UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Software Engineering Curriculum

Raigo Kõvask

State of the art VR Driving Simulation for Physical Test Car Using LiDAR for Mapping the Surrounding Environment

Master's Thesis (30 ECTS)

Supervisor(s): Ulrich Norbisrath, PhD

Tartu 2022

State of the art VR Driving Simulation for Physical Test Car Using LiDAR for Mapping the Surrounding Environment

Abstract:
The LiDAR (Light detection and ranging) sensor is a sensor that calculates the distance from the point based on the light reflected from the given surface. It creates a 3D representation of the surveyed environment. There are a lot of different usages in the automotive industry that are being discussed and developed in the autonomous vehicles market. As the usage of the LiDaAR sensors increases in different areas, the overall price of the LiDAR decreases, making it more accessible. As it gets more accessible, we can find more uses for this. One of the ideas was to create a remotely controllable vehicle that could reconstruct the surrounding environment. This thesis focuses on using the cost-effective time-of-flight LiDAR sensor Cygbot CygLiDAR D1 [31] with a controllable donkey car to create a virtual world based on the scanned points. A literature survey was recently conducted to find out what methods to use to localize and map the donkey car and how to reconstruct the environment in the virtual world. During the thesis, a working solution is built as a single working pipeline.

Keywords:
LiDAR, SLAM, modelling, localization

CERCS:
P170 - Computer science, numerical analysis, systems, control

VR-sõidusimulatsioon füüsilise katseauto jaoks, kasutades LiDAR-it ümbritseva keskkonna kaardistamiseks

Lühikokkuvõte:

LiDAR (Light detection and rangering) andur on seade, mis arvutab antud pinnalt peegelduva valguse põhjal kauguse punktist. Selle abil on võimalik luua uuritavast keskkonnast 3D esitus. Autotööstuses on palju erinevaid kasutusviise, mida arutatakse ja arendatakse autonoomsete sõidukite turul. See võib olla põhjus, miks LiDAR-andurite hind langeb, kuna anduri kasutamine suureneb mitmes valdkonnas. Kui see muutub kättesaadavamaks, leiame sellele rohkem kasutusvõimalusi. Üks ideedest oli luua kaugjuhitav sõiduk, mis suudaks rekonstrueerida ümbritsevat keskkonda. See lõputöö keskendub time-of-flight LiDAR anduri Cygbot CygLiDAR D1 [31] kasutamisele koos juhitava Donkey Car-iga, et luua skaneeritud punktide põhjal virtuaalne maailm. Enne praktilise töö alustamist viidi läbi kirjandusuuringu, et selgitada välja, milliseid meetodeid kasutada Donkey Car-i lokaliseerimiseks ja kaardistamiseks ning kuidas virtuaalmaailmas keskkonda rekonstrueerida. Lõputöö käigus ehitatakse töötav lahendus ühtse töötava torustikuna.

Võtmesõnad:
LiDAR, SLAM, modelleerimine, lokaliseerimine

CERCS:
P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction **5**

2 Goals **7**

3 Background and Related Work **9**
3.1 Related work 9
3.2 SLAM Algorithms 9
3.3 Hectorslam 10
3.4 Point Cloud to Model Algorithms 10
3.5 LiDAR 13
3.6 Donkey Car 14
3.7 Virtual World Generation 14

4 Methods **16**
4.1 LiDAR setup 17
4.2 Donkey Car 20
4.3 VR 25
4.3.1 Marching Cubes Algorithm 25
4.3.2 BPA algorithm 25

5 Implementation **28**
5.1 Architecture 31

6 Discussion **33**
6.1 Modelling Scan From the Car with Static Position 37
6.2 Problems 41
6.3 Scan From the Moving Car 42

7 Conclusion **45**

References **50**

Appendix **51**
I. Glossary 51
II. Licence 53

1 Introduction

LiDAR technology is an increasing technology in the recent automotive industry. The usages vary from doing accurate scans of the objects to analyzing the surrounding environment for autonomous cars. What LiDAR essentially is, is a sonar that uses pulsed laser waves to map the surrounding environment by calculating distances to each scan point. The LiDAR work range is usually from a few centimeters up to 50-100 meters, which gives it a lot of functionality. It is used by some autonomous vehicles to navigate environments in real-time. Its advantages include impressively accurate depth perception, allowing LiDAR to know the distance to an object from a few centimeters up to 60 meters away. It's also highly suitable for 3D mapping.

The LiDAR sensors used to be quite expensive. For example, google first self-driving car was built back in 2012. The sensor cost around USD 70,000. Nowadays, the cost of the sensors used for a similar purpose has dropped below USD 1000. Cost drop has mostly resulted from the larger usage of the LiDAR sensor [10].

These sensors are sold as separate sensors, but Apple has taken them to a new level by integrating them into their iPhone and IPad series. The newest devices (iPhone 13 Pro series and iPad Pro models) already have LiDAR sensors [4], which users can use to scan the world with ease. Having apple devices integrating them means that the scanners are already available widely to the public. The phone cost may vary, starting from USD 800. These sensors are not as accurate as the sensors built by companies specializing in sensor creation, such as the Velodyne, but they give more or less the result you would expect from them. Apple phones already have applications available that allow the users to scan the world and render the result of the collected point cloud, such as the 3D Scanner App, Polycam, Scaniverse [33].

Inspiration for the thesis came from the Autonomous driving lab(ADL) project, which was established in the year 2019 [2]. This project is managed at the University of Tartu. It aims to research the technological solutions for autonomous driving and develop new technologies. The autonomous driving lab at the University of Tartu uses mainly video streams in their development. However, the primary car also has Lidar data that can be used to potentially achieve more accurate information. Therefore it would be to potentially achieve more accurate information to integrate Lidar data in the process of controlling the vehicle.

This thesis goes a bit more into the technologies behind the scanner, and what is the whole pipeline from gathering the point clouds to recreating the meshes in the engine. One of the goals is to create a digital twin of the real world in the eyes of the user via VR set by only using the LiDAR sensor to do it. So the solution could give cost-effective inputs for the people who would like to create their custom scanners. The second goal would be to have a remotely controllable unit that can generate the world without the user directly being in close vicinity. So the goal is to have a remotely controlled car in the real world that users can control via VR.

It will be achieved with a remotely controllable donkey car and a VR headset. A remotely controllable car sends the user the input with a LiDAR scanner to the VR headset, and the VR headset displays the models in the virtual world. Before starting with this thesis, a literature review of the existing methods is necessary to start working on this thesis. I will use the research results to select the algorithms to be used in this thesis. A summary of that paper will be listed under the related work session.

In this work, I will test out the following steps of the solution that I found during the research phase:

1. Set up and define ROS nodes so that the LiDAR scanner works correctly and sends points to ROS.

2. Set up the algorithm that processes the cloud and outputs the correct position into a certain ROS topic.

3. Set up the engine that takes in the processed position and LiDAR scanned points and constructs the environment.

In this thesis, the LiDAR-only solution is used to calculate the position of the donkey car. The liDAR-only solution requires only a single sensor input to calculate the position, reducing the number of components needed for the car to work.

2 Goals

This thesis aims to create a teleoperation robot via VR headset using the LiDAR sensor and doing it in real-time. Each time a new area is scanned with the LiDAR sensor, it would be modeled close to real-time (1-2 seconds). To reach the following goal, I first researched the field a bit. After that, I did a literature review on possible topics that would become a problematic part of the thesis. The scope of this thesis would be to attempt to implement the chosen algorithm and to see if it is achievable. As my previous experience in the field is quite limited, there is a chance of moving in the wrong direction when attempting to solve the set problem. Based on previous research, the following requirements were created:

1. The first requirement with the LiDAR point data that needs to be solved is the transformation of the point cloud. The core of the problem tries to answer the question of where in the room the scan was taken? Which leads to the following question: This leads to the following question: What possible algorithms could be used in the localization and mapping process?

2. The second requirement is regarding the next step in the pipeline, which is modeling the point cloud. This is right after the correct transformation has been found for the point cloud. Based on this, I would try to find the possible algorithms that could be used in the modeling process.

3. The third requirement is to build up the whole pipeline from getting the point cloud to having this modeled in the virtual world. The requirement is that, in the end, it should be controlled via a VR headset.

4. The fourth requirement that arose is that this solution should work in a reasonable amount of time. The output should be received by the user as fast as possible to have a feeling that there is no real-time difference.

Based on that, the following list of requirements will be addressed during the thesis and will be evaluated if the thesis's goal is achieved. The first and second goals are tightly connected with the fourth goal. So if the chosen algorithm does not perform well time-wise, then both of the requirements will be unfilled. The time that should be achieved in this thesis for both algorithms is 2 seconds. If it goes over this limit, I will count it as unsuitable for the given requirement.

The second question will also be validated by observing the results and in-depth discussion on what was generated. If the models are not distinguishable and recognizable, then the goal for this question was not achieved.

When these four requirements have been met, I can count this thesis as a success.

Thesis is structured as follows:

1. Section 2 summarizes the related work and information collected during the research.
2. Section 3 covers the minimum techical components needed to implement the solution.
3. Section 4 describes different methods/algorithms used in the components to create the VR world and how it was implemented.
4. Section 5 gives an overview of the research and development process.
5. Section 6, 7, and 8 describe the overall result and what I did in the thesis.
6. Rest of the sections conclude the thesis and summarize the results.

3 Background and Related Work

As mentioned before, there was a literature review [18] done before the work was started on the practical side. The work focused on two key areas that would become problematic when solving this problem. The first problem that I addressed in the research was the positioning of the donkey car. The problem mainly focused on finding different SLAM (Simultaneous localization and mapping) algorithms. The second problem addressed was the creation of a virtual world. It focused on finding algorithms that can generate the meshes from the collected points. In the research, 26 different papers were excluded from the initial set of 1517 papers.

Under this section, there is information about which LiDAR sensor was chosen. This section will also cover the technical side of the thesis and what technologies were chosen for certain entities. Two different entities will be part of the final model (donkey car and VR headset, which will receive the donkey car's data).

The rest of the structure related to the constructed pipeline focuses on building a solution to transfer data between services.

The detailed mathematics involved in the algorithms is not covered in the following sections. This paper aims to focus on finding and combining multiple algorithms instead of going into detail about every algorithm that was analyzed. The focus is on developing a cost-effective and simplistic solution to the problem mentioned above.

3.1 Related work

A few papers were observed that could give me some insights on how to approach the given problem. The first article that was looked into was an article about point cloud visualization with the Poisson Reconstruction algorithm. The article gave good insights on what to use as a VR application and how the scanning of the room was done [32]. In this paper, the results of the scanning also included colored models. The paper explained the process quite well. It first created the room from the scan data and then allowed the user to interact with it. The downside of this is that the algorithm took around 10 minutes to create the room model. The goal of this thesis would be to attempt to do it in real-time. The second resource that was found at the end of the research was the implementation by Nick Shelton regarding the marching cubes algorithm. This was found after the initial tests on the modeling algorithms turned out to be too slow. This was used as a modeling algorithm to render the environment almost in real-time. The site [30] was used to gather initial ideas on how to use the algorithm and what its issues are.

3.2 SLAM Algorithms

Suitable SLAM algorithm was found by trying to answer the following research questions:

1. RQ1: What are the different sensors that have been used with LiDAR sensors to map surrounding environment?

2. RQ2: What are the effective and accurate methods to see and track the position of the donkey car in the real-world using LiDAR?

3. RQ3: What are the pros and cons in using different sensors and different algorithms?

The result of the research showed that there were over 30 different algorithms covered in the papers. The most popular algorithms were Hectorslam, GMapping, and different types of Iterative closest point(ICP) algorithms. However, research also showed that even though the Hectorslam was the most popular algorithm used for SLAM algorithms, there were many subsets of algorithms based on ICP.

A bit older methods were mostly focusing on solving the problem mathematically. New methods published in recent years have included machine learning and neural networks to predict the movement of the donkey car. Of the listed algorithms, Hectorslam was a chosen algorithm for the given purpose because it could support LiDAR-only SLAM, and it is also covered in a lot of different articles.

This meant that multiple implementations could be used as a basis. In the article that explained the Hectorslam, the authors also created a ROS implementation of the algorithm. With this algorithm, only the setup file needs to be created.

3.3 Hectorslam

Hectorslam is a SLAM algorithm that can rely only on the sensor input of the LiDAR sensor. It was chosen initially, as it was referenced in many of the reviewed papers, and the sensor was available at the beginning of work with the thesis. The Hectorslam works by comparing the new scan results with the occupancy grid that contains information about previous scans in the core. The occupancy grid consists of points compared by the log probabilities of being occupied or accessible in the room. Then, when a new scan is coming in, the Gauss-Newton equation is solved to find the transformation with the minimal error [17].

3.4 Point Cloud to Model Algorithms

For the problem of generating meshes from the point cloud, the following research question was added to the research paper:

- RQ4: What are possible solutions to generate and handle polygon meshes from the point cloud?

As few articles described algorithms or processes to create objects from the point clouds, this research question needed more input from different sources. A single algorithm was found in the papers. That was the Poisson surface reconstruction algorithm. After researching the modeling topic, I found another algorithm called Ball-Pivoting Algorithm (BPA). In the later phase of the research, I found two more algorithms (Crust algorithm and Co-cone algorithm), but those were not tested.

I found another article in the later phases of the thesis that focused on testing the performance of these algorithms. I added the article here to give broader knowledge about different algorithms that can be used for the given purpose. It included all of the algorithms I found previously and a few more published before 2000. The main focus of the research was to find out the differences in the quality and the computation time based on the algorithm parameters. There were six different algorithms discussed in the given paper, from which BPA and Poisson surface reconstruction algorithms were directly tested out [26]:

1. Tangent plane estimation. It was proposed in 1984.
2. Delaunay triangulation/Voronoi diagram based reconstruction
3. Tight Cocone Algorithm. It was proposed in 2002.
4. Power Crust algorithm, proposed in 2001.
5. The Ball-Pivoting Algorithm (BPA). It was proposed in 1999. In 2014 a parallel processing algorithm was proposed that increased the algorithm's processing speed.
6. Poisson surface reconstruction. It was proposed in 2006. In 2010, an improvement for this algorithm was created that incorporated the Marching Cube algorithm for surface extraction from indicator function.

When comparing the BPA and the Poisson surface reconstruction algorithm, the main issue that I found is that the BPA algorithm is data-driven. Therefore, it usually takes longer to generate a 3D surface than the Poisson reconstruction algorithm. The paper that was reviewed tested the parameters for the BPA and Poisson Surface Reconstruction algorithm.

Poisson Surface Reconstruction algorithm relies on two different parameters (Octree depth, Samples per node). Octree depth shows the quality of the surface. An increase in octree depth results in high quality and an increase in resolution. Samples per node show the amount of 3D points in each octree leaf.

BPA algorithm has three parameters that influence the result: the ball radius, clustering radius, and the angle threshold. A ball radius is a radius that shows what points in a certain area should be considered for the creation of the triangles. A clustering radius is a radius that is the minimum allowed radius between points. If it is smaller than this,

the points will merge. Finally, the angle threshold is a value of the maximum allowable angle between the active edge and the new edge created by the ball.

The results in the article showed a generation of a mesh with 8000 points with the BPA using the ball with a 0.1 radius, 90-degree angle, and clustering radius of 60 units. It took around 10 seconds to generate. The same object for the Poisson surface reconstruction algorithm took around 5-6 seconds with six samples per node and an octree depth of 6. This would generate similar results in the object generation. However, the generation times rely highly on the parameters, and the generation cannot be directly compared based on these results.

The marching cubes algorithm was also tested as it was found that running modeling algorithms is relatively slow. With the marching cubes algorithm, the speed was only reliant on the size of the grid [24].

3.5 LiDAR

LiDAR (Light detection and ranging) is a method for calculating distances to an object with a laser and measuring the time for the reflected light to return to the receiver [23].

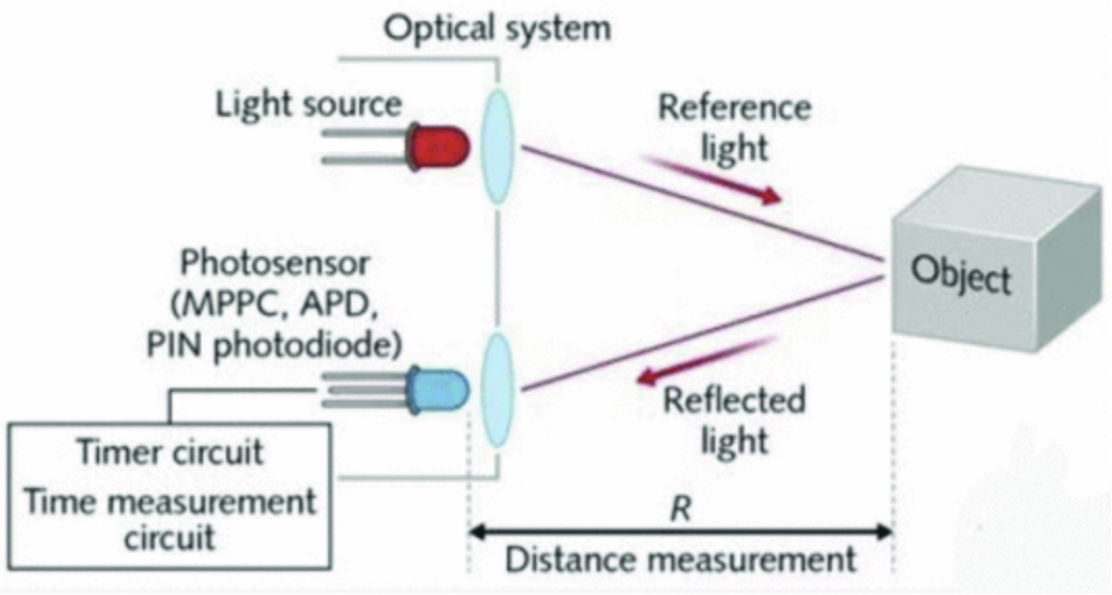

Figure 1. Time-of-flight sensor working principle. [23]

At the beginning of the research, I looked for a cheap and cost-effective LiDAR that I could use as a sensor to test out the given solution with a donkey car. The cheapest sensors that I could use for this purpose were solid-state sensors because the few or no moving parts are more reliable than a typical scanning LiDAR sensor. As a result, I found the following sensors:

- Velodyne Velarray M1600
- Cygbot CygLiDAR D1
- 3D Solid State LiDAR Sensor by Blickfeld

The Cygbot LiDAR was the one that stood out by its limitations and price. When the Velodyne sensor and sensor created by Blickfeld have a pretty extensive range of up to 75 meters. Then for the CygLiDAR, it is approximately 12 meters. As the other sensors are too expensive, the Cygbot was a chosen sensor for this thesis. This concluded the search for a suitable LiDAR sensor. I ordered the sensor as it was relatively inexpensive for a LiDAR sensor (USD 180). The sensor by Blickfeld costs over USD 2000, and the Velodyne sensor is not available yet. Why the Velodyne sensor was brought out here is that it was the cheapest Velodyne sensor that could be used for the given problem.

3.6 Donkey Car

For the donkey car, there are many different options to choose from. One of the options might be to build it by yourself. For example, the donkeycar.com site combines some of the different options for creating it. Another option would be to buy premade donkey cars. When googling for a prebuilt donkey car, numerous providers were found. For the given thesis, a Donkey Car S1 is used, which is provided to me by the University of Tartu. A more precise description of this specific car will be under the component section.

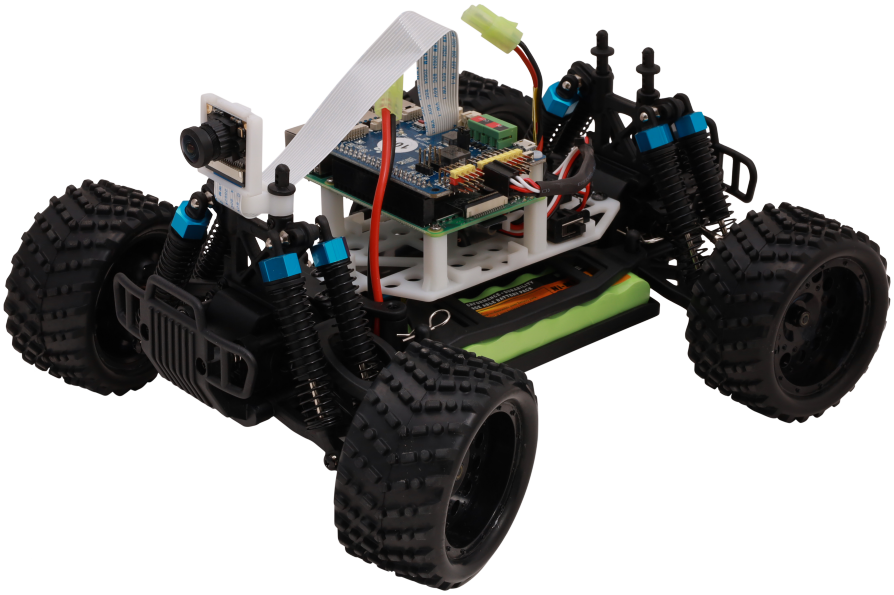

Figure 2. Donkey Car S1 [11].

3.7 Virtual World Generation

The virtual world generation can be done by multiple different softwares, which have quite a simple setup. Popular game engines such as Godot, Unreal, and Unity all have support for VR devices. The author collected reviews from the g2.com site, which is a site that collects reviews for different software and services on the VR game engines. It compares engines based on their popularity and overall satisfaction [13].

Six different engines were analyzed on that site: Unity, Unreal Engine, 3DS Max Design, Maya, Amazon Lumberyard, and Cryengine. The polls show that the most

popular engines are Unity and Unreal Engine, with many users having high satisfaction using them. 3DS Max Design and Maya were less popular choices for the users, but they have as high satisfaction as the Unity and Unreal game engines. Amazon Lumberyard and Cryengine were not popular, and the users were not satisfied with those applications. Results show that when choosing an engine for developing the VR application, the top choices for this would be Unity or Unreal Engine, as they have a large market cap, meaning that the support and community behind it are extensive for fixing specific problems [13]. However, options depend on the project's ultimate needs and prerequisites. For the given paper, I chose the Unity game engine because of my previous experience. The game engine is used to implement the mesh generation algorithm, which uses the preprocessed points from the robot. A deeper explanation of how the robot preprocesses data is available in the methods section.

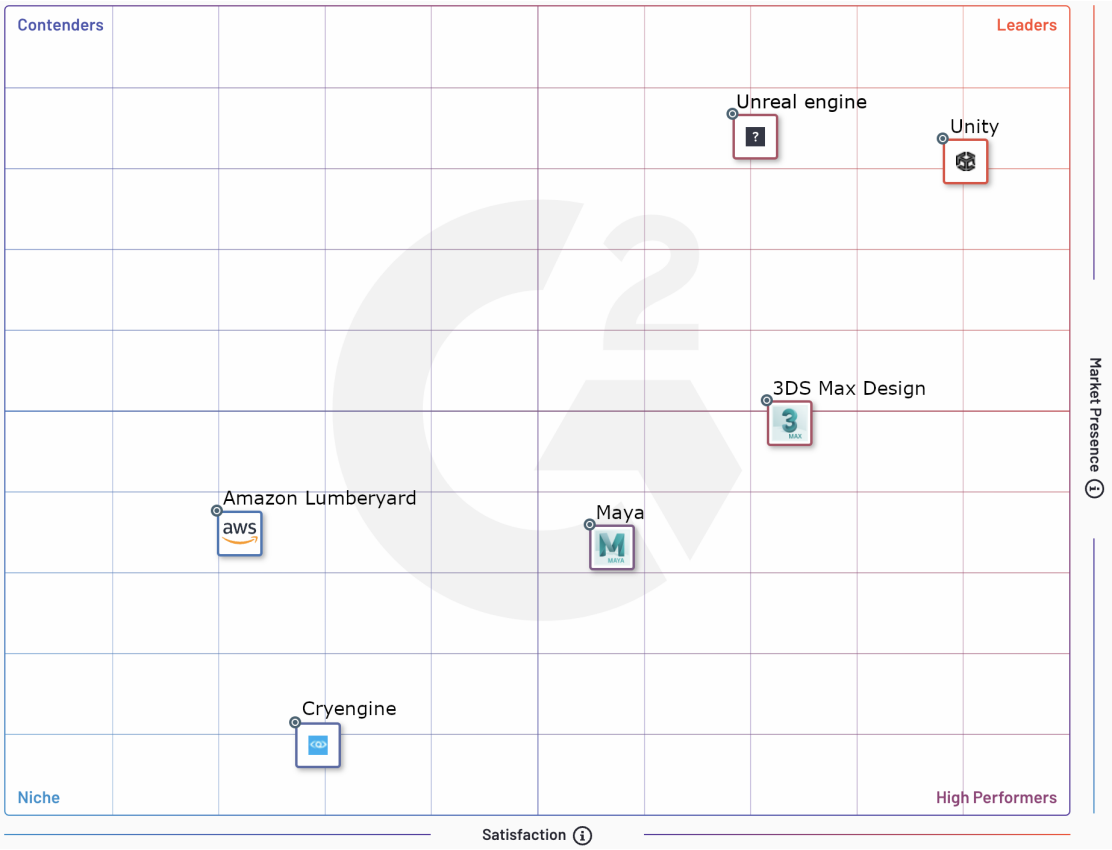

Figure 3. Game engine comparison from the g2.com site [13].

4 Methods

The following section covers all of the chosen methods to solve the main problem of creating a remotely controllable car that creates a virtual world with LiDAR point cloud data that the user can view via the VR headset. The whole process is visualized as a pipeline in the figure 4. The way the whole process communicates is described under the subsection where each section is the next item in the pipeline. The code is available in the private repository. The code for the donkey car [19] and the VR device [20] are located in separate repositories. The access rights have to be requested from the thesis author.

Under the method subsections, the first section covers the information about the LiDAR sensor and how it communicates with devices. The next section describes how and what I used in the robot raspberry device. Finally, the last section describes how the modeling software works and communicates with the system.

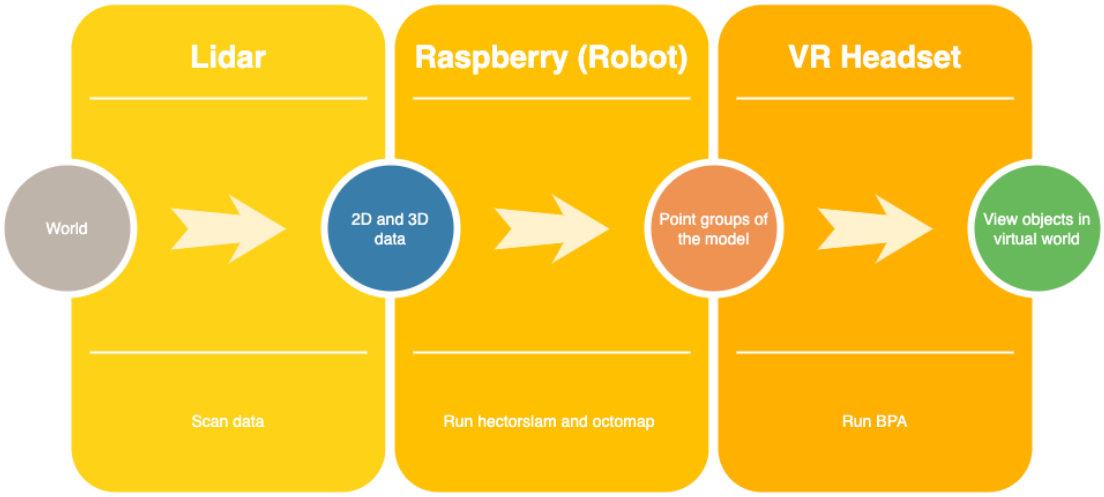

Figure 4. Simple pipeline that shows data flow in the object

4.1 LiDAR setup

The LiDAR sensor that I chose for the current purpose is CygLiDAR D1. It is produced by Cygbot, which is a company that is focused on different LiDAR sensors. As mentioned before, I chose this sensor because it is cheap, and testing out the solutions does not need high-precision or high-quality sensors.

Figure 5. CygLiDAR D1 [6].

Table 1 lists some of the specifications of the LiDAR that are needed later when defining the algorithms in the ROS, and Figure 7 gives an overview of what data is captured with the sensor.

The next step is to set it up for usage. First, the LiDAR sensor will need to be attached to the donkey car. The sensor is mounted on top of the camera by the two screw holes available on the sensor's side. The next step would be to connect it with a donkey car's raspberry controller. The sensor uses the UART protocol to communicate with the devices. The sensor comes with the UART to USB converter, which I connected to the USB ports. After I connected the sensor via USB, the communication via serial communication was established. The documentation of the sensor is available on the company webpage. It has documented information on what packages to send to activate the sensor or to receive information in 3D and 2D [8]. The communication with the sensor is initialized in the next section. At the core, the package seen in Figure 6 needs to be sent to the sensor to start the data stream.

Table 1. CygLiDAR D1 specifications [7].

Parameter	**Value**
Detection Range	2D : < 200mm ∼ 8,000mm 3D : 50mm ∼ 2,000mm
Field of View	2D/3D Horizontal : 120º 3D Vertical : 65º
Resolution	2D : 0.75º (Angle) 3D : 160 x 60 (Pixel)
Measurement speed	15 Hz

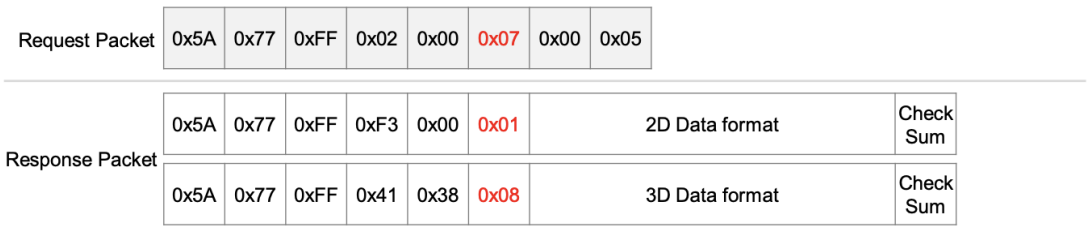

Figure 6. Cygbot data stream [8]

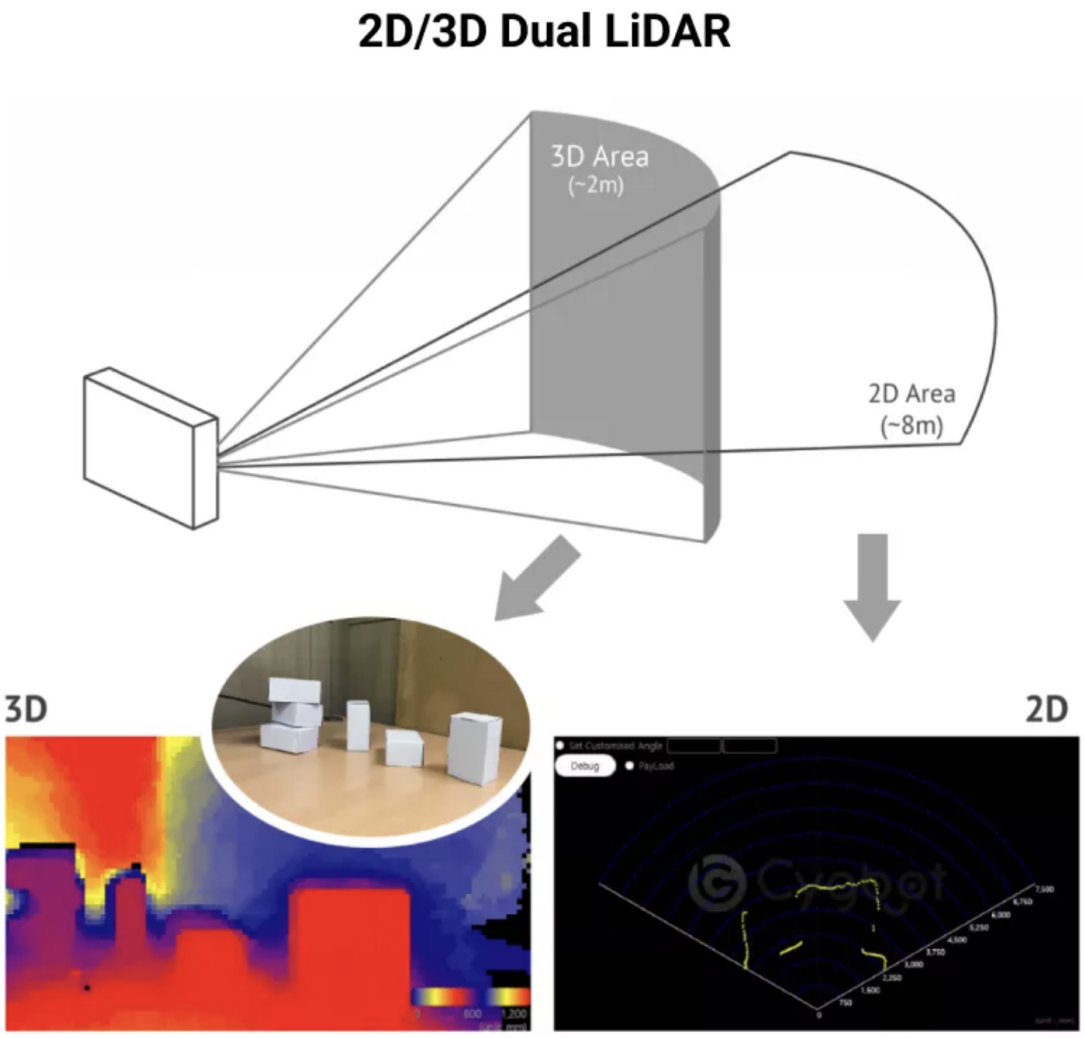

Figure 7. CygLiDAR D1 scanning area [6].

4.2 Donkey Car

The donkey car handles the core functionality of the SLAM algorithms, initializes the connection, and provides the VR set the data with the correct position. As mentioned before, the donkey car used in this thesis is Donkey Car S1. To test out the map generation, the only essential things are the device that can run a Docker and has the required ports to connect with the LiDAR device (The first tests that I did were with the computer running Linux OS, and the sensor was moved by hand). All of the components that the current donkey car has are to increase the comfort of testing the slam algorithm and whole pipeline and, in this case, add the remotely controllable entity. The donkey car comes with the following components:

- Robot body
- Raspberry Pi 4
- IMU(inertial measurement unit)
- Camera
- Batteries

The donkey car is already assembled, so there is nothing special to modify (only the LiDAR sensor needs to be attached to the car). The main changes that I made to the car were software changes. The Raspberry pi uses a Raspbian OS. So the first thing that I did was to setup the Docker in the raspberry. Docker is an open-source containerization platform. It enables packages to be added into containers which can be run in any operating system where the Docker is installed. The Docker is used here because setting up the Robot operating system(ROS) is quite challenging. I followed the steps on the howto [12] website, and the result was that the ROS was missing some unknown config. Since it was a pretty big package, it was hard to debug the problem. However, Docker has a base image "ros:noetic-ros-core-focal" [29] available, and setting it up was pretty simple without any issues. It also meant that if I needed to reinstall or test something, I could erase the old image or generate a new one without worrying about installed packages.

ROS is an open-source tool containing a set of different libraries and tools that can be used to build robot applications. ROS is an event-driven architecture where the nodes talk over the centralized node. All of the recorded data is published on specific topics in the ROS. The reason why I use ROS here is that some of the algorithms have already been implemented there. Furthermore, it has some useful libraries that I use, such as the server for managing WebSockets, to where the connection is set up from a VR device. In the given thesis, the noetic version is used as it supports ROS 1 packages, and it is the suggested version with long-term support. Everything that I will implement in the robot will be built on top of the ROS packages.

In the figure 8, there is a base Dockerfile that I wrote that generates the docker image for the ROS environment. The first stage of the installation installs the core packages of the ROS. The next step recompiles the Cyglidar package. The last step installs the packages that are needed for this thesis. The following packages are installed after the core of the ROS packages:

- cyglidar_d1 - This is a package from the Cygbot team [9]. This package is outdated and, for that reason, needs to be fixed and recompiled. The needed fixes of the code are available in the private repository [19]. After installing the package and running it, the package will send the start bytes to the sensor, which starts to send data to ROS. After that, the data is consumed and produced to the scan_3d, scan_2d, and scan_lidar topics. All of the scans are with the structure shown in the Table 2. The data received from the point cloud is visualized in the RViz (package that includes a graphical interface that allows users to visualize a lot of information, using plugins for different topics. [15]) in the Figure 9.

- ros-noetic-rosbridge-suite - The rosbridge is a package that creates a server to which the external services can connect over the websocket [25]. It enables the transfer of the info available in the topics to the external device.

- ros-noetic-hector-slam - Now that the LiDAR sensor has been set up to publish the relevant information to the ROS, the Hectorslam algorithm can be set up in the pipeline. It would start processing the information from the sensor. For the following step, the hector_slam package was used from the ROS [16]. This package was implemented with the help of the paper introducing the hectorslam [17]. A few things that need to be set up in this step are to define the laser parameters and how it interacts with the environment. The parameters that were defined are available in the car-lidar repository [19] under the launch_files folder in the hector_mapping folder.

- ros-noetic-octomap-mapping - This is used to reduce the number of points that need to be analyzed for modeling the environment [14]. Of course, this will reduce the quality but will increase the algorithm's speed as there are fewer points to process.

```
FROM ros:noetic-ros-core-focal
RUN apt-get update && apt-get install --no-install-recommends -y \
    build-essential \
    python3-rosdep \
    python3-rosinstall \
    python3-vcstools \
    && rm -rf /var/lib/apt/lists/*
RUN rosdep init && \
  rosdep update --rosdistro $ROS_DISTRO
RUN apt-get update && apt-get install -y --no-install-recommends \
    ros-noetic-ros-base=1.5.0-1* \
    && rm -rf /var/lib/apt/lists/*
RUN mkdir -p /home/catkin_ws/src
WORKDIR /home/catkin_ws

ADD /cyglidar_d1 /home/catkin_ws/src/cyglidar_d1/
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get \
    install -y ros-noetic-pcl-conversions ros-noetic-pcl-ros
RUN /bin/bash -c \
    '. /opt/ros/noetic/setup.bash; cd /home/catkin_ws; catkin_make'
RUN apt-get remove -y --auto-remove ros-noetic-pcl-conversions \
    ros-noetic-pcl-ros

RUN apt-get update && apt-get install -y \
    ros-noetic-rosbridge-suite ros-noetic-hector-slam \
    ros-noetic-octomap-mapping

ADD /launch_files/hector_mapping/mapping_cygbot.launch \
    /opt/ros/noetic/share/hector_mapping/launch
ADD /launch_files/hector_slam_launch/mapping_cygbot.launch \
    /opt/ros/noetic/share/hector_slam_launch/launch
ADD /launch_files/octomap/octomap_mapping_cygbot.launch \
    /opt/ros/noetic/share/octomap_server/launch/

ADD /run.sh /home/catkin_ws/run.sh

ENTRYPOINT [ "/bin/bash", "/home/catkin_ws/run.sh" ]
```

Figure 8. Base dockerfile for the robot image

Table 2. Data produced to the ROS topics from LiDAR

Parameter	**Value**
Header header	Contains data about the coordinate frame and the timeframe
int width	Width of the data
int height	Height of the data (1 for two dimensional data)
Fields fields	Field data (x, y, z) and how it is defined in data
bool is_bigendian	How the data is structured in the data
int point_step	Length of a point in bytes
int row_step	Length of a row in bytes
string data	Data

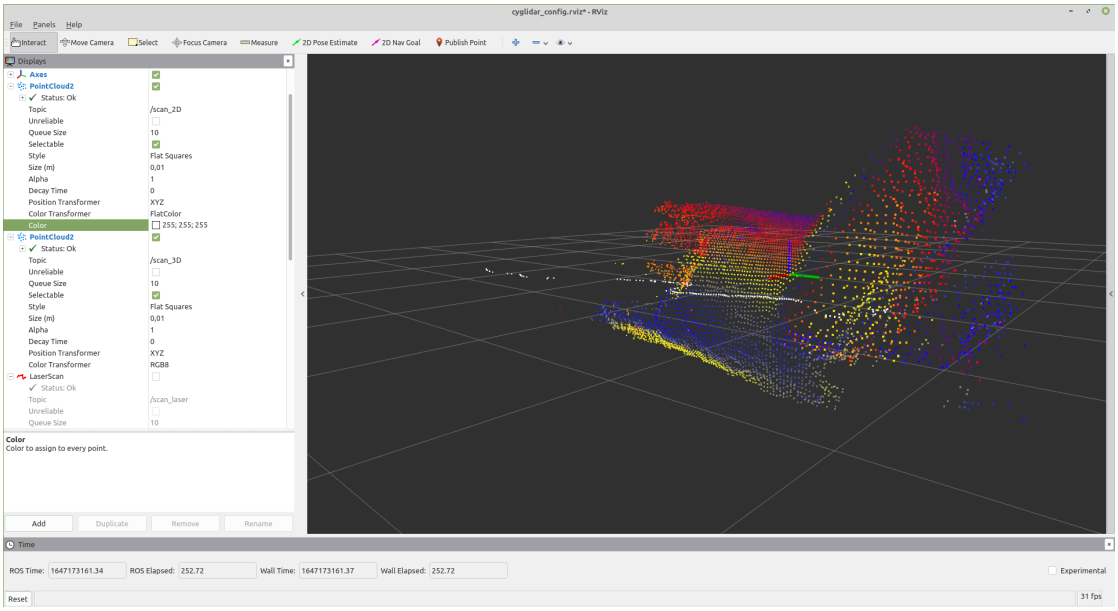

Figure 9. Image of the data from the Cygbot sensor in the RViz

The last phase of the installation in the Dockerfile contains information about the run.sh file that is copied to the container. It includes starting commands for the different launch files(In the ROS, these are the files that contain information about starting arguments for the packages [27, Chapter 6]). The next step would be to run the build command for the docker:

```
sudo docker build -t=ros_cygbot .
```

When everything is set up, and docker has finished building the image, I run the following command to start the created container in the docker:

```
sudo docker run -p 9090:9090 --device=/dev/ttyUSB0 ros_cygbot
```

This will start the following flow:

- LiDAR sensor will start to send information to the ROS on certain topics
- The SLAM algorithm will pick up the sensor data and calculate the robot's position based on that.
- The Octomap will reduce the points into voxels based on the number of points in a certain region. The centers of the voxels are produced to the topic in the ROS. That topic is accessible in the octomap_point_cloud_centers, and the data structure that it follows is sensor_msgs/PointCloud2 [3] structure. It is a general structure for defining the point clouds in the ROS.

Port 9090 is forwarded because it is used for incoming WebSocket connections. The device parameter shows the device that is forwarded to the container. On the raspberry, it is a LiDAR sensor so that the ROS can communicate with it. When this is done, the device should be running the SLAM algorithm. It produces the transformed points and position of the car to the topics.

4.3 VR

The final step in the process is where the world is generated based on the data received by the robot. The development environment that I chose for the current problem is Unity. The idea of this step is to run a modeling algorithm that generates the world based on the input from the Octomap points. The code for the project is available at private repository [20]. It contains the code for both BPA and Marching Cubes algorithms. For this case, the full implementation was added to Unity. In Unity, the whole flow was built up to set up the donkey car's connection and handle the point cloud by switching the modeling algorithms out. As I finished implementing the described flow, the only problem that needs to be solved for RQ3 is adding the VR device support and testing it. The whole flow from scanning the point cloud to modeling the environment is working as expected.

The following sections describe how the core of the modeling work and how they were implemented. Finally, the results of these algorithms are brought out in the Discussion section.

4.3.1 Marching Cubes Algorithm

The Marching Cubes algorithm was one of the algorithms that was found, which might be fast enough to process the point cloud. I did it to test and compare the speed of this algorithm to the other algorithms and found out that it could be used as a replacement for the other algorithms. The other algorithms took too much time to process the points during the physical testing. I added it as a method to receive the results in real-time. Marching Cubes is an algorithm that was published back in 1987. It consists of creating a model based on the grid map, where each grid point is with on or off state. Based on the activated grid points, it creates triangles inside the given voxel (a cube inside of the grid) as seen in Figure 11. Two hundred fifty-six different point combinations create the faces for the mesh. Figure 10 also shows the position of the triangle vertices. The current implementation of the mesh generator, which is done during the thesis, uses the vertices for each voxel as shown in the Figure 10. However, the vertices' position between the corner points can also be adjusted based on the weight of the points in the corner. If one of the corner points has a lower amount of points from the LiDAR scan related to that specific point in the Octomap, then the vertex position is moved towards another point [24].

4.3.2 BPA algorithm

Ball-pivoting Algorithm(BPA) was another algorithm that I tested out in this thesis. The main idea is that the virtual ball, which has a defined radius, checks if there are only three points in a specific area. How it works is that the "ball" starts going around the

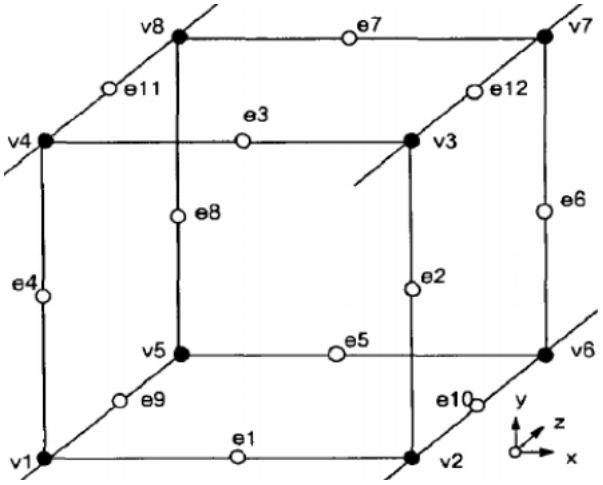

Figure 10. Marching cubes algorithm points [24].

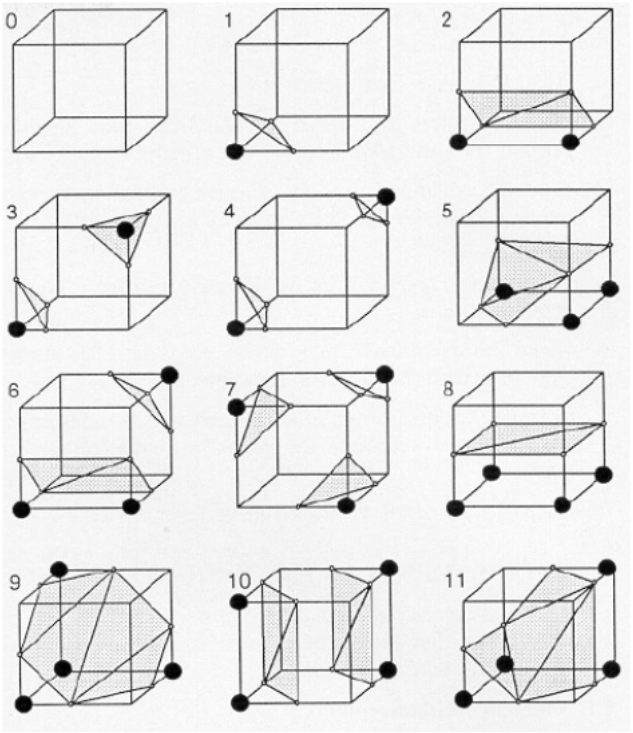

Figure 11. Illustration of the faces generated by the on or off points [24]

mesh from the seed triangle. The first step starts with searching for a seed triangle that could fit in a ball. When the search is done, it starts pivoting the ball around the edges of

the seed triangle. After that, the ball starts to go between points, and when only three points are inside the ball, then the triangle between them is created. The new edges are created, and the ball will pivot around them again. When all edges have been tried, and if there are more points available, the process is repeated. If all of the points have been processed, then the process is ended [5]. Figure 12 shows a simplification of how the algorithm works.

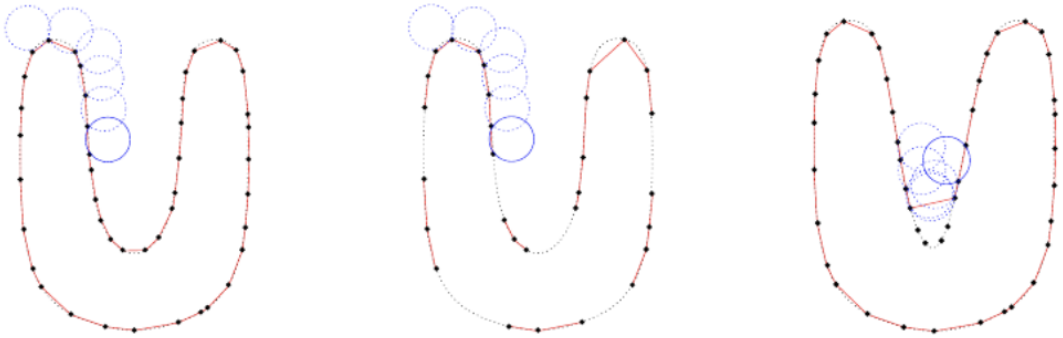

Figure 12. Ball-pivoting algorithm. Section a shows the complete picture of the points in the correct range. In the b section, some of the points are further than the ball radius, so they are excluded [5].

5 Implementation

This section will give an overall architecture of how it was created and how much time certain activities took. Table 3 shows the summary on how much work was done during each month. Research activities at the beginning took around 20 hours to understand how I could solve the given problem and how to approach it. During that time, I did a literature review to collect more data about the possible solutions. This took around 90 hours to process through all of the found papers. The next step was implementing the system in the given entities (Donkey car and VR device). I took the initial tests on a regular computer with Linux operating system. I tested it out before I got the donkey car for testing. The images that are taken in the RViz are taken from the period when I did testing on the computer. It was one of the reasons why I chose the LiDAR-only solution, as the computer did not have any other sensor available at that moment. Initial setup and understanding of how the ROS works took another 60 hours. Setting up the ROS was quite challenging as there were multiple problems with unsupported packages between different versions of ROS. After I set up the ROS, I started testing the algorithms and how they would work. As the package for the Cygbot was out of date and needed to be debugged, fixed, and recompiled to make it work correctly, it took me another 30 hours. After this, the configuration files (launch files) needed to be created for the ROS packages. For most of the packages, it went smoothly, but setting up Hectorslam took some time as it required a specific configuration from the sensor, which needed additional setting up on my part. It took another week to get it up and running, which took around 40 hours.

The chosen approach was to use docker when porting the system over to the donkey car. This meant that there would not be any problems with unresolved packages, and installation went without issues. Running and setting up the robot was relatively straightforward and took around 2-3 hours. However, I tried to install the ROS directly into the donkey car in the beginning. This took around 10-20 hours until the docker approach was chosen. It was because some of the packages had the same problems as the PC. Setting up the ROS was relatively fast, and creating a Dockerfile with testing it out took about 20 hours.

The next step was to focus on the algorithms in Unity. Initially, I tested how to set up the WebSocket server and create a connection between Unity and the car. It meant checking how to subscribe to different topics and to what topic I should subscribe to. This took around 10 hours. At first, I tried to process the point clouds directly in the Unity game engine. The first test was to implement the BPA algorithm. Looking into how it works and modifying it (a package was used for it, but it needed modification to process the point cloud). However, as the algorithms were too slow to process the number of points scanned from the LiDAR, the clustering algorithm was needed to reduce the number of points. When I investigated different modeling algorithms, some of them used octomap to reduce the algorithm processing speed. So I considered to, instead of

modeling the environment based on the points, I would model the environment based on the octomap grid. The described process took around 30-40 hours with testing to set up. The ROS was still run on the PC when this was being tested.

The next step was to look into implementing the Octomap in Unity. As searching for the possibilities, I found out that there is already a package available in the ROS for the given process. So the chosen approach was to move the logic of processing the point cloud to voxel from Unity to ROS. The implementation was pretty straightforward, and rewriting the whole process took another 30 hours. After this was implemented, the tests showed that this was relatively fast to process. However, the issue remained when I moved the car, and more points were scanned. Section 6 explains how much the time increased when scanning more points.

Another algorithm that was considered was the Poisson Reconstruction algorithm, but as its speed is not relatively faster than BPA, then this idea was dropped, and I started looking for other options. The first idea was to do some of the processing logic to GPU. This would mean that parallel processing would have been used to speed up the process. However, while researching the Poisson Reconstruction algorithm, I came upon the article that used the Marching Cubes algorithm to improve the Poisson Reconstruction algorithm. Moreover, as it worked with the grids, I thought I would test this out before implementing the parallel processing in GPU. At this point, I rewrote the core in Unity to increase the quality of the code. It took about 10 hours. The Marching Cubes algorithm was quite robust for the points, and as it did not try to find surrounding points but only worked by looping over the grid, it was relatively faster. Tests also showed that Marching Cubes could be used for a given thesis even though the quality is relatively low. Testing and implementing this took another 50 hours. The next plan was to implement the Marching cubes in the compute shaders to run it in the GPU and voxelize the world to generate the meshes based on the voxel where the donkey car is located. Unfortunately, this was not managed in this thesis, but the plan is described in Section 7.

Table 3. The working schedule which is broken down by the month.

Month	**Activity**
September	Initial research and planning of the thesis topic (5h) Research on similar approaches (10h)
October	I started working on a literature review. The first papers were reviewed and analyzed. (10h) Research on the LiDAR sensor, how they work, what to choose, etc. (10h)
November	Analyzed the rest of the papers and started writing the literature review. (70h)
December	I received the LiDAR and did the first tests with it. I started looking at approaches on how to implement this on ROS. (25h) Finished with literature review (20h)
January	Setting up the ROS and started implementing SLAM algorithms. Debugged issues with LiDAR. (30h)
February	Finished setting up the ROS and packagages, that are required to run the flow (30h) SLAM algorithm implemented and working (30h) Started working on initial setup on Unity and setting up the connection (40h)
March	implemented BPA in the Unity/rewrote some logic in the system (80h) Set up the donkey car, install files and set up the container (50h)
April	implemented and tested out marching cubes algorithm (50h)
May	Added testing functionality to the Unity to test out the speed of the algorithms. (20h)

5.1 Architecture

The following section gives an overview of the final architecture of the system. In Figure 13, we can see the component diagram of the whole system. It contains three components: a donkey car, a VR device, and a computer. The computer is added to the diagram as it is used for developing and debugging the system. The extra component that is added to the diagram is a Time calculation module in the VR/Computer component. It was added to measure the differences between the algorithms and their speed. The rest of the structure follows the description of the system that was described in the previous chapters. The individual components of the high-level components are following:

- Docker - Docker is an open-source containerization platform. Each package that I used in the generated Docker container is described in Section 4.2.
- Manage - Manage is a part of the donkey car initial setup, which opens up a server to which the user can connect. It allows the user to control the donkey car and see the camera image.
- Google Chrome - This is a web browser that connects to the Manage component. In theory, every browser should fit the given purpose.
- Websocket Manager - This component handles incoming topics between VR/Computer and donkey car. It is an interface between the algorithms that process the data and the topics in the ROS.
- Marching Cubes - A component that generates the mesh from the octomap points with the marching cubes algorithm.
- BPA - A component that generates the mesh from the octomap points with the BPA algorithm.
- Time calculations - A component that calculates the time between the received topics. Each scan/update in the topic is connected with a specific id, which stays the same between scanning, processing in slam algorithm, and voxelizing it the octomap.

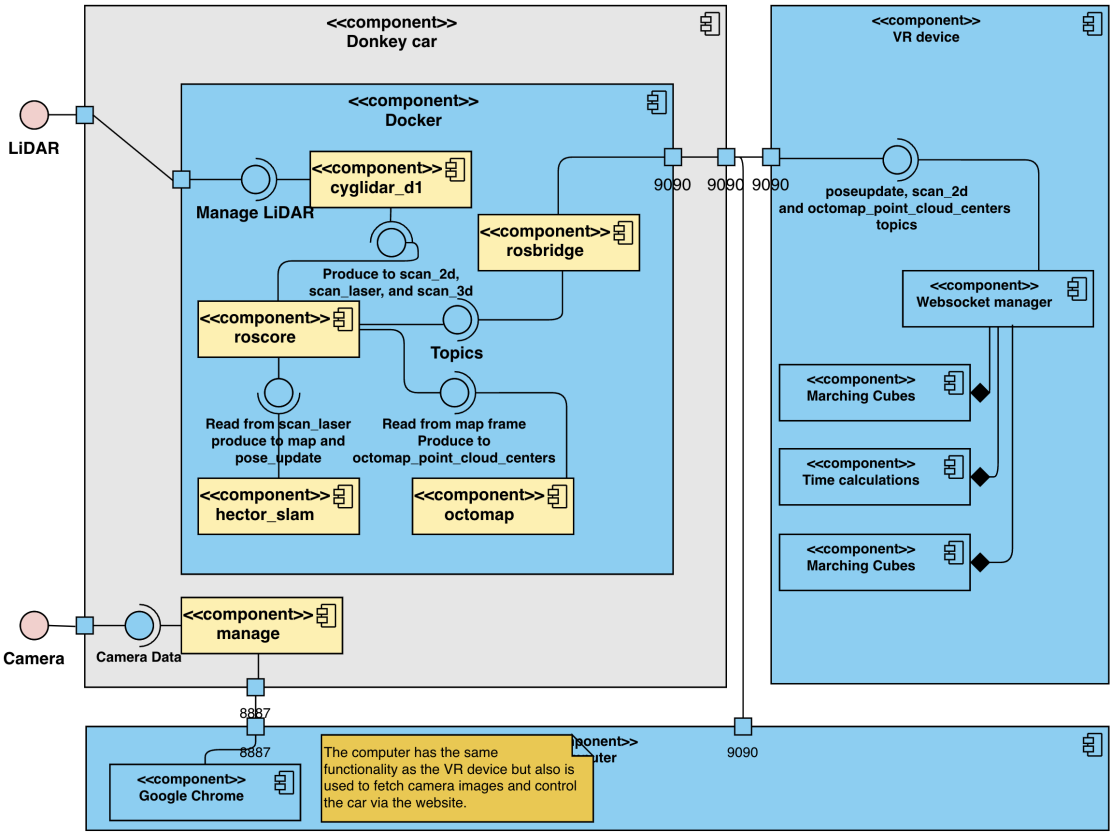

Figure 13. Component diagram of the system.

6 Discussion

This chapter presents the results of the practical side of the thesis and the general capabilities of the algorithms. The specifications of the computer used in the testing process can be see in the Table4.

Table 4. Computer used for testing [7].

Component	**Value**
Computer	MacBook Pro (16-inch, 2019)
Processor	2,6 GHz 6-Core Intel Core i7
OS	macOS Big Sur Version 11.6.5
Memory	32 GB 2667 MHz DDR4
Graphics	AMD Radeon Pro 5300M 4 GB

The first result that will be analyzed is the data from the robot. As mentioned before, the camera was one of the components of the donkey car, and the received camera images are used to compare the results from the donkey car. Figure 14 shows the first image of the data that is done with the camera. Figure 15 shows the same image fetched from the LiDAR cloud. Figure 16 shows the point cloud from the top-down view. As we can see from that image, the point cloud for the walls is quite noisy, causing uneven models for the wall generation. The used scan is the data from the Octomap, which contains the data about the point cluster centers from the scan. The red ball in the image shows the position of the donkey car. The localization algorithm has already been run for these cases, and the points are from the Octomap grid.

The first tests I made contained only the unprocessed point clouds, which on moving the car produced millions of points, and the algorithm was too inefficient to process them. That is why between localized points and the modeling algorithm, there is an extra layer that reduces the number of points called Octomap. As a result, the total time for the scanning algorithm is between 1-1.5 seconds. It is the time between when the scan is taken to the point cloud with the LiDAR sensor to it being voxelized by the Octomap.

There are a few problems with the Hectorslam that I found during the testing. One of these may be the reason why the localization of the car went out of sync. The first is the tunneling problem, and the second problem is the car rotating too fast, which made the car position go out of sync with the map. These issues are more accurately described in section 6.2.

However, we can conclude that the SLAM algorithm for the donkey car was found for the purpose of RQ1, and it fits the boundaries of our time goal in RQ4.

Figure 14. View from the camera.

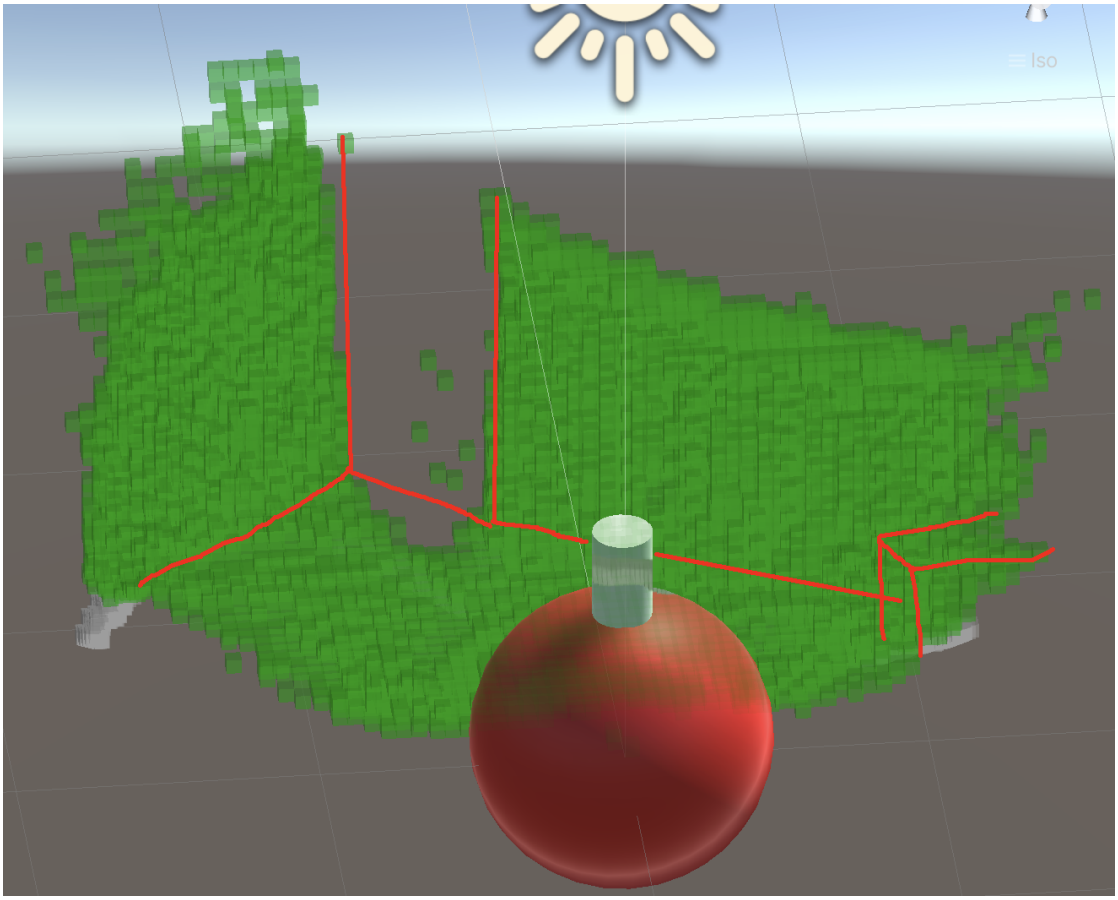

Figure 15. Lidar point cloud of the room.

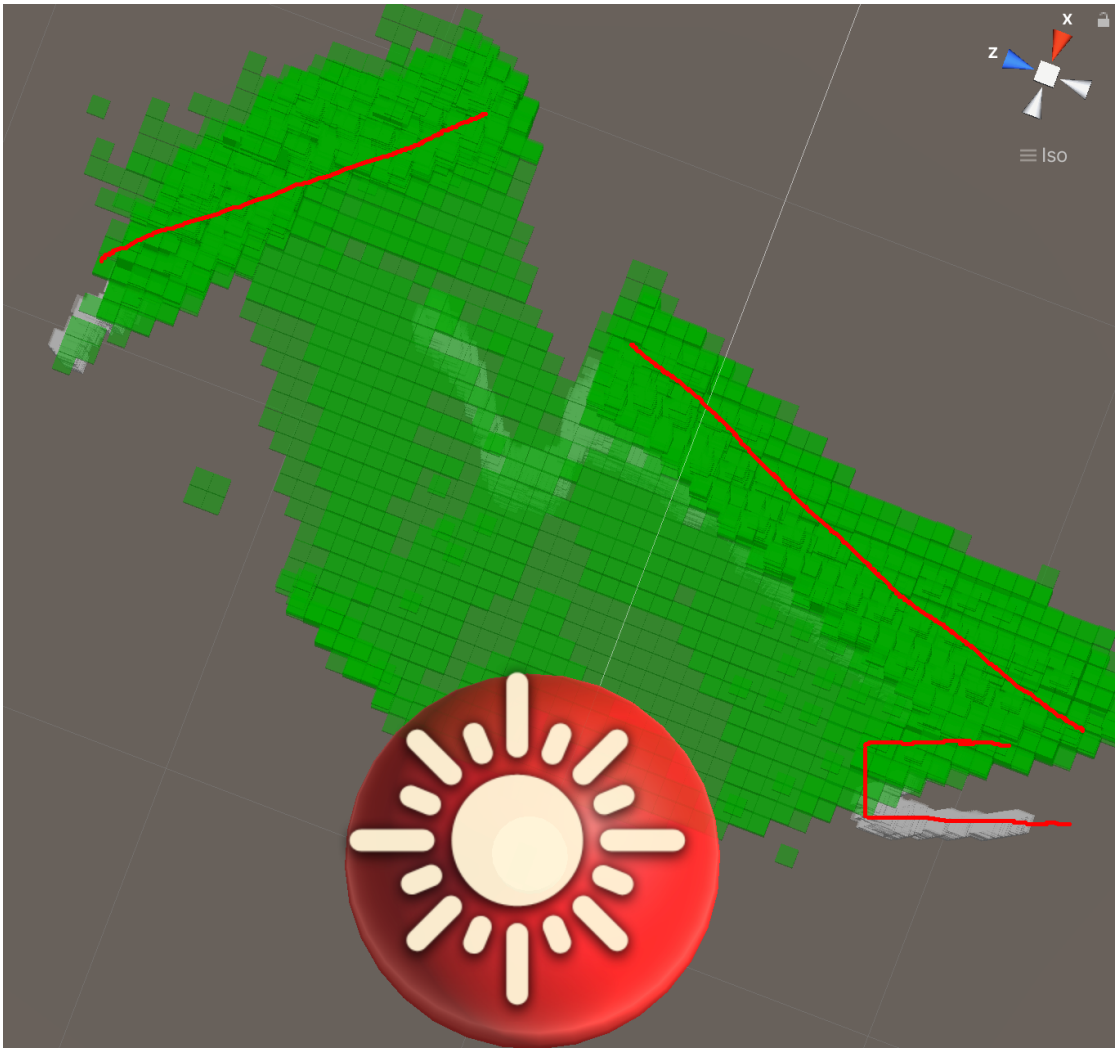

Figure 16. Lidar point cloud of the room from top.

6.1 Modelling Scan From the Car with Static Position

Next are the results for the model creation between the different algorithms.

The first algorithm that I tested out was the BPA algorithm. The code snippets and references for it were taken from the implementation of the BPA for Unity [28]. The issue that arose with this algorithm is that it is pretty slow. The test showed that 1100 points in the box with a size of 100x50x60 took around 3 seconds to process with a ball radius of 1. The main reason for this is that the calculations take time, and this algorithm also considers the normals in the calculations. In the figure 12, we can see the scan of the first test of the wall. The benefit of this algorithm is that it gives more consistent results as the faces behind other faces are excluded. This means that the generated mesh would not have multiple layers of mesh faces in the wall contrary to the way it is done with the marching cubes algorithm in the Figure 20. It can handle single outliers without creating a mesh from those points. However, the issue with the BPA is that it takes quite some time to generate the faces from the point cloud. With running the algorithm for 1100 points, the algorithm took 1 second longer then the limit set in the RQ4, meaning that this algorithm will be excluded as a possible solution for this problem. The following sections show that the time to process the data increases from seconds to minutes by having 20 times larger point set.

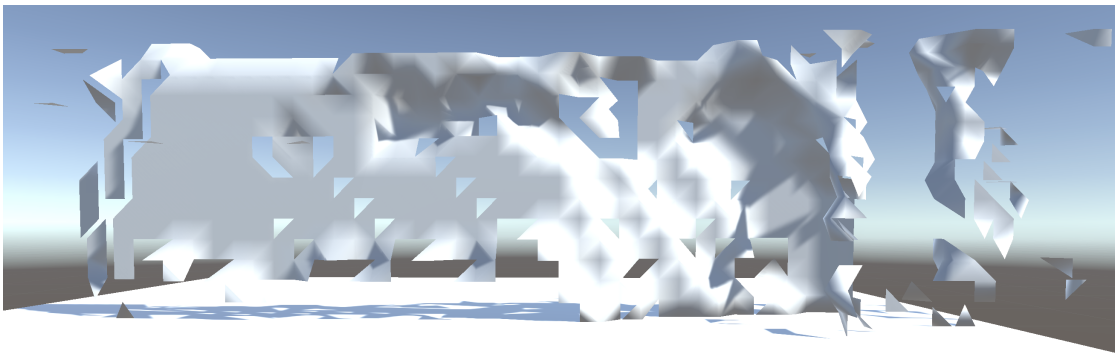

Figure 17. Scan done from the wall with ball-pivoting algorithm.

The next algorithm that I implemented was the marching cubes algorithm. The goal is to do it dynamically while the car is moving. The marching cube is a robust algorithm as it is not trying to find connections between points like the other algorithms. It instead checks specific coordinates, and because of that, we do not need to loop over points at specific areas. The problem with this algorithm is visible holes in the meshes when the

given cluster does not have enough points, as can be seen in the Figure 18. Marching cube is implemented based on the tutorial [21]. The algorithm is relatively fast when running on the CPU (100x60x50 size grid was processed in 0.5-1 second). From the previous images that were compared in the Figure 14, a mesh is created with the marching cubes algorithm that can be seen in the Figure 19.

The reason for the holes is the same as noted before. The holes are sharp because of the misdirection of the normals of the faces. The reason for this is, as noted before, noisy scan data that makes the wall thicker and creates multiple possible layers of points, as seen in the Figure 20. Also, the algorithm cannot handle outliers, and with single points, it will still create a face. This is visible on the left side in Figure 18. If we would do it dynamically, then this would be the algorithm to choose. Implementing this based on the weights of the Octomap would increase the quality, and adding a few more rules based on the surrounding coordinates would also help remove the holes in the mesh. Considering the point normals in the calculations, we could have similar result as with the BPA but with less time. The testing results show, that this algorithm is relatively fast and can be used as it satisfies RQ4. The world needs to be voxelized, and based on the current grid, where the car is positioned, only the surrounding grids get processed. As this loops over the whole grid, its complexity is $O(width * length * height)$.

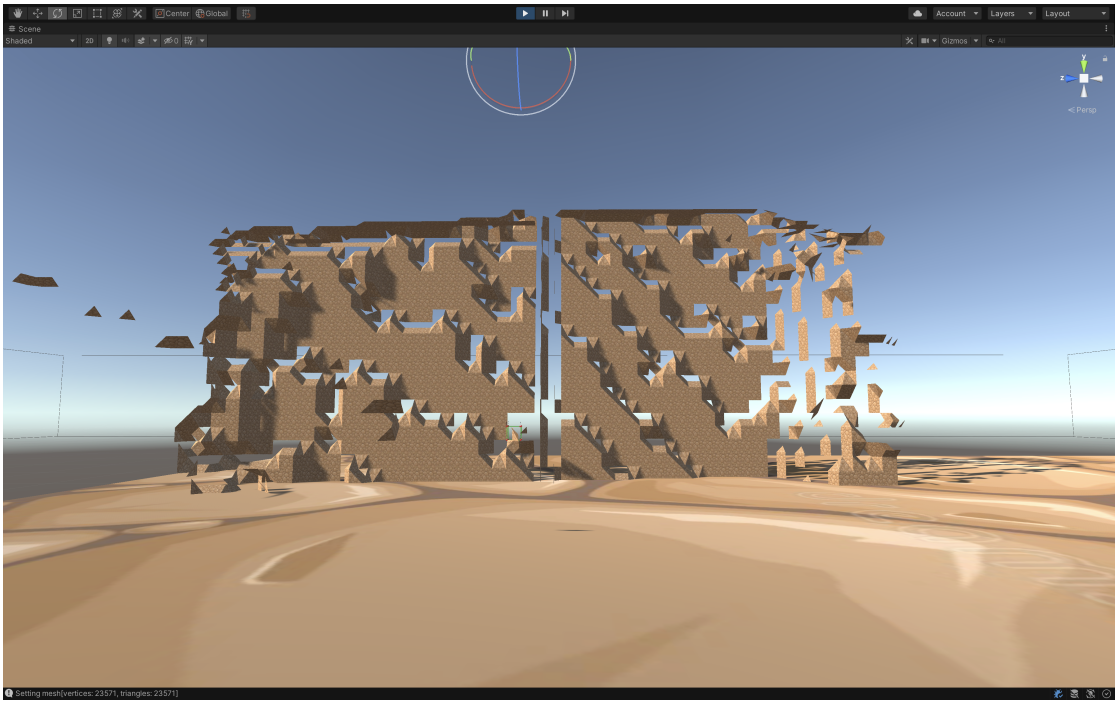

Figure 18. Scan done from the wall with marching cubes algorithm.

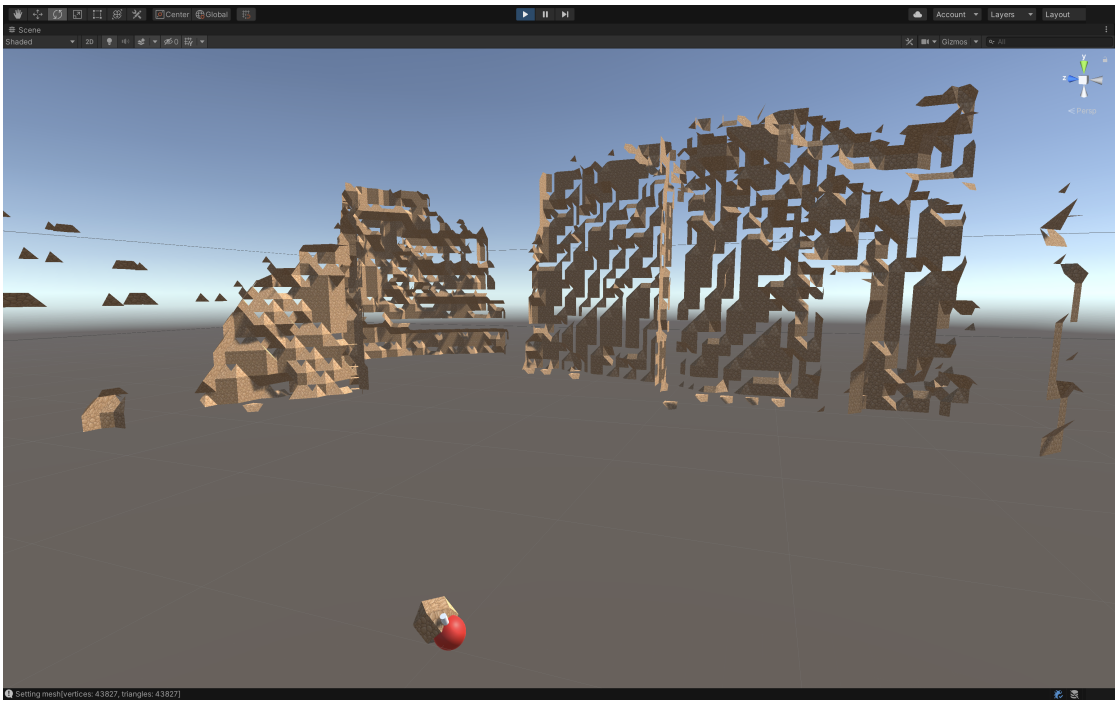

Figure 19. Scan done from the wall with marching cubes algorithm.

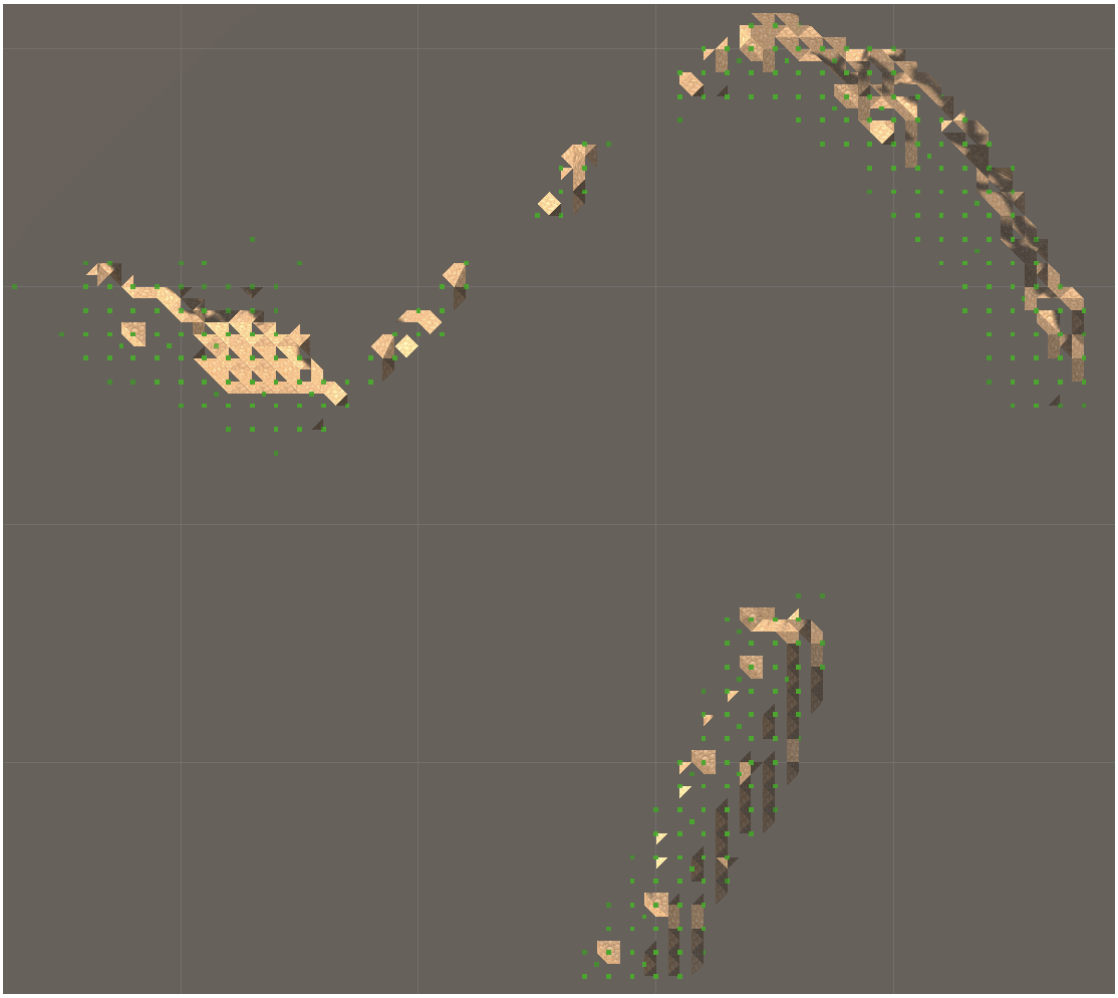

Figure 20. Scan done from the wall with marching cubes algorithm.

6.2 Problems

The first problem that occured was with the SLAM algorithm. As I chose the LiDAR-only algorithm as a SLAM algorithm, the environments with parallel walls might cause a drift in the SLAM algorithm. With that, the point coordinates will be with wrong transformations. Figure 21 shows the main issue with the scans. The scanned area points stay the same as the car moves from the red to the green point. This means that for the algorithm, the car has not moved. The error increases based on the distance the car has moved. This problem could be solved by introducing a second sensor, such as the inertial measurement unit (IMU), that would give us inertia, angular rate, and the body's orientation. Merging the outputs from both sources would give us better position estimates as the algorithm would have something to fallback to.

The second problem that I found during the thesis is the iteration and general slowness of the algorithms. As noted in the previous section, a thousand points for a BPA can take up to 3 seconds to process. As the car moves, the amount of points increases rapidly, which means that the algorithm's speed decreases as well. One of the possible solutions would be to move some of the calculations from the CPU to GPU, as the GPU has a lot more computing power for simple calculations.

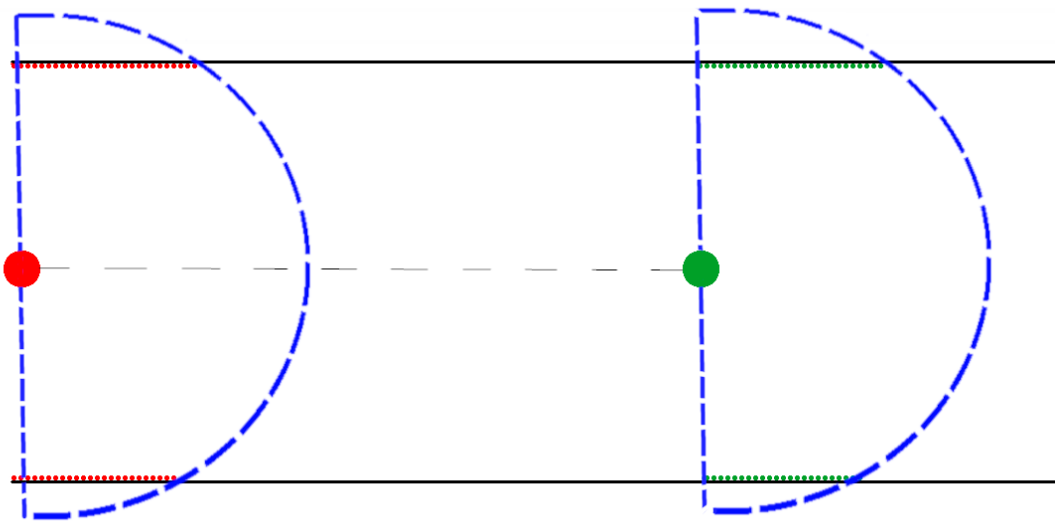

Figure 21. Hectorslam scans from different locations.

6.3 Scan From the Moving Car

In Figure 22, we can see the car scanning the room with the marching cubes algorithm. In that figure, I rotated the car in a single position. As mentioned before, the error produced by the Hectorslam is more visible in this situation. The wall that the robot is currently facing should be parallel with the wall on the other side of the room, but it is slightly out of place. The marching cube algorithm took around 0.5 seconds to generate the mesh. Cloud contained around 22600 points. In Figure 23, we can see the same room modeled with the BPA algorithm. For BPA, it took 5 minutes and 55 seconds to generate. With 1100 points the algorithm only took about 3 seconds to generate.

As mentioned before, the BPA algorithm is too slow to suit the RQs of the thesis. On the other hand, Marching Cubes can be used as a low poly environment generation tool, and it is relatively fast. Therefore, it fits the RQ4 goal to have the algorithm speed to build the environment down to a reasonable area.

Nevertheless, the results show that the quality of the models is too low at the current level. It is hard to distinguish or recognize any other objects except the walls. The next chapter talks about the potential use of improved Marching Cubes algorithm by moving the calculations to GPU that I found during the writing of the thesis, which I was not able too implement during this thesis.

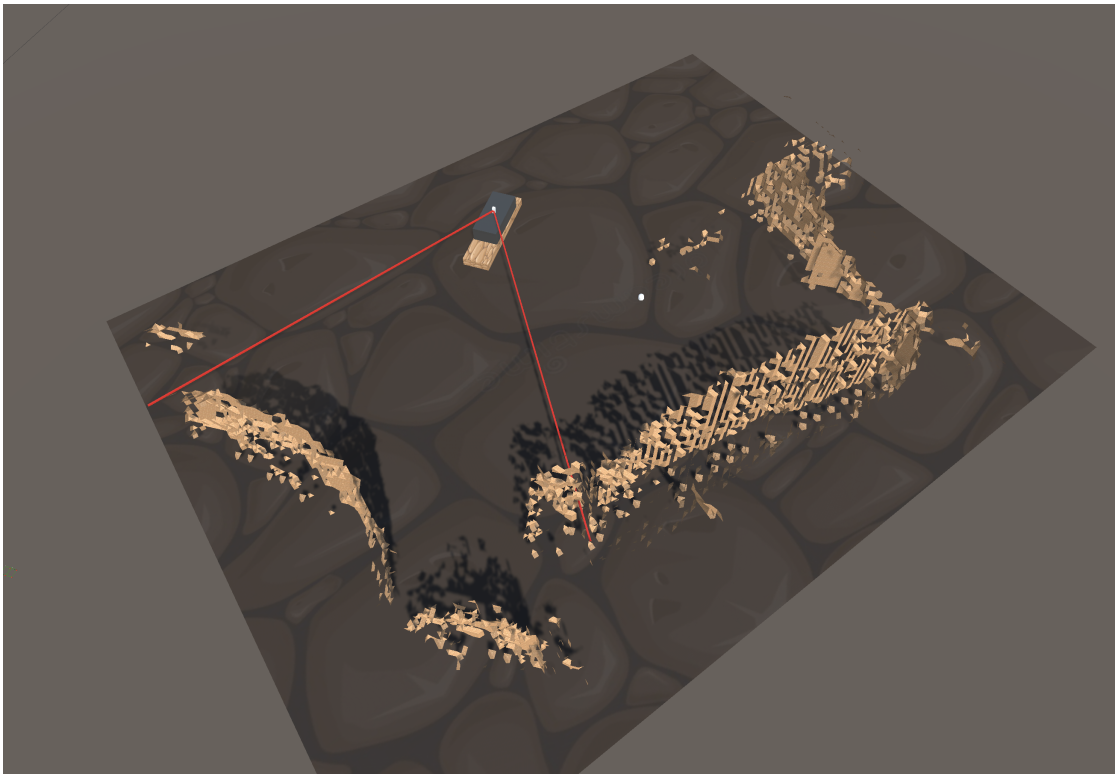

Figure 22. Scan done from the room with marching cubes algorithm.

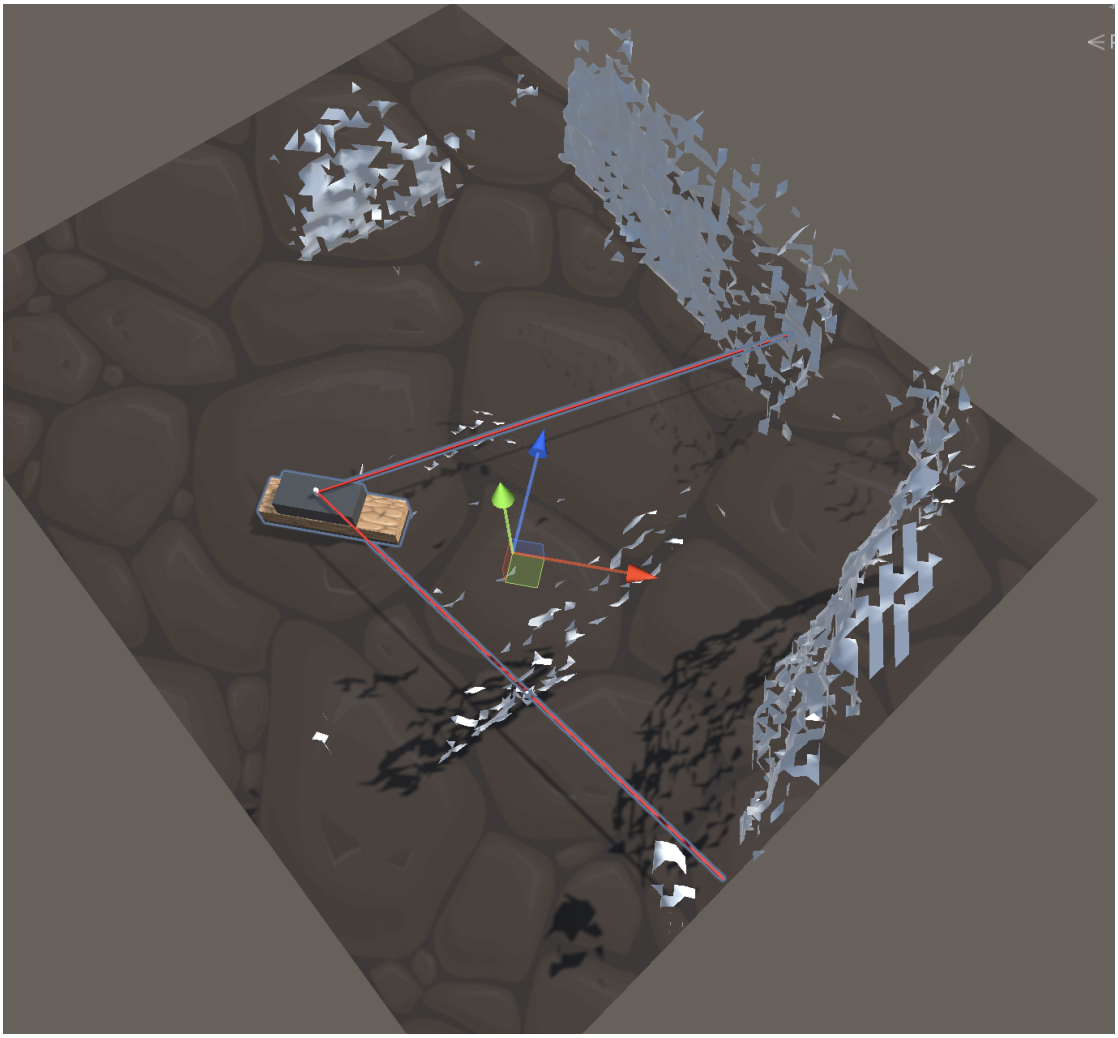

Figure 23. Scan done from the room with BPA algorithm.

7 Conclusion

The usage of LiDAR technologies varies quite largely and has become more prevalent in recent years. This thesis focused on investigating different ways to create a digital twin of the world based on the input from the LiDAR sensor. The donkey car and the VR device communicate over the network, filling the IoT category.

During the research, a literature review was done to look at the core issues that would come up when trying to achieving the goal of the thesis. The review is covered in section 2.2. It focused on finding different SLAM algorithms to track the car's position and different algorithms to model the mesh from the point cloud. From the SLAM algorithms, a lot of different algorithms were found. Of those algorithms, Hectorslam was chosen as an algorithm to implement because it was a LiDAR-only solution. This meant that I only needed input from a single sensor to test out the solution.

The research did not have enough related papers to conclude the result of generating the mesh from the point cloud. As extended research, I looked over multiple solutions to generate meshes and found a few new papers that contained information about it. This is summarized in section 2.3. In the final step, BPA and marching cubes algorithms were chosen as modeling algorithms to test out the whole flow. For collecting the LiDAR data, the Cygbot Cyglidar D1 sensor was used, and a Donkey Car S1 was the device that collected the data and processed the point cloud and the coordinates. The university provided the donkey car, and the sensor was bought separately.

Results show that the created flow takes some time to run the algorithms and adds delay when visualizing the results. The main quality assurance methods were speed and the quality of models (Generated models correspond to the actual world model). Future work based on the research may conclude the following points:

- Testing out multiple SLAM algorithms - This thesis focused on creating the main flow and finding solutions on how to do it. However, as seen from the research, there are a lot of different methods and ways to find out the position of the donkey car. Further research can be done to focus on testing out how other the algorithms would perform and fill the set criterias.

- Generating mesh clouds from the algorithms - Section 2 summarized different algorithms that can be used to generate meshes from the point clouds. Further research could be done to look into how to speed up the overall processing speed of the algorithm.

- Sensor fusion with the camera - Future research could be done to fuse the models with the input from the camera to achieve photogrammetric images.

There were four different requirements proposed at the beginning of the thesis regarding the requirements. The first requirement was finding a SLAM algorithm that could be

used in the donkey car. The result was found in the made literature review, and the chosen algorithm was Hectorslam. Furthermore, as the algorithm was fit for the set purpose, the requirement was fulfilled.

The second requirement covered the modeling algorithms that could be used to generate meshes from the point cloud. This requirement was not filled, as the algorithms that were tested did not provide meshes with quality that would make the objects distinguishable from each other. However, the Marching Cubes was quite a promising algorithm as it satisfied the RQ4, and in Section 7, an improved version of it was proposed to improve the quality to satisfy this requirement.

The third requirement required the flow from scanning the point cloud to having the scanned environment in the virtual world, where the user interacts in VR. This was not fulfilled as the modeling algorithms were too slow, and looking for improvements was prioritized over linking it with VR. It would have been fulfilled when the flow had been implemented to support the VR.

The fourth requirement required the algorithms to process the cloud in a reasonable amount of time. The input for the user should be pretty immediate to have a feeling that there is no real-time difference. For both algorithms (SLAM algorithm and modeling algorithm), the time was in the required timeframe, which was 2 seconds. However, the modeling algorithm was implemented to model the environment in a 3x3 area. It is not dynamic, and if we increased the area, it would increase the time spent on modelling the larger room, and it would go out of allowed time. I proposed an modification in the process in Section 7 to allow modeling to be dynamic.

A few things that were planned to do, but were skipped because of its scope, were the plans to increase the processing speed even further. The bottleneck in this thesis was mainly the processing speed of the modeling algorithms. Because of that, I thought of the idea of partially moving the processing from CPU to GPU. CPU cores are usually limited to a certain number (depending on the processor), whereas some household desktop computers might have up to 12 cores, whilst the GPU can have hundreds of cores. Therefore, I looked into how much the algorithm's performance would improve if moved from CPU to GPU. The article that I found compares the speed of the CPU and GPU when training the deep neural network models. It concluded that the results might improve up to 27 times. GPU that was used in the article's comparison was NVIDIA RTX 2080 Ti [22].

Using the compute shaders for this process would significantly increase the modeling speed. Unity has a wide range of support in the form of a broad community. When looking into this topic, I found a repository that had implemented the marching cubes algorithm that runs on the GPU. The goal of the project was to create a world using the Perlin noise. Meaning some of the ideas of that project can be used [1].

As the current scan renders about a 3x3 meter area, then to test it out, the idea would be to render the surrounding voxels (9x9 area around the car). Figure 24 shows the

idea of how to approach this. The blue-colored squares are the ones that are constantly updated and visible. The green squares are the squares where the data is stored, but they are not rendered. This would be the base for testing. If the algorithms perform well with the given test, then we could improve the quality of the scans by reducing the octomap grid size or rendering a larger area.

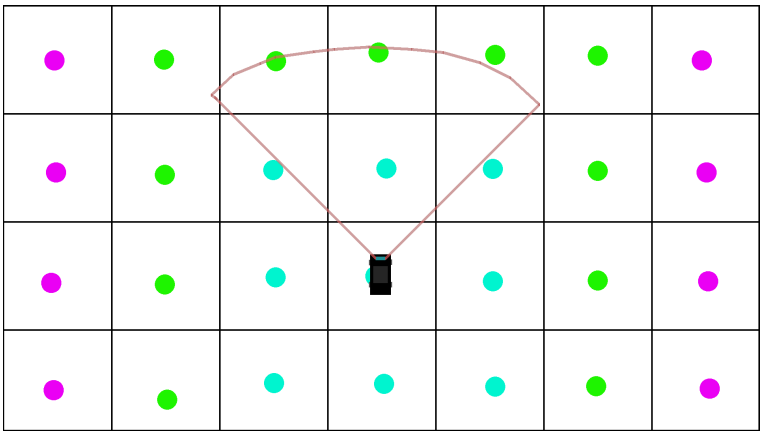

Figure 24. Rendering based on the scans.

In summary, I mainly achieved all goals of the thesis. There is now a solid foundation for continuing a deeper scope to explore it in different theses. Especially research on modeling the environment using the GPU as a processing unit. It will well complement this/my work which can be developed even further, which is available in the Donkey car [19] and VR [20] private repositories.

References

[1] Marching-cubes-on-the-gpu. https://github.com/Scrawk/Marching-Cubes-On-The-GPU, 2020. Accessed: 2022-05-15.

[2] The autonomous driving lab (adl). https://isa.ut.ee/blog/adl-tetiana-ukraine/, 2021. Accessed: 2022-05-15.

[3] Point cloud structure. http://docs.ros.org/en/api/sensor_msgs/html/msg/PointCloud2.html, 2022. Accessed: 2022-05-15.

[4] Apple. Apple unveils new ipad pro with breakthrough lidar scanner and brings trackpad support to ipados. https://www.apple.com/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/, 03 2020. Accessed: 2022-05-15.

[5] Fausto Bernardini, J. Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 5:349 – 359, 11 1999.

[6] https://www.alibaba.com/product-detail/Cygbot-CygLiDAR-D1-Small-Size-2D_10000001710595.html, 2022. Accessed: 2022-05-15.

[7] Cygbot. 2d/3d dual solid state tof lidar. https://www.cygbot.com/2d-3d-dual-solid-state-tof-lidar. Accessed: 2022-05-15.

[8] Cygbot. Cyglidar d1 user manual. https://cdn.sparkfun.com/assets/3/d/5/4/c/f5911d_726a54fc4f6644bcbec0d9b00236ffda.pdf, 2020. Accessed: 2022-05-15.

[9] Cygbot. Cyglidar d1 ros package. https://github.com/CygLiDAR-ROS/cyglidar_d1, 02 2022. Accessed: 2022-05-15.

[10] C. Domke and Q. Potts. Lidars for self-driving vehicles: a technological arms race. https://www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race/, 08 2020. Accessed: 2022-05-15.

[11] Deltax self-driving competition. https://courses.cs.ut.ee/t/DeltaXSelfDriving/. Accessed: 2022-05-15.

[12] VarHowto Editor. How to install ros noetic on raspberry pi 4, available at https://varhowto.com/install-ros-noetic-raspberry-pi-4/, 12 2022. Accessed: 2022-05-15.

[13] g2. Best virtual reality (vr) game engines. https://www.g2.com/categories/vr-game-engine?utf8=%E2%9C%93&order=g2_score, 09 2021. Accessed: 2022-05-15.

[14] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[15] HyeongRyeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60:1–9, 10 2015.

[16] S. Kohlbrecher and J. Meyer. Hectorslam. http://wiki.ros.org/hector_slam, 04 2014. Accessed: 2022-05-15.

[17] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[18] Raigo Kõvask. Survey of state of the art vr driving simulation for physical test car using lidar for mapping the surrounding environment. 01 2021.

[19] Raigo Kõvask. Car lidar repository. https://bitbucket.org/raigo132/car-lidar/src/master/, 05 2022. Accessed: 2022-05-15.

[20] Raigo Kõvask. Vr lidar repository. https://bitbucket.org/raigo132/vr-lidar/src/master/, 05 2022. Accessed: 2022-05-15.

[21] Sebastian Lague. Coding adventure: Marching cubes. https://www.youtube.com/watch?v=M3iI2l0ltbE, 05 2019. Accessed: 2022-05-15.

[22] Kevin C Lee. Parallel computing — upgrade your data science with gpu computing. https://towardsdatascience.com/parallel-computing-upgrade-your-data-science-with-a-gpu-bba1cc007c24, 08 2020. Accessed: 2022-05-15.

[23] Jingyun Liu, Qiao Sun, Zhe Fan, and Yudong Jia. Tof lidar development in autonomous vehicle. In *2018 IEEE 3rd Optoelectronics Global Conference (OGC)*, pages 185–190, 2018.

[24] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, aug 1987.

[25] Jonathan Mace. Rosbridge. http://wiki.ros.org/rosbridge_suite, 10 2017. Accessed: 2022-05-15.

[26] Abhik Maiti and Debashish Chakravarty. Performance analysis of different surface reconstruction algorithms for 3d reconstruction of outdoor objects from their digital images. *SpringerPlus*, 5:932, 06 2016.

[27] Jason M. O'Kane. *A Gentle Introduction to ROS*. Independently published, October 2013. Available at http://www.cse.sc.edu/~jokane/agitr/, Accessed: 2022-05-15.

[28] Renge-Games. Unitybpa. https://github.com/Renge-Games/UnityBPA, 2021. Accessed: 2022-05-15.

[29] Docker ros packages, available at https://hub.docker.com/_/ros, 2022. Accessed: 2022-05-15.

[30] Nick Shelton. Lidar processing. https://nshelton.github.io/home/lidar_processing/, 03 2016. Accessed: 2022-05-15.

[31] Sparkfun. Cygbot CygLiDAR D1. https://www.sparkfun.com/products/18580. Accessed: 2022-05-15.

[32] Lauri Varjo. Virtual reality walkthrough of laser scanned environments. *Tampere University*, page 26, 05 2019.

[33] Lance Whitney. The best lidar apps for your iphone and ipad, available at https://www.pcmag.com/how-to/the-best-lidar-apps-for-your-iphone-12-pro-or-ipad-pro, 01 2021. Accessed: 2022-05-15.

Appendix

I. Glossary

LiDAR - Stands for Light Detection and Ranging. It is a method for calculating distances to an object with a laser and measuring the time for the reflected light to return to the receiver.

SLAM - Stands for Simultaneous Localization and Mapping. It is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's(donkey car) location within it.

VR - Virtual reality(VR) is a computer-generated environment with scenes and objects that appear to be real, making the user feel like they are inside of the environment.

BPA - Ball-Pivoting Algorithm(BPA). One of the algorithms used for generating the mesh. It is described in Section 3.3.2.

ROS - Robot Operating System(ROS) is an open-source tool containing a set of different libraries and tools that can build robot applications.
Vertex - Vertex is a point in the virtual world where the single or multiple edges can connect.

Edge - Edge is a line between two different vertices.

Face - Face in the context of this thesis is a flat surface that is defined by three vertices.

Mesh - Mesh in the context of this thesis is a collection of vertices, edges and faces that defines the shape of a object.

CPU - Central processing unit (CPU) of the computer.

GPU - Graphics processing unit (GPU). A specialized processor designed to accelerate graphics rendering.

UART - Universal Asynchronous Receiver-Transmitter. It is a computer hardware device for asynchronous serial communication.

Low poly - Polygon mesh in 3D computer graphics that has a relatively small number of faces.

Shader - A program that runs on the GPU.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Raigo Kõvask**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   VR Driving Simulation for Physical Test Car Using LiDAR for Mapping the Surrounding Environment,
   (title of thesis)

   supervised by Ulrich Norbisrath.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Raigo Kõvask
17/05/2022