Marcus Olai Grindvik

Bachelor's thesis

Snake-like robot using LiDAR

December 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Bachelor's thesis

2020

NTNU
Norwegian University of
Science and Technology

Marcus Olai Grindvik

Snake-like robot using LiDAR

Bachelor's thesis
December 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Snake-like robot using LiDAR

Marcus Olai Grindvik

December 2020

PROJECT / BACHELOR THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor 1: Ottar L. Osen

Supervisor 2: Guoyuan Li

Preface

This thesis is written by a student at Automatiseringsteknikk at NTNU Ålesund, who has a certificate of apprenticeship as an electrician with 7 years of experience working in the field as a ship electrician.

The goal for this project is to make an autonomous snake-like robot that can navigate tight spaces and be less dependent on a stable environment than the last version. There are several challenges that will need to be considered when trying to localize and map while moving with such a robot.

Acknowledgement

I would like to give my thanks:

- My supervisors for all the help and guidance trough the project
- Friends for supporting me through the semester
- Anders Sætersmoen for supplying parts

Summary

This report has taken a robot that was made for a theoretical search and rescue mission and made it less dependent on a stable environment. To implement mapping and navigation ROS was used to ease the process. To give the navigation vision and optometry a LiDAR and IMU was used to help the robot understand its surroundings. This was done to see how it could be implemented on a robot that has challenges, with such sensors because of its constant movement of different joints of its body. The result shows that the snake could have a LiDAR on it and use that to navigate even though there were challenges with it. There were issues with having to few landmarks for the LiDAR to always recognize where it where, but the project showed great promise and could work very well with some adjustments.

Contents

Terminology

Python Programming language

Node A single module unit used by ROS

Rivz Visualization software for ROS

Abbreviations

LiDaR Light Detection and Ranging

SLAM Simultaneous Localization and Mapping

UDP User Datagram Protocol

TCP Transmission Control Protocol

IP Internet Protocol

ROS Robot Operating System

I2C Inter-Integrated Circuit

IMU Inertial Mesurement Unit

GUI Graphical User Interface

PWM Pulse-Width Modulation

List of Figures

List of Tables

Chapter 1

Introduction

This report will step-by-step go through the process of the thesis from the theoretical to the practical.

1.1 Background

The snake-like robot was originally a project that worked around the theoretical scenario where it was meant to search through rubble after an earthquake or similar. The project was successfully done during an elective course called Mechatronics. There were several challenges still left when the project was finished, but showed great potential in its state. This project is a continuation of this project and will focus on making it less dependent on stable surroundings as well as more autonomous with a greater focus on ease of use.

1.2 Problem formulation

The problem present in this project is to upgrade the snake-like robot and make it able to move more freely and be less dependent on a stable environment. The project will use ROS and implement a LiDAR to map and navigate.

Problems to be solved

- Implement mapping using a LiDAR

- Make the mapping stable
- Navigate an environment

1.3 Limitations

There are several limitations in this project. One of the big ones is that because the LiDAR in use is 2 dimensional, the objects need to be visible in the same height as the robot itself. The LiDAR also has a limited range of 4m as well as 260 °scope.

1.4 Requirements

The project requirements are as follows

- Make the robot move with ROS
- Autonomously navigate rom's
- Avoid Obstacles
- Object recognition

1.5 Structure of the Report

The structure of the report is as follows

Chapter 2 - Theoretical Basis: Chapter 2 gives an introduction to the theoretical basis for the system and parts used in the project.

Chapter 3 - Method: Chapter 3 Considers the different methods that can be used to reach the goal of the project.

Chapter 4 - Materials: Chapter 4 is a review of the materials used in the project.

Chapter 5 - Testing: Chapter 5 describes the test that are desired for the project.

Chapter 6 - Result: Chapter 6 presents the result of the project and what solutions was taken. The result of test is also presented here.

Chapter 7 - Discussion: Chapter 7 includes a discussion of the result in chapter 6. It also discusses my personal experience.

Chapter 8 - Conclusion: Chapter 8 draw a final conclusion for the project.

Chapter 2

Theoretical Basis

This chapter will present the necessary theoretical information needed to understand the methodology in this thesis. The structure of the chapter is to go through the topics in a natural progression.

2.1 SnakeLikeRobot

This project is a continued version of a mecatronic class project that was made in to a scientific paper. This robot was made for a theoretical search and rescue mission, it was used a overhead camera for it to search trough a maze. This was done by using image processing and an algorithm was used to find a path. The paper can be read at [2].

2.2 Servo Motor

A servo motor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration [22]. Some servos use a signal called a PWM (Puls-Width modulation) signal that controls the position of the output shaft. The controller line does no supply power to the motor directly but is used as an input to a controller chip inside the servo. A servo is a closed-loop servomechanism that uses position feedback to control its motion and final position [11].

2.3 Lidar

LIDAR, which stands for Light Detection and Ranging, is a remote sensing method that uses light from a pulsed laser to measure ranges(variable distances) to the Earth. It uses a laser radar to transmit light pulse and a receiver with sensitive detectors to measure the reflected light. The distance to the object or landmark is determined by recording the time between transmitted and back scattered by using the speed of light to calculate the distance traveled. [20] In figure 2.1 it can be seen a diagram of how a LiDAR can operate.

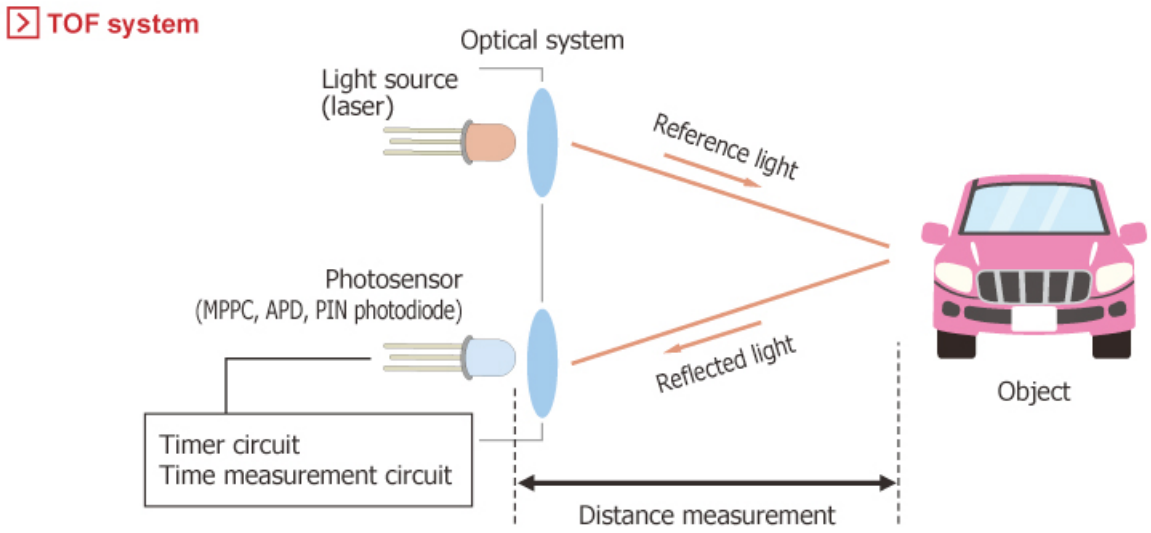


Figure 2.1: How a LiDAR works[12]

2.4 ROS

ROS is an open-source meta-operating system for your robot. That runs on Unix-based platforms. ROS provides tools and libraries for obtaining, building, writing and running code across multiple computers[7]. ROS is built to serve as a common platform for people to share code and ideas more rapidly. It also makes it so that users do not have to spend years writing software infrastructure before the robot starts moving. ROS makes it easier to make a robot and saves

time, all the code that normally would have been written can now be taken in as libraries and be adjusted by prams and scripts to fit a certain system[19].

ROS uses nodes and streaming topics, one node can for an example be Translater node. This node will then be responsible for translating information, and then publish it so other nodes can subscribe and use the data [8]. ROS uses a simplified messages description language for describing the messages that is published by nodes. This makes it possible to generate source code message type in several targeted languages. This simplifies the communication because the nodes have predefined what type of information they want to receive on a given topic[13].

2.5 Localizing

Localizing is an important part in the step to make autonomous mobile robots. To know where to move the robot it needs to know where it is. When localizing locally it is not needed to have a relation to a world map its own map is usually enough. Localization locally is often done together with mapping in an SLAM algorithm. To get the robot to know where it is it needs a way to understand what the environment around it looks like this is done with sensors, either on board or external sensors. That can help the robot gather data about its surroundings.

2.6 Path Planning

Path planing is to find a path to walk, drive, fly, roll etc. to an location. This is done by gathering data about the surroundings, mapping what is around the robot and localizing the robot. When the robot has knowledge about the environments it can use algorithms to make a way to get to a certain point. If there is a obstacle in front of the robot it needs to detect that and make a path that goes around the obstacle, or over the obstacle if the robot is built to handle that better. This is all decisions that needs to be taken in to account when planning a path.

ROS sends out two paths; One is a Global Plan and the second is a Local Plan. The Global planer uses a algorithm to plan the path for the robot to walk, this can be done by Dijksta's

algorithm[18]. The Local provides a controller that serves to connect the path to the robot. The local planers job is to follow the path made by the global plan and check if there is obstacles and if so it makes a path around that[6].

2.7 SLAM

For robots to navigate it needs some understanding of its surrounding, and its location in position of those surroundings. SLAM is used to simultaneously locate and map by gathering sensor data. This makes it possible for the robot to be able to be put in a new location and still be able to navigate trough terrain using the map that gets made from SLAM [4]. An Extended Kalman Filter is often the heart of the SLAM process. It is responsible for updating where the robot thinks it is based on features. These features are based on landmarks. These landmarks are features that can be re-observed and can be distinguished from the environment [23]. When a SLAM map can look like can be seen in figure 2.2.

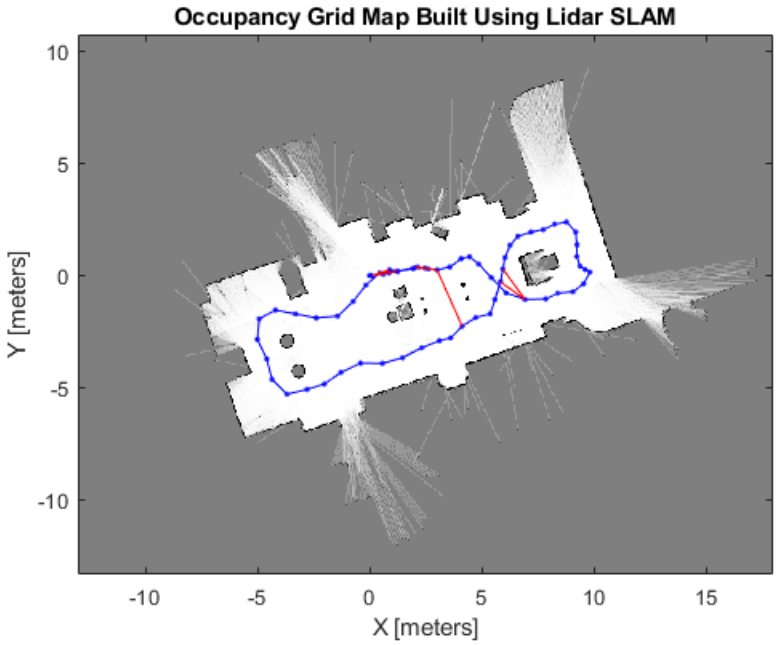


Figure 2.2: Picture of a slam map[15]

2.8 IMU

The robot needs to know its orientation. The most common sensor for this task is to use an IMU. An IMU is mainly build up of an accelerometer and gyroscope, this gives the data for 6 Degrees of freedom. The 6 axsis as its called is Ax, Ay, Az and Gx, Gy and Gz this make it possible to get out roll and pitch which helps find the orientation of the robot. If yaw is required the IMU needs to have a magnetometer. This is used to sense the earths magnetic field and is used with gyroscope to find absolute heading[21]. The last and the tenth axis is a pressure sensor that will give altitude. [10] The six degrees of freedom can be seen in picture 2.3

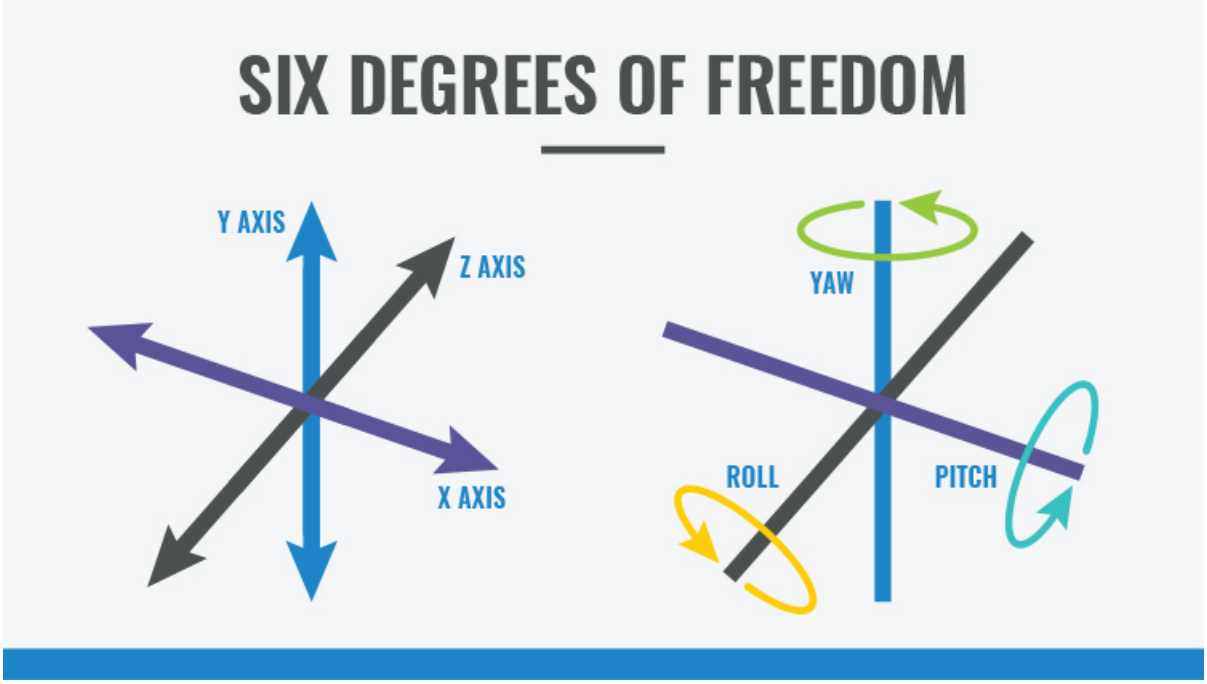


Figure 2.3: 6 Deg of freedom[21]

Chapter 3

Method

This chapter provides an explanation of how the project was organized and handled. It will also present how the different problems and challenges in the project might be handled.

3.1 Project Organization

This project was organized around the corona pandemic. While talking with the supervisors, Ottar L. Osen and Guoyuan Li it was recommended that the project was done from home to the extent that was possible. Because of this the project has been done from home. The project was set up to have report writing along side making functions for the robot.

3.2 Software

There are several software utilized in this project, listed in table 3.1

Program	**Description**
Arduino IDE	Arduino IDE is and IDE used for writing Arduino-code
PyCharm	Pycharm is and IDE used for writing Python-code

Table 3.1: List of software

3.3 Concept Studies

The snake-like robot is a concept that has been used for research. NTNU got a breakthrough in 2016 where they made a snake-like robot that could go underwater and showed the practical applications of such a robot. A company has now taken this concept to the next stage, and Equinor is now going to use them in the oil industry. [5] The snake used in this project was inspired heavily by Houxiang Zhang's snake-like robot. As it can be seen in figure 3.1.

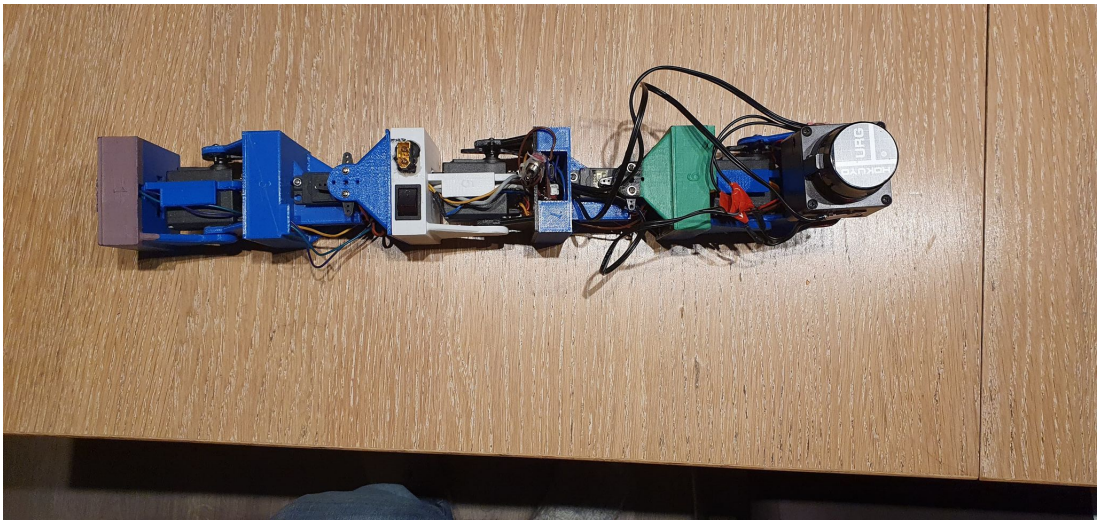


Figure 3.1: The snake like robot form a top view

The snake-like robot was already made. But the concept of the snake is to have a robot that can walk around in different terrains. This kind of robot moves its joint in angels in a sinus movement to walk forwards.

3.4 Design

3.4.1 Concept

The snake-like robot was already made. But the concept of the snake is to have a robot that can walk around in different terrains. This kind of robot moves its joint in angels in a sinus movement to walk forwards. The concept makes it possible to make a small robot that can get trough tight places and move over obstacles.

3.4.2 3D-Modeling

To design such a robot 3D-modeling is one of the strongest tools available. 3D-modeling has become one of the best ways to make parts that was just a thought into something real in a fast and efficient way. This makes it good for fast-prototyping. When this project first was taken on there was multiple ideas about how the snakelike-robot should be made. There was multiple revisions that was tried before the final concept which is how the snake is made today. The snake like robot still has many aspects that could be improved, to make it more reliable and practical.

3.5 Data

3.5.1 Data Acquisition

The acquisition of data can be made possible using a variety of techniques. Data from vision on the robot can be gathered either by having a cable connection to the computer. The data could also be taken in to an micro-controller to get calculated to the degree possible, then send to the PC that needs it. To have everything go trough a micro controller and get send by wireless have the advantages of not needing cables to the robot, but it can be limited how fast the data could be send.

3.5.2 Processing data

To process the data some different techniques can be used. It could be done on a micro-controller to process the data. This has some disadvantages, one of the disadvantages is that they usually have limited processing power. An esp32-WiFi has a clock frequency of up to 240MHz with a Xtensa dual-core 32-bit LX6 microprocessor [1] this makes the esp-32 decent for processing data. The other option for processing data is to get it to a computer and process it there, most computers has strong CPU to compute data at a high speed. There is also the option on a PC to use the GPU to process data this is a good option if there is many small process that needs to be done, like for machine learning where there are multiple small calculations that is needed to be done.

3.6 Sensors

There are several sensors that can be useful for a robot of this kind to navigate a given area. There has to be some kind of vision as well as some detection for orientation and translation. There are several sensors that can be used to be able to give the robot some sort of visions;

Sonic sensor could have been used. They can be found in all price ranges, and are often used as proximity sensors in self driving cars for an example [17]. It could be used for the robot to know the distance to a object.

LiDAR is also a kind of sensor that could be used. A LiDAR could be used to measure distances in a certain angle around it self, and is would give the robot a good feedback of its surrounding. LiDAR is a valid option for a project like this.

Depth camera could have been used. A depth camera uses a point cloud to extract distances to objects in its field of view. This can be used to give the robot perception of distance and data for image processing.

To estimate translation the most common way is to calculate from a wheel encoder or such. This could not be done with the kind of robot that is in this project because of the lack of wheels. Translation data can also be calculated from vision sensors such as LiDAR or depth camera.

To measure the robots orientation an IMU can be used. The IMU consist of several different sensors, witch can be combined in order to calculate the orientation. IMU's are described in 2.8. The three axes of the IMU's accelerometer can be used to calculate roll and pitch, but is affected by acceleration and will be imprecise when the robot is moving. The gyroscope measures the rate of change of the orientation and is precise even when the robot is moving. Orientation can be calculated by integrating the gyroscope data, but as a result suffers from drifting due to small measurement errors. One can combine the orientation calculated by accelerometer and

gyroscope data by using a complimentary filter.

$$\theta = \alpha \cdot \theta_G + (1 - \alpha) \cdot \theta_A \tag{3.1}$$

Where:

α - Filter Coefficient

θ_G - Orientation measured by gyro

θ_A - Orientation measured by accelerometer

This results in the accelerometer orientation being low pass filtered while the gyroscope data is high pass filtered. This reduces the issues with the individual sensors.

3.7 Servo Controller

To control a servo all that is needed is something that can generate a PWM signal. A PWM signal can be generated by and micro controller for an example. This could be done with an ESP32 by using its 16 independent channels that can be used to generate PWM signals. Other micro controllers could also be an option, there are multiple micro-controllers with PWM pins.

3.8 Mapping and Localization

To be able to map the surroundings the robot needs to be able to know what the surrounding look like and to know the localization of itself relative to the surroundings. The most common method of doing this is to do it simultaneous with SLAM. You can read more a bout SLAM in 2.7. In ROS hector_slam does the mapping and localization and presents the data in a good way. Every thing can be represented in a GUI called rviz, with makes it possible to choose what data is wanted to be represented and to some degree how to display it. The rviz GUI can be seen in fig 3.2.

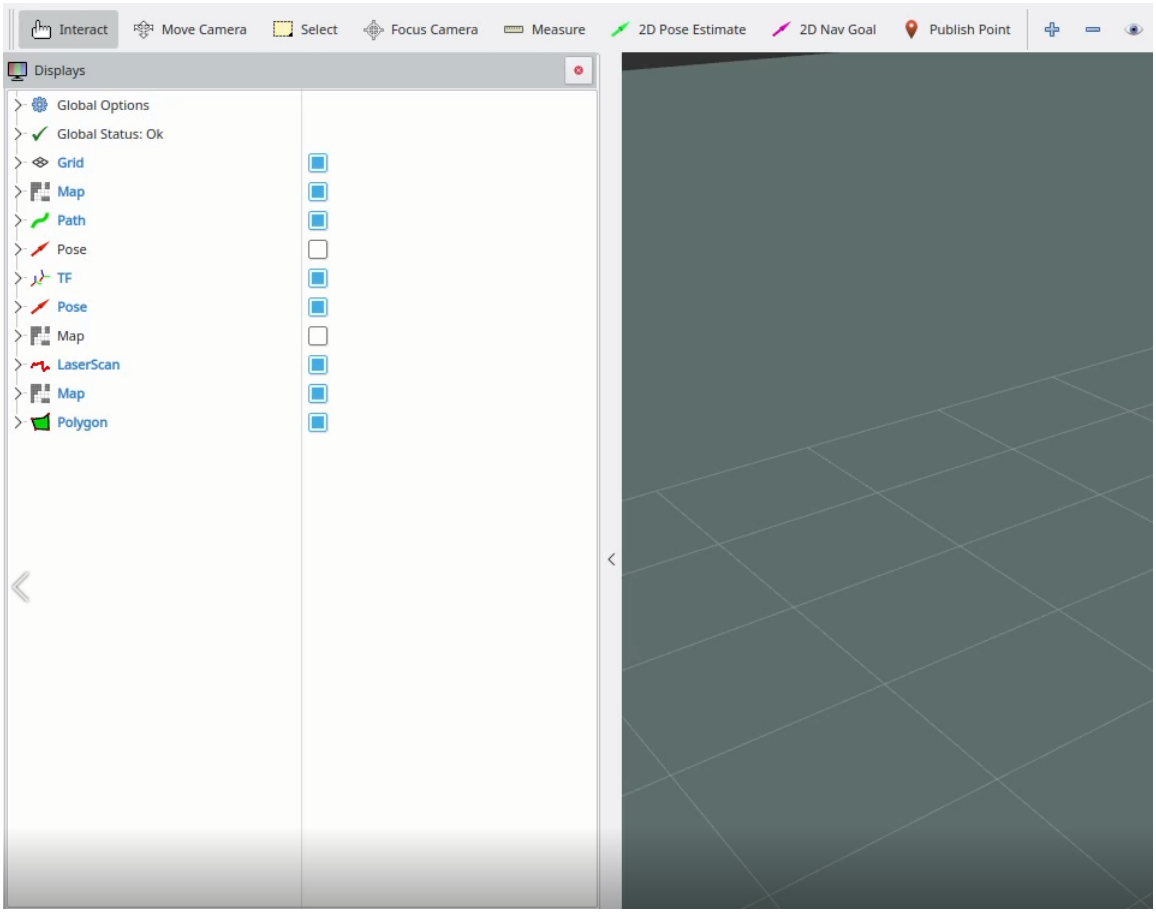


Figure 3.2: Rviz GUI layout

3.9 Navigation

To navigate there are multiple ways to approach; One of the options is to blindly go forwards until the robot can detect an objects in front of it and then change way. This is a simple but not very efficient way to navigate. This would be the same as the early generations of roomba's worked[16]. To be able to navigate efficiently it helps to map surroundings and localization with algorithms such as SLAM. Ros has multiple libraries for navigation, in the navigation stack in ROS there is one called move_base. This gives the option to either set a goal manually on the map made from SLAM or send in a goal that's collected from data [9].

Chapter 4

Materials

This chapter presents what materials and components are used for the physical part of the project. The electrical components will detail what their function is. For the construction parts themselves it will shortly detail what their use is.

4.1 Electrical system

The electrical system was already built. But there was added cables for connection of a IMU and a cable to connect the LiDAR to the system.

4.1.1 Computer for Processing

The computer in use consists of the parts that can be seen in table 4.1.

Parts	Description
CPU	Intel(R) i9-9900K CPU @ 3.60 GHz
GPU	NVIDIA GeForce GTX 1060 6GB
Memory	16GB DDR4

Table 4.1: "Specs of computer used"

4.1.2 Components

The components can be seen in the table 4.2.

Old or New	Object	Located	Function	Quantity
Old	Camera	In the head of the snake	None	1
New	Lidar	On top of the snakes head	Location and mapping	1
New	IMU	Located on each side of Lidar	Used to filter Lidar data	1
Old	Esp32-wifi	Located in the third led counted form the front	Used to move the snake and send sensor data to computer	1
Old	Battery charger	Located in the rear end of the snake	Used to charge the batterys	1
Old	Battery	Located in the rear led's	Not in use	4
Old	On/off switch	Located in the forth led counted from the front.	Used to turn the snake on and off	1
New	Arduino	Located in the second led counted from the front.	Used for sending IMU data	1

Table 4.2: SnakeParts

IMU

The IMU's are LSM9DS1 it can be seen in figure4.1. This is one of SparkFun's products. This has as the name suggests 9 axsis of reading data.

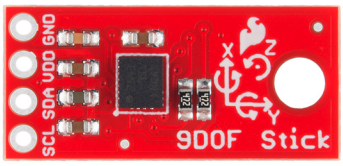


Figure 4.1: Picture of the IMU.[21]

LiDAR

The LiDAR is a HOKUYO URG-04LX-UG01 witch can be seen in figure 4.2. The LiDAR has 240°range in front of it. The max range on the LiDAR is 4095mm with an accuracy of +-3% but on lower ranges as 1000mm and down to 60mm it has an accuracy of +- 30 mm. A scan cycle takes 100ms.[14]

Figure 4.2: Picture of the LiDAR.[14]

Battery

There are four battery's on 2200Ah that is connected in parallel inside the snake thees powered the snake like robot before it was changed to a external power-supply.

4.2 Construction

The construction of the snake-like robot itself was already finished. For this project the prototype was mostly done, but was meant to be modified to be able to accomplish the new tasks given. The modifications that was needed was to put on a sensor for vision on the robot, and a sensor to know know the translation and orientation of the robot. The LiDAR was decided to be mounted on the head of the snake this was the most practical place to mount it. There was also added a IMU that was mounted on top of the LiDAR. How the snake looked like as a 3D rendering can be seen in fig 4.3.

Figure 4.3: The snake-like robot without its LiDAR and equipment

Chapter 5

Testing

5.1 Design

The design itself did not need any testing for the most part. The testing needed was to see that the modifications made were robust enough to handle the snakes rough movement and to see if it was possible to get data from the LiDAR under the rough movement from the snakelike-robot. The way it was decided to test this was by putting the LiDAR on to the robot and see how the data points got.

5.2 Testing of Navigation Stack

There are two steps for testing of the navigation stack that is planned, the first one was to test the navigation stack without the LiDAR attached to the robot see how it worked and how the path would look, and if it would change if there was put an object in the path. The next step was to do the same procedure but while the LiDAR was on the head of the robot. These test was done to understand how the navigation stack was working and how to use it.

5.3 Performance Tests

When choosing a performance test it was decided to do it in multiple steps to see what degree of impact it would make for the end result. The end goal is to get the best and most stable map

possible. First performance test would be without the LiDAR attached to the robot. The next step would be to test the robot with the LiDAR on the robot, but without IMU data and filtering. The last step would be with all data and filtering as well as using this data for odometry. These test will be done to see the improvement between the different implementations.

Chapter 6

Result

6.1 Physical Design

The final physical design of the robot can be seen in figure 6.1, this is the snake's rendered cad file. This was done in the last project. The dimensions of the snake like robot is as seen in 6.2 and the debt can be seen 6.3.

Figure 6.1: The snake-like robot without its LiDAR and equipment

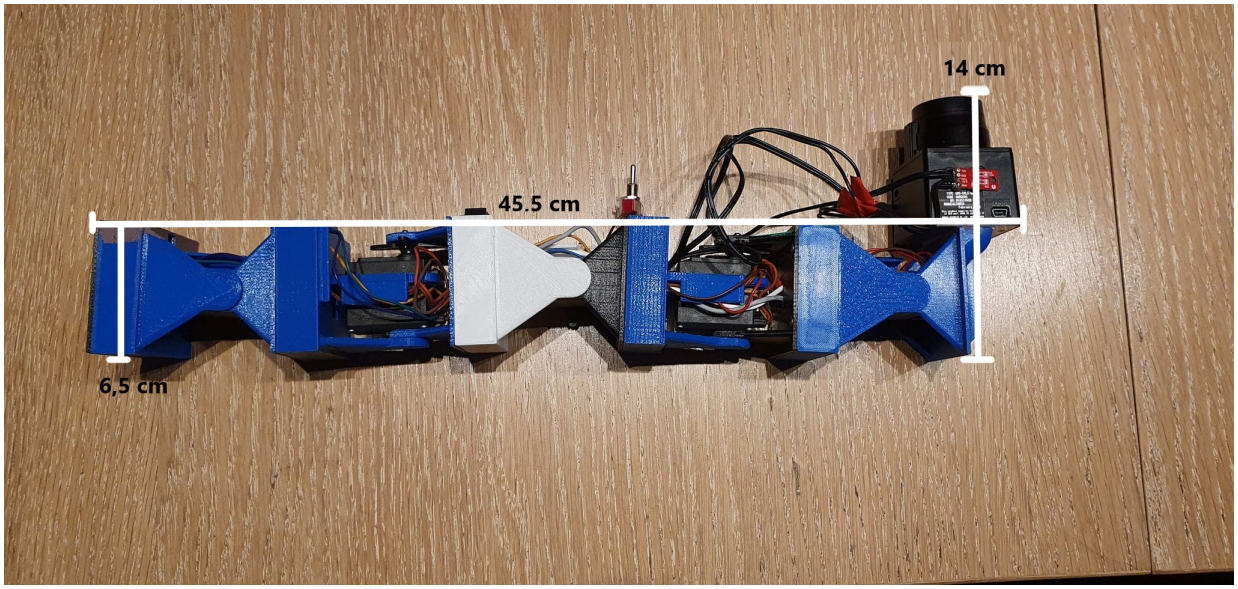


Figure 6.2: The snake-like robot side view with dimensions

Figure 6.3: The snake-like robot front view with dimensions

The snake ended up looking like 6.4 as it can be seen in the picture the LiDAR is mounted on the snakes head. This was to make the snake have the ability to see in front of itself and not see itself while walking. The LiDAR was mounted on with hot glue, this was chosen to not damage the LiDAR. It could have been chosen to make a 3D printed head with more properly made attachment unit for the LiDAR. This was not chosen size it was recommended to be at school at a minimum degree. There was also mounted a IMU on the top of the LiDAR this was also fastened with hot glue.

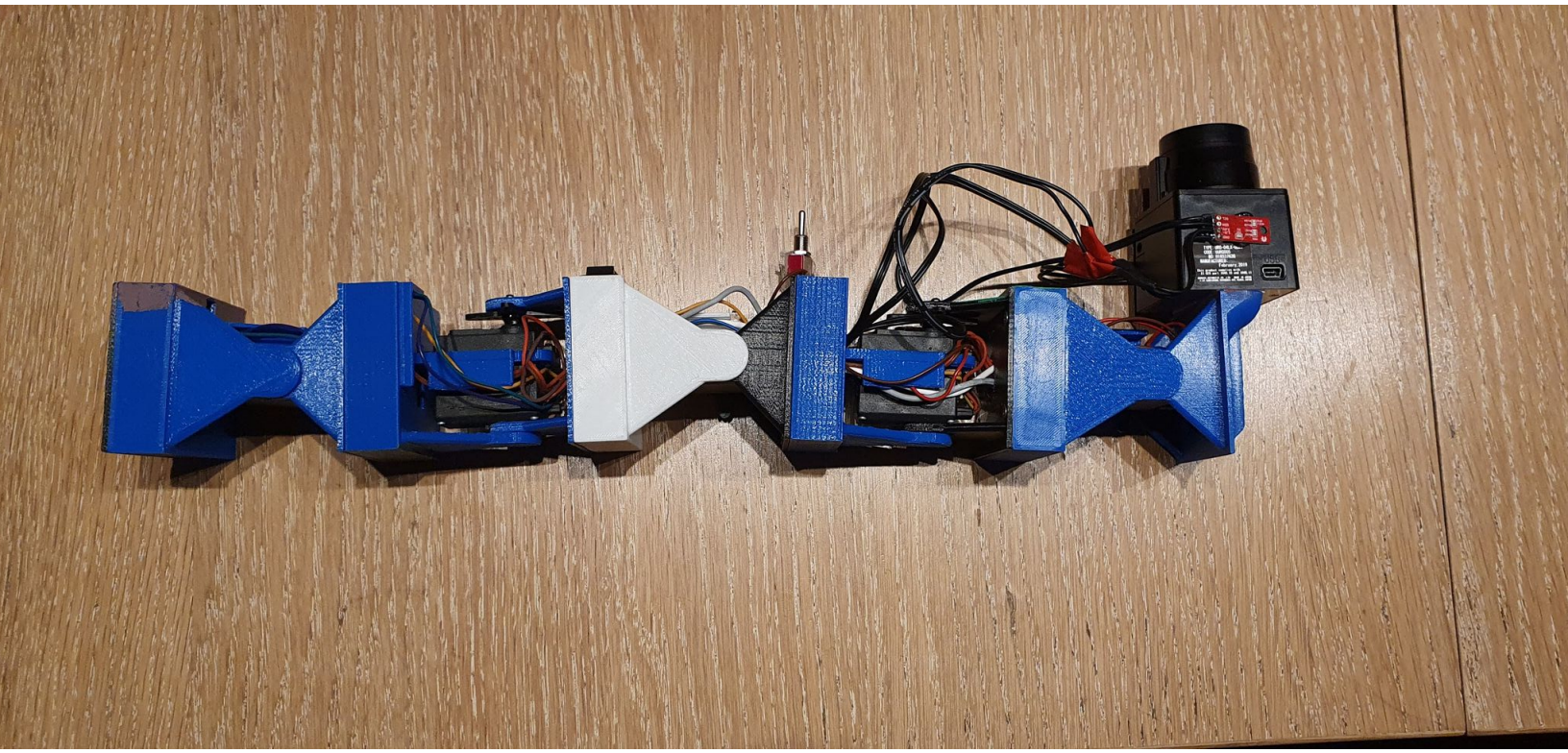


Figure 6.4: The snake-like robot side view with equipment

6.2 Electrical Design

The snake's electrical design was mostly done in the previous project but there has been challenges with the prototype. There was something that's called brownout error that occurred, the troubleshooting can be read about in section 6.6.2.

The sensors that was added to the electrical system was a IMU and a LiDAR. The IMU was first put on the side of the LiDAR and connected to the esp32, but after some trouble shooting it was changed. There was added a arduino to take care of the IMU's data and sending it forward. The reason can be read about in section 6.6.2. The IMU was sett on the top of the LiDAR with the new settup.

The battery's in the snake was not sufficient enough for power, so they were disconnected. Instead it was used a external power supply, first it was used a old PC power-supply. This worked

great for a couple of Weeks until it had some sparks flying out, after it was opened it was found out that one of the parts was defective. It was then changed to a old laptop charger that gave out 19 volt and 4,86 amps. The voltage was bucked down with a buck converter.

The buck-converter got hot, so it was decided to use the fan from the PC power-supply to cool it down. Since this fan needed 12 volt it was added one more old laptop charger that gave out 12 voltage to supply the fan. It can be seen in picture 6.5.

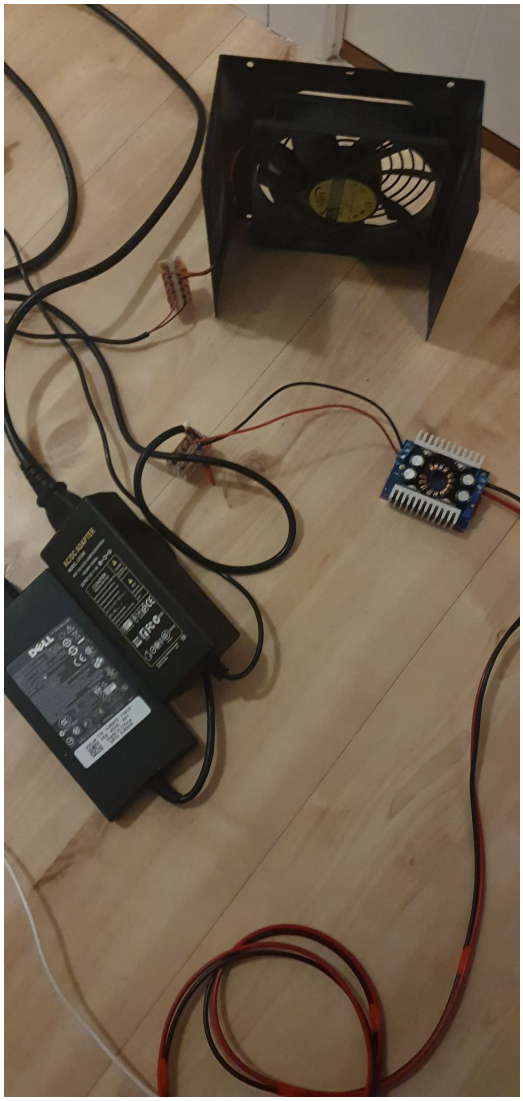


Figure 6.5: The power-supply that was used

6.3 Software

The robot was built up of different modules that had their own tasks. The arduino program had the responsibility to send IMU info when asked for it. The Esp32 code had the task to control the servo's and send an acknowledge when done. The translaters job was to translate where the navigation stack said to go. The IMU_node had the task of requesting IMU information when needed and publish that on a topic for other ROS nodes. The Loggers job was to logg when doing test's, and this script was adjusted when different data was needed to be logged. The logger saved the loggs with a time stamp and the data in a text file. To read this text file and get the data in a readeble way it was used the logg_viewer which made it possible to view it as graphs and adjust the data and what data to see in the graph.

6.3.1 System overview

This section will give a brief overview of how the different parts of the program are connected.

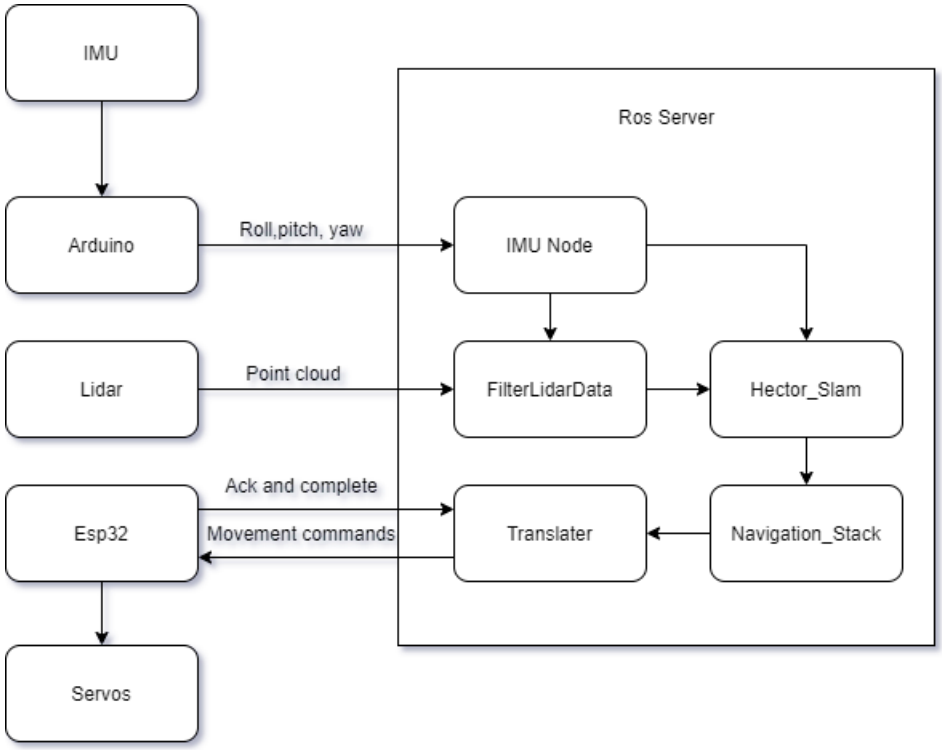


Figure 6.6: Simplified flow chart of the system

The system is divided in to five parts and each part with a specific task. The arduino has one

task and that's to send IMU data when needed, this is done by sending an request for IMU data over serial connection to the main computer trough the IMU_Node node. The Esp32 is responsible for moving the robot as specified by the translater, it receives commands for specific types of movements and translates this in to the servos.

The translater takes the commands from the navigation stack about where to walk, and translates these commands into something the esp32 can understand. It also checks if the esp32 is finished with its last task before sending a new one.

The navigation stack gets its input data from hector_slam and the LiDAR to determine where the robot should move to reach its destination. Hector_slam uses LiDAR and odometry data, the LiDAR data is gathered from the FilterLidarData node while odometry data is published by the IMU_Node node.

The FilterLidarData takes in LiDAR data from the LiDAR, and takes in IMU data from the IMU_Node node and uses the IMU data to pass on all scans while the pitch of the LiDAR is between +10°and -10°to hector_slam. The system is illustrated in figure 6.6.

IMU

The IMU ended up being on its own arduino, the code for this was built up to send data every time it got an request. This made it possible to make an python script that would ask for data in a given interval. The data that could be asked for was both raw data and calculated data as roll, pitch and yaw. The IMU_Node was a script that requested IMU data from the arduino and published it so it would be available for other nodes in ROS.

LiDAR

The LiDAR data was taken in by ROS, to do this it was implemented a library for the LiDAR called Urge_node. This was used to take in the LiDAR data and publish it as sensor data with the topic Scan in ROS. In the early stages of the project this data would be put right in to SLAM. It was early discovered that this did not work well because of the snakes movement, this was also pre-

dicted.

To fix this it was made a filter for the LiDAR data. The filters only function is to filter out all data that is gathered when the snake doesn't have its head tilted more than +-10°, this is based on where the laser from the LiDAR would have been parallel with the floor in approximation. This was done by a script called FilterLidarData, that subscribed on pitch data and the LiDAR data and publishing the result.

Translator

The translator is built up by parts of the old python script that was used to communicate with the snake, and a lot of new code. The main point of the translater was to do all the communication with the esp32. This was how it was done, until it was discovered that the esp couldn't handle to send the IMU data while running the servos. This can be read more about in 6.6.2.

6.3.2 Communication

ESP to computer

The communication protocols that was used to send data between esp32 and the computer was UDP connection over Wifi. The data that needed to be transmitted from the snake was a character and a float. The characters was something that was used to get the snake to send acknowledgments and done messages in the previous project.

Now it was needed to also send float for pitch data, since the UDP library only supports sending data as a byte, it was needed to use union to convert a float into a single byte. A float is four bytes in arduino[3]. The esp32 sends the byte's as little endian this is something that needs to be taken in to account when making the script to translate this information.

The script to translate this information is called Translater, This was designed to do all the talking to the esp32 and give the the information to deliver the information to where ever it is needed. The translater script reads the hex that gets delivered form the esp32, and translates

them to chars and floats. If the character is a "p", the script will know what comes next is a float. The characters that can be sent from the esp32 can be seen in table6.1. The characters that can be sent to the Esp32 is 6.2.

Character	Command
p	Float is next
a	Acknowledge
d	Done

Table 6.1: List of commands that can be send from ESP

Character	Command	Is implemented in python-script
f	Going forward	Yes
b	Going backwards	Yes
v	Adjusting left	Yes
h	Adjusting Right	Yes
m	Rotating CW	Yes
n	Rotating CCW	Yes
s	Stopping movement	Not needed
r	Adjusting straight	No
t	Change Turn-angle	No
p	Change T-parameter	No
a	Change A-parameter	No

Table 6.2: List of commands that can be send to ESP

Inside the computer

The project is build on ROS which implements nodes which can subscribe and publish data, this is used to send data between scripts and nodes. In each node its needed to subscribe on data, it gets initiated a subscriber and a publisher.

6.3.3 Sensors and Sensor Processing

The sensor data that is used in this project from a LiDAR and from a IMU. The LiDAR gathers a point cloud of the environment and sends this data to hector_slam for processing. The IMU

is used to gather odomotry data. The IMU's data get processed by the arduino, this is done by using a formula that takes in the different axes of IMU data to calculate roll, pitch and yaw. This can be read more about in 3.6. The formula used for pitch is

$$\theta = \alpha \cdot \theta_G + (1 - \alpha) \cdot \theta_A \tag{6.1}$$

Where:

α - Filter Coefficient

θ_G - Pitch measured by gyro

θ_A - Pitch measured by accelerometer

This creates a complimentary filter. The roll(ϕ) and yaw(ψ) is calculated the same way. The value of 0.9 was found to be suitable for the filter coefficient α.

6.3.4 Navigation and SLAM

The navigation and localization was done by sending data into Hector_slam in ROS. Hector_slam would use the LiDAR data to make a map and localizing itself. This map was used to navigate trough the room. To make the LiDAR data for navigation and SLAM more stable and have no problems with seeing the floor when the snake was moving it was added a filter that was based on the pitch data from the snakes IMU that was on its head. The map that was before the pitch data would look like 6.7 at this example it got a bad map because it was looking down into the floor.

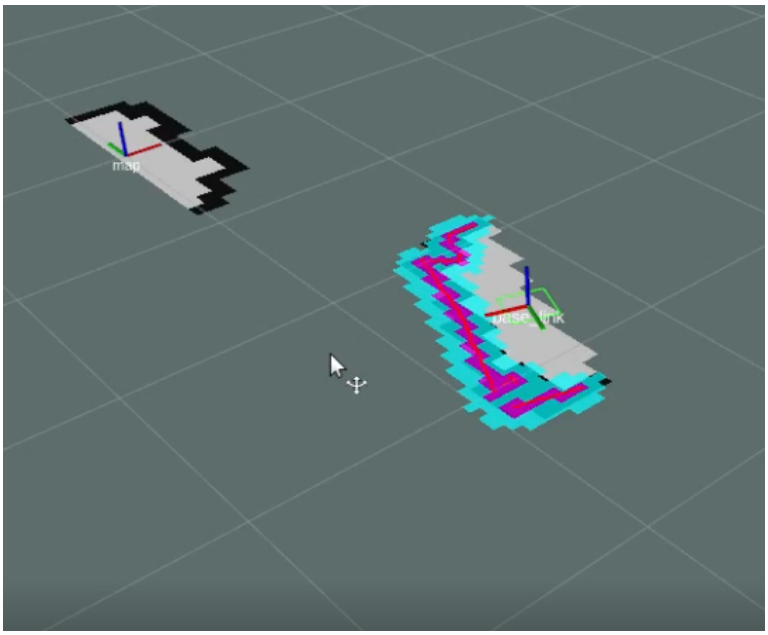


Figure 6.7: Map without pitch data

It could not understand that this data was not representative of the room. After the filter was added it looked like 6.8, this was a good improvement.

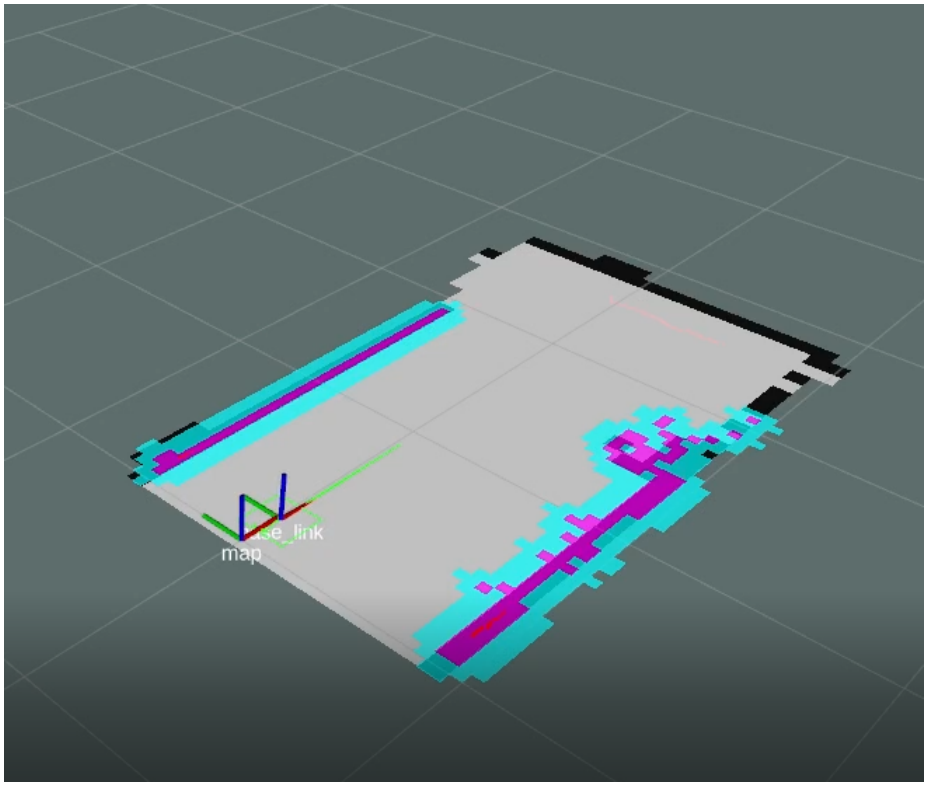


Figure 6.8: Map with pitch data

6.3.5 Movement

The snakes movement is decided by the navigation stack. The navigation stack sends out an velocity that the snake should move forward with and the turn rate needed. This is translated by the python script called translater that makes this into letters, based on how high these values are and what they are. These letters can be as an example "f" for go forward with no turning for one cyclus. Since one cyclus goes by incredibly fast the snakes movement seems continues but is a step by step process in the snakes point of view. The table for the different movements the snake can take can be seen in table 6.2.

6.3.6 Motor Control

The motor controller used for the servo motors are a ESP32. This controller sends all commands to all the servo motors, but receives information about what should be done from the PC using WiFi. To send different commands to the ESP32 a self-made communication protocol was used. A letter gets sent to the ESP32, which reads the command and uses a switch-case to understand what needs to be done.

6.4 Initial Tests

When testing the forward movement it was done some IMU logging to see how the movement would effect the LiDAR. This test was done with forward movement with an amplitude sett to 30. This will say that the servos can turn 30° up and down. The amplitude is sett to 30° because that's the amplitude the snake walks best on and its the amplitude used under normal walking as was found in the previous project.

When doing this test it was found that the IMU data was decently stable and had a max amplitude of 30°, as can be seen in figure 6.9.

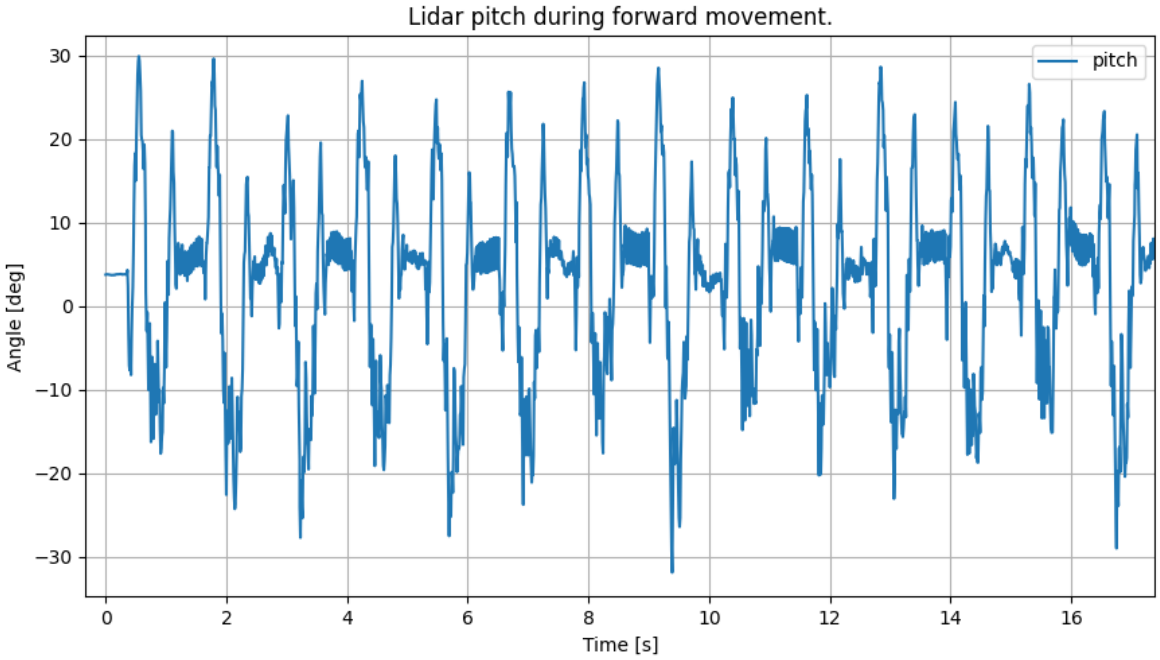


Figure 6.9: Pitch at forward movement

There is some noise between second one and two in the graph, this is assumed to be when the snake slams down in the ground.

It was also logged roll in the this same attempt, the roll should not give any reaction when the snake moves. This is because the snake should not be able to have roll without moving in terrain that is uneven. As it can be seen in the figure 6.10 the roll has some reaction up to 10°the high spikes. They seem to be in sync with the pitch's peaks and is assumed to have correlation to rapid acceleration and deceleration of the snakes head and also when the head impacts the ground.

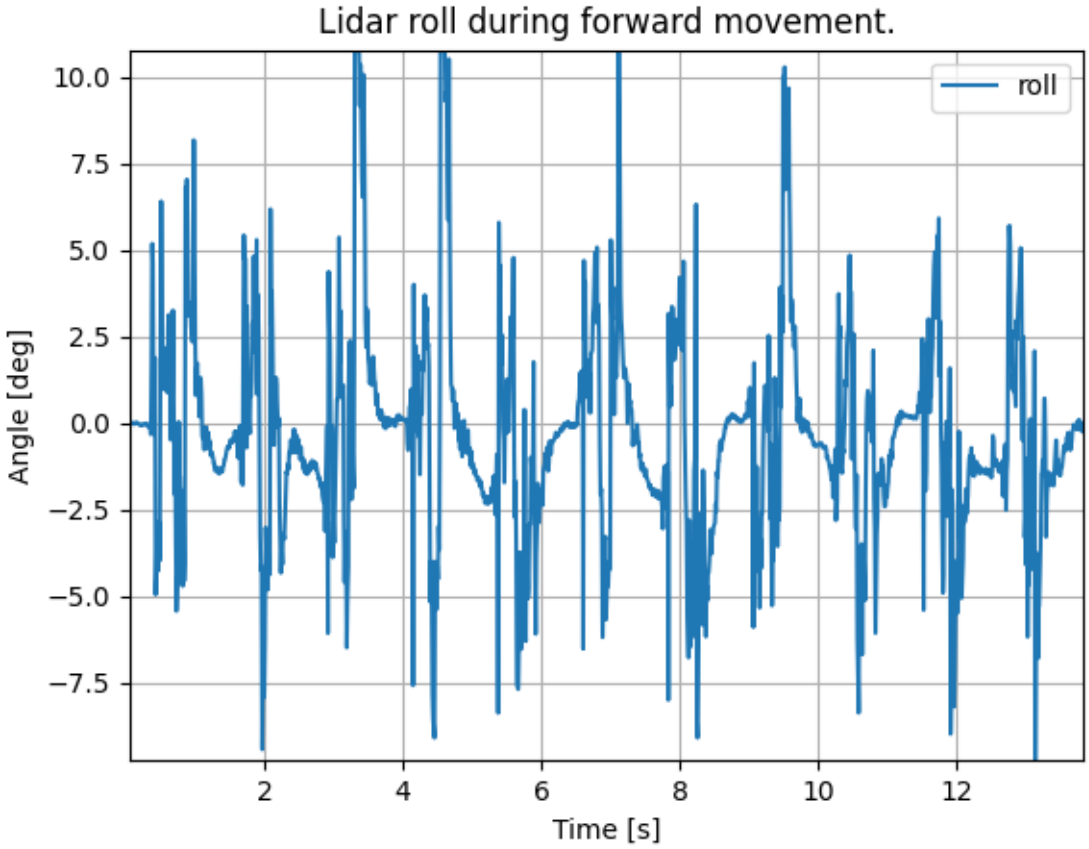


Figure 6.10: Roll at forward movement

It was also done a test where it was logged pitch and roll while turning, the turn angle was sett to 60 °. The result of the pitch can be seen in fig 6.11. As can be seen here the pitch is not as high and there is not as the same amount of jitter as in figure6.9. The increase in roll is probably because the snake can have the tendency to tilt some while turning and moving forwards at the same time. This can be a result of the heavy LiDAR at the snakes head.

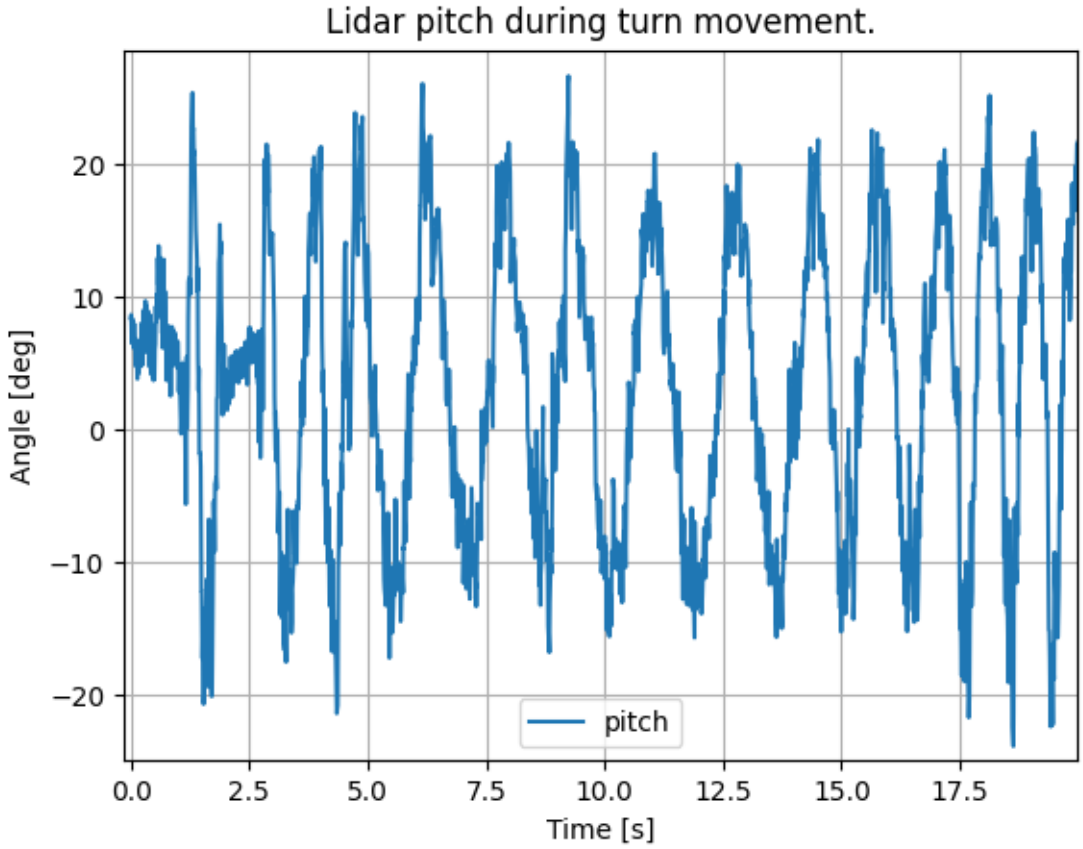


Figure 6.11: Pitch logged with turn sett to 60°

There was also done a logg while rotating clockwise the result from the the pitch data can be seen in fig 6.12. The roll can be seen in fig 6.13. As we can see from these graphs the head tilts. These test shows the extent of the roll and pitch motion experienced by the LiDAR which can impact mapping performance.

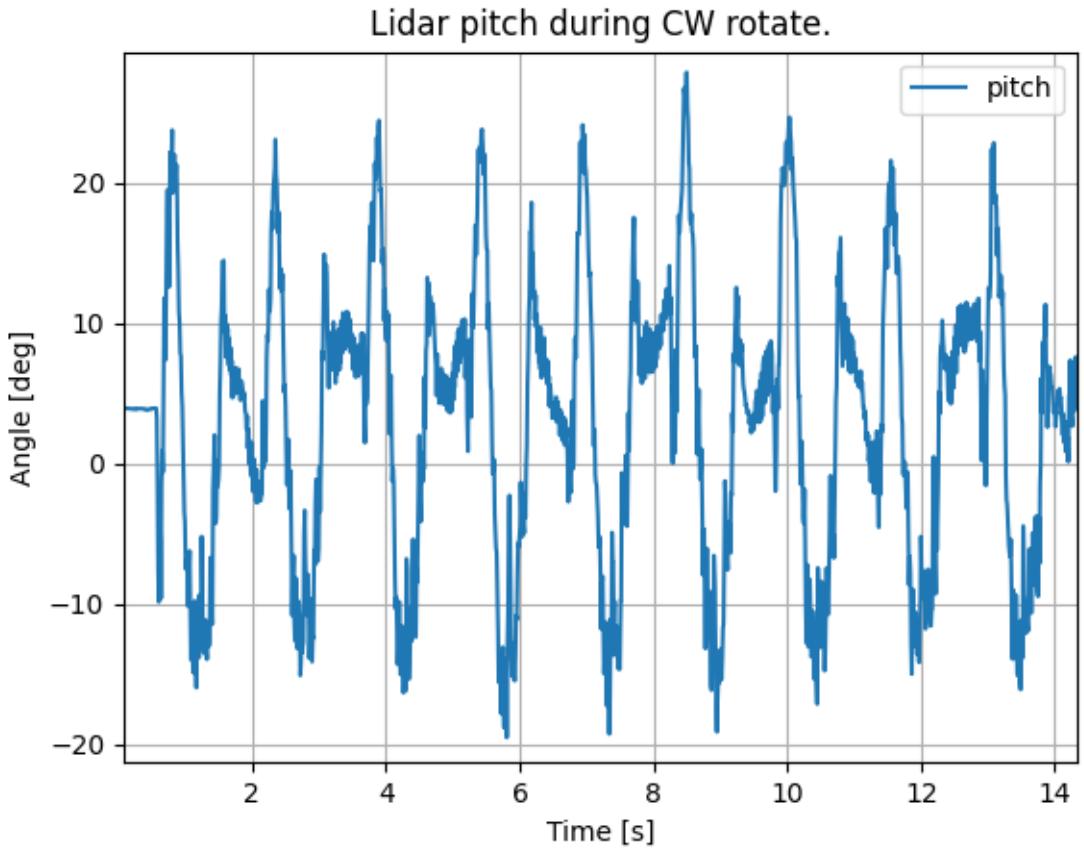


Figure 6.12: Pitch logged while moving clock wise

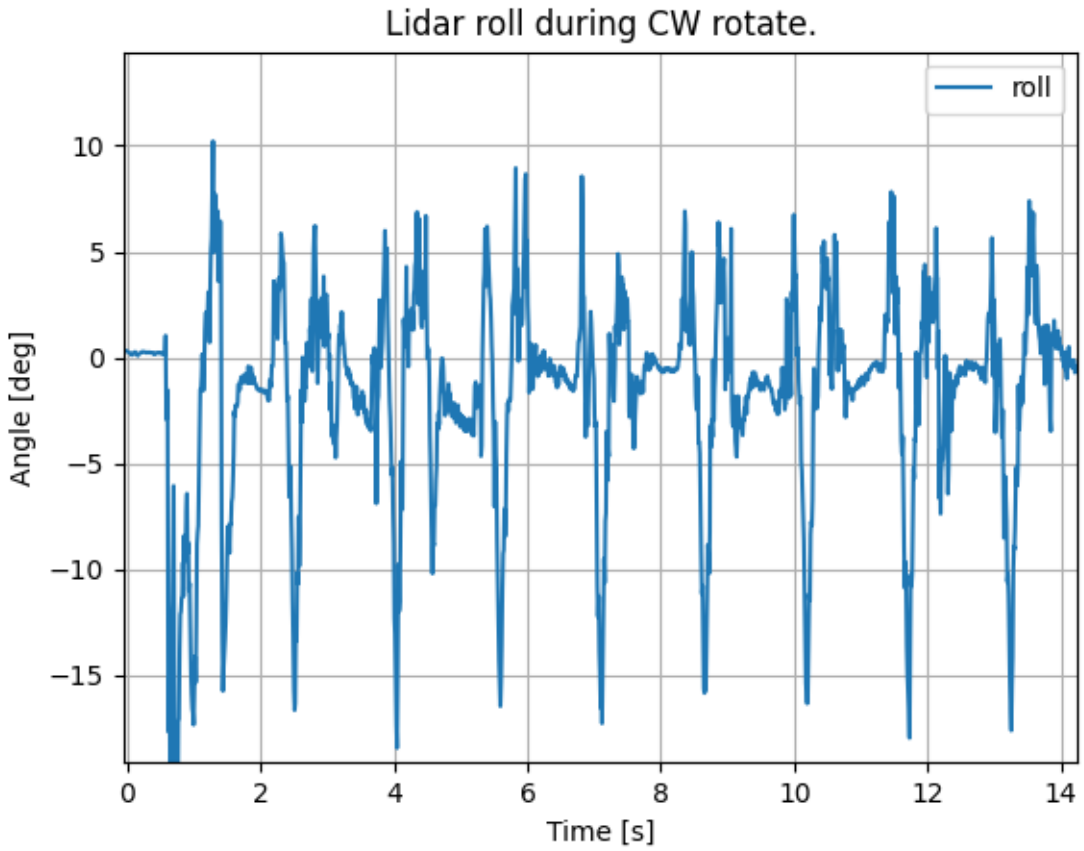


Figure 6.13: Roll logged while moving clock wise

6.4.1 Mapping Performance

After the initial test showed that the LiDAR moves a lot under movement a LiDAR filter was implemented, and a test to see the effects the LiDAR filter had on the hector_slam maps was done. There was done two tests, one without the filter and one with. During these tests the robot was tilted in the pitch direction manually, while IMU data was logged and the map was video recorded. The IMU data was recorded in order to make sure the test was done on similar terms.

In both tests it was aimed to have a pitch of ±30° as this is the pitch that will come when the snake walks forwards, as can be seen in 6.9. The first test implemented the LiDAR filter for values of ±10° meaning that LiDAR data captured while the pitch angle was outside this range would be discarded. The logged pitch values for this test can be seen in figure 6.14. The second test

did not implement the LiDAR filter and as such all LiDAR data was transmitted to the mapping algorithm. The IMU data can be seen in figure 6.15.

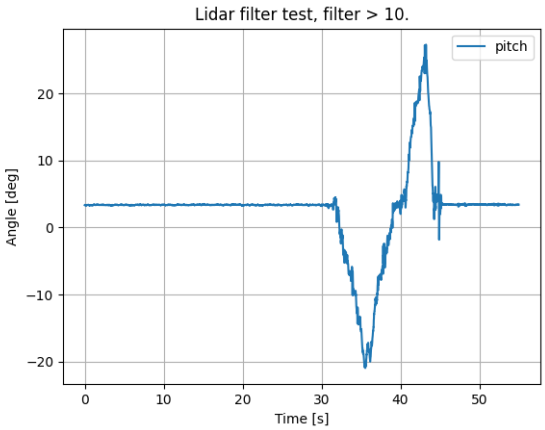


Figure 6.14: Pitch test +-10°filter

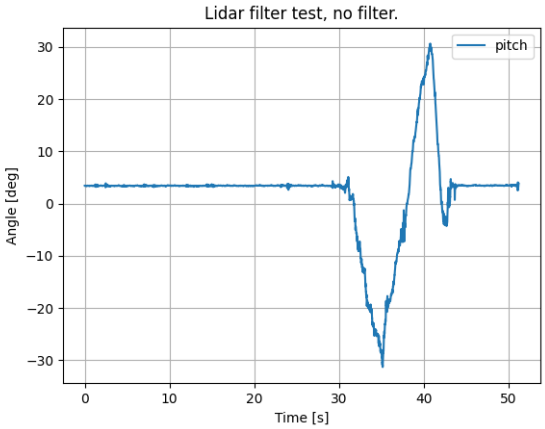


Figure 6.15: Pitch for Lidar test without filter

6.4.2 Test with filter

This section show the test done with the filter implemented. As can be seen in fig 6.16 this is now the lasers highest reading, while the lasers lowest reading is displayed in figure 6.17. In figure 6.17 the laser scan is now below the mapped area and is hard to see. It can be seen in this figure that the snake is still able to see the ground but as figure 6.18 shows hector_slam does not lose track of its position. It still moves the base link as can be seen in 6.17, but it goes back to its

original point when looking back up. This can be seen in figure 6.18.

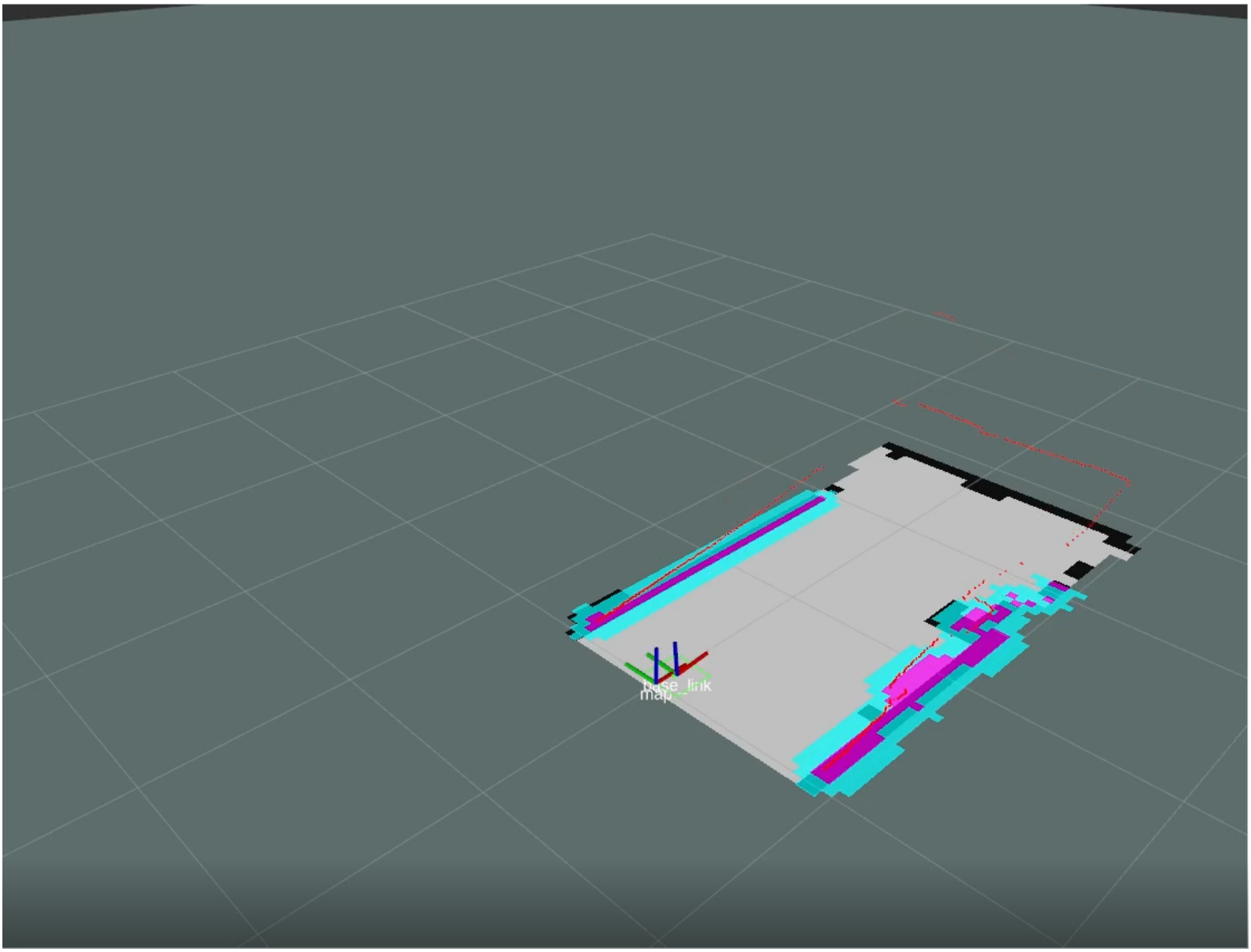


Figure 6.16: Laser max range upwards

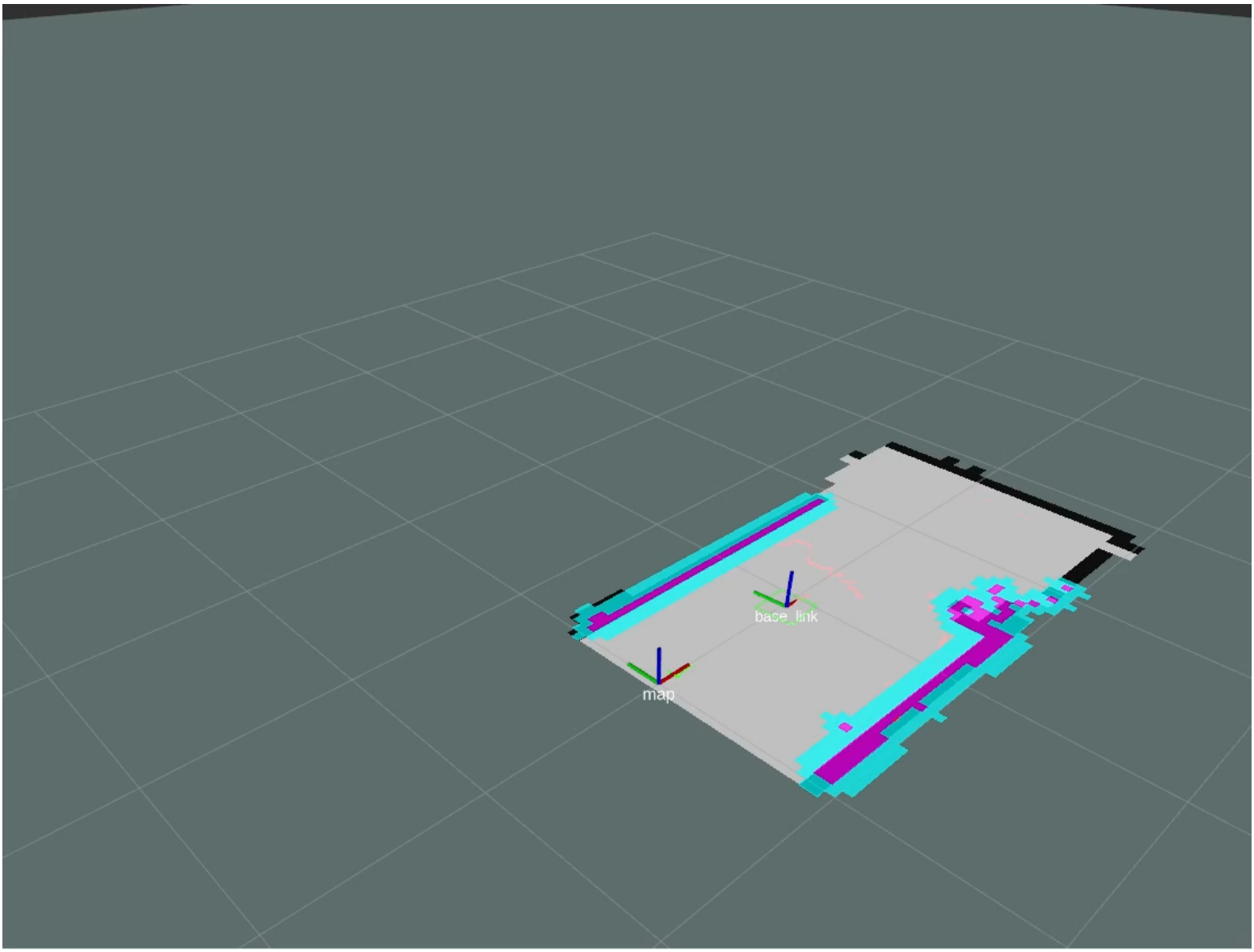


Figure 6.17: Laser max range downwards

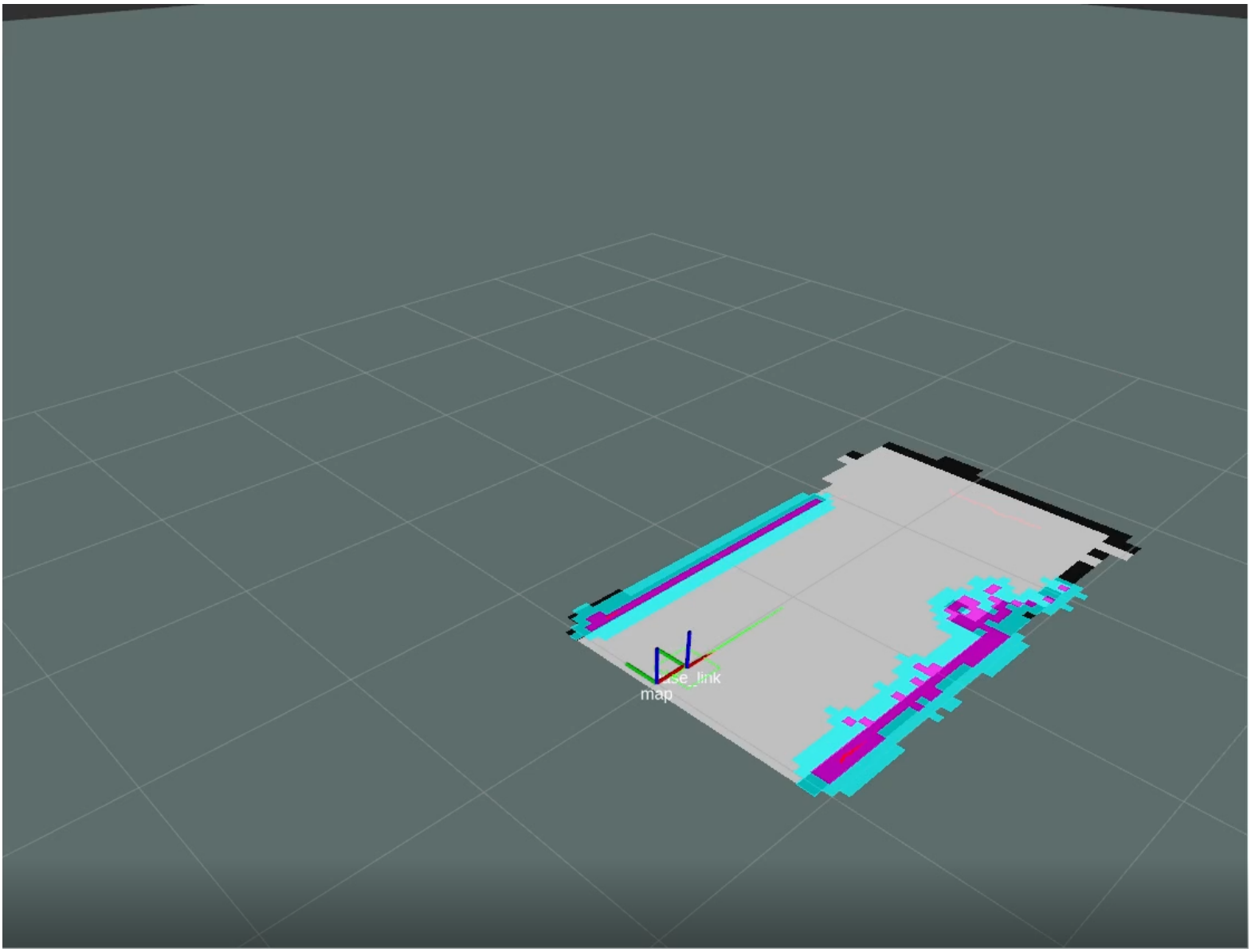


Figure 6.18: Map after pitch max downwards and upwards

6.4.3 Test without filter

When the filter is not present the pitch was a bit higher but this does not have a big impact since the first attempt would filter out all data over 10°. While the amplitude of the first test does not matter above 10° the rate of change does. The graph for the pitch in the attempt without the filter can be seen 6.15 and the rate of change is relatively similar to the first attempt as seen in figure 6.14. When the LiDAR filter was not implemented hector_slam could not keep track of the position of the robot.

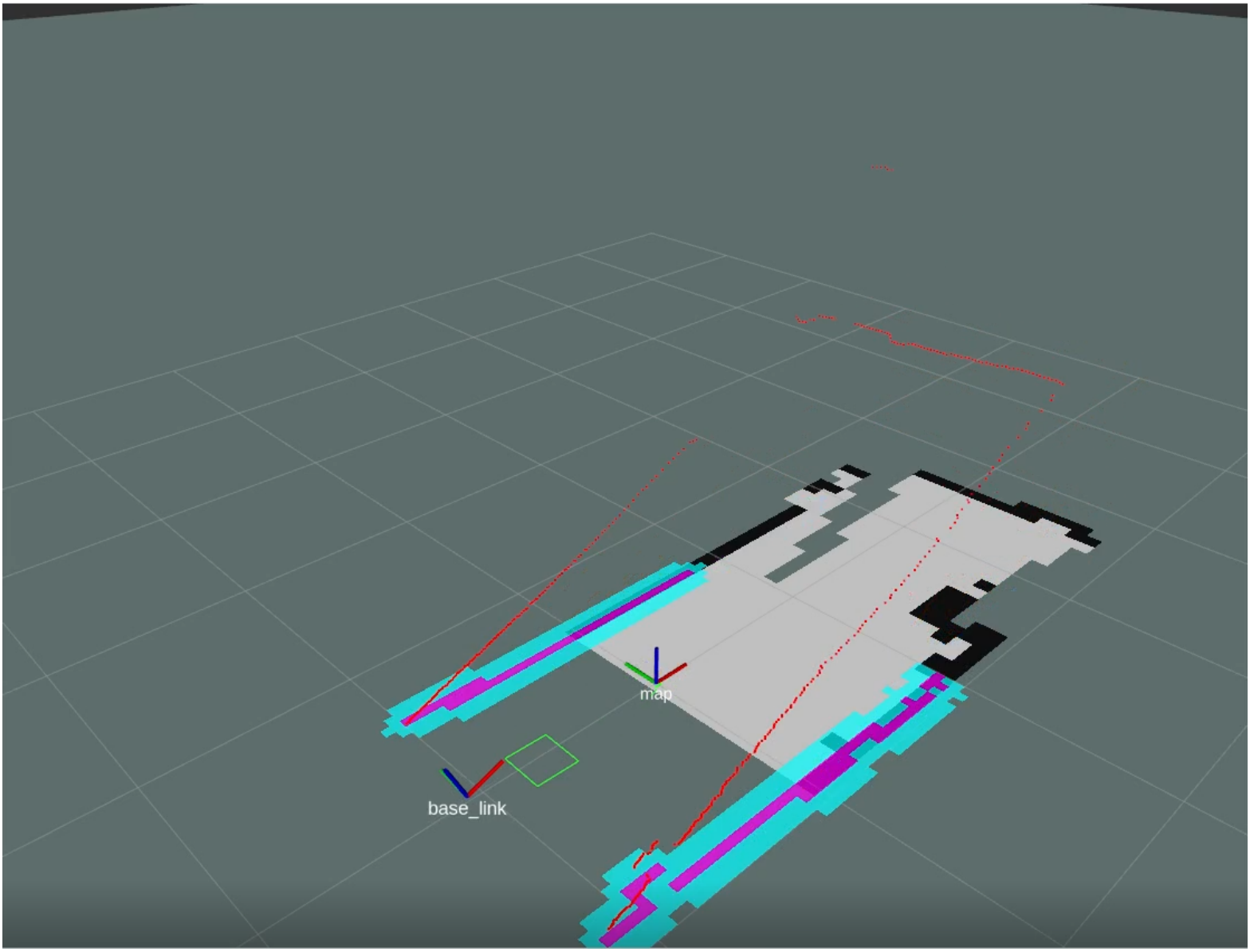


Figure 6.19: Laser without filter sees roof.

As it can be seen in the figure the the laser scan now can be taken at any pitch angle. This gives the snake the opportunity to see roof, this makes the base_link of the robot move drastically and make the robot think it is a different place on the map. When the snake looks downwards without the filter made it even worse as it can be seen in 6.20.

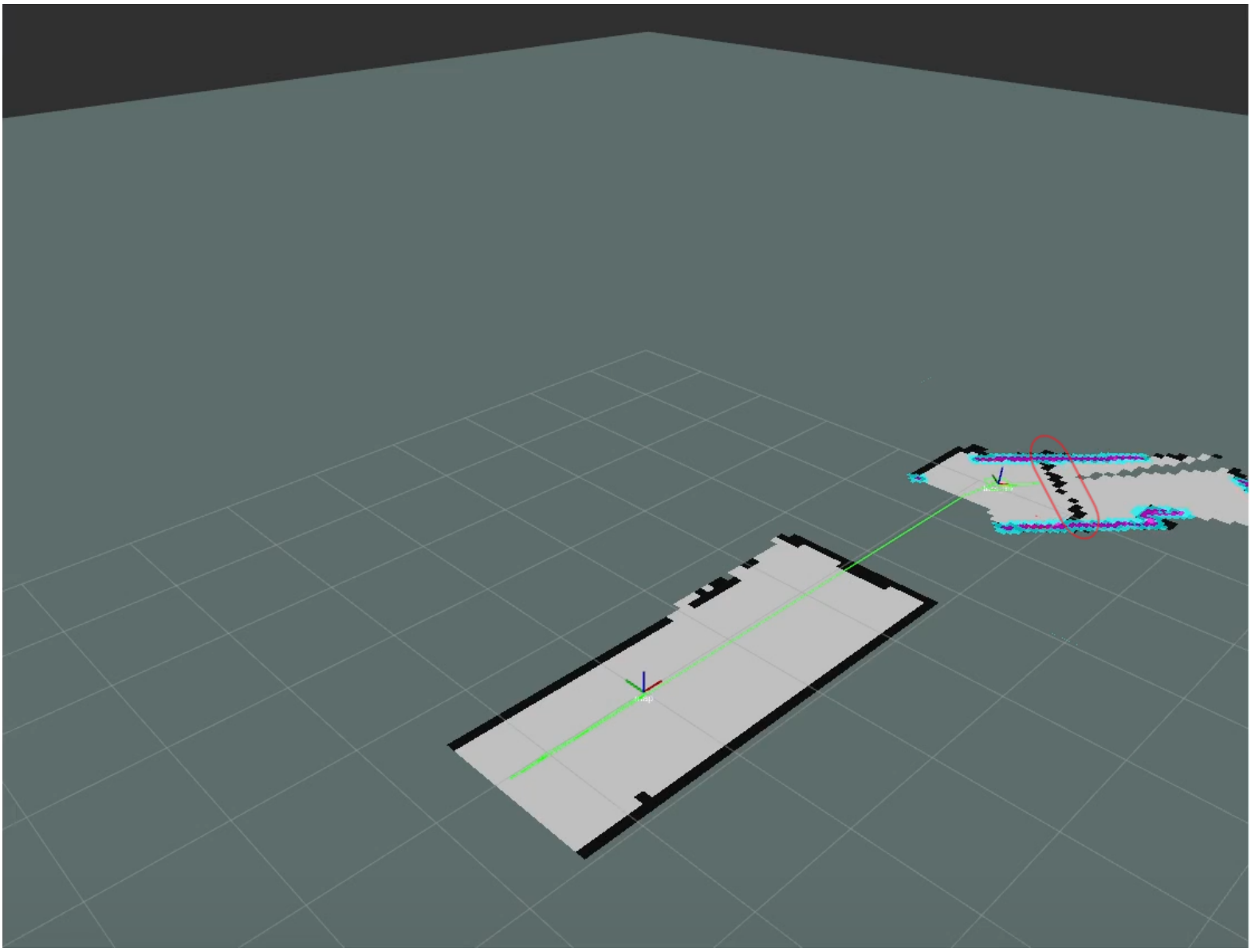


Figure 6.20: Laser without filter sees floor.

It can be seen here that hector_slam losses it original map and thinks it has moved drastically. Hector_slam moves the robot's position to a clear section of the map, and starts mapping from this point. A "wall" can be seen right in front of the robot, and this wall corresponds to the laser scan of the LiDAR seeing the floor.

6.5 Navigation Stack Testing

When testing the navigation stack it was done by setting a point to walk to in the rivz GUI. When this set point was putt down it could be seen on the map that there came a green line that the navigation stack wanted to follow. To see what commands the navigation gave out an what kind of format it was on it was used a ROS function to echo what was send out by the navigation stack and display it in the Linux terminal. The navigation stack send out a velocity and an angle to walk in.

6.6 Performance Test Results

The performance test was set up in to tree different modules. The first was to test the navigation stack and hector_slam without the Lidar on the snakes head. This worked well, hector_slam made a good map and the navigation stack sent commands to the snake about where to walk.

The next step was to test whit the LiDAR on the head of the snake. This was not very success full as the LiDAR would see the ground each time the snake was moving. This made hector_slam not understand where it was and the map and location of the snake would get ruined.

The third test was with the LiDAR on the head of the robot, and IMU data for odometry. This was done by feeding the IMU data in to the odometry of Hector_Slam and by filtering the LiDAR in to readings that would be within +10° and -10°. This made huge improvements for the map, it was more stable and was able to hold it's location.

6.6.1 Obstacle Avoidance

Obstacle avoidance was done by the navigation stack in ROS. To sett a point to walk to it was possible to make a navigation point in rviz in ROS, this shows a map of what the LiDAR on the snake can see. When setting a destination point the navigation stack would look for the possibility's of where to go. The navigation stack was given the odometry of the snake and could use this information to be able to know where it is possible to go. The navigation stack would then

make a path to walk to get to the destination.

But when having sudden new obstacle apere where the snake had already mapped, the snake would get problems with understanding where it was and often move where it thought it self's location was instead of putting up an obstacle on the map. This was happening because of to few data points on where the snakes location was and when it was moving.

6.6.2 TroubleShooting

There has occurred something called a brownout error on the Esp32, this is something that comes when the Esp32 gets a to low voltage. This challenge took sometime to locate. It was located after a lot of debugging in code that the problem was within the esp32 and that it was restarting at what seamed random.

It was connected a serial cable to the esp32 to debug internally and it was found that the Esp32 had brownout errors. This appears when the esp doesn't get enough power. To locate more precisely where the fault was it was made a arduino code that tested one and one part of the robot at the time. It was discovered that servo number tree had a short circuit. This servo was replaced by a new one. The first servo that was used as a replacement was a servo 360°servo with an encoder, and could not be used, together with the library that was used for the other servos. So once again it was replaced with a servo that could be used, this time it was successful.

After this there came a new challenge this time the ESP-32 wouldn't stay on the network it was assigned to, it disconnected then reconnect. To start troubleshooting the esp-32 was inspected and it was found that there was some potential short circuit's. The robot was opened and esp inspected it was discovered some small cords that had the potential to short the esp, without knowing if this actually had happened. These potential shorts was fixed. It was also discovered that the WIFI library in some cases could have problems whit pin 28 and 29 on the esp32 being used. This was the pins that was used for one of the IMU's, so this IMU was also soldered of.

When this was done it was time for a WiFi test, this was done with an test scatch with only WiFi library. The run whit a test scetch worked well, then the IMU was soldered on to pin 28 and 29 again to test if this was the actual problem. That was not the case, the esp32 connected to the network again with no problems discovered. The next test was to see if the esp32 would connect to network when the regular program was loaded on to it. It would now connect and disconnect again. The next step was to comment out everything that had anything to do with the IMU's in the code. This made the regular code work again, it was started to put in one and one step of the IMU's code to see when the problem occurred. The result that was ended up with was that the esp32 had problems when needing to send packages over the network.

What happened was that when the IMU's was sending packages over the network the servos would go in slow motion. To try and fix this it was decided to solder over the IMU to the orginal I2C connections on pin 22 and 21, and solder the servo to 26 and 27. This did not work the robot still acted the same way. So instead it was added a new micro-controller to be used for gathering of IMU data and speaking with the computer directly by serial communication.

The reason the IMU and servos wouldn't work at the same time on the esp32 is believed to be because of a finite number of internal interrupters for making the I2C communication work and PWM's signal but no concrete evidence was found. This was because of what was read on multiple forum posts. What was done to fixed this problem in the end was to put on a arduino that handles all IMU data, and sends it to the main computer by serial communication.

Chapter 7

Discussion

7.1 Test results

7.2 Software

7.2.1 FilterLidarData

The filterLidarData script was made for filltering LiDAR data this was done as an attempt to make hector_slam more stable, this worked as planned and improved the result of the project. But it could have been done in a more comprehensive way. So this is a function i would say is important and was successful even tho it was really simply made.

7.2.2 Translator

The tranlator script could have been improved by having a more advanced way of choosing movement. The final result was originally made just to test if translator could choose movement in the way i imagined. This worked well, an the reason it was not improved more was because of time restraint.

7.2.3 IMU_NODE

The IMU_NODE was made to request data from the arduino and publish it to other nodes. This worked well, and made it so that the result became as good as they did. The node was made last minute when the other original plan wouldn't work. This worked well and is implemented in a good way.

7.2.4 Servo Code

The arduino code for the esp32 was just done small changes to. But should have been refactored, this was not done because off time restraint. It worked for it purpose but to get better walking and more in sync with the navigation stacks out put this code and the translater code should have been completely reworked.

7.2.5 ROS

I think the choice to use ROS is one of the main factors that the project came as far as it did, even though it didn't reach the plans that was first sett. ROS took time to understand and learn to use, but when it first was up and running it worked well for its purpose.

7.3 Sensors

The sensors chosen was a 2D LiDAR and an IMU. It would have been a better option to use a 3D LiDAR, this would have given the project more stability by having more landmarks to associate its location to. To make this project work in a less controlled environment it would really help to have the 3rd dimension. The reason this was not done was a mixture of me being to optimistic and the school having the 2D LiDAR available. The first plan for this project was to use image processing and the 2D LiDAR to make a sort of 3D scan.

7.4 Prototype

The prototype was made in the last project and was not made for changes 2.1. It was sufficient for the last experiment but was not made to last much longer than for that project's duration. So when needing to change the purpose of the snake there was some problems. The ESP32 was not able to send IMU data and simultaneously control the servos. The battery pack was not able to send enough power, this was also a challenge. But was fixed by adding a external power supply.

7.4.1 Improvements

There are a bunch of ways i would have improved this project if there was time for it. The prototype that was used in this project should be thrown away, and there should be made a new one. The new one could take inspiration from this one, but it is important to understand its flaws. There should be 3 micro-controllers inside the snake. One for handling the servos, this could be a esp32. There also needs to be one for handling IMU data, this could be done with either with a esp32-camera module or a normal arduino. The last micro-controller should be a raspberry pi, and can use a camera module for vision. If this is done the esp32-camera would not be needed and it would be best to use a normal arduino for the IMU's. The raspberry pi has support for ROS and could be used as a node, this would make it easy to get data from the robot to a computer. For power-supply i would also recommend that it was made with a charging option that can charge the battery's for a bigger amount than the robot use in idle.

7.5 Personal Experiences

7.5.1 Corona's impact on the project

I was aware that corona would have some impact on how the project would be done. But since it was a something i was prepared for going in it didn't have a very big impact. It made it so that i would work from home, this was the biggest disadvantage. The loss of routines and social aspect of being at a school and have people to ask question and explain problems, is something that was missed. But other than that corona has not made this project suffer in any big way.

7.5.2 Project Organization

The project can be divided in to tree phases, the first one was the research phase. The second phase was the prototype phase, witch was making code for the concepts and making the robot as planned. The last phase was testing, these phases was planned to be in order. But as most project this was did not go as planned and ended up going back and fourth between the phases.

7.5.3 Work Flow

There was a good workflow in the start was good but each time there came a step i got stuck on it could come a period of time where the workflow would go down drastically. Since i was alone i didn't have anyone to discuss ideas and plans with, and that is something i missed trough this project. Instead i used friends that i discussed solutions with online, which was good help when stuck.

Chapter 8

Conclusions

The goal of this project was to make the snakelike-Robot less dependent on a controlled environment using LiDAR and whatever tools necessary. This goal has been reached in some way, but is not to the potential it could be at. By making a new and improved snakelike-Robot i think it would be possible to get some great result that could be really interesting to follow. Improvements that i can suggest are listed in 8.1. In my personal experience with this project is that the snake like robot is a good robot to use for learning, and have great potential for being an autonomous search robot.

8.1 Further work

There is a few things to be worked on and improve or changed,

- Make a new prototype with a raspberry pi as a communication node
- Optimize the arduino code in a whole
- Use a 3D LiDAR
- Make a sufficient power-supply
- Rework the translator script

What i would do is setting up an arduino that takes in all the data from the IMU, that would speak to a inboard raspberry over serial communication. I would also have an ESP32 that would

control the servos that also communicated to the raspberry over serial communication. The LiDAR could also be connected to the raspberry so there would be no cables needed to the snake. The raspberry would be sett up as a ROS node and communicate with a main computer to do processing. For image processing it could have been used a raspberry camera module.

Bibliography

[1] Sparkfun thing plus - esp32 wroom. URL https://www.sparkfun.com/products/15663. Visited 01.12.2020.

[2] Guoyuan Li, Håkon Bjerkgaard Waldum, Marcus Olai Grindvik, Ruben Svedal Jørundland, Houxiang Zhang. Development of a vision-based target exploration system for snake-like robots in structured enviorments. URL https://journals.sagepub.com/doi/full/10.1177/1729881420936141. Visited 10.12.2020.

[3] Arduino. Float. URL https://www.arduino.cc/reference/en/language/variables/data-types/float/. Visited 04.12.2020.

[4] Tim Bailey. Slam: The essential algorithms. URL https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf. Visited 10.12.2020.

[5] Steinar Brandslet. A giant subsea snake robot, August 2017. URL https://www.sintef.no/en/latest-news/a-giant-subsea-snake-robot/. Visited 19.11.2020.

[6] Eitan Marder-Eppstein, Eric Perko. local_planner. URL http://wiki.ros.org/base_local_planner?distro=melodic. Visited 14.12.2020.

[7] Open Source Robotic Foundation. Ros introduction, . URL http://wiki.ros.org/ROS/Introduction. Visited 02.12.2020.

[8] Open Source Robotic Foundation. Nodes, . URL http://wiki.ros.org/Nodes. Visited 04.12.2020.

[9] Open Source Robotic Foundation. Movebase, . URL http://wiki.ros.org/move_base?distro=noetic. Visited 02.12.2020.

[10] Randy Frank. Mems imu. URL https://www.sensortips.com/pressure/mems-imu-delivers-10-degree-of-freedom-capability. Visited 02.12.2020.

[11] Akshat Goel. Servomotor: types and working principle explained. URL https://engineering.eckovation.com/servo-motor-types-working-principle-explained/. Visited 09.12.2020.

[12] hamamatsu.com. Lidar. URL https://www.hamamatsu.com/sp/ssd/application/LiDAR/LiDAR_TOF_en.jpg. Visited 09.12.2020.

[13] Austin Hendrix. msg. URL http://wiki.ros.org/msg. Visited 12.12.2020.

[14] HOYUKO. Urg-04lx-ug01. URL https://www.hokuyo-aut.jp/search/single.php?serial=166. Visited 03.12.2020.

[15] https://www.mathworks.com/. Slam with lidar scan. URL https://www.mathworks.com/help/nav/ug/implement-simultaneous-localization-and-mapping-with-lidar-scans.html. Visited 19.12.2020.

[16] iRobot Corporation. Irobot. URL https://web.archive.org/web/20130511232754/http://www.irobot.com/EngineeringAwesome/images/iAdapt%20Fast%20Facts.pdf. Visited 02.12.2020.

[17] Danny Jost. What is an ultrasonic sensor? URL https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor. Visited 01.12.2020.

[18] David LU. global_planner. URL http://wiki.ros.org/global_planner?distro=noetic. Visited 14.12.2020.

[19] William D.Smart Morgan Quigley, Brian Gerkey. Programin robots with ros. URL https://books.google.no/books?hl=no&lr=&id=Hnz5CgAAQBAJ&oi=fnd&pg=PR2&dq=What+is+robot+operating+system&ots=-6pgK01BK2+&sig=

Pd3jIuuCf5Jp4El9IRyxKD-DMHY&redir_esc=y#v=onepage&q=What%20is%20robot%20operating%20system&f=false. Visited 02.12.2020.

[20] nasa. Remote sensors. URL https://earthdata.nasa.gov/learn/remote-sensors#hyperspectral. Visited 02.12.2020.

[21] Charles Pao. What is imu sensor? URL https://www.ceva-dsp.com/ourblog/what-is-an-imu-sensor. Vistited 02.12.2020.

[22] Darren Sawicz. Hobby servo fundamentals. URL http://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf. Visited 01.12.2020.

[23] Søren Riisgard, Morten Rufus Blas. Slamfordummys. URL https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf. Visited 14.12.2020.

Appendices

Appendix A

Project Planning

A.1 Pre-Project Report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Snake like robot V2

KANDIDATNUMMER(E):
Marcus Olai Grindvik

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
18.08.20	**IE303612**	**Bacheloroppgave**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
BACHELOR I INGENIØRFAG - AUTOMATISERINGSTEKNIKK	/	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Ottar L. Osen - NTNU Guoyuan Li - NTNU

OPPGAVE/SAMMENDRAG:
Oppgaven er gitt på grunn av etterspørsel på mere forskning innen slange robot med visuelle sensorer. Det er tatt et valg om å bruke lidar til å kunne kartlegge og posisjonere roboten i et rom. Det vil deretter brukes lidar data sammen med bilde for å sette sammen en 3D-visualisering av rommet. Det er satt samen en risikovurdering om hva som kan utsette prosjektet og hvilket risikoer som kan forekomme. Denne forprosjektrapporten tar for seg oppgaven omfang, planlagt fremgangsmetode, behandling av eventuelle avvik og generell organisering av prosjektet.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Det ble sett en mangel på kunnskap og løsninger innen «Learning based vision perception for industrail robots» det var oppgaven jeg fikk interesse for, ved hengivelse fra Gouyuan.

2 BEGREPER

BEGREP	BETYDNING
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping

3 PROSJEKTORGANISASJON

3.1 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

- Ottar L. Osen
- Guoyuan Li

4 AVTALER

4.1 Avtale med oppdragsgiver

Avtale med oppdragsgiver er fritt, kravet er å levere inn ukentlig rapport.

Det ble også avtalt å ha virtuelt møte hver 14 dag, for å ha oppdatering om prosjektet.

4.2 Arbeidssted og ressurser

Det vil være begrenset tilgang på arbeids plass på grunnlag av Covid-19. Dette vil begrense tilgang til ressurser som 3D printere og andre verktøy. Hovedsakelig vil arbeids plassen være min egen leilighet, som er begrenset størrelse på, men det skal være tilstrekkelig til oppgaven.

Tilgang på personer er også begrenset, Men det vil være mulighet for å ha Zoom møter og noen fysiske møter.

Tilgang til resurser som utstyr og deler har ikke blitt satt restriksjoner på, men det kan ta tid å få tak i deler på grunn av Covid-19.

Når det gjelder data sikkerhet og lagring av min data, vil jeg bruke Teams og Github med private miljø som gjør at det holdes sikkert. Rapportering til arbeidsgiver vil være annen hver uke.

4.3 Gruppenormer – samarbeidsregler – holdninger

Min rolle i samfunnet er å lagre automasjons løsninger som ikke vil være til skade til mennesker, men som vil øke effektivitet og sikkerhet for produkt produksjon som er bærekraftig. Det må være produkter som lett kan utvides eller bygges på i framtiden og lett kan vedlikeholdes.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Problemet vil være å få laget en autonom robot som klarer å navigere seg gjennom en labyrint bare ved hjelp av front kamera og LiDAR. Den må også kunne skille mellom objekter som skal bli funnet og objekter som er irrelevant. Den skal i tillegg bygge opp en 3D-visualisering av rommet slangen utforsker.
Målet er og få en autonom robot som kan finne fram i et rom ved hjelp av bilde prosessering og SLAM. Hensikten med dette er å få høyere kunnskap om hvordan man kan bruke bilde prosessering og 2D LiDAR som eneste sensorer for navigering og 3D-visualisering. Det kan bli vanskelig å klare å klare å lage en 3D-visualisering ved hjelp av en 2D LiDAR og et bilde uten IMU eller andre sensorer.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Krav til oppgaven er at det skal bli laget en autonom slange som kan bruke bare et front kamera og LiDAR til å lokalisere seg selv og andre objekter.

Prosjektet skal inneholde;

- Forprosjektrapport
- Fungerende prototype
- Rapport
- Generell dokumentasjon (Bruksanvisning, el-tegninger, kode, budsjett resultatdata)

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Planen blir å starte med den eksisterende koden av slange roboten, hente ut grunnlegendes kode som har blitt skrevet for å få bevegelse og kommunikasjon til Python, resten vil bli skrapet og startet på nytt. Der etter blir det å få LiDAR'en til å fungere med Slange roboten. Deretter vil det blir brukt iterativ utviklingsprosess får å prøve å finne fram til en måte å løse oppgaven, på en god og fornuftig måte. Planen er å finne en måte å bruke en 2D LiDAR til å lage en 3D-visualisering av et rom og helst

uten bruk av RoS. Det kunne ha vært lettere å løse oppgaven ved hjelp av RoS men har valgt å gjøre oppgaven uten. Vil bruke et kamera fra en ESP-32 til å ta bilder med som skal forsøkets å sy sammen med bildene fra LiDAR'en.

5.4 Informasjonsinnsamling – utført og planlagt

Det er hentet informasjon om SLAM og informasjon fra slange prosjektet som eksisterte fra før. Videre vil informasjon om hvilke teknikker som kan brukes og har blitt brukt før i lignende prosjekter sjekket opp.

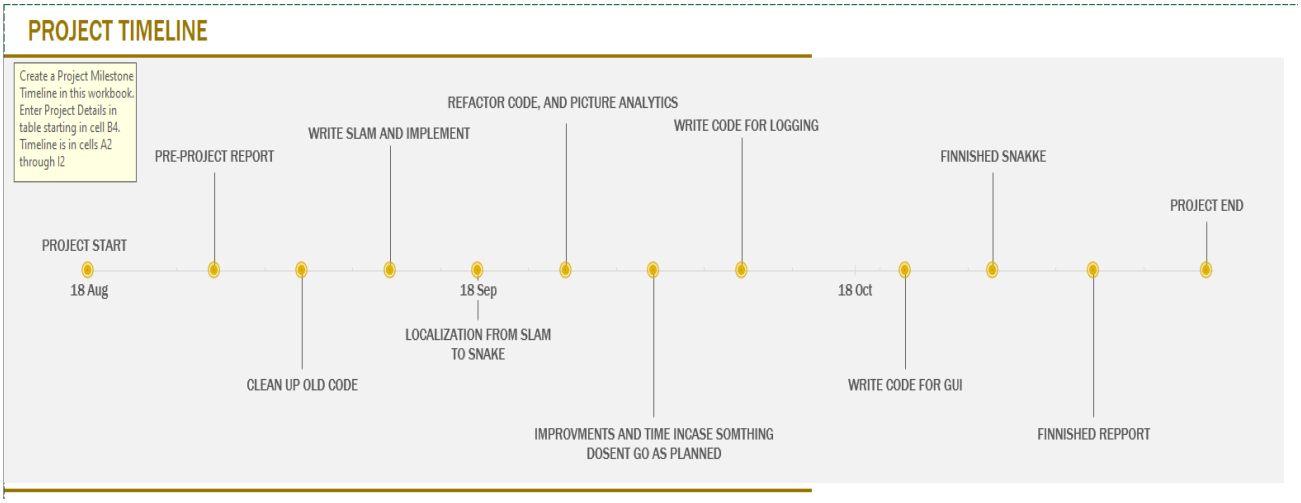
Det er også påbegynt research om andre slange robot prosjekt, som er skrevet vitenskapelige rapporter om.
Det er også gjort en del informasjonsinnsamling om hvordan få rå data fra LiDAR'en, og dette er blitt løst på en god måte.
Det trengs også informasjon på hvordan man kan benytte LiDAR og kamera for å bygge en 3D-visualisering.

5.5 Vurdering – analyse av risiko

Denne oppgaven burde kunne realiseres, men det kan bli noen problemer ved å finne et kamera som kan være i front av slangen som kan håndtere all bevegelsen og slag i bakken ved bevegelse. Det kan også bli behov for deler som kan ha lang levering tid på grunn av covid-19.
Det er også fare for ned stenging av barnehage på grunn av covid-19 som vil gjøre at min tid til å skrive på oppgaven vil bli drastisk redusert.
Det er også brukt LiPO batterier i slange roboten som kan være farlige vist de ikke blir håndtert riktig.

5.6 Hovedaktiviteter i videre arbeid

Nr	Aktivitet	Ansvar	Kostnad	Tidsbruk
A1	Kode	M.G	0	150 T
A1.1	Rydde eksisterende kode	M.G|	0	37,5 T
A1.2	Skrive Kode for bilde gjenkjenning av objekt	M.G	0	37,5 T
A1.3	Skrive Kode for lokalisering av robot	M.G	0	37,5T
A1.4	Sette sammen Kode	M.G	0	37,5 T
A2	Research	M.G	0	100T
A2.1	Bilde behandling	M.G	0	15T
A2.2	SLAM	M.G	0	30T
A2.3	Bevegelse Slange	M.G	0	15T
A2.4	Materieller	M.G	0	10T
A2.5	3D Visualisering	M.G	0	30T
A3	Dokumentasjon	M.G	0	50T
A4	Prototype-bygging	M.G	10 000 KR	10T
A4.1	Bygging	M.G	10 000 KR	5T
A4.2	Testing	M.G	0	5T



PROJECT DETAILS

DATE	MILESTONE	Position
18-Aug	Project Start	
28-Aug	Pre-project report	10
4-Sep	Clean up Old Code	-10
11-Sep	Write SLAM and implement	15
18-Sep	Localization from SLAM to snake	-5
25-Sep	Refactor Code, And picture analytics	15
2-Oct	Improvments and time incase somthing dosent go as planned	-15
9-Oct	Write Code For Logging	15
22-Oct	Write Code For GUI	-10
29-Oct	Finnished Snakke	10
6-Nov	Finnished Repport	-15
15-Nov	Project End	

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Det vil bli fokusert på å gjøre dokumentasjon og rapport gjennom hele prosjektet. De første ukene vil det holdes fokus på å gjøre reaserch på forskjellige måter å kunne løse prosjektet på, samt prinsipper som SLAM. Det neste steget vil bli å lage prototypen ferdig, dette innebærer å sørge for at kamera på slangen fungerer og montere på LiDAR'en.
Planen er å starte med å få rå data fra LiDAR'en, etterfulgt av å implementere SLAM. Deretter blir det fokus på få benytte kamera sammen med SLAM data, får også sy dette sammen til en 3D-visualisering. Til slutt vist det er tid vil objekt gjenkjenning implementeres.

5.7.2 Styringshjelpemidler

Det vil bli tatt i bruk:

- Timeliste
- Dagbok

Dette blir gjort i størst grad for min egen del slik at det er lett å se om det har blitt jobbet tilstrekkelig. Samt holde orden på ressurser.

5.7.3 Utviklingshjelpemidler

- PyCharm (Python)
- SLAM
- Matlab
- Arduino

5.7.4 Intern kontroll – evaluering

Internkontroll vil bli gjort samtidig som ukentlig rapport blir skrevet, da kan jeg sjekke at målene som var tenkt oppnådd er nådd eller om det har blitt feil beregnet i tid. Det må også tas gjennom gang i eventuelle mangler av ressurser.

5.8 Beslutninger – beslutningsprosess

Beslutning prosessen vil være enkel siden oppgaven skrives aleine, så alle beslutninger som ikke gjør at det kan ha stor betydning på om oppgaven er riktig utført i forhold til kravene vil gjøres av meg, vist det er usikkert om beslutningen vil ha store betydning på om arbeidskravene er forbeholdt vil Ottar og Guoyuan bli tatt med i beslutnings prosessen.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Dokumentasjon som skal gjøres gjennom prosjektet er:

- Forrapport

- Hovedrapport
- Timeliste
- Ukentlig statusrapport
- Bruksanvisning

Dette er dokumentasjon som blir ansett som krav at skal være med under utlevering av oppgaven.

Hovedrapport på skrives kontinuerlig på gjennom hele prosjektet, for å klare å holde kontrollen.

Timeliste og ukentlig status rapport vil bli skrevet samtidig, dette vil bli gjort ukentlig på samme dag som det vil være møte om statusrapport. Dette gjør at problemer og hva som har blitt utført denne uken er nylig gjennom gått i eget hode å kan lett formuleres til veiledere.

Bruksanvisningen vil bli tatt i slutten av prosjektet, denne vil hovedsakelig bli til eventuelle fortsettelser på prosjektet eller videre forskning på samme emnet.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Det er per nå avtalt møte med veiledere hver 14 dag på tirsdag, dette er for å ha statusrapport og tilbakemeldinger om arbeidet som har blitt utført.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Det er planlagt ukentlig rapport og møte med veileder annen hver uke. På denne måten vil veiledere ha kontroll på framdriften.

8 PLANLAGT AVVIKSBEHANDLING

Ved eventuelle av avvik i prosjektet, vil det bli tatt kontakt med veiledere for et møte, som kan hjelpe med hvordan oppgaven kan utføres på en ansvarlig måte uten å gå utenfor kravene for løst oppgave, selv med avvik. Vist det bare er et lite avvik vil det bare bli skrevet ned i rapport og løst på eget initiativ.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Slange robot
- Lidar
- Esp-32 Kamera

10 REFERANSER

VEDLEGG

Project plann.pdf

Risikomatrise.xlsx

Appendix B

Bill of Materials (BOM)

Materials	Quantity	Description	C
3D Printed chassi	5	The main part of the snake is 3D printed. (This was done in the last project)	?
Battery packs	4	2000mAh used to supply the robot with the power required	
ESP32	1	ESP 32 Used for the controlling the servos and controlling the snake	
ESP32 Camera Module	1	ESP 32 Camera used to take pictures (Is not implimented)	
Boost Converter	2	Boost Converter 2V-24V to 5-28V	
USB Battery Charger	1	USB Battery Charger so the batterys can be charged easy	
Hokuyo Laser Lidar	1	Scannes the roms	

B.1 BOM

Appendix C

Arduino Code

Esp32 code

```
#include <ESP32_Servo.h>
#include <WiFi.h>
#include <math.h>
#include <Wire.h>
#include <SPI.h>



////////////////////////////////////////////
// WIFI VARIABLES
////////////////////////////////////////////
// Information for connecting to WiFi
const char* ssid = "MSI";
const char* password =  "123456789";

// Information for sending information via UDP
const uint16_t port = 6969;
const char * host = "192.168.137.11";
```

```
// Buffer for packetsize
byte packetBuffer[128];

// Establish the UDP-Client
WiFiUDP udpClient;

////////////////////////////////////////////
// MOVEMENT VARIABLES
////////////////////////////////////////////
// Number of servos
const int numberOfServos = 5;

//Homemade timer for movement in cycles
int movementTimer = 0;

// Establish servo array
Servo myServo[numberOfServos];

// Amplitude for the servos
int A = 30;

// Different phase shifts for the different movement
float forwardPhi = (120.0 / 180.0) * M_PI;
float lateralPhi = (100.0 / 180.0) * M_PI;
float rollingPhi = (90.0 / 180.0) * M_PI;
float rotatePhiV = (120.0 / 180.0) * M_PI;
float rotatePhiH = (50.0 / 180.0) * M_PI;

float v2RotatePhiV = (50.0 / 180.0) * M_PI;
float v2RotatePhiH = (50.0 / 180.0) * M_PI;
```

```
// Time constant
int T = 6000; //12000

//Variable for speed of servos
int servSpeed = 0;

// DO NOT USE PINS 12-17, THESE MAKE PARSEPACKET CRASH THE WHOLE FUCKING
    SHIT
// SERVO LIBRARIES ARE FUCKING MORONIC
int servPins[5] = {4, 23, 26, 27, 25};

// Zero-point array for servos
int servZero[5] = {90, 93, 97, 87, 99};

// Booleans for movement
boolean goingForward = false;
boolean goingBackward = false;
boolean latLeft = false;
boolean latRight = false;
boolean rotCW = false;
boolean rotCCW = false;

////////////////////////////////////////
// SETUP
////////////////////////////////////////

void setup()
{
```

```
  ////////////////////////////
  // WIFI RELATED
  ////////////////////////////
  // Starting wifi
  WiFi.begin(ssid, password);

  // Trying to connect to WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    //Serial.println("...");
  }
  //Serial.print("WiFi connected with IP: ");
  //Serial.println(WiFi.localIP());

  // Begin listening on port 9696
  udpClient.begin(9696);

  ////////////////////////////
  // MOVEMENT RELATED
  ////////////////////////////

  // Establishing the servos in an array
  for (int i = 0; i < 0 + numberOfServos; i++) {
    myServo[i].attach(servPins[i]);
    myServo[i].write(servZero[i]);
  }
}

void loop()
```

```
{
  unsigned long t0 = millis();
  //Checks for incoming packets
  char command = checkPackets();
  //Serial.print(command);
  if (command != 'z') {

    // If packet is f, move forward
    if (command == 'f') {
      //Serial.println("Going forward");
      goingForward = true;
      goingBackward = false;
      sendAliveMessage();

      // If packet is b, move backwards
    } else if (command == 'b') {
      //Serial.println("Going backwards");
      goingForward = false;
      goingBackward = true;
      sendAliveMessage();

      // If packet is v, straighten out, lateral shift left
    } else if (command == 'v') {
      //Serial.println("Adjusting left");
      sendAliveMessage();
      latLeft = true;
      latRight = false;
      goStraight();

      // If packet is h, straighten out, lateral shift right
```

```
  } else if (command == 'h') {
    //Serial.println("Adjusting right");
    sendAliveMessage();
    latRight = true;
    latLeft = false;
    goStraight();

    // If packet is m, rotate CW
  } else if (command == 'm') {
    //Serial.println("Rotating CW");
    sendAliveMessage();
    rotCW = true;
    rotCCW = false;
    //goStraight();

    // If packet is n, rotate CCW
  } else if (command == 'n') {
    //Serial.println("Rotating CCW");
    sendAliveMessage();
    rotCCW = true;
    rotCW = false;
    //goStraight();

    // If packet is s, stop movement
  } else if (command == 's') {
    //Serial.println("Stopping movement");
    goingForward = false;
    goingBackward = false;
    latRight = false;
    latLeft = false;
```

```
    sendAliveMessage();

    // If packet is r, adjust everything straight
  } else if (command == 'r') {
    //Serial.println("Adjusting straight");
    sendAliveMessage();
    goingForward = false;
    goingBackward = false;
    latRight = false;
    latLeft = false;
    goStraight();
    sendDoneMessage();

    // If packet is t, change the turn-angle
  } else if (command == 't') {
    sendAliveMessage();
    int numb1 = (int)packetBuffer[1] - 48;
    int numb2 = (int)packetBuffer[2] - 48;
    int numb3 = (int)packetBuffer[3] - 48;
    int sum = numb1 * 100 + numb2 * 10 + numb3;
    //Serial.println(sum);
    turn(sum);
    sendDoneMessage();

    // If packet is p, change the T-parameter
  } else if (command == 'p') {
    int numb1 = (int)packetBuffer[1] - 48;
    int numb2 = (int)packetBuffer[2] - 48;
    int numb3 = (int)packetBuffer[3] - 48;
    int sum = numb1 * 100 + numb2 * 10 + numb3;
```

```
      sum = sum * 1000;
      T = sum;
      sendAliveMessage();

      // If packet is a, change the A-parameter
    } else if (command == 'a') {
      int numb1 = (int)packetBuffer[1] - 48;
      int numb2 = (int)packetBuffer[2] - 48;
      int sum = numb1 * 10 + numb2;
      if (sum > 80) {
        sum = 80;
      }
      A = sum;
      sendAliveMessage();

      // If packet is anything else, send error to UDP server
    } else {
      //Serial.println("Unknown command");
      sendErrorToServer();
    }
  }

  // If boolean goingForward is high, go forward one cycle
  if (goingForward) {
    goForward();
    movementTimer++;
    if (movementTimer >= T) {
      goingForward = false;
      sendDoneMessage();
      movementTimer = 0;
```

```
    }

    // If boolean goingBackward is high, go backward one cycle
  } else if (goingBackward) {
    goBackward();
    movementTimer++;
    if (movementTimer >= T) {
      goingBackward = false;
      sendDoneMessage();
      movementTimer = 0;
    }
    // If boolean latLeft is high, lateral shifts left one cycle
  } else if (latLeft) {
    lateralLeft();
    movementTimer++;
    if (movementTimer >= T) {
      latLeft = false;
      //delay(10);
      //goStraight();
      sendDoneMessage();
      movementTimer = 0;
    }
    // If boolean latRight is high, lateral shifts right one cycle
  } else if (latRight) {
    lateralRight();
    movementTimer++;
    if (movementTimer >= T) {
      latRight = false;
      //delay(10);
      //goStraight();
```

```
      sendDoneMessage();
      movementTimer = 0;
    }
    // If boolean rotCW is high, rotates clockwise one cycle
  } else if (rotCW) {
    rotateCW();
    movementTimer++;
    if (movementTimer >= T) {
      rotCW = false;
      //delay(10);
      //goStraight();
      sendDoneMessage();
      movementTimer = 0;
    }
    // If boolean rotCCW is high, rotates counter-clockwise one cycle
  } else if (rotCCW) {
    rotateCCW();
    movementTimer++;
    if (movementTimer >= T) {
      rotCCW = false;
      //delay(10);
      //goStraight();
      sendDoneMessage();
      movementTimer = 0;
    }
  }
}
```

```
////////////////////////////////
// WIFI FUNCTIONS
////////////////////////////////

// Checks for incoming packets, stores in the buffer.
char checkPackets() {
  if (udpClient.parsePacket()) {
    //Serial.println("Packet received");
    udpClient.read(packetBuffer, 128);
    //Serial.println(char(packetBuffer[0]));
    return packetBuffer[0];
  } else {
    return 'z';
  }
}

// Sends error message to UDP-server
void sendErrorToServer() {
  udpClient.beginPacket(host, port);
  udpClient.write('x');
  udpClient.endPacket();
}


// Sends message that command is done to UDP server
void sendDoneMessage() {
  udpClient.beginPacket(host, port);
  udpClient.write('d');
  udpClient.endPacket();
}
```

```
// Sends acknowledge-message to UDP-server
void sendAliveMessage() {
  udpClient.beginPacket(host, port);
  udpClient.write('a');
  udpClient.endPacket();
}

////////////////////////////////////////
// MOVEMENT FUNCTIONS
////////////////////////////////////////
// Go forward
void goForward() {
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, i * forwardPhi,
        A));
  }
}

// Go backward
void goBackward() {
  for (int i = 2; i >= 0; i--) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, -i * forwardPhi,
        A));
  }
}

/*
   Sets the turn angle
   param deg: the turn angle in degrees
```

```
*/
void turn(int deg) {
  if (deg < 45) {
    deg = 45;
  } else if (deg > 135) {
    deg = 135;
  }
  for (int i = 0; i < 2; i++) {
    myServo[(i * 2) + 1].write(deg + servZero[(i * 2) + 1] - 90);
  }
}

/*
   Lateral shifts left
*/
void lateralLeft() {
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, -i * rotatePhiV,
        A));
    if (i < 2) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, -i
          * rotatePhiH, A));
    }
  }
}

/*
   Lateral shifts right
*/
void lateralRight() {
```

```
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, i * rotatePhiV,
        A));
    if (i < 2) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, i
          * rotatePhiH, A));
    }
  }
}

/*
   Rotates Clockwise
*/
void rotateCW() {
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, i * rotatePhiV,
        A));
    if (i == 0) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, i
          * rotatePhiH, A));
    } else if (i == 1) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, i
          * (rotatePhiH + M_PI), A));
    }
  }
}

void rotateCWv2() {
  for (int i = 0; i < 3; i++) {
```

```
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, i * v2RotatePhiV
        , A));
    if (i == 0) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, i
          * v2RotatePhiH, A));
    } else if (i == 1) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, i
          * (v2RotatePhiH + M_PI), A));
    }
  }
}

/*
   Rotates Counter-Clockwise
*/
void rotateCCW() {
  // Pass
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, -i * rotatePhiV,
        A));
    if (i == 0) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, -i
          * rotatePhiH , A));
    } else if (i == 1) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, -i
          * (rotatePhiH + M_PI) , A));
    }
  }
}
```

```
void rotateCCWv2() {
  // Pass
  for (int i = 0; i < 3; i++) {
    myServo[i * 2].write(servZero[i * 2] + updateAngle(T, -i *
        v2RotatePhiV, A));
    if (i == 0) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, -i
          * v2RotatePhiH , A));
    } else if (i == 1) {
      myServo[(i * 2) + 1].write(servZero[(i * 2) + 1] + updateAngle(T, -i
          * (v2RotatePhiH + M_PI) , A));
    }
  }
}

/*
   Sets every module straight
*/
void goStraight() {
  for (int i = 0; i < 5; i++) {
    myServo[i].write(servZero[i]);
  }
}

/*
   Updates angle of servos
   param T: period time for cycle
   param phase: offset for angle
   param A: amplitude of movement
*/
```

```
int updateAngle(float T, float phase, float A) {
  float y = A * sin(((2 * M_PI) / T) * movementTimer + phase);
  return y;
}
```

Arduino program for IMU handling

```
#include <DataRegister.h>
#include <SerialSlave.h>
#include <Wire.h>

//--------------------------------
// IMU
//--------------------------------
const int MPU_ADDR = 0x68;
int acc_offset[] = {8, 17, 208};
int gyro_offset[] = {-89, 196, 34};
int mag_offset[] = {0, 0, 0};
int acc_scale = 16384;
int gyro_scale = 131;

//--------------------------------
// Serial
//--------------------------------
DataRegister reg;
SerialSlave ser_slave;

const int num_regs = 14;
const int num_buffer = 4 + 4 + 4 + 2 + (2 * 9) + 4;
int size_array[] = {4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4};
int index_array[num_regs];
```

```
byte buffer_array[num_buffer];

//--------------------------------
// Registers
//--------------------------------
float* roll;
float* pitch;
float* yaw;
unsigned int* loop_time;
int* rax;
int* ray;
int* raz;
int* rgx;
int* rgy;
int* rgz;
int* rmx;
int* rmy;
int* rmz;
float* comp_alpha; // Complementary filter alpha

void setup() {
  Serial.begin(115200);
  reg = DataRegister(num_buffer, num_regs, buffer_array, size_array,
     index_array);
  ser_slave.set_register(reg);

  roll      = (float*)         reg.link(0);  *roll      = 0;
  pitch     = (float*)         reg.link(1);  *pitch     = 0;
  yaw       = (float*)         reg.link(2);  *yaw       = 0;
  loop_time = (unsigned int*) reg.link(3);  *loop_time = 10;
```

```
  rax        = (int*)          reg.link(4);  *rax        = 0;
  ray        = (int*)          reg.link(5);  *ray        = 0;
  raz        = (int*)          reg.link(6);  *raz        = 0;
  rgx        = (int*)          reg.link(7);  *rgx        = 0;
  rgy        = (int*)          reg.link(8);  *rgy        = 0;
  rgz        = (int*)          reg.link(9);  *rgz        = 0;
  rmx        = (int*)          reg.link(10); *rmx        = 0;
  rmy        = (int*)          reg.link(11); *rmy        = 0;
  rmz        = (int*)          reg.link(12); *rmz        = 0;
  comp_alpha = (float*)        reg.link(13); *comp_alpha = 0.9;

  Wire.begin();
  imu_wakeup();
}

void loop() {
  unsigned long t0 = millis();

  // do stuff
  imu_read();
  calculate_orientation();

  ser_slave.scan();
  while (millis() < t0 + *loop_time) {
    ser_slave.scan();
  }
}

//--------------------------
//  IMU FUNCTIONS
```

```
//----------------------------
void imu_wakeup() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission();
}

void imu_read() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 3 * 2);
  *rax = Wire.read() << 8 | Wire.read();
  *ray = Wire.read() << 8 | Wire.read();
  *raz = Wire.read() << 8 | Wire.read();

  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x43);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 3 * 2);
  *rgx = Wire.read() << 8 | Wire.read();
  *rgy = Wire.read() << 8 | Wire.read();
  *rgz = Wire.read() << 8 | Wire.read();

  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x03);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 3 * 2);
  *rmx = Wire.read() << 8 | Wire.read();
```

```
  *rmy = Wire.read() << 8 | Wire.read();
  *rmz = Wire.read() << 8 | Wire.read();
}

void calculate_orientation() {
  int min_scale = -32760;
  int max_scale = 32760;
  float acc_x =  lin_map(*rax, min_scale, max_scale, -2,    2,
     acc_offset[0]);
  float acc_y =  lin_map(*ray, min_scale, max_scale, -2,    2,
     acc_offset[1]);
  float acc_z =  lin_map(*raz, min_scale, max_scale, -2,    2,
     acc_offset[2]);
  float gyro_x = lin_map(*rgx, min_scale, max_scale, -250,  250,
     gyro_offset[0]);
  float gyro_y = lin_map(*rgy, min_scale, max_scale, -250,  250,
     gyro_offset[1]);
  float gyro_z = lin_map(*rgz, min_scale, max_scale, -250,  250,
     gyro_offset[2]);
  float mag_x =  lin_map(*rmx, min_scale, max_scale, -4912, 4912,
     mag_offset[0]);
  float mag_y =  lin_map(*rmy, min_scale, max_scale, -4912, 4912,
     mag_offset[1]);
  float mag_z =  lin_map(*rmz, min_scale, max_scale, -4912, 4912,
     mag_offset[2]);

  float acc_roll = atan2(acc_y, acc_z) * 180.0 / PI; // Converted to
     degrees
  float acc_pitch = atan2(-acc_x, sqrt(pow(acc_y, 2) + pow(acc_z, 2))) *
     180.0 / PI;
```

```
  float mag_yaw = atan2(mag_y, mag_x) * 180.0 / PI;

  float dt = (float) * loop_time / 1000;
  *roll  = *comp_alpha * (*roll  + gyro_x * dt) + (1 - *comp_alpha) *
      acc_roll;
  *pitch = *comp_alpha * (*pitch + gyro_y * dt) + (1 - *comp_alpha) *
      acc_pitch;
  *yaw   = *comp_alpha * (*yaw   + gyro_z * dt) + (1 - *comp_alpha) *
      mag_yaw;
}

float lin_map(float x, float x0, float x1, float y0, float y1) {
  return (x - x0) * (y1 - y0) / (x1 - x0) + y0;
}

float lin_map(float x, float x0, float x1, float y0, float y1, float offs)
    {
  return lin_map(x - offs, x0, x1, y0, y1);
}
```

Appendix D

Python Code & Config-Files

filterLidarData

```
#! /usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist, Vector3
from sensor_msgs.msg import LaserScan
import tf
import math
scan_msgs = None
new_scan = False
roll = 0
pitch = 0
yaw = 0
new_pitch_msg = False
br = tf.TransformBroadcaster()


def handle_pitch(msg):
    global roll, pitch, yaw, new_pitch_msg
    roll = msg.x
    pitch = msg.y
```

```
    yaw = msg.z
    q0 = tf.transformations.quaternion_from_euler(0, 0, 0)
    yaw_q = tf.transformations.quaternion_from_euler(0, 0, yaw/180*math.pi
        )
    pr_q = tf.transformations.quaternion_from_euler(roll/180*math.pi,
        pitch/180*math.pi, 0)
    br.sendTransform((0, 0, 0), yaw_q, rospy.Time.now(), 'base_footprint',
         'odom')
    br.sendTransform((0, 0, 0), q0, rospy.Time.now(), 'base_stabilized', '
        base_footprint')
    br.sendTransform((0, 0, 0), pr_q, rospy.Time.now(), 'base_link', '
        base_stabilized')
```

```
pub = rospy.Publisher("scan2", LaserScan, queue_size=10)

def handle_laser_scan(msg):
    global scan_msgs, new_scan, pub
    if pitch > -10 and pitch < 10:
        pub.publish(msg)

if __name__ == "__main__":
    rospy.init_node("PitchFilter", anonymous=True)
    rospy.Subscriber("Pitch", Vector3, handle_pitch)
    rospy.Subscriber("scan", LaserScan, handle_laser_scan)
    rospy.spin()
```

Translater

```
#! /usr/bin/env python3

import rospy
import math
import socket
import time
from geometry_msgs.msg import Twist, Vector3
from std_msgs.msg import String


forward = 0
turn = 0


def handle_cmd_vel(msg):
    global forward, turn
    linear = msg.linear
    angular = msg.angular
    lx = linear.x
    ly = linear.y
    lz = linear.z
    ax = angular.x
    ay = angular.y
    az = angular.z
    forward = lx
    turn = int(math.degrees(az))


def main():
    global forward, turn
```

```
snak = Snake("192.168.137.174")
print("connected to snake")
moving = False
turning = False
snak.setAmplitude(30)
time.sleep(5)
while not rospy.is_shutdown():
    if moving:
        done = snak.isCommandDone()  # handel time out.
        if done:
            moving = False
    elif turning:
        ack = snak.moveForward()
        if ack:
            turning = False
            moving = True

    elif not moving and not turning:  # roter
        if forward == 0 and turn != 0:
            if turn < 0:
                ack = snak.rotateCW()



            else:
                ack = snak.rotateCCW()
            if ack:
                moving = True
        elif forward != 0 and turn != 0:
            ack = snak.turn(turn)
            if ack:
```

```
                    moving = True
                    turning = True
                # forward
            elif forward != 0 and turn == 0:
                ack = snak.moveForward()
                if ack:
                    moving = True
            else:
                snak.controller.receive()
        # if not moving:
        #     ack = snak.rotateCW()
        #     if ack:
        #         moving = True




class UdpConnection:


    def __init__(self, url: str):
        """
        :param url: The url to communicate with
        """
        self.pub = rospy.Publisher('snake_command', String, queue_size=10)
        self.timeOutTimer = time.time()
        self.connectionTimedOut = False
        self.timeOut = 3

        self.url = url
```

```
        udp_ip = "192.168.137.11"
        udp_port = 6969
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  #
            UDP
        self.socket.bind((udp_ip, udp_port))
        self.socket.settimeout(0.01)
        self.pitchData = None
        self.newPitchData = False


    def send(self, data: str) -> None:
        """
        Sends a udp diagram \n
        :param data: data to send
        :return: None
        """
        print("sendingData")
        print(data)
        string = String()
        string.data = data
        self.pub.publish(string)
        self.socket.sendto(data.encode(), (self.url, 9696))

    def receive(self):
        """
        Receives data
        :return: The body of the data
        """
        data = None
```

```
        try:
            data, addr = self.socket.recvfrom(10024)  # buffer size is
                10024 bytes
            print(data)
            data = data.decode()

        except socket.timeout:
            pass

        return data


class Snake:

    def __init__(self, controllerIp: str):
        self.controller = UdpConnection(controllerIp)
        self.timeOutTime = 0.5
        self.pitcharray = [0, 0, 0, 0, 0]

    def timeOut(self):
        """
        Checks for acknowledge from the snake. Times out after a set
            amount of time given from self.timeOutTime
        :return:  True if acknowledged, False if timed out
        """
        timeOutTime = time.time() + self.timeOutTime
        while True:
            data = self.controller.receive()
            if data == "a":
                return True
```

```
            if time.time() > timeOutTime:
                return False

    def setSpeed(self, speed: int):
        """
        Sets the period time of the cycle for the snake
        :param speed: the period time in 3 digits
        :return: True if acknowledged from snake
        """
        send = ""
        if speed < 10:
            send = "p00" + str(speed)
        elif speed < 100:
            send = "p0" + str(speed)
        else:
            send = "p" + str(speed)
        self.controller.send(send)
        return self.timeOut()

    def setAmplitude(self, amplitude: int):
        """
        Sets the amplitude of the snakes movement
        :param amplitude: amplitude in 2 digits
        :return: True if acknowledged from snake
        """
        send = ""
        if amplitude < 10:
            send = "a0" + str(amplitude)
        else:
            send = "a" + str(amplitude)
```

```
        self.controller.send(send)
        return self.timeOut()

    def moveForward(self):
        """
        Gives the command to the snake to move forward one cycle
        :return: True if acknowledged
        """
        self.controller.send("f")
        print("sending an F")
        return self.timeOut()

    def moveBackward(self):
        """
        Gives the command to the snake to move backward one cycle
        :return: True if acknowledged
        """
        self.controller.send("b")
        return self.timeOut()

    def turn(self, degrees: int):
        """
        Sends a turn command to the snake. \n
        positive degrees is right and negative is left \n
        :param degrees: turnrate in degrees
        :return: True if acknowledged
        """
        send = None
        degrees = 90 + degrees
```

```
        if degrees < 10:
            send = "t00" + str(degrees)
        elif degrees < 100:
            send = "t0" + str(degrees)
        else:
            send = "t" + str(degrees)
        self.controller.send(send)
        return self.timeOut()

    def moveLeft(self):
        """
        Sends command to snake to lateral shift left
        :return: True if acknowledged
        """
        self.controller.send("v")
        return self.timeOut()

    def moveRight(self):
        """
        Sends command to snake to lateral shift right
        :return: True if acknowledged
        """
        self.controller.send("h")
        return self.timeOut()

    def rotateCCW(self):
        """
        Sends command to snake to rotate counter-clockwise
        :return: True if acknowledged
        """
```

```
        self.controller.send("n")
        return self.timeOut()

    def rotateCW(self):
        """
        Sends command to snake to rotate clockwise
        :return: True if acknowledged
        """
        self.controller.send("m")
        return self.timeOut()

    def stop(self):
        """
        Sends command to snake to stop movement
        :return: True if acknowledged
        """
        self.controller.send("s")
        return self.timeOut()

    def reset(self):
        """
        Sends command to reset positions to zero point.
        :return: True if acknowledged
        """
        self.controller.send("r")
        return self.timeOut()

    def isCommandDone(self) -> bool:
        """
        Checks if the snake is done with the last command
```

```
        :return: True if command is done
        """
        data = self.controller.receive()
        if data == "d":
            return True
        else:
            return False


if __name__ == "__main__":
    rospy.init_node("translater", anonymous=True)
    rospy.Subscriber("cmd_vel", Twist, handle_cmd_vel)
    print("starting1234")
    main()
```

Logger

```
#! /usr/bin/env python3
import csv
import rospy
from datetime import datetime
from threading import Lock
import time
from geometry_msgs.msg import Twist, Vector3
from std_msgs.msg import String
from pathlib import Path


class Logger:

    def __init__(self, fields, id):
```

```
        self.sub = None  # rospy topic subscriber
        self.handler = None
        self.id = id
        self.fields = fields
        self.data = []
        self.datalock = Lock()

    def make_log(self, folder: str):
        if len(self.data) != 0:
            date_obj = datetime.now()
            log_name = f'{folder}/{self.id} {date_obj.date()} {date_obj.
                hour:02d}-{date_obj.minute:02d}-{date_obj.second:02d}.txt'
            # filename = Path(log_name)
            # filename.touch(exist_ok=True)  # will create file, if it
                exists will do nothing
            # log_name = f'{folder}/{self.id}.txt'

            with open(log_name, 'w', newline='') as csv_file:
                try:
                    writer = csv.DictWriter(csv_file, fieldnames=['time']
                        + self.fields)
                    writer.writeheader()
                    with self.datalock:
                        # t0 = self.data[0]['time']
                        # for d in self.data:
                        #     d['time'] = d['time'] - t0
                        writer.writerows(self.data)
                        self.data = []
                        print('saved log !!!!!!!!!!!!!!!!!!!!!')
```

```
                except IndexError:
                    print(f'Index error: {log_name}')
        else:
            print('log no data')

    def set_handler(self, handler, topic, message_type):
        self.sub = rospy.Subscriber(topic, message_type, handler)
        self.handler = handler


def main():
    log_interval = 60  # seconds

    fieldnames1 = ['roll', 'pitch', 'yaw']
    fieldnames2 = ['cmd']
    logger1 = Logger(fieldnames1, 'Orientation')
    logger2 = Logger(fieldnames2, 'Commands')

    def handler1(msg):
        # Format the message as a dictionary. Example below
        # data = {k: random.randint(0, 100) for k in logger1.fields}
        # data['time'] = time.time()
        data = {'roll': msg.x, 'pitch': msg.y, 'yaw': msg.z, 'time': time.
            time()}
        with logger1.datalock:
            logger1.data.append(data)

    def handler2(msg):
        data = {'cmd': msg.data, 'time': time.time()}
        with logger2.datalock:
```

```
            logger2.data.append(data)

    logger1.set_handler(handler1, 'Pitch', Vector3)
    logger2.set_handler(handler2, 'snake_command', String)

    t0 = time.time()
    while not rospy.is_shutdown():
        #logger1.handler('some_msg')  # For testing
        # logger2.handler('some_msg')  # For testing
        #time.sleep(0.01)  # For testing
        if time.time() > t0 + log_interval:
            logger1.make_log('/home/marcus/Documents/Loggs/Orientation')
            logger2.make_log('/home/marcus/Documents/Loggs/Commands')
            t0 = time.time()
            # logger2.make_log('logs')
    logger1.make_log('/home/marcus/Documents/Loggs/Orientation')
    logger2.make_log('/home/marcus/Documents/Loggs/Commands')


if __name__ == '__main__':
    rospy.init_node("logger", anonymous=True)
    main()
```

imu_node

```
#! /usr/bin/env python3

import rospy
import serial
import struct
import time
```

```
from geometry_msgs.msg import Twist, Vector3


def create_read_msg(regs: list, register_map: dict):
    header = struct.pack('B', 0xff) + struct.pack('B', 0xaa) + struct.pack
        ('B', len(regs))
    msg = b''
    for r in regs:
        msg += struct.pack('B', register_map[r]['num'])
    return header + msg


def create_write_msg(regs: list, register_map: dict):
    header = struct.pack('B', 0xff) + struct.pack('B', 0xbb) + struct.pack
        ('B', len(regs))
    msg = b''
    for r, v in regs:
        msg += struct.pack('B', register_map[r]['num'])
        msg += struct.pack(register_map[r]['type'][0], v)
    return header + msg


def read(regs: list, ser: serial.Serial, register_map: dict):
    data = {}
    expected_bytes = sum([register_map[r]['type'][1] for r in regs])
    response = ser.read(expected_bytes)
    i0 = 0
    for r in regs:
        i = i0 + register_map[r]['type'][1]
        part = response[i0:i]
```

```
            data[r] = struct.unpack(register_map[r]['type'][0], part)[0]
            i0 = i
        return data


def main():
    pub = rospy.Publisher("Pitch", Vector3, queue_size=10)

    port = "/dev/ttyUSB0"
    baudrate = 115200
    timeout = 2

    registers = {
      "roll": {
        "num": 0,
        "type": ["f", 4]
      },
      "pitch": {
        "num": 1,
        "type": ["f", 4]
      },
      "yaw": {
        "num": 2,
        "type": ["f", 4]
      },
      "loop_time": {
        "num": 3,
        "type": ["H", 2]
      },
      "rax": {
```

```
    "num": 4,
    "type": ["h", 2]
  },
  "ray": {
    "num": 5,
    "type": ["h", 2]
  },
  "raz": {
    "num": 6,
    "type": ["h", 2]
  },
  "rgx": {
    "num": 7,
    "type": ["h", 2]
  },
  "rgy": {
    "num": 8,
    "type": ["h", 2]
  },
  "rgz": {
    "num": 9,
    "type": ["h", 2]
  },
  "rmx": {
    "num": 10,
    "type": ["h", 2]
  },
  "rmy": {
    "num": 11,
    "type": ["h", 2]
```

```
    },
    "rmz": {
      "num": 12,
      "type": ["h", 2]
    },
    "comp_alpha": {
      "num": 13,
      "type": ["f", 4]
    }
}

while not rospy.is_shutdown():
    with serial.Serial(port=port, baudrate=baudrate, timeout=timeout)
        as ser:
        time.sleep(5)
        wregs = ['loop_time']
        wvalues = [10]

        msg = create_write_msg(list(zip(wregs, wvalues)), registers)
        ser.write(msg)
        initial_response = ser.read(1)
        print(initial_response)

        r = rospy.Rate(100)  # 10hz
        while not rospy.is_shutdown():
            regs = ['roll', 'pitch', 'yaw']

            msg = create_read_msg(regs, registers)
            ser.write(msg)
            initial_response = ser.read(1)
```

```
if initial_response == b'\xaa':
    data = read(regs, ser, registers)
    msg = Vector3()
    msg.x = data['roll']
    msg.y = data['pitch']
    msg.z = data['yaw']
    pub.publish(msg)
else:
    break


r.sleep()
# end_time = time.time() + 120
# regs = ['roll', 'pitch', 'yaw']# ['rax', 'ray', 'raz', '
   rgx', 'rgy', 'rgz', 'rmx', 'rmy', 'rmz']
# cal_list = {r: [] for r in regs}
# while time.time() < end_time:
#     time.sleep(0.02)
#     msg = create_read_msg(regs, registers)
#     ser.write(msg)
#     initial_response = ser.read(1)
#     data = read(regs, ser, registers)
#     print(data)
#     for r in regs:
#         a = cal_list[r]
#         a.append(data[r])
#         cal_list[r] = a

# rax_avg = sum(cal_list['rax']) / len(cal_list['rax'])
# ray_avg = sum(cal_list['ray']) / len(cal_list['ray'])
# raz_avg = sum(cal_list['raz']) / len(cal_list['raz'])
```

```
            # rgx_avg = sum(cal_list['rgx']) / len(cal_list['rgx'])
            # rgy_avg = sum(cal_list['rgy']) / len(cal_list['rgy'])
            # rgz_avg = sum(cal_list['rgz']) / len(cal_list['rgz'])
            # rmx_avg = sum(cal_list['rmx']) / len(cal_list['rmx'])
            # rmy_avg = sum(cal_list['rmy']) / len(cal_list['rmy'])
            # rmz_avg = sum(cal_list['rmz']) / len(cal_list['rmz'])
            #
            # print(rax_avg, ray_avg, raz_avg, rgx_avg, rgy_avg,
               rgz_avg, rmx_avg, rmy_avg, rmz_avg)
            # target = [0, 0, int(65535/4), 0, 0, 0]
            # real = [rax_avg, ray_avg, raz_avg, rgx_avg, rgy_avg,
               rgz_avg]
            #
            # for t, r in zip(target, real):
            #     print(r-t, r, t) # gy_target = gy_real - offset


if __name__ == '__main__':
    rospy.init_node("imu_node", anonymous=True)
    main()
```

Logg viewer

```
import numpy as np
import matplotlib.pyplot as plt
# import decorators as deco
import sys, getopt
import os


# @deco.dir_active(__file__)
```

```
def main(log_path: str, plot_names=None, scale_names=None):
    """
    Displays columns from a csv file as a plot. Specific columns can be
        selected from header names
    :param log_path: path to csv file
    :param plot_names: list containing header names of columns to plot.
        Default all
    :param scale_names: list containing scale factors for each column.
    :return: None
    """
    with open(log_path, 'r') as log_file:
        names = log_file.readline().strip().split(',')
    plot_names = names if plot_names is None else plot_names
    scale_names = [1 for _ in plot_names] if scale_names is None else
        scale_names

    data = np.genfromtxt(log_path, delimiter=',', names=names, skip_header
        =1)  # Read CSV file

    time0 = data['time']  # Extract time column
    names.remove('time')
    t0 = time0[0]  # 0 dersom klokketid
    time = [t - t0 for t in time0]

    plt.figure()  # Create figure

    # Plot selected variables
    plotted_names = []
    for name in names:
        if name in plot_names:  # Ignore typo in plot_names
```

```
            plotted_names.append(name)
            index = plot_names.index(name)  # Find index of current name
                in plot_names list
            scale = scale_names[index]      # Extract scale corresponding
                to name
            y = [d * scale for d in data[name]]  # Scale values in column
            plt.plot(time, y)  # Add data vs time to plot
    plt.grid()
    plt.legend(plotted_names)
    plt.xlabel('Time [s]')
    plt.ylabel('Angle [deg]')
    plt.title('Lidar filter test, filter > 10.')
    plt.show()


if __name__ == '__main__':
    path = os.path.dirname(os.path.abspath(__file__))
    os.chdir(path)

    def parse():
        """
        Function for parsing commandline arguments and options
        :return: filename, kwargs
        """
        kwarguments = {}

        def help_func(*help_args):
            print(f"python log-viewer.py <opts> <args>\n"
                  f"    opts:\n"
                  f"        -n: String with variable names to plot i.e '
```

```
            var1, var2, var3'\n"
        f"        -s: String with scales for plotted variables i
            .e '1, 10, 2'\n"
        f"        -h: Help\n"
        f"    args:\n"
        f"        filename (is assumed to be located in 'logs/'
            relative to script)")
    exit(0)

def n(names):
    kwarguments['plot_names'] = [text.strip() for text in names.
        strip().split(',')]

def s(scales):
    kwarguments['scale_names'] = [float(text.strip()) for text in
        scales.strip().split(',')]

try:
    opt_dict = {'-h': help_func, '-n': n, '-s': s}
    argv = sys.argv[1:]
    opts, args = getopt.getopt(argv, 'hn:s:', ['plotnames=', '
        scalenames='])
    for opt, arg in opts:
        opt_dict[opt](arg)
    filename = args[0]
    return filename, kwarguments
except IndexError as e:
    print(f'{e}: Has the filename been supplied?')
    help_func()
except getopt.GetoptError as e:
```

```
            print(f'{e}')
            help_func()

    file, kwargs = parse()
    main(f'/home/marcus/Documents/Loggs/{file}', **kwargs)
```