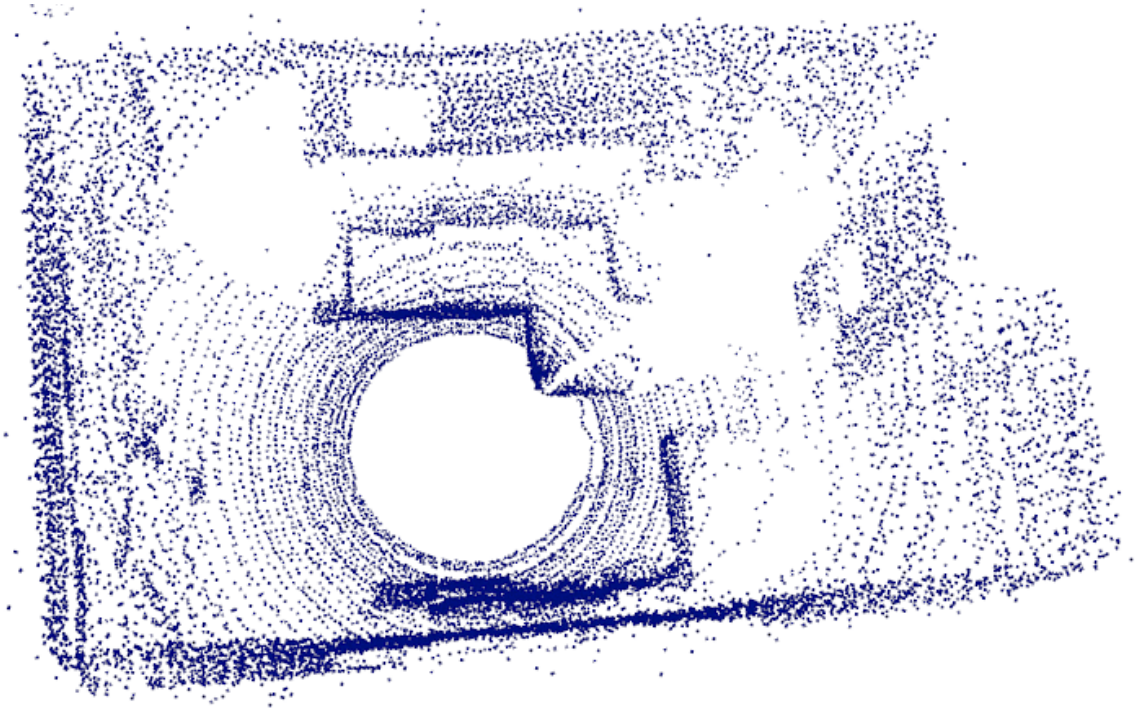
CHALMERS



Automated Dimension Calculations using a Constructed LiDAR 3D Scanner

Bachelor's thesis in Computer Science and Engineering

RASMUS CLAESÉN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

BACHELOR'S THESIS

Automated Dimension Calculations using a Constructed LiDAR 3D Scanner

RASMUS CLAESÉN

Department of Computer Science and Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG

Gothenburg, Sweden 2020

Automated Dimension Calculations using a Constructed LiDAR 3D Scanner

RASMUS CLAESÉN

Supervisor: Sakib Sistek, Department of Computer Science and Engineering.
Examiner: Jonas Duregård, Ph.D., Department of Computer Science and Engineering.

ISSN 1654-4676

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
3D-scan Point Cloud Visualization

Chalmers Reproservice
Gothenburg, Sweden 2020

Automated Dimension Calculations using a Constructed LiDAR 3D Scanner

RASMUS CLAESÉN
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg

Bachelor's thesis

Abstract

With an increased interest in autonomous agents, the need for digital models of the surrounding environment is increasing. This thesis explores the usages of a LiDAR sensor in a rotating platform, and in particular the ability to perform automated dimension calculation of closed rooms environments. The construction of the platform is proceeded by error estimation and the implementation of a histogram filtering method. Further, processing of *Point Clouds* with Point Cloud Library is performed and methods for surface estimations developed. In the presented test cases, automated estimations are successful and the usability is discussed.

Keywords: LiDAR, 3D-Scanning, Automated Dimension Calculation, ROS, PCL, RANSAC

Acknowledgements

The author would like to express gratitude towards supervisor Sakib Sistek at Chalmers University of Technology for his immense engagement and support in the project, it has been fundamental for the success of the project. Further, gratitude is expressed towards friends and family for their support during this thesis.

Rasmus Claesén, Gothenburg, March 2020.

Contents

List of Figures

List of Tables

1 Introduction

The first chapter gives an introduction to the general background of the area covered by this thesis. Further, the purpose and objective is declared within the given scope. Finally, related work is presented and discussed.

1.1 Background

As technology continuously improve our understanding of the world, a field which concerns itself with modelling the physical world is Light Detection and Ranging (LiDAR). LiDAR systems are used to perform digital examinations of environments with accuracy and precision. It has commonly been used in geographic information systems, shoreline and coastal maps as well as digital elevations models[1]. LiDAR application within robotics and autonomous agents has been increasing in recent years, as the interest and need for digital modelling of physical surroundings has increased[2].

In general, the output of a LiDAR system is represented as a digital 3D model or 3D scan. This coincides with developments within 3D modelling, often performed by stereo-camera setups[3]. Coupled with the development of simpler and cheaper LiDAR sensors, it provides an interesting platform to build and explore usage areas of these types of 3D scanning systems.

This thesis will explore hardware construction utilizing a static single-laser LiDAR sensor in a rotational platform, and explore what level of performance this type of system is able to yield. It builds on a platform of established robotics software and conducts measurements, pre-processing, processing and visualization of 3D scans in what is known as a *Point Cloud*.

1.2 Purpose

The purpose of this project is to construct a platform which performs 3D scans of closed room environments using a LiDAR sensor. Further, the collected data, the 3D scan, is to be evaluated and tested in the application of automated dimensions calculation and extended applications is discussed.

1.3 Objective

There are three distinct objective's for this project.

- Construct and evaluate the performance of a 3D scanning platform using a LiDAR-sensor.
- Process and evaluate the quality of the 3D scan in a closed environment.
- Determine if it possible to automate dimension estimation in a closed environment.

1.4 Scope

Performance and quality is an effect of complex interactions between the rotational platform and the LiDAR sensor. Therefore, performance is evaluated in terms of speed and resolution provided by the platform and quality is evaluated by visualization of the *Point Cloud* and in terms of measurement errors of the data points scanned.

No tests are conducted on the constructed platform and errors caused solely by the platform itself, although potential issues and how they affect the scan is discussed. Dimension estimation are limited to appliances of surface estimations in closed room environments with geometrical simplifications done. Further, common algorithms for 3D-spaces are utilized.

1.5 Related Work

The idea of combining hardware and software systems to create customized LiDAR scans was in part inspired by the *The Digital Michelangelo Project*[4], which utilizes LiDAR sensors together with digital still camera images to reconstruct the shape and color of large fragile objects under non-laboratory conditions. While the scope of the project is of particular interest to extensions to the system in this thesis, it is an inspirational project within 3D scanning and how to increase and fuse information in 3D scans.

A similar project, *LIDAR Application for Mapping and Robot Navigation on Closed Environment*[2], performs conduct Simultaneous localization and mapping (SLAM) on a 2D rotational LiDAR platform and shares similarities with this thesis. It utilizes the same LiDAR sensor and provides detailed information on the sensor distance measurement quality. This provides an understanding to how this sensor is affected by rotation.

Sensor Systems Simulations: From Concept to Solution[5] presents a virtual white cane concept using a 3D imaging camera where the problem and proposed solution was of guidance in the pre-processing of the data, the disparity histogram was of particular interest for this in this thesis.

2 Method

The method discusses the literature and choices made prior to constructing the 3D scanning system based on given factors and their influence on the hardware and software of the project.

2.1 Literature Review

This theses literature consists of scientific articles, technical information sheets of the components and blog posts, or similar online entries, of specific implementations. The article search was done through the Chalmers University Library and Google Scholar. Keywords used are *LiDAR*, *3D Scan*, *point cloud*, *filter*, *histogram* and *rotational platform* in various combinations. Furthermore, references from relevant articles and articles which referred to the current article was explored among the articles that emerged during the search.

2.2 Development

A given part of the project is utilizing a specific LiDAR sensor to perform scans, detailed further in 3.2.1. Therefore, a platform which is able to rotate this sensor was required where solutions offering low complexity and rapid prototyping was preferred. Keeping the sensor static, only performing rotation on a mirror was a solid choice to reduce noise from rotating the sensor itself. However, this was overlooked in favour of a lesser complex design rotating the sensor directly. This decreases the need for precision in the construction and thus reducing the development time. This choice meant performing a two-axis rotation of the sensor while not obstructing the field-of-view of the sensor.

Another influencing decision of the hardware was the foundation of the platform. With the previous decision, ensuring knowledge of rotation relative the sensor's origo is necessary. One solution was to build the platform on top of a motor thus solving rotation of one axis while allowing rotation of the second axis on top. Another solution consists of a structure containing an independent centred pole on which the LiDAR sensor is mounted, which was preferred. This solution causes extra build complexity as rotation of one axis would have to be transferred to the pole, but allows the sensor to be aligned with the rotation of this axis and provides straightforward cable-management through a hollow pole.

Given these specifications, a deductive method was used to decide the specifications of the components needed to construct the scanner platform, supported by Computer Aided Design (CAD) models. Further, a micro-controller could be assumed to generally lack the processing power and connectivity to control the scanner platform as well as process and visualize the data contained within point clouds in this kind of setup and therefore the system is designed to utilize an external processing device for processing and visualization of the data.

On the top level processing methods such as filtering, segmentation and visualization are applied. With respect to the magnitude of available options to perform this, existing software libraries will be utilized and this thesis will develop supplementary functions as parts of an existing system to achieve the stated goals.

3 Technical Background

This chapter describes the technical aspect of the project such as the terminology used, specifications of the components, core software libraries and an introduction to the algorithm applied.

3.1 Motion Terminology Definitions

This thesis utilizes a spherical coordinate representation while performing the scan, as this corresponds well to the rotations of the platform with the sensor located in the origin. For processing of the *Point Cloud*, a Cartesian coordinate system used. The order of spherical representation is defined here as

$$(radial, azimuthal, polar) \tag{3.1}$$

with the notation

$$(\rho, \theta, \phi). \tag{3.2}$$

They are related to Cartesian coordinates

$$(x, y, z) \tag{3.3}$$

as

$$\rho = \sqrt{x^2 + y^2 + z^2} \tag{3.4}$$
$$\theta = \arctan \frac{y}{x} \tag{3.5}$$
$$\phi = \arccos \frac{z}{r} \tag{3.6}$$

where

$$\rho \in [0, \infty), \tag{3.7}$$
$$\theta \in [0, 2\pi), \tag{3.8}$$
$$\phi \in [0, \pi]. \tag{3.9}$$

And translates to Cartesian coordinates as

$$x = \rho \cos\theta \sin\phi \tag{3.10}$$
$$y = \rho \sin\theta \sin phi \tag{3.11}$$
$$z = \rho \cos\phi. \tag{3.12}$$

The rotating motion around the axes y and z is referred to as *pitch* and *yaw* respectively. Figure 3.1 illustrates the spherical measurements in relation to the scanner platform and the rotational axis.

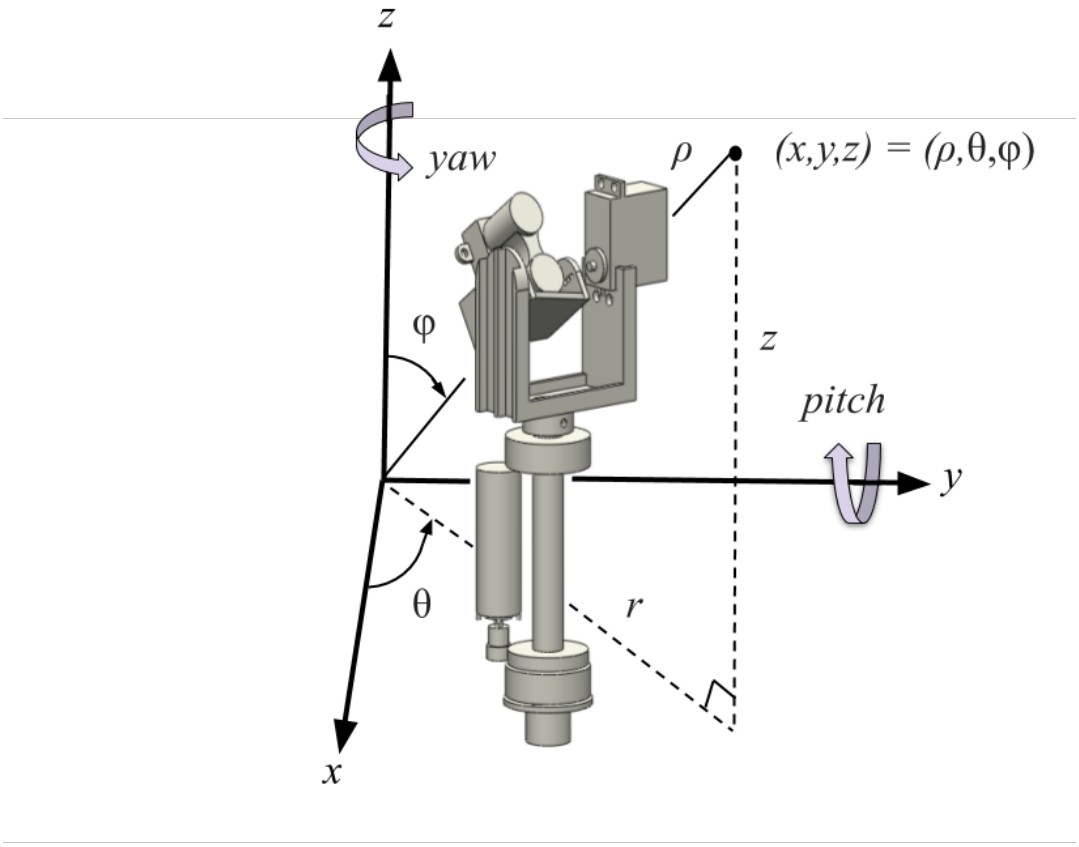


Figure 3.1: Spherical coordinate system in the world frame. ϕ represents the pitch angle of the sensor mounted on the platform, while θ represents the yaw angle of the mounted sensor. Yaw and Pitch is only for illustrative purposes.

3.2 Hardware

The hardware section introduces the main hardware components, and their specifications, used to construct the 3D scanning platform.

3.2.1 LiDAR

The LiDAR used is LiDAR-Lite v.3 produced by Garmin[6], where technical specifications of the sensor can be found. It is a compact and lightweight optical distant measurement sensor solution for drones, robotics and UAVs.

3.2.2 Motors

In order to to facilitate the yaw rotation of the platform a DC motor used, due to the linear movement as well as the speed provided. The chosen DC motor is a planetary gear brush motor produced by Robot Zone. It has a no-load maximum speed of 350(35) RPM and a gear-ratio of 42.875: 1. The DC motor contains a built in hall-effect magnetic encoder, an inductive sensor which registers each time a corresponding magnetic field passes with a desired resolution[7]. This specific encoder has a resolution of 3 measuring points per revolution (ppr) of the DC motor and is used to measure the yaw angle change during the scan.

The pitch rotation is performed by a servo motor. It was chosen for being lightweight and the ease of reading the angular position. The motor used is a standard servo motor produced by Parallax. It provides a 180° range of motion and position, and is used to control and measure the pitch rotation and angle respectively.

3.2.3 Gears

Two gears are used to transfer the rotational force from the DC motor to a pole with the top mounted sensor along the yaw axis. A two-gear train is utilized to facilitate the transfer, which is illustrated in figure 3.2. The gear ratio, or speed ratio, R, is calculated as

$$R = \frac{N_{driven}}{N_{drive}} \tag{3.13}$$

In the case where N_{drive} has fewer number of teeth than N_{driven}, the gear train is called a speed reducer and amplifies the input torque. In this case, the two-gear train also increases the final resolution of the yaw angle.

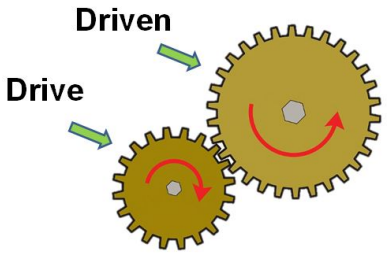


Figure 3.2: Two-gear train illustration with rotations shown.

3.2.4 Arduino

Control of the LiDAR sensor and the motors, as well as transmission of the scanned data is performed by a microcontroller. There are plenty of choices of microcontrollers, and a common choice is Arduino microcontrollers. It is an open-source electronics platform focused on ease-of-use hardware and software. The Arduino Uno Rev 3 microcontroller board, based on the ATmega328P produced by Atmel[8], is used as the scanner platform controller. It contains 14 digital input/output pins as well as a USB connector, which is enough to connect all hardware. To further simplify the motor control, Adafruit Motorshield v2.3 is used. It is a Arduino shield based on the TB6612 MOSFET driver[9]. The shield has terminal block connectors for the DC motor as well as Arduino software packages, all which help reduce the time spent developing the scanner platform.

3.3 Software

The software section describes libraries and methods used in the thesis. ROS provides a back-end for the operations, PCL is utilized as the processing library for *Point Clouds* and the RANSAC algorithm is introduced.

3.3.1 Robot Operating System

Figure 3.3: Robot Operating System (ROS)[10].

Robot Operating System (ROS)[10] is commonly referred to as middleware, and provides at its core a message passing interface that provides inter-process communication. It is a framework used to create software within robotics and provides an appealing modular node-based infrastructure for this thesis. It contains tools, libraries, and conventions that simplifies creating complex and robust behaviors in robotic applications. It is a meta-operating system and released as open-source, and supports the programming languages C++ as well as Python. Features of ROS include hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

Some of the core components of ROS utilized in this project are the communications infrastructure. This includes communication between distributed nodes through an anonymous publish/subscribe mechanism, which benefits implementations of clear interfaces between the nodes in the system and helps improving encapsulation. A ROS message is a data structure comprised of typed fields. There are pre-defined message types for geometric concepts like poses, transforms, and vectors as well as for sensors like cameras, IMUs and lasers which integrates well with the Point Cloud data structures used within processing.

The publish/subscribe system is anonymous and asynchronous, and ROS provides tools so that the data can easily be captured and replayed through the tool rosbag [11]. This is beneficial when a scan is done, as the scanned data easily can be captured and later replayed while the abstraction of the message-passing allows the processing nodes to be agnostic to the source of the data. This also remove the need for developing log systems for the scanned data within the project.

Another tool provided by ROS beyond rosbag is rviz [12]. Rviz provides general purpose, three-dimensional visualization tools for common sensor data types as laser scans and three-dimensional point clouds. By simply subscribing to a topic, it provides seamless visualization of published data in a common coordinate frame.

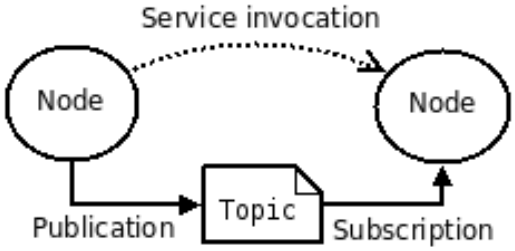


Figure 3.4: ROS nodes with publication and subscription concept. Service invocation feature not present in this implementation[13].

3.3.2 Point Cloud Library

In order to represent and process the 3D scanned data, Point Cloud Library (PCL)[14] is used. PCL is described as a free, fully templated, comprehensive C++ library for n-D Point Clouds and 3D geometry processing and is fully integrated with ROS. This project utilizes PCL for 3D (*x, y ,z*) point representations, 4D (*x, y, z, (r, g, b, a)*) point representations, as well as the *Point Cloud* data structure itself. All of which are supported data types in the ROS environment.

Figure 3.5: Point Cloud Library (PCL)[14].

From an algorithmic perspective, PCL incorporate a multitude of 3D processing algorithms that operate on point cloud data, including filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration, etc. For the purpose of this project, filtering and segmentation algorithms are of interest and to some extent the surface reconstruction algorithms. PCL is presented as efficient and designed with performance in mind. The underlying data structures in PCL make heavy use of Streaming SIMD Extensions (SSE), a processor technology that enables single instruction multiple data optimizations[3]. Most mathematical operations are implemented based on Eigen[15], an open-source template library for linear algebra.

3.3.3 Random Sample Consensus

Random Sample Consensus (RANSAC) [16] is a Sample Consensus (SAC) algorithm and is used to robustly fit model estimations to experimental data using regression. It was originally designed to solve the Location Determination Problem (LDP). LDP states that given an image representing a set of landmarks whose locations are known, the point in space from which the image was obtained should be determined.

4 Implementation

The Implementation chapter describes construction of the hardware platform and system controller, followed by the ROS based software architecture and functions developed.

4.1 3D-Scanner Platform

The implementation of the scanner platform is described starting with the hardware related design, rotation and yaw/pitch resolutions. Finally the implementation of the Arduino that binds the platform together is described.

4.1.1 Platform Construction

In the construction of the scanner platform the specified hardware components were used and 3D-printed models designed in CAD software were used for creating the structures. The platform itself consists of three parts. It has a three-legged base split into a two-legged base supporting the double ball bearings and a third leg which mounts on a circular mount of the bottom ball bearing, containing a supporting backbone structure for the DC-motor and Arduino. The DC motor is equipped with a drive gear, and is aligned with the centric pole which in turn is equipped with a driven gear. These form a two-gear train which facilitates rotational transfer from the DC motor to the pole. The centric pole is mounted with two ball bearings attached to the platform.

There was a synergy in the parts used which allowed the centric pole to rest on the driven gear, which in turn was mounted on the bottom ball bearing. Beneath the ball bearing a slip ring was placed and thus, cables were drawn from the slip ring through the ball bearing, driven gear and the hollow centric pole to the top mount containing the servo motor and LiDAR. Originally, the driven gear would not fit into this position but required only some small modifications with a lathe machinery to fit. Figure 4.1 shows the 3D-printed structure to the left and the rotational components configuration to the right. Image of the prototype can be found in Appendix A.

Figure 4.1: Left: CAD model of the 3D-printed scanner platform. Right: Principal layout of the rotational components.

4.1.2 Yaw Resolution

The yaw resolution is dependent on the encoder attached to the DC-motor and the gear transmission. The specifications of the gears used are listed in table 4.1.

Attribute	**Drive gear**	**Driven gear**
Module	0.8	0.8
No. of Teeth	15	50
Pitch Diameter [mm]	12	40

Table 4.1: Specifications of gears used in the two-gear train.

The encoder measures 3 points per revolution (ppr) and the DC-motor gear ratio is 42.875: 1. Thus, the yaw resolution is calculated the following, taking the two-gear train ratio calculated as described in (3.13)

$$\frac{15}{50} = 0.3 \tag{4.1}$$

encoder ppr and the two gear ratios corresponds to a yaw resolution of

$$\frac{Degrees\ per\ yaw\ revolution}{DC-motor\ gear\ ratio \times Encoder\ resolution} \times two-gear\ train\ ratio, \tag{4.2}$$

and yields resolution as degrees per count (dpc), that is the angular difference between two measurements of attached encoder on the DC-motor.

Calculating (4.2) with values gives a resolution of

$$\frac{360}{42.875 \times 3} \times 0.3 = 0.839... \approx 0.84^\circ\ per\ count. \tag{4.3}$$

The maximum available resolution is given by the minimum of the encoder resolution and the LiDAR measurement rate given the rotational speed of the yaw-axis. With the maximum recorded speed of 28 RPM, the frequency capability of the LiDAR at 50 Hz is calculated as converting RPM to Hz as

$$\frac{RPM}{60} = \frac{28}{60} = 0.466...\ Hz, \tag{4.4}$$

which corresponds to

$$\frac{1}{0.466} \approx 2.146\ seconds\ per\ cycle. \tag{4.5}$$

With the LiDAR at 50 Hz this is equal to

$$50 \times 2.146 = 107.3\ measurements\ per\ cycle, \tag{4.6}$$

and gives a resolution of

$$\frac{360}{107.3} = 3.355^\circ. \tag{4.7}$$

This means that the LiDAR measurement frequency is a limiting factor of the yaw resolution at 28 RPM. Further, 20 RPM and 12 RPM speed settings were implemented and calculations are summarized in Table 4.2.

Setting	RPM	Hz	Seconds per cycle	Measurements per cycle	Resolution
res 0	12	0.200	5.000	250.000	1.440°
res 1	20	0.333	3.003	150.150	2.398°
res 2	28	0.466	2.146	107.300	3.355°

Table 4.2: LiDAR dependent yaw resolution for different RPM.

Ergo, the highest resolution provided by the LiDAR is 1.44° at 12 RPM which is higher than the 0.84° provided by the encoder. The LiDAR is however able to perform at 100 Hz at the expense of reliability, but the resolutions provided at 50 Hz was deemed sufficient. When multiple cycles are performed, the yaw angle is not reset but instead carries over to the new cycle.

4.1.3 Pitch Resolution

The pitch resolution is limited only by the resolution of the servo motor, which is capable of a resolution of 1°. Unlike the horizontal field of view which is 360 degrees, the vertical field of view can be chosen between 0° and 180° The chosen vertical angular resolution and the vertical field of view will however affect the scan time. A common setting used in this thesis was a field of view corresponding to 81°, starting at a 42° angle with a resolution of 3°. At 28 RPM, shown in (4.8), this corresponds to a scan time of the cycle time multiplied with the number of pitch increments and gives a scan time of

$$2.184 \times \frac{81}{3} = 58.968\,\text{s}. \tag{4.8}$$

When multiple cycles are performed, the starting angle is shifted as

$$Pitch\ startingangle + current\ cycle \bmod total\ cycles \tag{4.9}$$

for each cycle, with the starting cycle defined as cycle 0.

4.1.4 Arduino Controller

A Arduino controller is designed to control the LiDAR sensor, DC-motor, encoder and servo-motor using the LIDARLite Adafruit Motorshield and Servo libraries. The controller communicates via USB and contains functionality to receive start, stop and status commands. Apart from the Initialization routine, it can be segmented into the interrupt routine and the measurement routine.

In the initialization routine, pins 2 and 3 are configured as input pullup to handle interrupt from the encoder on a rising edge. Further, the serial port, motor shield, servo motor, DC-motor and LiDAR are configured. Communication with the servo motor is done through software PWM, the DC-motor through PWM and the LiDAR through I^2C. All variables and interrupts are initialized and a state variable is set to running, which starts the motion of the motors.

The interrupt routine is based on the interrupts caused by the encoder registering rotation. The signal interface and reading is based on the post *Improved Arduino Rotary Encoder Reading*[17]. Once an interrupt is read, the yaw angle is updated according to the value calculated in (4.3), and a flag for new measurements is set to true. If the incremented yaw value is greater than 360°, a cycle is complete and the pitch angle resets, or set according to (4.9).

The measurement routine is invoked in a main loop given that flag for new measurements is true. The yaw and pitch angle is read, followed by a measurement from the LiDAR. The yaw and pitch are measured in degrees and converted to radian units before sending the measured data as a string on the serial port and the flag for new measurement is set to false. The routine then proceeds to the same step as if the flag for new measurements is false, checking for incoming data on the serial port is and acting on any commands. The controller is presented as a flow chart in Figure 4.2.

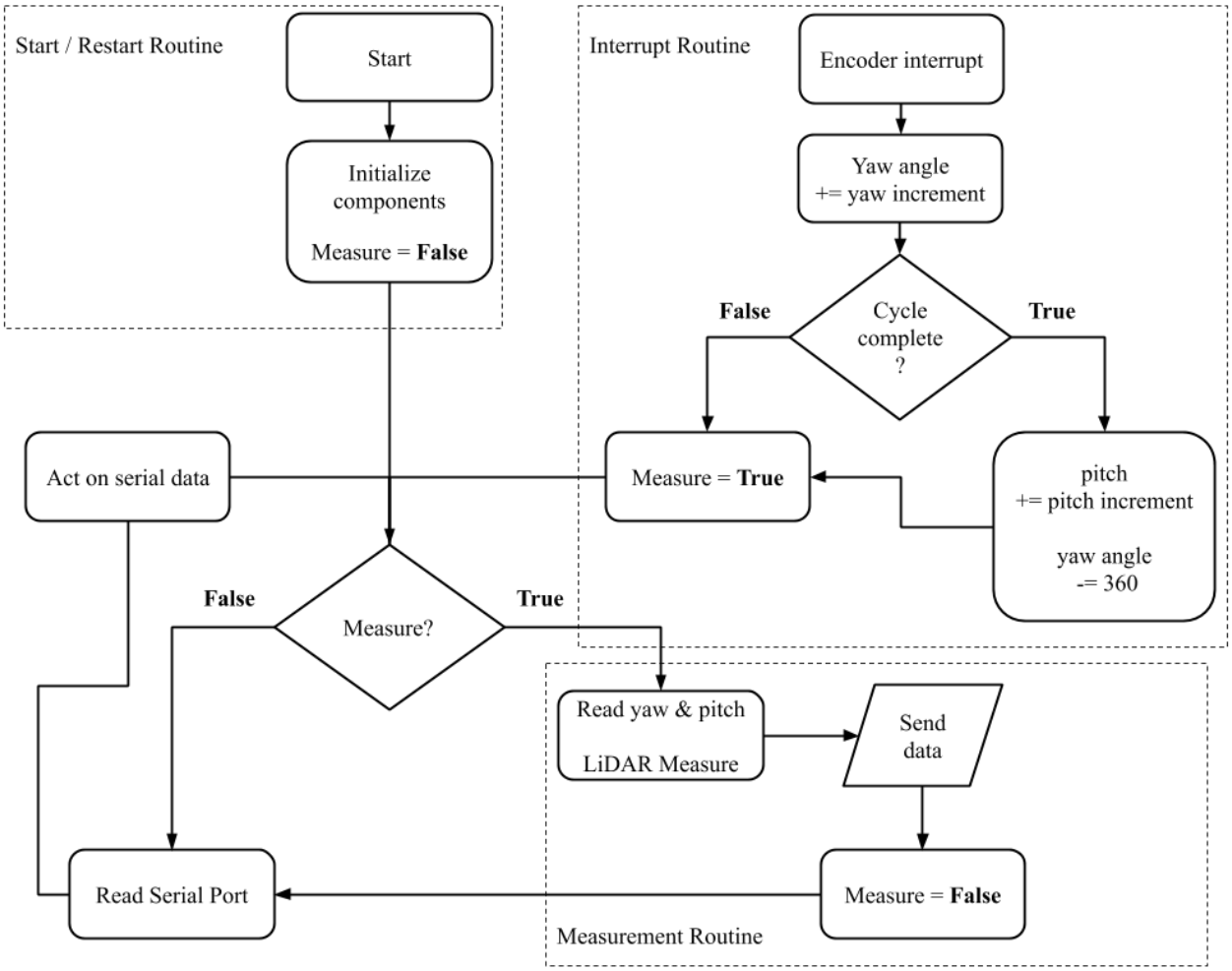


Figure 4.2: Arduino controller flow chart with routine segmentation.

4.2 Point Cloud Processing

In the Point Cloud Processing section implementation of the ROS back-end is described, followed by the data pre-processing implementation and finally the method of conducting automatic dimension calculation is presented.

4.2.1 Software Architecture

On the external processing device, a node-based software architecture is used in compliance with ROS. The implementation contains three nodes: USB Communicator Node, Point Cloud Processing Node and Point Cloud Operations Node. Further, the ROS Rviz tool is utilized as a fourth node and provided an interface to choose which topics to visualize.

The USB Communicator node is responsible for the USB communication with the scanner platform, receiving measurements and sending commands. The Linux serial interface setup is based on 'Linux Serial Ports Using C/C++' [18]. The nodes uses data callbacks for when measurements are received from the scanner platform, and stores them in a temporary point class which handles convertion from spherical coordinates to Cartesian coordinates as described in (3.10), (3.11), (3.12). The Serial Communication Node converts the local point to Point32 Geometry Messages, a ROS compliant message type from PCL. The Point32, now representing a single point in the *(x,y,z)* space, is published per received measurement to the topic /SerialCommunicator/Point.

The *Point Cloud* processing node subscribes to the /SerialCommunicator/Point and acts as a buffer for the published Point32 type, storing them in a *Point Cloud* structure provided by PCL. The Point32 data type contains only *x,y,z* information and this node adds RGBA color information, resulting points with *xyzRGBA* information. RGBA information is however not provided in the LiDAR measurement, making this a pre-defined coloring with regards to visualization. The *Point Cloud* is published on the topic /LidarProcessor/PointCloud per receieved buffered point from the scan.

The final node implemented is the Point Cloud Operations Node, which performs pre-processing of the data with filters further described in 4.2.2 and processing of the data with operations further described in section 4.2.3. This node subscribes to the topic /LidarProcessor/PointCloud. The node publishes several new Point Clouds with pre- processing and processing applied, which is shown in Figure 4.3, along with the other nodes and topics.

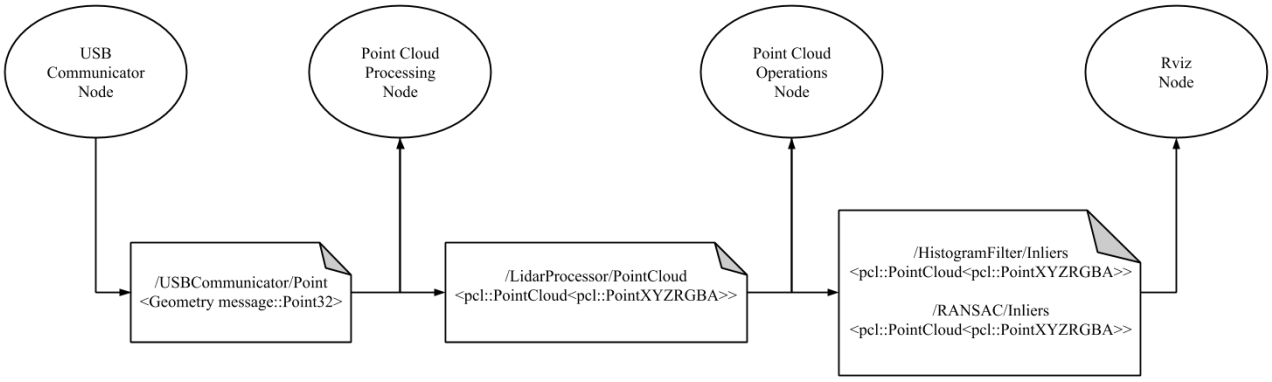


Figure 4.3: Node based architecture used with subscriptions/publications topics.

4.2.2 Data Pre-Processing

The raw scan data contains more information than just walls and unwanted measurements such as reflections. Thus, pre-processing needs to be performed in order for the RANSAC to work properly on the scanned data. The main method of pre-processing is done through applying a histogram filter, described in Algorithm 1.

The idea is to divide the x,y-plane containing all of the scanned points into a grid of bins, where each bin holds the reference to all points contained within the particular bin. The density of points within the scanned data can thus be represented with a histogram, and density is used as a first qualifier in the filtering. Along with a second qualifier of the distance between the lowest and highest point along the z-dimension, filtering is applied to remove information that likely is not related to finding the bounding walls.

Algorithm 1: Histogram filtering algorithm

Data:
pointcloud: A list of coordinates in 3D,
numberOfBins: The histograms number of bins in each dimension,
densityThreshold: A threhold on the number of points a bin has to hold in order to qualify as inliers,
heightDisparityThreshold: A threshold on the lowest z-dimension spread in a single bin in order to qualify as inliers

Result:
listOfInlierIndexes: A list of indexes in the input *pointcloud*

xMin, xMax, yMin, yMax = findMinMaxCoordinates(*pointcloud*);
xResolution = (xMax-xMin)/numberOfBins;
yResolution = (yMax-yMin)/numberOfBins;
listOfBins = list<list>[$numberOfBins^2$];

// Distribute all points in the input pointcloud into the *listOfBins*
for *Iterate over yMin to yMax with steps of yResolution* **do**
  for *Iterate over xMin to xMax with steps of xResolution* **do**
    Create a bin (list) of indexes to all points in the input *pointcloud* within the current x- and y-bounds;
    listOfBins[*binsStoredSoFar++*] = bin;
  end
end

// Iterate over the *listOfBins* and save all indexes stored in bins which pass all quilifier tests into *listOfInlierIndexes*:
for *bin in listOfBins* **do**
  if *bin.size < densityThreshold* **then**
    continue;
  end
  zMin, zMax = findMinMaxCoordinates(pointcloud(bin));
  if *zMax - zMin < heightDisparityThreshold* **then**
    continue;
  end
  listOfInlierIndexes = [*listOfInlierIndexes*, *bin*];
end

After the the histogram filter is applied, an optional downsampling filter is applied. The downsampling, or points reduction, is done through a PCL-provided algorithm called VoxelGrid Filter. The idea is to represent the Point Cloud as a 3D voxel grid, where each voxel essentially represents a small 3D cube. Once the Point Cloud is represented in this way, all points present within a voxel is approximated into a single point based on the centroid. While the approximation could be done using the center point of the voxel. This downsampling filter is very useful in this application, as the inherent measurement error of the LiDAR can cause errors large enough to interfere with plane fitting. This approach yields smoother walls and mitigates some measurement errors.

4.2.3 Point Cloud Operations

Once the Point Cloud has been been pre-processed, the RANSAC algorithm from PCL is used to perform segmentation and find planes. To find the desired planes, two limitations are applied to the RANSAC algorithm. The plane must be parallel to the z axis, thus eliminating any plane parallel to the xy axis like the floor or the roof. Depending on the amount of noise in the scan, planes that span through the noise and other objects could be more eligible than the walls. I.e, a plane spanning from wall to wall along the edge of furniture. To reduce this risk an angle epsilon delta threshold is used, a maximum allowed difference between the model normal and a given parallel axis. Once a wall is found, the inliers who fits the plane coefficients is removed from the Point Cloud while the plane information is kept separate. This is done iteratively until all walls are found. Once four walls and their plane coefficients are found, the corresponding four intersection points is calculated, assuming they are the confining walls of a quadrilateral room.

The intersection point along the z=0 plane for two planes defined by their corresponding plane coefficients as

$$\begin{aligned} Ax + By + Cz &= D \\ \bar{A}x + \bar{B}y + \bar{C}z &= \bar{D} \end{aligned} \tag{4.10}$$

can be derived as

$$\begin{aligned} & z = 0 \\ \Rightarrow\ & Ax + By = D,\ \bar{A}x + \bar{B}y = \bar{D} \\ \Rightarrow\ & y = \frac{D - Ax}{B},\ y = \frac{\bar{D} - \bar{A}x}{\bar{B}} \\ \Rightarrow\ & \frac{D - Ax}{B} = \frac{\bar{D} - \bar{A}x}{\bar{B}} \\ \Rightarrow\ & B\bar{A}x - \bar{B}Ax = B\bar{D} - \bar{B}D \\ \Rightarrow\ & x = \frac{B\bar{D} - \bar{B}D}{B\bar{A} - \bar{B}A}. \end{aligned}$$

Thus, the equations for calculating the intersection points along the z=0 plane becomes

$$\begin{aligned} x &= \frac{B\bar{D} - \bar{B}D}{B\bar{A} - \bar{B}A} \\ y &= \frac{D - Ax}{B} \\ y &= \frac{\bar{D} - \bar{A}x}{\bar{B}}, \end{aligned} \tag{4.11}$$

with two different ways to calculate the y-coordinate depending on if either B-coefficient happens to be 0.

With the corner intersection points derived, an algorithm for the surface area of quadrilaterals is required to estimate the area within the walls. Given the assumption that all inner corners of the quadrilateral representing a room have angles less than 180°, it can be divided along its diagonal as in Figure 4.4.

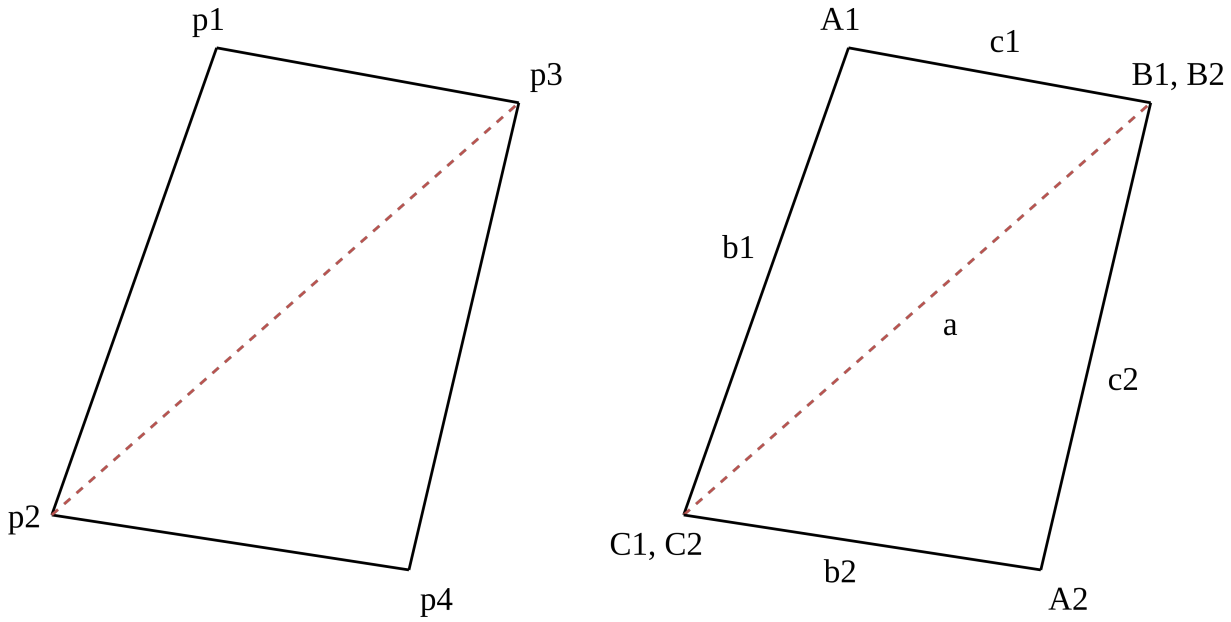


Figure 4.4: Quadrilateral room in top down view and parameterization of triangular split along the diagonal. pi represents a 2D-coordinate, A1, A2, B1, B2, C1 and C2 represents the inner angle of the corners, and a, b1, b2, c1, c2 represents the length of the lines.

Thus, the area of the quadrilaterals can be calculated as the sum of the areas of the two inner triangles as

$$\begin{aligned} A_1 &= \frac{a \times b1 \times Sin(C1)}{2} \\ A_2 &= \frac{a \times b2 \times Sin(C2)}{2} \\ A &= A_1 + A_2. \end{aligned} \tag{4.12}$$

The dimension calculation is implemented through two algorithms. Firstly, the intersections between the planes are found as described in Algorithm 2. The idea is to firstly compute an *areaOfInterest* within which all intersections in terms of corners of the room must lie. This is in order to easier find pairs of planes which create reasonable corner intersections. It is done by iterating all inliers for all planes and finding the minimum and maximum x- and y- coordinates among the points, and adding a margin of 100cm in each dimension.

Further, the *normalAngles* are computed, that is the angle from the x-plane to the normal along the z=0 -plane for each of the *planeCoefficients*. Based on the calculated *normalAngles* alone, *parallelPairs* of planes are calculated. That is 2 sets of 2 planes which have the least differences in *normalAngles*.

Algorithm 2: Quadrilateral Room Corner Intersections

Data:
planeCoefficients: List of coefficients for planes, has to contain at least 4 planes. Each set of coefficients contains the the coefficients A,B,C,D of the plane equation Ax+By+Cz=D,
inlierClouds: Lists of points corresponding to each of the plane coefficients,
parallelPlanesAngleThreshold: A threshold for the angle between two planes for them to be defined as parallel (or opposing)

Result:
xIntersections: A list of 4 x-coordinates representing the corner intersections with indexing as defined in Figure 4.4,
yIntersections: A list of 4 y-coordinates representing the corner intersections with indexing as defined in Figure 4.4.

areaOfInterest = findMinMaxCoordinates(*inlierClouds*) +/- 100cm;

for *coefficients : planeCoefficients* **do**
    $normalAngles = [normalAngles, atan2(coefficients.B, coefficients.A)]$;

parellelPairs[0] = findIndexesOfNormalAnglesWithLeastDifference(*normalAngles*);
parellelPairs[1] = findIndexesOfNormalAnglesWithLeastDifference(*normalAngles(∼parellelPairs[0])*);

if *any(differenceInAngles(parallelPairs, normalAngles) > parallelPlanesAngleThreshold)* **then**
    AbortDimensionEstimation;

// Find the intersections between the two sets of parallel pairs
for *plane1 : parallelPairs[0]* **do**
    for *plane2 : parallelPairs[1]* **do**
        $xIntersection = \frac{plane1.B \times plane2.D \; - \; plane2.B \times plane1.D}{plane1.B \times plan2.A \; - \; plane2.B \times plane1.A}$;
        if *plane1.B != 0* **then**
            $yIntersection = \frac{plane1.D - plane1.A \times xIntersection}{plane1.B}$;
        else if *plane2.B != 0* **then**
            $yIntersection = \frac{plane2.D - plane2.A \times xIntersection}{plane2.B}$;
        else
            AbortDimensionEstimation;
        if *withinBounds(areaOfInterest, xIntersection, yIntersection* **then**
            AbortDimensionEstimation;
        $xIntersections = [xIntersections, xIntersection]$;
        $yIntersections = [yIntersections, yIntersection]$;

Once the intersections are found, the dimension of the room is estimated as described in Algorithm 3, using the surface area calculations as shown in Figure 4.4. Finally, the length of the 4 walls along the z=0 plane is calculated.

Algorithm 3: Dimension Estimation

Data:
xIntersections: A list of 4 x-coordinates representing the corner intersections with indexing as defined in Figure 4.4,
yIntersections: A list of 4 y-coordinates representing the corner intersections with indexing as defined in Figure 4.4

Result:
wallLengths: A list of the estimated length of each of the 4 walls in the room,
surfaceArea: The estimated surface area within the 4 walls over the z=0 -plane defined by *zPlaneValue*

xi = xIntersections[i];
yi = yIntersections[i];

$a = |\sqrt{(x1-x2)^2 + (y1-y2)^2}|$;
$b = |\sqrt{(x1-x3)^2 + (y1-y3)^2}|$;
$C = |atan2((y1-y2),(x1-x2)) - atan2((y3-y2),(x3-x2))|$;

if $C > \pi$ **then**
  $C = 2\pi - C$;

$triangArea1 = a \times b \times sin(C)/2$

$a = |\sqrt{(x4-x2)^2 + (y4-y2)^2}|$;
$b = |\sqrt{(x4-x3)^2 + (y4-y3)^2}|$;
$C = |atan2((y4-y2),(x4-x2)) - atan2((y3-y2),(x3-x2))|$;

if $C > \pi$ **then**
  $C = 2\pi - C$;

$triangArea2 = a \times b \times sin(C)/2$ $surfaceArea = triangArea1 + triangArea2$

$wallLengths[0] = |\sqrt{(x1-x2)^2 + (y1-y2)^2}|$;
$wallLengths[1] = |\sqrt{(x3-x4)^2 + (y3-y4)^2}|$;
$wallLengths[2] = |\sqrt{(x1-x3)^2 + (y1-y3)^2}|$;
$wallLengths[3] = |\sqrt{(x2-x4)^2 + (y2-y4)^2}|$;

5 Results

The results chapter introduces the two test cases used to present the results with visualization, followed by an error estimation indication histogram. Further, the histogram filter results are shown and finally, the results of the dimension estimation for the two cases is presented.

5.1 Configurations

Two configurations which is made up of two similar rooms with mirrored layouts is used to present the results. They contain alike furniture and have the same dimensions. Unless stated otherwise, the 20 RPM (res 1) resolution setting is used. 28 RPM (res 2) resolution settings can be found in Appendix B and C.

5.1.1 Case 1

Case 1 represents a single room and is visualized in Figure 5.1, with one open exit (top wall, left side), as well as two closed exits (top wall, right side; right wall, top side). The major objects present in this scan is a sofa (middle) and TV (bottom) as well as a bed (top right). There are 3 windows (right wall) and a mirror (top wall, left side, right of exit). The 3D scan generated a point cloud with 18 560 points in total.

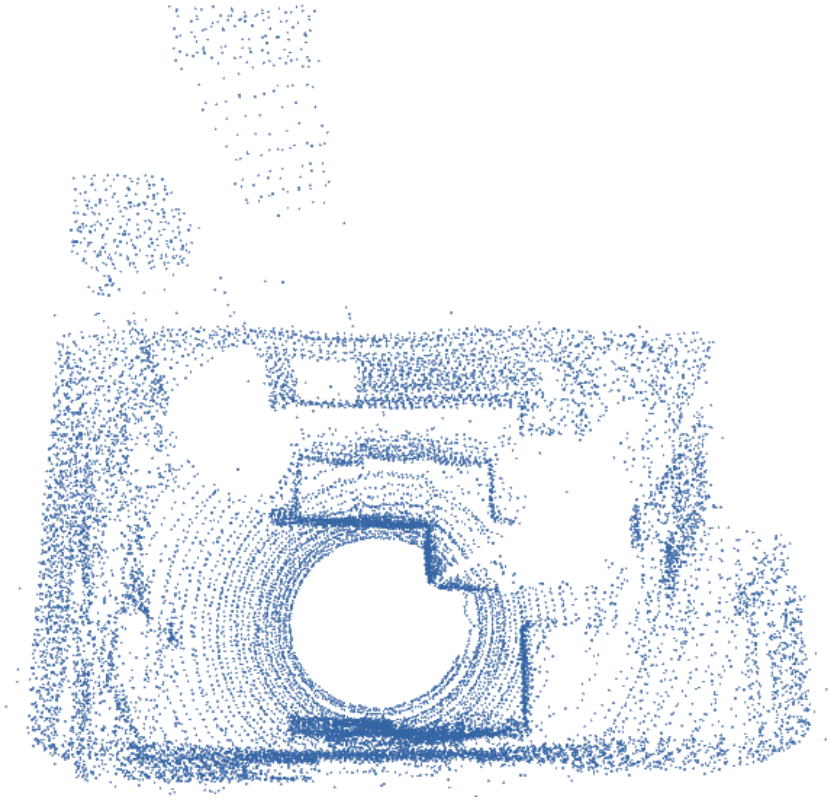


Figure 5.1: Point Cloud visualization of the 3D scan performed in the case 1 setting.

5.1.2 Case 2

Case 2 represents a single room and is visualized in Figure 5.1, with two open exits (bottom wall, left and right side), as well as a closed exit (right wall, bottom side). The major objects present in this scan is a sofa (middle) and TV (bottom) as well as a bed (left side). There are 3 windows (right wall). The 3D scan generated a point cloud with 18 382 points in total.

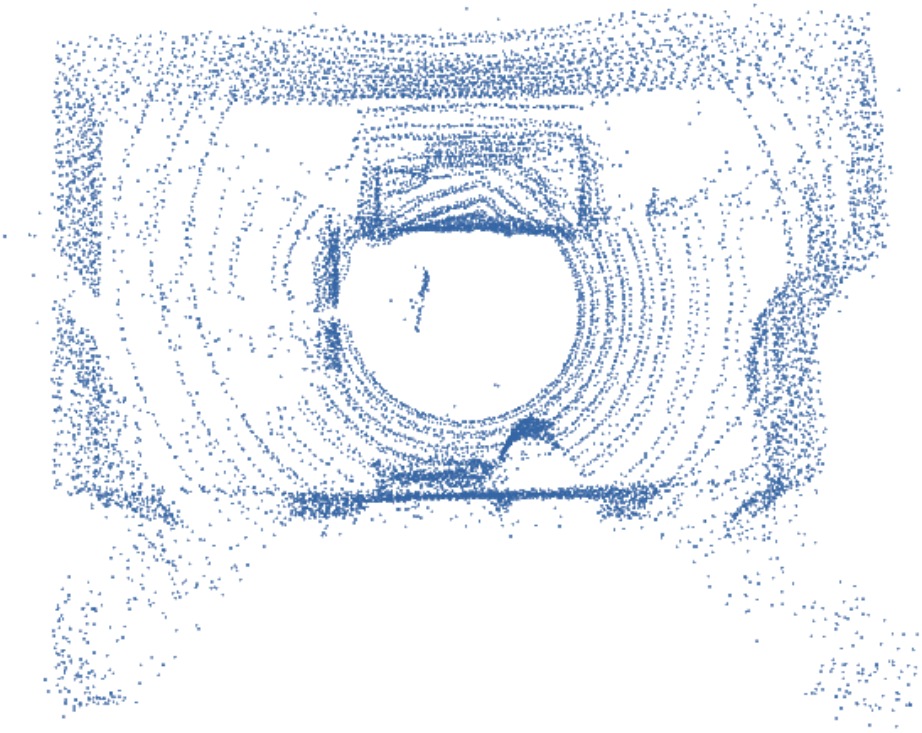


Figure 5.2: Point Cloud visualization of the 3D scan performed in the case 2 setting.

5.2 Measurement Errors

In order to examine the measurement errors of the 3D scan, a histogram is shown in Figure 5.3 to illustrate absolute distance errors relative the planar equation of the greatest length wall found with RANSAC. It is thus not relative the true wall and is conducted with all the unprocessed points obtained from case 1 scan.

The left graph in Figure 5.3 shows all points with an absolute distance of less than 50 cm, with the wall seen in the span of ≈ 0 cm to 6 cm and the TV from ≈ 25 cm to 30 cm.

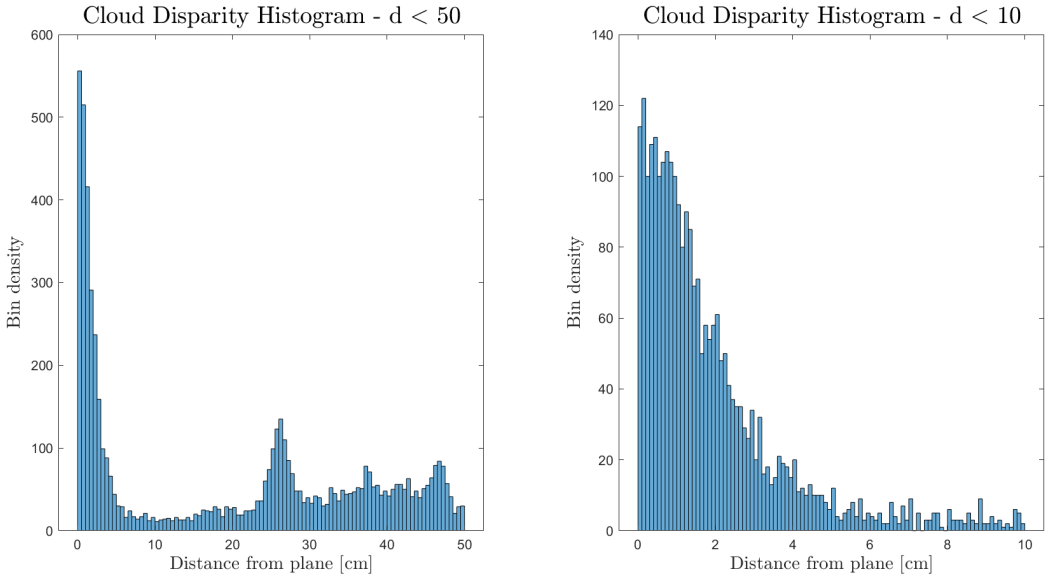


Figure 5.3: Density of points relative the absolute distance to the wall of highest length, for points of distance less than 50 cm and 10 cm.

The right graph in Figure 5.3 shows all points with an absolute distance of less than 10 cm, which gives an overall indication of the disparity of measurements. Most of the scanned points have an absolute distance of less than 5 cm to the estimated plane at a measuring distance of ≈ 169 cm when performing a scan with a rotational speed of 20 RPM.

5.3 Pre-Processing

In the two cases presented in this chapter, only the histogram filter is applied to the point cloud. This filter was sufficient for RANSAC to work properly in these cases. There was however cases where this filter would not suffice, and the downsampling filter would then be applied to reduce variation in measurement data and allow consensus in planar estimations.

Figure 5.4 shows a histogram of the original point cloud of case 1 and the filtered point cloud of case 1. The filter successfully removes most of the unrelated points present within the room as well as points scanned which lies outside the open exits, as well as points reflected by the mirror. The points representing the TV is however not filtered out in case 1. The filter removes 12 305 points, leaving a total of 6 255 points in this case.

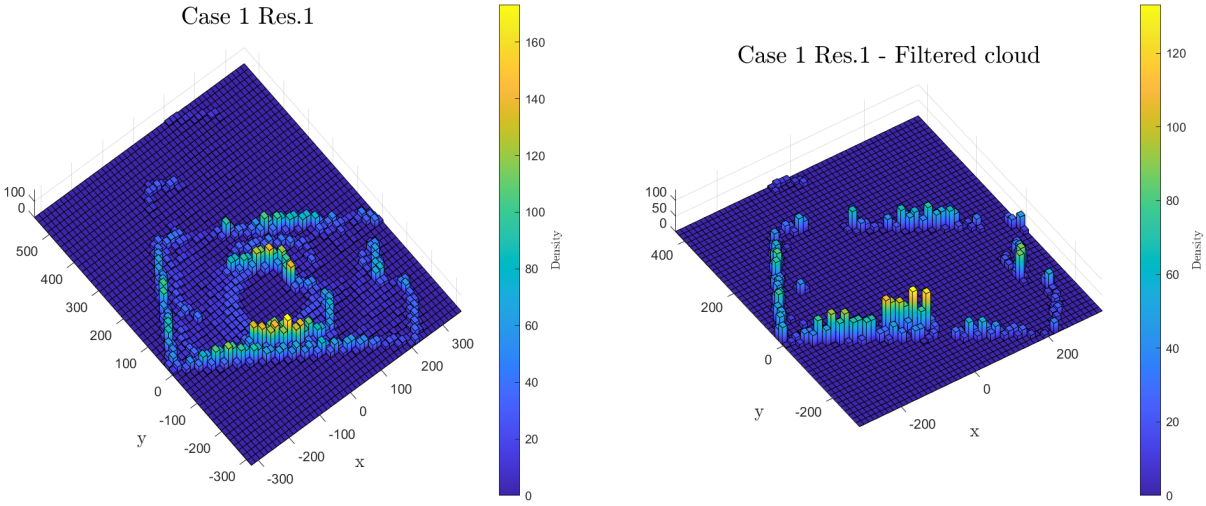


Figure 5.4: Pre and post filtering on the case 1 point cloud represented as a histogram.

Case 2 is shown similar to case 1 in Figure 5.5. The filter successfully removes almost all of the unrelated points present within the room as well as points scanned which lies outside the open exits. The points representing the TV is is unlike case 1, filtered out in case 2. The filter removes 13 728 points, leaving a total of 4 654 points in this case.

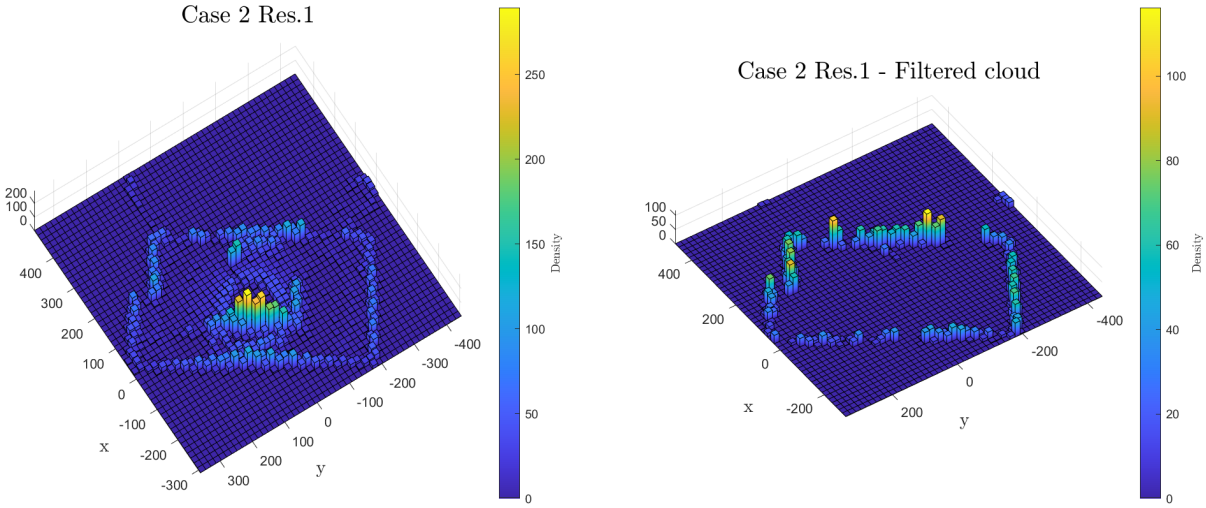


Figure 5.5: Pre and post filtering on the case 2 point cloud represented as a histogram.

5.4 Area Estimation

The true measurements for the room is shown in Figure 5.6. A_1 represents the smaller quadrilateral area and A_2 the larger quadrilateral area. In both cases the smaller area A_1 was measured. All corners are assumed to be 90°.

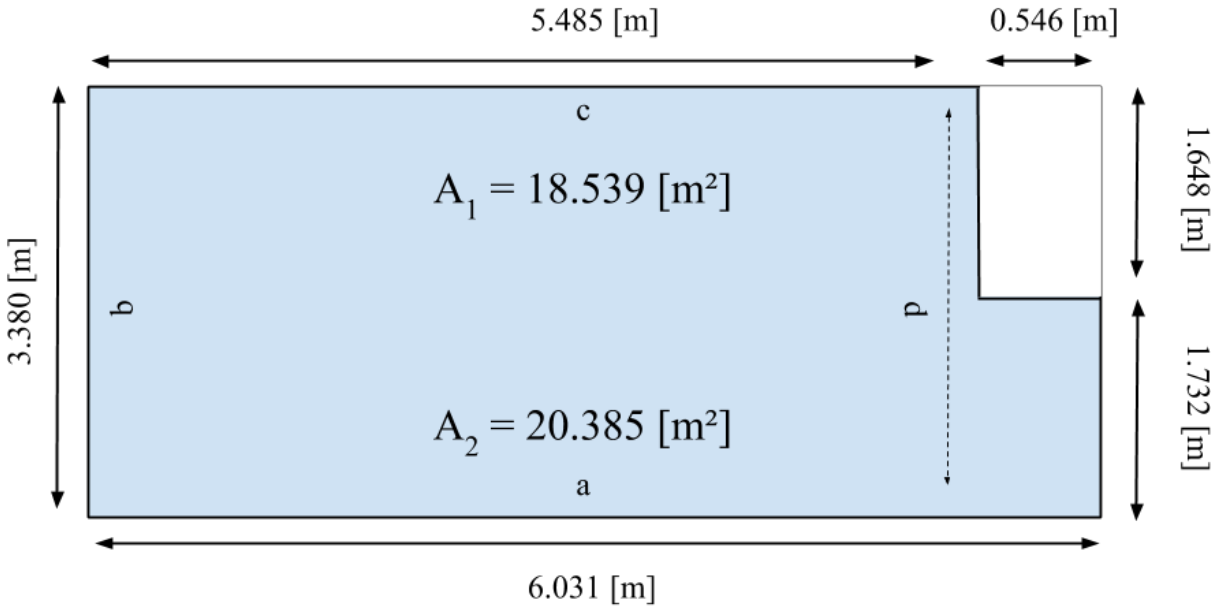


Figure 5.6: Measured dimensions of the quadrilateral rooms in case 1 and mirrored to case 2, and the relative variables a, b, c, d which represents the approximated dimensions of the cases.

The algorithm yielded the following results for case 1

$$a_{\text{case1}} = 5.465\,\text{m} \tag{5.1}$$
$$b_{\text{case1}} = 3.470\,\text{m} \tag{5.2}$$
$$c_{\text{case1}} = 5.487\,\text{m} \tag{5.3}$$
$$d_{\text{case1}} = 3.482\,\text{m} \tag{5.4}$$
$$A_{\text{case1}} = 21.075\,\text{m}^2 \tag{5.5}$$

and the following for case 2

$$a_{\text{case2}} = 5.634\,\text{m} \tag{5.6}$$
$$b_{\text{case2}} = 3.478\,\text{m} \tag{5.7}$$
$$c_{\text{case2}} = 5.617\,\text{m} \tag{5.8}$$
$$d_{\text{case2}} = 3.519\,\text{m} \tag{5.9}$$
$$A_{\text{case2}} = 21.823\,\text{m}^2 \tag{5.10}$$

6 Discussion

In terms of resolution achieved the results were very satisfying. The platform generated *Point Clouds* with good resolution relative state-of-the-art rotational LiDARs, in particular the pitch resolution was very high in the scans. From a visual point of view the visualized *Point Clouds* from the scans contain accurate information of the room and objects present. Scanning time was long relative state-of-the-art rotational LiDARs. It is however possible to further increase the rotational speed while keeping a high resolution, as tests performed with lower resolution and higher speed showed a low loss of information and increased errors. It could also be an option to increase measurement frequency of the LiDAR sensor to further increase scan speed.

Rotating the scanner platform likely added further measurement errors on top of the inherent sensor errors. Measurement performance tested in[2] yielded a measurement standard deviation of 2.12 cm at 150 cm and 3.51 cm at 300 cm. With the results in this thesis from a range of 200 cm, as shown in Figure 5.3, the scanner platform seems to double the average measurement error and thus has a higher standard deviation. Overall there is a clear distinction between different objects present with some level of noise in between. This indicates that the measurements contains mostly precision errors of surfaces and does well in distinction between closely located objects in general. In a macro sense this has a low effect on the 3D reconstruction representing the room.

The histogram filter did well pre-processing the data. In general the filter managed to remove most of the points not related to walls and thus ensured accurate RANSAC planar coefficients estimations. In some cases a large TV would pose problems, while a smaller TV did not cause any issues. This method of filtering has limited applications as the qualifiers for removal is based on density and disparity, as the TV example illustrates. Roofs were mostly excluded from the scans which further increased the efficiency of the filter. Extensions could be made to perform this type of filtering on spherical-coordinate level which could allow for further efficiency and synergy with the nature of the rotational 3D scan.

Dimension estimation was successful, though the dimensions were over-estimated in the cases. While restricted to only quadrilateral surfaces, more advanced techniques could be applied to estimate more complex shapes. The assumption of 90° corner angles is likely false in the ground truth of the test cases and thus affects the room dimensions. This could most likely be further improved, but within the scope only a 'proof-of-concept' was performed.

The resulting quality of the scan and the operations is likely utilizable in other applications. It does a good job finding the walls of the room and can thus, for example, be used in applications such as robotics where it can provide boundaries for autonomy. Further, using the walls for triangulation of position within the room would likely be possible. Currently the speed of the platform poses a mobility problem in such applications, as registration of points while moving is not possible.

6.1 Environmental Aspects

Performing 3D scans with the resulting platform and system is not very energy efficient. It does lack mobility and in the test cases, an excess of rotation relative the possible number of scans is used. Using this system to measure the area of a room is always inferior to, for example, an hand held LiDAR measurement device from an energy efficiency point of view. While the end results are comparable, the data generated is however not.

6.2 Critical Discussion

Improvements to the scanner platform would yield great improvements on the results. The decision to rotate the sensor itself did cause some minor issues with scanning, while the DC motor and encoder made repeatability of scans difficult. The option to rotate a mirror would possibly generate better results in this case as noise and errors caused by rotation likely has a greater impact on a rotating sensor. Platform quality was overall a limiting factor.

Having absolute positions for the yaw angle would also increase the control of the scan and increase functionality. The full reset of the pitch angle was a bad options and would cause interrupts in the scan. Instead of resetting, a decrement of the pitch angle could be a better solution.

Additional filtering would be necessary in order for the system to work in more complex environments. The developed filter is very restricted to the specific scenario in this thesis and thus would need to be replaced for more general appliances. Performing sequential filters in stages could further improve usability and effectiveness of the application.

7 Conclusions

Overall the performance of the 3D scanner was sufficient for the objective and automating the dimension calculations was successful. The constructed scanner system produced high resolution scans with accurate 3D reconstructions, albeit with a scan time around 60 seconds.

The scanner provided very high pitch resolution along with a decent yaw resolution. While the platform doubled the measurement error of the sensor, it did not impact the ability to perform RANSAC planar estimations. The generated *Point Clouds* were of high quality with clear distinction between objects.

It does show signs that 3D scanning applications with this type of LiDAR system can yield good results. While it successfully estimated the area in the test cases, it did over-estimate the dimensions and was only applied on surface estimation on simplified quadrilateral models of closed environments.

References

[1] NOAA. (Jun. 2018). What is lidar?, [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html.

[2] I. Maulana, A. Rusdinar, and R. Priramadhi, Lidar application for mapping and robot navigation on closed environment, *Journal of Measurements, Electronics, Communications, and Systems* **4** Jun. 2018, 20, Jun. 2018. DOI: 10.25124/jmecs.v4i1.1696.

[3] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[4] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, and et al., "The digital michelangelo project: 3d scanning of large statues", *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00, USA: ACM Press/Addison-Wesley Publishing Co., 2000, 131–144, ISBN: 1581132085. DOI: 10.1145/344779.344849. [Online]. Available: https://doi.org/10.1145/344779.344849.

[5] W. D. Van Driel, O. Pyper, and C. Schumann, *Sensor Systems Simulations*. Springer, 2020.

[6] Garmin. (Aug. 2016). Lidar lite v3 operation manual and technical specifications., [Online]. Available: http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf.

[7] DYNAPAR. (2020). Hall effect encoder overview, [Online]. Available: https://www.dynapar.com/technology/encoder_basics/hall_effect_encoders/.

[8] Atmel. (2015). 8-bit avr microcontroller with 32k bytes in-system programmable flash, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[9] Toshiba. (2020). Tb6612fng brush motor driver ic, [Online]. Available: https://toshiba.semicon-storage.com/us/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html.

[10] ROS. (2020). Robot operating system (ros), [Online]. Available: http://ros.org/.

[11] ——, (2015). Rosbag, [Online]. Available: http://wiki.ros.org/rosbag.

[12] ——, (2018). Rviz, [Online]. Available: http://wiki.ros.org/rviz.

[13] ——, (2014). Ros basic concepts, [Online]. Available: http://wiki.ros.org/ROS/Concepts.

[14] PCL. (2014). Point cloud library (pcl), [Online]. Available: http://pointclouds.org/.

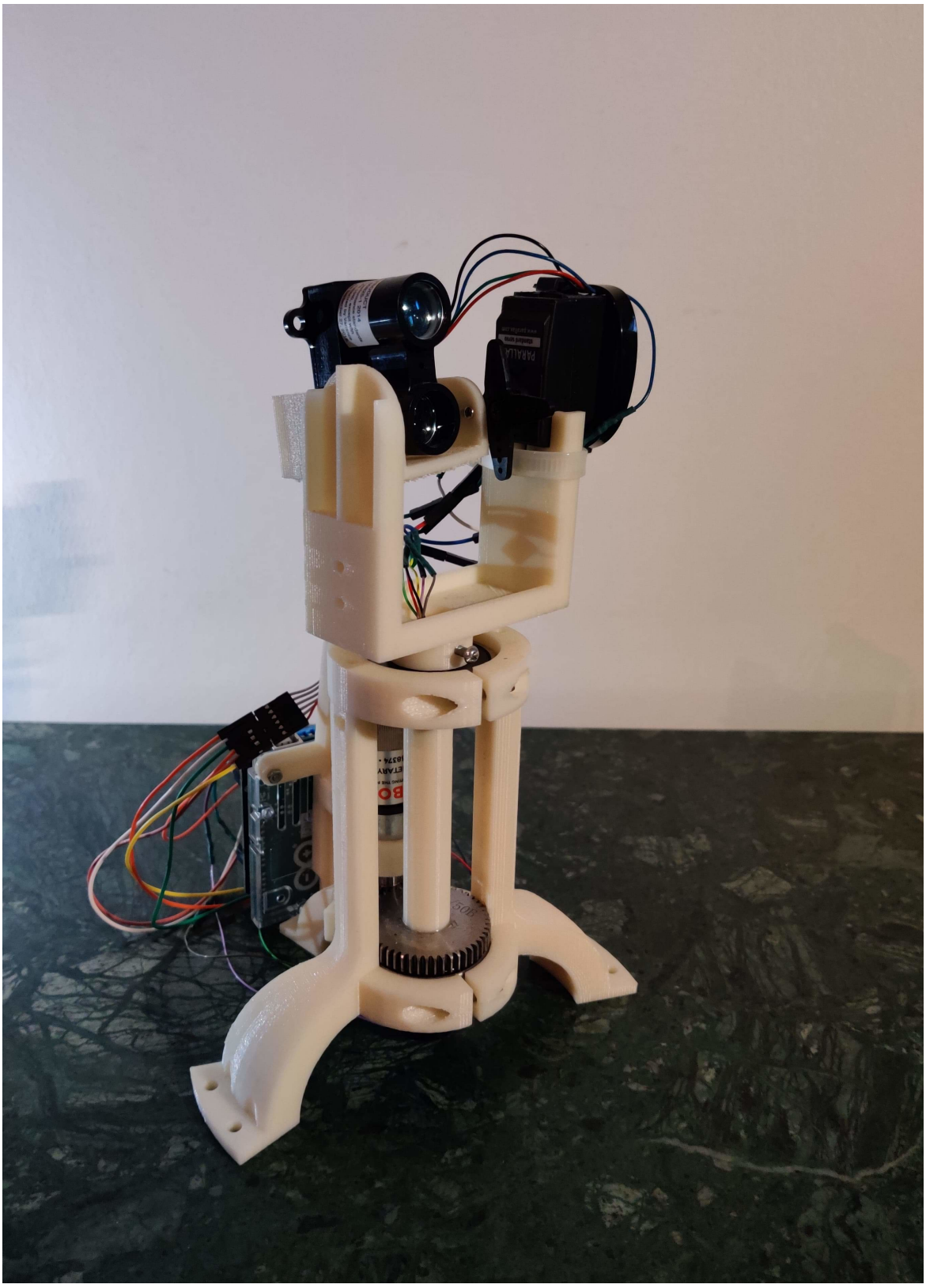
[15] Eigen. (2019). Eigen c++ library, [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page.

[16] M. A. Fischler and R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, *Communications of the ACM* **24**, no. 6 1981, 381–395, 1981.

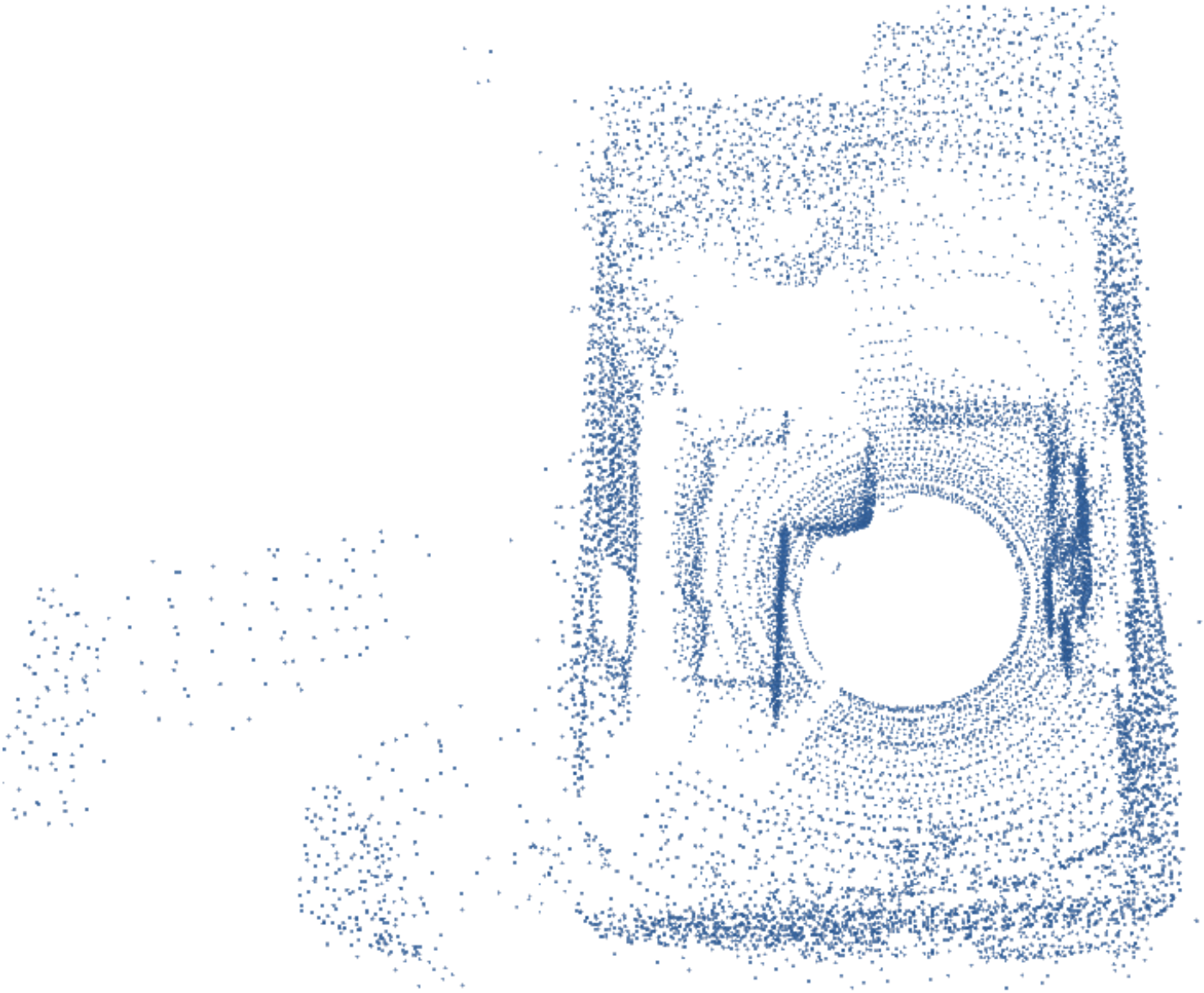
[17] (2019). Improved arduino rotary encoder reading, [Online]. Available: https://www.instructables.com/id/Improved-Arduino-Rotary-Encoder-Reading/.

[18] (2018). Linux serial ports using c/c++, [Online]. Available: https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/.

A Scanner Platform Prototype



B Case 1 - Resolution 2 (Low) Point Cloud



C Case 2 - Resolution 2 (Low) Point Cloud