DEGREE PROJECT

3D modeling of mineshaft using autonomous quad rotor

Lars Jonsson

Engineering Physics and Electrical Engineering, master's level
2017

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Master's Thesis in Engineering Physics and Electrical Engineering

3D modeling of mineshaft using autonomous quad rotor

Author: Lars Jonsson

Examiner: Per Lindgren

Supervisor: Johan Eriksson

2017-08-30

Abstract

In this master's thesis a multirotor with the ability to scan its surroundings was built. To be able to produce these scans the multirotor will be equipped with a custom built 3D LIDAR. In the future, the scans will be used to generate a 3D map to visualize mineshafts in a well suited way for inspections.

This multirotor is designed with the purpose to map mineshafts that are inaccessible to humans, due to safety reasons. To produce a 3D map of the multirotor surroundings the absolute position is needed. Since the multirotor will be used in an environment where GPS is unavailable, the positioning is solved by utilizing an IMU and Computer Vision technology with a Ranging device.

The functionality has been tested in a lab environment resembling real life operational conditions, and confirms that it is possible to use this approach to scan an environment where it is possible to have the multirotor in line-of-sight for the camera.

The 3D scanning is relaying on a stable Wi-Fi connection and absolute position and as long as this is established it is possible to use this point cloud for inspection.

The positioning of the multirotor is tested up to a range of 40 m, with a maximum measured accuracy of $\pm$ 5 cm, which is well within the range of the requirements of the application.

Preface

This thesis concludes my master's degree in Engineering Physics and Electrical Engineering with specialization towards embedded systems, control theory and electronics from Luleå University of Technology.

I started working with this project in the fall of 2016 as a part of a group taking the project course D70039E-E7025E. As the project course ended I was asked to continue working with this project as my master's thesis.

Special thanks to my supervisor Johan Eriksson at CONEX and my examiner Per Lindgren at LTU for the opportunity to work on this project with the latest technology, as well as for their support and helpful discussions.

Also I would like to thank my classmate Max Unander for the many hours spent together in the project room while working on this project, with many interesting discussions and sharing ideas.

Last but not least I would like to thank my family, my partner Julia and my daughter Thea for supporting me during my studies and all late nights spent at this thesis.

Lars Jonsson, Böle

Table of Contents

List of Figures **III**

1 Introduction **1**
1.1 Background . . . 1
1.1.1 Project background . . . 1
1.2 Goals . . . 2
1.2.1 Scope . . . 3
1.3 Motivation . . . 3
1.4 Outline . . . 3

2 Method **5**
2.1 Copter positioning . . . 7
2.1.1 Ranging Device . . . 7
2.1.2 Computer Vision . . . 7
2.1.3 Camera pixel to distance calculation . . . 10
2.1.4 Kalman Filter . . . 15
2.1.5 IMU . . . 18
2.2 Distance Measurement . . . 19
2.2.1 LIDAR . . . 19
2.3 Mapping of coordinate systems . . . 23
2.3.1 Rotation and Translation matrices . . . 24
2.3.2 Mapping . . . 25
2.4 Software . . . 28
2.4.1 Main function . . . 28
2.4.2 Interrupts . . . 30
2.4.3 Data sending protocol . . . 32
2.4.4 Data acquisition . . . 33
2.5 3D model . . . 35

3 Results **37**
3.1 Copter Positioning . . . 37
3.1.1 Camera pixel conversion . . . 37
3.1.2 Computer Vision . . . 38
3.1.3 Kalman Filter . . . 39
3.2 3D point cloud . . . 42

4 Discussion **45**
4.1 Copter Positioning 45
4.1.1 Camera pixel conversion 45
4.1.2 Computer Vision 45
4.1.3 Kalman Filter 46
4.2 3D point cloud 46
4.3 Future Work 47
4.4 Conclusions 47

Appendices

A Parts list **51**

B Nucleo Shield **53**

C Power Distrubution board **57**

D Matlab code **59**
D.1 Pointcloud generation script 59
D.2 Mapping of coordinate systems 63

Bibliography **65**

List of Figures

2.1 Component chart showing how the components are connected with each other and which communication protocols that are being used. 5

2.2 Picture showing the Shield mounted on top of the Nucleo development board, it fits to the connectors mounted on the Nucleo and can be easily detached. The Inertial Measurement Unit (IMU) is inserted into its socket and at the top right we can see the connector that connects to the power distribution board, powering the entire board with 3,3 V, 5 V and connects the stepper motor driver with the F7 processor pins. 6

2.3 Snapshot showing what the camera sees without any filtering or algorithm 8

2.4 Screenshots showing what the computer sees after running the vision algorithm. a) showing one detected blob which is the LED mounted on the copter, and this is marked both with a red and blue border. b) showing two detected blobs, one is marked with a red and blue border and the other only marked with a red border. 9

2.5 Figure showing the global XY-plane with X as the position of the copter in the X-direction and Y as the perpendicular distance from the camera to the copter. Where θ is defined as the angle between the X and Y position. 11

2.6 Figure showing the basic pinhole camera model in 2D showing only the X-Y plane. Used for calculating the cameras Horizontal Field Of View (HFOV) angle, α_H as well as the cameras focal length in pixels, f given the resolution of the camera and the diagonal Field Of View (FOV). 12

2.7 Figure showing how the different vertical, horizontal and diagonal FOV is defined. Showing also the the pixel resolution of the Image Plane as $H \times W$, where D is the diagonal pixel resolution [11]. 13

2.8 Pictures showing how the measurement for the pixel conversion was taken. 14

2.9 Figure showing how an Object at position X in the Object plane is represented as position, P_X in the Image plane. In Figure 2.5 you see the representation of the same object, but in the object plane looking from above, so that θ in the Image plane equals θ in Figure 2.5. 15
2.10 Figure showing the basic principle used by many a 2D Light Detection And Ranging (LIDAR). 19
2.11 Figure showing the scanning angle and the dead-zone of the LIDAR in default setup[1]. 20
2.12 Figure showing the LIDAR and the stepper motor mounted on the quad copter, also showing in which direction the stepper motor and the LIDAR is rotating. 20
2.13 The field of operation for the rotating LIDAR-setup. As the stepper motor has a full 360° revolution and the LIDAR has a dead zone of 45° we will get almost a complete sphere that the copter can detect. Since the LIDAR is mounted on top of the drone, the dead-zone is pointed towards the body of the copter to get the least amount of bad hits. 21
2.14 Figure showing the quadcopter attitude, yaw, pitch and roll axes[18]. 23
2.15 Quadcopter-fixed coordinate system in relation to the camera fixed coordinate system [6]. 25
2.16 Figure showing a high level schematic view over the interrupt driven software that is running on the MCU. 29
2.17 LAMP protocol definition - LIDAR frames 32

3.1 Plot showing how pixel width relates to distance from the camera. Theoretical calculations plotted as a blue line, and measurements plotted as red stars. All measurements can calculations is made using the cameras highest resolution, `2560x1920`. 38
3.2 Plot showing the position of the copter in X, Y and Z positions. The blue line shows the Kalman filtered positions and the orange dots represents the unfiltered positions from the Computer Vision. 40
3.3 Plot showing the Kalman filtered position and the raw unfiltered position from the Computer Vision. This is the same data that is shown in the middle subplot in Figure 3.2 but over a shorter time span, to easier see the difference between the filtered and unfiltered data. The Kalman filtered data is marked with a red line and the unfiltered Computer Vision data is marked with blue dots. 41

3.4 Plot of our first test scan, LIDAR mounted stationary at middle of the project room. Each subplot is from the same dataset but shown from different viewing angles. The software VRMesh[15] was used for plotting of the point cloud data. . . 42
3.5 30 m test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the corridor from above ($X - Y$ plane). Plotted in Matlab using the Vision-toolbox[9]. 43
3.6 30 m test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the right wall from the inside of the corridor. Plotted in Matlab using the Vision-toolbox[9]. 44
3.7 30 m test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the front entrance of the corridor, the same place as the camera is located. Plotted in Matlab using the Vision-toolbox[9]. 44

B.1 Part of the schematic of the Nucleo shield, showing the DW1000 Ultra Wide-Band (UWB) module (not in use), the shield connector to the development board, the IMU socket, the power board connector, LIDAR connector and battery to ADC connection 53
B.2 The remaining part of the schematic of the Nucleo shield. Showing of connectors to the receiver, the flight controller, sonar, hall sensor and some GPIO LED. 54
B.3 Board layout of the nucleo shield. 55

C.1 Schematic of the power distribution board with two Buck converters and one voltage regulator is used. 57
C.2 PCB layout of the power distribution board. 58

Chapter 1

Introduction

1.1 Background

Autonomous robots of all sorts are going to be common in the future, in everything from self driving cars, to Unmanned Aerial Vehicle (UAV)'s. One branch of these UAVs that are being more frequently used are the ones that have the ability of Vertical Take-Off and Landing like the multirotor. Since they need less room to take-off and land and due to their stable flight performance and fast maneuvers they can be used in many different applications. One such application could for example be to fly inside underground mineshafts.

The local mining industry is struggling with mineshafts that are exposed to wear and erosion which can widen the walls to unsafe levels. These shafts need to be scanned at regular intervals, which is expensive due to the fact that these shafts are inaccessible due to safety concerns. As of today the only way of inspecting the shafts is to drill long holes, sometimes several hundred meters and insert stationary scanners along the shafts. This method is expensive and time-consuming and therefore a cheaper and faster method is desired. This master's thesis is a part of a project to build a prototype of a drone system for underground mapping in the environments mentioned above.

1.1.1 Project background

The primary objective of the project is to test if it is possible to produce a multirotor that can fly autonomously through a mine shaft while at the same time scanning its surroundings. It should also be cheaper than the current method to use these drones. Output from a scan should be a 3D point cloud that represents the mine shaft. Since the shafts are inaccessible there will not be possible to retrieve the drone after a crash, therefore scanned data should be continuously streamed to a base-station which enables the scanned data

to be retrieved even if the drone is lost. The drone should be easily replaced and the cost of each drone should be kept at a minimum.

Part of this project was already implemented before this master thesis started. These parts were implemented by students (including me) in a project course at the fall of 2016. The report that was written for this project course showing what was already made can be viewed in the D70039E-E7025E report written by K.Alkawati et al. [7]

Here is a list showing what is already implemented:

- The drone was airborne and able to carry its entire payload long enough to safely map an entire mineshaft.
- Almost all hardware electronics was built and tested.
- Mounts for Light Detection And Ranging (LIDAR) and all hardware electronics was done.
- Communication between the flight controller and the MCU was working.
- A simple collision avoidance was implemented.
- Ethernet stack was written, but not yet finished.
- Code for communication with the Inertial Measurement Unit (IMU) was written, but not fully functional.

1.2 Goals

The primary goals of this thesis is to produce a 3D point cloud of the copters surroundings, by utilizing a LIDAR and an IMU, and to visualize this point cloud in a well suited way in order to detect larger defects on the surrounding walls.

In order to do this the position of the copter is needed, and this is made using a ranging device and Computer Vision. The position part of this thesis is done in cooperation with another master's thesis student, Unander [14].

Technical specification for this master's thesis:

- Position of copter established up to 40 m with good enough refresh rate.
- Point cloud representing the copters surrounding where it is possible to detect larger holes.

- Data streamed wirelessly from the MCU.

What is considered "good enough" will be determined through testing.

1.2.1 Scope

Some limitations regarding the goals of this thesis:

- No testing of the system inside a mine shaft is necessary, since this is only a proof of concept.
- A corridor near the project room will be used as a proving ground as it is about 40 m long and 3 m wide, making it equivalent with the proportions of a mine shaft.
- Copter should only return scans as long as it is in line-of-sight for the camera.
- Real time plotting of the scans is not required.

1.3 Motivation

Why are we doing this project, and where can it be used? The position part could be used for applications that need to utilize some sort of positioning in an environment where feature detection and/or positioning using GPS or satellite is impossible. Since there is no need for great on board computational power, the cost for each unit will be lower than other autonomous multirotor that is relying on heavy-duty Simultaneous Localization And Mapping (SLAM).

1.4 Outline

This thesis is organized as follows:

- Chapter 1 (this chapter) provides an introduction of the subject and to the whole project. It provides information about what is already implemented and what the goals are, as well as the structure of this thesis.
- Chapter 2 presents the method and theory of how everything was implemented, and which things that was needed to fulfill the requirements for the project.
- Chapter 3 presents the results of this thesis.

- Chapter 4 discusses and evaluates the results presented in chapter 3.
- Bibliography contains references to literature referred to in this thesis.
- Appendix A contains a component parts list.
- Appendix B contains schematic and layout of the Nucleo Shield.
- Appendix C contains schematic and layout of the Power distribution board.

CHAPTER 2

Method

To be able to determine the position of the walls of the shaft in 3D space (x,y,z coordinates) we need to consider two major things. First we need to know the position of the copter, or rather the position of the LIDAR. Second thing we need to know is the position of the wall relative the copter. So how is this really done?

Our approach is that we have mounted a couple of sensors on the quadrotor, we have an IMU, a LIDAR, a Ranging Device, a Wi-Fi module and a bright shining LED. We will also need to have a base station (Laptop/micro controller), which also has a couple of peripherals like a camera and Wi-Fi connection. This setup is illustrated in the component chart shown in Figure 2.1. A part list with all components is presented in Appendix A.

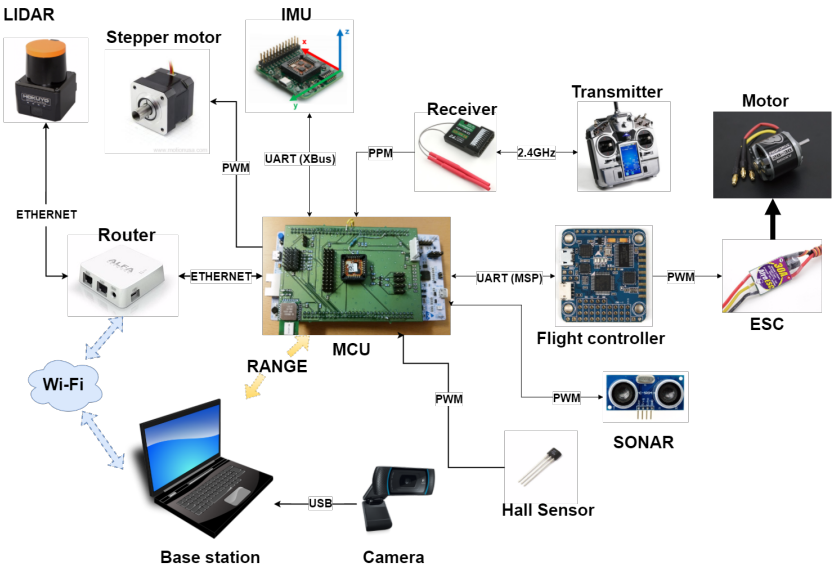


Figure 2.1: *Component chart showing how the components are connected with each other and which communication protocols that are being used.*

As our Main Computational Unit where all the communication and on board computations will be done, we are using a Nucleo development board with a `STM32F767ZI` processor. On top of this development board we have mounted a shield that, see Figure 2.2. This shield holds the connectors for the different sensors and the other circuit boards that we need to use, like the power distribution board, which isn't shown in the Component chart above.

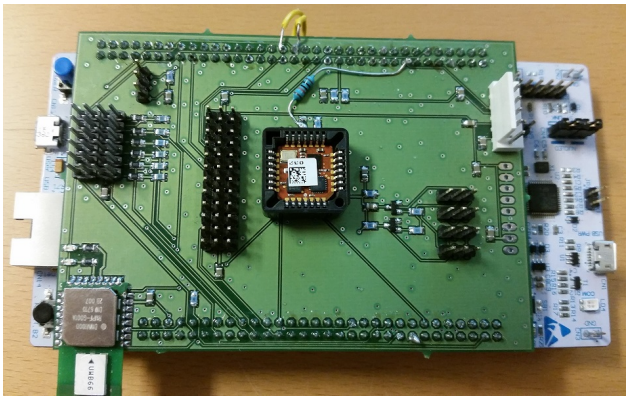


Figure 2.2: *Picture showing the Shield mounted on top of the Nucleo development board, it fits to the connectors mounted on the Nucleo and can be easily detached. The IMU is inserted into its socket and at the top right we can see the connector that connects to the power distribution board, powering the entire board with* 3,3 V, 5 V *and connects the stepper motor driver with the F7 processor pins.*

The power distribution board connects directly to the 4S ($\approx$ 16 V) Li-poly battery and distributes power to all the peripherals on the copter as well as the MCU. For complete wiring schemes of the shield and the power distribution card see Appendix B and C. Why some of these components have been chosen is written in the *Conex Copter report* by K.Alkawati et al. [see 7, Ch.5].

2.1 Copter positioning

Determining the position of an object is usually done using a GPS but since we are flying inside a mine the GPS is impossible to use. Another approach is using some sort of Indoor Positioning System (IPS) like having a couple of anchors with known position, or using the signal strength of the Wi-Fi.

Since we would like to have a position that has centimeter accuracy with high refresh rate none of the above will work. Our approach is to combine a Ranging device and computer vision with a multi-rate Kalman filter. The Ranging Device will be used for measuring the distance from the base station to the copter and the Computer Vision will determine the position of the copter in the 2D plane. The Kalman filter will be used to filter out measured noise and also increase the refresh rate in between measurements.

2.1.1 Ranging Device

The Ranging Device that we will use consists of two units, one is mounted on the copter and one is mounted at the base station. The devices utilizes Ultra Wide-Band (UWB) radio technology and uses this technology to measure the range between the units. UWB is a radio transmission technology that uses as the name says a wide bandwidth with low energy level, to not interfere other currently licensed carrier based transmissions. Both units outputs the distance between them in a serial stream. The device have an estimated range up to 200 m with an uncertainty of about 5 cm. When they have good connection with each other they have the ability to stream data at about 100 Hz.

2.1.2 Computer Vision

The Vision part of the positioning will be done using a web camera mounted to a laptop. The computer vision algorithm is coded in Python using the SimpleCV[12] framework, which is an open source framework for building computer vision applications. SimpleCV stands for Simple Computer Vision. SimpleCV is simple to use, straight forward, and quite powerful since it is mainly a OpenCV[3] wrapper, that can be used without having to first learn about bit depths, color spaces or buffer management. SimpleCV provides a concise, readable interface for cameras, image manipulation, feature extraction, and format conversion. Pseudocode for the position tracking algorithm is shown below:

Algorithm 1 Position tracking

```
Take background snapshot
loop
  Take snapshot
  Subtract background from snapshot
  Binarize the difference Image
  Run blob detection algorithm
  if Blob is found then
    Get blob pixels
    if pixel diff ≤ MAX pixel deviation then
      Get distance from Ranging device
      Calculate X,Y,Z position
      Send coordinates to copter
    end if
  end if
end loop
```

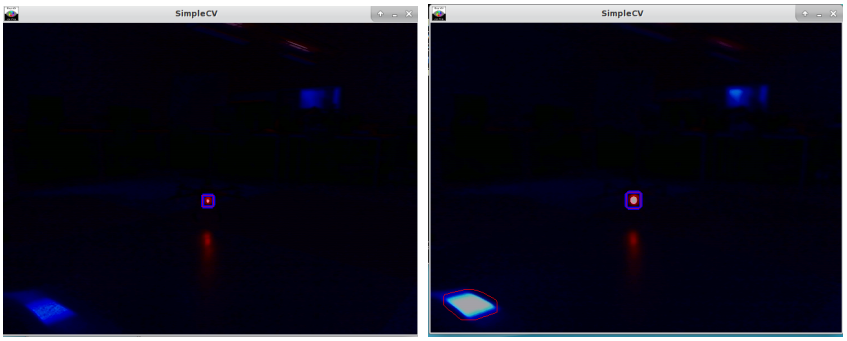
In Figure 2.3 you can see what a raw picture from the camera looks like before any vision algorithm is added.

Figure 2.3: *Snapshot showing what the camera sees without any filtering or algorithm*

The first thing that is done within this algorithm is taking a background of the environment (without the copter) that we are going to fly in, and then

subtracting this background from the snapshots taken by the camera. This is done because when flying in the corridor we need to take away all the bright spots from the image, so that the only bright spot is the LED mounted on the copter.

Since we only are taking one snapshot of the background before we are starting to fly, it is very important that the camera is properly mounted and that there are not any moving light sources. In Figure 2.4 we can see this problem, where in Figure 2.4b we are using an old image background for subtraction where the sun had not yet reached the window and thus we have a bright spot on the floor that is not subtracted. In Figure 2.4a the background image is updated and thereby the vision does not detect that blob.



(a) *Single blob detected.* **(b)** *Multiple blobs detected.*

Figure 2.4: *Screenshots showing what the computer sees after running the vision algorithm. a) showing one detected blob which is the LED mounted on the copter, and this is marked both with a red and blue border. b) showing two detected blobs, one is marked with a red and blue border and the other only marked with a red border.*

Next thing that is done is running the Blob Detection algorithm that is available within the SimpleCV library. The Blob algorithm runs over each pixel in the image and searches for chunks with bright pixels. How bright and how large these chunks need to be to get detected can be tuned. The algorithm can return the size, the brightness, and the pixel position of these chunks.

For each blob that the algorithm detects it prints out a red border around them, but it is only the one that is marked with a blue border that we are tracking. This is determined with a max pixel deviation from the previous blob we are tracking, and require that the first blob that we detect is the copter LED, this can be seen in Figure 2.4b.

When we have the position of the blob that we want to track, we then ask for a distance measurement from the Ranging device, and calculate the

X,Y,Z coordinate by utilizing "camera pixel to distance calculation" as will be shown below. These positions are then sent within a User Datagram Protocol (UDP) packet over Wi-Fi to the MCU.

An alternative algorithm to the one described earlier, has also been tested. This version of the algorithm works almost the same as previously mentioned, but instead of running blob detection on the whole image each time, the region of interest within the image is cropped, and the blob detection is run only in this region. Pseudo code for the alternative position tracking algorithm will look like:

Algorithm 2 Position tracking (Crop)

```
Take background snapshot
loop
  Take snapshot
  Crop the snapshot and background around the latest blob
  Subtract background from snapshot
  Binarize the difference Image
  Run blob detection algorithm
  if Blob is found then
    Get blob pixels
    Get distance from Ranging device
    Calculate X,Y,Z position
    Send coordinates to copter
  end if
end loop
```

The major difference between Algorithm 1 and 2 is that the blob detection only runs at a small area where the interested blob is. To find this "area of interest" the blob detection needs to be applied to the whole image at the first iteration, and the image will then be cropped around the latest blob position.

2.1.3 Camera pixel to distance calculation

While running the blob detection algorithm it is possible to find at which pixel we have a blob. But how do we convert the pixel into a distance in the global coordinate system?

To do this we need to implement some sort of pixel-to-position conversion. For this it is needed to know how wide each pixel is and at which distance

from the camera the object is located.

To calculate the distance, Y, the relationship between the X and Z position and the angle θ is needed since the measurement from the UWB only gives the shortest distance to the copter (the hypotenuse). To more easily show how this is done we can address this problem in two dimensions showing only the XY-plane, see Figure 2.5.

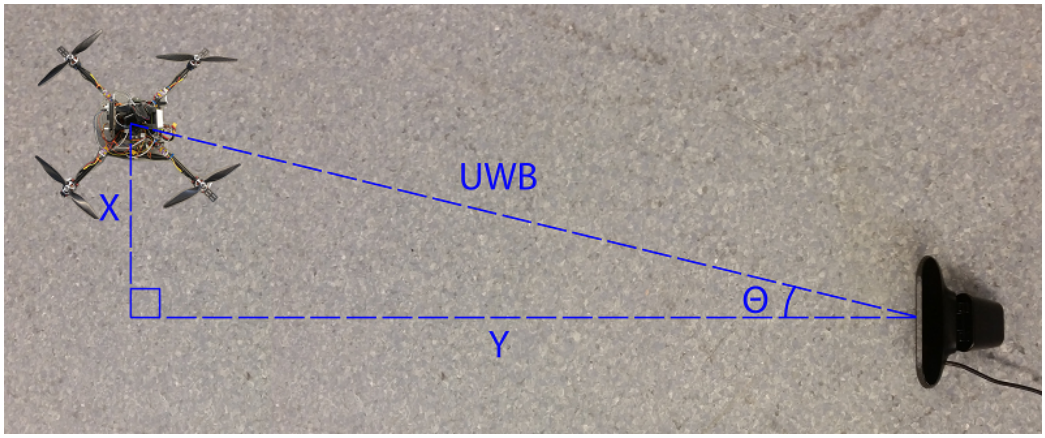


Figure 2.5: *Figure showing the global XY-plane with X as the position of the copter in the X-direction and Y as the perpendicular distance from the camera to the copter. Where θ is defined as the angle between the X and Y position.*

From Figure 2.5 we get that,

$$Y = UWB_{range}cos(\theta) \tag{2.1}$$

But since the position, X, is only known in pixels, and θ is unknown this problem has to be addressed in another way.

By seeing the camera as the most basic camera model, the pinhole model as described by Banerjee [2] and considering this problem in pixels, focal lengths and Field Of View (FOV) instead, this will result in Figure 2.6.

The camera's focal length expressed as f is not really the focal length, for these calculation it is said that f is a crude approximation of the focal length expressed as an imaginary pixel distance. In Figure 2.6 the width X_{max} is the maximum width that the camera can see at a given Distance, Y, from the lens.

The Image Plane represents what is seen on the computer screen when taking a picture with the camera, and thus W is represented as the Horizontal Pixel

resolution. As the rays within the pinhole model goes straight through the small aperture of the pinhole, the angle α_h will be the same on both sides and thus the FOV for the object plane equals the FOV of the image plane.

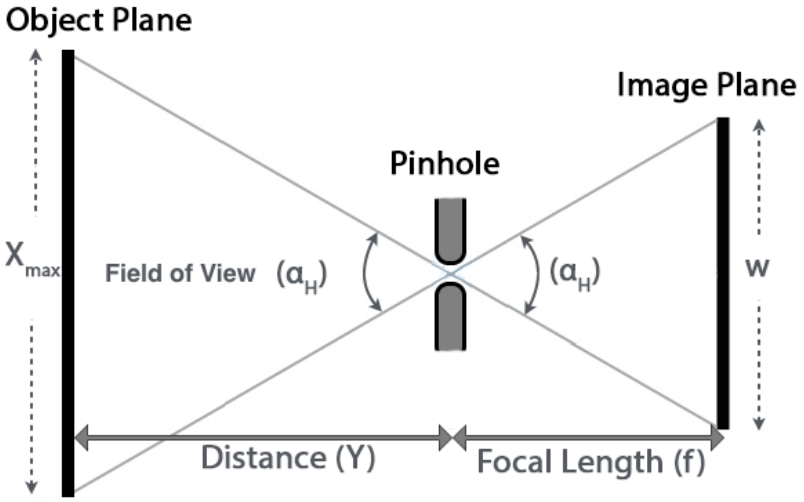


Figure 2.6: *Figure showing the basic pinhole camera model in 2D showing only the X-Y plane. Used for calculating the cameras Horizontal Field Of View (HFOV) angle, α_H as well as the cameras focal length in pixels, f given the resolution of the camera and the diagonal FOV.*

From Figure 2.6 we get that,

$$f = \frac{W}{2} cot\left(\frac{H_{FOV}}{2}\right) \tag{2.2}$$

Since the camera that is used in this project only have specification of the Diagonal Field Of View (DFOV), it needs to be converted from HFOV into DFOV. This is achieved by,

$$\begin{cases} H_{FOV} = 2tan^{-1}\left(tan\left(\frac{D_{FOV}}{2}\right) \cdot \left(\frac{H}{D}\right)\right) \\ V_{FOV} = 2tan^{-1}\left(tan\left(\frac{D_{FOV}}{2}\right) \cdot \left(\frac{V}{D}\right)\right) \end{cases} \tag{2.3}$$

where H/D and V/D is the horizontal/diagonal and vertical/diagonal aspect ratio. Another way to do this is to derive the focal length, f, directly from the DFOV. To do this we need to take this concept into 3D space, which is shown in Figure 2.7.

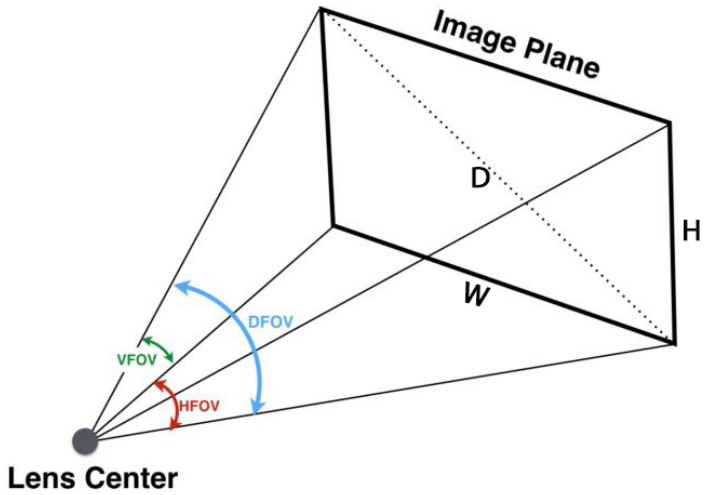


Figure 2.7: *Figure showing how the different vertical, horizontal and diagonal FOV is defined. Showing also the the pixel resolution of the Image Plane as $H \times W$, where D is the diagonal pixel resolution [11].*

Rewriting Equation 2.2 into DFOV gives,

$$f = \frac{D}{2} cot\left(\frac{D_{FOV}}{2}\right), \tag{2.4}$$

where the diagonal pixel resolution, D, is calculated from Figure 2.7 by,

$$D = \sqrt{H^2 + W^2}. \tag{2.5}$$

It is now possible to set up how the width, X_{max} depends on the distance to the Object plane, Y from Figure 2.6 by,

$$\frac{X_{max}}{Y} = \frac{W}{f}. \tag{2.6}$$

By substituting $\frac{x_{max}}{W}$ with $Pixel_{width}$ the equation can be written as,

$$Pixel_{width} = \frac{Y}{f} \quad \left[\frac{m}{Pixel}\right] \tag{2.7}$$

This equation gives the width of one pixel in meters at a given distance, Y to the object plane. To simplify things the assumption that the pixels are squares is made, thus the width of each pixel equals the height of each pixel. To verify the correctness of this theory with the approximations that were made, some tests were set up. Pictures were taken of a fixed size object at different distances. An A4 paper were used at five different distances, 4, 8, 12, 16 and 20 meters from the camera. Two of these pictures are shown in Figure 2.8.

(a) *Distance* = 4 m **(b)** *Distance* = 16 m

Figure 2.8: *Pictures showing how the measurement for the pixel conversion was taken.*

The width of this object in pixels was measured on each picture and a function for how the pixel size relates over distance was calculated and plotted, these results can be seen in Section 3.1.3. Since the pixel size is calculated it is possible to calculate the perpendicular distance to an object, if the pixel position is known as well as the UWB_{range}. By mapping Figure 2.5 into the image plane in 3D we will get something similar to Figure 2.9.

In Figure 2.9 an object is located at a pixel position P_x and P_z in the image plane, thus representing a Position X, Y, Z in the object plane. Since both P_x and P_y is known from the Computer Vision algorithm, as well as the UWB_{range}, the the perpendicular distance, Y, can be calculated by,

$$Y = UWB_{range} cos(\Phi), \tag{2.8}$$

with,

$$\Phi = tan^{-1}\left(\frac{\sqrt{P_x^2 + P_z^2}}{f}\right) \tag{2.9}$$

As this is only needed when the copter is close to the camera, small angle

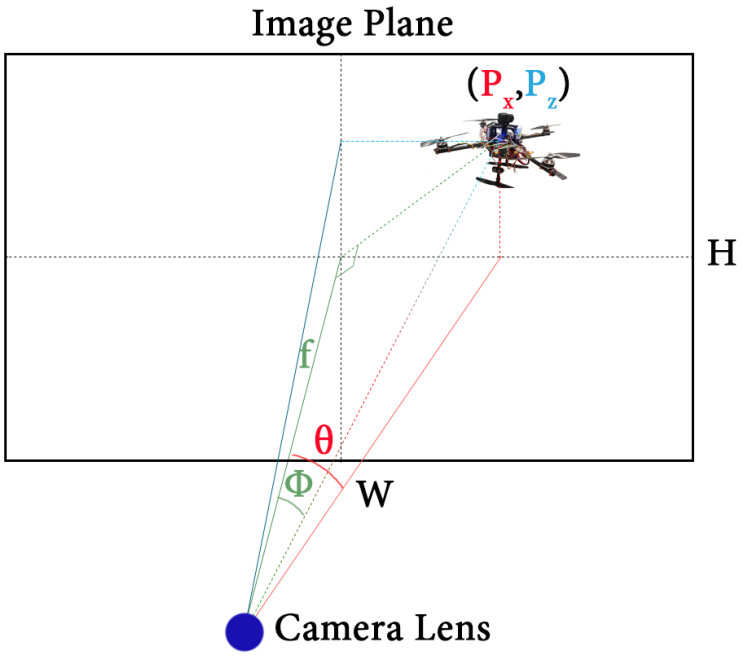


Figure 2.9: *Figure showing how an Object at position X in the Object plane is represented as position, P_X in the Image plane. In Figure 2.5 you see the representation of the same object, but in the object plane looking from above, so that θ in the Image plane equals θ in Figure 2.5.*

approximation can be used as the copter moves further away from the camera and then,

$$y \approx UWB_{range} \tag{2.10}$$

2.1.4 Kalman Filter

The Kalman filter is an optimal recursive filter or algorithm that is often applied on noisy measurements where there is knowledge of the dynamic system. The Kalman filter can be divided in two major steps, there is the Predict stage and the Update stage. These two stages can run at the same frequency, and this is called a single rate filter. They can also run at different frequencies, then it is a multi-rate filter. The multi-rate filter has the advantage that it can run the update stage when we have incoming measurements, and the predict stage can still run at very high frequency giving estimated values between sensor measurements.

As we want to know the position of the copter in higher rates than we get from the Image processing and we want to filter out some noise that is presented

with the measurements we will utilize the multirate Kalman filter. The theory behind the Kalman filter is given below:

First we need to determine our states that we are interested in. For this application we are interested in the X,Y and Z position of the copter as well as their velocities, and therefore we say that our states are,

$$\boldsymbol{x} = [P_x, P_y, P_z, v_x, v_y, v_z]^T . \tag{2.11}$$

The Kalman filter model assumes that the true states is evolved from the previous state according to:

$$\boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{w}_k, \tag{2.12}$$

where $\boldsymbol{A}_k$ is the transition model, $\boldsymbol{B}_k$ is the control-input model and $\boldsymbol{w}_k$ is the process noise. Since we have no control input the simplified model will look like,

$$\boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k. \tag{2.13}$$

Another simplification of our model is that we say the we are always traveling with constant velocity which gives us that the next estimated position is the previous position plus the estimated velocity multiplied by dt which give this transition model,

$$\boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{I_3} & dt \cdot \boldsymbol{I_3} \\ \boldsymbol{0} & \boldsymbol{I_3} \end{bmatrix} \tag{2.14}$$

where $\boldsymbol{I_3}$ equals a 3-by-3 Identity matrix.

Our measurement, $\boldsymbol{z}_k$ which equals the position, can be written as

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k = \begin{bmatrix} P_x & P_y & P_z \end{bmatrix}^T \tag{2.15}$$

where $\boldsymbol{H}_k$ is a matrix that maps the state $\boldsymbol{x}$ into the measurement $\boldsymbol{z}_k$.

The Kalman filter is a recursive estimator, which means that the only thing that is needed to estimate the current state is the estimated state from the previous time step and the current measurement.

From here the estimated states is notated $\hat{\boldsymbol{x}}$ and the true state is notated as previous, $\boldsymbol{x}$.

The Predict stage:

Calculate the predicted state estimation as,

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{A}_k \hat{\boldsymbol{x}}_k. \tag{2.16}$$

Update the predicted estimate covariance with,

$$\boldsymbol{P}_{k+1} = \boldsymbol{A}_k \boldsymbol{P}_k \boldsymbol{A}_k^T + \boldsymbol{Q}_k, \tag{2.17}$$

where $\boldsymbol{Q}_k$ equals the covariance of the process noise.
For keeping the covariance matrix non-singular we need to utilize,

$$\boldsymbol{P}_k = \frac{\boldsymbol{P}_k + \boldsymbol{P}_k^T}{2} \tag{2.18}$$

The Update stage:

Calculate the error between the newest measurement and the predicted states,

$$\hat{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_k. \tag{2.19}$$

Update the residual covariance,

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_k \boldsymbol{H}_k^T + \boldsymbol{R}_k, \tag{2.20}$$

where $\boldsymbol{R}_k$ is the covariance of the observation noise.

Calculate the Optimal Kalman gain,

$$\boldsymbol{K}_k = \boldsymbol{P}_k \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1}. \tag{2.21}$$

Update the estimated states with the calculated Kalman gain,

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k \hat{\boldsymbol{y}}_k \tag{2.22}$$

Update the estimated covariance (Joesph form)

$$\boldsymbol{P}_{k+1} = (I - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k (I - \boldsymbol{K}_k \boldsymbol{H}_k)^T + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T \tag{2.23}$$

The Kalman filter will be implemented to run on the MCU since we need real-time position to autonomously fly the quadrotor.

The Kalman filter will run the estimation stage each time the MCU receives positions from the Image Processing and will run the predict stage to fill up the gaps in between measurements.

By using the Ranging device with computer vision and running the output through a multi-rate Kalman filter we now have positions of the Copter in global coordinates with great refresh rate. But we do not yet know the orientation of the copter. That is why we need an IMU.

2.1.5 IMU

The IMU will be used for keeping track of Quadcopter frame orientation. Since we have no clue of the orientation of the copter from the Image Processing it is very import that the angles from the IMU are accurate, since small angle deviations will make large error at measurements far away.

As we are trying to accomplish to fly inside a mine shaft it is not possible to use a compass to keep reference of the orientation of the copter. Because the IMU measures angular velocity and accelerations it will tend to drift since it will need to integrate to get the angles. Therefore we will use the Mti-3 from XSens, which have on board filtering with Vertical Reference Unit (VRU) and each device is individually calibrated.

In the datasheet[17], it is stated that the IMU with VRU enabled have a drift in unreferenced yaw less than 1 degree after 60 minutes, even in magnetically disturbed environments.

As seen in Figure 2.1 the IMU and the MCU communicates with each other using a UART communication protocol called Xbus. The Xbus protocol is described in the datasheet[17] for the IMU. At the beginning of this thesis it was stated that the communication between the IMU and the MCU should have been working, however, this was unfortunately not the case.

To spend as little time as possible on this problem the communication now only works one way. The MCU can only get data from the IMU if it is setup to be in "data sending mode" as it powers up. To change any settings in the IMU you now need to plug it in the development board and programming it through the MT Software Suite[16] from your computer.

But since we never need to change the settings of the IMU this is not a problem.

2.2 Distance Measurement

How do we measure distances from the copter, and how is this done in 3D?

There are many different sensors that can be used for distance measurement, the two groups of sensors that are mainly used are the optical sensors or the ultrasonic sensors.

For this application where long range is needed (up to 10 m) with good enough accuracy to be able to detect objects the Ultrasonic sensor is of no use since its measurements are too unreliable because of their sensitivity against interference, and their range is often less than what is needed.

2.2.1 LIDAR

In this project a LIDAR was chosen for distance measurment, the LIDAR consists of a Laser that produces optical pulses, these pulses is transmitted, reflected on the surface and returned to the receiver that accurately measures the travel time, so called time-of-flight measurement. This describes how a regular distance measurement is done in one direction, so how to make this work in 2D?

It's actually pretty simple, just rotate the Laser diode and the receiving photo detector and you will have a 2D LIDAR. But an even simpler way is to shoot the laser onto a tilted mirror that reflects the laser in the wanted direction and then just rotate the mirror, see Figure 2.10.

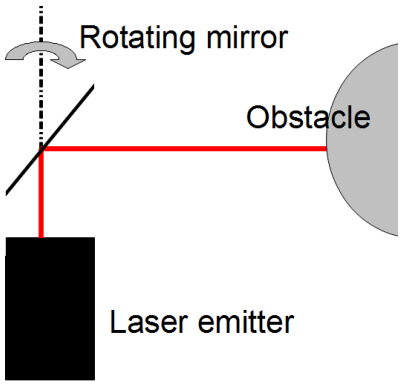


Figure 2.10: *Figure showing the basic principle used by many a 2D LIDAR.*

For this project we are using a 2D scanning Laser Rangefinder (LIDAR) from Hokuyo that is using the same technique mentioned above. The model that we are using is the Hokuyo UST-10LX which have from the datasheet[5] a range of up to 10 m, with a resolution of 0,25°.

The LIDAR rotates at 40 Hz continuously, but cables to the small stepper motor that is mounted on top of the LIDAR that rotates the mirror will make a 90° dead zone. Because of this the LIDAR have a scan angle of 270°, see Figure 2.11.

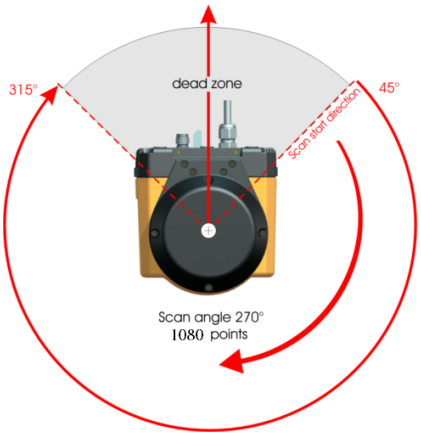


Figure 2.11: *Figure showing the scanning angle and the dead-zone of the LIDAR in default setup[1].*

But still we are only measuring in 2D so we need to somehow rotate it around another axis to be able to get a measurement in 3D.

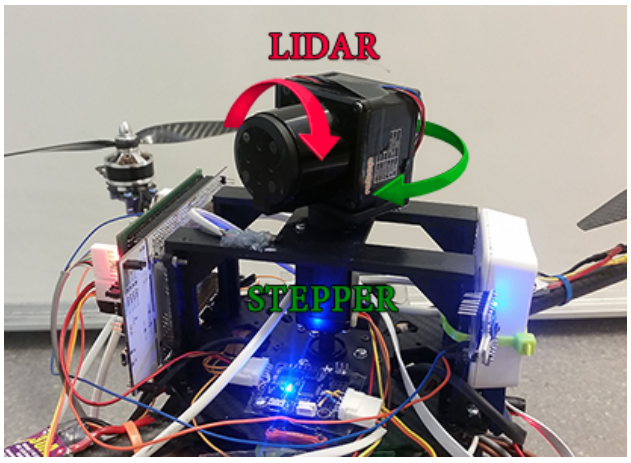


Figure 2.12: *Figure showing the LIDAR and the stepper motor mounted on the quad copter, also showing in which direction the stepper motor and the LIDAR is rotating.*

This is done by mounting the LIDAR vertically on a hollow shaft stepper motor with the scanning dead zone facing the copter frame. The cables that need to connect to the LIDAR are pulled through the hollow shaft of the stepper motor and to be able to rotate the LIDAR multiple turns without

twisting the cables they are mounted to a slip-ring as well. The stepper motor will be rotating with constant speed in one direction, since this will give a better estimation of the stepper motor position than rotating the stepper motor back and forth like a windshield wiper. This setup is shown in Figure 2.12. As this hardware solution already was made at the beginning of this thesis, more can be read about it in the ConexCopter report [7].

With this solution the field of operation for the LIDAR will be almost a complete sphere, which can be seen in Figure 2.13. The Dead Zone of the LIDAR was chosen to be facing down because the copter frame would have been hit by the LIDAR and the measurements would have become useless if the Dead zone had been placed anywhere else.

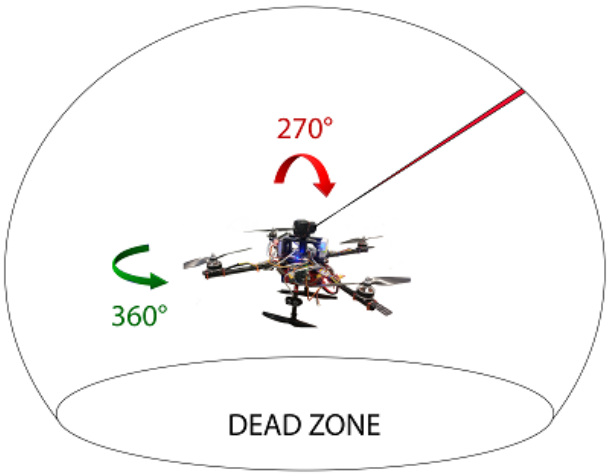


Figure 2.13: *The field of operation for the rotating LIDAR-setup. As the stepper motor has a full* 360° *revolution and the LIDAR has a dead zone of* 45° *we will get almost a complete sphere that the copter can detect. Since the LIDAR is mounted on top of the drone, the dead-zone is pointed towards the body of the copter to get the least amount of bad hits.*

The LIDAR is doing 1081 measurements for each scan and when rotating at 40 Hz it measures over 43 000 distances per second. The LIDAR do have knowledge at which angle each measurement is taken. Which is possible since it knows that the stepper motor is rotating with constant speed and it know each time the mirror passes the 0° mark, where a hall effect sensor is mounted. Data from the LIDAR is sent through Ethernet with a Transmission Control Protocol (TCP), and each scan is marked with a time stamp making it easier when building up the whole 3D picture later on.

The output from the internally placed hall-effect sensor is also available as

one of the outputs from the LIDAR, which will be used for synchronizing and sending of data, which will be discussed later on in 2.4.4.

The same approach that is used with the hall-effect sensor internally will also be used on the stepper motor to be able to know when it has finished one rotation. Since it is critical to know the exact position of the stepper motor for building up the surroundings in the right direction.

2.3 Mapping of coordinate systems

To be able to determine the "global coordinates" of the distances that is measured by the rotating LIDAR, lots of trigonometry and mapping is needed. In this case we have three different coordinate systems. The camera has one coordinate systems which is fixed with the camera, and we do know the cameras global position, let us say that this position is located at $x, y, z = (0, 0, 0)$ [m].

The quad copter does also have its own coordinate system, which is fixed to the copters frame and it is positioned in the copters point of rotation and when dealing with spacecrafts we often use yaw, pitch and roll as names for rotation about the principle axes, this is shown in Figure 2.14 and 2.15.

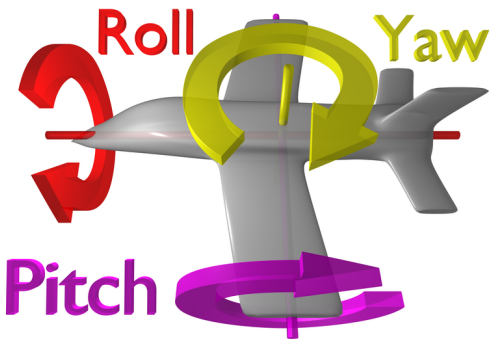


Figure 2.14: *Figure showing the quadcopter attitude, yaw, pitch and roll axes[18].*

The last coordinate system is fixed to the rotating LIDAR. The copter fixed- and LIDAR fixed coordinate system do share the same y-axis since both rotate around the same point in the x-y plane.

So how to convert the LIDAR measurement to a x,y,z point represented in the same coordinate system as the camera?

2.3.1 Rotation and Translation matrices

This is done by utilizing linear algebra with rotation and translation matrices. The three rotation matrices around the three axes, x, y and z is given by:

$$\begin{cases} R(\Phi, x) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\Phi) & -sin(\Phi) \\ 0 & sin(\Phi) & cos(\Phi) \end{bmatrix} \\ R(\Theta, y) &= \begin{bmatrix} cos(\Theta) & 0 & sin(\Theta) \\ 0 & 1 & 0 \\ -sin(\Theta) & 0 & cos(\Theta) \end{bmatrix} \\ R(\Psi, z) &= \begin{bmatrix} cos(\Psi) & -sin(\Psi) & 0 \\ sin(\Psi) & cos(\Psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{cases} \tag{2.24}$$

and the translation matrix is given by:

$$T(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

Since we will use both translation and rotation matrices our rotation matrices will be extended by another column and row to a 4-by-4 matrix. The extended rotation matrix will look like this:

$$\begin{cases} R_{ext}(\Phi, x) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\Phi) & -sin(\Phi) & 0 \\ 0 & sin(\Phi) & cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ R_{ext}(\Theta, y) &= \begin{bmatrix} cos(\Theta) & 0 & sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\Theta) & 0 & cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ R_{ext}(\Psi, z) &= \begin{bmatrix} cos(\Psi) & -sin(\Psi) & 0 & 0 \\ sin(\Psi) & cos(\Psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{cases} \tag{2.26}$$

2.3.2 Mapping

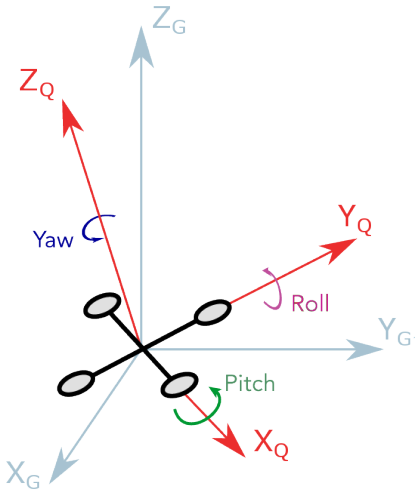


Figure 2.15: *Quadcopter-fixed coordinate system in relation to the camera fixed coordinate system [6].*

The first mapping that we need to do, is the mapping from the camera to the LED that is mounted on the copter frame. This mapping is done by using the translation matrix given in Equation 2.25. The translation matrix from Camera to LED is given by:

$$T_{Cam/Led} = \begin{bmatrix} 1 & 0 & 0 & LED_X \\ 0 & 1 & 0 & LED_Y \\ 0 & 0 & 1 & LED_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

Where $LED_{x,y,z}$ is given by computer vision and a ranging device. The next step is to go from the coordinates of the LED to the copter fixed coordinate system. This is done by applying a rotation matrix, this rotation matrix is a multiplication of all the given rotation matrices in Equation 2.24. The rotation matrix from camera- to quadcopter coordinate system is given by:

$$\boldsymbol{R}_{Cam/Quad} = \begin{bmatrix} C_\Psi C_\Theta & -S_\Psi C_\Theta + C_\Psi S_\Theta S_\Phi & S_\Psi S_\Phi + C_\Psi S_\Theta C_\Phi & 0 \\ S_\Psi C_\Theta & C_\Psi C_\Phi + S_\Psi S_\Theta S_\Phi & -C_\Psi S_\Phi + S_\Psi S_\Theta C_\Phi & 0 \\ -S_\Theta & C_\Theta S_\Phi & C_\Theta C_\Phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.28}$$

with representation of $C_{angle} = cos(angle)$ and $S_{angle} = sin(angle)$ and where $[\Psi,\ \Phi,\ \Theta] = [Yaw,\ Pitch,\ Roll]$

Since the angles of the copter should be smaller than 10° we can use small angle approximation to speed up computations. Substituting $sin(x) = x$ and $cos(x) = (1 - x^2/2)$ in Equation 2.28 we get the approximated rotation matrix,

$$\hat{\boldsymbol{R}}_{Cam/Quad} \approx \boldsymbol{R}_{Cam/Quad} \tag{2.29}$$

Mapping from the center of the copter frame to the rotation point of the LIDAR is done by utilizing another translation matrix:

$$\boldsymbol{T}_{Led/Lidar} = \begin{bmatrix} 1 & 0 & 0 & L_x^o \\ 0 & 1 & 0 & L_y^o \\ 0 & 0 & 1 & L_z^o \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

with L_i^o = LIDAR offset x,y,z coordinates.

The LIDAR is mounted on top of a stepper motor, and the motor is mounted in the center of the copter frame in the x,y plane with a offset in the z plane. Thereby the rotation matrix for the mapping between the LIDAR fixed coordinate system and the copter fixed system is only a rotation around the z-axis, and this is given by:

$$\boldsymbol{R}_{Quad/Stepper} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

where $\alpha$is the angle of the stepper motor. The LASER diode that is measuring the distances is placed with an offset from the origin of the LIDAR fixed coordinate system, and thereby another translation matrix is needed.

$$\boldsymbol{T}_{Lidar/Laser} = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

where $\boldsymbol{d}$ = distance between LIDAR center of rotation to the LASER diode. The last transformation matrix that is needed is a matrix that describes the

measured distances.

$$\boldsymbol{T}_{Laser/Point} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_d sin(\beta) \\ 0 & 0 & 1 & L_d cos(\beta) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

This is a translation matrix where L_d is the measured distance from the LIDAR, and β is the angle of the stepper motor mounted inside the LIDAR.

Multiplying all the matrices together will give the complete mapping from the measured laser distance to the fixed camera global coordinate system, W_p = Wall point

$$\boldsymbol{W}_p = \boldsymbol{T}_{Ca/Le}\hat{\boldsymbol{R}}_{Ca/Qu}\boldsymbol{T}_{Le/Li}\boldsymbol{R}_{Qu/St}\boldsymbol{T}_{Li/La}\boldsymbol{T}_{La/P}\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^T \tag{2.34}$$

2.4 Software

To be able to build a 3D map of the surroundings from all the data that is retrieved from the different sensors such as the LIDAR, the IMU, the stepper motor and the position from the Kalman filter everything needs to be synchronized. Since the MCU has its own `SysTick` timer, updating at 1 ms it is easy to add a timestamp to every incoming measurement. But since the LIDAR does not go through the MCU there is almost impossible to know which LIDAR scan that belongs to which position and angle of the copter.

This problem is solved by synchronizing the MCUs timestamp with the timestamp of the LIDAR. Lots of code were written before beginning of this thesis, this includes almost all drivers for the different peripherals and the most basic communication protocols where defined. Everything about this can be read about in the in the ConexCopter report [7].

Since this is a time-critical system the software is interrupt based and a high-level representation of the software that is used for this system is seen in Figure 2.16.

2.4.1 Main function

The first thing that is done when booting the MCU is that it is initializing the drivers for its peripherals and setting all the GPIOS to the right pins.

After all the drivers are set up it will wait until it has established a connection to the LIDAR. If no connection is established, the MCU will just sit and wait, this loop can be aborted by pressing on a button on the MCU. When there is a stable connection to the LIDAR the MCU asks for its time stamp and synchronizes their stamps with each other. Then it will disconnect so that the base station can connect to the LIDAR, since only one device can be connected at the same time.

As everything is setup, all the interrupts can be started. At the end of the `main` function there is a infinite `while` loop that is used for sending of the LIDAR and Accelerometer Measurement Protocol (LAMP) packets, what these packets are is shown in Section 2.4.3.

The infinite loop checks which sending flag that is set and then sends the corresponding packet over Ethernet. Which packets that can be sent can be defined before compiling the code, since sometimes you only want a specific packet to be sent. This is also shown in Figure 2.16 where `#define` shows

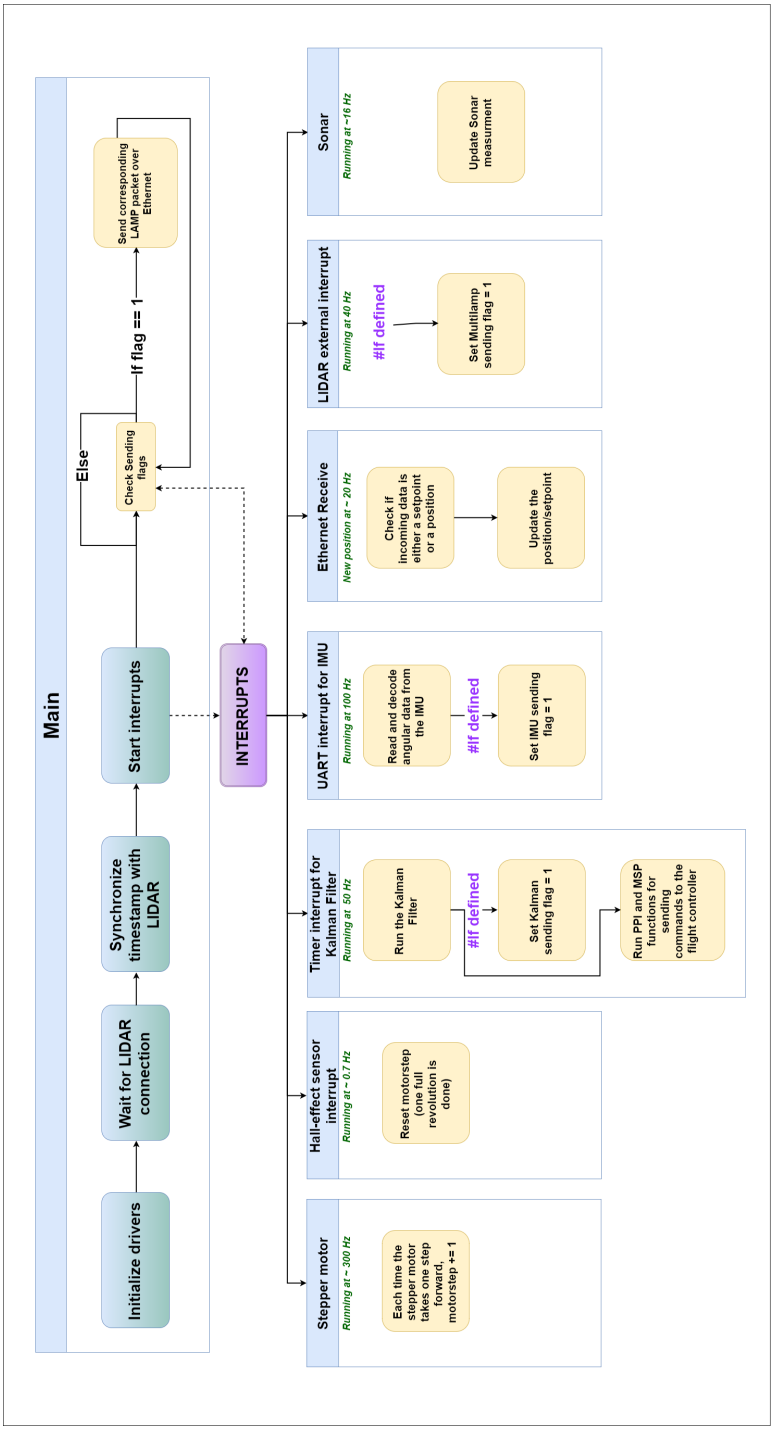


Figure 2.16: *Figure showing a high level schematic view over the interrupt driven software that is running on the MCU.*

which sending flag that can toggled on or off.

2.4.2 Interrupts

This system do consist of many different peripherals and communicates with a lot of sensors and other devices such as the flight controller and the base station. Since the MCU needs to gather lots of information and that it needs to always get the latest available data, all devices that are talking with the MCU needs to be interrupt driven, since polling of data isn't an alternative. The main interrupts that is running on the MCU is shown in Figure 2.16 and will be explained further below.

Stepper motor

To take a step with the stepper motor, a short pulse is sent to the stepper motor driver. Each time that this pulse is sent, an interrupt on the MCU will occur and a counter will count up the motor step, like `motorstep = motorstep + 1`. The stepper motor driver is programmed in such a way that 400 steps is one full revolution.

Hall-effect sensor

As the stepper motor only counts the steps that it has taken and not at which angle the motor started. A hall-effect sensor and a magnet will be used to make it possible to detect each turn the motor has taken. Each time the magnet which is mounted on the motor passes the sensor it will make an interrupt. This interrupt handler does only reset the motor step with, `motorstep = 0`. Thus the position of the stepper motor is known, for simplicity we can say that the stepper motor is rotating with constant speed under one revolution.

Kalman Filter

The Kalman Filter mentioned in Section 2.1.4 has been ported into C-code and is being executed by the MCU. The Kalman Filter will be running at 50 Hz since this is the maximum frequency we can run the MultiWii Serial Protocol (MSP) commands and be talking to the flight controller. Running the Kalman filter at a higher frequency than the maximum frequency of the MSP commands will be waste of computational power, as the MSP commands depends on the positional controller [14] since sending the same command twice is completely unnecessary.

To run the filter in the desired frequency a timer will be used, the timer will

interrupt the main loop at 50 Hz and this interrupt handler will first run the Kalman Filter, then set the Kalman Filter sending flag to one, then it will run the positional controller and lastly it will send the MSP command to the flight controller. More about this can be read about in the Master thesis by Max Unander [14].

IMU

The IMU will interrupt the MCU each time it has new incoming measurement, this will run in 100 Hz since that is the update rate of the IMU. In this interrupt handler also the parsing of the data from the IMU will be made. The communication between the IMU and the MCU is a UART communication protocol and to handle this large amount of data without occupying the CPU the MCU utilizes Direct Memory Access (DMA).

Ethernet

Each times when there is incoming data on the Ethernet line to the MCU an interrupt will occur. This interrupt handler will check whether the incoming data is a position setpoint or whether it is a position sent from the computer vision.

If it is a position it will update the old measurement and notify the Kalman filter that there is a new measurement and then the next time the Kalman filter runs it will do the update stage instead of the predict stage, see Section 2.1.4.

If it is a setpoint it will only update the setpoint for the positional controller, this is discussed further in the the Master Thesis by Unander [14].

LIDAR

As mentioned in Section 2.2.1 the LIDAR has an internal hall sensor that keeps track of its rotation. This is also available as an output of the LIDAR where the signal on the output pin goes low each time the internal part of the LIDAR passes the sensor. This output is connected to the MCU and is set up to trigger an interrupt each time the pin goes low. This interrupt handler will only set the `multilamp send flag` to one. As this package consists of a timestamp, stepper motor angle, IMU angles and copter position this is the only package, plus the LIDAR scans of course that is needed to be able to construct a 3D image.

Sonar

The Sonars mounted on the copter uses ultrasonic sound to measure the distance from the copter to objects, that could for example be a wall. The Sonars will be used for collision avoidance, so that the copter will not collide with anything. All sonars will trigger an interrupt after each measurement, in each interrupt handler the measured distance of the sonar will be calculated. More about how this is implemented can be read about in the *ConexCopter report*, K.Alkawati et al. [7].

2.4.3 Data sending protocol

In order to encode sent and received data, a protocol was constructed. The protocol got the name LAMP, LIDAR and Accelerometer Measurement Protocol. The protocol needs to pass along data from the LIDAR, the IMU and data from the Kalman filter. All LAMP packages are passed along with a time stamp and are currently sent using the Wi-Fi module that is mounted on the copter.

The major objective of the LAMP protocol was to send LIDAR frames with certain resolution and with stop and start angles for the given frame as well as angle of the stepper motor, so it would be easy to piece together all the data.

For now the LAMP protocol only handles the IMU, angle of stepper motor and Kalman filter data. LAMP is now defined as follows:

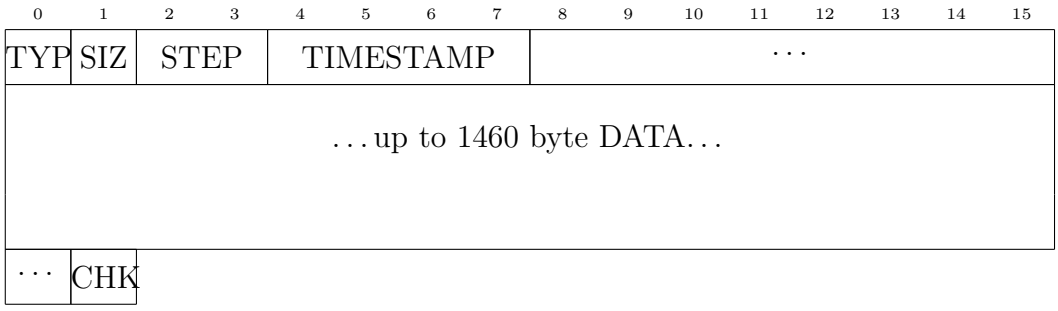


Figure 2.17: *LAMP protocol definition - LIDAR frames*

Field definitions:

TYP: Contains type of the current frame.

`0x00`: IMU package

`0x01`: Kalman filter package

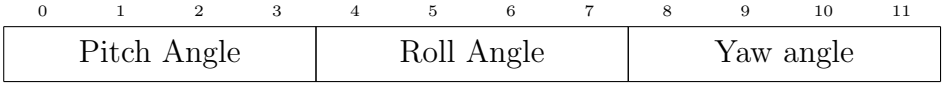
`0x02`: Data (any available data up to 1460 bytes)

`0x03`: Multilamp (IMU, Kalman position, motor step)

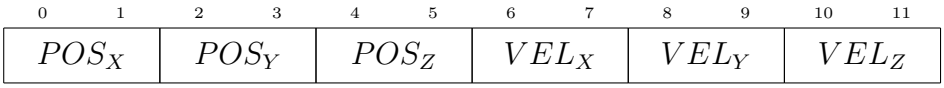
SIZ: Total size of the whole frame, in bytes.

STEP: Current step position of the stepper motor, in total steps from zero point. Should be set 0 for IMU and Kalman frames.

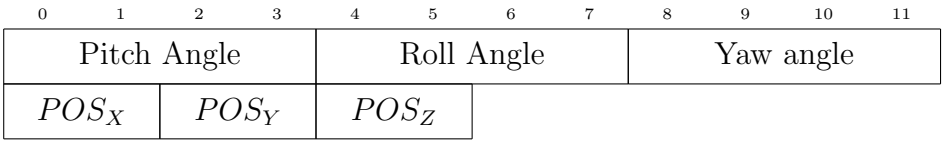
DATA: `IMU`: Current IMU angles, rotated around the copter fixed coordinate system. Each angle represented by an `int32_t`.



`KALMAN`: Estimated Kalman filter data, $POS_{x,y,z}$ and $VEL_{x,y,z}$ all represented by an `int16_t`.



`MULTILAMP`: All necessary data to be able to construct a correct 3D image from LIDAR scans.



CHK: Checksum of the package, calculated as an XOR of every byte of the frame.

2.4.4 Data acquisition

Now that everything is set up and done on the MCU end, all data that is needed is sent through LAMP packages and the LIDAR can be easily connected to over Ethernet. But how is this data handled in the other end, on the base station?

On the PC that is used as a base station, a simple Python script is written that decodes the incoming lamp packets and saves them into a Comma-Separated Values (CSV) text-file that can easily be read by Matlab[8] to be able to construct the 3D image later on.

This Python script is combined with the Python script that runs the Com-

puter Vision algorithm so that only one script for both sending position to the copter and retrieving data from the copter is needed.

Retrieving and decoding data from the LIDAR is also done using a Python script. Since writing a script that can decode the encoded measurement data sent by the LIDAR in a reliable way was a little bit tricky a Python library named HokuyoLX[10] was found and used.

This library was easy to interface with and just writing a script for saving all the data into a CSV-file was needed.

2.5 3D model

The 3D model was built using Matlab [8] using the collected data and the mapping of coordinate system is shown in Equation 2.34. The whole Matlab script can be found in Appendix D.

In more detail the first thing that this script does is that it loads the LIDAR and `Multilamp` data from their respective CSV-file. As both theses data sets have their own timestamp (synced with each other though) they need to be matched with each other, to prevent misalignment if some data is lost.

As the only thing that is needed for the mapping is the position of the copter, the angle of the copter and the angle of the stepper motor for each LIDAR scan. Since the `Multilamp`-packet is sent with the same frequency as the LIDAR-frames we can for simplicity say that the copter angles and the copter position is constant over one scan.

The timestamp syncing, loops through and checks which LIDAR timestamp that belongs to which timestamp from the `Multilamp`-packet. If no match is found that LIDAR frame is thrown away. With this syncing all data can now be stored with the same timestamp.

The approximation that the stepper motor is standing still under one scan can not be made since for one LIDAR scan the stepper motor will take a few steps and this will induce an angular error for the mapping. The number of steps the stepper motor will take under one LIDAR scan can be calculated by,

$$steps/scan = steps \cdot \frac{f_{stepper}}{f_{LIDAR}}, \tag{2.35}$$

where steps is the amount of steps that the stepper motor need to take to make a full revolution, with $f_{stepper}$ and f_{LIDAR} as the frequency of the stepper motor and the LIDAR.

As the angle of the stepper motor is known when the LIDAR is at its starting position from the LAMP protocol and also knowing how many steps the stepper motor takes for one LIDAR scan. By saying that the stepper motor is rotating with constant speed it is simple to use the Matlab's `linspace` function to make a linearly spaced vector with equally number of elements as the scan has.

A vector containing the angles of the LIDAR is also needed and as one scan goes from an angle of 45° to an angle of 315° linearly spaced with a resolution

of 0,25° (see Figure 2.11), the `linspace` function will also be used here.

Now that all the data is set up and synced it is time to use the translation and rotation matrices given in Section 2.3. To get the global coordinate of each point measured by the LIDAR, each point will be multiplied with the matrices given in equation 2.34.

This will be computational heavy since each lidar scan consists of 1080 points and the scans will be running at 40 Hz, thus will generate 43200 points per scanned second.

The output from this equation is a vector containing x,y and z values representing the global coordinates of the scanned surrounding. These vectors are concatenated on top of each other, making a large matrix containing all the x,y and z positions. This matrix is then made into a Point cloud using Matlab's Computer Vision System Toolbox[9]. With this toolbox the Point Cloud can be plotted easily, and with greater performance than making the 3D plot with Matlab's 3D figure tool.

A point cloud software for LIDAR data named VRMesh Studio[15] was tested, and is shown in Figure 3.4.

CHAPTER 3

Results

3.1 Copter Positioning

The results from the Copter Positioning which includes the results from the Camera Pixel conversion as well as the results established from the Computer Vision and the Kalman filter will be shown, and later on discussed in Chapter 4.

3.1.1 Camera pixel conversion

Calculating the size of one pixel in relation to the distance from the camera was done using the theory shown in Section 2.1.3 (Camera Pixel to distance calculation). The "imaginary" pixel focal length, f is calculated by,

$$f = \frac{D}{2} cot\left(\frac{D_{FOV}}{2}\right), \qquad \text{(2.4 revisited)}$$

with $D_{FOV} = 78°$ from the camera specification[2] and

$$D = \sqrt{H^2 + W^2}, \qquad \text{(2.5 revisited)}$$

with $H = 1920$ and $W = 2560$ which is the camera's maximum resolution.

The width of one pixel can thus be calculated by

$$Pixel_{width} = \frac{Y}{f}, \qquad \text{(2.7 revisited)}$$

where the length from the camera to an object, Y can be varied.

A graph over how the pixel width relates to the distance can be seen in Figure 3.1. In the same graph, the measurement of the pixel size from the experiment with the fixed size paper, also explained in Section 2.1.3 is plotted as stars.

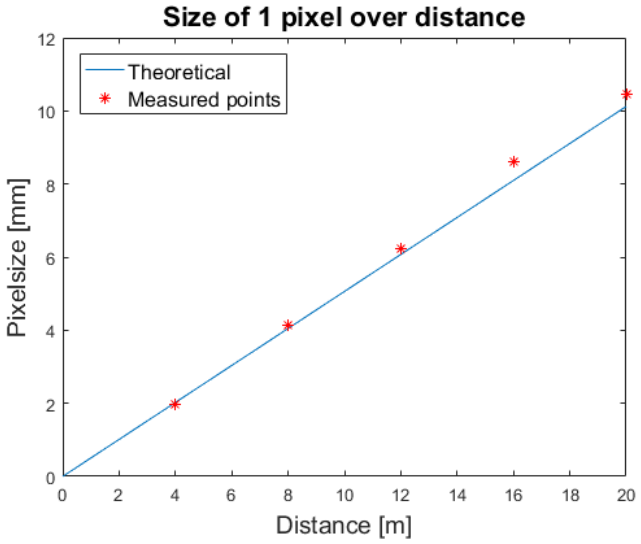


Figure 3.1: *Plot showing how pixel width relates to distance from the camera. Theoretical calculations plotted as a blue line, and measurements plotted as red stars. All measurements can calculations is made using the cameras highest resolution,* `2560x1920`*.*

From the graph the $Pixel_{size} \approx 0{,}51\,\text{mm}/m$ at the resolution of `2560x1920`, but can be scaled linearly between different resolutions. The Computer Vision algorithm will run with the lowest camera resolution (`640x480`) to get the maximum frame rate (will be discussed later). Calculating the scale of $Pixel_{size}$ into the wanted resolution is made by,

$$S = \sqrt{\frac{2560x1920}{Resolution}}. \tag{3.1}$$

With $Resolution =$ `640x480` it gives $S = 4$, and thus $Pixel_{size}$ needs to be divided by 4 to scale it to the wanted resolution.

3.1.2 Computer Vision

A camera can be modeled to have about one to two pixels of noise, and from the project specification it is said that the Computer Vision should work at a distance up to 200 m. With the highest resolution one, pixel at 200 m will have a size of 10 cm, and thus a noise of 20 cm is expected.

Lowering the resolution of the camera will increase the noise, but will give a better refresh rate. This is a trade-off that will be discussed further in Chapter 4.

Both the Algorithms 1 and 2 were tested at different resolutions, giving the following table:

Table 3.1: *Frame rate with respect to resolution for two different Computer Vision algorithms as well as the cameras maximum update rate.*

Resolution	**Algorithm 1**	**Algorithm 2**	**Camera MAX**
$640x480$	$\approx$ 23 FPS	$\approx$ 70 FPS	60 FPS
$1280x720$	$\approx$ 8 FPS	$\approx$ 50 FPS	30 FPS
$2560x1920$	$\approx$ 1 FPS	$\approx$ 12 FPS	15 FPS

Running Algorithm 2 at higher frequency than the camera can manage is of course just a waste of processing power, but this shows the maximum Frames Per Second (FPS) the algorithms can run.

As most of the tests are made at short distances, (up to 40 m) it really doesn't matter that much which algorithm that is used, as long as the FPS is greater than 10.

Algorithm 2 with the `1280x720` resolution is the one that will be used though, but the frequency of the algorithm will be set to the camera's maximum supported FPS.

With this choice of algorithm it will give a fast refresh rate of 30 Hz and an expected noise of less than 10 cm at 40 m.

3.1.3 Kalman Filter

A plot over the estimated position by the Kalman filter and the measured position by the Computer Vision can be seen in Figure 3.2.

As the Kalman Filter is running at 50 Hz and the Computer Vision is running at about 30 Hz you can not see that much of a difference between these two. But the Kalman Filter do filter out noise, and if some data sent by the Computer Vision is lost, the Kalman filter will continue to update the position at 50 Hz, which is needed for the positional controller and the MSP protocol.

By zooming in on the middle subplot in figure 3.2 (Y position) the noise filtering established by the Kalman filter can easier be seen, this is shown in Figure 3.3.

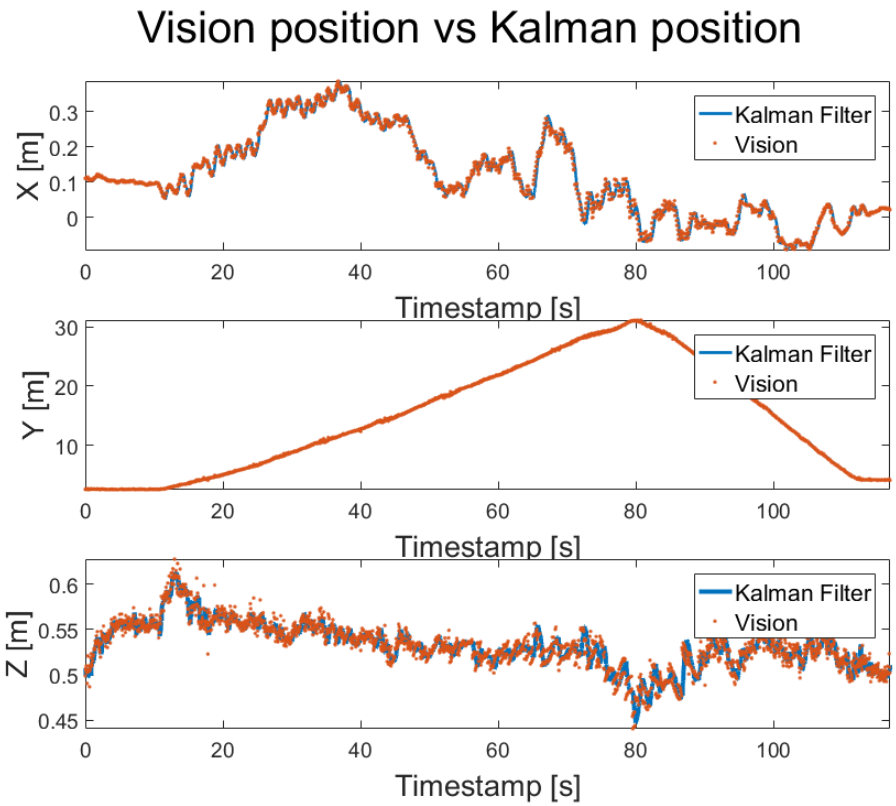


Figure 3.2: *Plot showing the position of the copter in X, Y and Z positions. The blue line shows the Kalman filtered positions and the orange dots represents the unfiltered positions from the Computer Vision.*

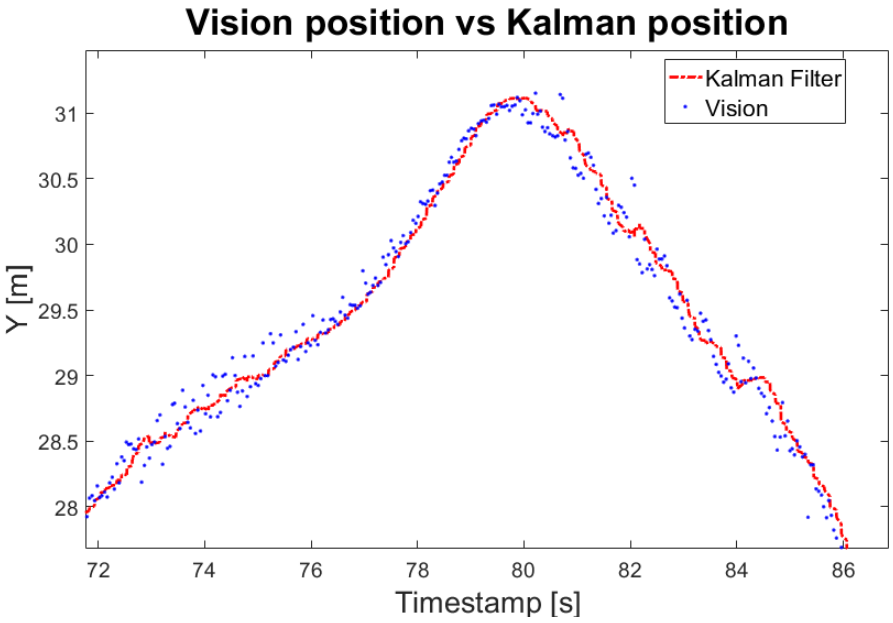


Figure 3.3: *Plot showing the Kalman filtered position and the raw unfiltered position from the Computer Vision. This is the same data that is shown in the middle subplot in Figure 3.2 but over a shorter time span, to easier see the difference between the filtered and unfiltered data. The Kalman filtered data is marked with a red line and the unfiltered Computer Vision data is marked with blue dots.*

3.2 3D point cloud

The initial test scan of the 3D LIDAR setup gave the following point cloud illustrated in two separate plots of the same point cloud from different viewing angles in Figure 3.4. As this test where only for testing the mapping from the stepper motor to the measurement of the walls both the positions and IMU angles were set to zero.

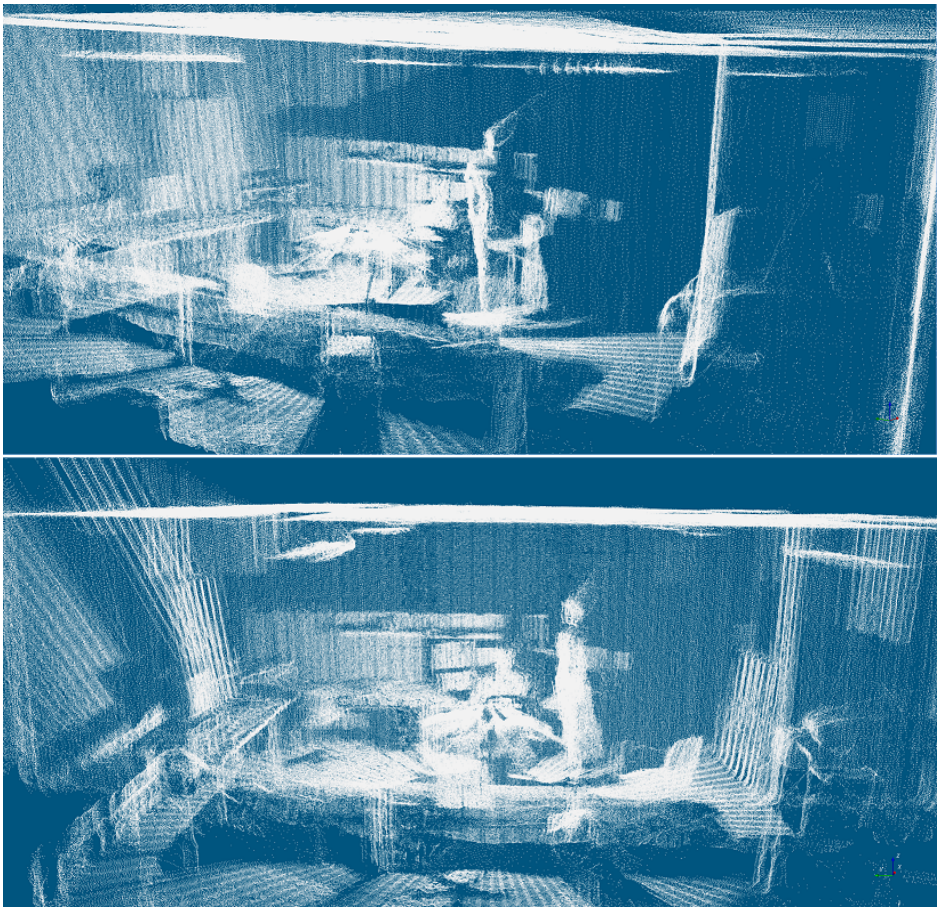


Figure 3.4: *Plot of our first test scan, LIDAR mounted stationary at middle of the project room. Each subplot is from the same data-set but shown from different viewing angles. The software VRMesh[15] was used for plotting of the point cloud data.*

As said in Section 1.2 a corridor near the project room will be used for testing during this thesis. A scan of this corridor with the LIDAR mounted on the flying quadcopter, when also adding both the position from the Kalman filter and the angle of the copter from the IMU is shown in Figure 3.5 to 3.7.

Figure 3.5 shows the corridor from above (X-Y plane) where Y is defined

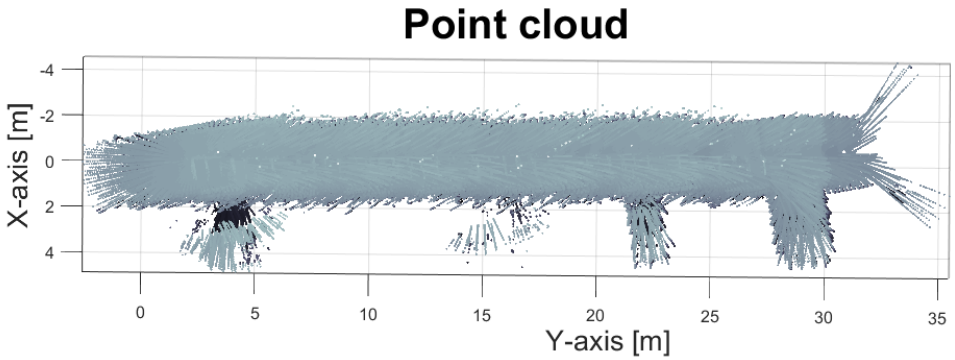


Figure 3.5: 30 m *test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the corridor from above ($X-Y$ plane). Plotted in Matlab using the Vision-toolbox[9].*

along the corridor and X is the width. As seen in this Figure there are measurements that are not along the walls of the corridor and this is because there are two glass doors along this corridor which the LIDAR can penetrate ($Y \approx 5$ and $Y \approx 15$), and there are also two other crossing corridors ($Y \approx 22$ and $Y \approx 30$).

This can easy be seen in Figure 3.6, which is the same point cloud but shows only the right part (positive X) of the corridor in the $Z - Y$ plane.

Figure 3.7 represent the corridor standing where the camera is mounted. On the right hand side the LIDAR measurements through the doors and the crossing corridors can be seen, and as the copter is not standing still while scanning these holes, the amounts of scans in these areas will not be that many.

All positions in these plots is relative the camera, so a position of $(0, 0, -1)$ is positioned 1 m below the camera, and not 1 m below the floor since the camera is mounted on a table in the middle of the corridor.

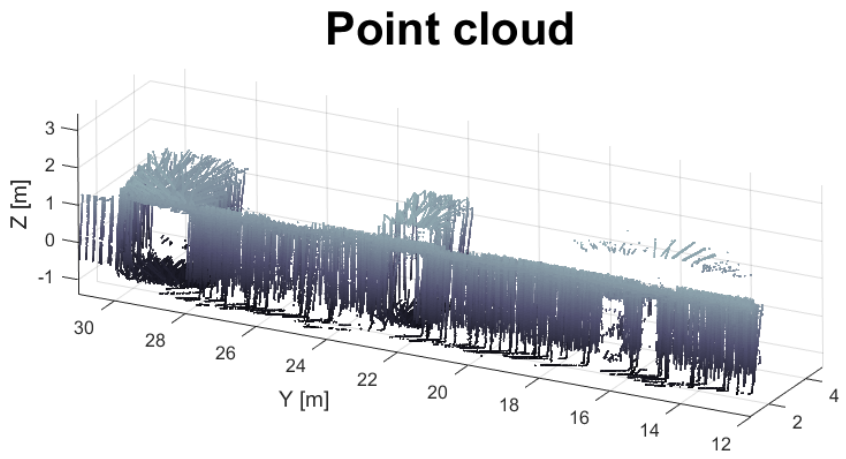


Figure 3.6: 30 m *test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the right wall from the inside of the corridor. Plotted in Matlab using the Vision-toolbox[9].*

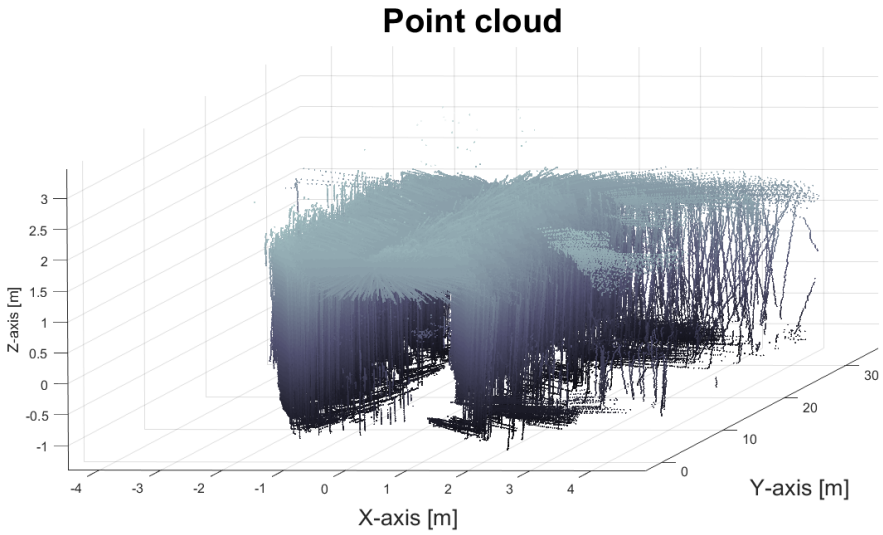


Figure 3.7: 30 m *test scan of a corridor which is about the same size as the proposed mine shaft. This plot shows the front entrance of the corridor, the same place as the camera is located. Plotted in Matlab using the Vision-toolbox[9].*

Chapter 4

Discussion

In this chapter, an evaluation of the results from the copter positioning and the point cloud will be discussed. Conclusions are drawn from those evaluations and some possible future work will be presented.

4.1 Copter Positioning

4.1.1 Camera pixel conversion

In Figure 3.1 it is seen that the theoretical calculations with approximations is sufficiently correct since the measured points lies within a small margin of the theoretical function. The measurement of the pixel size were made only to verify the correctness of the theory.

4.1.2 Computer Vision

At the start of this thesis I had no experience in Computer Vision, and at an early stage of this thesis lots of different tracking algorithms were tested. Most of these algorithm that was tested, like the one used by face recognition and color detection did not work that well at distances over 3 m.

The first test of the blob detection algorithm, which is the one that is used in the algorithms described in Section 2.1.2 was not that successful. It did not just detect the LED mounted on the copter, but all roof lights as well. That is why the background subtraction and the binarizing of the difference image was added, to get rid of light sources that are not wanted.

The first algorithm (Algorithm 1) that was coded does work well within the tested range up to 40 m, which is the length of the whole corridor. By using the lowest resolution the frame rate will be over 20 FPS and the expected noise will be less than 8 cm, which is okay for the goals of this thesis.

But as the goals of the project is to get the Vision working well up to a range of 200 m, the expected noise with this setup will be 40 cm to 80 cm,

and this is of course way too much. By doubling the resolution of the camera, the noise could be reduced by 2, and thus the expected noise would be less than 20 cm to 40 cm. But increasing the resolution will decrease the FPS as shown in Table 3.1, and an FPS greater than 15 is wanted since otherwise the Kalman filter will not perform as good.

Therefore a modified algorithm was needed, and thus Algorithm 2 was coded. With this algorithm it is possible to run at a higher resolution while keeping the FPS sufficiently high, since the time consuming blob detection algorithm only need to iterate over a small portion of the image, instead for over the whole image.

4.1.3 Kalman Filter

The Kalman filter that was used is based on a simple approximation that the copter is moving with constant speed between measurements. As the incoming position measurements have high refresh rate, this approximation is good enough. But if the incoming position is updated too seldom, this approximation will cause the Kalman filter's prediction between the measurements to be unreliable. A solution to this could have been to fuse the IMU which has a high update rate, but tends to drift with the absolute position from the Computer Vision which has a lower update rate, but no drift. How this fusion can done can be read about in the *Quaternion kinematics for the error-state KF* [see 13, chap 3]. As this implementation should alone have been enough material for one Master Thesis this is something that was directly discarded to fit within this project. One such Master Thesis where sensor fusion and a Kalman filter for a drone utilizing an IMU and a GPS for positioning is done can be read about in *UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals*, [4].

4.2 3D point cloud

Figure 3.4 shows a scanning with the LIDAR kept stationary and Figure 3.7 shows a scan where the lidar was mounted on the multirotor that was carried through the corridor. If you compare Figure 3.4 with Figure 3.7 you can see that the point cloud from the stationary scan has less noise than the moving point cloud. Thus we can say that most of the noise that is generated in the point clouds comes from the measured position and angles of the copter. To reduce the noise in the point clouds some filters could have been used, but this will not be handled in this thesis.

In Figure 3.6 it is seen that is is possible to detect holes that are about 30 cm

wide, as the side window to the door located at about 14 m is 30 cm wide.

4.3 Future Work

If this project does not end here, there are some things that is needed to be done.

- Computer Vision: Test and evaluation of the algorithm at longer ranges than 40 m. If this algorithm does not work that well at long distances, a new algorithm is needed. Maybe some GPU based algorithm could have been used to get better FPS at higher resolutions.
- Point cloud: As discussed above, some filtering and denoising of the point cloud is needed. Maybe some feature detection could be of use, to be able to localize the copter even if there is some drift in the IMU.
- 3D Scanner: A better construction of the 3D scanner, where the rotation of the stepper motor is mounted to the center of mass of the LIDAR (now it has a little offset).
- Software: All data should be parsed within the MCU and sent with LAMP-packets as proposed in the Conex Copter report[7]. As for now the LIDAR data is sent directly to the base station through Wi-Fi. With parsing of the LIDAR data in the MCU all synchronizing of timestamps will be done there. Each LAMP packet should hold the starting angle of the LIDAR, the angle of the stepper motor, the amount of LIDAR data and of course the LIDAR data itself. With this protocol working, there will not be any hassle with timestamps that drifts nor synchronization problems.

4.4 Conclusions

As said in the Introduction in Section 1.2 *"The primary goals of this thesis is to produce a 3D point cloud of the copters surroundings, by utilizing a LIDAR and an IMU, and to visualize this point cloud in a well suited way to be able to detect larger defects on the surrounding walls. But to do this the position of the copter is needed, and this is made using a ranging device and Computer Vision."* Based on the measurements from the Kalman filter which can be seen in Figure 3.2 and 3.3 a position of the copter can be established up to at least 32 m, with an appreciated accuracy of about 10 cm at this distance. From the point clouds shown in Figure 3.6 it is seen that it is possible to detect holes in the walls that are about 30 cm wide.

Since positioning of the copter is established up to at least 30 m and it is possible to detect holes in the corridor walls from the point clouds that is at least 30 cm wide, I will say that the goals for this master thesis have been reached.

Appendices

Appendix A

Parts list

- STM32F767, mbed-Enabled Development Nucleo STM32F7 MCU 32-Bit ARM Cortex-M7 Embedded Evaluation Board.
- Hokuyo UST-10LX Scanning Laser Rangefinder.
- MTI-3-8A7G6T, Accelerometer, Gyroscope, Magnetometer, 3 Axis Sensor.
- Logitech B910 HD Webcam.
- ALFA AIP-W512 wireless router.
- FLIP32 F3 AIO-Lite Flight Controller.
- Turnigy TGY-i10 10ch 2.4GHz Digital Proportional RC System with Telemetry.
- Afro 30A Muti-Rotor ESC (SimonK Firmware).
- Dualsky ECO 3508x (580 Kv) BLDC motor.
- Zippy 8000 mA 30C 4S Li-poly battery.
- Ultrasonic sensor, HC-06 and LV-MaxSonar-EZ1.

APPENDIX B

Nucleo Shield

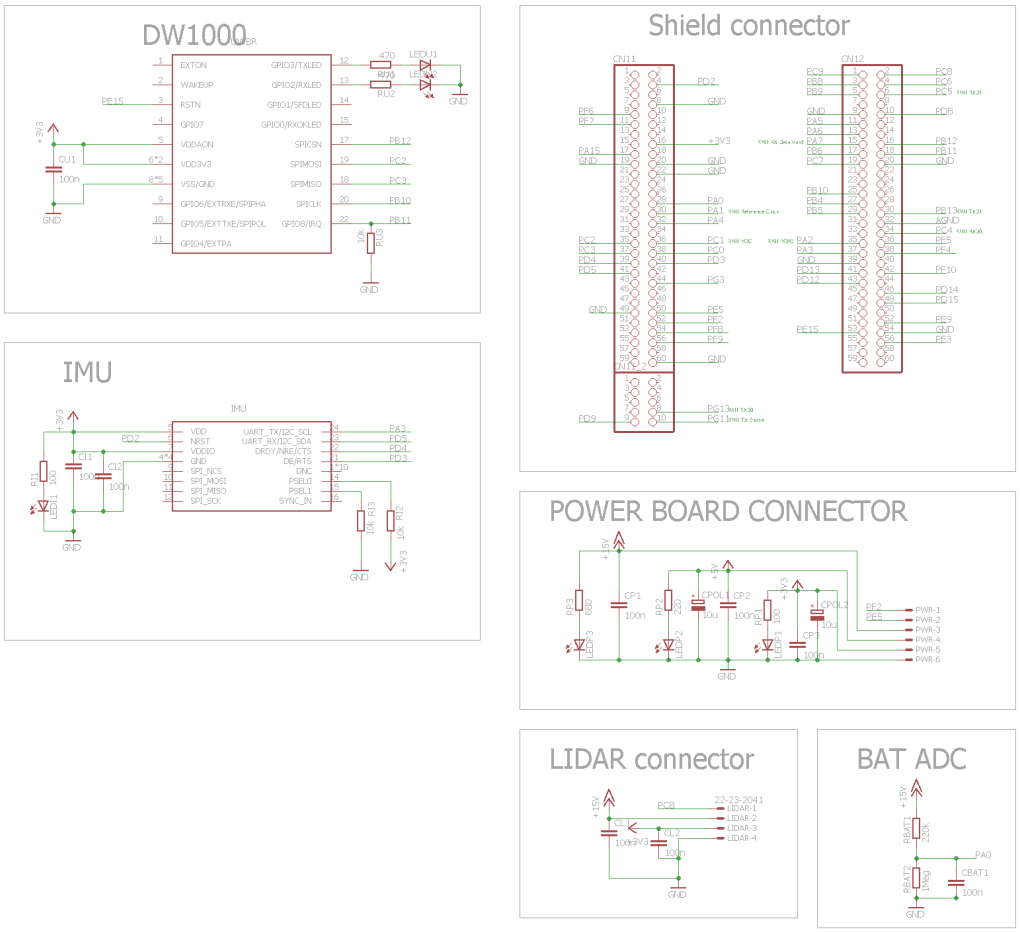


Figure B.1: *Part of the schematic of the Nucleo shield, showing the DW1000 UWB module (not in use), the shield connector to the development board, the IMU socket, the power board connector, LIDAR connector and battery to ADC connection*

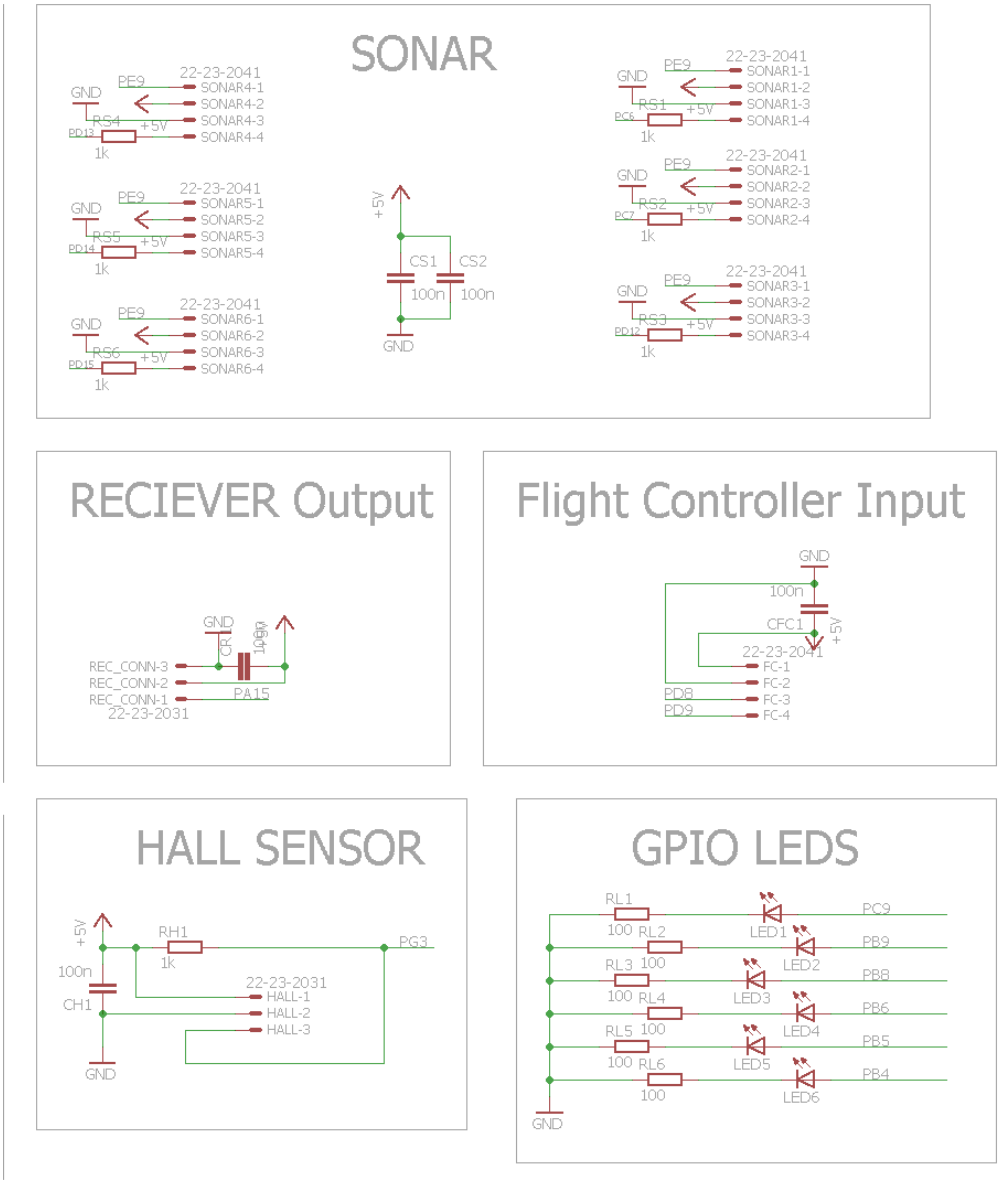


Figure B.2: *The remaining part of the schematic of the Nucleo shield. Showing of connectors to the receiver, the flight controller, sonar, hall sensor and some GPIO LED.*

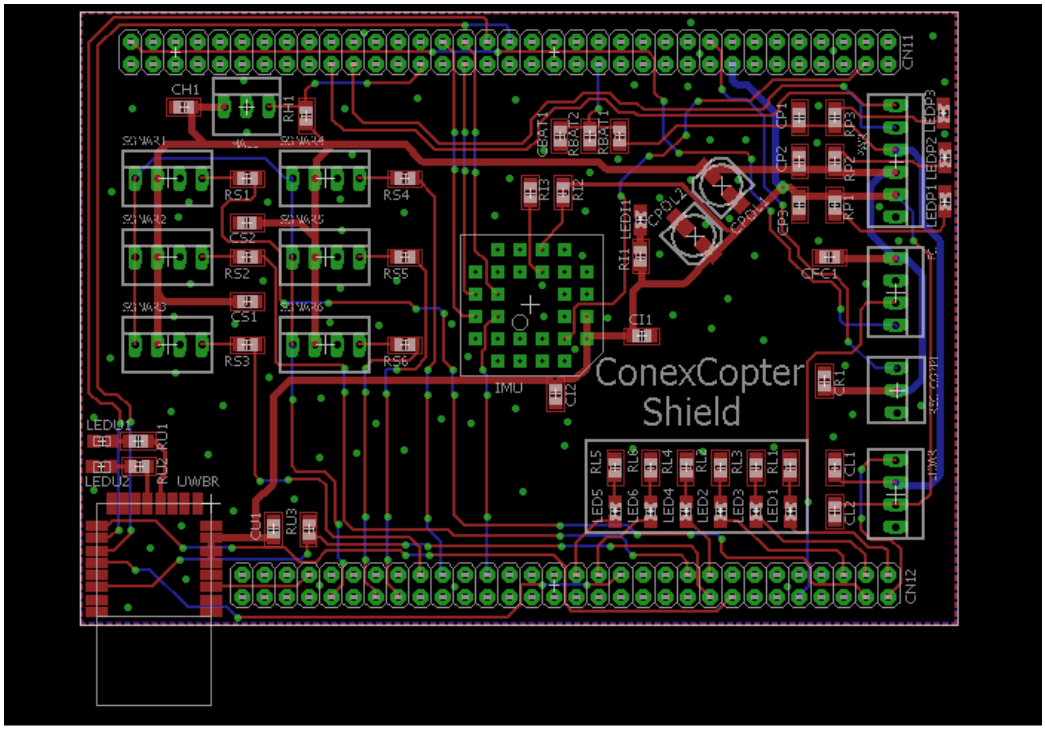


Figure B.3: *Board layout of the nucleo shield.*

APPENDIX C

Power Distrubution board

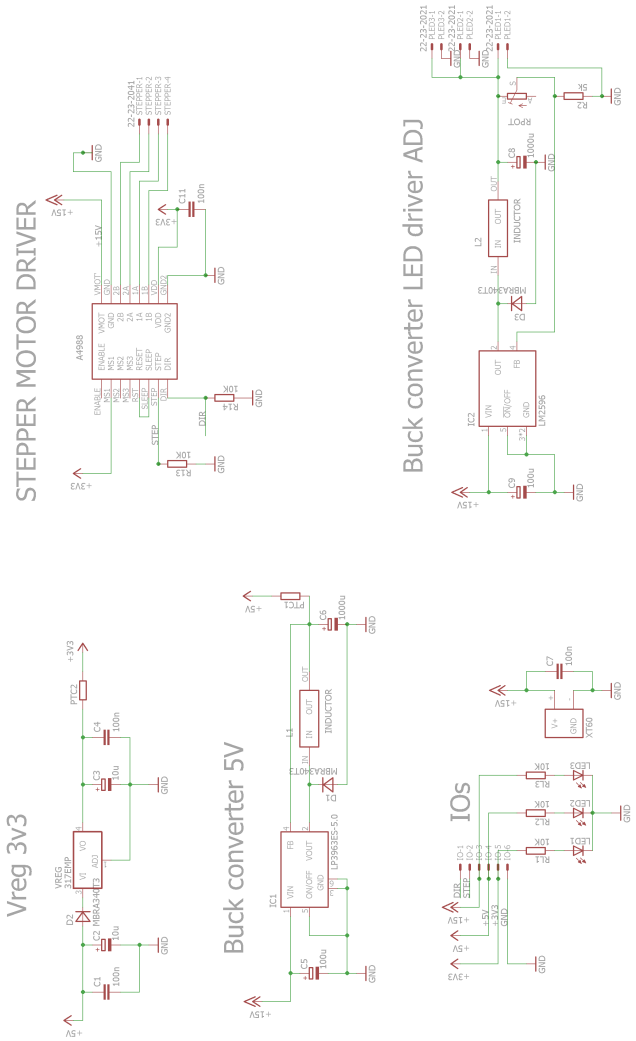


Figure C.1: *Schematic of the power distribution board with two Buck converters and one voltage regulator is used.*

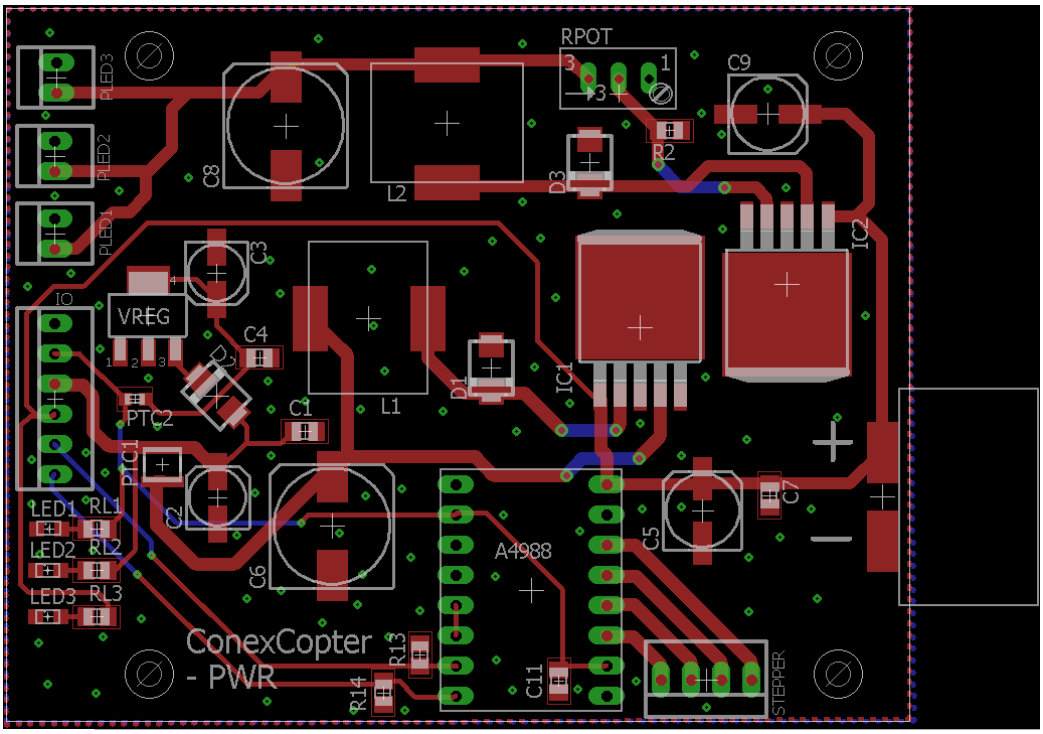


Figure C.2: *PCB layout of the power distribution board.*

APPENDIX D

Matlab code

D.1 Pointcloud generation script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MAPPING OF A 3D ENVIRONMENT USING A LIDAR,
% STEPPER MOTOR AND POSITIONING DEVICE
% LAMP_data3D.m
% Lars Jonsson - aloja-2@student.ltu.se
% 2017-05-20
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc; clear;
tStart = tic;
disp('Started running LAMP_data3D.m')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% OFFSETS AND LIMITATIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
time_offset = 10;
dMAX = 5;% max distance of LIDAR measurment
dMIN = 0.8; % min distance of LIDAR measurment
IMU_OFFSET = [0 0 0]';
motorstep_offset = 48;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOADING LIDAR, POS, MOTORSTEP and IMU DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

filename_LIDAR_timestamp = ...
filename_LIDAR = ...
filename_LAMP = ...

data_LIDAR = importdata(filename_LIDAR)';
data_LAMP = importdata(filename_LAMP);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% REMOVE START NOISE %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

data_LAMP = data_LAMP(round(length(data_LAMP)/10):end,2:end);
LIDAR_dist = data_LIDAR(:,time_offset:end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SORTING OF DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% data_LAMP = sortrows(data_LAMP);
```

```
% Timestamp
tstamp_LIDAR = importdata(filename_LIDAR_timestamp);
tstamp_LIDAR = tstamp_LIDAR(time_offset+1:end);

tstamp_LAMP = data_LAMP(:,1);
tstamp_LAMP = tstamp_LAMP - tstamp_LIDAR(1);
tstamp_LIDAR = tstamp_LIDAR - tstamp_LIDAR(1);

% Check FPS
dtLIDAR = diff(tstamp_LIDAR);
dtLAMP = diff(tstamp_LAMP);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% TIMESTAMP SYNCING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
j = 1;
for(i=1:length(tstamp_LIDAR))
    for(k = 1:length(tstamp_LAMP))
        if(abs(tstamp_LIDAR(i) - tstamp_LAMP(k)) < 10)
            TSTAMP(1,j) = tstamp_LAMP(k);
            TSTAMP(2,j) = tstamp_LIDAR(i);
            LAMP(j,:) = data_LAMP(k,2:end);
            LIDAR_DIST(:,j) = LIDAR_dist(:,i);
            j = j + 1;
        end
    end
end

% clearvars -except TSTAMP LAMP LIDAR_DIST LIDAR_ANGLE tStart

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Defines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dataLd = 1081; % FULL LIDAR RESOLUTION
N = size(LAMP); % NUMBER OF RECIEVED MOTORSTEPS
N = N(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LIDAR DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Ld = LIDAR_DIST; % raw Lidar data in [mm]
% Ld = LIDARfilter(Ld); % filter of raw data.
Ld = Ld./1000; % convert to [m]
% INTENS = zeros(N*dataLd,1);
LidarDataSize = 0;
for i=1:N
    for j=1:dataLd
        if(Ld(j,i) < dMAX && Ld(j,i) > dMIN)
            LidarDataSize = LidarDataSize + 1;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Copter position
%% (LED Position, determined by Camera and UWB) [m]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pos_x = LAMP(:,5);
pos_y = LAMP(:,6);
pos_z = LAMP(:,7);

copter_pos = [pos_x pos_y pos_z];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Yaw, Pitch, Roll angles of the copter
%% (should be < 10 degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% copter_angles = zeros(3,N); % No tilting of the copter
pitch = -LAMP(:,2); %PITCH
roll = LAMP(:,3); %ROLL
yaw = LAMP(:,4); %YAW
yaw = yaw-yaw(1);

copter_angles = [pitch-pitch(1) roll-roll(1) yaw]';
copter_angles = (copter_angles - IMU_OFFSET)*pi/180;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Stepper motor position
%% (angle of stepper motor, alpha)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

motorstep = LAMP(:,1);
motorstep_degree = motorstep/400*360;
motorstep_degree = motorstep_degree + motorstep_offset;

lidarstepper_conv = 7.006*(360/400); % one LASER lap = xx degree turn of stepper motor

alpha = zeros(dataLd,N);
alpha(1,:) = motorstep_degree;

% Interpolating motorstep over 1 lidar rotation
for k = 1:N
    for i = 2:dataLd
        alpha(i,k) = alpha(i-1,k) + lidarstepper_conv/1080;
        if(alpha(i,k) >= 360)
            alpha(i,k) = alpha(i,k) - 360;
        end
    end
end
alpha = alpha.*(pi/180);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LASER position
%% (angle of the emitting LASER, beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
beta = linspace(-135*pi/180,135*pi/180,1081);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOOPING THORUGH ALL LIDAR DATA
%% AND COPTER POSITIONS AND MAPPING
%% TO CAM FIXED SYSTEM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LIDAR_XYZ = zeros(LidarDataSize,4);
tic
disp('Looping throug LIDAR and Position data.....')
percent = 0;
k = 0;
l = 1;
fprintf('Percent of loop finished: %i%%', percent);
for i=1:round(N)
    for j = 1:dataLd
        if(Ld(j,i) < dMAX && Ld(j,i) > dMIN)
            LIDAR_XYZ(l,:) = mapping3D(copter_pos(i,:),copter_angles(:,i),...
            alpha(j,i),beta(1,j),Ld(j,i));
```

```
            l = l + 1;
        end
    end
    k = k + 1;
    if(k == round(N/100))
        percent = percent + 1;
        if percent > 10
            fprintf('%c%c%c%d%%',8,8,8,percent);
        else
            fprintf('%c%c%d%%',8,8,percent);
        end
        k = 0;
    end
end
fprintf('%c%c%c100%%\n',8,8,8);
toc

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %% SAVING DATA
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
prompt = 'Want to save the Point Cloud to file? [y/n] ';

X = LIDAR_XYZ(:,1);
Y = LIDAR_XYZ(:,2);
Z = LIDAR_XYZ(:,3);

str = input(prompt,'s');
if str == 'y'
    tic
    disp('Saving to file....')
%    data = [X,Y,Z,INTENS];
    data = [X,Y,Z];
    dlmwrite('ptcloud.xyz',data)
    disp('Saving successful!')
    toc
else
    disp('No file was saved');
end
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
tEnd = toc(tStart);
fprintf('Total time elapsed is: %f seconds.\n', tEnd);
```

D.2 Mapping of coordinate systems

```
function [Wall_point] = mapping3D(copter_pos,copter_angles,alpha,beta,Ld)
%% Constant variables
persistent d LIDAR_offset LOa T_lidar_laser R_copter_stepper_offset_x ...
    T_led_lidar cam_offset_angle_x cam_offset_angle_z
% offset from LED to LIDAR in [m]
LIDAR_offset = [-0.1 0.2 0.2];
LOa = 2.0*pi/180; % LIDAR OFFSET ANGLE
% Offset of LASER diode from center of LIDAR [m]
d = 0.03;

% Offset of the camera
cam_offset_angle_x = -1.9*pi/180;
cam_offset_angle_z = 0*pi/180;

Psi = copter_angles(3); Phi = copter_angles(1); Theta = copter_angles(2);

%%  CAM TO LIDAR
% copter_angle = [yaw pitch roll] = [Psi, Phi, Theta]
%       (yaw == Psi, pitch = Phi, roll = Theta)
% copter_pos = [copter_pos_x copter_pos_y copter_pos_z]
% LED_LIDAR_offset = [LED_offset_x LED_offset_y LED_offset_z]
%

% Camera offset angle matrices
camera_angle_matrix_x = [1 0 0 0;
    0 cos(cam_offset_angle_x) -sin(cam_offset_angle_x) 0;
    0 sin(cam_offset_angle_x) cos(cam_offset_angle_x) 0;
    0 0 0 1];

camera_angle_matrix_z = [cos(cam_offset_angle_z) -sin(cam_offset_angle_z) 0 0;
    sin(cam_offset_angle_z) cos(cam_offset_angle_z) 0 0;
    0 0 1 0;
    0 0 0 1];

% Translation matrix from Camera to LED
T_cam_led = [1 0 0 copter_pos(1);
    0 1 0 -copter_pos(2);
    0 0 1 copter_pos(3);
    0 0 0 1];

% Cam to copter rotation matrix
 R_cam_copter = [cos(Psi)*cos(Theta) -sin(Psi)*cos(Theta)+cos(Psi)*sin(Theta)*sin(Phi)
      sin(Psi)*sin(Phi)+cos(Psi)*sin(Theta)*cos(Phi) 0;
    sin(Psi)*cos(Theta) cos(Psi)*cos(Phi)+sin(Psi)*sin(Theta)*sin(Phi)
         -cos(Psi)*sin(Phi)+sin(Psi)*sin(Theta)*cos(Phi) 0;
     -sin(Theta) cos(Theta)*sin(Phi) cos(Theta)*cos(Phi) 0;
     0 0 0 1];


% Small Angle approximation sin(x)~x and cos(x)~(1-x^2/2)
% R_cam_copter_approx = [(1-Psi^2/2)*(1-Theta^2/2)
     -Psi*(1-Theta^2/2)+Phi*Theta*(1-Psi^2/2) Psi*Phi+Theta*(1-Psi^2/2)*(1-Phi^2/2) 0;
%     Psi*(1-Theta^2/2) (1-Psi^2/2)*(1-Phi^2/2)+Psi*Theta*Phi
     -(1-Psi^2/2)*Phi+Psi*Theta*(1-Phi^2/2) 0;
%     -Theta Phi*(1-Theta^2/2) (1-Theta^2/2)*(1-Phi^2/2) 0;
%     0 0 0 1];

T_led_lidar = [1 0 0 LIDAR_offset(1);
    0 1 0 LIDAR_offset(2);
```

```
    0 0 1 LIDAR_offset(3);
    0 0 0 1];


%% LIDAR measurement
R_copter_stepper = [cos(alpha) -sin(alpha) 0 0;
    sin(alpha) cos(alpha) 0 0;
    0 0 1 0;
    0 0 0 1];

R_copter_stepper_offset_x = [1 0 0 0;
    0 1-LOa^2/2 -LOa 0;
    0 LOa 1-LOa^2/2 0;
    0 0 0 1];

T_lidar_laser = [1 0 0 d;
    0 1 0 0;
    0 0 1 0;
    0 0 0 1];

T_laser_wall = [0 0 0 0;
    0 0 0 Ld*sin(beta);
    0 0 0 Ld*cos(beta);
    0 0 0 1];


Wall_point = camera_angle_matrix_x*camera_angle_matrix_z...
*T_cam_led*T_led_lidar*R_cam_copter*R_copter_stepper...
*R_copter_stepper_offset_x*T_lidar_laser*T_laser_wall*[0 0 0 1]';

end
```

Bibliography

[1] AutonomouStuff. 2d lidar scanning angle, 2010. URL https://autonomoustuff.com/wp-content/uploads/2016/08/PSxxx-270_270FOV_web.png. [Online; accessed January 18, 2017].

[2] S. Banerjee. Projective geometry , camera models and calibration. *University of Illinois*, 2011.

[3] G. Bradski. Open source computer vision library. https://http://opencv.org/, 2000.

[4] K. Gustavsson. UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals. *Uppsala University*, 2015.

[5] Hokuyo. *UST-10LX Specification*, 2014.

[6] Juan C. Mendez. Image of pitch, roll and yaw angles, 2014. URL http://theuav.net/assets/article_images/2014-11-19-quadcopter-dynamics/angles.png. [Online; accessed April 2, 2017].

[7] K.Alkawati, E.Lundell, L.Jonsson, M.Unander, H.Tjäder, F.Grönlund, and D.Björk. ConexCopter report. *Luleå University of Technology*, 2017. URL https://gitlab.henriktjader.com/D7039E-E7025E/conexreport/raw/master/D7039E-ConexCopterProjectReport.pdf.

[8] MATLAB. *version R2016b*. The MathWorks Inc., Natick, Massachusetts, 2016.

[9] MATLAB. *Computer Vision System Toolbox, R2016b*. The MathWorks Inc., Natick, Massachusetts, 2017.

[10] A. Pavlov. Hokuyolx, python libarry. https://github.com/SkRobo/hokuyolx, 2017.

[11] Satya Mallick. Camera field of view, 2016. URL http://www.learnopencv.com/wp-content/uploads/2016/06/hfov-vfov-dfov.jpg. [Online; accessed March 1, 2017].

[12] SightMachine. Simplecv, python library. http://simplecv.org/, 2012.

[13] J. Sola. Quaternion Kinematics for the Error-State KF. pages 40–44, 2012.

[14] M. Unander. Autonomous flying of quadrotor in mineshaft for 3D modeling and inspection. Master's thesis, Luleå University of Technology, June 2017.

[15] VirtualGrid. Vrmesh studio. http://www.vrmesh.com/default.asp.

[16] XSens. Mt software suite. https://www.xsens.com/mt-software-suite/, December 2016.

[17] *Data sheet MTi 1-series*. XSens Technologies BV, December 2015. Rev D.

[18] ZeroOne, Wikipedia User. Roll, yaw and pitch axis definition for an airplane, 2007. URL https://commons.wikimedia.org/wiki/File%3AFlight_dynamics_with_text.png. [Online; accessed Mach 6, 2017].