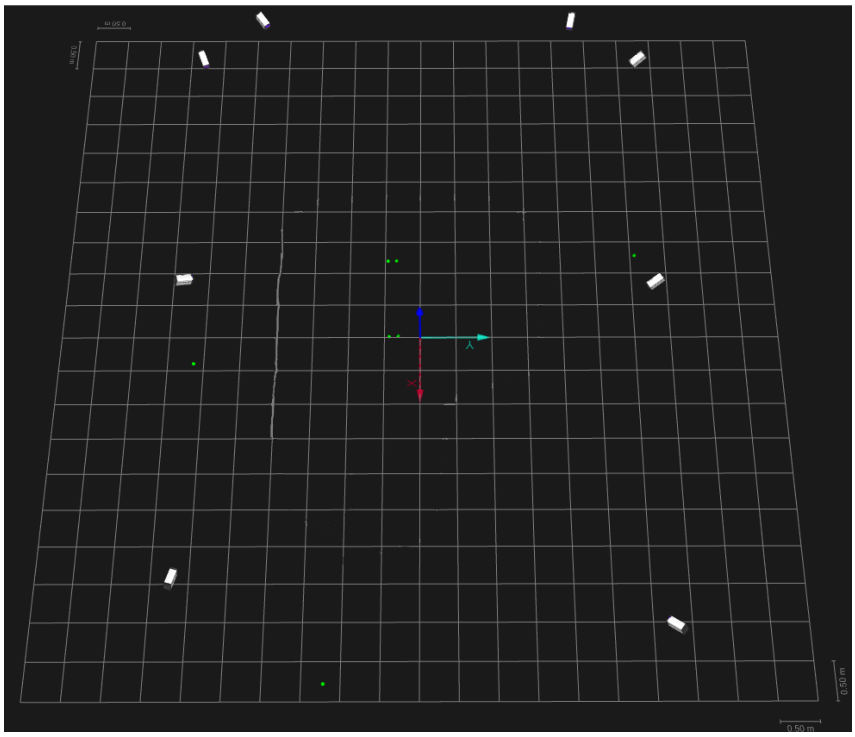


Figure 4.1: Illustration of the eight Qualisys Cameras placed in the room and with the three outermost static markers (green) placed on the walls and four static markers placed on a dummy-wall in the center.

The data from the Ground truth system, 2D-LiDAR, and 3D-Cameras will not use the same origin and orientation. Thus alignment is needed in order to obtain data that is comparable between these systems. ICP will be used to align a LiDAR trajectory to the ground truth trajectory, in order to align the map as accurately as possible to the room. The match will then be verified by manually placed reference markers which are then detected by the ground truth system, seen as three points in Figure 4.1, the bottom one, the left-most one, and the right-most one. These are used to give a known location of room walls. The aligned map of the room will then be stored and used as a reference for the rest of the obtained data. In the subsequent data acquisitions the maps are matched to the reference map with ICP. It could be argued that matching maps is better than continuing to match trajectories of the 2D-LiDAR and 3D-Cameras to the ground truth trajectory. This since the comparison desired is of how accurate the trajectory is, and therefore shall the trajectory not be matched to prevent overfitting.

The systems are not synced, therefor the localization estimations will not necessarily produce positions at comparable times and places. To obtain a comparable position error evaluation, a spline between the points of the ground truth system is interpolated. Then the position error for each evaluated system is computed as the minimal distance for each point to the interpolated spline. Note that, special care needs to be taken in a situation where loops in the path occur, e.g. circular paths, as a distance might be calculated to the spline corresponding to the wrong point in time if it is closer. As a result, only a smaller region around the last evaluated point of the ground truth path is interpolated at a time, any non interpolated points are ignored.

4.2 Evaluation of optimization filters

The optimization filters are the implemented filters that, in addition to filter data, also augment the data to optimize the localization result, i.e the developed temporal averaging filter and plane-fitting filter. Their importance is evaluated by comparing the resulting performance towards a similar test system implementation where these filters are omitted. Note that, as previously mentioned, only the proposed panoramic-camera system have these optimization filters applied. Therefore, the systems for this test are of two versions of the proposed panoramic camera system, implemented as in one way, the complete proposed system, and in the other way as a simplified version where the temporal averaging filter and plane fitting filter are omitted. These versions are hereinafter referred to as *Optimized* and *Unoptimized.*

Both the optimized and unoptimized system versions are tested on the same environment data obtained from the Lab-room. Therefore ground-truth reference data is also available. Localization estimations, i.e path estimation, for both systems are obtained via the Cartographer SLAM algorithm and with the same tuning parameters. The performance is evaluated based on the localization error produced by each system, and in comparison to the ground truth data.

4.3 Evaluation criteria

To achieve an accurate evaluation of both system's estimation errors in reference to the ground truth system, three evaluation criteria are defined. The parameters investigated are *Accuracy*, *Trueness*, and *Repeatability*, these can be seen in Figure 4.2 and they are defined as:

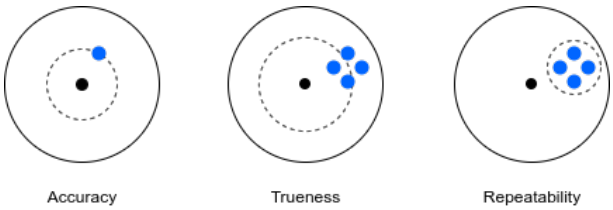


Figure 4.2: Illustration of Accuracy, Trueness, and Repeatability.

Accuracy is hereby defined as the absolute distance error of a single localization estimation, to the ground truth reference.

Trueness is hereby defined as the distance error from the mean value of multiple samples of the same data, to the ground truth reference.

Repeatability is hereby defined as the variation of multiple samples, specifically the difference of the best and worst estimation.

4.4 Reference map

This section will present how a reference map is obtained in order to be able to compare obtained data against the ground truth system. The reference map is generated by matching one LiDAR trajectory with ICP to the same trajectory generated by the ground truth. The resulting map alignment is then verified by reference markers can be seen in Figure 4.3.

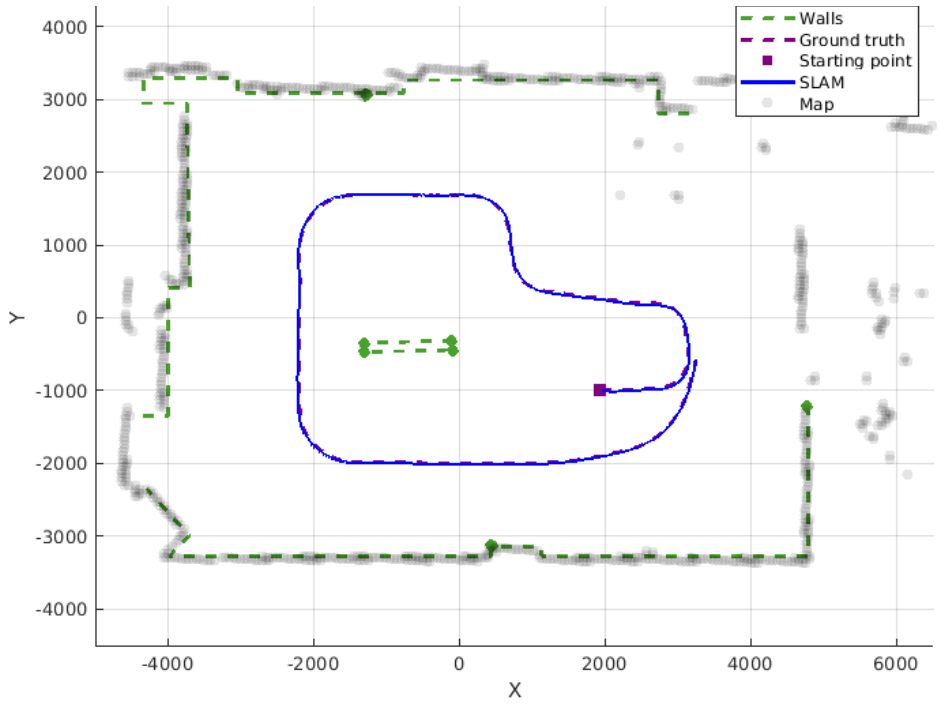


Figure 4.3: Reference map aligned to room, three green dots on the walls are markers from the ground truth system, walls are drawn from laser measurements and data from the ground truth system. AGV starts at (1936, −990) and runs a left-hand lap around the room.

The walls of Figure 4.3 are manually placed by inspecting multiple maps generated by LiDAR. Note that these are only intended as a visual aid and will not affect any results. On the walls are three static markers used to verify that the maps from the LiDAR are correct. Additionally, as the walls are drawn based on LiDAR data which are in 2D, the following maps generated by the 3D-camera system are expected to not perfectly match them.

This method of alignment is not perfect due to the additive error when using multiple ICP matchings, i.e path to path matching and map to map matching, and also the assumption that the LiDAR trajectory is perfectly aligned with the ground truth. An exact method utilizing precision timestamps of all position data, which would improve the matching, as well as a know initial pose would be preferable. However, this data is not available for either the tested systems nor the ground truth system. The only other identified alternative would be to hand align every map and trajectory, but this is considered as an alternative with the potential of introducing larger errors than the chosen solution.

5
Results

This chapter will present all results obtained from both the developed panoramic 3D-system and the reference 2D-LiDAR system and in both described test environments. The first part of the data presented will be from the panoramic system and where the effects of optimization filters are evaluated. The second part is of test performed on two data sets sampled from the lab-room environment. That is, from two different test runs, one shorter and one longer, hereinafter called Case 1 and Case 2 respectively. The performance of the systems and on the test cases will be quantified by the evaluation criteria as defined in Section 4.3. The reference map was generated from matching trajectories with ICP as described in Section 4.4. The third and last section presents data obtained in the industrial environment and will present a qualitative evaluation of the systems as no ground truth reference was available.

The AGV was operating autonomously for the sampling of the lab-room environment. This was to simulate the movements of a real-world autonomous application in terms of speed, positioning, and repeatability and was achieved by the pre-existing navigation system of the AGV. For the warehouse, the AGV was maneuvered manually by an operator as the autonomous system could not be configured for the environment at the time of sampling.

5.1 Effects of optimization filters

In this section, data is presented illustrating the performance of running the SLAM algorithm with and without the optimization filters, i.e. temporal averaging filter and plane-fitting filter. The data is generated in the lab-room environments Case 1. The mean distance error to the ground truth reference for 3D-Camera SLAM with and without optimization filter of this test was:

$$\begin{aligned} \text{Optimized: } & 29.9mm \\ \text{Unoptimized: } & 126.9mm \end{aligned}$$

Figure 5.1 illustrates the ICP aligned maps produced by the optimized and unoptimized filter scheme, their respective path estimation, and the ground truth reference path. Note the large deviation in the generated maps in the lower right region indicating a poor map generated by the unoptimized filter scheme.

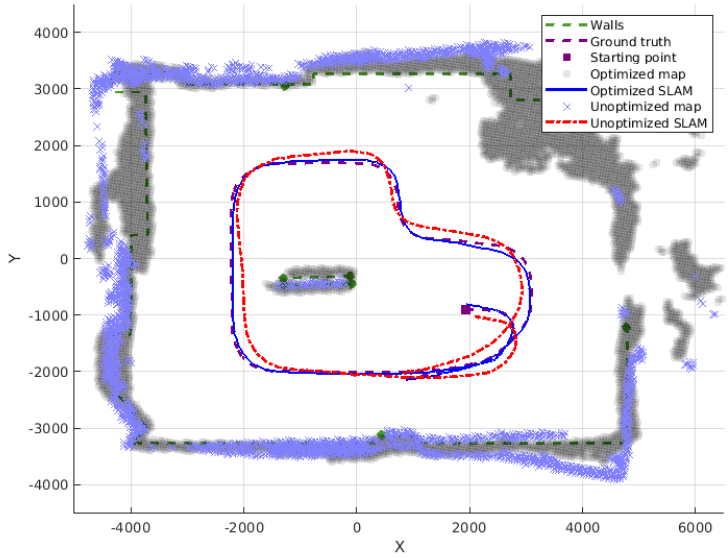


Figure 5.1: Case 1: SLAM with and without the optimization filters. AGV starts at (1930, −905) and runs a right-hand lap around the room.

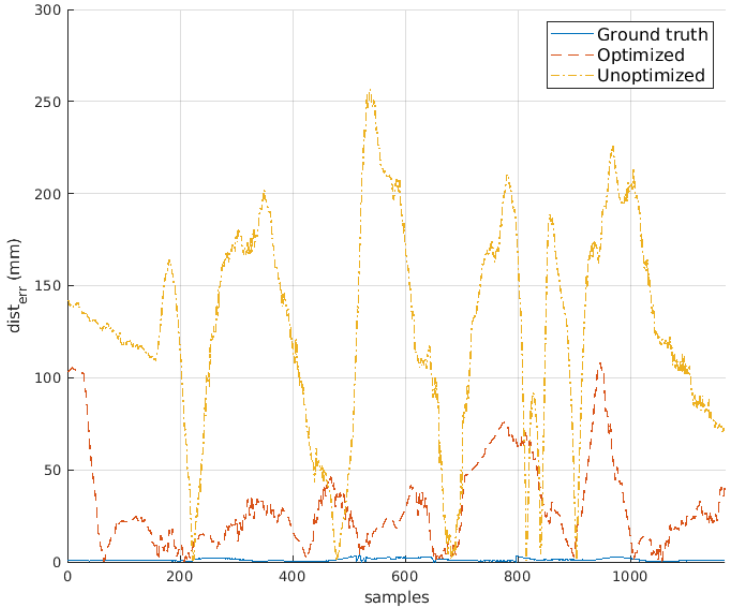


Figure 5.2: Case 1: SLAM error with and without the optimization filters. The blue line shows the residual of the ground truth system.

Analyzing the data in Figure 5.1 and Figure 5.2 shows that without filtering the algorithm is not able to generate an accurate trajectory, neither is it able to create an accurate map of the environment. This means that it would neither be able to localize in the map later on. Thus further on in the results only filtered data will be used for evaluation.

5.2 Quantitative systems evaluation

The data presented here is obtained from the 2D-LiDAR system and the panoramic 3D-camera system. The plots contain a map produced by the corresponding SLAM algorithms. This map has been fitted against the reference map by ICP. Each figure contains the trajectory produced by the SLAM algorithm and the localization algorithm for that sensor and case. Two different cases are presented from the lab-room. As previously mentioned SLAM is running both localization and mapping, while localization only runs localization on an existing map. The raw input data to both SLAM and localization will be the same. The results presented will be used to analyze the systems based on the criteria presented in Section 4.3.

5.2.1 2D-LiDAR

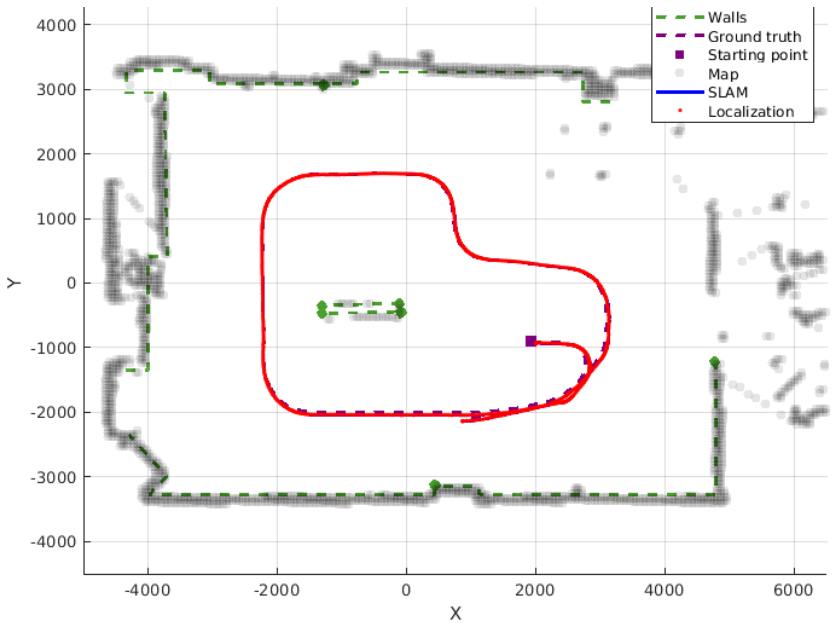


Figure 5.3: Case 1: LiDAR map matched to reference map, AGV starts at (1930, −905) and runs a right-hand lap around the room.

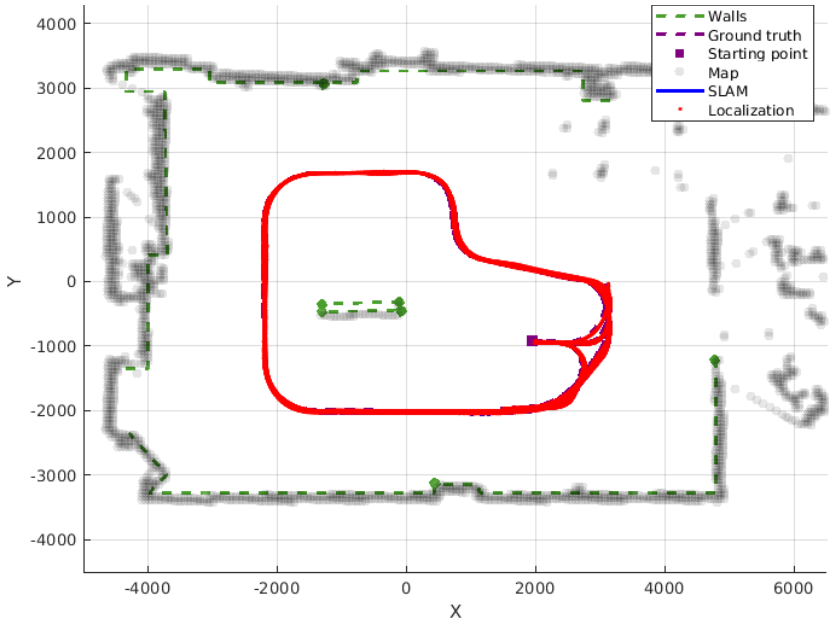


Figure 5.4: Case 2: LiDAR map matched to reference map, AGV starts at (1942, −923) and runs two left-hand laps around the room, three-point turn, and another two laps right-hand laps around the room. Lastly is a three-point turn and a last left-hand lap.

5.2.2 3D-Camera

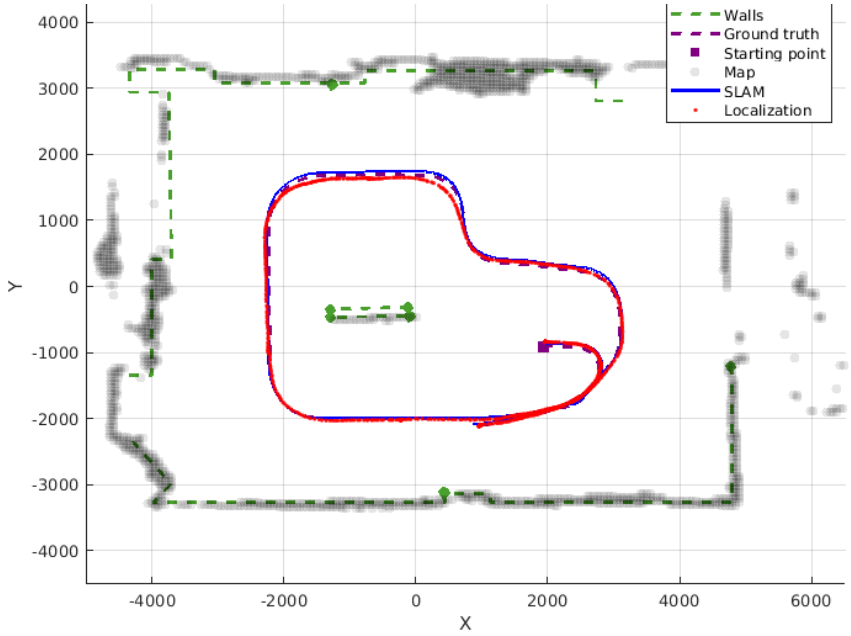


Figure 5.5: Case 1: 3D-Cameras map matched to reference map, AGV starts at (1930, −905) and runs a right hand lap around the room.

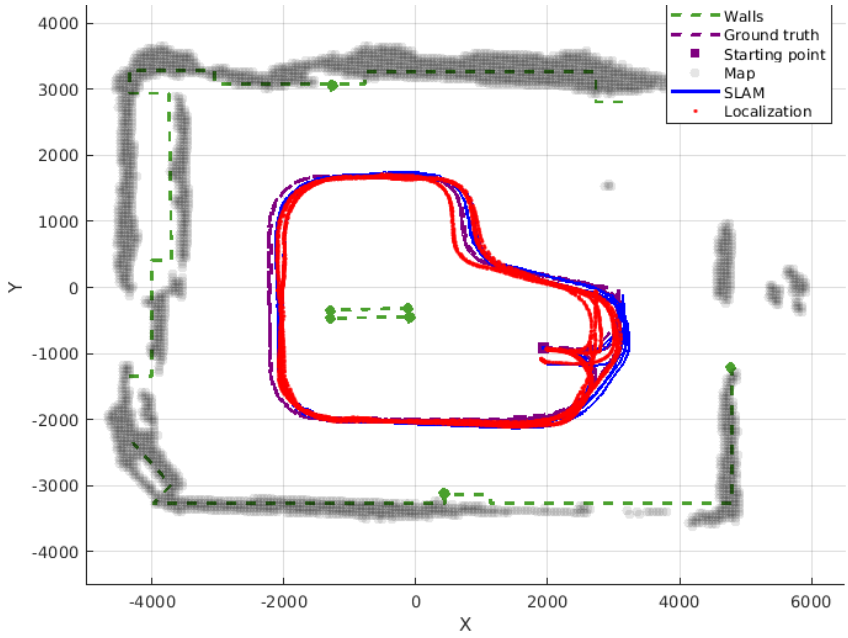


Figure 5.6: Case 2: 3D-Cameras map matched to reference map, AGV starts at (1942, −923) and runs two left-hand laps around the room, three-point turn, and another two laps right-hand laps around the room. Lastly is a three-point turn and a last left-hand lap.

5.2.3 Accuracy

Comparing and analyzing the accuracy is done on the data presented in previous sections to the ground truth system. Figure 5.7 and 5.8 contains the residual and errors for Case 1 and 2 respectively. The error is how far each trajectory is from the ground truths trajectory. The mean of each residual and error in Figure 5.7 and 5.8 is found in Table 5.1.

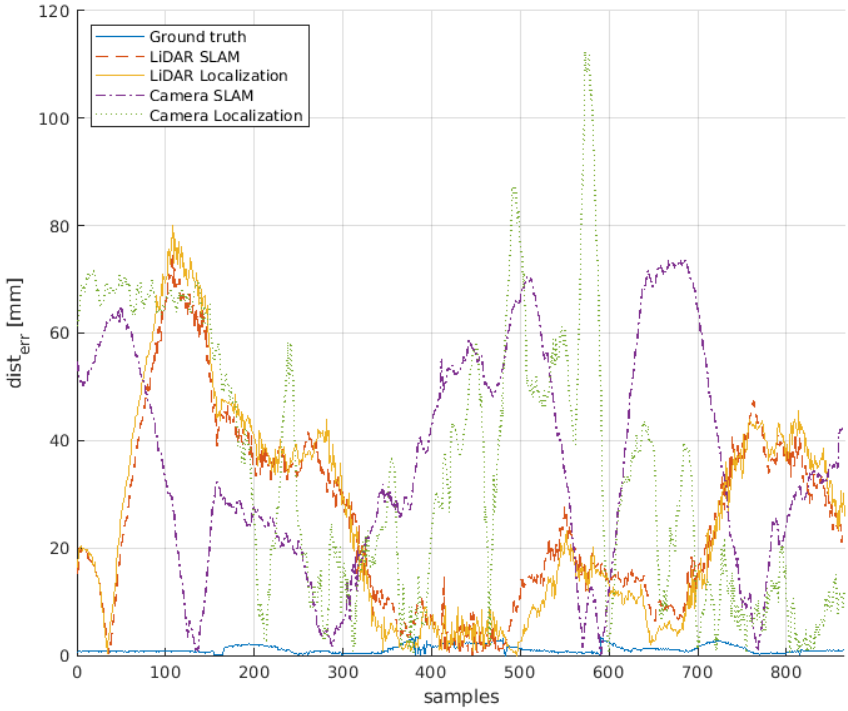


Figure 5.7: Case 1: Trajectories from Figure 5.3 and Figure 5.5 compared to the ground truth.

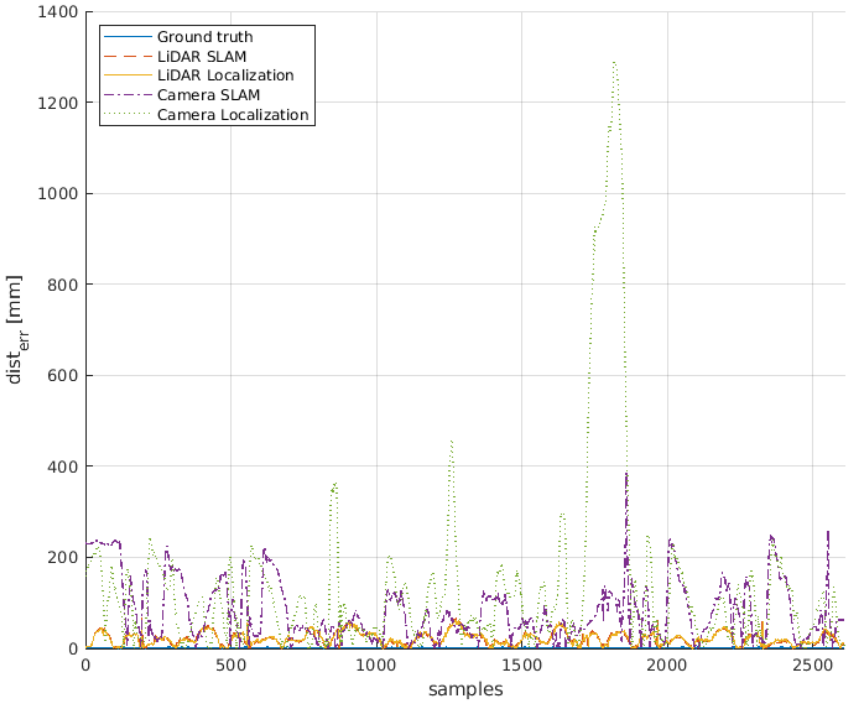


Figure 5.8: Case 2: Trajectories from Figure 5.4 and Figure 5.6 compared to the ground truth.

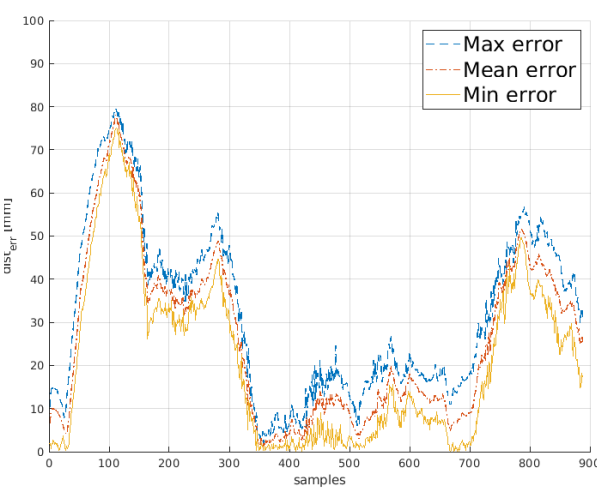
Table 5.1: Mean residuals and mean errors from case 1 and case 2. All measurements are in mm.

Method:	Ground truth	LiDAR SLAM	LiDAR Localization	Camera SLAM	Camera Localization
Case 1:	1.1	25.1	25.4	36.0	35.2
Case 2:	1.2	21.3	21.4	82.8	141.3

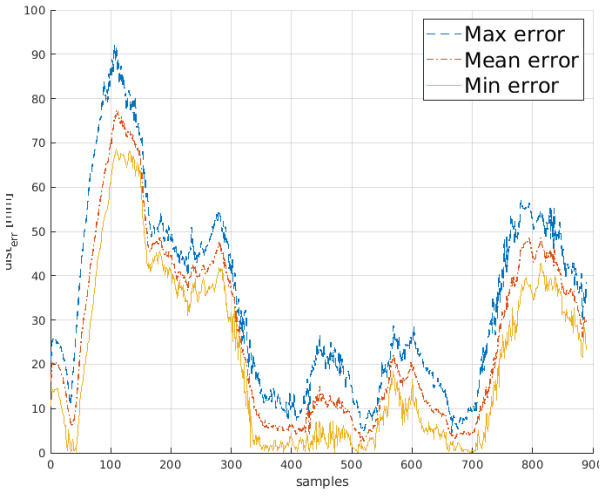
By analyzing the data presented above it is shown that both LiDAR SLAM and localization have approximately 1cm higher accuracy (lower mean residual) respectively than the developed camera system for Case 1. For Case 2 does LiDAR have approximately 6cm higher accuracy in SLAM and approximately 12cm for localization. One may also note that in both cases did the camera localization have higher peaks of error than the SLAM algorithm, even though it performed better than Camera SLAM for Case 1. This could indicate a poor map generated by the SLAM algorithm.

5.2.4 Trueness & Repeatability

In this section data is presented from running the SLAM algorithms five times for both the LiDAR and camera, and analyzed with regards to trueness and repeatability. The data is the same in every instance. Localization for both LiDAR and camera uses the first generated map by each corresponding SLAM algorithm. The localization are then tested five times for both systems.

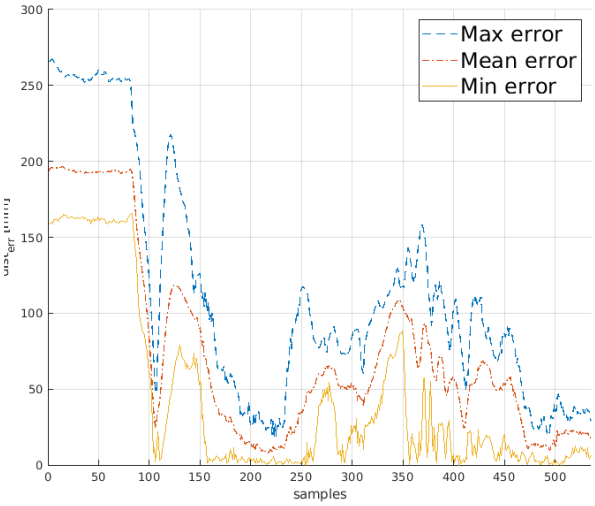


(a) Case 1: LiDAR SLAM five times on raw data.

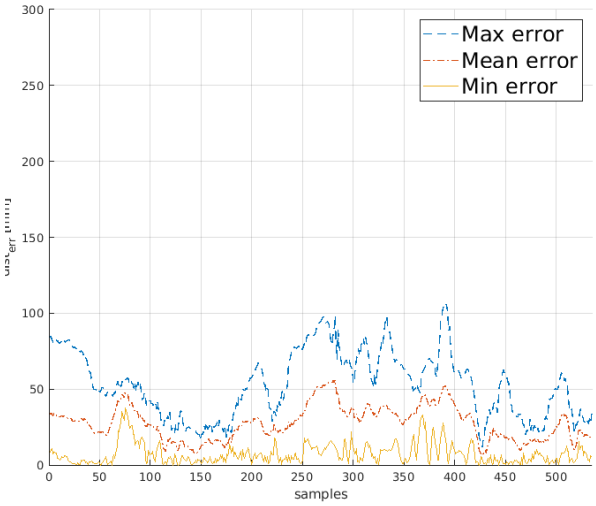


(b) Case 1: LiDAR Localization five times on raw data with the first map from LiDAR SLAM.

Figure 5.9

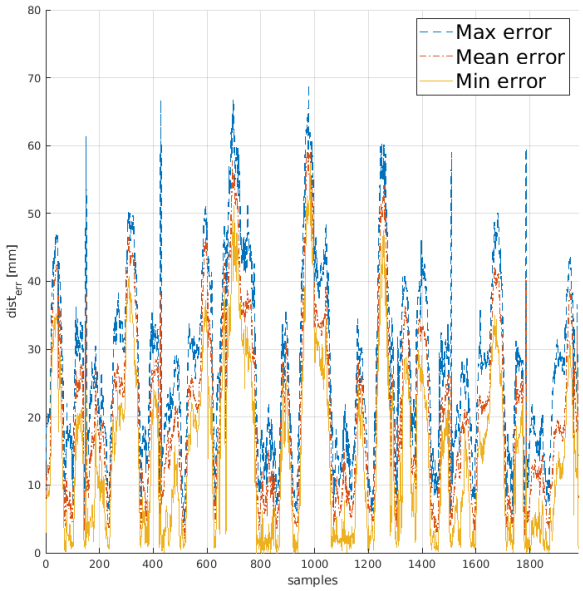


(a) Case 1: 3D-Cameras SLAM five times on raw data.

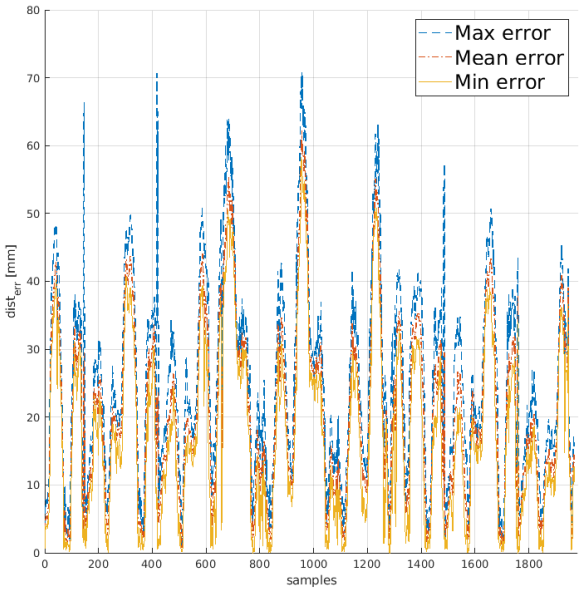


(b) Case 1: 3D-Camera Localization five times on raw data with the first map from 3D-Camera SLAM.

Figure 5.10

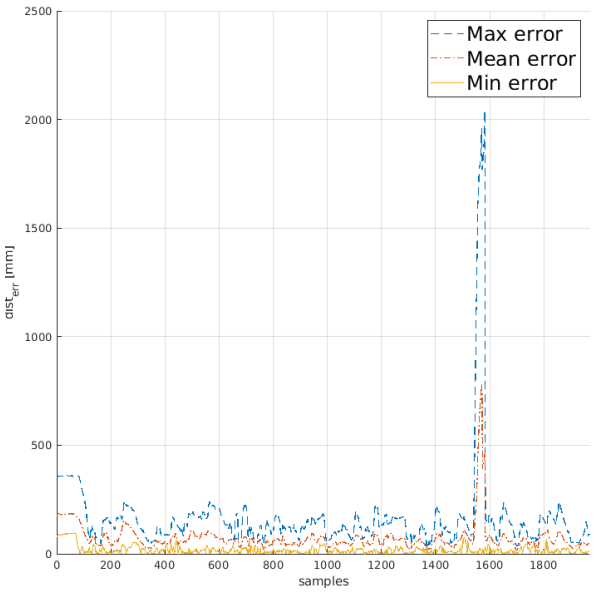


(a) Case 2: LiDAR SLAM five times on raw data.

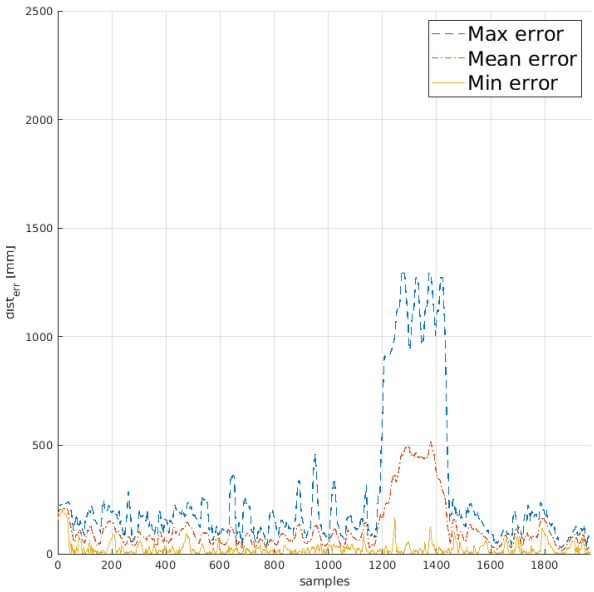


(b) Case 2: LiDAR Localization five times on raw data with the first map from LiDAR SLAM.

Figure 5.11



(a) Case 2: 3D-Cameras SLAM five times on raw data.



(b) Case 2: 3D-Camera Localization five times on raw data with the first map from 3D-Camera SLAM.

Figure 5.12

Table 5.2: Deviation error from running the same data five consecutive times in Case 1. Max and min is calculated by taking the max and min of each sample and the taking the mean of them respectively. All measurements are in mm.

Method:	LiDAR SLAM	LiDAR Localization	Camera SLAM	Camera Localization
Max:	26.8	31.8	82.9	32.8
Mean:	26.6	27.0	64.5	27.7
Min:	26.1	24.1	48.1	23.4

Table 5.3: Deviation error from running the same data five times consecutive in Case 2. Max and min is calculated by taking the max and min of each sample and the taking the mean of the respectively. All measurements are in mm.

Method:	LiDAR SLAM	LiDAR Localization	Camera SLAM	Camera Localization
Max:	23.0	24.0	112.1	141.3
Mean:	21.1	21.1	74.7	117.9
Min:	16.8	19.9	49.6	78.6

From Table 5.2 and Table 5.3 is it observable that the trueness (mean deviation error) of the LiDAR is generally below 3cm while the camera system struggles and only makes it

below 3cm in Case 1 when running localization. The mean repeatability for Case 1 and Case 2 be calculated to:

Table 5.4: Mean repeatability from Case 1 and Case 2. All measurements are in mm.

Method:	LiDAR SLAM	LiDAR Localization	Camera SLAM	Camera Localization
Case 1:	0.7	7.7	34.8	9.4
Case 2:	6.2	14.1	62.5	62.7

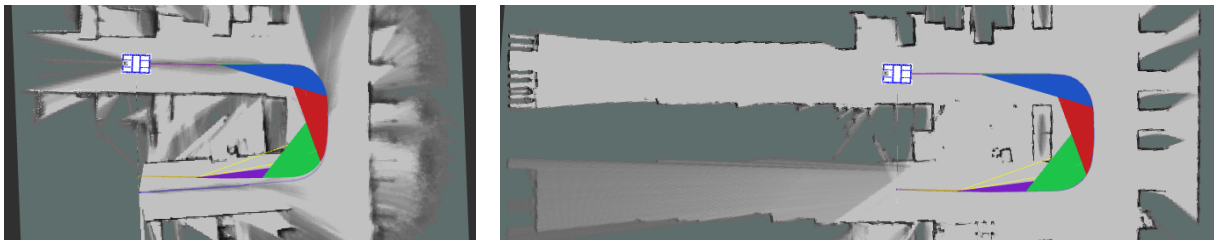
Analyzing the Figures 5.9a, 5.9b, 5.11a, and 5.11b yields that LiDAR is quite small in the spread between max and min, this can be verified with the mean repeatability in Table 5.4. From the plots can it also be noted that none of the cases have major separations between the max and min. When analyzing the Figures 5.10a, 5.10b, 5.12a, and 5.12b does they show that the camera on the other hand have a large spread overall between max and min in all instances except for localization in Case 1. Here is camera localization in the size range of the LiDAR repeatability. However, from the plots can it also be noted that there is some major separation between the max and min at some peaks. All these factors combined imply that the result from the camera is not as repeatable when compared to the LiDAR.

5.3 Qualitative systems evaluation

As no ground truth reference was available in the warehouse environment, the generated data was evaluated qualitatively. That is, by visual inspection and evaluating the relative distance deviation. The following evaluations are all from two data sets generated in the warehouse, one smaller illustrating the sequence of driving through two aisles as to complete a trajectory loop, and one larger to representing a full mapping of the operation area. This area approximates 750 square meters.

5.3.1 Map and constraint generation

The two images in Figure 5.13 illustrates both the 3D-camera 5.13a and 2D-LiDAR SLAM 5.13b algorithms initiating a map and a path estimation. The starting point of each trajectory, as seen in the bottom center of each image, is from the test platform facing the end of an aisle. An operator then manually moves the AGV, through the corner and down the next aisle. The trajectory constraints are also highlighted for both algorithms.



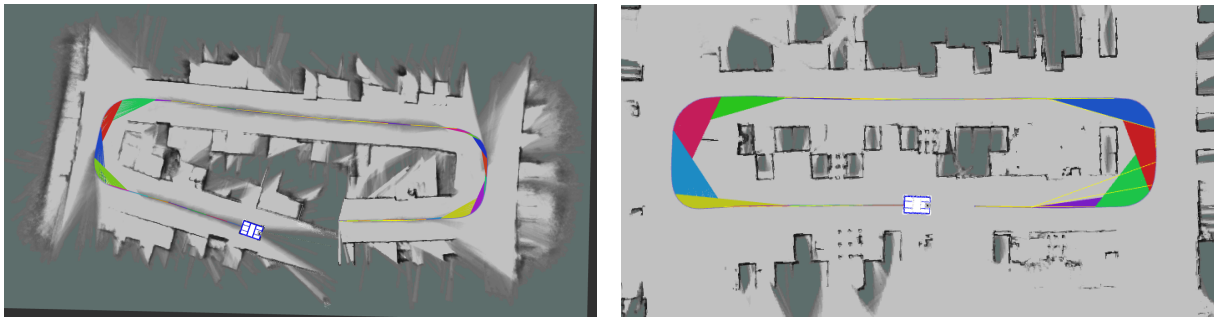
(a) Camera SLAM map and constraints generation

(b) LiDAR SLAM map and constraints generation

Figure 5.13: Map and constraints generation of Camera and LiDAR SLAM. Note the multicolored intra-map constrains are in similar size for both systems also indicating a similar size of submaps.

Note that the Camera SLAM in 5.13a generates a map that has started to warp when compared to the more accurate map generated by the LiDAR in 5.13b. Nonetheless, the constraints and trajectory are near identical for both systems, indicating a still comparable path estimation. It is also well apparent that the detection range of the LiDAR is far superior.

In the same sample sequence, the AGV continues down the aisle and the operator then takes another left turn, back into the previous aisle. The corresponding SLAMs are shown in Figure 5.14.



(a) Camera SLAM with major misalignment

(b) Equivalent position of LiDAR SLAM

Figure 5.14

In Figure 5.14a it is apparent that the two, parallel in real life, aisles do not align. The path however still correctly shows the AGV driving in the center of the aisle, as compared to the LiDAR trajectory seen in Figure 5.14b. Another noteworthy comparison of both maps is that, due to the sparse surfaces placed at the detection level of the LiDAR, the map it produces lacks many of the obstacles at each side of the aisle, whereas the camera detects multiple surfaces.

Finally, after continuing further down the same aisle, the camera SLAM detects a loop-closure, i.e it recognizes its position in the current environment as somewhere it has been

before. The SLAM algorithm can then realign the path and map to the correct position, as seen in Figure 5.15.

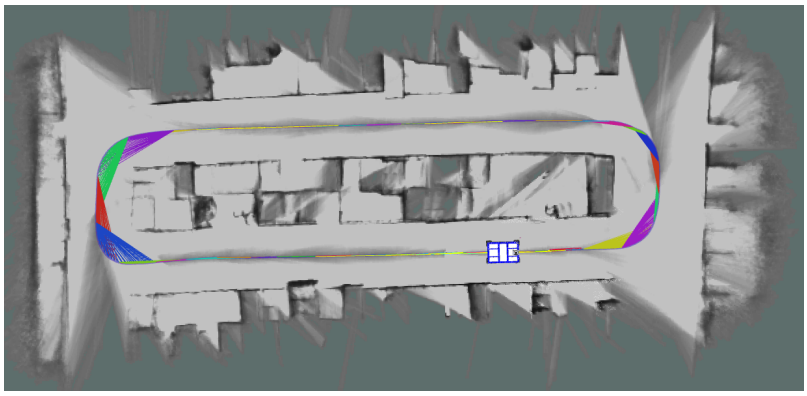
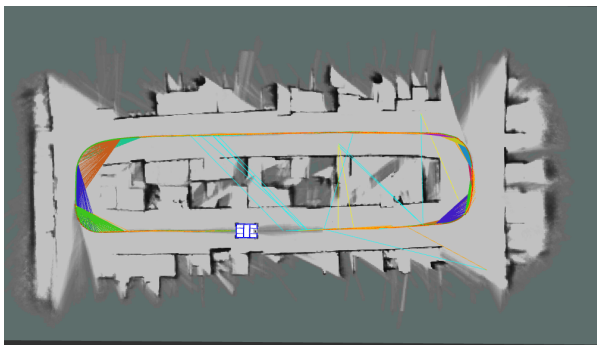
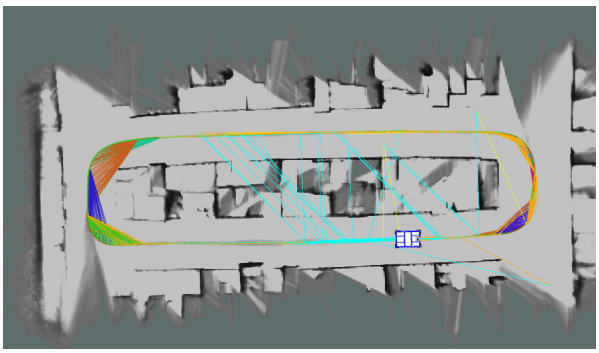


Figure 5.15: Camera SLAM post loop closure

In the above evaluations, all maps and trajectories were produced by their corresponding SLAM algorithm and with no prior knowledge of the environment. Below is the same environment data first processed by the SLAM algorithms to then generate a full map of the environment. Now, this map is used to achieve pure localization, i.e, the map itself is no longer update. This effectively prevents the misalignment that was seen in Figure 5.14a as the map is already realigned.



(a) Camera localization full map closure.



(b) Camera localization completer run

Figure 5.16

5.3.2 Pure Localization and SLAM comparison

For both localization systems, a larger map of the warehouse was generated to represent a case where a larger operation area was required. To get a comparable metric, the distance deviation of each trajectory was evaluated and which is plotted in Figure 5.17. This shows a maximal deviation of **335.7mm** and a mean deviation of **99.2mm**. The impact of this deviation is implementation specific and the LiDAR trajectory shall not be considered as a ground truth reference.

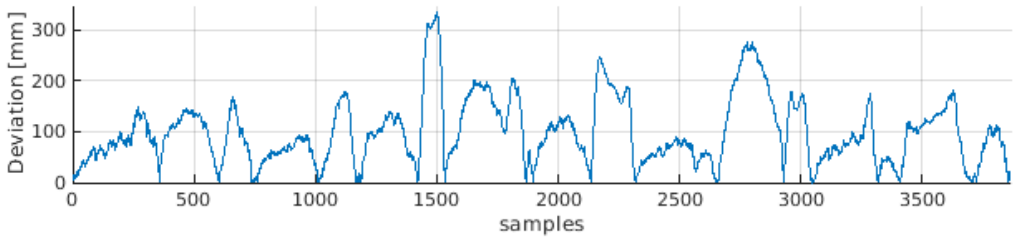


Figure 5.17: Per-sample deviation in LiDAR SLAM and camera SLAM trajectory estimations. **Max: 335.7mm, Mean: 99.2mm**

The generated map produced by the camera SLAM algorithm was then used to achieve pure localization. The result of the final map can be seen in Figure 5.18. The same figure also shows the trajectories generated by the camera algorithm, both by SLAM in blue, and pure localization in green.

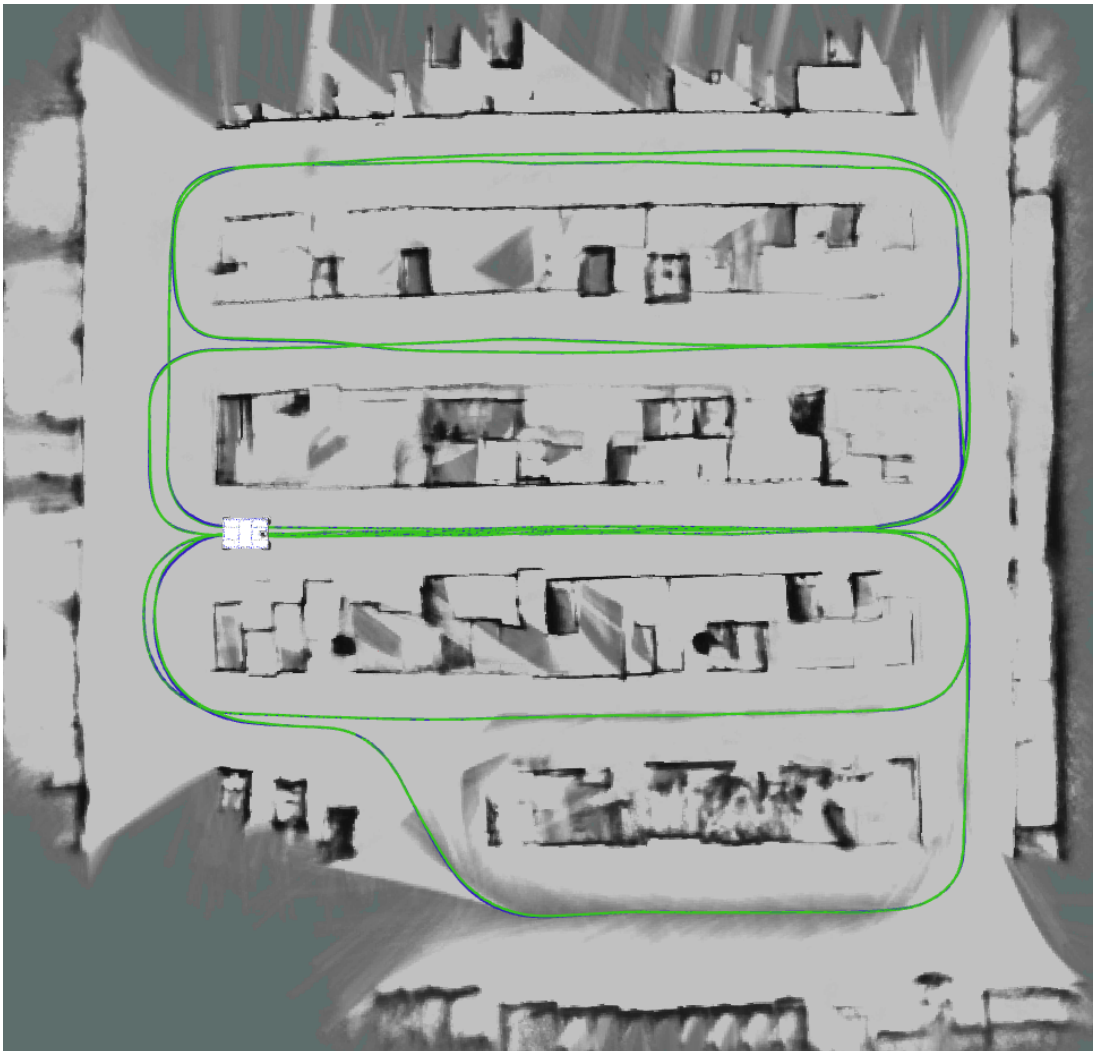


Figure 5.18: Full map generated by the camera SLAM. The Blue trajectory is generated by SLAM and green is generated by localization on the final map generated by SLAM.

The localization trajectory successfully follows the SLAM trajectory with a maximal deviation of **93.2mm** and mean deviation of **10.3mm**. The per-sample deviation is plotted in Figure 5.19. This indicates that pure localization is highly accurate and performs very well based on the map generated by the camera SLAM algorithm.

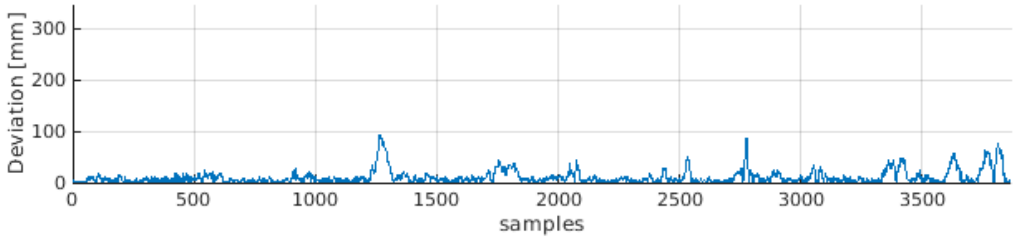


Figure 5.19: Per-sample deviation in camera localization and camera SLAM trajectory estimations. **Max: 93.2mm, Mean: 10.3mm**

6
Discussion

This section will discuss the results presented in the previous chapter and motivations for some of the choices made throughout the making of this thesis.

The SLAM algorithms were evaluated on a separate computer. This was done due to software limitations on the integrated computer which prevented the use of the Temporal averaging filter. Instead, raw-data was first captured on the internal computer of the AGV. The data was then offloaded onto a separate computer where all filters could be applied. The computer did however not have sufficient cooling and would thermal throttle under heavy load. Especially for the camera SLAM due to the additional filtering. After adding external cooling to the computer the results for the camera SLAM significantly improved in both accuracy and repeatability. We can however not be sure that the final results are the actual optimal as an even faster computer may improve the results even further. In addition, it have been indicated that a software update of the integrated computer is pending and which then may allow for complete implementation of the whole system on the integrated computer.

Cartographer was chose as the framework for SLAM due to its ability to handle both 2D- and 3D-SLAM. This was argued to generate a result from both investigated systems that are less dependent on the SLAM framework and more on the sensor solutions themselves. However, Cartographer was not developed with 3D-cameras in mind and is not optimized for this sensor type, neither does it benefit from the color image that 3D-cameras can produce. There exist SLAM frameworks that are designed for 3D-cameras and in theory, could produce better results for the panoramic 3D-camera solution than whats is possible with Cartographer.

Due to the result in Section 5.1 illustrating the importance of the optimization filters, the decision was made to not continue testing the performance difference on unoptimized filters in more cases since the performance of the unoptimized had such a hard time creating a correct map in the shorter case of the two used from the lab-room. Thus it would not be any point to try and run SLAM on the long case as the map was far from correct and the localization in further testing would also be guaranteed to fail.

In the testing of both the 2D-LiDAR and the panoramic 3D-camera does the setup heavily rely on the odometry data provided by the AGV. As can be noted in the results, it seem that most of the areas contributing to inaccurate maps and trajectories for the panoramic 3D-camera setup, are around corners. One theory for this behavior is that in corners are the AGVs wheels slipping and underestimating the rotation of the AGV. As can be seen

in Section 5.3, it is also observed that in the straight corridors no slip is occurring, thus that section works quite well, further strengthening this theory. In the same section is it however also observed that the LiDAR is not affected to the same extent, if any. A theory to this is the longer range and higher accuracy measurements make the SLAM algorithm see more features and therefore is better at parrying the slippage occurring in the corners.

By using color data could visual odometry and visual SLAM be utilized. We cannot say if this would yield any performance benefits but it would be an alternative or additional approach to the problem. The color image could also be used for object detection and object avoidance as previously mentioned. Being able to do this with one single sensor were the alignment between color and depth data is already known could prove beneficial, since it would be possible to use Machine Learning or AI in order to detect objects in the image and then with the help of the depth data get a depth reading.

Only structured light and stereo vision cameras were further investigated in this thesis. However, one sensor technique also considered was that of the LiDAR-camera. This is a sensor that could potentially combine both the benefits of the long and precise depth measurements of a LiDAR and RGB data from a camera. The sensor type was however discarded due to the lack of scientific publications evaluating its performance in comparison to other 3D-cameras. Additionally, the only camera found at market in a comparable price range could not handle other sensors of the same type being in the same view. This would make it unsuitable for our proposed panoramic solution which relies on a overlap in the view. It would also be impossible to apply on multiple AGVs if these are to be operating in the same area. However, due to it's larger detection range and higher accuracy compared to the previously mentioned sensor types, making this kind of sensor an interesting alternative for future work. As long as the inability to handle crosstalk is solved.

7
Conclusion

To conclude, this work has proposed a panoramic 3D-camera system based on the Realsense D455 camera for natural localization in industrial environments. In order to improve the data obtained by the 3D-cameras, two optimization filters have been developed. These have then been shown to significantly improve the proposed localization system. Compared to previous work in this field, is it now possible to obtain usable maps and trajectories with the 3D-cameras if using a system such as the panoramic 3D-camera setup and optimization filter proposed in this thesis.

Comparing the trajectories produced by both systems via a ground truth reference, is it shown that the trajectories produced by the panoramic 3D-camera system do not produce the same level of accuracy, trueness, or repeatability as the 2D-LiDAR. However, it still manages to produce a continuous trajectory in both small and large environments, as shown in the presented tests.

There could be a performance to gain from testing a SLAM framework designed for 3D-cameras and new sensors are coming to market rapidly improving the performance of 3D-cameras.

References

[1] A. Chihani. (Nov. 2018). "What is an agv?" [Online]. Available: https://www.flexqube.com/news/what-agv/.

[2] (Feb. 2020). "Automated guided vehicle market size, share & trends analysis report by vehicle type, by navigation technology, by application, by end-use industry, by component, by battery type, by region, and segment forecasts, 2020 - 2027," [Online]. Available: https://www.grandviewresearch.com/industry-analysis/automated-guided-vehicle-agv-market.

[3] L. Sabattini, V. Digani, C. Secchi, G. Cotena, D. Ronzoni, M. Foppoli, and F. Oleari, "Technological roadmap to boost the introduction of agvs in industrial applications," in *2013 IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2013, pp. 203–208. DOI: 10.1109/ICCP.2013.6646109.

[4] BlueBotics. (Jan. 2021). "Agv navigation methods 1: Line following and tags," [Online]. Available: https://bluebotics.com/agv-navigation-line-following-tags/.

[5] A. TREVOR and H. CHRISTENSEN, "Automated guided vehicle survey," Georgia Institute of Technology, Tech. Rep., 2009.

[6] N. MADALI. (Jul. 2020). "Light detection and ranging (lidar)," [Online]. Available: https://towardsdatascience.com/light-detection-and-ranging-lidar-8f22971eeaea.

[7] H. Weber, "Lidar sensor functionality and variants," Product Unit Ranging LiDAR sensors at SICK AG, Whitepaper, Jul. 2018.

[8] T. Q. Tran, A. Becker, and D. Grzechca, "Environment mapping using sensor fusion of 2d laser scanner and 3d ultrasonic sensor for a real mobile robot," *Sensors*, vol. 21, no. 9, 2021, ISSN: 1424-8220. DOI: 10.3390/s21093184. [Online]. Available: https://www.mdpi.com/1424-8220/21/9/3184.

[9] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, p. 2068, Apr. 2020.

[10] R. Bostelman, T. Hong, and R. Madhavan, "Towards agv safety and navigation advancement obstacle detection using a tof range camera," in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, 2005, pp. 460–467. DOI: 10.1109/ICAR.2005.1507450.

[11] P. Pratama, N. Trong Hai, H.-K. Kim, D. H. Kim, and S. Kim, "Positioning and obstacle avoidance of automatic guided vehicle in partially known environment,"

International Journal of Control, Automation and Systems, vol. 14, Oct. 2016. DOI: 10.1007/s12555-014-0553-y.

[12] J. Manhed, "Investigating simultaneous localization and mapping for an automated ground vehicle," M.S. thesis, Linköping University, 2019.

[13] MathWorks. (Jan. 2021). "Slam (simultaneous localization and mapping)," [Online]. Available: https://www.mathworks.com/discovery/slam.html.

[14] J. Dai, L. Yan, H. Liu, C. Chen, and L. Huo, "An offline coarse-to-fine precision optimization algorithm for 3d laser slam point cloud," *Remote Sensing*, 2019.

[15] S. Ji, Z. Qin, J. Shan, and M. Lu, "Panoramic slam from a multiple fisheye camera rig," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 159, pp. 169–183, 2020, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2019.11.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271619302758.

[16] G. Robots. (Jan. 2019). "What is lidar technology?" [Online]. Available: https://blog.generationrobots.com/en/what-is-lidar-technology/#:~:text=For%20a%202D%20LiDAR%20only,on%20X%20and%20Y%20axes.%5C&text=For%20a%203D%20LiDAR%2C%20the,X%2C%20Y%20and%20Z%20axes..

[17] T. Raj, F. Hashim, A. Huddin, M. F. Ibrahim, and A. Hussain, "A survey on lidar scanning mechanisms," *Electronics*, vol. 9, p. 741, Apr. 2020. DOI: 10.3390/electronics9050741.

[18] W. Boonsuk. (Aug. 2016). "Investigating effects of stereo baseline distance on accuracy of 3d projection for industrial robotic applications," [Online]. Available: http://cd16.iajc.org/wp-content/uploads/Camera-ready-papers/101-x-16___Investigating%20Effects%20of%20Stereo%20Baseline%20Distance%20_REVISED--Boonsuk_.pdf.

[19] T. Jia, Z. Zhou, and H. Gao, "Depth measurement based on infrared coded structured light," *Journal of Sensors*, vol. 2014, p. 8, 2014. DOI: https://doi.org/10.1155/2014/852621.

[20] R. Hamzah, M. S. Hamid, H. Rosly, N. M. Z. Hashim, and Z. A. F. M. Napiah, "An aligned epipolar line for stereo images with multiple sizes roi in depth maps for computer vision application," *International Journal of Information and Education Technology*, pp. 15–19, Jan. 2011. DOI: 10.7763/IJIET.2011.V1.3.

[21] G. Farnebäck, "The stereo problem," M.S. thesis, Link¨oping University, Sweden, 2002.

[22] V. Tata. (May 2020). "3-d reconstruction with vision," [Online]. Available: https://towardsdatascience.com/3-d-reconstruction-with-vision-ef0f80cbb299.

[23] S. Routray, A. K. Ray, and C. Mishra, "Analysis of various image feature extraction methods against noisy image: Sift, surf and hog," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2017, pp. 1–5. DOI: 10.1109/ICECCT.2017.8117846.

[24] I. Cabezas, V. Padilla, and M. Trujillo, "A measure for accuracy disparity maps evaluation," vol. 7042, Nov. 2011, pp. 223–231, ISBN: 978-3-642-25084-2. DOI: 10.1007/978-3-642-25085-9_26.

[25] R. Jain, R. Kasturi, and B. G. Schunck, "Machine vision," in. Published by McGraw-Hill, Inc, 1995, ch. 11.

[26] M. F. Abu Hassan, A. Hussain, M. H. Md Saad, and K. Win, "3d distance measurement accuracy on low-cost stereo camera," *Scientific International*, vol. 29, pp. 599–605, May 2017.

[27] M. Gustafsson and P. Jarnemyr, "3d camera selection for obstacle detection in a warehouse environment," M.S. thesis, Linköping University, 2020.

[28] von Terry Arden. (Mar. 2017). "The advantages of stereo snapshot over single camera design," [Online]. Available: https://lmi3d.com/blog/advantages-stereo-snapshot-over-single-camera-design/.

[29] S. Doggett. (Jan. 2020). "What are point clouds, and how are they used?" [Online]. Available: https://www.dronegenuity.com/point-clouds/.

[30] T. M. |. (Jan. 2021). "What is point cloud modeling?" [Online]. Available: https://www.takeoffpros.com/2020/07/14/what-is-point-cloud-modeling/.

[31] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," May 2009, pp. 3206–3211. DOI: 10.1109/ROBOT.2009.5152493.

[32] K. Pulli and M. Pietikäinen, "Range image segmentation based on decomposition of surface normals," 2004.

[33] T. Hashimoto and M. Saito, "Normal estimation for accurate 3d mesh reconstruction with point cloud model incorporating spatial structure," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019.

[34] Rbrusu, J. Sprickerhof, G. Brindeiro, and F. Kuang. (Sep. 2018). "The pcd (point cloud data) file format," [Online]. Available: https://pcl.readthedocs.io/projects/tutorials/en/latest/pcd_file_format.html#pcd-file-format.

[35] C. Moreno and M. Li, "Frame filtering and skipping for point cloud data video transmission," *Advances in Science, Technology and Engineering Systems Journal*, vol. 2, pp. 76–83, Jan. 2017. DOI: 10.25046/aj020109.

[36] W. Boonsuk. (20119). "Post-processing filters," [Online]. Available: https://dev.intelrealsense.com/docs/post-processing-filters.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[38] P. C. Lib. (Jan. 2021). "Downsampling a pointcloud using a voxelgrid filter," [Online]. Available: https://adioshun.gitbooks.io/pcl/content/Tutorial/Filtering/pcl-cpp-downsampling-a-pointcloud-using-a-voxelgrid-filter.html.

[39] Q.-Y. Zhou, J. Park, and V. Koltun, *Open3d*, https://github.com/intel-isl/Open3D, 2021.

[40] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM TOG*, vol. 30, no. 4, 69:1–69:12, 2011, Proceedings of SIGGRAPH 2011.

[41] D. T. Anders Grunnet-Jepsen, *Depth post-processing for intel® realsense™ depth camera d400 series*, https://dev.intelrealsense.com/docs/depth-post-processing, Jun. 2020.

[42] E. Gedraite and M. Hadad, "Investigation on the effect of a gaussian blur in image filtering and segmentation," Jan. 2011, pp. 393–396, ISBN: 978-1-61284-949-2.

[43] R. L. Lagendijk, J. Biemond, A. Rareş, and M. J. Reinders, "Chapter 4 - video enhancement and restoration," in *The Essential Guide to Video Processing*, A. Bovik, Ed., Boston: Academic Press, 2009, pp. 69–108, ISBN: 978-0-12-374456-2. DOI: https://doi.org/10.1016/B978-0-12-374456-2.00005-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123744562000050.

[44] R. Kromer, A. Abellan, D. Hutchinson, M. Lato, T. Edwards, and M. Jaboyedoff, "A 4d filtering and calibration technique for small-scale point cloud change detection with a terrestrial laser scanner," *Remote Sensing*, vol. 7, pp. 13 029–13 052, Oct. 2015. DOI: 10.3390/rs71013029.

[45] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. DOI: 10.1109/34.121791.

[46] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," Nov. 2013, pp. 225–230, ISBN: 978-1-4799-1201-8. DOI: 10.1109/RAM.2013.6758588.

[47] R. Bondemark, "Improving slam on a tof camera by exploiting planar surfaces," M.S. thesis, Linköpings Universitet, Sep. 2016.

[48] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison, "Dense planar slam," in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014, pp. 157–164. DOI: 10.1109/ISMAR.2014.6948422.

[49] A. Trevor, S. Gedikli, R. Rusu, and H. Christensen, "Efficient organized point cloud segmentation with connected components," *Proceedings of Semantic Perception Mapping and Exploration, S.*, pp. 1–6, Jan. 2013.

[50] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," vol. 1, Jan. 2009, pp. 331–340.

[51] R. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI - Künstliche Intelligenz*, vol. 24, Nov. 2010. DOI: 10.1007/s13218-010-0059-6.

[52] E. W. Weisstein. (). "Hessian normal form.," [Online]. Available: https://mathworld.wolfram.com/HessianNormalForm.html (visited on 05/24/2021).

[53] H. Durrant-Whyte, D. Rye, and E. Nebot, "Localization of autonomous guided vehicles," in *Robotics Research*, G. Giralt and G. Hirzinger, Eds., London: Springer London, 1996, pp. 613–625, ISBN: 978-1-4471-0765-1.

[54] M. Veisman, Y. Noam, and S. Gannot, "The hybrid cramér-rao lower bound for simultaneous self-localization and room geometry estimation," *EURASIP Journal on Advances in Signal Processing*, vol. 2021, no. 1, Jan. 2021. DOI: 10.1186/s13634-020-00702-6. [Online]. Available: http://dx.doi.org/10.1186/s13634-020-00702-6.

[55] H. Durrant-Whyte, D. Rye, and E. Nebot, "Localization of autonomous guided vehicles," in *Robotics Research*, G. Giralt and G. Hirzinger, Eds., London: Springer London, 1996, pp. 613–625, ISBN: 978-1-4471-1021-7.

[56] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, and Y. Chong, "Sensor technologies and simultaneous localization and mapping (slam)," *Procedia Computer Science*, vol. 76, pp. 174–179, 2015, 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IEEE IRIS2015), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.12.336. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915038375.

[57] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001. DOI: 10.1109/70.938381.

[58] roboticsknowledgebase. (Aug. 2018). "Apriltags," [Online]. Available: https://roboticsknowledgebase.com/wiki/sensing/apriltags/#:~:text=AprilTags%20is%20a%20visual%20fiducial,even%20in%20low%20visibility%20conditions..

[59] F. Li, J. Hao, J. Wang, J. Luo, Y. He, D. Yu, and X. Cheng, "Visiomap: Lightweight 3-d scene reconstruction toward natural indoor localization," *IEEE Internet of Things Journal*, vol. 6, pp. 8870–8882, Oct. 2019. DOI: 10.1109/JIOT.2019.2924244.

[60] G.-S. Cai, H.-Y. Lin, and K. Shih-Feng, "Mobile robot localization using gps, imu and visual odometry," Nov. 2019, pp. 1–6. DOI: 10.1109/CACS47674.2019.9024731.

[61] J. G. da Silva Neto, P. J. da Lima Silva, F. Figueredo, J. M. X. N. Teixeira, and V. Teichrieb, "Comparison of rgb-d sensors for 3d reconstruction," in *2020 22nd Symposium on Virtual and Augmented Reality (SVR)*, 2020, pp. 252–261. DOI: 10.1109/SVR51698.2020.00046.

[62] A. Grunnet-Jepsen, A. T. Paul Winer, J. Sweetser, K. Zhao, T. Khuong, D. Nie, and J. Woodfill, "Multi-camera configurations - d400 series stereo cameras," Tech. Rep., 2020.

[63] Hokoyo, *Safety laser scanner safety laser scanner uam-05lp user's manual*, English, Hokuyo, 151 pp., published.

[64] A. Pålsson and M. Smedberg, "Investigating simultaneous localization and mapping for agv systems," M.S. thesis, Chalmers University of Technology / Department of

Computer Science and Engineering, https://hdl.handle.net/20.500.12380/250073, 2017.

[65] B. Gerkey. (Feb. 2019). "Gmapping," [Online]. Available: http://wiki.ros.org/gmapping.

[66] S. Kohlbrecher and J. Meyer. (Apr. 2014). "Hector slam," [Online]. Available: http://wiki.ros.org/hector_slam.

[67] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[68] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," Sep. 2018. DOI: 10.1109/IS.2018.8710464.

[69] R. Milijas, L. Markovic, A. Ivanovic, F. Petric, and S. Bogdan, *A comparison of lidar-based slam systems for control of unmanned aerial vehicles*, 2021. arXiv: 2011.02306 [cs.RO].

[70] A. Nuchter, M. Bleier, J. Schauer, and P. Janotta, "Improving google's cartographer 3d mapping by continuous-time slam," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W3, pp. 543–549, Feb. 2017. DOI: 10.5194/isprs-archives-XLII-2-W3-543-2017.

[71] C. Authors. (Jan. 2021). "Cartographer ros integration," [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/.

[72] ——, (2021). "Algorithm walkthrough for tuning." Revision c06879b6, [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html.

[73] *Qualisys track manager*, 2.5, Qualisys AB, Mar. 2017.

DEPARTMENT OF ELECTRICAL ENGINERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS