Bachelor's project

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

Erlend Holseker
Arvin Khodabandeh
Erik Bjørnøy
Isak Gamnes Sneltvedt

Sensor Fusion for Position and Spatial Attitude Estimation of Offshore Motion Compensated Gangways

Bachelor's project in Automation Technology
Supervisor: Aleksander Larsen Skrede
Co-supervisor: Ottar Laurits Osen
May 2021

NTNU
Kunnskap for en bedre verden

Erlend Holseker
Arvin Khodabandeh
Erik Bjørnøy
Isak Gamnes Sneltvedt

Sensor Fusion for Position and Spatial Attitude Estimation of Offshore Motion Compensated Gangways

Bachelor's project in Automation Technology
Supervisor: Aleksander Larsen Skrede
Co-supervisor: Ottar Laurits Osen
May 2021

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

NTNU
Norwegian University of
Science and Technology

NTNU

Kunnskap for en bedre verden

Sensor Fusion for Position and Spatial Attitude Estimation of Offshore Motion Compensated Gangways

Erik Bjørnøy

Isak Gamnes Sneltvedt

Arvin Khodabandeh

Erlend Holseker

May 2021

PROJECT / BACHELOR THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor 1: Aleksander L. Skrede

Supervisor 2: Ottar L. Osen

Preface

This bachelor thesis is written by four Automation Engineering students at NTNU in Ålesund, and marks the end of three years of engineering studies. The purpose of the project is to study possible solutions for estimating the position and orientation of a moving object in relation to another object. Development of sensor data processing software and testing instrumentation solutions is also central in this project.

In an age where renewable and environmentally friendly energy sources are more important than ever, wind turbines play a significant role. Wind turbines are large and noisy, and they are unpopular among many because of this. To avoid building wind turbines on land close to people, making excessive interventions in beautiful nature, the number of offshore wind turbines is increasing. These wind turbines need regular maintenance, and transportation of personnel to the wind turbines must take place in a safe and secure manner. Finding solutions that can increase the safety is therefore important.

Our motivation to choose this project was the opportunity to utilize a large part of the knowledge we have acquired throughout three years of studying Automation Engineering, as well as the need to learn new techniques to conduct the project. The problem statement presented by Seaonics intrigued us, and we saw it as an opportunity to further increase our knowledge in sensor technologies, and to work with topics that has not been lectured during our studies. We hope that later projects can utilize experiences from this project in further development of position and orientation estimation systems.

Acknowledgement

We would like to thank all the contributors who have given us support throughout the project, and we would especially like to thank:

- Our supervisors Aleksander L. Skrede and Ottar L. Osen for good support, motivating conversations and good guidance throughout the project.
- Seaonics, Senior Control System Engineer Daniel Nordal Bjørneseth especially, for advice and guidance throughout the project. We would also like to thank them for providing us with office facilities when the local covid-19 regulations has allowed it. They have also been very helpful in procuring equipment.
- Lab engineer Anders Sætersmoen at NTNU for lending us necessary equipment.
- Associate professor Guoyuan Li at NTNU for lending us IMU sensors.
- Family and friends for being supportive and understanding throughout the project period.

Abstract

This report concerns a bachelor project proposed by Seaonics, a company based in Ålesund that develops software and hardware solutions for the maritime sector. Seaonics wants to find an appropriate solution for estimating position and orientation of an offshore gangway.

The purpose of this project is to develop a position and orientation estimation system, which can form the basis for further development in later projects. Important aspects of the development are explained in the report. Choice of sensor technologies, sensor fusion algorithms, and the implementation of stereo-camera for position estimation is explained in detail. Testing of the developed solutions is explained, and the results from these tests are presented.

The results from the tests shows that estimating orientation with reliable results is possible using a gyroscope and an accelerometer in combination with sensor fusion algorithms. Including a magnetometer in these estimations yields unreliable estimates and is therefore not verified as a good solution. Using a stereo-camera to estimate position proves to be efficient under the right conditions. However, it does not give any information that can be used to estimate position along the z-axis (up and down). Additional techniques that can be investigated further is also discussed in the report.

Contents

Terminology

A posteriori ..... Latin for *From the later*

A priori ..... Latin for *From the earlier*

atan2 ..... two-argument arctan function, can give a solution in range $(-\pi, \pi]$

Baseline ..... In terms of a stereo camera, baseline is the distance between the two cameras

Extrinsic parameters ..... Includes the relative rotation and translation between two cameras

Heave ..... A linear motion along the vertical z-axis

Intrinsic parameters The parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

***n*-tuple** ..... A list of n elements, where each value is an unsigned (positive) integer

Orientation ..... Angular position of one object in relation to another object

Pitch ..... A rotation around the traverse axis

Position ..... Placement of an object in space in relation to another object

Roll ..... A rotation around the longitudinal axis

Surge ..... A linear motion along the longitudinal x-axis

Sway ..... A linear motion along the traverse y-axis

Yaw ..... A rotation around the vertical axis

Notation

s$\cdot$ ..... Sine function, $\sin(\cdot)$

c$\cdot$ ..... Cosine function, $\cos(\cdot)$

$\mathbf{I}_{n\times n}$ ..... Identity matrix with dimensions $n \times n$

${}^{A}_{B}\hat{\mathbf{q}}$ ..... Orientation of frame B relative to frame A

${}^{A}\hat{\mathbf{r}}$ ..... Vector described in frame A

$\Re$ ..... All real numbers

Abbreviations

ADC ..... Analog-to-Digital Converter

AFOV ..... Angular Field of View

AGC ..... Automatic Gain Control

AMC ..... Active Motion Compensation

CCD ..... Charge-coupled Device

CFA ..... Color filter array

CMOS ..... Complementary Metal Oxide Semiconductor

CVG ..... Coriolis Vibratory Gyroscope

DOF ..... Degrees of Freedom, number of configurations for a object

dps ..... Degrees per second

GUI ..... Graphical User Interface

FLANN ..... Fast Library for Approximate Nearest Neighbors

FMCW ..... Frequency Modulated Continuous Waves

FOV ..... Field of View

G ..... Gravitational acceleration, 1G $\approx$ 9.81 m/s^2

HSV ..... Hue, saturation, value

I^{2}C ..... Inter-Integrated Circuit

IDE ..... Integrated Development Environment

IMU ..... Inertial Measurement Unit

I/O ..... Input/Output

IR ..... Infrared

KNN ..... K-Nearest Neighbour

LED ..... Light-Emitting Diode

MARG ..... Magnetic Angular Rate and Gravity

MCU ..... Micro Controller Unit

MEMS ..... Microelectromechanical Systems

MSB/LSB ..... Most Significant Bit/Least Significant Bit

RGB ..... Red, Green, Blue

ROI ..... Region of Interest

SIC ..... Soft Iron Calibration

SIFT ..... Scale Invariant Feature Transform

SNAME ..... Society of Naval Architects and Marine Engineers

Stereo VIO ..... Stereo Visual Odometry

TDOA ..... Time Difference of Arrival

ToF ..... Time of Flight

UWB ..... Ultra Wide Band

VSLAM ..... Visual Simultaneous Localization and Mapping

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Norway is one of the pioneering nations in offshore technology [23]. The maritime cluster on the north-western part of Norway has been, and still is a significant contributor to this development. For decades, the maritime industry has been providing the community with employment, and this is still the case. A large part of the people living in this area are employed by companies that are directly or indirectly related to this industry. Even if the world's nations are slowly working towards a goal of no oil, there will still be a need to utilize the resources in the ocean. Wind and waves are great energy sources that will be even more important to utilize in the future. In addition, the safety of offshore workers will always be important. Developing systems that are efficient and safe is therefore important for Norway to remain competitive in the industry.

1.2 Introduction

Seaonics delivers systems for active motion compensation (AMC) of offshore gangways and cranes. The gangways are used to transport equipment and personnel from the ship to wind turbines in the sea, among other things. Safety is therefore important. Their system utilizes motion sensors stationed on the ship to detect how the ship is moving. The system uses this data to counteract the motions to keep the gangway, or the load hanging from a crane, as still as possible. However, their system currently has no way to monitor the actual movement of the

load or gangway in relation to the ship. Data like this can be useful to increase the performance and safety of their system. The aim of this bachelor project is to research possible ways to estimate the motion of an offshore gangway in relation to another object. Areas of focus in the research is choice of sensor technology, how to combine the sensor data using sensor fusion algorithms, and how to calculate the orientation and position with the provided data. Additionally, a prototype for testing the orientation estimation system, including a graphical user interface, is developed. The project is proposed by Seaonics.

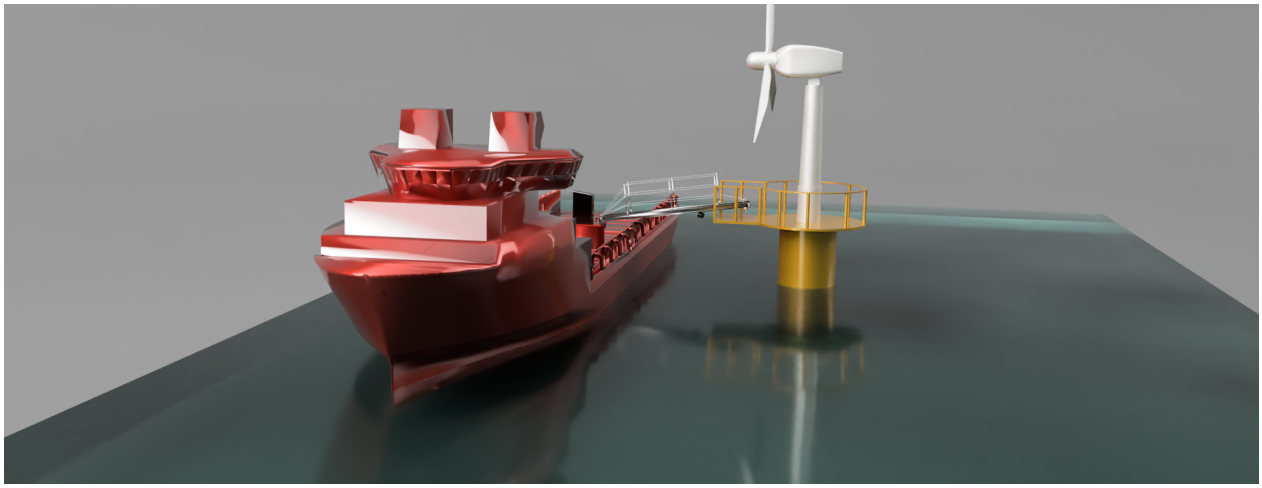


Figure 1.1: 3D model of ship, gangway and wind turbine

1.3 Problem Formulation

The problem of this project can be divided into three parts. The first part concerns the research of different types of sensor technologies applicable for this project. The second part is about exploring different sensor fusion algorithms to estimate orientation. The third part is concentrating on different ways to estimate distance and position in relation to another project.

Problems to be addressed

- Research the advantages and disadvantages of different sensor technologies in the scope of this project.
- Explore different algorithms to estimate orientation based on sensor data.

- Explore different ways to estimate distance and position in relation to another object.

1.4 Literature Survey

Parts of this report is based on Rudolf E. Kálmán's famous paper describing a recursive solution to the discrete-data linear filtering problem, known as the Kalman filter, published in 1960 [24]. It is also partly based on Sebastian O.H. Madgwick's paper describing an efficient way to estimate orientation using inertial and magnetic sensors, known as the Madgwick filter, published in 2010 [14].

1.5 Limitations

A solution to the problem statement must satisfy the following criteria:

- Sensors can only be placed on the ship and/or the AMC gangway. This is because the wind turbines are not standardized (yet), and the system can therefore not be dependent on equipment mounted on the wind turbines.
- Sensors must be placed in a way that does not impede the functionality of the gangway.
- When choosing sensors, it is important to take weather conditions into account.
- If magnetometers are to be used, magnetic field distortion must be taken into account.

1.6 Structure of the Report

The rest of the report is structured as follows:

Chapter 2 - Theoretical basis: Contains a summary of the theoretical background necessary for assessments and choices later in the report.

Chapter 3 - Materials and Methods: Contains a description of materials and methodology used in the project, and a description of the project's organization. This chapter explains how different sensor fusion algorithms are applied, as well as how a stereo camera is created for distance- and position estimation. In the end of the chapter, the test procedures are explained.

Chapter 4 - Result: Contains a description of the application that is made for orientation estimation. It also presents the results from the testing.

Chapter 5 - Discussion: Contains a critical discussion of the results from Chapter 4. This includes a discussion of what distinguishes the developed solution from a complete solution.

Chapter 6 - Conclusions: This chapter presents an overall conclusion of the project, and suggests an answer to the problem statement.

Chapter 2

Theoretical basis

This chapter contains the theoretical basis that is important to understand when conducting the methods of this project. This includes kinematics, reference systems, sensor technologies, sensor fusion including the statistical theory behind this concept, and image processing.

2.1 Kinematics

Kinematics is the mathematical description of motion [25]. Kinematics is a subfield of physical dynamics, which only treats the geometrical aspects of motion without considering the forces that causes the motion.

2.1.1 Reference systems

When calculating and analyzing ship motions, or the motions of any rigid body, it is appropriate to use several coordinate systems to reference the position and orientation of the body. If a rigid body is said to have a given orientation and position, this information will be useless if it is not given what it is positioned and oriented **in relation to**. Figure 2.1 shows a ship with two reference frames; a body-fixed frame, and the Earth-fixed frame.

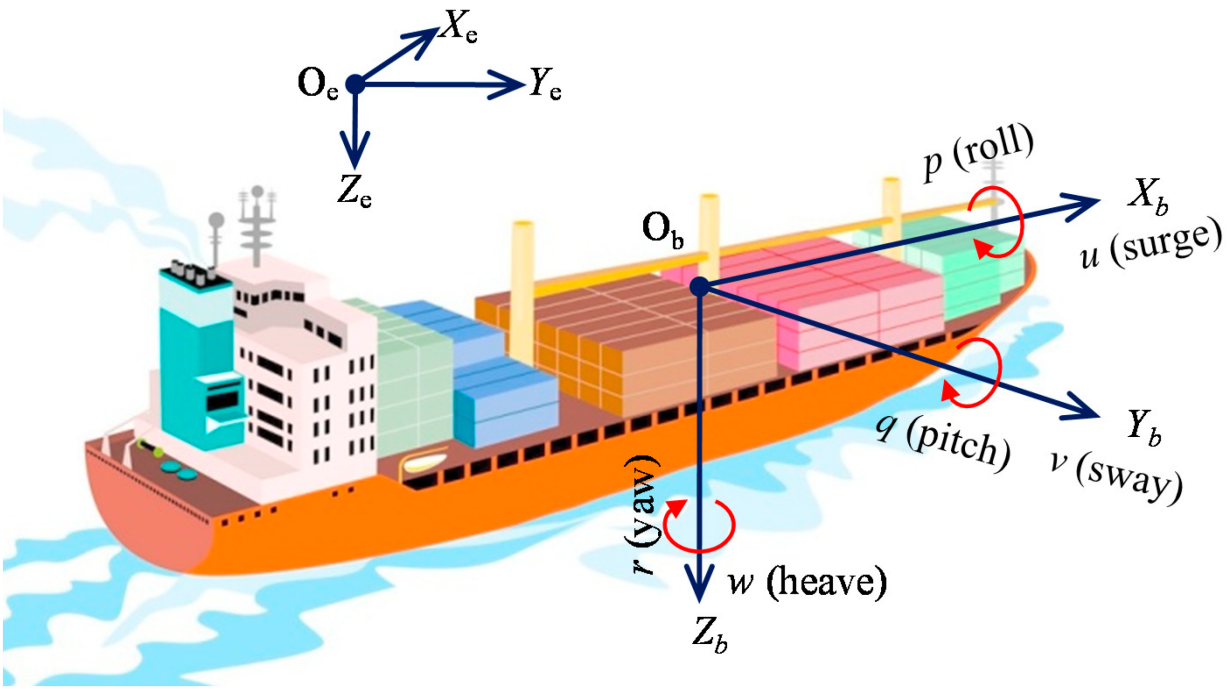


Figure 2.1: A ship's body-fixed and earth-fixed coordinate frames [1]

2.1.2 Rigid-body motions in the plane

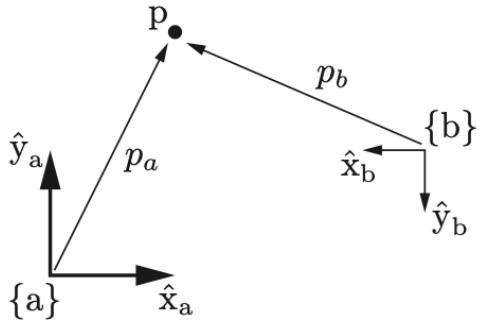


Figure 2.2: The point p represented in two different ways [2]

As seen in figure 2.2, a point can be represented in many different ways, depending on what it is referenced in relation to. Let us say that reference frame {a} has unit coordinate axes, $\hat{x}_a$ and $\hat{y}_a$. Then, the point p is represented as $p_a = (1,2)$. However, if reference frame {b} is used, which has a different placement, orientation, and length scale, the point p is represented as $p_b = (4,-2)$.

When analyzing the motions of a rigid body, it is convenient to attach a reference frame with unit axes to the rigid body. This reference frame is called the body frame and is denoted {b}. For

simplicity, the body frame is considered to be attached to the moving rigid body, but that is not the case. In fact, the body frame is the stationary frame that is instantaneously coincident with the frame moving along with the rigid body, at a specific moment [2].

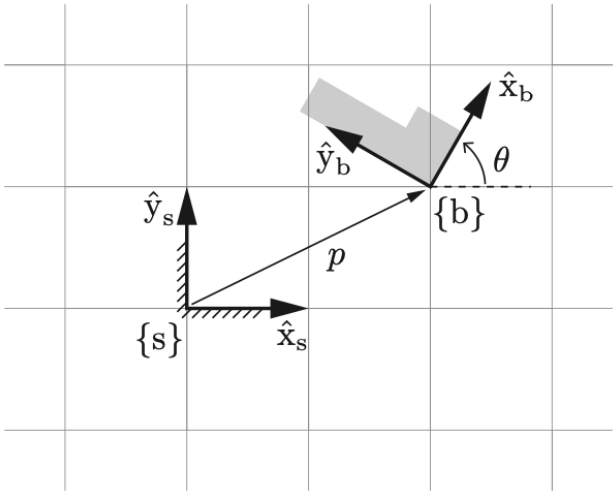


Figure 2.3: Example of a body frame [2]

The gray shape in figure 2.3 represents a planar body. The reference frame {b} is the body frame, and {s} is a fixed reference frame. Both frames have unit axes. The position and orientation of the planar body can be described by specifying the position and orientation of the body frame with respect to the fixed reference frame. The origin of the body frame, p, can be expressed in terms of the coordinate axes of the fixed reference frame as

$$p = p_x \hat{x}_s + p_y \hat{y}_s \tag{2.1}$$

In equation 2.1, $\hat{x}_s$ and $\hat{y}_s$ indicates that p is defined in respect to the fixed reference frame {s}. To describe the orientation of the body frame {b}, the directions of the unit axes, $\hat{x}_b$ and $\hat{y}_b$, should be specified relative to the fixed reference frame {s} on the form

$$\hat{x}_b = \cos\theta \cdot \hat{x}_s + \sin\theta \cdot \hat{y}_s \tag{2.2}$$

$$\hat{y}_b = -\sin\theta \cdot \hat{x}_s + \cos\theta \cdot \hat{y}_s \tag{2.3}$$

If everything is expressed in terms of the fixed reference frame {s}, the point p can be represented as a column vector of the form

$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2.4}$$

The two vectors of the body frame, $\hat{x}_b$ and $\hat{y}_b$, can also be written as column vectors inside a 2 x 2 matrix as follows:

$$P = \begin{bmatrix} \hat{x}_b & \hat{y}_b \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.5}$$

The matrix P is a rotation matrix. Each column of P must be a unit vector, and the two columns must be orthogonal to each other. The last degree of freedom is parameterized by θ. Now, using equation 2.4 and 2.5, the pair (P, p) describes the orientation and the position of the body frame {b} in relation to the fixed reference frame {s}.

2.1.3 Rigid-body motions in three dimensions

The concepts above can be generalized to three-dimensional rigid-body motions. Let {s} be the fixed frame and {b} the body frame. The unit axes of the fixed frame can be denoted $\{\hat{x}_s, \hat{y}_s, \hat{z}_s\}$, and the unit axes of the body frame can be denoted $\{\hat{x}_b, \hat{y}_b, \hat{z}_b\}$. The vector p is the vector from the origin of {s} to the origin of {b}. Then, in terms of the coordinates of {s}, p can be expressed as

$$p = p_1 \cdot \hat{x}_s + p_2 \cdot \hat{y}_s + p_3 \cdot \hat{z}_s \tag{2.6}$$

The axes of {b} can be expressed as

$$\hat{x}_b = r_{11} \cdot \hat{x}_s + r_{21} \cdot \hat{y}_s + r_{31} \cdot \hat{z}_s \tag{2.7}$$

$$\hat{y}_b = r_{12} \cdot \hat{x}_s + r_{22} \cdot \hat{y}_s + r_{32} \cdot \hat{z}_s \tag{2.8}$$

$$\hat{z}_b = r_{13} \cdot \hat{x}_s + r_{23} \cdot \hat{y}_s + r_{33} \cdot \hat{z}_s \tag{2.9}$$

This yields

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}, \qquad R = \begin{bmatrix} \hat{\mathrm{x}}_b & \hat{\mathrm{y}}_b & \hat{\mathrm{z}}_b \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.10}$$

Then, the 12 parameters given by the pair (R, p) provide a description of the position and orientation of a three-dimensional rigid body, relative to the fixed frame.

2.1.4 Motion variables

A ship can move in six degrees of freedom (DOF), which means that six independent variables are needed to describe the position and orientation of the ship. The first three variables (x, y, z), and their time derivatives (u, v, w), describe the position and the translational motion along the x, y and z axes. The last three variables (ϕ, θ, ψ), and their time derivatives (p, q, r), describe the orientation and rotational motion about the axes. The notation of SNAME (Society of Naval Architects and Marine Engineers) for these six different motion components are *surge*, *sway*, *heave*, *roll*, *pitch* and *yaw*. See figure 2.1 and table 2.1.

Table 2.1: The notation of SNAME (1950) for marine vessels [22]

DOF		Forces and moments	Linear and angular velocities	Positions and Euler angles
1	motions in the x direction (surge)	X	u	x
2	motions in the y direction (sway)	Y	v	y
3	motions in the z direction (heave)	Z	w	z
4	rotation about the x axis (roll, heel)	K	p	ϕ
5	rotation about the y axis (pitch, trim)	M	q	θ
6	rotation about the z axis (yaw)	N	r	ψ

2.2 Representation of position and orientation

2.2.1 Euler angles

The Euler angles are used to describe the orientation of a rigid body in relation to a fixed coordinate system, either by the initial frame or the frame resulting in the most recent rotation (body fixed frame). By using Euler angles, any rotation can be described by three successive rotations about linearly independent axes. The Euler angles are commonly denoted as ψ, θ and ϕ for *yaw*, *pitch* and *roll* respectively [26].

Singularity

Euler angle representation of an objects orientation can suffer from singularity, often called gimbal lock. This happens when two rotation axes coincide, e.g. if an object pitches 90°, the x-axis and z-axis will become parallel. Then the ϕ and ψ angles will describe the same rotations. When this occurs, one degree of freedom is lost, and an infinite number of solutions exists for the Euler rotation sequence [27].

2.2.2 Quaternions

The quaternion number system is an extension of the complex numbers. The concept of quaternions was discovered by the Irish mathematician Sir William Rowan Hamilton in 1843.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.11}$$

Following equation 2.11, Hamilton defined a set of rules;

$$\begin{aligned} ij = k, \qquad jk = i, \qquad ki = j, \\ ji = -k \qquad kj = -i, \qquad ik = -j, \end{aligned} \tag{2.12}$$

and a quaternion q as

$$q = s + ia + jb + kc \qquad s, a, b, c \in \mathbb{R} \tag{2.13}$$

[4]. The relationship between i, j and k is similar to the cross product rules for unit cartesian vectors:

$$\begin{aligned} x \times y = z, \qquad y \times z = x, \qquad z \times x = y, \\ y \times x = -z \qquad z \times y = -x \qquad x, \times z = -y \end{aligned} \tag{2.14}$$

The imaginary numbers i, j, and k can be used to represent three cartesian unit vectors **i**, **j**, **k** with the same properties as imaginary numbers, meaning $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$. See figure 2.4.

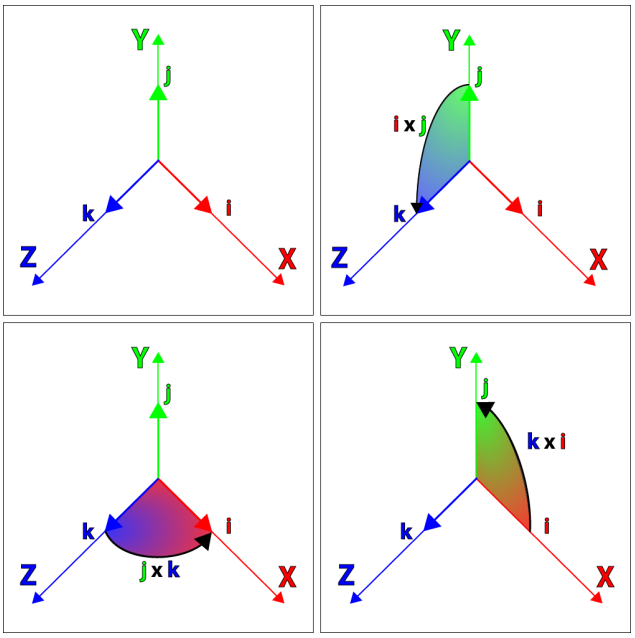


Figure 2.4: The relationship between the cartesian unit vectors represented by **i**, **j** and **k** [3]

A quaternion can be presented as an ordered pair:

$$q = [s,\ \mathbf{v}] = [s,\ x\mathbf{i} + y\mathbf{j} + z\mathbf{k}] \qquad s, x, y, z \in \mathbb{R}, \mathbf{v} \in \mathbb{R}^3 \tag{2.15}$$

Quaternion calculation rules

A complete derivation of the equations and calculation rules in this subsection can be found in the book *Quaternions for Computer Graphics* written by Professor John Vince at Bournemouth University [4].

Adding and subtracting quaternions is done in a similar way as with complex numbers:

$$\begin{aligned} q_a &= [s_a,\ \mathbf{a}] \\ q_b &= [s_b,\ \mathbf{b}] \\ q_a + q_b &= [s_a + s_b,\ \mathbf{a} + \mathbf{b}] \\ q_a - q_b &= [s_a - s_b,\ \mathbf{a} - \mathbf{b}] \end{aligned} \tag{2.16}$$

A quaternion product has the following general equation:

$$[s_a,\ \mathbf{a}][s_b,\ \mathbf{b}] = [s_a s_b - \mathbf{a}\cdot\mathbf{b},\ s_a\mathbf{b} + s_b\mathbf{a} + \mathbf{a}\times\mathbf{b}] \tag{2.17}$$

Multiplying a quaternion by a scalar is done by the following rule:

$$\begin{aligned} q &= [s,\ \mathbf{v}] \\ \lambda q &= \lambda[s,\ \mathbf{v}] \\ &= [\lambda s,\ \lambda\mathbf{v}] \end{aligned} \tag{2.18}$$

A **Real** quaternion is a quaternion where the vector term is $\mathbf{0}$, $q = [s, \mathbf{0}]$, whereas a **Pure** quaternion is a quaternion with zero scalar term; $q = [0, \mathbf{v}]$. An arbitrary vector $\mathbf{v}$ can be expressed in both its scalar magnitude, and its direction:

$$\mathbf{v} = v\hat{\mathbf{v}} \quad \text{where} \quad v = |\mathbf{v}|, \quad |\hat{\mathbf{v}}| = 1 \tag{2.19}$$

Combining the definition above with the definition of pure quaternions, yields:

$$\begin{aligned} q &= [0,\ \mathbf{v}] \\ &= [0,\ v\hat{\mathbf{v}}] \\ &= v[0,\ \hat{\mathbf{v}}] \end{aligned} \tag{2.20}$$

Equation 2.20 yields the description of a unit quaternion that has a zero scalar and a unit vector:

$$\hat{q} = [0,\ \hat{\mathbf{v}}] \tag{2.21}$$

The quaternion conjugate can be computed by negating the vector part:

$$\begin{aligned} q &= [s,\ \mathbf{v}] \\ q^* &= [s,\ -\mathbf{v}] \end{aligned} \tag{2.22}$$

The product of a quaternion with its conjugate is:

$$\begin{aligned} qq^* &= [s,\ \mathbf{v}][s,\ -\mathbf{v}] \\ &= [s^2 - \mathbf{v}\cdot -\mathbf{v},\ -s\mathbf{v} + s\mathbf{v} + \mathbf{v}\times -\mathbf{v}] \\ &= [s^2 + \mathbf{v}\cdot\mathbf{v},\ \mathbf{0}] \\ &= [s^2 + v^2,\ \mathbf{0}] \end{aligned} \tag{2.23}$$

The quaternion norm (magnitude) is defined similar to the norm of complex numbers:

$$\begin{aligned} q &= [s,\ \mathbf{v}] \\ |q| &= \sqrt{s^2 + v^2} \end{aligned} \tag{2.24}$$

When $|q| = \sqrt{s^2 + v^2} = 1$, it is called **unit norm**. By comparing equation 2.23 and 2.24, one can see that

$$qq^* = |q|^2 \tag{2.25}$$

Quaternion normalization is done using the quaternion norm:

$$q' = \frac{q}{|q|} = \frac{q}{\sqrt{s^2 + v^2}} \tag{2.26}$$

The quaternion inverse is computed by taking the conjugate of the quaternion and divide it by the square of the norm:

$$q^{-1} = \frac{q^*}{|q|^2} \tag{2.27}$$

The dot product between two quaternions can be computed by multiplying the correspond-

ing scalar parts and summing the result:

$$\begin{aligned} q_1 &= [s_1,\ x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}] \\ q_2 &= [s_2,\ x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}] \\ q_1 \cdot q_2 &= s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2 \end{aligned} \tag{2.28}$$

The quaternion dot product can be used to calculate the angular difference between two quaternions:

$$\cos\theta = \frac{q_1 \cdot q_2}{|q_1||q_2|} = \frac{s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2}{|q_1||q_2|} \tag{2.29}$$

Rotations

Complex numbers can be used to rotate a point through the 2D complex plane:

$$q = \cos\theta + i\sin\theta \tag{2.30}$$

Similarly, it should be possible to express a quaternion to be used to rotate a point in 3D-space:

$$q = [\cos\theta,\ \sin\theta\mathbf{v}] \tag{2.31}$$

This can be tested by computing the product of the quaternion q and the vector **p**. Let p be the pure quaternion

$$p = [0,\ \mathbf{p}] \qquad \mathbf{p} \in \mathbb{R}^3, \tag{2.32}$$

and q be the unit-norm quaternion

$$q = [s,\ \lambda\hat{\mathbf{v}}] \qquad s, \lambda \in \mathbb{R}, \quad \hat{\mathbf{v}} \in \mathbb{R}^3 \tag{2.33}$$

Now, the product $p' = qp$ can be computed to examine if the vector **p** has been rotated:

$$\begin{aligned} p' &= qp \\ &= [s,\ \lambda\hat{\mathbf{v}}][0,\ \mathbf{p}] \\ &= [-\lambda\hat{\mathbf{v}}\cdot\mathbf{p},\ s\mathbf{p} + \lambda\hat{\mathbf{v}}\times\mathbf{p}] \end{aligned} \tag{2.34}$$

The result of equation 2.34 is a general quaternion with both a scalar and a vector component.

Considering the "special" case where the vector **p** is perpendicular to $\hat{\mathbf{v}}$, where the dot product $-\lambda\hat{\mathbf{v}}\cdot\mathbf{p}=0$, one can see that the result of equation 2.34 is a **Pure** quaternion:

$$p' = [0,\ s\mathbf{p} + \lambda\hat{\mathbf{v}}\times\mathbf{p}] \tag{2.35}$$

To rotate **p** about $\hat{\mathbf{v}}$ in this case, s and λ is substituted as $s=\cos\theta$ and $\lambda=\sin\theta$. This yields

$$p' = [0,\ \cos\theta\mathbf{p} + \sin\theta\hat{\mathbf{v}}\times\mathbf{p}] \tag{2.36}$$

If, for example **p** were to be rotated 45° about the z-axis, the quaternion would be:

$$\begin{aligned} q &= [\cos\theta,\ \sin\theta\mathbf{k}] \\ &= \left[\frac{\sqrt{2}}{2},\ \frac{\sqrt{2}}{2}\mathbf{k}\right] \end{aligned} \tag{2.37}$$

Choosing a vector **p** that belongs to the special case where **p** is perpendicular to **k**:

$$p = [0,\ 2\mathbf{i}] \tag{2.38}$$

The product of qp is:

$$\begin{aligned} p' &= qp \\ &= \left[\frac{\sqrt{2}}{2},\ \frac{\sqrt{2}}{2}\mathbf{k}\right][0,\ 2\mathbf{i}] \\ &= \left[0,\ 2\frac{\sqrt{2}}{2}\mathbf{i} + 2\frac{\sqrt{2}}{2}\mathbf{k}\times\mathbf{i}\right] \\ &= [0,\ \sqrt{2}\mathbf{i} + \sqrt{2}\mathbf{j}] \end{aligned} \tag{2.39}$$

The result of equation 2.39 is a pure quaternion that is rotated 45° about the **k** axis. The magnitude of the resulting vector is

$$|\mathbf{p'}| = \sqrt{\sqrt{2}^2 + \sqrt{2}^2} = 2, \tag{2.40}$$

which confirms that the magnitude of the vector is the same as before being rotated. The rotations is shown in figure 2.5.

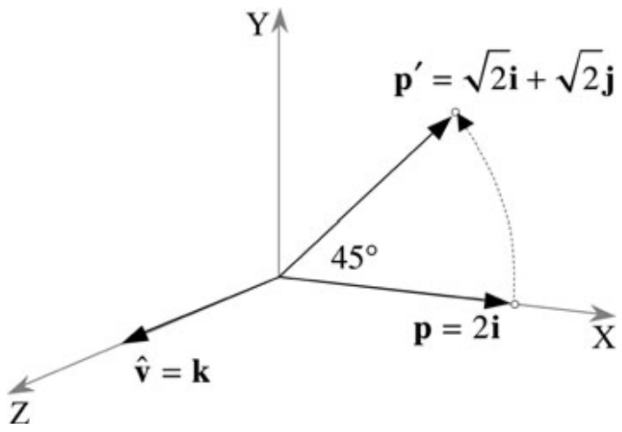


Figure 2.5: The vector 2**i** is rotated 45° [4]

Now, considering a case where the quaternion is not orthogonal to **p**, for example 45° offset from **p**:

$$\begin{aligned} \hat{\mathbf{v}} &= \frac{\sqrt{2}}{2}\mathbf{i} + \frac{\sqrt{2}}{2}\mathbf{k} \\ \mathbf{p} &= 2\mathbf{i} \\ q &= [\cos\theta,\ \sin\theta\hat{\mathbf{v}}] \\ p &= [0,\ \mathbf{p}] \end{aligned} \tag{2.41}$$

Multiplying the vector **p** by q now yields:

$$\begin{aligned} p' &= qp \\ &= [\cos\theta,\ \sin\theta\hat{\mathbf{v}}][0,\ \mathbf{p}] \\ &= [-\sin\theta\hat{\mathbf{v}}\cdot\mathbf{p},\ \cos\theta\mathbf{p} + \sin\theta\hat{\mathbf{v}}\times\mathbf{p}] \end{aligned} \tag{2.42}$$

Substituting $\hat{\mathbf{v}}$, **p** and $\theta = 45°$, yields:

$$\begin{aligned} p' &= \left[-\frac{\sqrt{2}}{2}\left(\frac{\sqrt{2}}{2}\mathbf{i} + \frac{\sqrt{2}}{2}\mathbf{k}\right)\cdot(2\mathbf{i}),\ \frac{\sqrt{2}}{2}2\mathbf{i} + \frac{\sqrt{2}}{2}\left(\frac{\sqrt{2}}{2}\mathbf{i} + \frac{\sqrt{2}}{2}\mathbf{k}\right)\times 2\mathbf{i}\right] \\ &= [-1,\ \sqrt{2}\mathbf{i} + \mathbf{j}] \end{aligned} \tag{2.43}$$

The result from equation 2.43 is not a pure quaternion. Also, it has not been rotated 45° and the norm of the vector has been reduced from 2 to $\sqrt{3}$. However, if the result of qp is post-multiplied by the inverse of q, q^{-1}, the result will be a pure quaternion where the norm of the

vector is maintained [3]. The inverse of q is:

$$\begin{aligned} q &= \left[\cos\theta,\ \sin\theta\left(\frac{\sqrt{2}}{2}\mathbf{i}+\frac{\sqrt{2}}{2}\mathbf{k}\right)\right] \\ q^{-1} &= \left[\cos\theta,\ -\sin\theta\left(\frac{\sqrt{2}}{2}\mathbf{i}+\frac{\sqrt{2}}{2}\mathbf{k}\right)\right] \end{aligned} \tag{2.44}$$

For $\theta = 45°$:

$$\begin{aligned} q^{-1} &= \left[\frac{\sqrt{2}}{2},\ -\frac{\sqrt{2}}{2}\left(\frac{\sqrt{2}}{2}\mathbf{i}+\frac{\sqrt{2}}{2}\mathbf{k}\right)\right] \\ &= \frac{1}{2}\left[\sqrt{2},\ -\mathbf{i}-\mathbf{k}\right] \end{aligned} \tag{2.45}$$

Post-multiplying qp with q^{-1} now yields:

$$\begin{aligned} qp &= \left[-1,\ \sqrt{2}\mathbf{i}+\mathbf{j}\right] \\ qpq^{-1} &= \left[-1,\ \sqrt{2}\mathbf{i}+\mathbf{j}\right]\frac{1}{2}\left[\sqrt{2},\ -\mathbf{i}-\mathbf{k}\right] \\ &= \frac{1}{2}\left[-\sqrt{2}-\left(\sqrt{2}\mathbf{i}+\mathbf{j}\right)\cdot(-\mathbf{i}-\mathbf{k}),\ \mathbf{i}+\mathbf{k}+\sqrt{2}\left(\sqrt{2}\mathbf{i}+\mathbf{j}\right)-\mathbf{i}+\sqrt{2}\mathbf{j}+\mathbf{k}\right] \\ &= \frac{1}{2}\left[-\sqrt{2}+\sqrt{2},\ \mathbf{i}+\mathbf{k}+2\mathbf{i}+\sqrt{2}\mathbf{j}-\mathbf{i}+\sqrt{2}\mathbf{j}+\mathbf{k}\right] \\ &= \left[0,\ \mathbf{i}+\sqrt{2}\mathbf{j}+\mathbf{k}\right] \end{aligned} \tag{2.46}$$

From equation 2.46 it is visible that the result is a pure quaternion, and the norm is

$$|p'| = \sqrt{1^2+\sqrt{2}^2+1^2} = \sqrt{4} = 2, \tag{2.47}$$

which means that the norm has been maintained. The result is visualized in figure 2.6.

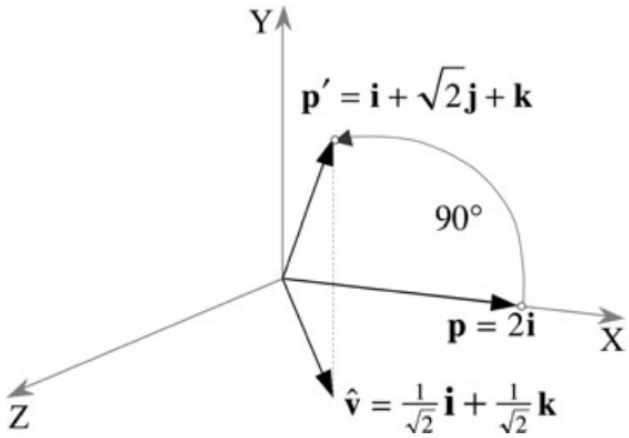


Figure 2.6: The vector $2\mathbf{i}$ is rotated 90° to $\mathbf{i} + \sqrt{2}\mathbf{j} + \mathbf{k}$ [4]

As seen on figure 2.6, the vector is rotated 90° rather than 45°, i.e. twice as much as intended. The conclusion is that in order to rotate a vector $\mathbf{p}$ by an angle θ about an arbitrary axis $\hat{\mathbf{v}}$, the quaternion to be used for rotation should consider the half of the desired angle:

$$q = \left[\cos\frac{1}{2}\theta,\ \sin\frac{1}{2}\theta\hat{\mathbf{v}}\right] \tag{2.48}$$

Equation 2.48 is the general form of a rotation quaternion. Representing rotation using quaternions eliminates the possible problem of singularity because a quaternion is not a sequence of rotations, but only represents one rotation. It therefore requires a complete solution.

2.2.3 Conversion from quaternions and Euler angles

Considering the ZYX rotation sequence, quaternions can be converted to Euler angles. Assuming a normalized quaternion, the *discriminant* Δ is defined as:

$$\Delta = q_1 q_3 - q_2 q_4 \tag{2.49}$$

In most situations, $|\Delta| < \frac{1}{2}$. Then, equation 2.50 can solve for yaw(ψ), pitch(θ) and roll(ϕ) [28]:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} tan^{-1}\left(2\frac{q_1q_2+q_3q_4}{1-2(q_2^2+q_3^2)}\right) \\ sin^{-1}(2\Delta) \\ tan^{-1}\left(2\frac{q_1q_4+q_2q_3}{1-2(q_3^2+q_4^2)}\right) \end{bmatrix} \tag{2.50}$$

However, when $|\Delta| \approx \frac{1}{2}$, the solutions are approaching singularities. This happens when pitch approaches ±90°. These special cases can be solved as in equation 2.51:

$\Delta = -\frac{1}{2}$	$\Delta = \frac{1}{2}$
$\phi = 0$	$\phi = 0$
$\theta = -\frac{\pi}{2}$	$\theta = \frac{\pi}{2}$
$\psi = 2tan^{-1}(\frac{q_2}{q_1})$	$\psi = -2tan^{-1}(\frac{q_2}{q_1})$

(2.51)

2.2.4 Rotation matrices

There are three uses for a rotation matrix [2]:

- To represent a configuration of a rigid body
- To change the reference frame in which a vector or frame is represented
- To displace a vector or a frame.

The rotation matrices are defined as

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \tag{2.52}$$

$$R_y(\theta) = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \tag{2.53}$$

$$R_z(\psi) = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.54}$$

where R_x, R_y and R_z are the rotations about the x-, y- and z-axis respectively. These rotation matrices are known as the **special orthogonal group** (SO(3)). To get the total rotation from one matrix, these three rotation matrices can be combined by multiplying them:

$$R(\phi,\theta,\psi) = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi) \tag{2.55}$$

$$R(\phi,\theta,\psi) = \begin{bmatrix} c\theta c\psi & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{2.56}$$

The order of rotations in the matrix from equation 2.56 is:

1. Roll by ϕ
2. Pitch by θ
3. Yaw by ψ

2.2.5 Conversion from rotation matrix to Euler angles

Considering an ZYX rotation sequence, and an arbitrary rotation matrix in this form:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.57}$$

To extract the Euler angles from the rotation matrix, a singularity check must be performed [29]:

$$sy = \sqrt{r_{11}^2 + r_{21}^2} \tag{2.58}$$

If $sy \approx 0$ from equation 2.58, the system is close to a singularity (see section 2.2.1). If the system is **not** approaching singularity, this equation is used:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(r_{32},\ r_{33}) \\ atan2(-r_{31},\ sy) \\ atan2(r_{21},\ r_{11}) \end{bmatrix} \tag{2.59}$$

If the system **is** approaching singularity, this equation is applicable:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(-r_{23},\ r_{33}) \\ atan2(-r_{31},\ sy) \\ 0 \end{bmatrix} \tag{2.60}$$

2.2.6 Homogeneous transformation matrix for representation of position and orientation

The matrices to describe position and orientation, shown in equation 2.10, can be combined to a single homogeneous transformation matrix. The form of this matrix is:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.61}$$

where $R \in SO(3)$ and $p \in \Re^3$ is a column vector. In practical terms, matrix R is the rotation matrix from equation 2.56 and p is the position in x-, y- and z-axis, represented in relation to the reference frame [2]. An example with rotation along the z-axis, $R = R_z(\psi)$:

$$\begin{bmatrix} c\theta & -s\theta & 0 & p_x \\ s\theta & c\theta & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.62}$$

The Matrix from equation 2.62 can be altered slightly to just rotate or just translate. An example of pure rotation along z-axis without translation:

$$Rot(\hat{\omega},\theta) = \begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.63}$$

where the translation parts are set to 0. $\hat{\omega}$ denotes which axis should be rotated about, and θ represents the rotation angle. Similarly, pure translation can be achieved by replacing the R component in equation 2.61 with $I_{3\times3}$. An example of pure translation [2]:

$$Trans(p) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.64}$$

2.3 Statistical Theory

This section contains statistical theory that is applied in sensor fusion algorithms later in the report.

2.3.1 Population variance

Variance is a measure of variation. In probability theory, the variance is the expectation of the squared deviation of a random variable from its mean value. When the whole population of observations is known, the variance can be calculated by equation 2.65 [30].

$$\sigma^2 = \frac{\sum(x-\mu^2)}{n} \tag{2.65}$$

In equation 2.65, σ^2 is the population variance, x_1, x_2,..., x_n are the observations in the population of size n, and μ is the mean value of the observations which has the formula described in equation 2.66.

$$\mu = \frac{1}{n}\sum x \tag{2.66}$$

2.3.2 Covariance

Covariance is a measure of the connected variability of two random variables. The covariance between the two stochastic variables X and Y is defined in equation 2.67 [30].

$$cov(X,Y) = \sigma_{XY} = E[(X-\mu_x)(Y-\mu_y)] \tag{2.67}$$

2.3.3 Normal distribution

A normal distribution is a continuous probability distribution for a real-valued random variable. The normal distribution is often called the Gaussian distribution, named after the German mathematician Carl Friedrich Gauss (1777-1855) [31]. A normal distribution is often written in short form as $\mathcal{N}(\mu,\sigma^2)$, where σ^2 is the variance and μ is the mean value [5]. Equation 2.68 shows the probability density function of a normally distributed stochastic variable X [30].

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad for -\infty < x < \infty, \qquad \pi \approx 3.1416 \tag{2.68}$$

Figure 2.7 shows a normal distribution. This curve is often referred to as a Gauss curve. The curve has the following properties:

- It is bell formed and symmetric about a vertical line through μ.
- It has inflection points along the horizontal axis at $\mu-\sigma$ and $\mu+\sigma$.

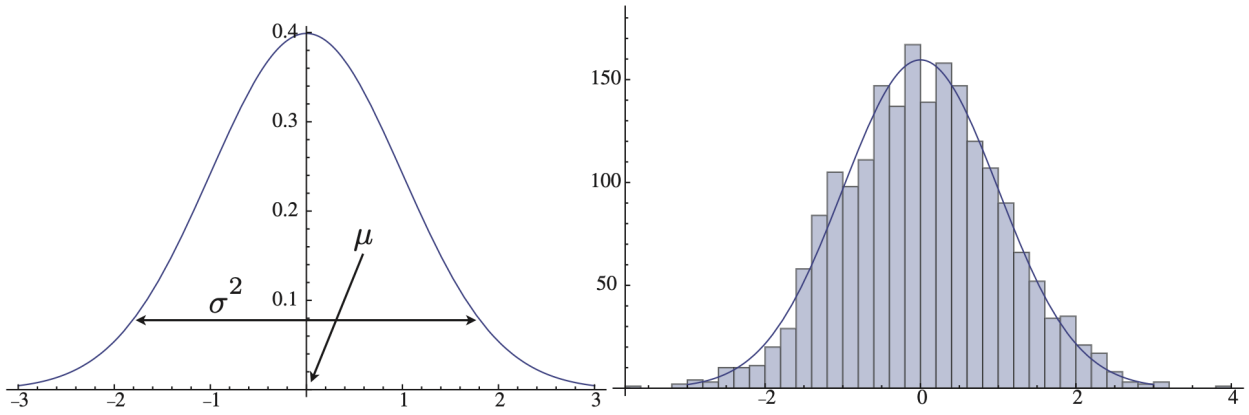


Figure 2.7: (Left) Normal distribution with $\mu = 0$ and $\sigma^2 = 1$. (Right) A bell curve approximating a data set which is not normalized. [5]

2.4 Sensors

A sensor is a device that measures a physical environment (such as heat, light, sound, pressure, magnetic field, or a particular motion) and transmits a resulting impulse [32]. Likewise, humans can sense heat, touch/force, visual observations, smell and taste. When an observation is made, one or multiple electrical impulses are sent to the brain where it is processed. Sensors enable robots and machines to have a similar ability to sense the environment and process observations in an MCU or computer. This section describes how a selection of sensor types applicable for position and orientation estimation work.

Measurement range

The range of a sensor describes the maximum physical value it can accurately measure. A sensor with a high range will not be able to measure small changes accurately. Conversely, a low range sensor will not be able to measure high values. For more information, see section 2.5.1.

Sensitivity

The sensitivity of a sensor describes the difference in the physical values based on the change in output voltage. For example, if a gyroscope sensor has a sensitivity of $10 mV/dps$ ($dps = °/s$),

and the measured voltage is $50mV$, then the sensor value is

$$Sensor\ Value = \frac{50mV}{10mV/dps} = 5dps \tag{2.69}$$

When the sensitivity increases, a higher voltage is required per sensor unit, and the range will also decrease [33].

Bias

Sensor bias is the steady-state error of a sensor reading. Sensor bias can also be considered an offset from the actual sensor value. For example, if a gyroscope is kept completely stationary without any motion, the expected sensor values is $0°/s$. If the measured value is $1.5°/s$, then the bias is $1.5°/s$.

Integration error (Drift)

If sensor readings are integrated (e.g. accelerometer readings are integrated to get velocity), the bias of the sensor reading will lead to an accumulation of error. This accumulation of integration error is often referred to as drift.

2.4.1 Gyroscope

A gyroscope is a sensor which measures angular velocity. A common and inexpensive type of gyroscope is the Coriolis vibratory gyroscope (CVG). This type of gyroscope measures angular velocity by measuring the Coriolis force on a vibrating object [34]. During a rotation, the Coriolis force will change the resonance direction of the vibrating ring inside the gyroscope, see figure 2.8.

Coriolis force

The Coriolis force is a fictitious force that acts on objects in motion within a reference frame that rotates with respect to an inertial frame [35].

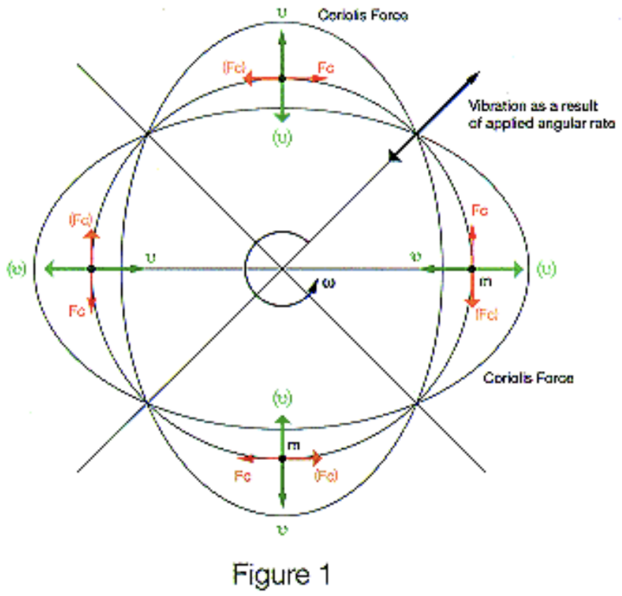


Figure 2.8: Visualizing the Coriolis force on the vibrating structure inside a CVG [6]

2.4.2 Magnetometer

A magnetometer is a device that measures the strength and optionally the direction of nearby magnetic fields. Magnetometers that only measures the total field strength is called **scalar magnetometers**, and the ones measuring direction is called **vector magnetometers** [36].

2.4.3 Accelerometer

An accelerometer is a device that can measure accelerations. The three main types of accelerometers are piezoelectric, piezoresistive and capacitive accelerometers.

Piezoelectric accelerometers use the piezoelectric effect to measure acceleration. Piezoelectric elements produces electricity when under physical stress.

Piezoresistive accelerometers work in a similar way. Instead of producing electricity when under physical stress, the electrical resistance increases proportionally to the stress. These accelerometers are less sensitive than its piezoelectric counterparts.

The third kind of accelerometers are capacitive accelerometers. These consist of stationary capacitor plates and capacitor plates mounted on a spring loaded, movable diaphragm. During movement, this diaphragm moves, and the distance between the capacitor plates changes,

resulting in a varying capacitance as seen in figure 2.9 [37].

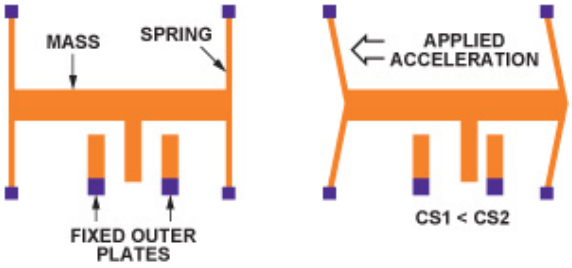


Figure 2.9: MEMS capacitive accelerometer [7]

2.4.4 IMU/MARG

An IMU (Inertial Measurement Unit) is a system consisting of a gyroscope and an accelerometer. They can also include a magnetometer. In that case, they are called MARG (Magnetic, Angular Rate, and Gravity) sensors, or 9-DOF IMUs. The raw measurements can be used to calculate attitude, linear velocity, angular rates, and position relative to a global frame. IMUs are often delivered on a MEMS, which includes software that processes the raw sensor data. The output can be Euler angles, filtered and calibrated acceleration and velocity data, among other things.

An IMU consisting of a 3-axis gyroscope and a 3-axis accelerometer is called a 6-DOF IMU, because it measures both angular velocities and angular accelerations in three dimensions each. This means that it measures six independent variables. When a 3-axis magnetometer is included, three degrees of freedom is added, which is why they are called 9-DOF IMUs in that case.

2.4.5 Radar Sensor

Radar sensors are used to detect both moving and stationary objects. They are often used in cars and other mobile machines for collision avoidance [38]. Radars use Frequency Modulated Continuous Waves (FMCW) to measure the distance to an object. By comparing the different frequencies, the velocity of the object can be calculated. A radar is able to detect and distinguish between small objects close to each other by using a high frequency carrier. The radar works by emitting a signal on a frequency that changes over time (sweeping time) and reading the signal that is returned after hitting an object (echoing signal). The frequency is given a minimum and

maximum value. This is called the bandwidth. The distance to any object is calculated by:

$$d = \frac{c \cdot T_s \cdot f_b}{2 \cdot B_{sweep}} \tag{2.70}$$

where c is the speed of light, f_b is the frequency sweep, T_s is the sweep time and B_{sweep} is the bandwidth [39].

2.4.6 Ultrasonic Sensor

An ultrasonic sensor measures distance using reflection of sound waves. The sensor consists of two main parts; emitter and detector. The emitter sends out ultrasonic sound waves, which cannot be heard by humans. When the sound waves hit an object, they are reflected back to the sensor and received by the detector. The distance is calculated by measuring the time between emitting and detecting the sound. The measured time is then divided by two to get the distance. This is because the measured time is the time the sound travels from the emitter to the object and then back to the detector. In other words, half of the traveled distance is the actual distance from the sensor to the object. The time is then multiplied by the speed of sound, giving the distance:

$$d = \frac{t \cdot v}{2} \tag{2.71}$$

where t is the time from emittance to receival, and v is the speed of sound.

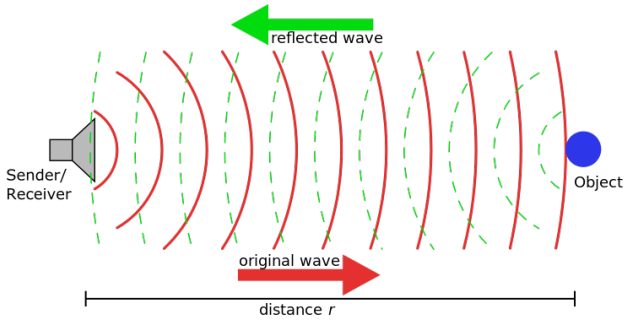


Figure 2.10: Ultrasonic sensor principle [8]

2.4.7 Infrared Sensor

An infrared distance sensor (IR sensor) uses light to measure the distance to an object. A beam of infrared light is emitted from an IR LED and reflected back to the sensor when it hits an object. The sensor measures the angle of the reflected light, and uses triangulation to calculate the distance to the object. See figure 2.11.

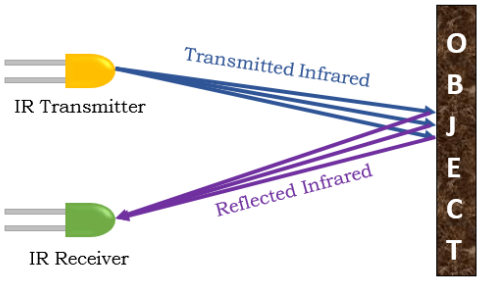


Figure 2.11: Infrared sensor principle [9]

2.4.8 Digital Camera

A digital camera is a device that utilizes a light-detector (commonly called an image sensor) to create a picture. It has a housing that protects all internal components as well as prevents light to reach the image sensor from unwanted directions. The housing has an aperture where the light is coming in to hit the image sensor. In front of the aperture there is a lens, which can be controlled to adjust the light coming in to get the desired focus in the picture. Figure 2.12 shows a general block diagram of how a digital camera converts the incoming light to a digital represented picture.

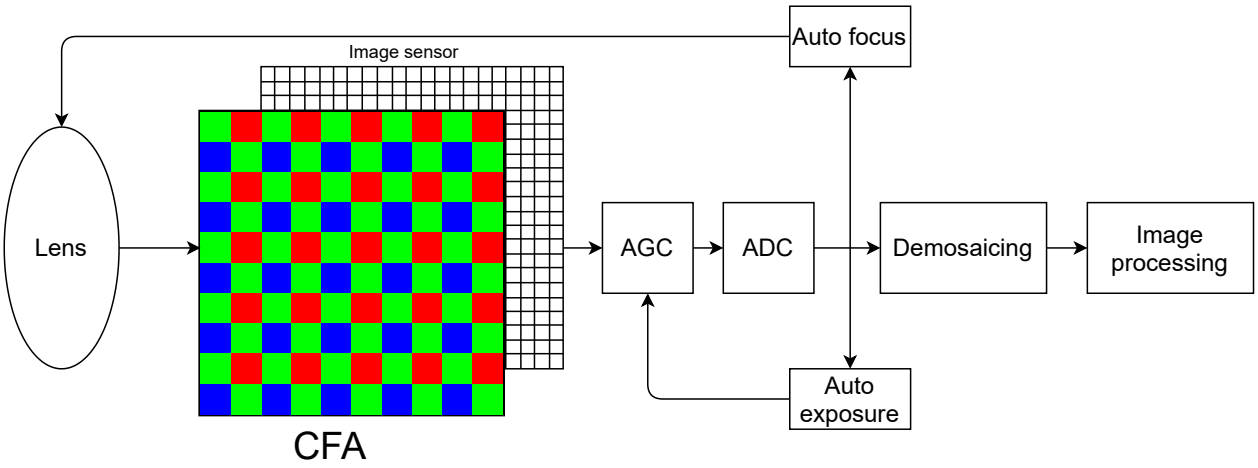


Figure 2.12: General block diagram of a digital camera

As seen in figure 2.12, the light travels through the lens and hits the color filter array (CFA). The color filter array is placed over the image sensor to capture the color information in the light. In the case of the figure, the CFA is a Bayer filter, which gives information about the intensity of light in red, green and blue (RGB) wavelength regions [40]. The signal from the image sensor is then transmitted to an automatic gain control (AGC), which amplifies the signal and transmits it to the analog-to-digital converter (ADC) (see section 2.5.1). The digital information from the ADC is then "demosaiced", which is a digital process that renders the image into a viewable format. From there, the digital picture can be processed further using different image processing techniques.

Image sensor

The image sensor in a digital camera can be of two types; either a CCD (charge-coupled device), or a CMOS (complementary metal oxide semiconductor) image sensor [41]. Both of these types converts light into electric charge an transmits it as electronic signals.

A CCD image sensor consists of an array of capacitors. Each of these capacitors carries an electric charge that corresponds to the light intensity of a pixel [42]. Each capacitor transfers its charge to its neighbor in the array, where the last capacitor in the array transmits all the content into a charge amplifier.

A CMOS image sensor consists of a photodiode and a CMOS transistor for each pixel. This allows each pixel signal to be amplified individually. An advantage of amplifying each pixel

signal individually is that it reduces the noise in the electrical signals that is converted from the captured light. Figure 2.13 shows a basic introduction of how the CCD and CMOS image sensors processes the pixels.

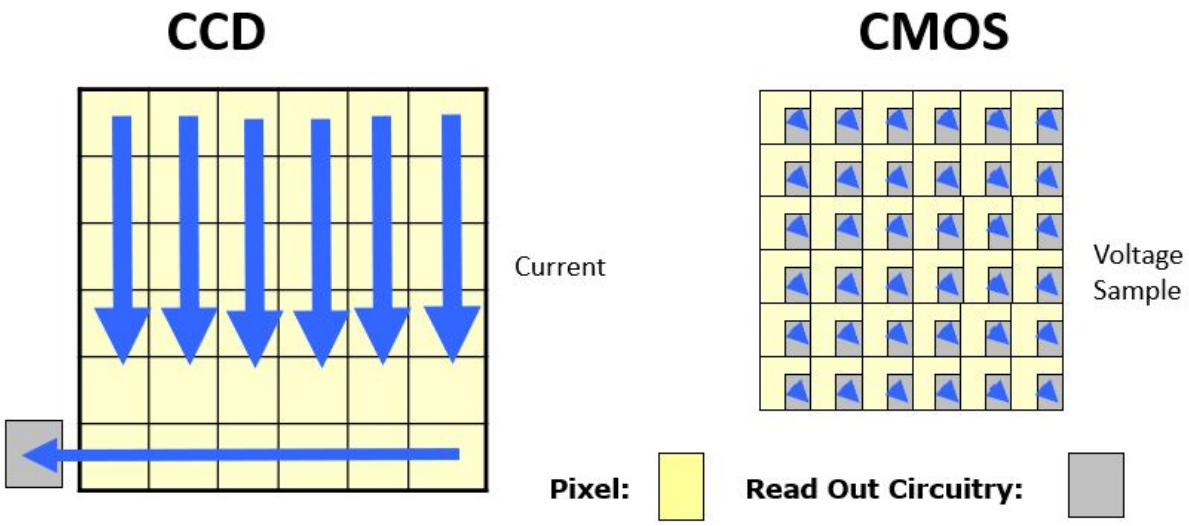


Figure 2.13: Figures showing how CCD and CMOS sensors work [10]

Field of view

The field of view (FOV) is defined as the maximum area of a sample that a camera can image [11]. It is determined by the focal length of the lens and the size of the image sensor. The focal length is the distance between the optical center of the lens and the image sensor [43]. The sensor size is determined by the size of the pixels, and the number of pixels. The angular field of view (AFOV) is the angle between light captured at the horizontal, and light captured at the edge. See figure 2.14.

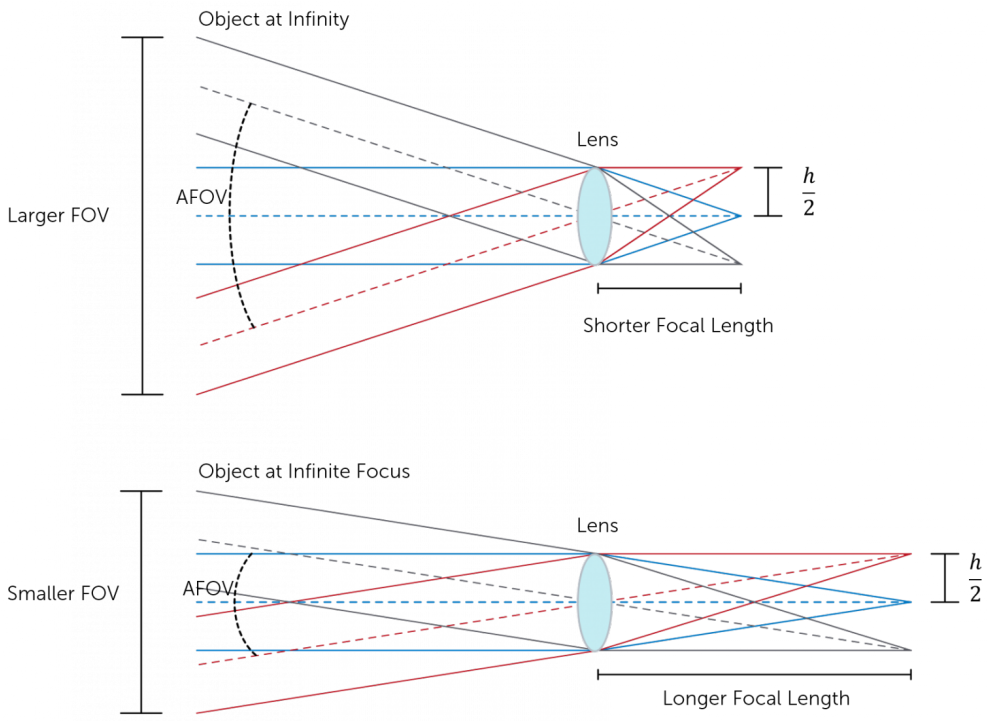


Figure 2.14: Schematics showing the relations between focal length, FOV and AFOV [11]

Resolution

The resolution of an image is given by the number of pixels in the image. The data in an image is stored as a matrix with rows and columns (x and y). Each cell in the matrix represents one pixel. Figure 2.15 shows some of the most common image resolutions.

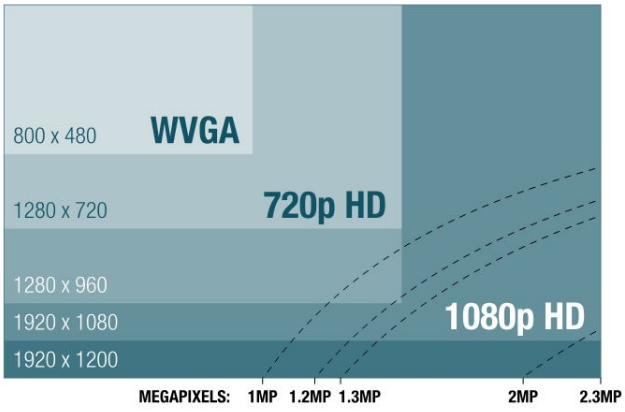


Figure 2.15: Common image resolutions [12]

Each pixel usually consists of one or three different values. When each pixel consists of one

value, the image is a gray scale image. Images with three values in each pixel is known as an RGB image, or a color image. Each of the values represents the strength of the red, green and blue components respectively. The combination of the values results in the color in the pixel.

2.4.9 ToF camera

A Time of Flight camera uses IR light to get the depth in an image. The sensor consist of two main components; an IR light emitting diode and a special photosensitive matrix. A pulse or FMCW of IR light is emitted from the camera, and the depth is calculated by using the speed of light and the time between emitting and detecting the light. The technology is quite similar to infrared sensors, but with the possibility to get the distance in a large area in contrast to a single point [44].

2.4.10 Stereo Camera

A stereo camera is a combination of two cameras pointed in the same direction. It works similarly to the human vision where the distance to an object can be determined by an objects displacement seen in one image compared to the other [45]. In other words, the same object will appear in two different locations within the images. To acquire depth from a stereo image, the displacement of the objects from one image to the next is utilized. Knowledge about camera resolution and distance between the two cameras are needed. Combined with the disparity in the images it is possible to know the depth of each pixel in the frame. It is often referred to as binocular disparity.

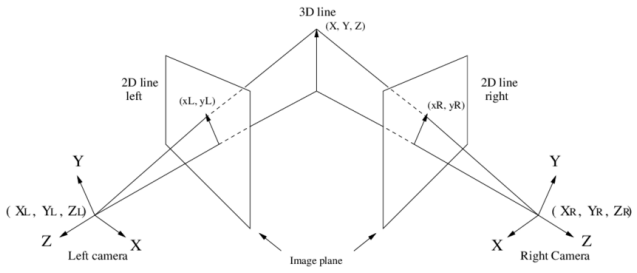


Figure 2.16: Stereo camera principle [13]

2.5 Sensor Fusion

When using electronic sensors to observe environmental information, there are several challenges. First of all, it is impossible for an electronic sensor to be completely accurate because it provides digital information to describe an analogue world. Because a sensor is limited to a specific physical range, which is mapped to a specific amount of bits, quantification error can occur (see section 2.5.1).

Electronic sensors are also subject to a phenomenon called white noise. White noise is random noise where the frequency and power spectrum is constant and independent of the sensors sampling frequency [46]. The consequence of this is that even if the physical property the sensor is measuring is held constant, the sensor will output values that is distorted by the white noise.

To tackle the problem of imperfect sensors, sensor fusion can be applied. Sensor fusion is a technique where two or several sensor outputs are combined "to produce enhanced data in form of an internal representation of the process environment [47]". Sensor fusion enables a more robust sensing system with improved resolution and increased confidence.

2.5.1 Analog-to-Digital conversion

Analog-to-digital converters converts a continuous, analog signal to discrete, digital values. However, ADCs have limitations. During the discretization of a signal, information can be lost. An ADC maps analog values to digital, sampled values. The *resolution* of an ADC determines how small increments in voltages the ADC manages to perceive as a change [48].

$$\frac{ADC\ resolution}{System\ voltage} = \frac{ADC\ reading}{Measured\ voltage} \tag{2.72}$$

As an example, a micro controller has a 10-bit ADC and a system voltage of 5V. 10-bit resolution translates to $2^{10} - 1 = 1023$ unique digital values. To find the ADC reading x in this case, where the analog voltage is for example 3.27V, equation 2.73 would be acquired by substituting the values in equation 2.72:

$$\begin{aligned} \frac{1023}{5.0V} &= \frac{x}{3.27V} \\ \frac{1023}{5.0V} \cdot 3.27V &= x \\ x &\approx 669 \end{aligned} \tag{2.73}$$

Because the sampled values are limited by the resolution of the ADC, continuous analog values must be limited within a specific range. As an example: A distance sensor measures in the range from 0-5V.

$$Q = \frac{5V}{1023\ LSB} \approx 0.00488V/LSB = 4.88mV/LSB \tag{2.74}$$

From equation 2.74, the change of 1 LSB (*Quantization step*) equals a change of 4.88mV. This exposes one limitation of digital quantization; the quantization error.

Quantization error is the difference between the actual physical value and the digital sampled value (equation 2.75) [49].

$$Quantization\ error = Quantization\ value - Actual\ value \tag{2.75}$$

For example, if the actual voltage is 1955mV, inserting in equation 2.72 yields:

$$\frac{1955mV}{4.88mV/LSB} = 400.61\ LSB \approx 401\ LSB \tag{2.76}$$

According to equation 2.76, the correct sampled value should be 401.

$$x = 401\ LSB \cdot 4.88mV/LSB = 1956.88mV \tag{2.77}$$

$$Quantization\ error = 1956.88mV - 1955mV = 1.88mV \tag{2.78}$$

As seen in equation 2.74, 2.77 and 2.78, the quantization error is almost half a quantization step. In a case where high accuracy is important, this might be unacceptable. An ADC with higher resolution will yield a lower quantization error at the same sensor range.

2.5.2 Gradient Descent

Gradient descent is an iterative first-order optimization algorithm. It is used to find the local minimum of a differentiable function by taking repeated steps in the opposite direction of the gradient at the current point. The general equation for gradient descent is shown in equation 2.79:

$$x_{n+1} = x_n - \gamma \nabla f(x_n) \tag{2.79}$$

where x_{n+1} is the next point, x_n is the current point, γ is the gain/learning rate and $\nabla f(x_n)$ is the gradient in the current point. The gradient descent will consider the slope of the gradient in the current point, and then move the next point in the opposite direction of the slope. As seen in figure 2.17, at point A the gradient is negative and the next point (B) is moved towards the positive direction of the horizontal axis [50].

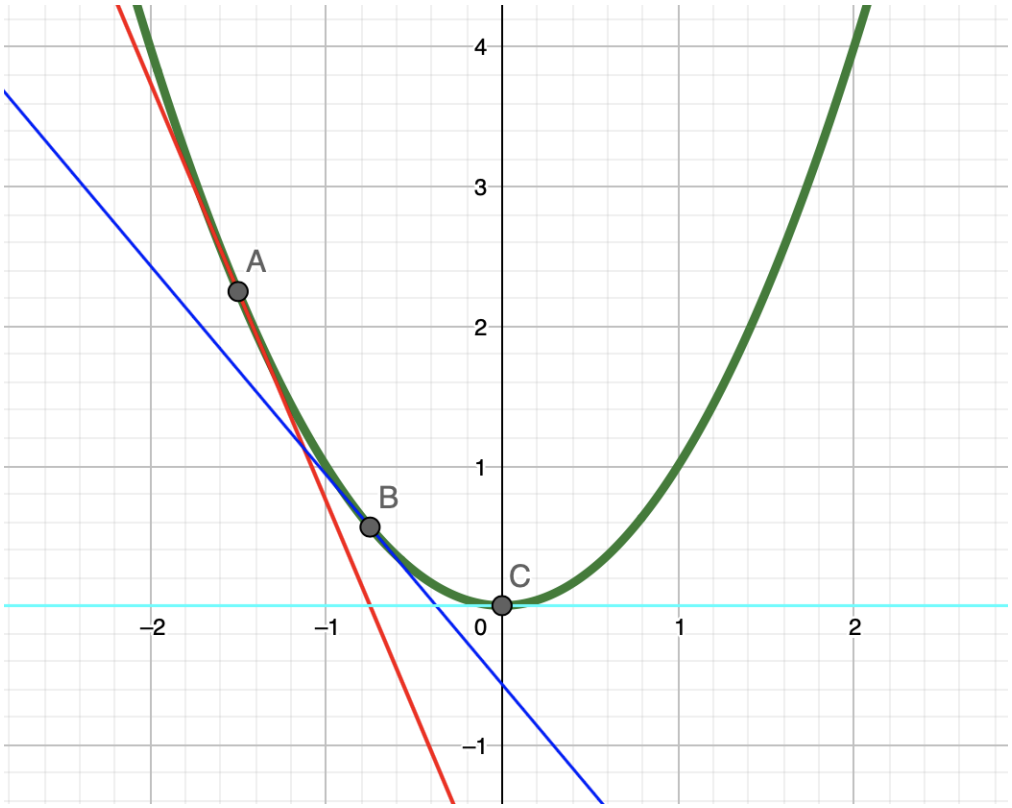


Figure 2.17: Gradient descent

Learning rate

The learning rate of a gradient descent algorithm determines how much the points should move in each iteration. Too high learning rate results in overshooting the local minimum (see figure 2.18) and too low learning rate results in a slow convergence (see figure 2.19). The optimal learning rate results in a fast convergence without overshooting.

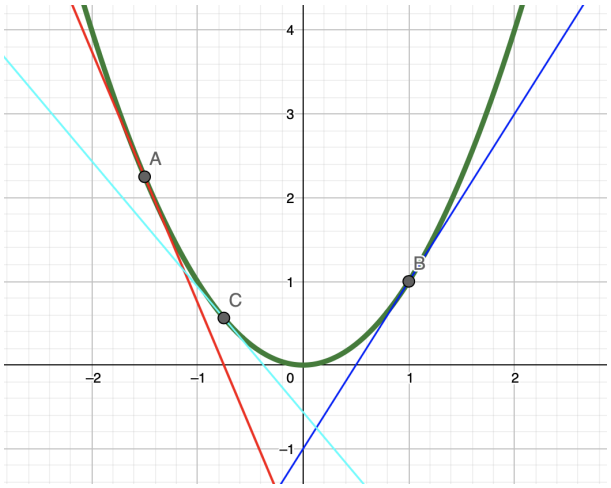


Figure 2.18: Too high learning rate, overshooting

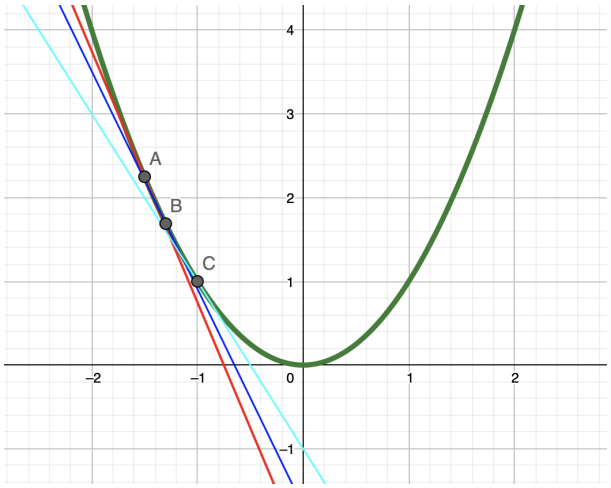


Figure 2.19: Too low learning rate, slow convergence

2.5.3 Jacobian matrix and determinant

The Jacobian matrix is the matrix of all the first-order partial derivatives of a vector function. Jacobians are used for transformations from one coordinate system to another. This allows for linear approximations at and near individual points of a non-linear vector function. The general form of a $m \times n$ Jacobian matrix is shown in equation 2.80 [51]:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{2.80}$$

If $m = n$, the Jacobian matrix is a square matrix. Thus, the Jacobian Determinant (also called the Jacobian), can be found. Equation 2.82 shows the general equation for a $\Re^2 \rightarrow \Re^2$ transformation.

$$F\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} \tag{2.81}$$

$$J(f_1,f_2) = \frac{\partial(f_1,f_2)}{\partial(x,y)} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} \tag{2.82}$$

As an example, $f_1(x,y)$ and $f_2(x,y)$ is:

$$\begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} = \begin{bmatrix} x+sin(y) \\ y+sin(x) \end{bmatrix} \tag{2.83}$$

Inserting the values for f_1 and f_2 from equation 2.83 in equation 2.82 yields equation 2.84:

$$J(f_1,f_2) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & cos(y) \\ cos(x) & 1 \end{bmatrix} \tag{2.84}$$

2.5.4 Kalman Filter

The Kalman filter was invented in 1960 by the Hungarian-American electrical engineer and mathematician Rudolf Emil Kálmán [24]. The filter has many applications in technology, such as applications for guidance, navigation, and dynamic positioning on ships.

A Kalman filter is a set of mathematical equations that is used to efficiently compute an estimation of the state of a process, while minimizing the mean of the squared error [52]. The filter does this by utilizing a mathematical model of how the physical process behaves. The filter can estimate both past, present and future states, given that the behavior of the physical process is modeled accurately enough.

The Kalman filter is a recursive algorithm which essentially consists of two steps; *predict* and *update*. The predict-step estimates the state at time **k** based on the state at time **k-1**, along with values representing how much uncertainty there are in the estimated state. It does this by utilizing the mathematical model of how the system's state changes over time. The update-step updates the state based on the weighted average of the predicted state and the measured state. The weighting of the states is based on the uncertainty in the state, where higher uncertainty

yields lower weight.

The estimated state x_k and measured state y_k is described in equation 2.85. The random variables q_{k-1} and r_k represent the process and measurement noise, respectively. These random variables are assumed to be normally distributed white noise, independent of each other.

$$\begin{aligned} x_k &= A_{k-1}x_{k-1} + q_{k-1}, \qquad p(q) \sim \mathcal{N}(0,\ Q_{k-1}) \\ y_k &= H_k x_k + r_k, \qquad p(r) \sim \mathcal{N}(0,\ R_k) \end{aligned} \tag{2.85}$$

Q and R represent the process noise covariance matrix and the measurement noise covariance matrix, respectively. These matrices might change with each time step, and they contain information on how the noise in the variables in the states vary in relation to each other.

The process of estimating the state is as follows:

- Predict

  1. $\hat{x}_{k|k-1} = A_{k-1}\hat{x}_{k-1|k-1}$
  2. $P_{k|k-1} = A_{k-1}P_{k-1|k-1}A_{k-1}^T + Q_{k-1}$

- Update

  3. $v_k = y_k - H\hat{x}_{k|k-1}$
  4. $S_k = HP_{k|k-1}H^T + R_k$
  5. $K_k = P_{k|k-1}H^T S_k^{-1}$
  6. $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k v_k$
  7. $P_{k|k} = P_{k|k-1} - K_k S_k K_k^T$

Following these steps, the Kalman filter will compute the posterior distribution of the state x_k at every time-step. The *predict* equations are responsible for projecting the current state and error covariance estimates forward in time to obtain the *a priori* estimates of the next time step. The *update* equations are responsible for the feedback. This means that it incorporates a new measurement into the *a priori* estimate to obtain an improved *a posteriori* estimate [52]. The equations are described in detail below.

1. The predicted mean is acquired by taking the past posterior mean and multiplying it with the matrix A_{k-1}. The matrix A is the mathematical description of how the state evolves over time.

2. The predicted covariance $P_{k|k-1}$ is calculated by multiplying the past posterior covariance $P_{k-1|k-1}$ with A_{k-1} twice, and adding Q_{k-1}, which describes the uncertainty in the model.

3. v_k is called the measurement innovation. This equation can be seen as a representation of the new information that is gained when comparing the current measurement with the predicted measurement. The matrix H is used to relate the state to the measurement y_k. If v_k is zero, it means that the predicted measurement and the actual measurement are in complete agreement.

4. S_k represents the predicted measurement covariance. R_k represents the uncertainty (variance) in the measurement, and S_k combines this uncertainty with the uncertainty of the predicted state.

5. K_k is called the Kalman gain. The Kalman gain gives information about how much the predicted state and predicted covariance should be adjusted, using the new information gained from the measurement covariance.

6. The posterior mean, $x_{k|k}$, is computed by taking the predicted mean and adjusting it using the Kalman gain, K_k, and the measurement innovation, v_k. As seen in the equation, the Kalman gain is used as a scaling factor to determine how much of the measurement innovation should be added.

7. The posterior covariance, $P_{k|k}$ is calculated by taking the predicted covariance and adjusting it with the new information gained from the measurement. As seen in the equation, the new information is subtracted from the predicted covariance. This is because the new information helps to decrease the uncertainty regarding the state.

2.5.5 Madgwick Filter

The Madgwick filter was invented by Sebastian O.H. Madgwick and was presented in his report "An efficient orientation filter for inertial and inertial/magnetic sensor arrays" in 2010 [14]. The Madgwick filter is a computational inexpensive orientation filter that utilizes the combination of a tri-axis gyroscope and a tri-axis accelerometer. It can also utilize sensor input from a tri-axis magnetometer, but that is optional. The Madgwick filter represents orientation as a quaternion, which allows data from accelerometers and magnetometers "to be used in an analytically derived and optimised gradient-descent algorithm to compute the direction of the gyroscope measurement error as a quaternion derivative [14]".

Orientation from angular rate

The orientation of the Earth frame relative to the sensor frame at time t can be defined as ${}^{S}_{E}\mathbf{q}_{\omega,t} = [q_w \quad q_x \quad q_y \quad q_z]$. The measured angular rates from the gyroscope can be arranged into the vector ${}^{S}\boldsymbol{\omega} = [0 \quad \omega_x \quad \omega_y \quad \omega_z]$. The quaternion derivative describing the rate of change of the orientation of the Earth frame relative to the sensor frame can be calculated as ${}^{S}_{E}\dot{\mathbf{q}} = \frac{1}{2}\,{}^{S}_{E}\hat{\mathbf{q}} \otimes {}^{S}\boldsymbol{\omega}$. Using these definitions, the orientation of the Earth frame relative to the sensor frame at time t can be calculated by equations 2.86 and 2.87. In these equations, Δt is the sampling period, ${}^{S}\boldsymbol{\omega}_t$ is the measured angular rate at time t, and ${}^{S}_{E}\hat{\mathbf{q}}_{est,t-1}$ is the previous estimated orientation. The subscript ω is used to indicate that the quaternion is calculated from the angular rate.

$$ {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t} = \frac{1}{2}\,{}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} \otimes {}^{S}\boldsymbol{\omega}_t \tag{2.86} $$

$$ {}^{S}_{E}\mathbf{q}_{\omega,t} = {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} + {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t}\Delta t \tag{2.87} $$

Orientation as a solution of gradient descent

If the direction of Earth's field (magnetic or gravitational) is known in the Earth frame, measuring the respective field in the sensor frame will make it possible to calculate the orientation of the sensor frame relative to the Earth frame. However, for any given measurement, there will not be a unique solution to the sensor orientation. Instead, there will be an infinite amount of solutions represented by all orientations achieved by the rotation of the true orientation around

an axis parallel to the field [14]. A quaternion representation requires a complete solution to the orientation to be found. Madgwick proposes to achieve this by formulating an optimization problem where an orientation of the sensor, ${}^{S}_{E}\hat{\mathbf{q}}$, aligns a predefined reference direction of the field in the Earth frame, ${}^{S}\hat{\mathbf{d}}$, with the measured direction of the field in the sensor frame, ${}^{S}\hat{\mathbf{s}}$. This is done using quaternion rotation operation and equation 2.88 shows the objective function to be minimized. In other words, ${}^{S}_{E}\hat{\mathbf{q}}$ can be found as the solution to $\min_{{}^{S}_{E}\hat{\mathbf{q}}\in\Re^4} f({}^{S}_{E}\hat{\mathbf{q}}, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}})$. Equations 2.89, 2.90 and 2.91 defines the components of each vector.

$$f({}^{S}_{E}\hat{\mathbf{q}}, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}}) = {}^{S}_{E}\hat{\mathbf{q}}^{*} \otimes {}^{E}\hat{\mathbf{d}} \otimes {}^{S}_{E}\hat{\mathbf{q}} - {}^{S}\hat{\mathbf{s}} \tag{2.88}$$

$${}^{S}_{E}\hat{\mathbf{q}} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} \tag{2.89}$$

$${}^{E}\hat{\mathbf{d}} = \begin{bmatrix} 0 & d_x & d_y & d_z \end{bmatrix} \tag{2.90}$$

$${}^{S}\hat{\mathbf{s}} = \begin{bmatrix} 0 & s_x & s_y & s_z \end{bmatrix} \tag{2.91}$$

In his report, Madgwick indicates that several optimization algorithms can be used to solve the optimization problem, but the gradient descent algorithm is both simple to implement and to compute. Gradient descent is therefore used to solve the optimization problem in the Madgwick filter. Equation 2.92 yields an orientation estimation of ${}^{S}_{E}\hat{\mathbf{q}}_{n+1}$ using the gradient descent algorithm for n iterations, based on an "initial guess" orientation, ${}^{S}_{E}\hat{\mathbf{q}}_0$, and a step-size, μ. The gradient of the solution is defined by the objective function (equation 2.94) and its Jacobian (equation 2.95) as seen in equation 2.93.

$${}^{S}_{E}\mathbf{q}_{k+1} = {}^{S}_{E}\hat{\mathbf{q}}_k - \mu \frac{\nabla f({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}})}{\|\nabla f({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}})\|}, \qquad k = 0, 1, 2, ..., n \tag{2.92}$$

$$\nabla f({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}}) = J^{T}({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}) f({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}}) \tag{2.93}$$

$$f({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}, {}^{S}\hat{\mathbf{s}}) = \begin{bmatrix} 2d_x(\frac{1}{2} - q_3^2 - q_4^2) + 2d_y(q_1q_4 + q_2q_3) + 2d_z(q_2q_4 - q_1q_3) - s_x \\ 2d_x(q_2q_3 - q_1q_4) + 2d_y(\frac{1}{2} - q_2^2 - q_4^2) + 2d_z(q_1q_2 + q_3q_4) - s_y \\ 2d_x(q_1q_3 + q_2q_4) + 2d_y(q_3q_4 - q_1q_2) + 2d_z(\frac{1}{2} - q_2^2 - q_3^2) - s_z \end{bmatrix} \tag{2.94}$$

$$J({}^{S}_{E}\hat{\mathbf{q}}_k, {}^{E}\hat{\mathbf{d}}) =$$

$$\begin{bmatrix} 2d_yq_4 - 2d_zq_3 & 2d_yq_3 + 2d_zq_4 & -4d_xq_3 + 2d_yq_2 - 2d_zq_1 & -4d_xq_4 + 2d_yq_1 + 2d_zq_2 \\ -2d_xq_4 + 2d_zq_2 & 2d_xq_3 - 4d_yq_2 + 2d_zq_1 & 2d_xq_2 + 2d_zq_4 & -2d_xq_1 - 4d_yq_4 + 2d_zq_3 \\ 2d_xq_3 - 2d_yq_2 & 2d_xq_4 - 2d_yq_1 - 4d_zq_2 & 2d_xq_1 + 2d_yq_4 - 4d_zq_3 & 2d_xq_2 + 2d_yq_3 \end{bmatrix} \tag{2.95}$$

Equations 2.92 - 2.95 is a description of the general form of the algorithm which can be applied to a field which is predefined in any direction. However, to simplify the equations, the direction of the field can be assumed to only have components within one of the principle axes of the global coordinate frame. This is done by assuming that the direction of gravity defines the vertical z-axis as shown in equation 2.96. Equation 2.97 defines the components of the normalized accelerometer measurement vector. Substituting ${}^{E}\hat{\mathbf{g}}$ for ${}^{E}\hat{\mathbf{d}}$ and ${}^{S}\hat{\mathbf{a}}$ for ${}^{S}\hat{\mathbf{s}}$ in the equations above, yields a simplified objective function (equation 2.98) and its Jacobian (equation 2.99).

$${}^{E}\hat{\mathbf{g}} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.96}$$

$${}^{S}\hat{\mathbf{a}} = \begin{bmatrix} 0 & a_x & a_y & a_z \end{bmatrix} \tag{2.97}$$

$$f_g({}^{S}_{E}\hat{\mathbf{q}}, {}^{S}\hat{\mathbf{a}}) = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - a_x \\ 2(q_1q_2 + q_3q_4) - a_y \\ 2(\frac{1}{2} - q_2^2 - q_3^2) - a_z \end{bmatrix} \tag{2.98}$$

$$J_g({}^S_E\hat{\mathbf{q}}) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{2.99}$$

If Earth's magnetic field is considered, an assumption can be made regarding the components of the magnetic field. The reference magnetic field ${}^E\hat{\mathbf{b}} = [0 \quad b_x \quad b_y \quad b_z]$ has components along three axes. However, in his report, Madgwick assumes the east component of this field is negligible, reducing the vector to only have two components (see equation 2.100). Equation 2.101 defines the components of the normalized magnetometer measurement vector. By substituting ${}^E\hat{\mathbf{b}}$ for ${}^E\hat{\mathbf{d}}$ and ${}^S\hat{\mathbf{m}}$ for ${}^S\hat{\mathbf{s}}$, a new object function and its Jacobian is formed, as seen in equations 2.102 and 2.103.

$${}^E\hat{\mathbf{b}} = [0 \quad b_x \quad 0 \quad b_z] \tag{2.100}$$

$${}^S\hat{\mathbf{m}} = [0 \quad m_x \quad m_y \quad m_x] \tag{2.101}$$

$$f_b({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) = \begin{bmatrix} 2b_x(\frac{1}{2} - q_3^2 - q_4^2) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z(\frac{1}{2} - q_2^2 - q_3^2) - m_z \end{bmatrix} \tag{2.102}$$

$$J_b({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}) = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix} \tag{2.103}$$

To provide a unique orientation of the sensor, the measurement reference directions of both fields (gravitational and magnetic) has to be combined. This is shown in equations 2.104 and 2.105. The solution surface of the objective function in equation 2.104 will have a minimum defined by a single point, provided that $b_x \neq 0$ [14]. In contrast, the solution surface of the objective functions in equations 2.98 and 2.102 alone, will have a minimum defined by a line,

and not a single point.

$$f_{g,b}({}^S_E\hat{\mathbf{q}}, {}^S\hat{\mathbf{a}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) = \begin{bmatrix} f_g({}^S_E\hat{\mathbf{q}}, {}^S\hat{\mathbf{a}}) \\ f_b({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}) \end{bmatrix} \tag{2.104}$$

$$J_{g,b}({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}) = \begin{bmatrix} J_g^T({}^S_E\hat{\mathbf{q}}) \\ J_b^T({}^S_E\hat{\mathbf{q}}, {}^E\hat{\mathbf{b}}) \end{bmatrix} \tag{2.105}$$

Normally, an approach to the optimization problem would be to compute multiple iterations of equation 2.92 for each new orientation and corresponding sensor measurement. The step-size μ would also need to be adjusted to an optimal value for each iteration. However, this will make the algorithm significantly heavier to compute, and regarding the Madgwick filter and its field of use, this is not necessary. For the Madgwick filter, one iteration per time sample can be computed as long as the convergence rate governed by μ_t is equal to or greater than the physical rate of change regarding the orientation [14]. This results in equation 2.106, where the estimated orientation, ${}^S_E\mathbf{q}_{\nabla,t}$, is calculated at time t based on the previous estimated orientation, ${}^S_E\hat{\mathbf{q}}_{est,t-1}$, and the gradient of the objective function, ∇f. The sub-script ∇ indicates that the gradient descent algorithm is used to calculate the quaternion. As seen in equation 2.107, ∇f is chosen according to the sensors used, and can be defined only by the measurements from an accelerometer (${}^S\hat{\mathbf{a}}_t$) sampled at time t, or both the measurements from an accelerometer (${}^S\hat{\mathbf{a}}_t$) and a magnetometer (${}^S\hat{\mathbf{m}}_t$).

$${}^S_E\mathbf{q}_{\nabla,t} = {}^S_E\hat{\mathbf{q}}_{est,t-1} - \mu_t \frac{\nabla f}{\|\nabla f\|} \tag{2.106}$$

$$\nabla f = \begin{cases} J_g^T({}^S_E\hat{\mathbf{q}}_{est,t-1}) f_g({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^S\hat{\mathbf{a}}_t) \\ J_{g,b}^T({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^E\hat{\mathbf{b}}) f_{g,b}({}^S_E\hat{\mathbf{q}}_{est,t-1}, {}^S\hat{\mathbf{a}}_t, {}^E\hat{\mathbf{b}}, {}^S\hat{\mathbf{m}}_t) \end{cases} \tag{2.107}$$

Madgwick defines the optimal value of the step-size μ_t as the value which ensures that the convergence rate of ${}^S_E\mathbf{q}_{\nabla,t}$ is limited to the physical orientation rate. This avoids overshooting which can happen if the step-size is too large (see section 2.5.2). Equation 2.108 shows how the step-size can be calculated. In this equation, Δt is the sampling period, ${}^S_E\dot{\mathbf{q}}_{\omega,t}$ is the physical orientation rate measured by a gyroscope, and α is an augmentation of the step-size to take

noise in the accelerometer and/or magnetometer measurements into account.

$$\mu_t = \alpha \| {}^S_E\dot{\mathbf{q}}_{\omega,t} \| \Delta t, \quad \alpha > 1 \tag{2.108}$$

Filter fusion algorithm

To estimate the orientation of the sensor frame relative to the Earth frame, ${}^S_E\mathbf{q}_{est,t}$, the orientation calculations ${}^S_E\mathbf{q}_{\omega,t}$ (from equation 2.87) and ${}^S_E\mathbf{q}_{\nabla,t}$ (from equation 2.106) can be fused together. This is described in equation 2.109, where γ_t defines the weighting of the two orientation calculations.

$${}^S_E\mathbf{q}_{est,t} = \gamma_t {}^S_E\mathbf{q}_{\nabla,t} + (1-\gamma_t) {}^S_E\mathbf{q}_{\omega,t}, \quad 0 \leq \gamma_t \leq 1 \tag{2.109}$$

In Madgwick's report, the optimal value of γ_t is defined as the value which ensures the weighted divergence of ${}^S_E\mathbf{q}_{\omega,t}$ to be equal to the weighted convergence of ${}^S_E\mathbf{q}_{\nabla,t}$. This is shown in equation 2.110. Here, $\frac{\mu_t}{\Delta t}$ is the convergence rate of ${}^S_E\mathbf{q}_{\nabla,t}$ and β is the divergence rate of ${}^S_E\mathbf{q}_{\omega,t}$.

$$\begin{gathered} (1-\gamma_t)\beta = \gamma_t \frac{\mu_t}{\Delta t} \\ \Downarrow \\ \gamma_t = \frac{\beta}{\frac{\mu_t}{\Delta t} + \beta} \end{gathered} \tag{2.110}$$

Equations 2.109 and 2.110 ensures the optimal fusion of ${}^S_E\mathbf{q}_{\nabla,t}$ and ${}^S_E\mathbf{q}_{\omega,t}$. This is based on the assumption that the convergence rate of ${}^S_E\mathbf{q}_{\nabla,t}$, which is governed by α as seen in equation 2.108, is equal or greater than the physical rate of change of the orientation. That is why α has no upper limitation. By assuming that α is very large, the step-size μ_t will become very large. Consequently, equation 2.106 can be simplified to equation 2.111 because a large value of μ_t will make ${}^S_E\hat{\mathbf{q}}_{est,t}$ negligible.

$${}^S_E\mathbf{q}_{\nabla,t} \approx -\mu_t \frac{\nabla f}{\|\nabla f\|} \tag{2.111}$$

Equation 2.110 can also be simplified to equation 2.112 since the β in the denominator becomes negligible due to the assumption of a large μ_t. From this, it is also possible to assume

that $\gamma_t \approx 0$ since the denominator will be significantly larger than the numerator.

$$\gamma_t \approx \frac{\beta \Delta t}{\mu_t} \tag{2.112}$$

By substituting equations 2.87, 2.111 and 2.112 into equation 2.109, equation 2.109 can be rewritten as equation 2.113.

$${}^{S}_{E}\mathbf{q}_{est,t} = \frac{\beta \Delta t}{\mu_t}\left(-\mu_t \frac{\nabla f}{\|\nabla f\|}\right) + (1-0)({}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} + {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t}\Delta t) \tag{2.113}$$

If ${}^{S}_{E}\dot{\mathbf{q}}_{est,t}$ is the estimated rate of change of the orientation (see equation 2.114) and ${}^{S}_{E}\hat{\dot{\mathbf{q}}}_{\epsilon,t}$ is the direction of the error of ${}^{S}_{E}\dot{\mathbf{q}}_{est,t}$ (see equation 2.115), equation 2.113 can be simplified to equation 2.116.

$${}^{S}_{E}\dot{\mathbf{q}}_{est,t} = {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t} - \beta\, {}^{S}_{E}\hat{\dot{\mathbf{q}}}_{\epsilon,t} \tag{2.114}$$

$${}^{S}_{E}\hat{\dot{\mathbf{q}}}_{\epsilon,t} = \frac{\nabla f}{\|\nabla f\|} \tag{2.115}$$

$${}^{S}_{E}\mathbf{q}_{est,t} = {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} + {}^{S}_{E}\dot{\mathbf{q}}_{est,t}\Delta t \tag{2.116}$$

In summary, the Madgwick filter calculates the orientation ${}^{S}_{E}\mathbf{q}_{est,t}$ by numerical integration of the estimated orientation rate ${}^{S}_{E}\dot{\mathbf{q}}_{est,t}$. The estimated orientation rate ${}^{S}_{E}\dot{\mathbf{q}}_{est,t}$ is computed as the rate of change of the orientation measured by the gyroscope, ${}^{S}_{E}\dot{\mathbf{q}}_{\omega,t}$, with the magnitude of the gyroscope measurement error, β, removed in the direction of the estimated error, ${}^{S}_{E}\hat{\dot{\mathbf{q}}}_{\epsilon,t}$, which is computed from accelerometer and magnetometer measurements. Figure 2.20 shows a block diagram representation of how the Madgwick filter can utilize gyroscope and accelerometer measurements to estimate orientation.

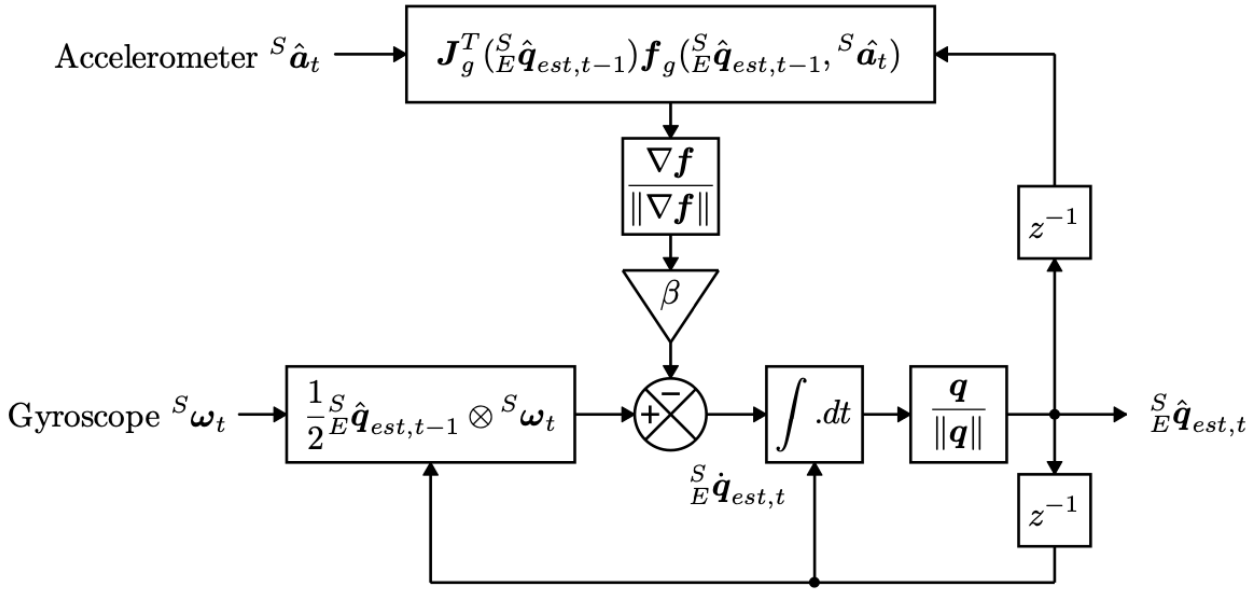


Figure 2.20: Block diagram from Madgwick's original report, representing a complete orientation filter using a gyroscope and an accelerometer [14]

Magnetic distortion compensation

When including magnetometer measurements in the Madgwick filter, the measured direction of Earth's magnetic field in the Earth frame at time t, ${}^{E}\hat{\mathbf{h}}_t$, can be computed as the normalized magnetometer measurement, ${}^{S}\hat{\mathbf{m}}_t$, rotated by the orientation of the sensor computed in the previous estimation, ${}^{S}_{E}\hat{\mathbf{q}}_{est,t-1}$. See equation 2.117.

$$ {}^{E}\hat{\mathbf{h}}_t = [0 \quad h_x \quad h_y \quad h_z] = {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} \otimes {}^{S}\hat{\mathbf{m}}_t \otimes {}^{S}_{E}\hat{\mathbf{q}}^{*}_{est,t-1} \tag{2.117} $$

Error in the inclination of the measured direction of Earth's magnetic field (${}^{E}\hat{\mathbf{h}}_t$) can be corrected if the filter's reference direction of the geomagnetic field (${}^{E}\hat{\mathbf{b}}_t$) has the same inclination. This is achieved by computing the direction of the geomagnetic field as the measured direction of Earth's magnetic field normalized to only have components in the Earth frame's x- and z-axes. See equation 2.118.

$$ {}^{E}\hat{\mathbf{b}}_t = \begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix} \tag{2.118} $$

According to Madgwick, by substituting equation 2.118 for equation 2.100, magnetic distortions is compensated for in a way that ensures that these kind of disturbances are limited to only affect the estimated heading component of orientation. Also, the need of a predefined reference direction of the Earth's magnetic field is eliminated [14]. Figure 2.21 shows a block diagram representation of how the Madgwick filter can estimate orientation utilizing an accelerometer, a gyroscope and a magnetometer, while compensating for magnetic distortion.

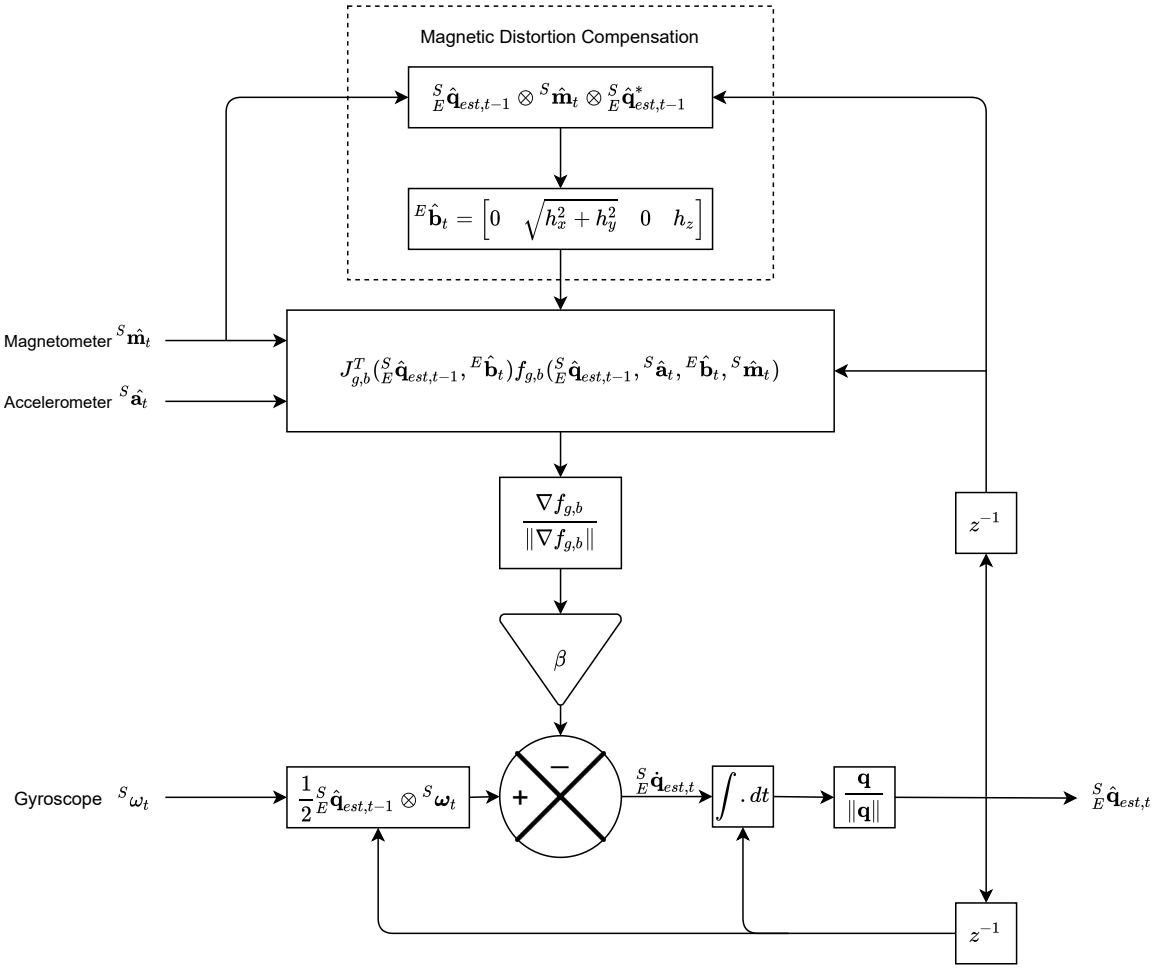


Figure 2.21: Block diagram representing a complete orientation filter using a gyroscope, accelerometer and magnetometer, including magnetic distortion compensation

Filter gain

The filter gain β represents all mean zero gyroscope errors. It is expressed as the magnitude of a quaternion derivative, and is defined using the angular velocity, as seen in equation 2.119. In this equation, $\bar{\omega}_\beta$ is the estimated mean zero gyroscope measurement error of each axis. It can

be computed by logging the measurements from the gyroscope for a chosen amount of time while the gyroscope is held still, and then calculating the mean of the measurements of each axis.

$$\beta = \sqrt{\frac{3}{4}}\bar{\omega}_\beta \tag{2.119}$$

2.6 Image processing

Image processing is a way to perform enhancement, adjust, resize and filter images. It can also be used to recognize colors, shapes and even human faces in an image. Image processing is performed on a single image at a time, but by sequentially processing multiple images, it is possible to determine displacement or movement of an object by comparing images. In other words, when it comes to image processing the input is an image and the output may be an enhanced image, gray scaled image, or characteristics found in the image [53].

2.6.1 RGB vs HSV

RGB (Red, Green, Blue) [54] is one form of color representation. Is it usually described with 8-bit values in a 3-tuple. Each of the values describes the intensity of the colors red, green and blue. The combination of these three values, ranging from 0-255 gives a total of $256^3 \approx 16.8 million$ different colors.

HSV/HSB (Hue, Saturation, Value/Brightness) [55] also uses a 3-tuple to describe colors, but unlike RGB it does not use a combination of primary color intensities in the representation. Hue is the color value. It ranges from 0-360 degrees. Saturation is the second element and can be considered as the purity of the color [56]. It ranges from 0-100%, where a low value gives a gray color. The last element is the Value or Brightness of the color. It also ranges from 0-100% and describes the intensity of the color. Figure 2.22 gives a visualization of how the values in both RGB and HSV affects the resulting color.

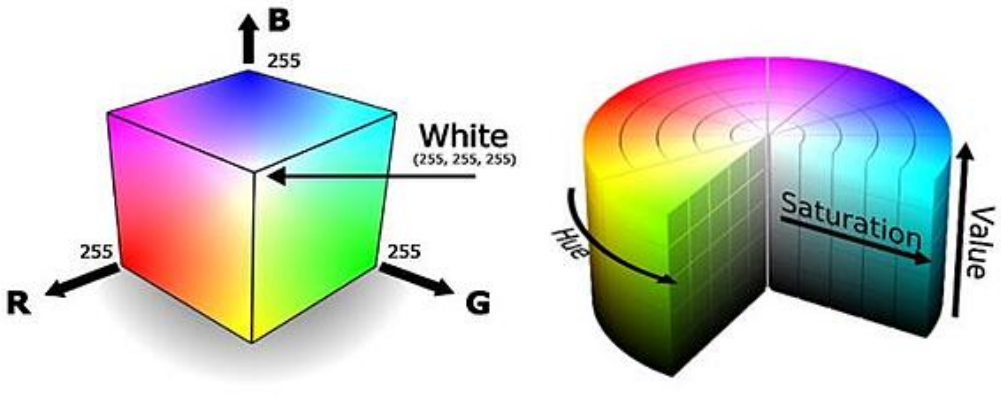


Figure 2.22: Colormap visualizing RGB and HSV color values [15]

2.6.2 Color filtering

Color filtering is a method to filter an image based on a color. One way to perform color filtering is by defining a lower and upper bound of a color range, either in RGB or HSV format (See section 2.6.1) depending on the image encoding. After the color range is defined, each cell (pixel) in the image matrix can be compared to the color range. The result can be represented by creating a matrix with an equal amount of rows and columns as the original image and all values is set to 0, also known as a mask. In the mask image, all the pixels which is within the specified color range in the original image is set to white (255 in a grayscale image). The mask gives a representation of where the specified color is true (greater than 0). Figure 2.23 shows an RGB image and a yellow color mask.

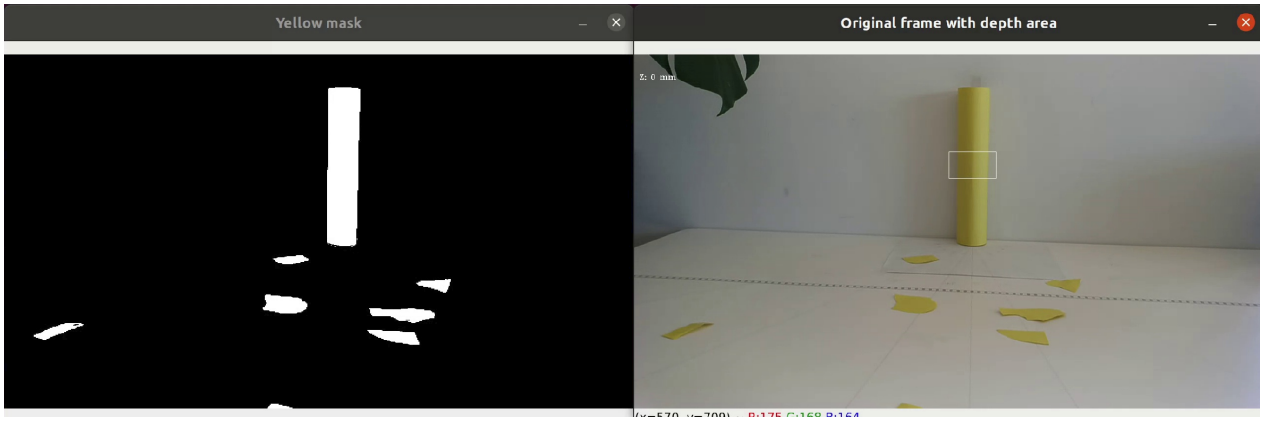


Figure 2.23: RGB image and a yellow color mask

2.6.3 Contours

A contour is a line or a curve which is drawn between points of equal values or attributes [57]. For instance, on a map, contours are often used to describe elevation. It is also commonly used to find edges in an image based on distance or color. A **color mask** where the edges of the color area containing a specified color are clearly visualized, can be used as a good tool to find the contours of a specified color. The contour can be used to determine the area (often number of pixels) of each of the contours, as well as the center of each contour. In figure 2.23, a white box is drawn in the RGB image around the center of the biggest contour in the image.

2.7 KNN (K-nearest neighbor)

KNN (K-nearest-neighbor) is a machine learning algorithm used for classification and regression. The algorithm is used when there is little or no prior knowledge about how the data is distributed.

There are two different distinctions: KNN-classification and KNN-regression. In KNN-classification, the data is segmented into classes according to a popular vote decided by its neighbors. In KNN-regression the output is the property value of the object, with the value being the average of the k nearest neighbors. [58]

Chapter 3

Materials and methods

This chapter starts off by describing how the project has been organized. Then, it introduces the materials that have been used in this project. This is followed by a detailed description of how the group decided on which sensors to use and the methods used to estimate both orientation and position. Finally, the test procedures are described.

3.1 Project Organization

The project group has been organized with a project manager, a secretary, and two group members. The project manager's areas of responsibility have been to update the progress of the project in the Gantt chart and chair meetings with the steering group. The secretary's tasks have been to convene meetings, write and distribute meeting reports, as well as write progress reports before each meeting with the steering group. Although the group has had a project manager, a flat management structure has been practiced, where decisions have been made together.

Throughout the project, meetings have been held with the steering group every 14 days. These meetings have been arranged online through Microsoft Teams, due to the Covid-19 pandemic. During the meetings, progress for the project has been discussed, the group has presented its proposals for solutions to problems, and the steering group has provided guidance, tips and suggestions for solutions. During the pre-project, a project plan was made based on the given task. A Gantt diagram based on the project plan was made, and this has been guiding the project implementation. Google Drive has been used to store and share all relevant files and

documents throughout the project. Code has been shared on Github.

3.2 Software

Software and programming languages used in this project is described in this section.

- **Python:** Python has been used with various libraries to run the GUI, filter and fuse the IMU data, as well as image processing.
- **Arduino IDE:** Arduino IDE has been used to program the Arduino Uno micro controller. The Arduino Uno micro controller was programmed using the Arduino programming language, which is similar to the C++ programming language.
- **Autodesk Fusion 360:** Autodesk Fusion 360 is a cloud based software platform. In this project, Autodesk Fusion 360 has been used to create 3D-modeled representations used as illustrations in this report.
- **Git:** Git has been used as code repository, for sharing of code, and for version control.
- **Qt Designer:** Qt Designer has been used to create the GUI.
- **Draw.io:** Draw.io is a free online diagram software. It has been used to create flowcharts, diagrams and illustrations used in this report.

3.3 Hardware

3.3.1 OpenCV AI Kit: OAK-D

The OAK-D is a stereo camera which provides depth from two cameras which is placed on each end of the housing. It also has a 4K resolution camera in the center which provides color information. The OAK-D stereo camera has been used in this project to capture RGB and depth images.

Figure 3.1: OpenCV AI Kit: OAK-D [16]

3.3.2 Intel RealSense T265

The Intel RealSense T265 is a tracking camera with two fisheye lens sensors, an IMU and an Intel Movidius Myriad 2 VPU [59]. This camera has been used to go through the steps of creating a stereo camera as explained in section 3.8.

Figure 3.2: Intel Realsense T265 [17]

3.3.3 Adafruit BNO055

Adafruit BNO055 is an 9-DOF IMU sensor. The sensor includes a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. It also includes a temperature sensor. The sensor chip contains software which includes sensor fusion algorithms that calculates absolute and relative orientation represented by Euler angles and quaternions. In this project, only raw data from the gyroscope, accelerometer and magnetometer is extracted from the sensor.

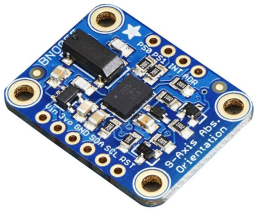


Figure 3.3: Adafruit BNO055 [18]

3.3.4 Arduino UNO

Arduino UNO is a microcontroller board equipped with sets of digital and analog input/output (I/O) pins. In this project, the microcontroller is used to read the data from the IMU and transmit the data to a computer via serial communication.

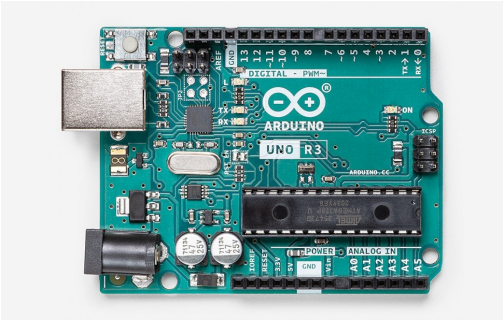


Figure 3.4: Arduino UNO [19]

3.3.5 Svive Hydra Microphone Arm

Svive Hydra is a microphone arm. This arm is used in the prototype to act as a model of an offshore gangway.

Figure 3.5: Svive Hydra Microphone Arm [20]

3.4 Choosing sensors

When choosing which sensor types to use for this project, the group started out by listing a number of different sensors suitable for distance and orientation estimation. Before choosing which sensors to implement in the solution, the advantages and disadvantages for each sensor (in the aim of this project) was thoroughly considered. External factors such as influence of weather, wind and sunlight were taken into account. Below is a description of these considerations.

3.4.1 Sensors for distance/position

Measuring and estimating distance can be done in several ways. Sensor technologies that is applicable for this project is considered below.

Ultrasonic sensor

Using sound to measure the distance implies that the measurements are not affected by many external factors such as rain and snow. This gives the ultrasonic sensor an advantage over the distance sensors that uses light to measure distance. Yet, this makes the measurements slower, as the speed of sound is slower than the speed of light. In addition, ultrasonic sensors are relatively inexpensive, small in size and have a decent resolution [60][61].

Infrared sensor

One advantage for the IR sensor is that it is not dependent on environmental light. This means that it can be used in both light and dark conditions. On the other hand, it is affected by environmental conditions such as water and snow which can reflect the emitted light, resulting in the light returning from multiple angles. The IR distance sensors are relatively cheap and have a varying (commonly short) distance range [61].

Radar sensor

An advantage of the radar is that it is robust and it works under rough weather conditions [62]. It can penetrate clouds, fog and snow. However, large objects that are close to the transmitter

can disturb the receiver [63]. This is unfortunate since the wind turbine will be relatively close to the sensor during operation.

LiDAR

An advantage of LiDAR is that it creates an accurate map of the scenery. This enables the possibility to calculate the distance to any point within the point cloud with high accuracy. The amount of information possible to extract from a point cloud is substantial, this includes different angles, distances and so on.

One drawback of LiDAR technology is that it can be difficult to recognize objects. By for example using camera there are many tools available for image processing, where objects can be segmented by color and so on, in contrary to point cloud processing. Another downside of using LiDAR is the cost of the hardware, which is relatively expensive compared to for example cameras [64].

ToF camera

ToF cameras offer precise and fast measurements over long ranges. Yet, a factor that may affect the measurements of the sensor is sunlight. Sunlight may make it difficult for the sensor to read the reflected light. Also, when measuring bright and concave shaped surfaces, the light may reflect back and forth inside the concave shape, causing the time of flight to be longer than it is in reality.

Stereo camera

An advantage to using this method is that there are few external disruptions to the depth calculation, if not considering major or complete obstructions. It also offers a high accuracy on close objects, and is able to give accurate and reliable measurements over long distances. Additionally, by using image processing, a stereo camera can be used during nighttime, rain and snow. Stereo cameras are also relatively cheap to develop, as they mainly require two cameras to work. This also allows customization in order to work optimally under the intended circumstances.

On the other hand, a stereo camera uses feature detection algorithms to estimate depth to objects. The features in both the images are then compared and matched to each other. The

features that are detected are often corners, edges and highlighted areas depending on which feature detection algorithm is chosen. This means that in order to find the depth, the cameras must be able to capture these features. In other words, a stereo camera would not be able to determine the distance to a homogeneous colored surface. Another disadvantage to stereo cameras is that the features may be mismatched, meaning that one feature is matched to another by the algorithm, but in reality these are not the same features. This causes the distance to that feature to be miscalculated. Lastly, it requires high resolution cameras to give accurate measurements over long distances [65].

The chosen solution

Based on the considerations above, the group decided to investigate the possibilities of distance estimation using a stereo camera. This was because it provided the ability of measuring the distance to an area, and not only a single point. Additionally, a stereo camera provided an RGB image which could be processed using various methods to localize the wind turbine. By finding the center of the wind turbine in the RGB image, the subsequent images could be compared to the initial image, hence giving information about the displacement of the wind turbine in pixels in the image. The pixel displacement in combination with the distance to the wind turbine gave the required information to use trigonometry to calculate the displacement in multiple directions. This method is described in more detail in section 3.9.

3.4.2 Sensors for orientation

To estimate orientation, it is convenient to combine data from several sensor types. Sensor technologies that is applicable for this project is considered below.

Gyroscope

A gyroscope sensor can accurately and quickly measure angular velocity. By integrating these measurements, the current angular position can be estimated. However, due to accumulating integration errors, the orientation estimates from a gyroscope will drift over time.

Accelerometer

A tri-axis accelerometer can be used to find the absolute roll and pitch angles of an object with constant velocity. When an object has a constant velocity, the sum of forces should be zero. In this case, the only acceleration the accelerometer would measure is 1G towards the earths center. The accelerometer measures in three axes, and the values from these axes can be considered the decomposition of the gravitational acceleration vector. By using this decomposition, the pitch and roll angles can be found (see section 3.6.1 for implementation). However, since the gravitational acceleration vector is perpendicular to the horizontal plane and parallel to the yaw axis, it is impossible to find the yaw angle with just an accelerometer.

Magnetometer

A tri-axis magnetometer can be used to measure the magnetic field around it. By knowing where the magnetic north is located, the magnetometer readings can be used to find the angle between "north" on the magnetometer and the magnetic north. See "Magnetic distortion compensation" in section 2.5.5. Magnetometers are inherently prone to magnetic interference around the sensor. Magnetic sources in proximity of the magnetometer can reduce the accuracy significantly (see section 3.5.3).

The chosen solution

Based on the considerations above, it was decided to investigate different combinations of sensor fusion algorithms to fuse measurements from a gyroscope, accelerometer and magnetometer. The different sensor fusion combinations is described in section 3.6.

3.5 Sensor calibration

3.5.1 Gyroscope Calibration

Zero offset error in gyroscopes are the offset shown in gyroscope measurements when it is standing still. One way to calculate this error is to collect samples with the gyroscope stationary, and find the mean value of all the sampled values. Because the gyroscope is standing still,

the value should ideally be zero. By subtracting this calculated offset, the sensor values will be centered around zero. This will not remove white noise, but will center the values around zero [66].

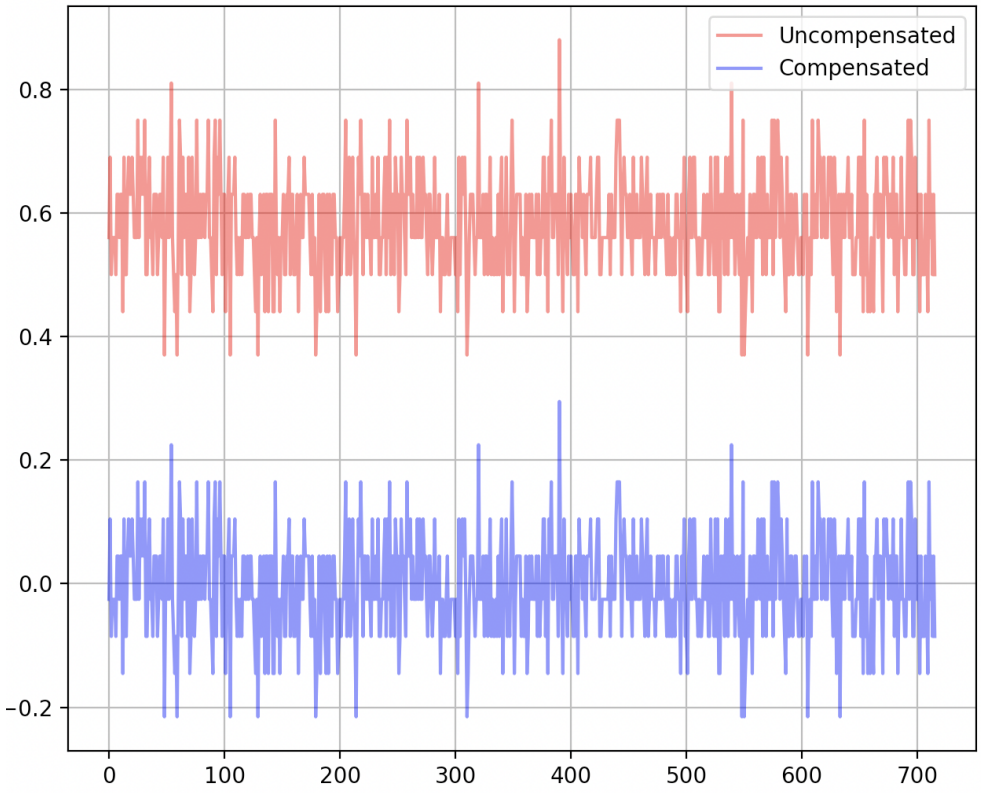


Figure 3.6: Showing uncalibrated and calibrated gyro values

3.5.2 Accelerometer Calibration

To calibrate the accelerometer, the gravity acceleration can be used. By aligning the accelerometer axes with the gravity vector, ideally the accelerometer should display acceleration in only one direction, and this should be equal to ±1G [67].

Data should be gathered in 6 different orientations: Z pointing up, Z pointing down, Y pointing up, Y pointing down, X pointing up and X pointing down (see figure 3.7).

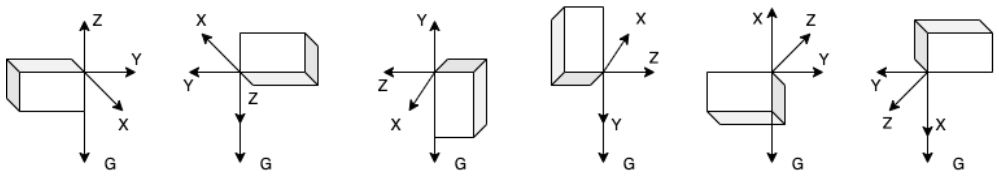


Figure 3.7: Orientation during data capture

The zero-G is the measured value from the accelerometer corresponding to zero acceleration on an axis. To calculate this value, the maximum value (the positive direction measurement) is added to the minimum value (the negative direction measurement) and divided by 2 (see equation 3.1).

$$m_{axis} = \frac{max_{axis} + min_{axis}}{2} \tag{3.1}$$

The scale factor is the value used to scale measurements into G´s. The measured values can be of any unit, e.g. m/s^2, mV etc.. The scale factor is calculated by subtracting the minimum measured value from the maximum measured value and dividing by 2 (see equation 3.2).

$$\delta_{axis} = \frac{max_{axis} - min_{axis}}{2} \tag{3.2}$$

The calibrated results is then calculated by following equation 3.3:

$$a_{axis} = \frac{sensorvalue_{axis} - m_{axis}}{\delta_{axis}} \tag{3.3}$$

To get the values back to m/s^2, equation 3.4 can be used:

$$a_{axis} = \frac{(sensorvalue_{axis} - m_{axis}) \cdot 9.81}{\delta_{axis}} \tag{3.4}$$

3.5.3 Magnetic Field Calibration

Magnetic field measurements are subject to soft iron and hard iron distortions. These distortions can have a significant impact on the measurement accuracy [68].

Before starting the calibration, raw magnetometer data must be gathered. To gather data, the magnetometer must be moved freely to capture the magnetic field in all directions. One

way to capture this data is to move the magnetometer in figure eight motions. By plotting the magnetometer values with X in terms of Y, X in terms of Z and Y in terms of Z, the soft and hard iron distortions can be visualized. With optimal conditions, the magnetometer plots should show a perfect uniform circle centered around the origin.

Hard iron distortions will offset the blob of measurements from the origin (see figure 3.8). These types of distortions are created by objects with a magnetic field, such as permanent magnets in speaker drivers or magnetized iron.

Soft iron distortions will change the shape of the blob of measurements from a circle to an ellipse (see figure 3.9). This means the soft iron distortions will alter the magnetic field, but not move it. This type of distortion can be caused from metals, such as nickel and iron. Soft iron distortions are harder to compensate for compared to hard iron distortions.

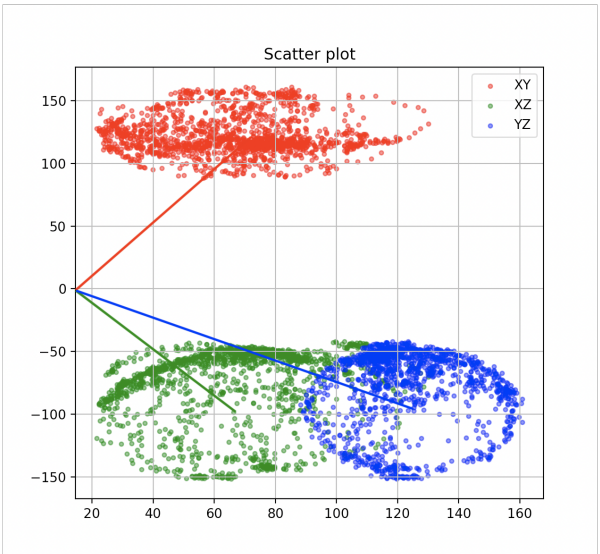


Figure 3.8: Display of the hard iron distortion

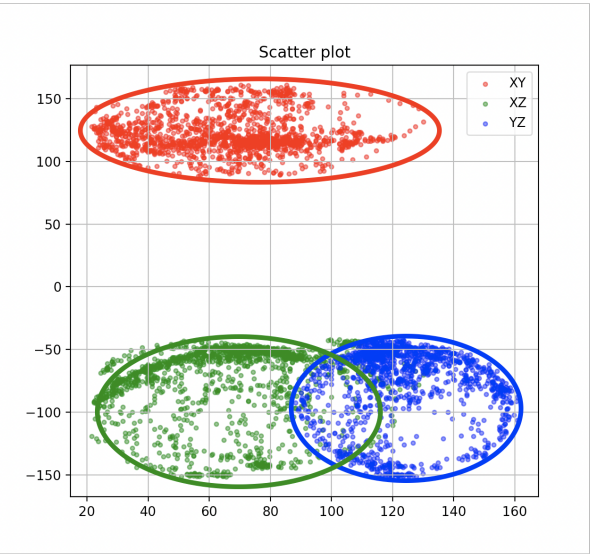


Figure 3.9: Display of the soft iron distortion

To find the offset, half of the mean value of maximum and minimum measurements are used (equation 3.5). If the measurements are perfectly centered around the origin, this value would be 0.

$$offset = \frac{max + min}{2} \tag{3.5}$$

To find the scaling factor, half of the difference between maximum and minimum is calculated for each axis (equations 3.6, 3.7 and 3.8). The average of these values are then calculated

to find the average difference between maximum and minimum values of all the axes (equation 3.9). Then, the scale factor for each axis is calculated by dividing the average difference by the difference in the specific axis (equations 3.10, 3.11 and 3.12).

$$\Delta x = \frac{max_x - min_x}{2} \tag{3.6}$$

$$\Delta y = \frac{max_y - min_y}{2} \tag{3.7}$$

$$\Delta z = \frac{max_z - min_z}{2} \tag{3.8}$$

$$\overline{\Delta} = \frac{\Delta x + \Delta y + \Delta z}{3} \tag{3.9}$$

$$scale_x = \frac{\overline{\Delta}}{\Delta x} \tag{3.10}$$

$$scale_y = \frac{\overline{\Delta}}{\Delta y} \tag{3.11}$$

$$scale_z = \frac{\overline{\Delta}}{\Delta z} \tag{3.12}$$

The final result after calibrating for offset and scaling is shown in figure 3.10. The blobs are centered around the origin, and the shape is more circular. The shape is still slightly elliptical. To get even more precise results, matrix operations can be performed [69].

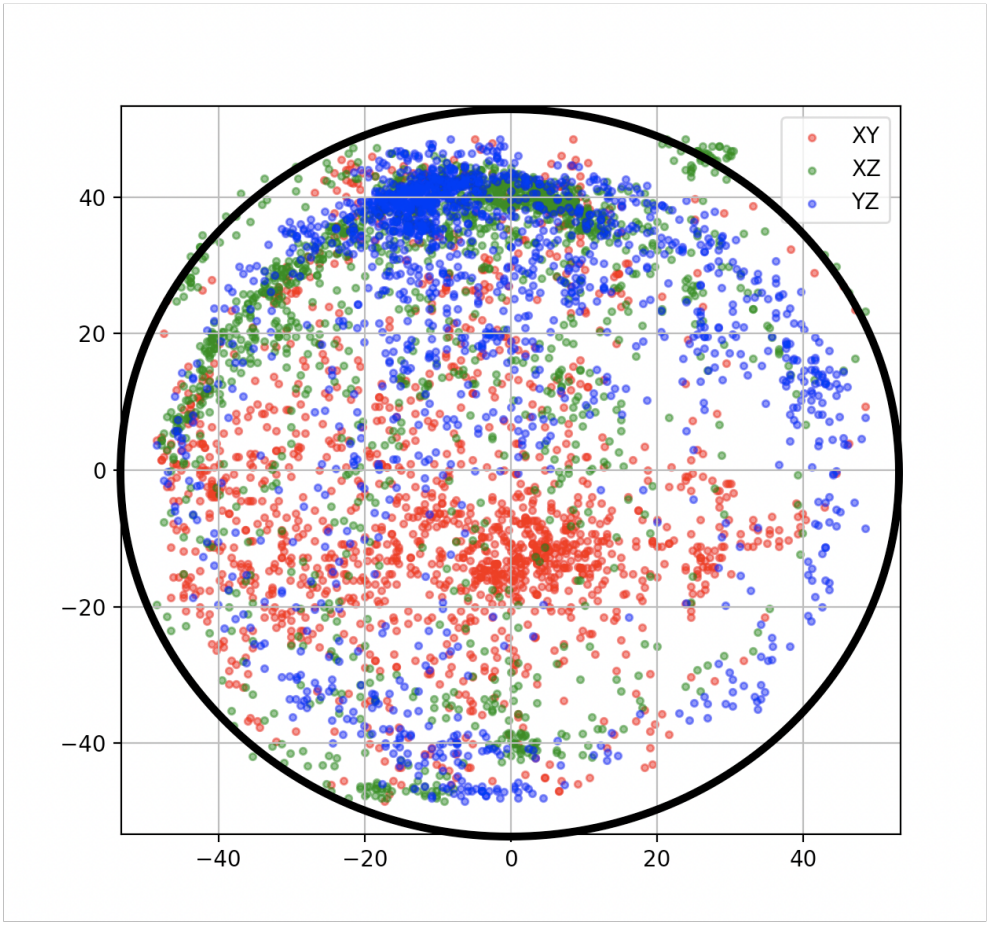


Figure 3.10: Display of the calibrated result

3.6 Sensor Fusion for Orientation Estimation

To get reliable estimates of the orientation of the gangway, an Adafruit BNO055 IMU is used. This IMU consists of a 3 axis raw gyroscope, a 3 axis raw accelerometer, and a 3 axis raw magnetometer. The output values from these sensors are subject to white noise, and filtering and fusing the outputs is therefore necessary to achieve stable and reliable estimates of orientation. For the aim of this project, different filter setups is developed and tested. This section contains a description of the alternatives considered for filtering.

The Arduino reads the raw values from the IMU through the I^2C protocol. These values are then calibrated, and transmitted through serial communication to a computer which processes the signals and visualizes the angle outputs. This is done using a script written in Python.

3.6.1 Filter Setup 1

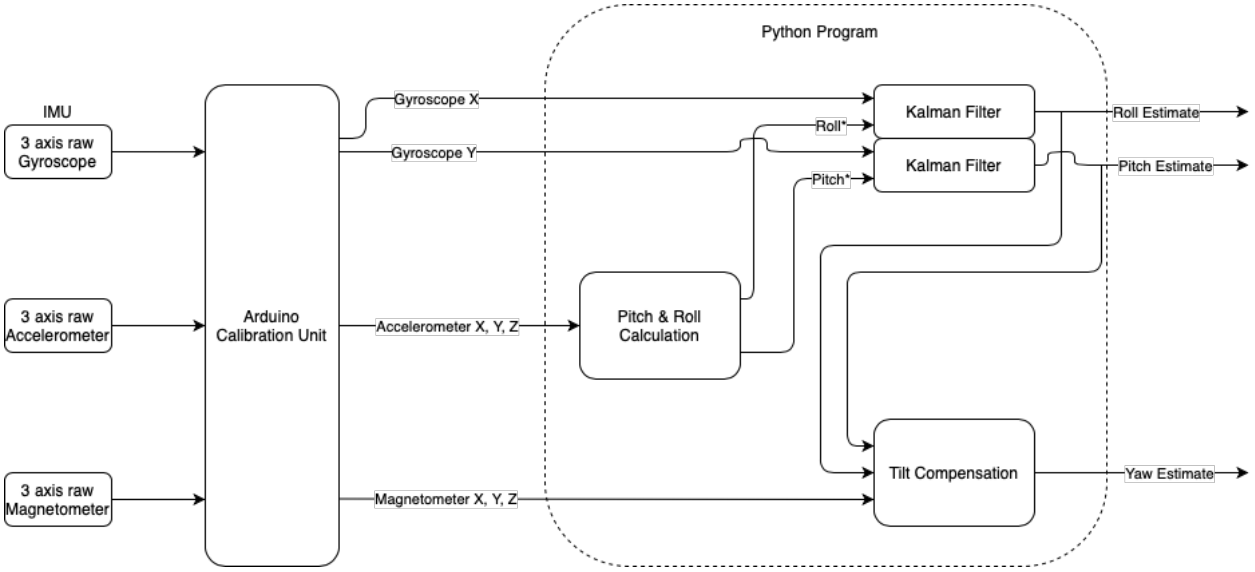


Figure 3.11: Filter Setup 1

In this filter setup, roll* and pitch* are obtained by using the calibrated accelerometer signals in equations 3.13 and 3.14. These equations are derived in "Tilt Sensing Using a Three-Axis Accelerometer" by Mark Pedley [70].

$$Pitch = \arctan \frac{Accel_X}{\sqrt{{Accel_Y}^2 + {Accel_Z}^2}} \tag{3.13}$$

$$Roll = \arctan \frac{Accel_Y}{\sqrt{{Accel_X}^2 + {Accel_Z}^2}} \tag{3.14}$$

The calculated roll* and pitch* are noisy due to the fact that the accelerometer values that are used in the calculations are subject to white noise. Therefore, the roll* and pitch* calculations is logged for 5 minutes while the IMU is kept still. The variance of the roll* and pitch* calculations is then achieved by using the logged calculations.

To get reliable estimates of roll and pitch, two separate Kalman filters are used. Calculated roll* is combined with the measurement from the gyroscope around the x-axis through a Kalman filter, and calculated pitch* is combined with the measurement from the gyroscope around the y-axis through another Kalman filter. Their respective variances are put into the R-

matrices of the two filters, as seen in equations 3.15 and 3.16. The outputs from the Kalman filters are the estimated roll and pitch rotation where unwanted noise are filtered out.

$$R_{roll} = \begin{bmatrix} Roll^* \, Variance & 0 \\ 0 & Gyro_X \, Variance \end{bmatrix} \tag{3.15}$$

$$R_{pitch} = \begin{bmatrix} Pitch^* \, Variance & 0 \\ 0 & Gyro_Y \, Variance \end{bmatrix} \tag{3.16}$$

To estimate the yaw rotation angle, also called heading, a tilt compensation algorithm is used. This algorithm utilizes the magnetometer signals in combination with the estimated pitch and roll. The earth's magnetic field has a component parallel to the earth's surface [71], and while the 3-axis magnetometer is parallel to earth's surface, it can be able to measure the absolute heading accurately through the direction of earth's magnetic field [72]. However, when the magnetometer is tilted and no longer is parallel to earth's magnetic field, the direction of axial sensitivity will change. Consequently, different amounts of error will appear, depending on how much the magnetometer is tilted. To tackle this problem, the tilt compensation algorithm is made to map the magnetometer data to the horizontal plane (which is parallel to earth's magnetic field), regardless of the orientation of the magnetometer. This way, the algorithm should provide an accurate yaw estimate.

The algorithm is described by equations 3.17, 3.18 and 3.19. In the equations, m_x, m_y and m_z are the normalized magnetometer outputs, and α, β and γ represent roll, pitch and yaw respectively. Equations 3.17 and 3.18 maps the magnetometer data to the horizontal plane, and equation 3.19 calculates the estimated yaw rotation angle.

$$XH = m_x \cos(\beta) + m_y \sin(\beta) \sin(\alpha) + m_z \sin(\beta) \cos(\alpha) \tag{3.17}$$

$$YH = m_y \cos(\alpha) + m_z \sin(\alpha) \tag{3.18}$$

$$\gamma = \arctan\left(\frac{-YH}{XH}\right) \tag{3.19}$$

3.6.2 Filter Setup 2

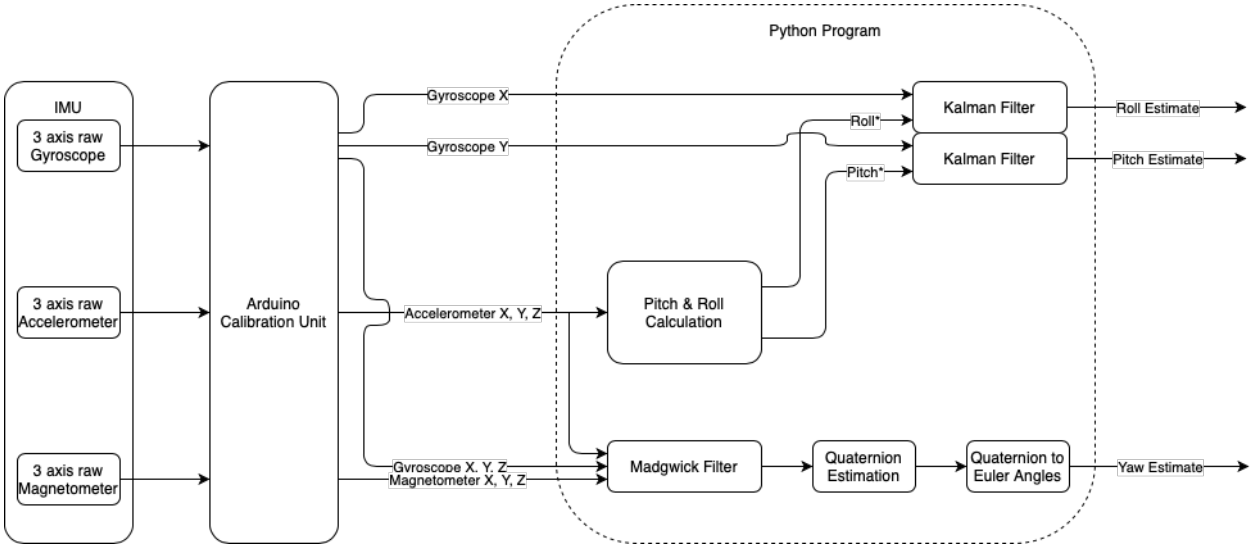


Figure 3.12: Filter Setup 2

In this filter setup, roll and pitch are estimated in the same way as in filter setup 1.

To estimate yaw, a Madgwick filter is used. As seen in figure 3.12, the Madgwick filter uses the calibrated measurements from the gyroscope, the accelerometer and the magnetometer. Using the magnetometer is optional, and the Madgwick filter works without using the magnetometer measurements, but it is essential to include the magnetometer if the goal is to achieve absolute orientation. The output from the Madgwick filter is a quaternion representation of the orientation. The Euler angles are then calculated from the quaternion, and the yaw estimate is retrieved from these angles.

3.6.3 Filter Setup 3

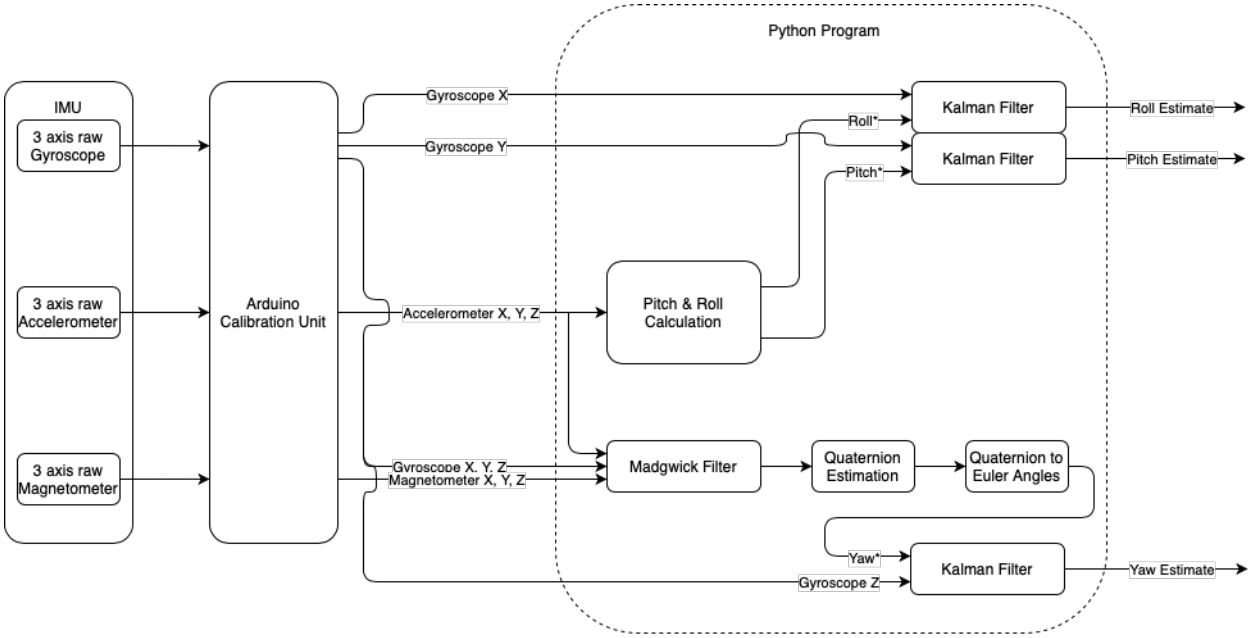


Figure 3.13: Filter Setup 3

Filter setup 3 is very similar to filter setup 2. However, in this setup, the difference is that the estimated yaw rotation angle from the Madgwick filter is combined with the measurement from the gyroscope around the z-axis through a Kalman filter. The output from the Kalman filter is a filtered yaw estimate, where noise from the Madgwick filter is reduced.

3.7 Transforming orientation from sensor to ship

The gangway can move in relation to the vessel. It can raise or lower the boom (tilt the gangway up or down) as well as slew (rotate the gangway around the z-axis of the ship). By having the IMU mounted on the tip of the gangway, the measured sensor orientation will not necessarily be the same orientation as the ship. By using rotation matrices (section 2.2.4) with the known boom angle and slew angle, the sensor orientation can be transformed to match the ship.

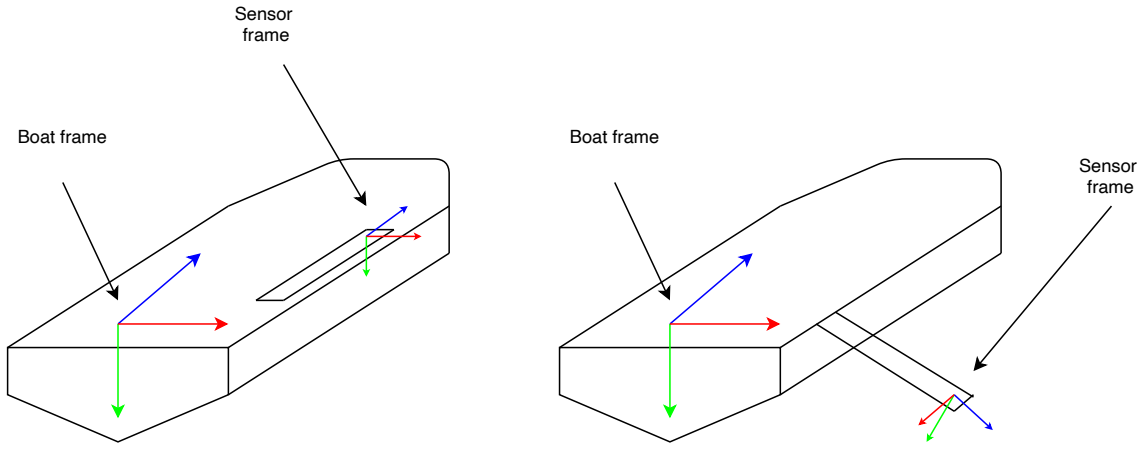


Figure 3.14: Showing the relation between ship frame and sensor frame when the gangway moves

During transformation from sensor to ship, the ship orientation in relation to the sensor frame is considered. When the gangway is aligned with the ship (left in figure 3.14), the slew angle corresponds to yaw, and boom angle corresponds to pitch. Thus, the boom and slew can be compensated by rotating around pitch and yaw using rotation matrices.

The sensor's angle values are represented in relation to a stationary world frame. Pitch and roll angles are absolute (in relation to the horizontal plane), and the yaw is relative from a given starting position. One suggestion to set the stationary world frame is to set the initial pose when starting estimation as the world frame.

The following sequence is used to transform sensor angles to boat representation in relation to the world:

1. Update sensor rotation matrix ${}^{W}_{S}\mathbf{R}$ with new sensor values
2. Update boat rotation matrix ${}^{S}_{B}\mathbf{R}$ with new boom and slew values
3. Multiply: ${}^{W}_{\not S}\mathbf{R}\,{}^{\not S}_{B}\mathbf{R} = {}^{W}_{B}\mathbf{R}$

Consider an example with the following sensor readings:

- $\psi_{sensor} = 78°$
- $\theta_{sensor} = 17°$

- $\phi_{sensor} = -6°$
- $\alpha_{slew} = 60°$
- $\alpha_{boom} = 5°$

General rotation matrix with ZYX rotation sequence:

$$\mathbf{R}(\phi,\ \theta,\ \psi) = \begin{bmatrix} c\theta c\psi & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{3.20}$$

Adding sensor values to equation 3.20 to get the rotation matrix ${}^{W}_{S}\mathbf{R}$:

$${}^{W}_{S}\mathbf{R}(-6,\ 17,\ 78) = \begin{bmatrix} 0.1988269 & -0.9791432 & -0.0417898 \\ 0.9354072 & 0.1768794 & 0.3061487 \\ -0.2923717 & -0.0999611 & 0.9510660 \end{bmatrix} \tag{3.21}$$

Add boom and slew angles to equation 3.20 to get the rotation matrix ${}^{S}_{B}\mathbf{R}$:

$${}^{S}_{B}\mathbf{R}(0,\ 5,\ 60) = \begin{bmatrix} 0.49809735 & -0.8660254 & 0.04357787 \\ 0.86272992 & 0.5 & 0.07547909 \\ -0.08715574 & 0. & 0.9961947 \end{bmatrix} \tag{3.22}$$

Multiplying the rotation matrices from equation 3.21 and 3.22 will result in the following rotation matrix representation of the ships orientation in relation to the world:

$${}^{W}_{B}\mathbf{R} = {}^{W}_{S}\mathbf{R}\,{}^{S}_{B}\mathbf{R} = \begin{bmatrix} -0.74205877 & -0.66176079 & -0.10687113 \\ 0.59184038 & -0.7216467 & 0.35909749 \\ -0.31475984 & 0.20322079 & 0.92716102 \end{bmatrix} \tag{3.23}$$

To get the orientation of the ship represented as Euler angles, the rotation matrix must be converted to Euler angles (see section 2.2.5). From equation 2.58, singularities are checked:

$$sy = \sqrt{-0.74205877^2 + 0.59184038^2} \approx 0.949 \neq 0 \tag{3.24}$$

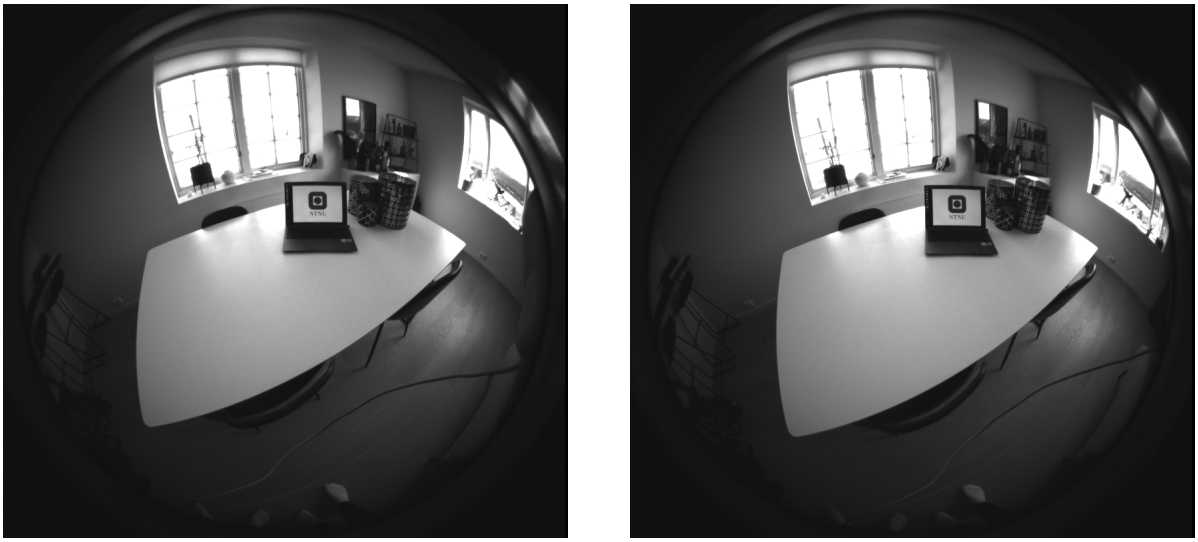
This shows that the current example is not approaching singularity. The Euler angles are then calculated:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(0.20322079,\ 0.92716102) \\ atan2(0.31475984,\ 0.949) \\ atan2(0.59184038,\ -0.74205877) \end{bmatrix} \approx \begin{bmatrix} 12.36° \\ 18.35° \\ 141.43° \end{bmatrix} \tag{3.25}$$

3.8 Creating a stereo camera

To create a system that is adaptable to offshore operations a general system for setting up a stereo camera is proposed. Due to time constraints, the group was not able to implement this in the final solution. However, the approach is explained thoroughly in this section. For testing purposes the Intel RealSense T265 tracking camera was used. This section describes the steps and methods used to create the system. For testing purposes the *Intel RealSense T265* tracking camera was used.

The process of creating the stereo camera started by taking two images simultaneously, with identical cameras. This gave a *left image* and a *right image* (See figure 3.15).



(a) Left image (b) Right image

Figure 3.15: Original photos

3.8.1 Camera calibration

Calibrating the camera pair was done as follows: First, mounting the cameras fixed in place, and having the lenses aligned correctly. As having the two lenses placed perfectly aligned was practically impossible in addition to removing the distortion from the lens, the cameras was calibrated with software as well. OpenCV offered this functionality [73]. If the cameras were not calibrated correctly there would be an error, which would make it difficult to calculate the disparity between the frames.

Calibration of the camera pair was performed both on the left and right camera. The process started by taking pictures of a calibration plate, which was a checker patterned board. Note that it was important that the checker pattern dimensions was not square, but rectangular. For example 6x9 squares, not 9x9 squares. In total there were taken 50 images of the calibration plate with the left and right camera, all from different angles. This returned two matrices **K** and **D**. **K** which holds the cameras intrinsic parameters like focal length, optical centers and so on. This is also called the camera matrix. And **D** which describes the cameras extrinsic parameters. This corresponds to rotation and translation, in other words the distortion in the image. A selection of the calibration images can be seen in figure 3.16.

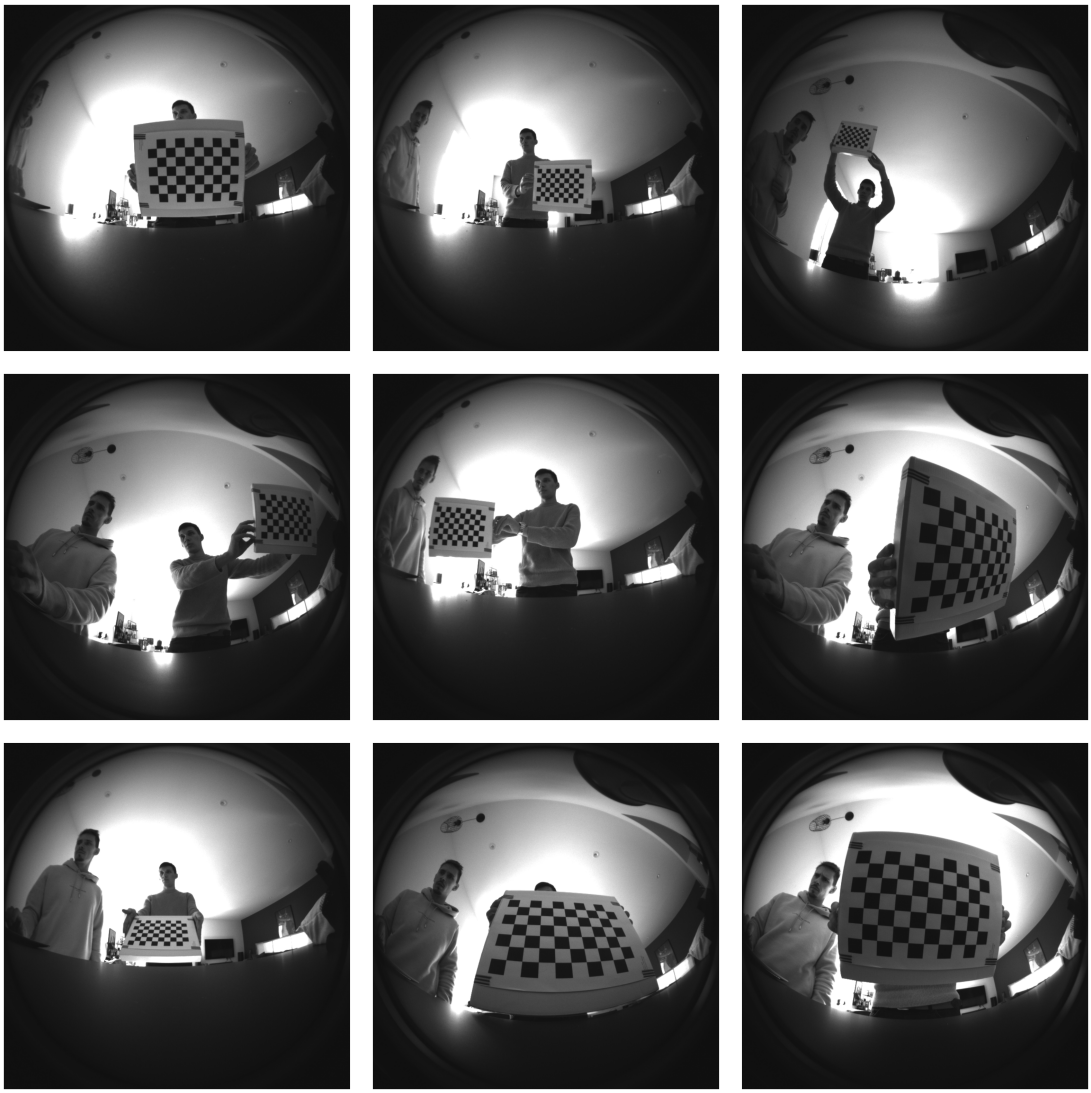


Figure 3.16: Calibration collage

After the calibration matrices **K** and **D** got acquired (see equation 3.26), the next step was to rectify the left and right images. This implied finding the rotation factor between the frames. In other words, creating a matrix that represented the skew between the frames. By doing so it was possible to project the *left image frame* on to the *right image frame*. This projected both images onto the same plane (See figure 3.17).

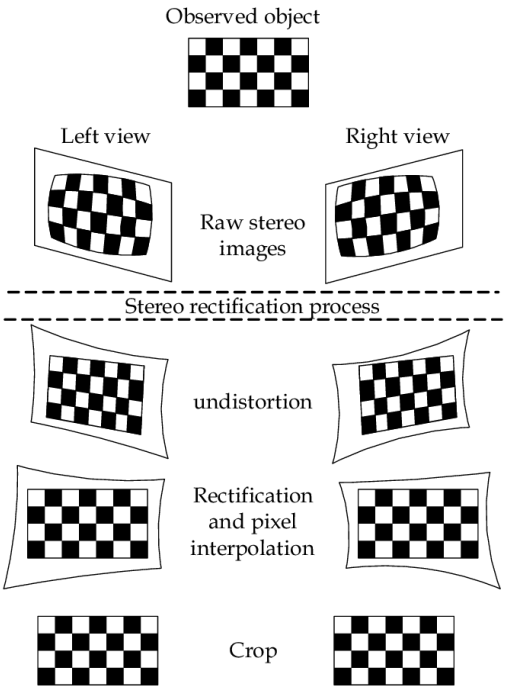
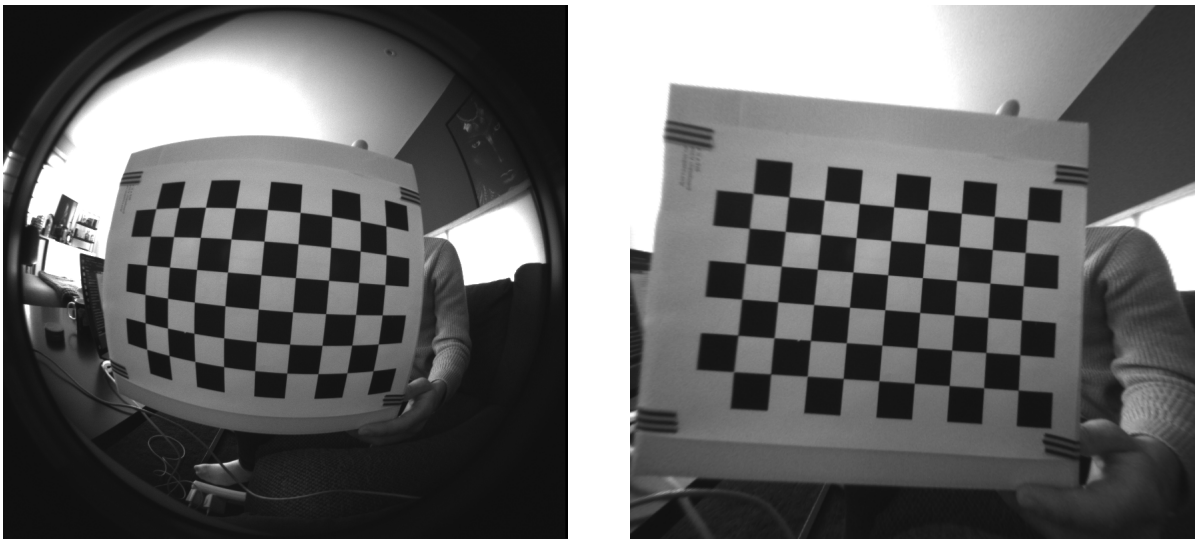


Figure 3.17: Rectified Image

Rectification of the cameras implies that the coordinate of a given object in both images matches location in the Y-axis. In other words, the lines between the matching features are perfectly horizontal, and not tilted.

$$\mathbf{K} = \begin{bmatrix} 285.62 & 0.0 & 419.39 \\ 0.0 & 284.78 & 402.56 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} -0.0064 \\ 0.0034 \\ -0.0262 \\ 0.0007 \end{bmatrix} \tag{3.26}$$

After the images was undistorted, one apparent feature was that the lines of the checkered board was no longer curved, but became straight as shown in figure 3.18. I.e, the fish eye effect was gone. Yet, this came at a cost. A portion of the outer edges of the image got lost in the process and the size of the images got reduced.



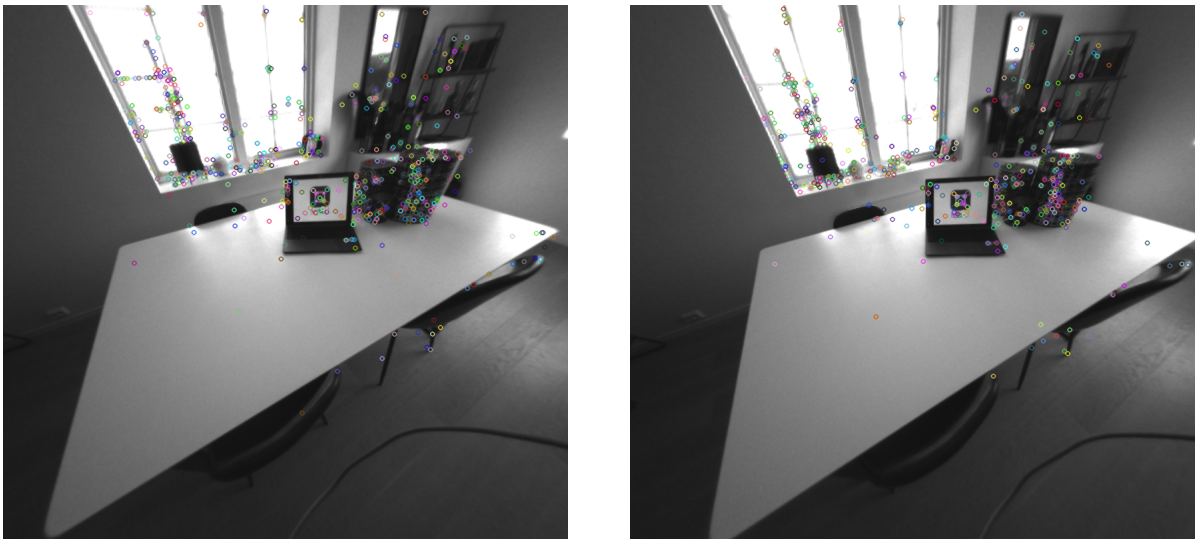
(a) Distorted image (fisheye)

(b) Undistorted image

Figure 3.18: Image before and after undistortion

3.8.2 Feature detection

One of the main steps towards calculating depth from a stereo camera was to find the difference in the two images. These differences was found by using a feature detection algorithm that detected the same features in the two images, which can then be compared. There are several algorithms available, such as Harris Corner Detection [74], SIFT (Scale-Invariant Feature Transform) [75] and SURF (Speeded Up Robust Features) [76]. Figure 3.19 shows the left and right images after SIFT feature detection has been applied.



(a) Features left image

(b) Features right image

Figure 3.19: Features left and right image

After identifying the features in each of the images, the FLANN [77] feature matching algorithm was used to compare and match the same features in both the images. FLANN is based on a machine learning technique, KNN [78] and returned the probability of two points matching up. In other words, which feature in one image corresponded with a specific feature in the other image. This described the correlation between the same feature in the two images. The features that was matched in both the images got marked with a purple circle and had drawn lines between them as seen in figure 3.20. The features that was not found in both the images got marked in red and disregarded.

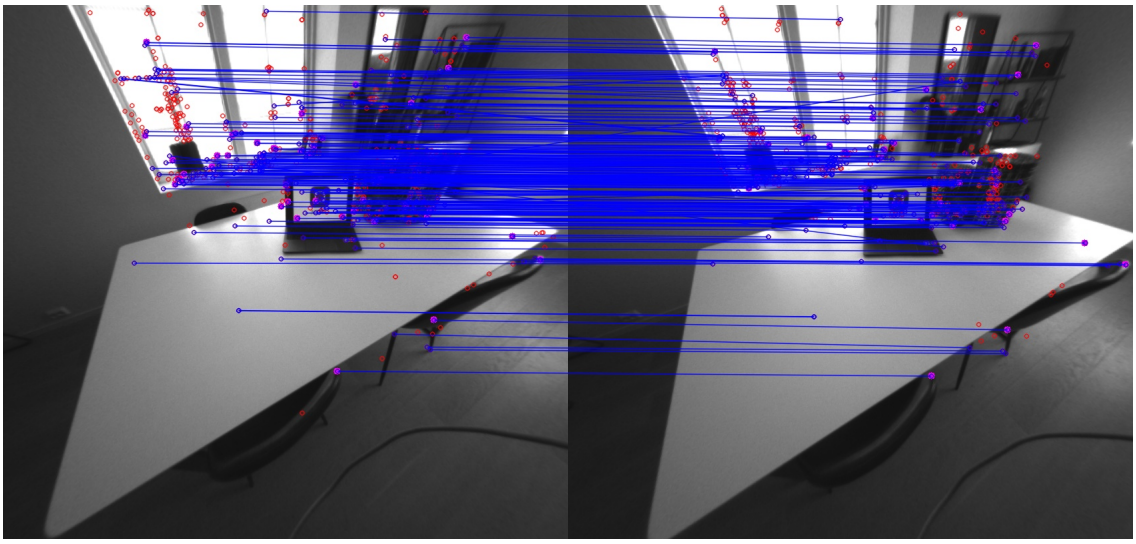


Figure 3.20: Matching features withing the images

3.8.3 Calculating depth

Calculating the distance within a stereo image is done by looking at the displacement of an object in both of the images. For this, certain parameters is needed. Figure 3.21 displays the stereo camera setup with its given parameter.

- B - Baseline, distance between the cameras
- D - Depth to a given object
- AFoV - Angular field of view
- FoV - The distance in meter of the field of view
- R_h - Horizontal resolution
- Px_d - Displacement of an object from the left to the right camera
- x - Object in the left image
- x' - Object in the right image

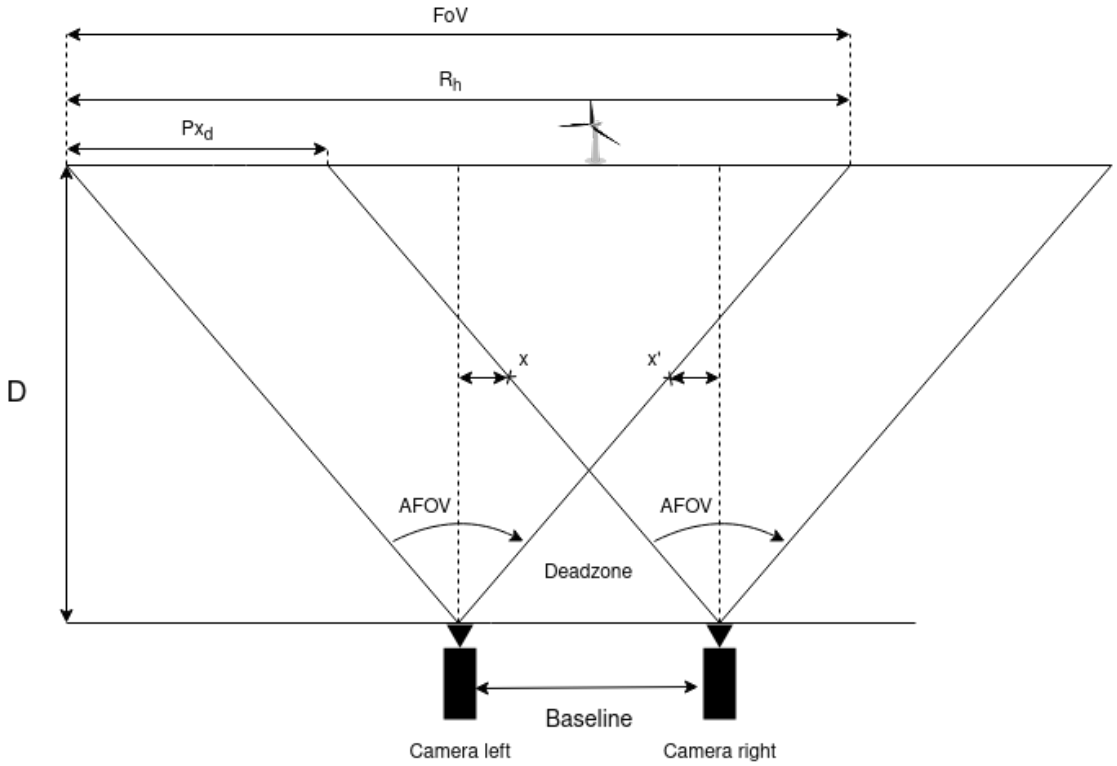


Figure 3.21: Stereo camera setup

The equations described in this section is derived in "Distance measuring based on stereoscopic pictures" by Jernej Mrovlje and Damir Vrančić [79]. Following is an explanation for finding the depth by looking at the relation between Px_d and R_h which is equivalent to the relation between **FoV** and the **Baseline**. This can be seen from figure 3.21 by imagining the right camera being shifted to the right, the **Baseline** would increase equivalently to Px_d, resulting in the following equation:

$$\frac{FoV}{B} = \frac{R_h}{Px_d} \tag{3.27}$$

By solving for **FoV** we get:

$$FoV = \frac{R_h}{Px_d} \cdot B \tag{3.28}$$

The **FoV** can also be calculated using trigonometry:

$$tan\left(\frac{FoV}{2}\right) = \frac{\frac{FoV}{2}}{D} \tag{3.29}$$

Solving for **FoV** yields:

$$FoV = 2 \cdot tan\left(\frac{Fov}{2}\right) \cdot D \tag{3.30}$$

Substituting equation 3.30 inside equation 3.28 yields:

$$\frac{R_h}{Px_d} \cdot B = 2 \cdot tan\left(\frac{Fov}{2}\right) \cdot D \tag{3.31}$$

Lastly solving for **D** gives the depth to a given object in the stereo frame:

$$D = \frac{B \cdot R_h}{2 \cdot Px_d \cdot tan\left(\frac{FoV}{2}\right)} \tag{3.32}$$

Equation 3.32 is a general solution for finding the depth **D**. From this it is apparent that all the elements besides from one is known constants. Px_d is a variable, which represents the measured difference from one object in the left image to the right image. Px_d is in principal the same as **x - x'** which is the difference in coordinates of the same observed object in both the left and right image frame.

3.8.4 Camera setup parameters

Different stereo camera setups offers different characteristic. The systems parameters determines the range and sensitivity of the measurements. When deciding on a setup there are three main factors taken into consideration; FoV, camera resolution and Baseline.

The FoV helps determine the camera setups depth perception. This can be seen by manipulating equation 3.32 and making B, R_h and Px_d constant:

$$\begin{aligned} K &= \frac{B \cdot R_h}{2 \cdot Px_d} \\ D &= K \cdot \frac{1}{tan\left(\frac{FoV}{2}\right)} \end{aligned} \tag{3.33}$$

In a stereo camera with an $FoV = 90°$ the depth would be estimated to:

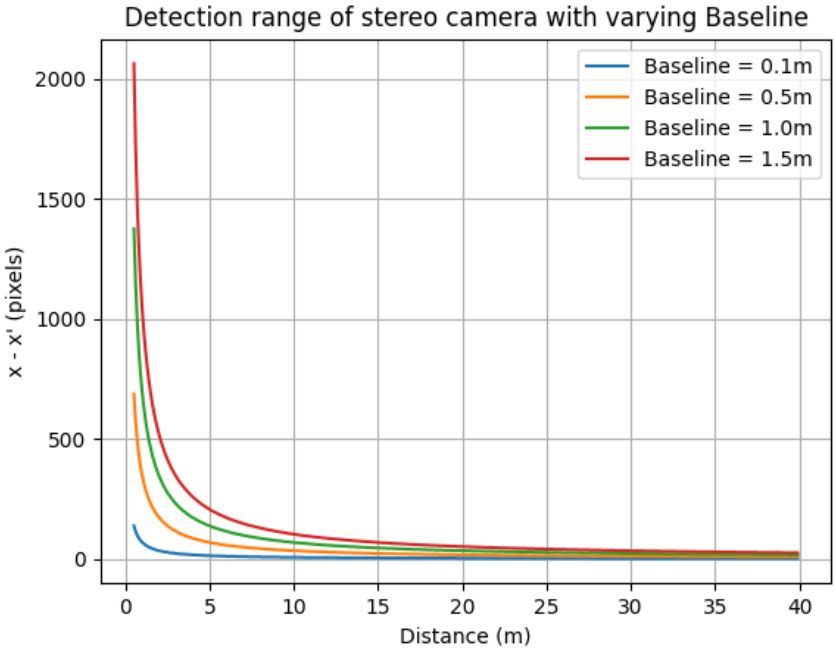
$$D = 1 \cdot K \tag{3.34}$$

On the other hand, a stereo camera with an $FoV = 40°$ the depth would be:

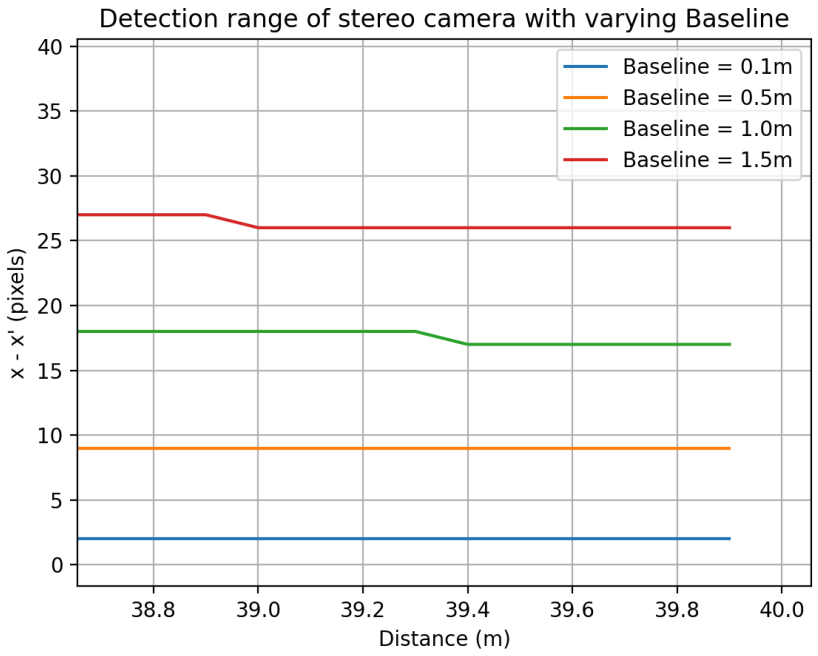
$$D \approx 2.75 \cdot K \tag{3.35}$$

Likewise, a higher camera resolution will yield a higher accuracy overall.

Lastly, the baseline affects the cameras ability to perceive depth. Firstly, a higher Baseline increases the cameras dead zone, hence limiting the ability to perceive the depth to close objects. On the other hand, a higher Baseline increases accuracy on the perceived depth on objects that are far away. This can be seen by evaluating the pixel displacement on an object with different Baselines as shown in figure 3.22a.



(a) Plot showing x - x' as a function of the depth using different baselines



(b) Zoomed in at higher depths

The plot was made with a $R_h = 2600$ and $FoV = 65^\circ$. The respective Baselines are shown in the figure.

3.9 Stereo camera for position estimation

Important note: *The coordinates **x** and **z** in this section describes the position in relation to the camera. **x** represents the linear movement to the left and right, and **z** represents the linear movement backwards and forwards. Figure 3.23 shows the directions seen from above.*

This section proposes a method to estimate a position by using various image processing techniques. This method was developed using OAK-D stereo camera (see section 3.3.1). This method is based on the assumption that the wind turbine is standing on a yellow base.

3.9.1 Finding a point of reference

As mentioned in section 2.4.8, a camera has a specific field of view. Also, an image is built up by $x \times y$ pixels which are distributed evenly in the image. Taking this into consideration, it is a constant angle between each pixel. This angle can be calculated by dividing the field of view by the total number of pixels in either x or y direction.

$$\frac{degrees}{pixel_x} = \frac{FoV_x}{n\_pixels_x} \tag{3.36}$$

$$\frac{degrees}{pixel_y} = \frac{FoV_y}{n\_pixels_y} \tag{3.37}$$

In addition to calculating the $\frac{degrees}{pixel}$, a reference point must be found and stored. Image processing is used to find that reference point. This is done by filtering out all colors in the image except for the color yellow as described in section 2.6.2. Afterwards, it is assumed that the biggest yellow object in the image will be the wind turbine, due to the placement of the camera. Therefore, the area of all the contours in the image is calculated, and the biggest contour is found. This is done by the following code using the OpenCV library in Python:

```
1 def get_biggest_contour(mask, return_area=0):
2     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
3     biggest_area = 0
4     biggest_contour = 0
5     for contour in contours:
6         area = cv2.contourArea(contour)
```

```
        if area > biggest_area:
            biggest_area = area
            biggest_contour = contour
    if return_area:
        return biggest_contour, biggest_area
    return biggest_contour
```

Source code 3.1: Method for iterating through all contours in an image and return the biggest in terms of area

The method takes a list of contours and uses a for-loop to iterate over all of the contours and returns the data of the biggest one.

Next, the center coordinates of the biggest contour is determined by the following function:

```
def get_contour_center(contour):
    # Get the moments in the contour
    M = cv2.moments(contour)
    cx=-1
    cy=-1
    # Make sure not to divide by 0
    if(M['m00']!=0):
        # Calculate x and y coordinates for the center of the contour
        cx=int(M['m10']/M['m00'])
        cy=int(M['m01']/M['m00'])
    return cx,cy
```

Source code 3.2: Method for finding the center of a contour

In the first run, the center coordinates are stored in two variables which are used as a reference of where in the image the center of the biggest contour should be. After the first run has been completed, the *pixel_offset* is calculated (See source code 3.4).

```
cx, cy = get_contour_center(biggest_contour)
# If this is the first run, the contour center coordinates is stored as a
# reference for future images
if(first_run):
    cx_ref = cx
    cy_ref = cy
```

Source code 3.3: Storing the pixel coordinates in the center of the biggest contour in the first image

3.9.2 Angle Offset

After finding the coordinates to the center of the biggest contour and storing them as references, the following images can be processed and the new coordinates to the center can be compared to the reference.

```
# Calculate the current offset compared to the reference center point
pixel_offset_x = cx_ref - cx
pixel_offset_y = cy_ref - cy
```

Source code 3.4: Calculating the pixel offset using the reference

This gives the number of pixels that the current center is displaced compared to the initial position. Now, the Angle Offset in both **x** and **y** direction can be calculated by multiplying with the $\frac{degrees}{pixel}$.

```
# Convert the pixle offset to degrees by multiplying the pixle
# offset with degrees per pixle
x_angle_offset = pixel_offset_x*deg_per_pix
y_angle_offset = pixel_offset_y*deg_per_pix
```

Source code 3.5: Calculating the angle offset

3.9.3 Calculating the position

After calculating the **Angle Offset**, the depth to the center of the contour is estimated with the stereo camera. The current depth, in combination with the angle offset is then used to calculate the current position in relation to the initial position.

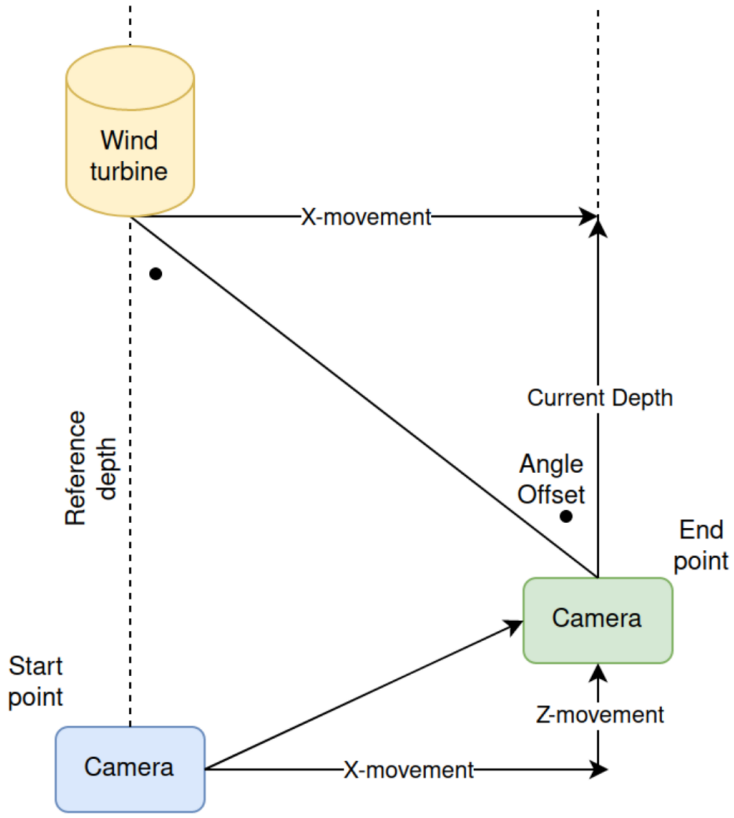


Figure 3.23: Angle offset after moving in the x- and z-direction

Figure 3.23 illustrates the directions and placement of the angle offset and x- and z-direction. The X-movement is calculated by multiplying the current depth (**D**) with the tangent to the **Angle Offset** (α):

$$\text{X-movement} = D \cdot \tan(\alpha) \tag{3.38}$$

Z-movement is calculated by first calculating the z position in relation to the wind turbine and using that position as a reference.

$$z_{ref} = D_1 \tag{3.39}$$

Where D_1 represents the calculated depth in the first image. In the following iterations the subsequent depths are subtracted from the reference depth to calculate the displacement in z-direction from the initial point.

$$\text{Z-movement} = z_{ref} - D_n \tag{3.40}$$

3.10 Creating the Graphical User Interface

To create the graphical user interface (GUI), Qt Designer was first used to create the layout. Buttons, graphs and pages was placed using this software. Figure 3.24 shows the layout of the Qt Designer software.

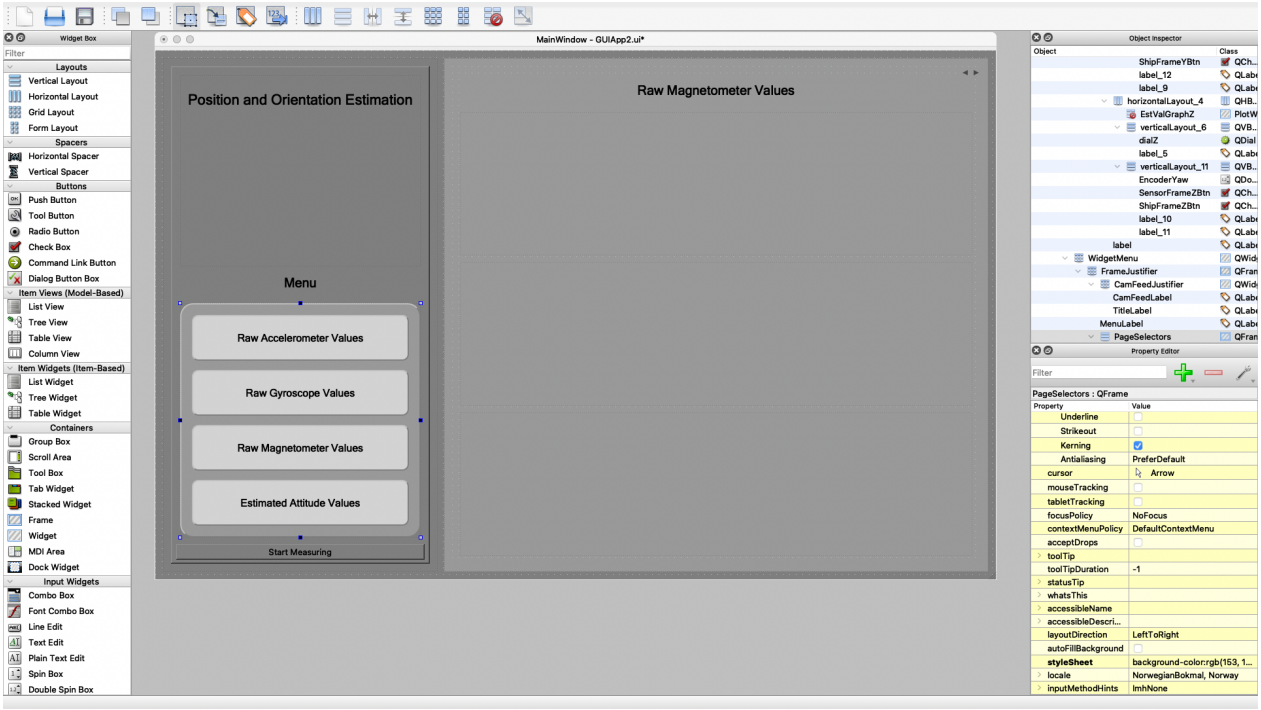


Figure 3.24: The Qt Designer interface

When the layout was finished, the application was saved as a .ui-file. The .ui file was then converted to a Python file. This was done by writing the following statement in the terminal on the computer:

```
1 pyuic5 -x GUIApp.ui > GUIApp.py
```

Source code 3.6: Converting the .ui file to a .py file

The Python file that was generated from this statement contained all code necessary to build the GUI application. However, the application did not have any functionality at this point. For example, pressing buttons did not result in any action. Therefore, the functionality of the application had to be implemented in this Python file. The code below shows how the menu buttons was connected to the different pages in the GUI.

```
# Connecting the menu buttons
self.ui.RawAccBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(0))
self.ui.RawGyroBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(1))
self.ui.RawMagBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(2))
self.ui.EstValBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(3))
```

Source code 3.7: Connecting the menu buttons to the different pages

3.11 Testing

This section explains how testing of the different solutions has been conducted.

3.11.1 Test procedure for Orientation Estimation

To test the solutions for orientation estimation, a microphone stand was used. An Arduino UNO and an Adafruit BNO055 IMU was mounted on the microphone stand (see figure 3.25). The microphone stand was chosen to test the orientation estimation because it gave the opportunity to lock the IMU in different known angles. This made it possible to verify the test results against the actual angles.

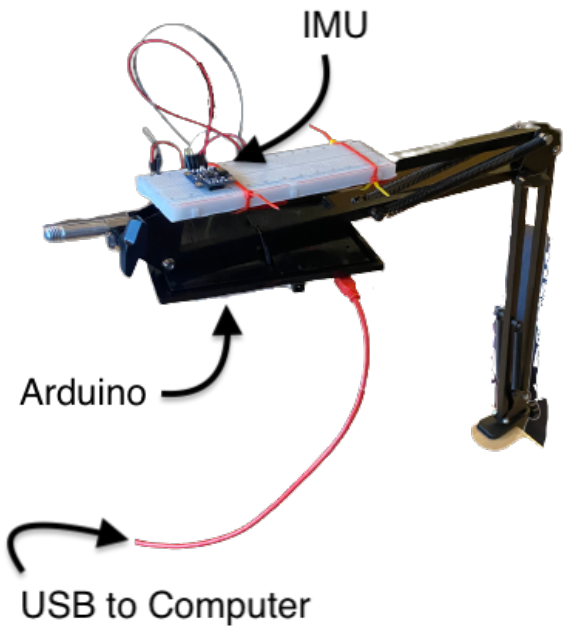


Figure 3.25: Microphone stand equipped with IMU and Arduino

The testing was done by recording the data from the IMU sensor on the Arduino, and then processing the data in the filter setups written in Python after. Three different scenarios was tested:

- 15 minutes of standing completely still
- 5 minutes standing still, followed by yawing 40°, standing still another 5 minutes, and yawing back to original position where it stands still for another 5 minutes.
- Pitching and rolling

These scenarios was tested for all three filter setups that is described in section 3.6. The results from the orientation testing is presented in section 4.2.

3.11.2 Test procedure for Position Estimation

The following sections describes the steps of testing that was performed to establish a proof of concept for the stereo camera in combination with an IMU to estimate position.

Test 1, Calculating camera angle

The angle measurement testing was performed prior to integrating the stereo camera with the IMU. This was done as a proof of concept for the theory proposed in section 3.9.1 and 3.9.2. Because the stereo camera was unable to detect any rotational movement, the stereo camera was moved along the straight edge of a table, without changing it's heading, thus removing the need to eliminate any errors in the calculated angle.

Figure 3.26 is a sketch of the test setup seen from above. The arrow X represents the movement from the starting point to the end point. The two dotted lines is the heading of the camera at the starting and end point.

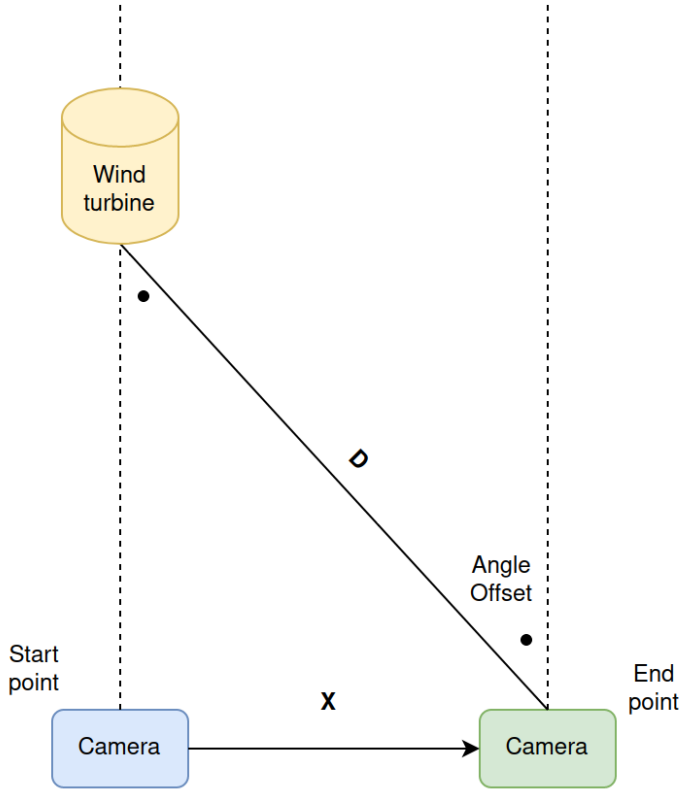


Figure 3.26: Draft of the test setup

To verify that the output was correct, an angle of approximately 10° was measured using a protractor. To verify that the angle was accurate, the length of the diagonal line D was carefully measured to $86cm$. The length of distance X was measured to $14.7cm$. The drawn angle was

then verified by calculation:

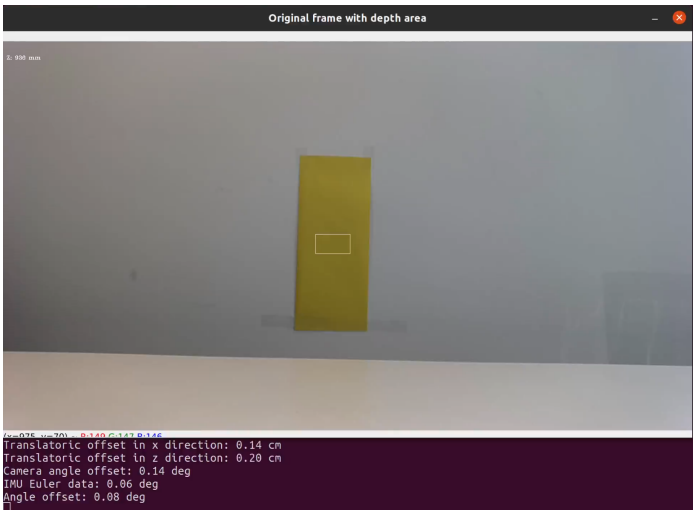
$$\begin{aligned} \arcsin\left(\frac{X}{D}\right) = \\ \arcsin\left(\frac{14.7}{86}\right) = \\ \arcsin(0.1709) \approx \underline{\underline{9.84°}} \end{aligned} \tag{3.41}$$

The results from this testing is presented in section 4.3.1.

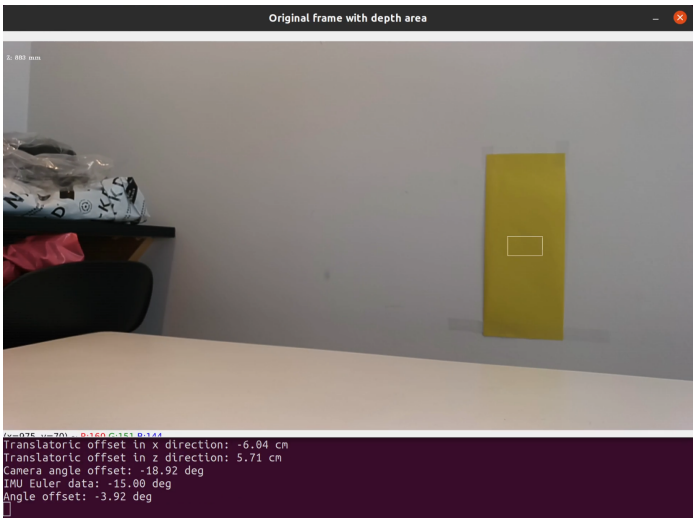
Test 2, IMU integration

The intention of this test was to evaluate whether the IMU could be used to eliminate errors in the **Angle Offset** caused by rotational movement to the sides (yaw in terms of the gangway). This was important because the camera was unable to detect rotations with the current software implementation. The purpose of the IMU was to constantly return the current heading. The IMU was configured to return a heading with the initial heading as a reference (0°).

The testing was performed by placing the camera and the IMU in front of the simulated wind turbine. The data from the IMU angle, camera angle and the calculated "true" angle was recorded while the camera was rotated in a yaw motion, while held in the same position. Figure 3.27a shows the output from when the system first was initialized. Figure 3.27b shows the output from the reading with the highest offset recorded while the camera was held still.



(a) Screenshot taken at the beginning of the test



(b) Screenshot taken of the highest angle offset

Figure 3.27: Screenshots taken during the IMU angle compensation tests

The camera was rotated back and forth to a random angle multiple times. The goal was to get an **Angle offset** $\approx 0.0^{\circ}$. If the IMU and camera was kept in the same position and the **Angle offset** ≈ 0.0, it meant that the IMU had successfully managed to compensate for the rotation. The results from this test is presented in section 4.3.2.

Test 3, Position estimation test

The purpose of this test was to evaluate if the system as a whole was able to give reliable results consistently. The test setup was measured carefully in order to be able to verify the estimated

outputs and evaluate the system. The test was performed by moving the camera and the IMU around a surface, facing the simulated wind turbine. Each of the corners in the rectangle were referred to as checkpoints. The x and z coordinates of each of the checkpoints were measured, with the reference point (Checkpoint **A**) in the origin (0, 0).

Figure 3.28 gives an overview of all the checkpoints and the distance between each point.

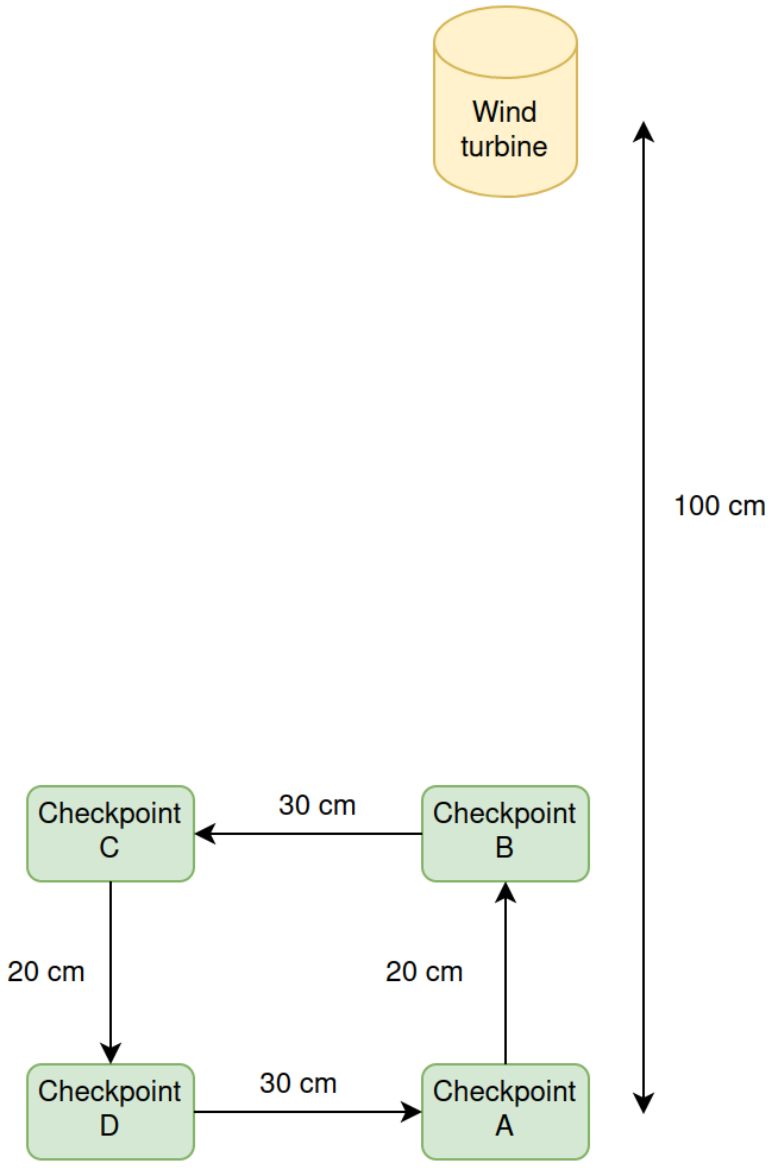


Figure 3.28: Overview of the test setup

The camera and the IMU was held together manually, to ensure none of them rotated more

than the other, to the best of ability. The IMU and camera started in checkpoint **A**, moved to checkpoint **B**, then **C**, then **D**, and lastly back to **A** to compare the final output to the initial values. The results from this test is presented in section 4.3.3.

Chapter 4

Result

This chapter presents the results from the project. First, the graphical user interface is described. This is followed by the test results from the different sensor fusion algorithms for orientation estimation. Then, the test results from the position estimation is presented. Finally, a sketch showing a model of how this can be applied on an actual gangway is described.

4.1 Graphical User Interface

The main purpose of the graphical user interface was to visualize the outputs from the orientation estimation algorithms. This made it easier to verify whether or not the estimations were good or bad, by comparing them to the physical orientation of the IMU sensor.

4.1.1 The layout

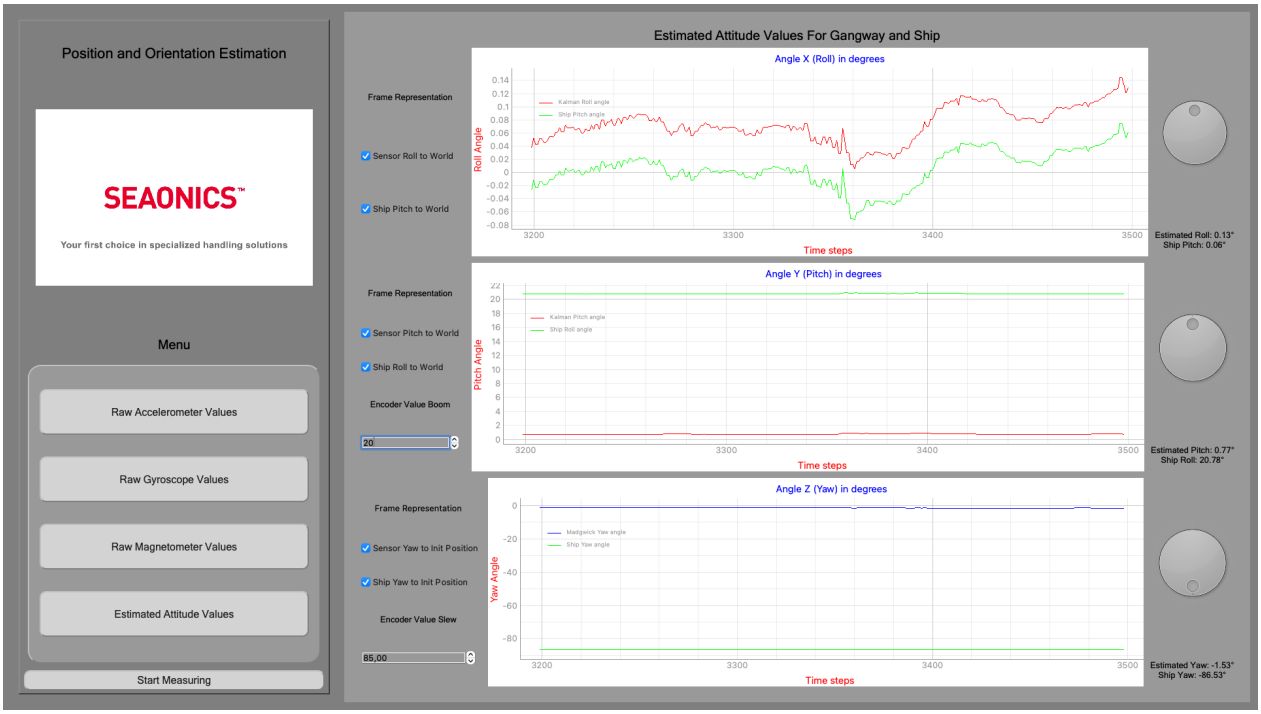


Figure 4.1: The GUI Layout

Figure 4.1 shows the layout of the GUI. It consists of four pages; one page for each sensor to show the raw values that is read from them, and a page showing the estimated angle values. It contains check-boxes which can be used to control which graphs are currently being shown. On the right side, there are 'dial-knobs' which turns according to the angles. The area in the left upper corner showing the Seaonics logo is meant to show a video stream from the stereo camera that estimates the position. This has not been implemented due to lack of time.

4.1.2 Encoders for boom and slew

To simulate the impact of the boom- and slew angles of the gangway (explained in section 3.7), there are two fields in the GUI where the values of these angles can be changed. This is to verify that the estimated angles of the ship changes according to both the measured angles of the gangway as well as the angles of the gangway in relation to the ship.

4.2 Orientation estimation results

When conducting the tests, the encoder values was set to 70° slew, and 10° boom. As seen in section 3.6, roll and pitch are estimated the same way for all three setups. The results from the tests are shown below. In the graphs, the y-axis represents the angle in degrees, and the x-axis represents the time steps. Each time step is 0.05 seconds.

Important Note: *Pitch, roll and yaw in this context means the orientation of the gangway. The green lines shows the estimated angles of the ship, which is based on the estimated angles of the gangway. Plots of the raw data from the accelerometer, gyroscope and magnetometer can be found in appendix H.*

4.2.1 Roll and Pitch results

15 minutes stationary

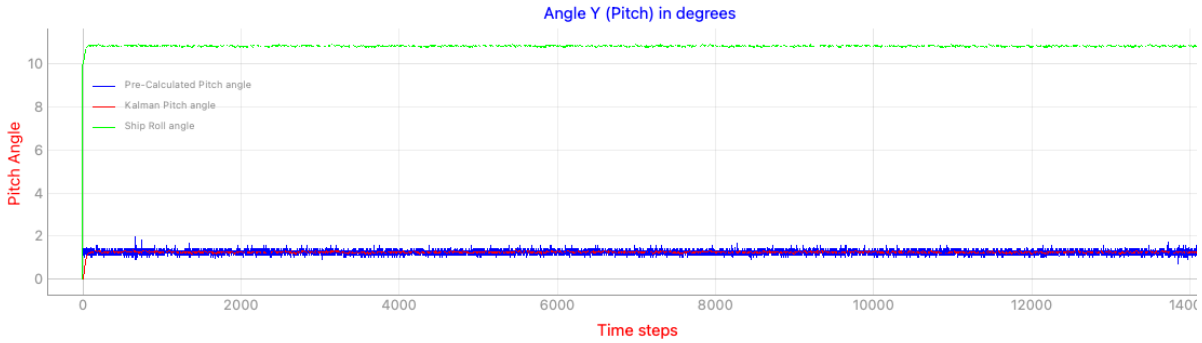


Figure 4.2: Pitch estimation while not moving

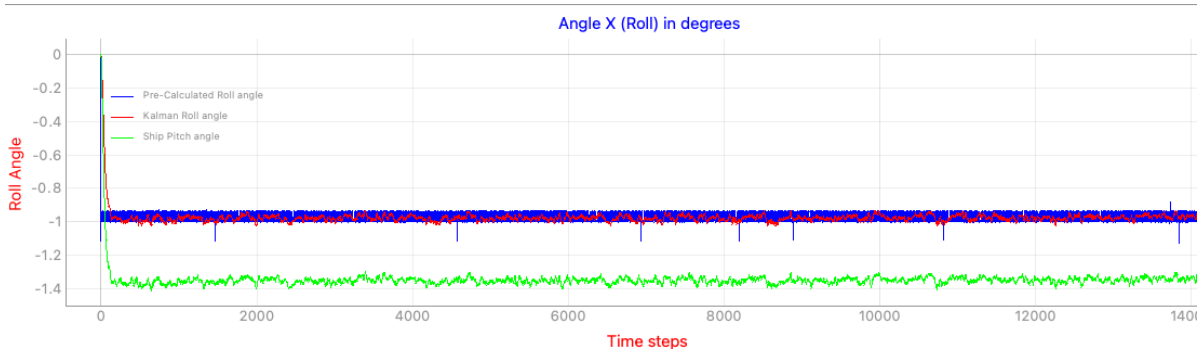


Figure 4.3: Roll estimation while not moving

15 minutes with only yaw motion

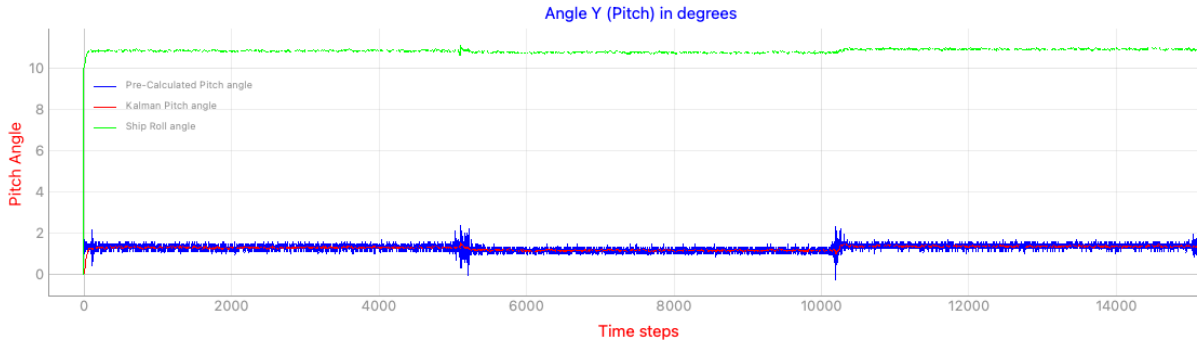


Figure 4.4: Pitch estimation while only yawing

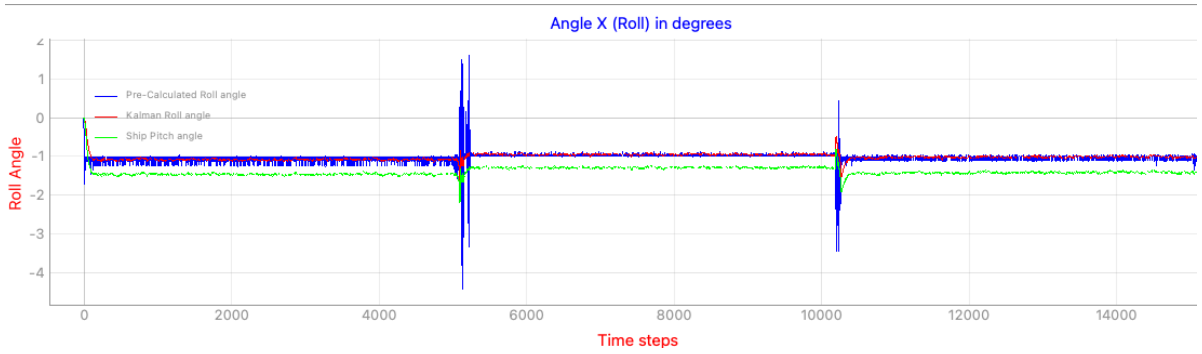


Figure 4.5: Roll estimation while only yawing

Pitching and rolling

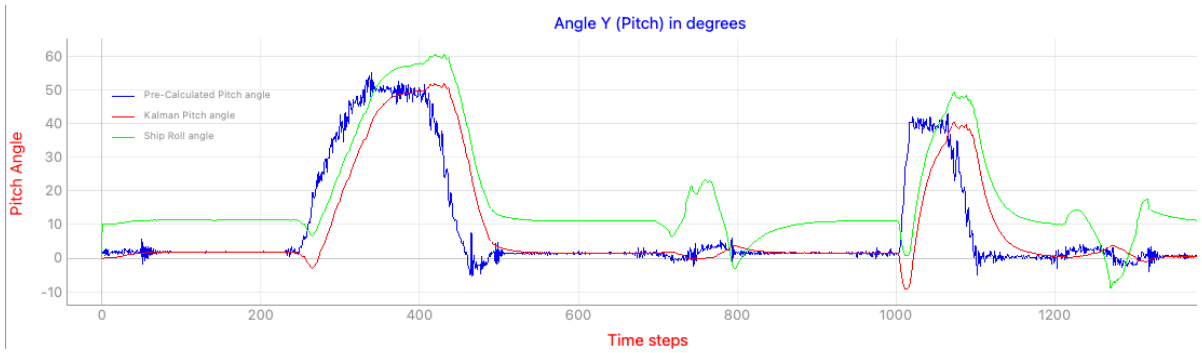


Figure 4.6: Pitch estimation while rolling and pitching

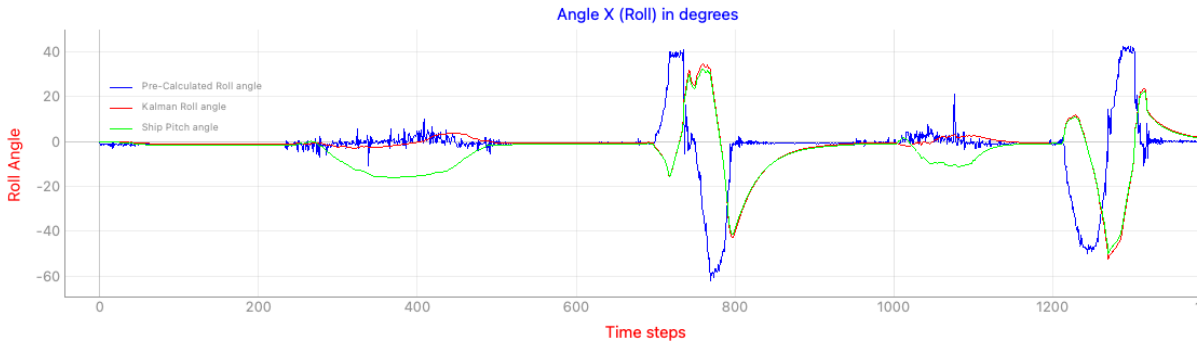


Figure 4.7: Roll estimation while rolling and pitching

4.2.2 Yaw results: Filter setup 1

15 minutes stationary

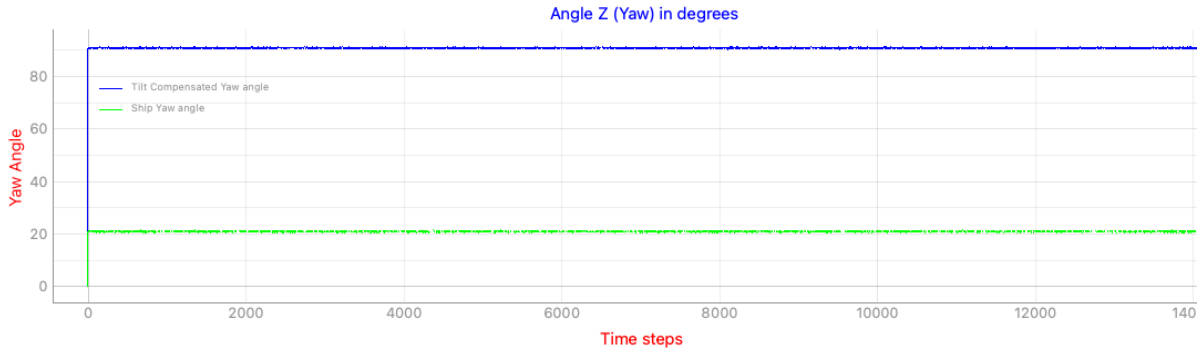


Figure 4.8: Yaw estimation while not moving, Filter 1

15 minutes with only yaw motion

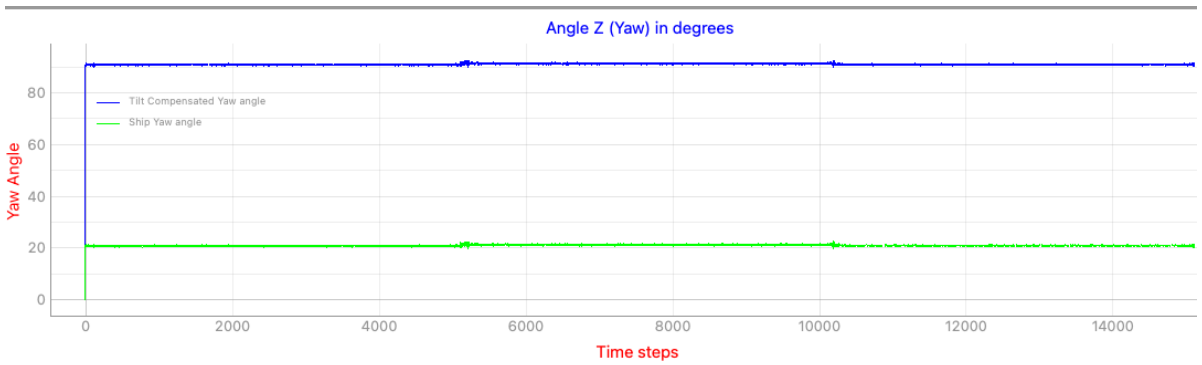


Figure 4.9: Yaw estimation while only yawing, Filter 1

Pitching and rolling

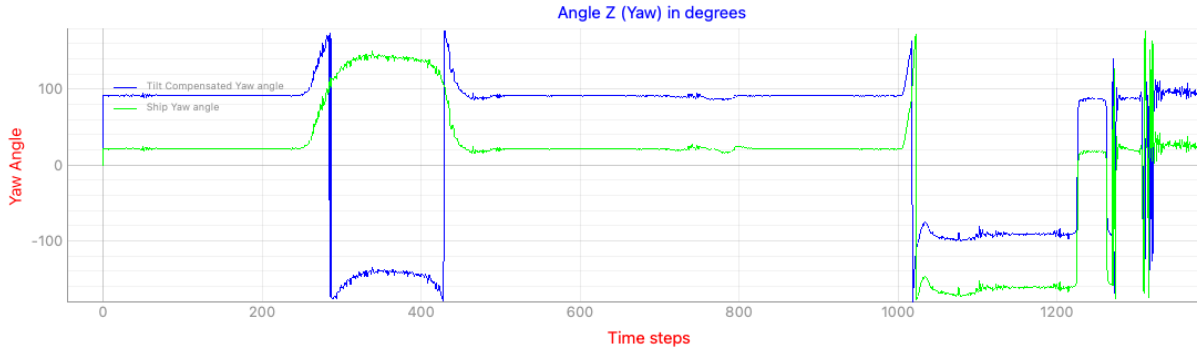


Figure 4.10: Yaw estimation while rolling and pitching, Filter 1

4.2.3 Yaw results: Filter setup 2, with magnetometer

15 minutes stationary

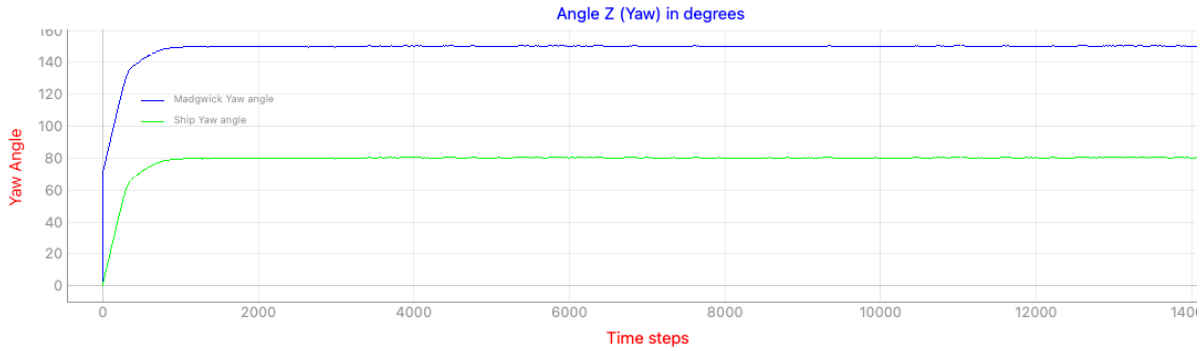


Figure 4.11: Yaw estimation while not moving, Filter 2 with magnetometer

15 minutes with only yaw motion

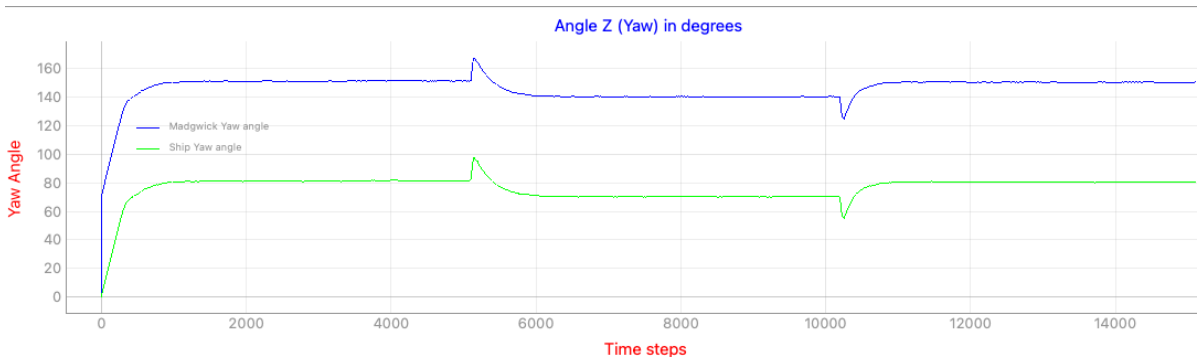


Figure 4.12: Yaw estimation while only yawing, Filter 2 with magnetometer

Pitching and rolling

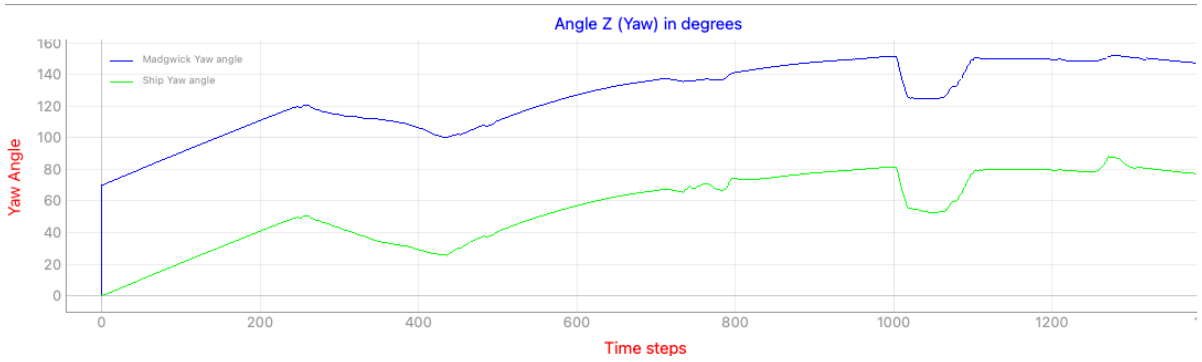


Figure 4.13: Yaw estimation while rolling and pitching, Filter 2 with magnetometer

4.2.4 Yaw results: Filter setup 2, without magnetometer

15 minutes stationary

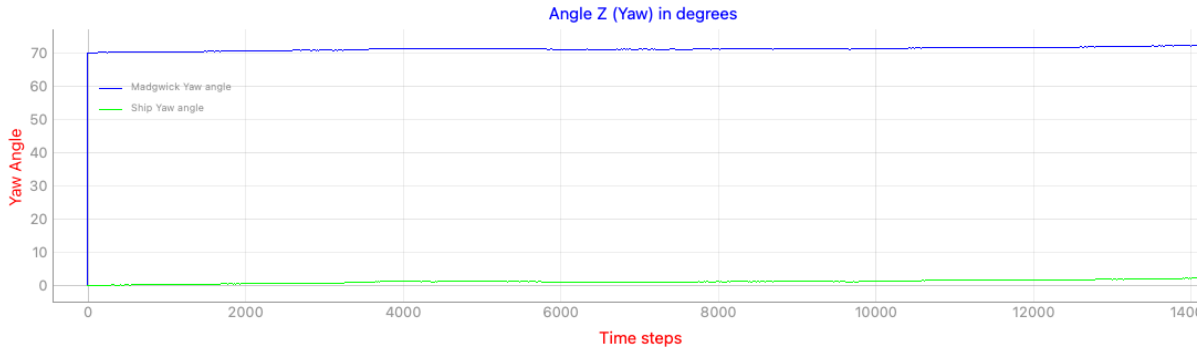


Figure 4.14: Yaw estimation while not moving, Filter 2 without magnetometer

15 minutes with only yaw motion

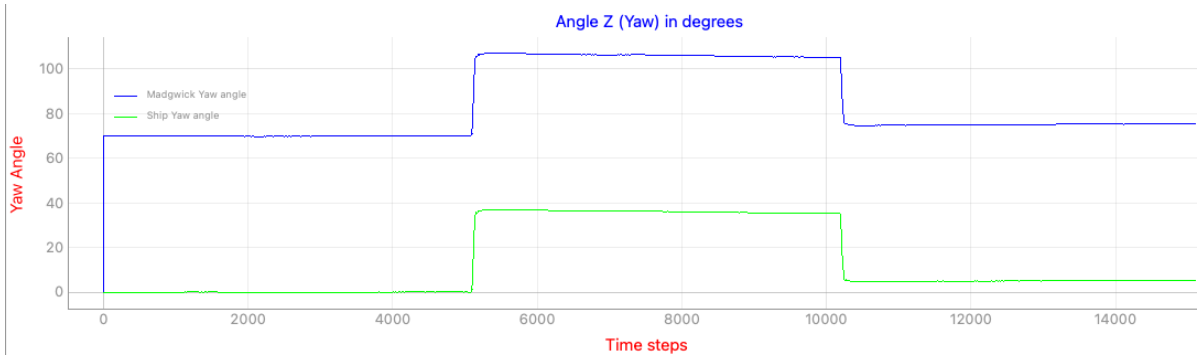


Figure 4.15: Yaw estimation while only yawing, Filter 2 without magnetometer

Pitching and rolling

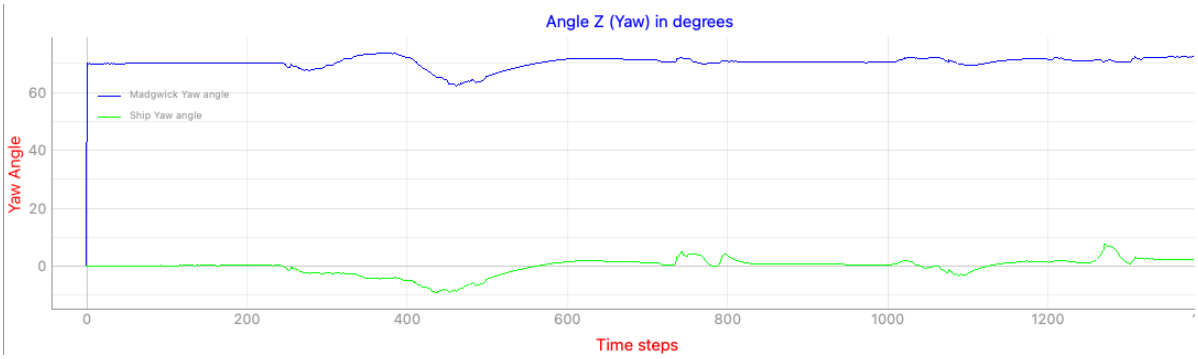


Figure 4.16: Yaw estimation while rolling and pitching, Filter 2 without magnetometer

4.2.5 Yaw results: Filter setup 3, with magnetometer

15 minutes stationary

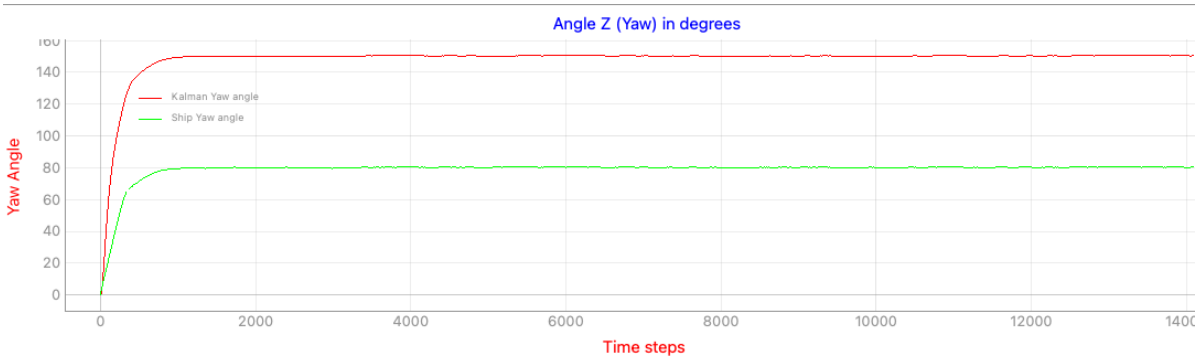


Figure 4.17: Yaw estimation while not moving, Filter 3 with magnetometer

15 minutes with only yaw motion

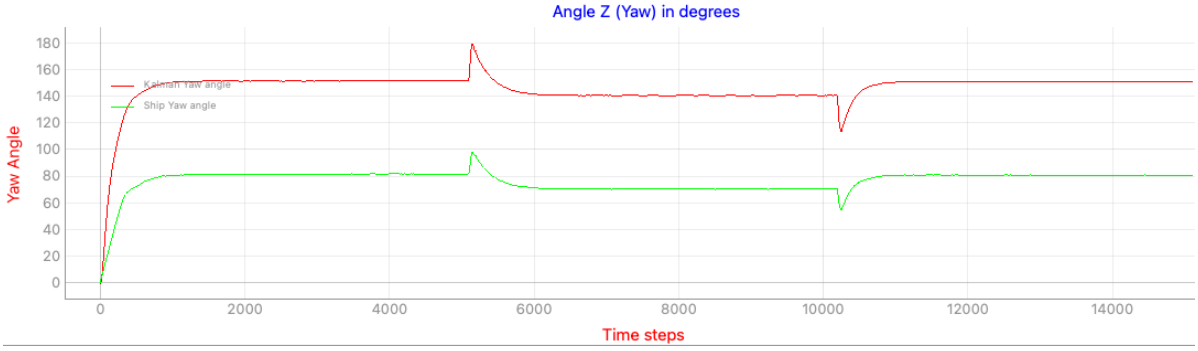


Figure 4.18: Yaw estimation while only yawing, Filter 3 with magnetometer

Pitching and rolling

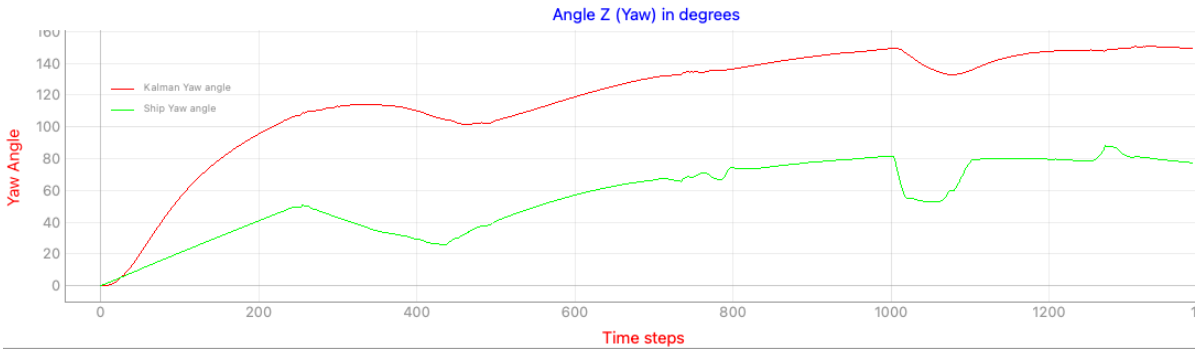


Figure 4.19: Yaw estimation while rolling and pitching, Filter 3 with magnetometer

4.2.6 Yaw results: Filter setup 3, without magnetometer

15 minutes stationary

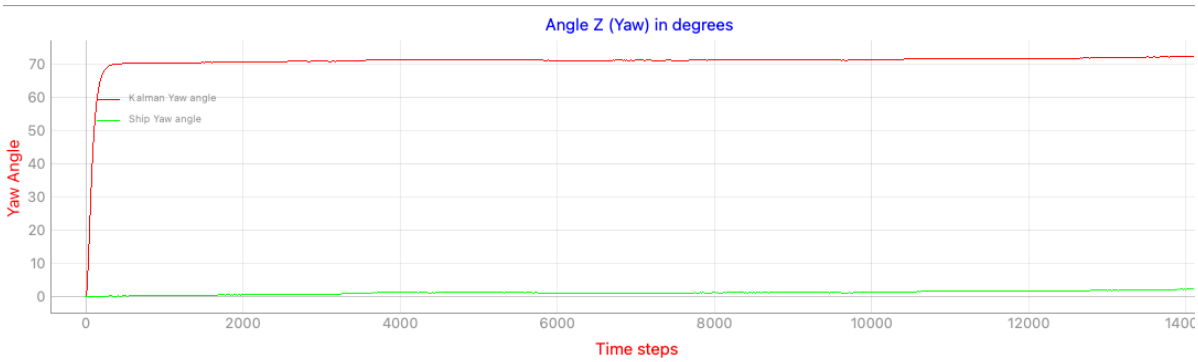


Figure 4.20: Yaw estimation while not moving, Filter 3 without magnetometer

15 minutes with only yaw motion

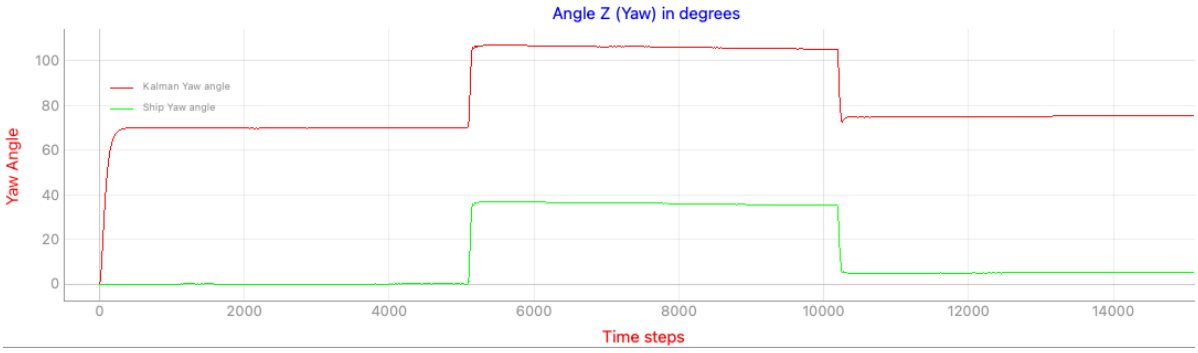


Figure 4.21: Yaw estimation while only yawing, Filter 3 without magnetometer

Pitching and rolling

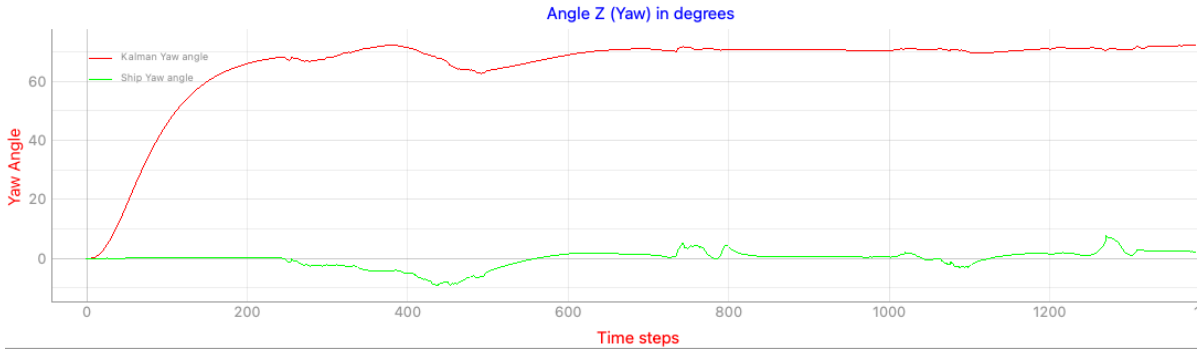


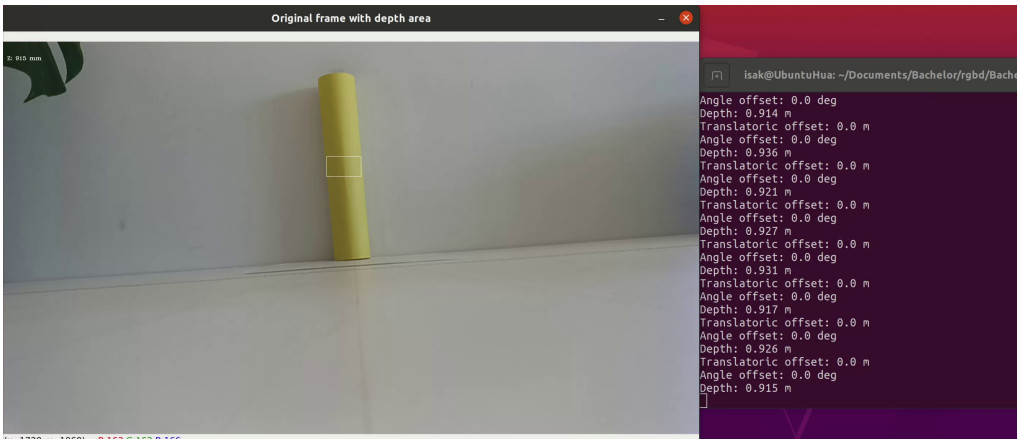
Figure 4.22: Yaw estimation while rolling and pitching, Filter 3 without magnetometer

4.3 Position estimation results

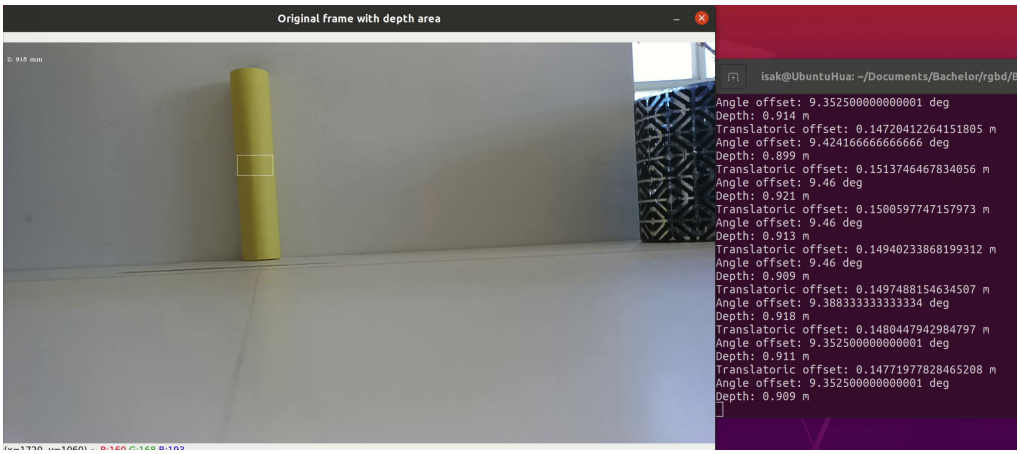
This section presents the results from the tests explained in section 3.11.2.

4.3.1 Results test 1, Calculating camera angle

Figure 4.23a shows an image that was taken at the beginning of the test with the calculated angle offset to the wind turbine. Figure 4.23b shows an image with the output after moving the camera along X (see figure 3.26). The output shows an $AngleOffset \approx 9.35°$.



(a) Screenshot taken at the beginning of the test



(b) Screenshot taken at the end of the test

Figure 4.23: Screenshots taken at the beginning and the end of the test

4.3.2 Results test 2, IMU integration

Table 4.1 shows the recorded data from the testing explained in section 3.11.2.

Test run nr.	Camera angle	IMU angle	Angle offset
Desired values	α°	α°	0.00°
1	0.14°	0.06°	0.08°
2	16.77°	15.44°	1.33°
3	-26.62°	-23.50°	-3.12°
4	-2.87°	-1.69°	-1.18°
5	12.97°	12.88°	0.09°
6	9.25°	11.25°	-2.00°
7	3.48°	5.75°	-2.27°
8	13.69°	13.56°	0.13°
9	-18.92°	-15.00°	-3.92°

Table 4.1: Test data samples from angle compensation testing.

4.3.3 Results test 3, Position estimation test

The following tables (4.2, 4.3, 4.4, 4.5 and 4.6) contains all the data recorded during the test explained in section 3.11.2 at each of the checkpoints. There were performed a total of ten tests, where the **Test run nr.** corresponds to each of the test runs. The **X** and **Z** columns shows the offset distance from the initial point in cm. The **Camera Angle**, **IMU Angle** and **Angle Offset** are the calculated angles and heading.

The desired **Angle Offset** is approximated based on the manually measured distance to each point from the wind turbine. The desired **Angle Offset** in checkpoint **B**, **C** and **D** has been calculated to 0.00°, -20.556° and -16.7° respectively. The desired **Angle Offsets** was calculated by using $tan\left(\frac{0}{100}\right)$, $tan\left(\frac{-30}{80}\right)$ and $tan\left(\frac{-30}{100}\right)$.

Checkpoint A

Test run nr.	X	Z	Camera Angle	IMU Angle	Angle offset
Desired values	0.00	0.00	0.00°	0.00°	0.00°
1	−0.07	0.00	−0.04°	0.00°	−0.04°
2	−0.46	−0.40	0.00°	0.25°	−0.25°
3	−0.39	0.10	−0.04°	−0.25°	0.21°
4	0.37	−0.10	0.14°	−0.06°	0.20°
5	0.18	0.00	0.04°	−0.06°	0.10°
6	0.02	0.40	−0.07°	−0.06°	0.01°
7	0.64	0.00	0.54°	0.19°	0.35°
8	−0.62	0.10	0.04°	0.37°	−0.33°
9	0.05	−0.20	−0.04°	−0.06°	0.02°
10	0.29	0.00	0.04°	−0.12°	0.16°

Table 4.2: Recorded data at checkpoint **A**

Checkpoint B

Test run nr.	X	Z	Camera Angle	IMU Angle	Angle offset
Desired values	0.00	20.00	0.00°	0.00°	0.00°
1	0.31	20.20	0.90°	0.69°	0.21°
2	0.78	20.08	2.22°	2.75°	−0.53°
3	−0.13	20.70	1.72°	1.81°	0.09°
4	−0.48	19.90	1.86°	2.19°	−0.33°
5	−0.55	19.90	0.75°	1.12°	−0.37°
6	0.47	20.70	1.76°	1.44°	0.32°
7	0.45	21.00	1.86°	1.56°	0.30°
8	−0.85	21.10	2.62°	3.19°	−0.57°
9	−1.27	21.40	1.15°	2.00°	−0.85°
10	0.62	20.50	0.36°	−0.06°	0.42°

Table 4.3: Recorded data at checkpoint **B**

Checkpoint C

Test run nr.	X	Z	Camera Angle	IMU Angle	Angle offset
Desired values	−30.00	20.00	−20.556°	0.00°	−20.556°
1	−29.68	20.00	−19.71°	0.50°	−19.21°
2	−30.15	20.90	−18.13°	1.44°	−19.57°
3	−30.28	19.10	−18.56°	0.75°	−19.31°
4	−30.43	19.10	−17.09°	2.50°	−19.59°
5	−30.78	19.00	−19.42°	0.25°	−19.67°
6	−29.74	20.00	−19.60°	−0.44°	−19.16°
7	−29.41	19.80	−18.17°	−0.63°	−18.80°
8	−30.78	20.30	−17.59°	2.25°	−19.84°
9	−30.07	20.20	−17.02°	2.19°	−19.21°
10	−29.93	20.00	−19.03°	0.31°	−19.34°

Table 4.4: Recorded data at checkpoint **C**

Checkpoint D

Test run nr.	X	Z	Camera Angle	IMU Angle	Angle offset
Desired values	−30.00	0.00	−16.7°	0.00°	−16.7°
1	−29.63	0.40	0.40°	−17.41°	−15.79°
2	−29.78	0.90	−15.55°	0.31°	−15.86°
3	−29.10	0.08	−16.16°	−0.63°	−15.53°
4	−30.60	0.30	−14.76°	1.50°	−16.26°
5	−30.75	−0.20	−16.34°	−0.06°	−16.28°
6	−29.19	0.90	−16.70°	−1.12°	−15.58°
7	−30.52	−0.70	−16.05°	−0.12°	−15.93°
8	−30.82	0.50	−15.15°	2.19°	−16.34°
9	−29.93	0.40	−15.87°	−0.12°	−15.75°
10	−29.54	0.30	−17.77°	−2.06°	−15.71°

Table 4.5: Recorded data at checkpoint **D**

Checkpoint A

Test run nr.	X	Z	Camera Angle	IMU Angle	Angle offset
Desired values	0.00	0.00	0.00°	0.00°	0.00°
1	0.13	0.00	−0.18°	−0.25°	0.07°
2	−0.31	0.30	−0.29°	−0.12°	−0.12°
3	−0.81	0.60	−0.07°	0.37°	−0.44°
4	−0.82	−0.30	1.61°	2.06°	−0.45°
5	−0.28	−0.50	1.04°	1.19°	−0.15°
6	0.26	0.40	−0.11°	−0.25°	0.14°
7	0.08	1.10	−0.90°	−0.94°	0.04°
8	−1.98	0.50	0.36°	1.44°	−1.08°
9	−0.64	0.10	0.47°	0.81°	−0.34°
10	0.25	0.40	−1.36°	−1.50°	0.14°

Table 4.6: Recorded data at checkpoint **A** to compare to the initial recorded data

Chapter 5

Discussion

In this chapter, the results presented in the previous chapter will be discussed and analyzed. This includes a discussion of how the current solution differs from a complete solution. This discussion is the basis of the conclusion which is found in the next chapter.

5.1 Test results

5.1.1 Orientation estimation

The test results from orientation estimation using the Adafruit BNO055 9-DOF IMU shows a few things that is worth to note.

Pitch and Roll

The combination of pre-calculating roll- and pitch-angles and fusing these angles with gyroscope measurements about the x- and y-axes trough Kalman filters yields accurate and stable, non-drifting estimations of pitch and roll. However, the current solution is a bit slow. For example, by looking at figures 4.6 and 4.7 on page 98 and 99, it is visible that the Kalman filtered angles are significantly slower than the pre-calculated angles. From the graphs it seems that the filtered angles are approximately 1 second slower than the pre-calculated angles. This can probably be improved by tuning the parameters of the Kalman filters. The process matrix, **Q**, which represents the uncertainty in the model, is a parameter which has to be tuned through

experimenting. If this parameter is tuned optimally, the accuracy and filter speed can probably increase. Another aspect that is worth noting is the sampling time. In the tests, the sampling time was set to 50ms. By decreasing the sampling time, the Kalman filters would always work with more "fresh" samples, which in turn would increase the speed of the filtering.

Yaw

The yaw-angle is the most difficult angle to estimate. This is because accelerometers only give information that can be used to accurately estimate pitch and roll, as explained in section 3.4.2. Therefore, different combinations of accelerometer, gyroscope, magnetometer and different sensor fusion algorithms was applied to find the best solution.

In theory, using a magnetometer should provide information that can be used to find the absolute orientation of the sensor using a Madgwick filter, as explained in section 2.5.5. This means that it should be possible to find true North using this technique. However, the test results does not verify this. The test results show that including the magnetometer in any of the filter setups yields inaccurate estimations of yaw.

In Filter Setup 1, where the yaw angle is estimated by taking the estimated roll- and pitch-angles and fusing them with magnetometer data in a tilt-compensation algorithm, the estimated yaw angle is completely unreliable. When performing a yaw-rotation on the sensor, the estimated yaw-angle does not change. However, when performing pitch- and roll-rotations, the estimated yaw changes accordingly. A reason for this can be that the mathematics behind it is not correct.

In Filter Setup 2, where the yaw angle is estimated by using a Madgwick filter, the estimated yaw angle is reliable and stable when the magnetometer is not included. However, when the magnetometer is included in this setup, the estimates becomes unreliable. By looking at figures 4.11, 4.12 and 4.13 at page 100 - 101, a few things are worth to note. When yawing 40°, the filter only registers a 10° motion. When only pitching and rolling, without having any significant motion in yaw rotation, the estimated yaw angle is affected and it seems that the estimate "dips" each time roll- or pitch-motion is registered. By looking at the results from Filter Setup 2 where the magnetometer is not included (figures 4.14, 4.15 and 4.16 at page 101 - 102), it is clear that these results are more accurate and stable. The estimated yaw is accurate and it does not drift.

The estimate varies slightly when pitching and rolling, but the reason for this could be that when pitching and rolling the sensor, the sensor was also rotated slightly about the z-axis.

Filter Setup 3 yields the same characteristics as Filter Setup 2. The difference is that the output from the Madgwick filter is fed through a Kalman filter. The results from this solution shows that this leads to slower estimates. As the Madgwick filter in itself produces stable and accurate estimates (when the magnetometer data is not included), running the output through a Kalman filter seems to be unnecessary. The estimates becomes slightly more stable, but they are also slower.

It is interesting to see that utilizing magnetometer data in the filter setups actually makes the estimates more unreliable. The most explaining reason for this may be that the stand on which the sensor is mounted is made of magnetic metal. This can disturb the magnetic field around the magnetometer, and thus the estimates. Using this solution in a real industrial installation would potentially yield the same disturbances, because the offshore gangway can be made out of steel. Additionally, the IMU sensor would most likely need to be installed inside a water-tight box. This box has to be made of a material that is not weakened by sunlight and saltwater. Therefore, the box should be made of aluminum or stainless steel. An advantage of this is that these types of metal are not magnetic, and will probably not disturb the magnetic field around the sensor in any significant way.

However, it seems that a solution only dependent on a gyroscope and an accelerometer would yield accurate estimates of orientation. The current solution does not verify that a magnetometer can increase the accuracy. In fact, the results from testing indicates the opposite, but a theory is that this is because of the magnetic structure the sensor is mounted on.

5.1.2 Position estimation

Test 1, Calculating camera angle

Looking at the results from test 1, we can see that the estimated **Angle Offset** is approximately 9.35°. By comparing this to the measured desired **Angle Offset** from equation 3.41, we get that the error in the estimation is:

$$9.84^\circ - 9.35^\circ = 0.49^\circ \tag{5.1}$$

This assumes that the desired **Angle Offset** was correctly measured and that the camera was stopped in the correct position. As it was difficult to measure the distances and angles with a high precision, there are many sources of error.

After looking at the results, it was concluded that this method is valid and gives sufficient estimations.

Test 2, IMU integration

The goal of this test was to verify that the IMU can be used to correct the cameras angle by subtracting the IMU angle from the calculated camera angle. In the results, it is seen that the **Angle Offset** varies. The least desired **Angle Offset** recorded is estimated to -3.92° in test nr 9, while the most desired **Angle Offset** is estimated to 0.08° in test run nr 1. The results in general are inconsistent. One of the two main factors the may affect the results are that the IMU and the stereo camera was not moved aligned. The other being that the IMU and the stereo camera was not kept in a fixed location during the testing. In other words, linear movement is likely to have occurred during the rotation.

Evaluating the test, the group finds the results satisfying. Yet, the method will require further and more accurate testing in order to give a solid conclusion based on proper data, collected under a controlled environment. Based on the results, the group considers the IMU as a reliable way to compensate for errors in the angle offset estimated by the camera.

Test 3, Position estimation test

By comparing the desired values to the estimated values, it is possible to see that the estimated values are close to the desired output. The results are consistent and accurate.

To illustrate the margin of error when positioning the system in the different checkpoints, the system was returned to its starting position (checkpoint **A**) after each test run. In theory, the values in table 4.2 should be equivalent to table 4.6. However, due to the testing facilities and the system being maneuvered by hand, returning to the exact position each test run was

unachievable. This is considered the largest source of error throughout this test.

Table 5.1, 5.2 and 5.3 contains the average, average offset, standard deviation and variance of the data.

Checkpoint B	**X**	**Z**	**Angle Offset**
Desired value	$0.00cm$	$20.00cm$	$0.00°$
Average	$-0.065cm$	$20.548cm$	$-0.131°$
Average offset	$0.065cm$	$0.548cm$	$0.131°$
Standard Deviation	$0.6599cm$	$0.4963cm$	$0.4270°$
Variance	$0.4355cm^2$	$0.2463cm^2$	$0.1823°^2$

Table 5.1: Analytical data based on table 4.3

Checkpoint C	**X**	**Z**	**Angle Offset**
Desired value	$-30.00cm$	$20.00cm$	$-20.556°$
Average	$-30.125cm$	$19.84cm$	$-19.37°$
Average offset	$0.125cm$	$-0.16cm$	$-1.186°$
Standard Deviation	$0.4329cm$	$0.5783cm$	$0.2870°$
Variance	$0.1874cm^2$	$0.3344cm^2$	$0.0824°^2$

Table 5.2: Analytical data based on table 4.4

Checkpoint D	**X**	**Z**	**Angle Offset**
Desired value	$-30.00cm$	$0.00cm$	$-16.7°$
Average	$-29.986cm$	$0.2880cm$	$-15.903°$
Average offset	$-0.0139cm$	$0.2880cm$	$-0.7970°$
Standard Deviation	$0.6108cm$	$0.4557cm$	$0.2793°$
Variance	$0.3731cm^2$	$0.2077cm^2$	$0.0780°^2$

Table 5.3: Analytical data based on table 4.5

The analytical data in the tables above shows satisfying results, with a low average offset and variance and a high consistency. The largest errors is displayed in checkpoint **C** and **D**. This may

be a result of the ROI being inconsistent on the wind turbine, yet, this should be tested further in a more controlled environment with the stereo camera and IMU fixed to eachother. Additionally, solutions to ensure that the system is placed in the same position in each run should be implemented for the testing.

5.2 Placement of sensors in a real installation

In order for the stereo camera to be able to capture the yellow base on the wind turbine, the stereo camera should be placed close to the tip of the gangway. Additionally, in order to avoid obstructions from, for instance, people walking in front of the camera view, the stereo camera should be attached underneath the gangway. The IMU can be placed in the same area. Figure 5.1 illustrates the sensor placement.

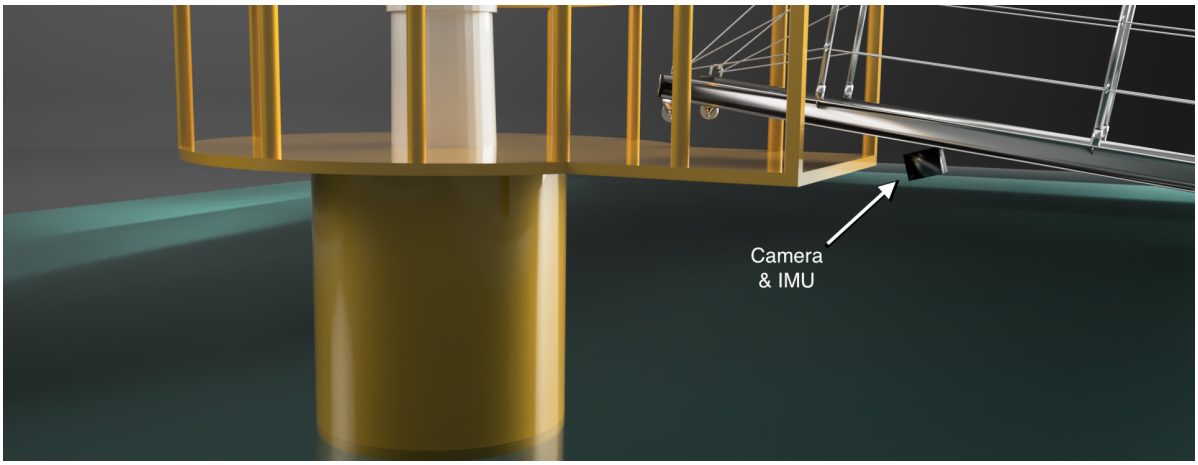


Figure 5.1: Image of how the sensors can be placed

Note: *The illustration above is not an accurate representation of the reality. The figure is based on information that was given to the group and assumptions made based on research during the project.*

5.3 Future considerations on system integration

Due to lack of time, there are a few things that has not been implemented in the current solution. This section explains what should be implemented to achieve a complete solution.

5.3.1 Two systems that should be merged together

The current solution offers estimates of orientation and position in two separate systems. The GUI application currently only includes orientation estimation. The plan was to include a video-stream showing the captured images from the stereo-camera, and also show the estimated position of the gangway relative to the wind turbine.

The position estimation system should utilize data from the orientation estimation system to reference the position estimates in relation to the orientation. Also, the current solution only estimates relative position in x- and y-direction (in relation to the gangway). Motion along the z-axis is not taken into account, as it is difficult for a stereo-camera to register displacement along the z-axis when the reference object is a cylinder-shaped wind-turbine base with the current system.

5.3.2 Suggestions for further work

PID Controlled Servo for Camera Control

One of the biggest limitations to the current position estimation solution is when the ROI (region of interest) is on the edge of the image. This may cause some of the ROI not to be captured by the camera, hence the center point will not be in the same location. Even if Seaonics' AMC system will constantly work towards keeping the **Angle Offset** = 0, this is an error that may occur and cause faulty data.

A suggested solution to this problem is to use a PID regulated servo. By attaching the camera to the servo, the **Angle Offset** of the camera can be used as feedback to the PID-regulator, which always works towards keeping the **Angle Offset** = 0. This is a quick system which will significantly decrease the chances of the ROI falling out of the image. The **X** and **Z** position will then be calculated by the **Angle Offset** recorded by the encoder in the servo, rather than directly from the Image Processing.

VSLAM/Stereo VIO

One alternative to improve the position estimation is by using VSLAM or Stereo VIO (visual odometry) based algorithms [80], or a tailored version of this. It is apparent that the method

used to visually localize an object can be some what inaccurate and unreliable. Selecting the ROI (region of interest) based on the center of an object defined by a color can leave room for error. As an alternative, a form of visual odometry is proposed. VSLAM is based on the same principles as humans have used for navigation for centuries. "By recognizing special constellations or specific stars, people could find North (i.e. direction) as well as calculate their own latitude and longitude (i.e. location), usually using look-up tables" [81]. The current system selects one ROI. An alternative approach is to create a system where the visual odometry is based on several points spread out in the image, which creates a more robust system and eliminates the "single point of failure". This can be done by utilizing feature detection algorithms [82]. Combining this with the information from a stereo camera or 3D camera, it is possible to derive the odometry of the observer relative to the observed targets, as well as calculating velocity. This is due to having the depth measurement to each feature, which can be compared from one image to the next, and thus get the displacement in the image, which can be used to derive movement. Similar systems yields high accuracy, as in the case of the Intel RealSense T265 Tracking Camera with drift as low as under one percent (based on indoor testing) [59]. In other words, a visual odometry based system which uses multiple points for orientation has the potential to be more robust. It does not depend on a single color or feature, nor does it only depend on a single point.

Include position estimation in the GUI

As mentioned in section 5.3.1, the position estimation system should be included in the graphical user interface. A video-stream showing the captured images from the stereo camera could be implemented in the GUI to give the operator a better overview of the situation. The position data should also be presented in the GUI. Also, the stereo camera system should utilize the orientation estimates to account for angular displacement.

Test the solution on an actual offshore gangway

In this project, all tests has been small-scale indoor tests. This has provided useful information about the solutions, but it has to be tested on an offshore gangway in order to verify that the solution is good. This has unfortunately not been possible in this project.

Ultra-wide band for position tracking

For this project, placing any external devices on the actual wind turbine should be avoided. This is because the wind turbines are not standardized and Seaonics wants a general solution where they are not dependent on equipment placed on the wind turbine.

Considering a scenario where external devices could be placed on the wind turbines, Ultra Wide Band (UWB) positioning system could have been used. UWB sensors are high bandwidth radio frequency sensors. These sensors can provide accurate positioning estimation by utilizing time difference of arrival (TDOA) between a tag and several anchors [21] (see figure 5.2). When using TDOA, the tag sends out a signal which three or more anchors receive. By timing the signal, a precise location estimation can be calculated [83]. According to Dädeby et al., at 20m between anchor and tag, an average value of 19.88m was recorded with a standard deviation of 15.35cm [84]. A tag in this case would be placed on the wind turbine.

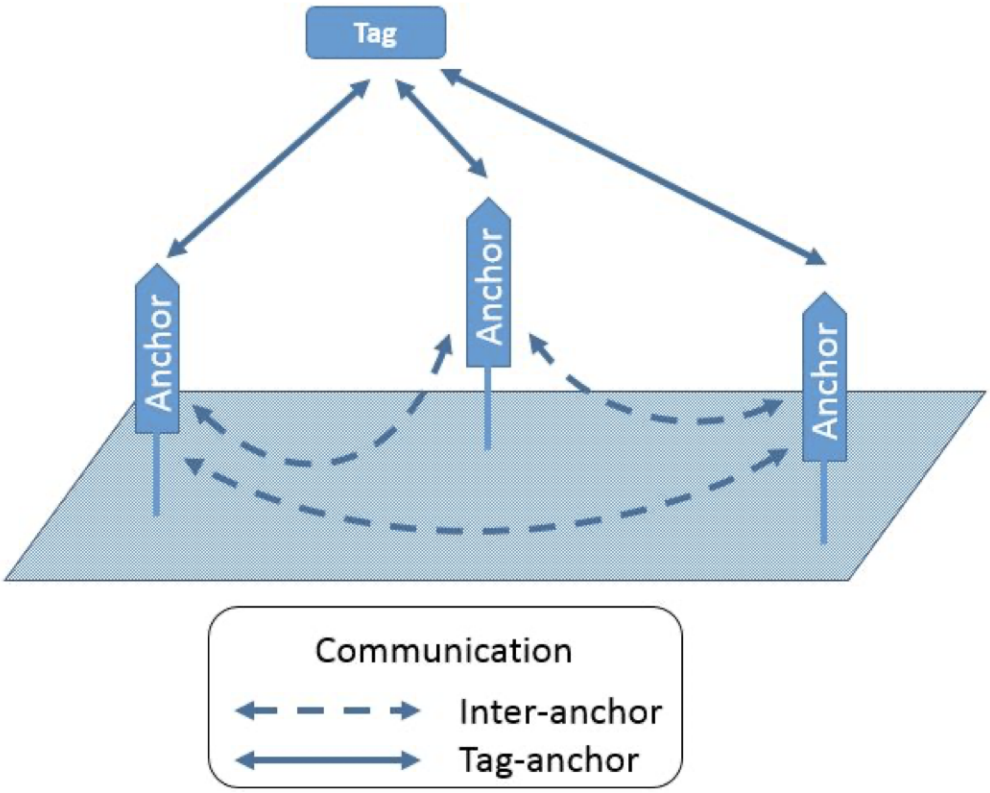


Figure 5.2: Illustration showing principles of Ultra Wide Band locating system [21]

Chapter 6

Conclusions

The problem to be addressed in this project was how to estimate the position and orientation of an offshore gangway in relation to a ship and an offshore wind turbine. Seaonics delivers offshore gangways with active motion control (AMC), and their control system can take great benefit from reliable estimates of position and orientation.

Algorithms for estimating position using stereo-imaging and algorithms for estimating orientation by utilizing a gyroscope, accelerometer and magnetometer have been described and tested. The work performed in this project has provided a basis for solutions that can contribute to the further development of an estimation system for position and orientation. Although the result of the project is not a solution that can be directly integrated in Seaonics' control system at this time, it can contribute as a good starting point in the development of an improved motion compensation system for offshore gangways.

One of the most important experiences is that the use of a magnetometer can lead to unreliable estimates of orientation, especially if there are magnetic interference in the proximity of the sensor. From the results of the filter setups for orientation estimation, the conclusion is that Filter Setup 2, without magnetometer, is the best solution. The tests of the stereo-camera shows that it can be useful to estimate linear displacement in the horizontal plane, but linear motions in the vertical plane is hard to estimate using this method. However, it has potential of improvement with further development. The solutions are currently working as two independent systems, and a complete solution should include both systems as one. That way, the position estimation system can utilize data from the orientation estimation system to take an-

gular displacement into account when estimating position.

The project has provided the group with useful experience in planning and conducting projects. It has linked many of the various topics through three years of study. Additionally, the project has provided the group with a lot of new knowledge, not only in the field of automation, but also knowledge that might not have been gained otherwise. Sensor fusion algorithms and image processing are disciplines that were quite unknown to the group before this project was carried out.

6.1 Further work

- Mount the stereo-camera on a PID controlled servo. This can help the camera to always be aimed towards the region of interest.
- Look into the possibilities of VSLAM/Stereo VIO. This, in combination with the stereo-camera, can increase the precision. It will also make it possible to estimate vertical motion.
- Include position estimation in the GUI, as well as utilize the orientation estimates in the position estimation to take angular displacement into account.
- Test the solution on an actual offshore gangway to verify if the solution works in the real world.
- Investigate how the ultra-wide band technology could be used for position tracking. This would require a tag to be placed on the wind turbine, which should be avoided, but the technology is interesting and could prove to be useful in a case like this.

Bibliography

[1] Sisi Wang, Lijun Wang, Zixuan Qiao, and Fengshan Li. Optimal robust control of path following and rudder roll reduction for a container ship in heavy waves. *Applied Sciences*, 8(9), 2018.

[2] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, Cambridge University Press, University Printing House, Shaftesbury Road, Cambridge, CB2 8BS, United Kingdom, 2017.

[3] Jeremiah van Oosten. Understanding quaternions. https://www.3dgep.com/understanding-quaternions/. Accessed 16 Feb. 2021.

[4] John Vince. *Quaternions for Computer Graphics*. Springer-Verlag London, 6 Floor London, WC1X 8HB, United Kingdom, 2011.

[5] Aidan Lyon. Why are normal distributions normal? https://aidanlyon.com/normal_distributions.pdf. Accessed 30 Apr. 2021.

[6] Watson Gyro. Vibrating structure gyro (vsg) principles of operation. https://watson-gyro.com/support-service/legacy-products/vibrating-structure-gyro-vsg-principles-of-operation/. Accessed 8 May 2021.

[7] Rob O'Reilly, Alex Khenkin, and Kieran Harney. Sonic nirvana: Using mems accelerometers as acoustic pickups in musical instruments. Accessed 9 May 2021.

[8] Alberto Naranjo. Building a sonar sensor array with arduino and python. https://towardsdatascience.com/

building-a-sonar-sensor-array-with-arduino-and-python-c5b4cf30b945. Accessed 8 May 2021.

[9] Apogeeweb. Deep analysis of infrared sensor. https://www.apogeeweb.net/electron/Deep-Analysis-of-Infrared-Sensor.html. Accessed 8 May 2021.

[10] Jon Chouinard. What are the benefits of cmos based machine vision cameras vs ccd? https://www.1stvision.com/machine-vision-solutions/2019/07/benefits-of-cmos-based-machine-vision-cameras-vs-ccd.html. Accessed 8 May 2021.

[11] Teledyne Princeton Instruments. Field of view and angular field of view. https://www.princetoninstruments.com/learn/camera-fundamentals/field-of-view-and-angular-field-of-view. Accessed 8 May 2021.

[12] Alice Matthews. Advances in next-gen camera module circuit design. https://www.electronicspecifier.com/products/artificial-intelligence/advances-in-next-gen-camera-module-circuit-design. Accessed 9 May 2021.

[13] Stefan Winkler. Stereo triangulation from a pair of cameras. https://www.researchgate.net/figure/Stereo-triangulation-from-a-pair-of-cameras_fig4_215482736. Accessed 8 May 2021.

[14] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf. Accessed 3 May 2021.

[15] Jim Cahill. Improving oil and gas exploration performance. https://www.emersonautomationexperts.com/2018/industry/oil-gas/improving-oil-gas-exploration-performance/. Accessed 10 May 2021.

[16] Luxonis. Luxonis oak-d depthai stereo camera. https://www.antratek.com/luxonis-oak-d-depthai-hardware. Downloaded 7 May 2021.

[17] Intel. Intel® realsense™ tracking camera t265. https://www.intelrealsense.com/tracking-camera-t265/. Downloaded 7 May 2021.

[18] Amazon. Adafruit 9-dof absolute orientation imu fusion breakout - bno055. https://www.amazon.ca/Adafruit-Absolute-Orientation-Fusion-Breakout/dp/B017PEIGIG. Downloaded 7 May 2021.

[19] Arduino. Arduino uno rev3. https://store.arduino.cc/arduino-uno-rev3. Downloaded 7 May 2021.

[20] Komplett. Svive hydra studioarm. https://www.komplett.no/product/941659/datautstyr/pc-tilbehoer/streaming/streaming-tilbehoer/svive-hydra-studioarm?feature=freightwidget. Accessed 9 May 2021.

[21] Krzysztof Cisek, Artur Zolich, Kristian Klausen, and Tor Arne Johansen. Ultra-wide band real time location systems: Practical implementation and uav performance evaluation. In *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 204–209, 2017.

[22] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. JOHN WILEY and SONS, Ltd., John Wiley and Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2011.

[23] Regjeringen. Norway leading maritime nation. https://www.regjeringen.no/contentassets/05c0e04689cf4fc895398bf8814ab04c/maritim_strategi_engelsk_trykk.pdf. Accessed 20 Apr. 2021.

[24] R.E. Kalman. A new approach to linear filtering and prediction problems. https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf. Accessed 29 Apr. 2021.

[25] Randall Dewey Knight. *Physics for scientists and engineers: a strategic approach with modern physics*, page 54–86. Pearson, 2017.

[26] Dennis S. Bernstein. *Geometry, kinematics, statics and dynamics*. Princeton Univeristy Press, 41 William St, Princeton, NJ 08540, USA, 2012.

[27] Marcelo H. Ang and Vassilios D. Tourassis. Singularities of euler and roll-pitch-yaw representations. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(3):317–324, 1987.

[28] Jose Luis Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization. 09 2010.

[29] Satya Mallick. Rotation matrix to euler angles. https://learnopencv.com/rotation-matrix-to-euler-angles/. Accessed 12 May 2021.

[30] Frede Frisvold and Jan Gunnar Moe. *Statistikk for ingeniører*. Fagbokforlaget, Postboks 6050 Postterminalen, 5892 Bergen, 2004.

[31] Store Norske Leksikon. Normalfordeling. https://snl.no/normalfordeling. Accessed 30 Apr. 2021.

[32] Merriam-Webster.com Dictionary. Sensor. https://www.merriam-webster.com/dictionary/sensor. Accessed 12 Feb. 2021.

[33] Sparkfun. Gyroscope. https://learn.sparkfun.com/tutorials/gyroscope/all. Accessed 27 Jan. 2021.

[34] Epson Device Blog Team. Gyro sensors - how they work and what's ahead. https://www5.epsondevice.com/en/information/technical_info/gyro/. Accessed 27 Jan. 2021.

[35] Steven C. Frautschi. *The mechanical universe: mechanics and heat*. Cambridge University Press, 2008.

[36] Store Norske Leksikon. Magnetometer. https://snl.no/magnetometer. Accessed 8 May 2021.

[37] Fierce Electronics. What is an accelerometer? https://www.fierceelectronics.com/sensors/what-accelerometer. Accessed 22 Apr. 2021.

[38] Banner Engineering Corp. Radar sensors. https://www.bannerengineering.com/us/en/products/sensors/radar-sensors.html?pageNum=1&#all. Accessed 27 Jan. 2021.

[39] Baumer A/S. Functionality and technology of radar sensors. https://www.baumer.com/dk/en/service-support/function-principle/functionality-and-technology-of-radar-sensors/a/Know-how_Function_Radar-sensors. Accessed 27 Jan. 2021.

[40] Arrow Electronics. Introduction to bayer filters. https://www.arrow.com/en/research-and-events/articles/introduction-to-bayer-filters. Accessed 8 May 2021.

[41] Chris Woodford. Digital cameras. https://www.explainthatstuff.com/digitalcameras.html. Accessed 8 May 2021.

[42] Nanotec Museum. What is a cmos image sensor? https://www.tel.com/museum/exhibition/principle/cmos.html. Accessed 8 May 2021.

[43] Elisabeth Gray. What is focal length in photography? https://photographylife.com/what-is-focal-length-in-photography. Accessed 8 May 2021.

[44] Denis Koshelev. What is a tof camera and how modern smartphones use it. https://root-nation.com/en/articles-en/tech-en/en-what-is-a-tof-camera/#What_is_ToF. Accessed 10 May 2021.

[45] Vision Team. What is a stereo vision camera? https://www.e-consystems.com/blog/camera/what-is-a-stereo-vision-camera/. Accessed 8 May 2021.

[46] Texas Instruments. Op amps for everyone. https://web.mit.edu/6.101/www/reference/op_amps_everyone.pdf. Accessed 29 Apr. 2021.

[47] Wilfried Elmenreich. Sensor fusion in time-triggered systems. https://mobile.aau.at/~welmenre/papers/elmenreich_Dissertation_sensorFusionInTimeTriggeredSystems.pdf. Accessed 29 Apr. 2021.

[48] Sparkfun. Analog-to-digital conversion. https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all. Accessed 29 Apr. 2021.

[49] Lonnie C. Ludeman. *Fundamentals of digital signal processing*, page 44–48. Wiley, 1987.

[50] Daksh Trehan. Gradient descent explained. https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c. Accessed 3 May 2021.

[51] Robert Alexander Adams and Christopher Essex. *Calculus: a complete course*, page 709–743. Pearson, 2014.

[52] G. Welch and G Bishop. An introduction to the kalman filter. page 1, 2006.

[53] Gholamreza Anbarjafari. Introduction to image processing. https://sisu.ut.ee/imageprocessing/book/1. Accessed 15 Apr. 2021.

[54] Priya Pedamkar. Rgb color model. https://www.educba.com/rgb-color-model/. Accessed 03 May 2021.

[55] Li Shuhua and Guo Gaizhi. The application of improved hsv color space model in image processing. In *2010 2nd International Conference on Future Computer and Communication*, volume 2, pages V2–10–V2–13, 2010.

[56] John Bradley. Rgb & hsv colorspaces. http://www.trilon.com/xv/manual/xv-3.10a/cover.html#Table%20of%20Contents. Accessed 19 Apr. 2021.

[57] Civil Lead. What is contour? what is contour interval? complete guide. https://www.civillead.com/what-is-contour/. Accessed 19 Apr. 2021.

[58] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. revision #137311.

[59] Intel. Realsense t265 motion tracker. https://www.intelrealsense.com/tracking-camera-t265/. Accessed 28 Apr. 2021.

[60] Danny Jost. What is an ultrasonic sensor? https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor. Accessed 28 Jan. 2021.

[61] Shawn. Types of distance sensor and how to select one? https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/. Accessed 28 Jan. 2021.

[62] Giulio Reina, James Patrick Underwood, and Graham Brooker. Short-range radar perception in outdoor environments. https://www.researchgate.net/publication/220942086_Short-Range_Radar_Perception_in_Outdoor_Environments. Accessed 9 May 2021.

[63] LiDAR and RADAR Information. Advantages and disadvantages of radar systems. https://lidarradar.com/info/advantages-and-disadvantages-of-radar-systems. Accessed 9 May 2021.

[64] Flyguys. Advantages and disadvantages of lidar. https://flyguys.com/advantages-disadvantages-lidar-technology/. Accessed 9 May 2021.

[65] ZMP. What is robovision® stereo camera? https://www.zmp.co.jp/en/knowledge/adas_dev/adas_sensor/adas_camera/adas_stereo. Accessed 9 May 2021.

[66] Adafruit. Gyroscope calibration. https://learn.adafruit.com/adafruit-sensorlab-gyroscope-calibration. Accessed 22 Apr. 2021.

[67] Rolfe Schmidt. Getting started with accelerometers and micro-controllers: Arduino + adxl335. https://chionophilous.wordpress.com/2011/06/20/getting-started-with-accelerometers-and-micro-controllers-arduino-adxl335/. Accessed 28 Apr. 2021.

[68] VectorNav. Magnetometer errors & calibration. https://www.vectornav.com/resources/magnetometer-errors-calibration. Accessed 21 Apr. 2021.

[69] Kris Winer. Simple and effective mangetometer calibration. https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration. Accessed 22 Apr. 2021.

[70] Mark Pedley. Tilt sensing using a three-axis accelerometer. https://www.nxp.com/files-static/sensors/doc/app_note/AN3461.pdf. Accessed 28 Apr. 2021.

[71] Michael J. Caruso. Applications of magnetoresistive sensors in navigation systems. https://d1.amobbs.com/bbs_upload782111/files_50/ourdev_711348H43I50.pdf. Accessed 29 Apr. 2021.

[72] Fatemeh Abyarjoo, Armando Barreto, Jonathan Cofino, and Francisco R. Ortega. Implementing a sensor fusion algorithm for 3d orientation detection with inertial/magnetic sensors. https://daimonmicha.bplaced.net/media/raspberrypi/Algo3DFusionsMems.pdf. Accessed 29 Apr. 2021.

[73] Kaustubh Sadekar and Satya Mallick. Camera calibration python opencv. https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html. Accessed 26 Apr. 2021.

[74] OpenCV. Harris corner detection. https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html. Accessed 28 Apr. 2021.

[75] Alexander Mordvintsev and Abid K. Sift (scale-invariant feature transform). https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html. Accessed 28 Apr. 2021.

[76] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[77] OpenCV. Flann (fast library for approximate nearest neighbours). https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html. Accessed 28 Apr. 2021.

[78] Scikit-Learn. Nearest neighbours). https://scikit-learn.org/stable/modules/neighbors.html. Accessed 28 Apr. 2021.

[79] Jernej Mrovlje1 and Damir Vrančić. Calculating depth. http://dsc.ijs.si/files/papers/S101%20Mrovlje.pdf. Accessed 7 May 2021.

[80] Nicolas de Palézieux, Tobias Nägeli, and Otmar Hilliges. Duo-vio: Fast, light-weight, stereo inertial odometry. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2237–2242, 2016.

[81] Anders Grunnet-Jepsen, Michael Harville, Brian Fulkerson, Daniel Piro, Shirit Brook, and Jim Radford. Intel vslam. https://dev.intelrealsense.com/docs/intel-realsensetm-visual-slam-and-the-t265-tracking-camera. Accessed 27 Apr. 2021.

[82] Jan Hartmann, Jan Helge Klüssendorff, and Erik Maehle. A comparison of feature descriptors for visual slam. In *2013 European Conference on Mobile Robots*, pages 56–61, 2013.

[83] Abdulrahman Alarifi, AbdulMalik Al-Salman, Mansour Alsaleh, Ahmad Alnafessah, Suheer Al-Hadhrami, Mai A Al-Ammar, and Hend S Al-Khalifa. Ultra wideband indoor positioning technologies: Analysis and recent advances, May 2016.

[84] Sebastian Dädeby and Joakim Hesselgren. A system for indoor positioning using ultra-wideband technology. Master's thesis, Chalmers University of Technology, 2017. Accessed 3 May 2021.

Appendices

Appendix A

Preproject Report

PILOT PROJECT - REPORT

FOR BACHELOR THESIS

NTNU
Kunnskap for en bedre verden

TITLE:
Position Estimation for Heave-Compensated System

CANDIDATES:
Erlend Holseker **Erik Bjørnøy** **Arvin Khodabandeh** **Isak Gamnes Sneltvedt**

DATE:	SUBJECT CODE:	SUBJECT:	DOCUMENT ACCESS:
15.01.2021	**IE303612**	**Bachelor Thesis**	- Open

STUDY:	NO. PAGES/ATTACHMENTS:	BIBL. NR:
AUTOMATION	10/1	- Not in use -

CLIENT/SUPERVISORS:
Seaonics Aleksander L. Skrede Ottar L. Osen

TASK/SUMMARY:
The theme of this report is the pilot project for a bachelor thesis. The thesis is to be conducted by four third-year automation students. The client is Seaonics. The task is to develop a sensor-package to improve their system for heave-compensated cranes and/or gangways. **To solve the problem, the group will make mathematical models of the issue, as well as develop a simulation. The group will collect information about relevant sensor types, and assess which sensor technology fits best for the purpose of the project. The justification for choice of sensor(s) will be properly documented with advantages and disadvantages.** **The group will develop an interface for reading and processing the sensor data. The data will be used further to control the heave-compensation.** **The group will have weekly status meetings, as well as a meeting with the steering group every other week.** **This report contains concepts, project organization, agreements, project description, documentation, planned meetings and reportation, planned deviation consideration and equipment needs.**

Post address
NTNU i Ålesund
6025 Ålesund
Norway

Visiting address
Larsgårdsvegen 2
Website
www.ntnu.no

Phone
73 59 50 00
E-mail address
postmottak@alesund.ntnu.no

This assignment is an exam submission conducted by student(s) at NTNU in Ålesund.

TABLE OF CONTENTS

1 INTRODUCTION

The client of this project is Seaonics. Seaonics was founded in 2011, and they deliver load handling technology to the marine and offshore industry. This includes crane systems and winches, with integrated control systems.

Seaonics has developed their own system for heave compensated cranes on ships. This system uses an MRU to measure the impact the waves have on the movement of the ship, in order to control the crane to hold the load still in relation to a fixed point. This system is used both for free-hanging loads, as well as gangways that are intended to transport crew from ship to, for example, a wind turbine in the sea. This system works well for pitch, roll and heave motions, but not so well for surge, sway and yaw motions. The goal of this project is to develop a sensor-package to improve the system to perform better on these motions.

The group will look primarily at two different cases; ship motion in relation to a fixed object, and ship motion in relation to a moving object. The group aims to develop a solution which can be adapted to both of these scenarios. It is also intended to be adaptable for both hanging load and gangways. To solve the problem, the group will look at and assess which sensor types are best to use, where the sensors should be placed, and how the signals from the sensors should be processed and used further.

The group has chosen this project because they think the issue is interesting and challenging, and they think it will be very educational.

2 CONCEPTS

Heave compensated crane
A heave compensated crane is a crane which uses a technique called AHC (Active heave compensation). This technique aims to reduce the impact ocean waves have on a ship's movement, to be able to hold a load as still as possible. This means that even if the ship is moving, the load will be hanging still at the same point.

MRU (Motion Reference Unit)
A MRU is a device containing a wide range of motion sensors. This device is used on ships for many different applications, such as dynamic positioning (DP), wave height monitoring, heave compensation of cranes, etc.

DOF (Degrees of Freedom)
Degrees of Freedom refers to the freedom of movement of a rigid object in the three-dimensional space (Wikipedia). More specifically, a rigid object is free to change position by moving in a straight line on the X, Y or Z-axis, or by rotating around one of these axes. Therefore, a rigid object has six degrees of freedom (6DOF).

3 PROJECT ORGANIZATION

3.1 Project group

Student number(s)
498356 - Erlend Holseker
485477 - Arvin Khodabandeh
487781 - Isak Gamnes Sneltvedt
494690 - Erik Bjørnøy

Table: Student numbers for all the group members

3.1.1 Tasks for the project group - organization

1) Project manager
2) Secretary
3) Group members

3.1.2 Tasks for project manager

The project manager is responsible for appointing and leading meetings, both internally and with the steering group. The project manager will ensure that the meetings are relevant and on point. Also, the project manager is responsible for contacting any group members that do not attend the meetings unless this has been clarified beforehand. The project manager is also responsible for following up on all deadlines and ensuring that each group member fulfills their assignments in a timely manner with high quality.

3.1.3 Tasks for secretary

The secretary is responsible for taking notes during meetings, as well as email communication with the steering group. The notes should give a short and understandable summary of the meeting. When the summary is written it should be published for all group members to access.

3.1.4 Tasks for remaining members

Each group member is required to solve all tasks that are assigned to them in a timely manner with high quality. If a group member is unable to finish their task within the time frame that has been set, the group member is to inform the project manager as soon as possible or ask any other group member for assistance. Every group member is responsible for helping other group members to the best of their ability when necessary.

3.2 Steering group(supervisor and client contact)

Seaonics - 71 39 16 00
Client: Daniel Nordal Bjørneseth
Client: Stig Espeseth
Supervisor: Aleksander L. Skrede
Assistant supervisor: Ottar L. Osen

4 AGREEMENTS

4.1 Agreement with client

The client has given a proposition of a task that the group may conduct. The group is to formulate the specific assignment within the boundaries of what has been discussed with the client.
The client will provide the group with an office at Seaonics premises on the grounds that the active restrictions due to Covid-19, provided either by the Norwegian government or Seaonics, allows the group to stay at the premises.

The group is not required to pay for any equipment during this project. If any equipment is needed to solve the project, the group may present this to the client which will decide whether or not to buy it.
The client will provide the group with help and consultation if needed.

The group has not discussed any disclosure agreement with the client.
Every group member will sign a standard agreement provided by NTNU for conducting a project assignment in collaboration with a company. See attachment 2.

4.2 Workplace and resources

Seaonics will provide the group with an office at their premises, unless the active Covid-19 restrictions does not allow the group to stay at the premises. In such a case, the group is required to work from home or find another solution.

Any equipment used to solve this project is to be paid for in its full by Seaonics. Seaonics will decide whether or not to buy the equipment.

Seaonics is to select a number of people which will act as their representatives. Any questions from the group regarding the project will be directed to the representatives from Seaonics.

The group will have weekly meetings with status updates internally in the group. Every 14th day there will be a meeting with the supervisor(s). During this meeting, the group will present their progress and get feedback on their work.

4.3 Group norms - Cooperation terms - Attitude

Every group member is required to aid and assist other group members when requested. Every group member is also responsible to fulfill their specific assignments within a given timeframe with as high quality as possible. The group members are required to be at office or available during the hours that have been agreed by the group, unless the group has been informed of the absence beforehand. In the event of illness, the group member is required to inform the rest of the group as soon as possible and is required to follow the current Covid-19 restrictions at that time.

Working hours are 08.00 - 16.00, Monday - Friday. Lunch break 11.30 - 12.00.
In week 3, 5, 9 and 10 the focus on this project will be smaller, since the group has to attend the Industry 4.0 course.

5 PROJECT DESCRIPTION

5.1 Problem - Goal - Intent

The goal of this project is to improve the positional estimation accuracy of the heave-compensated gangway. Today, the sensor used is an MRU. The existing heave-compensation system is well suited to compensate for pitch, roll and heave motions, but does not perform well at yaw, sway and surge motions. Our main goal is to find different sensors that can aid and improve the compensation to increase the positional accuracy of the system. To reach this goal, we have to set a number of sub-goals.

The first goal will be to find a mathematical model to describe the motions of a ship in relation to earth. Next goal is to expand the model to describe the motions of the ship in relation to a fixed point. After this is done, the goal is to mathematically describe the motion of two independent floating objects (e.g. a ship and a wind turbine) in the ocean, in relation to each other.
Once the group has understood the mathematics and physics behind this, they will create a simulation of it, in Matlab or Python or Unity. Seaonics uses Unity for their simulations, so the group will try to use this. The purpose of the simulation will be to gain a better understanding of the physics, and the fact that making a simulation is instructive in itself.
In parallel with these goals, the group will look at the possibilities of sensors on the market. The plan is to create a well-descriptive list of sensors that describe the advantages and disadvantages of all types of sensors.
The group understands that they will most likely need to use Kalman filter in the project. Therefore, the group will set aside some time to learn the theory behind the Kalman filter.

When the group has chosen one or several sensors to use, they will develop an interface for reading and processing the input from the sensors. The purpose is to make the information from the sensor(s) easy to read and utilize for Seaonics' external system. In addition to this, the group has to figure out where the sensor(s) should be placed. An option is to create a small prototype with small-scale sensors to see when the prototype performs best.

If the main goal is reached and the group has extra time, they will try to expand the solution to also fit on heave-compensated cranes.

The group aims to start writing the bachelor report early, and write it alongside the project.

5.2 Requirements for solution or result - specification

One of the risks in the project is that the system does not work under all conditions. Due to the fact that the system will be used on a ship in the ocean, there is a possibility that the measurements from the sensors can be disrupted or interfered by water. Another risk is that the system will perform well for countering big waves, but not for smaller waves or movements, or vice versa.

One important aspect to succeed is the placement of the sensor(s). The sensor(s) has to be placed in a location where they are not exposed to the elements or any other factors that may influence the sensor readings

The result should include a theoretical solution to minimize the error that the MRUs is unable to counteract. Since the system will be used on the ocean, it should meet the requirements for Seaonics' current standards. The results should also include a proposition of which sensor(s) to use.

The result that is handed over to the client must contain a thorough justification of the choice of sensor(s) and their placement, as well as all of the mathematical solutions that form the basis for the choices that have been made. In addition, the project report must be part of the delivery. If a prototype is made, this will also be handed over.

5.3 Planned procedure(s) for development – method(s)

In the beginning of the project, the group will focus on the theoretical part of the project. This includes calculating the mathematical models of a ship's motion in relation to ocean waves. After the first mathematical model is complete, the group will continue by calculating the mathematical model of a ship's motion in relation to a fixed point. The third mathematical model will be a model that describes the

motion of two moving objects in relation to each other. Lastly, the models will be simulated in either Python or Matlab. While calculating the mathematical models, the group will also be focusing on comparing different sensors that can be used to solve the project. The group will also learn how to implement a Kalman Filter.

The second phase of the project is the implementation. In this phase, the group will start off by finding a few different places where the sensors can be placed. The location must allow the sensor to get good readings, meaning it will not get blocked by any other objects, for as much time as possible. The sensor should also be placed in a location where it will not be exposed to rain, waves and other factors that may disrupt the readings and cause noise. The group will also implement an interface for reading and processing the input from the chosen sensors.

The last phase is the testing phase. The testing phase will be used to determine how effective the system is and how good it works under different conditions. The group will try to make it fail by causing "worst case scenarios". This may include unplugging one or more of the sensors while running, splash water on the sensors and different types of waves. During this phase, there may be some changes to the parameters of the mathematical models, the sensor type, specification or placement of the sensor(s).

5.4 Collection of information – completed and planned

Seaonics has already developed a system that uses MRU's. This means that they have already calculated the mathematical models and implemented the system. This may prove useful if the group is struggling or is unable to solve some of the tasks without the help from the representatives from Seaonics.

5.5 Assessment – risk analysis

One of the greatest risks for this project is the current Covid-19 pandemic. If the Norwegian government introduces a curfew, the group must work from home. This will make the project more difficult, because the group can not meet physically, and will not get help as easily from Seaonics.

Another risk is if the group decides to order any equipment, and there's problems with the delivery. It could be late delivery, or if the ordered items are lost in transit. By using equipment in local stock to the greatest extent possible, this risk could be minimized.

5.6 Main activities in further work

The main activities, time frames and responsibilities can be found in Attachment 1, Gantt-chart.

5.7 Progress plan – project management

5.7.1 Main plan

For this project the goal is to locate the position of a given entity in relation to another object, while both objects are in motion. To achieve this the group must recommend a sensory system that can be fitted to the equipment delivered by Seaonics.
There can be up to six degrees of freedom for each object which must be accounted for.

Ideally, a general solution would be most efficient, with one sensor-pack that can fit on both the gangway and a crane, but abstracting and generalizing the sensor-package can be impractical, so there is a possibility for another solution, with different sensors-packages for each case. The main goal is to create a solution that fits the gangway.
The group must find the most practical and suitable solution.

In the preliminary stages the aim is to first achieve parts of the overall goal, to reach a milestone.
This is to not tackle the full magnitude of the assignment at once, and rather being able to fragment the different challenges into smaller pieces, and solve them one at a time.

For the first goal the group will focus on finding a mathematical model that represents and describes a vessel's movement at sea.

The second goal will be to mathematically describe the movement of two moving objects in relation to each other.

At some stage, after acquiring enough knowledge around the conditions of the assignment, sensor equipment must be explored, and be considered for purchase.

After the mathematical models are in place, the focus will be on the compensated gangway delivered by Seaonics. Mainly, orienting a floating vessel in relation to a stationary target.
In this case there will only be accounted for one set of DOF. This is on the vessel itself, and the object that the vessel is oriented in relation to is assumed to be stationary, as in the case of a windmill anchored to the seafloor.

When orienting in relation to a stationary target is achieved, the next focus will be to orient a vessel in relation to a moving object, as in the case of a windmill or vessel that is floating.

In regards to the timeline and individual responsibilities of each goal, see attachment 1.

5.7.2 Steering aids

The group will be using a Gantt-chart to keep track of sub-goals, milestones, timeframes and responsibilities. The Gantt-chart will be updated weekly on the weekly status meetings within the group. The Gantt-chart has been divided into 3 main phases; Theory, Implementation and Testing. Each phase contains multiple sub-categories which describe each task during the project. The Gantt-chart will be changed during the project as is it hard to accurately determine the necessary time spent on each task. Additionally, each task may also be divided into more sub tasks. This will be done to give a more accurate visualisation of the progress.

5.7.3 Development aids

During the project, the group assumes it will take use of 3D-printers and different software. The 3D-printers will mainly be used to make prototypes for the solution. The software that will be used will predominantly be free, but some may require a licence.

5.7.4 Internal control - evaluation

Progress evaluation will be done weekly on the weekly status meeting. A goal is reached after an evaluation from the group as a whole. A completed goal should be a fully functional/complete part of the assignment. The project manager is responsible for following up on the progress.

5.8 Decisions – decision making process

During the pilot project, the group has been making decisions in meetings. The group discussed advantages and disadvantages before settling on a decision. A similar approach will be done during the main project. An exception is that when it comes to major decisions, the group members will be given a deadline to prepare arguments. This is done to ensure that the best possible decisions are made and that the group knows more about each side of the issue. This may prove useful when the group is to decide on what kind of sensor they want to use and the main approach to solving the assignment.

6 DOCUMENTATION

6.1 Reports and technical documents

To keep track of all the documentation related to the project, the group has created a Google Drive folder where everything is stored. This includes meeting minutes, weekly progress reports, Gantt charts, technical documents and datasheets, and references.

7 PLANNED MEETINGS AND REPORTS

7.1 Meetings

7.1.1 Meetings with the steering group

The group will convene a status meeting every other monday. A notice will be sent to every involved part. At these meetings, the latest status report will be discussed, as well as further work.

7.1.2 Project meetings

The group has planned a weekly summary meeting every friday at 14.00, as well as a kick-off meeting every monday at 09.00. The purpose of these meetings is to maintain a good continuity in the project, and keep the progress on schedule.

7.2 Periodic reports

7.2.1 Progress reports (incl. milestones)

The Gantt chart will be updated with completed milestones every friday at the summary meetings. A status report will also be written on Fridays, in addition to the Gantt chart. The status report will be sent to the supervisors every week.

8 PLANNED DEVIATION CONSIDERATION

By using the summary reports and the Gantt chart, any progress deviations will be detected quickly. If any deviations are found, corrective measures could be implemented, e.g. dedicate more time to a specific task or add an extra person to the task.

Any changes to the project will be determined at meetings, and the changes will be noted in summary reports and updated Gantt chart.

One specific deviation the group has in mind is that if the Covid-19 situation worsens, the Norwegian government can introduce a curfew. If this happens, the group will not be able to meet each other physically, and lab work will therefore not be possible. In this case, the group will continue to work as best as possible and have all meetings online.

If one of the group members falls ill and is unable to perform his tasks, the group will consider how these tasks should be distributed during the period the group member is ill.

9 EQUIPMENT NEEDS/PREREQUISITES FOR IMPLEMENTATION

In the beginning of the project, there is no need for any software or hardware. During this project, the group assumes that some sensors and/or software licenses are needed, but this will be discussed with the client when the time comes. The group will most likely use the 3D printers located at NTNU to make prototypes.

10 REFERENCES

Wikipedia. n.d. "Six degrees of freedom." Wikipedia.

https://en.wikipedia.org/wiki/Six_degrees_of_freedom.

ATTACHMENTS

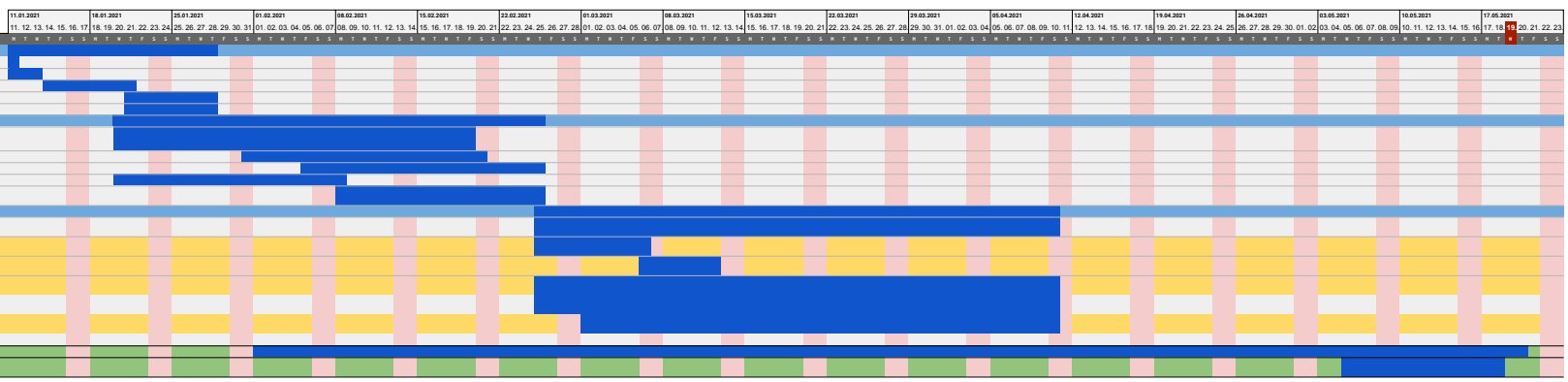
Attachment 1	Gantt-diagram
Attachment 2	Standard Agreement Guidance NTNU

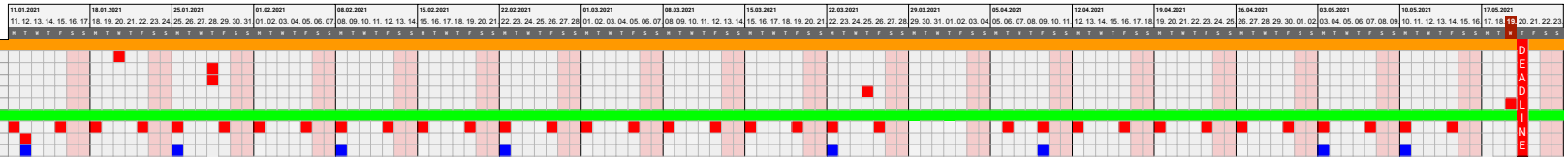
Appendix B

Gantt Chart

Arvin Khodabandeh	**Erlend Holseker**	**Project Start:**	**11.01.21**
Isak Gamnes Sneltvedt	**Erik Bjørnøy**	**Display Week:**	**1**

TASK	ASSIGNED TO	PROGRESS	START	DAYS	END
Phase 1 - Pilot Project		100%	**11.01.21**		**28.01.21**
Divide responsibility	All members	100%	11.01.21	1	11.01.21
Project description	All members	100%	11.01.21	3	13.01.21
Pre project report Draft	All members	100%	14.01.21	7	21.01.21
Pre project report	All members	100%	21.01.21	7	28.01.21
Progress plan	All members	100%	21.01.21	7	28.01.21
Phase 2 - Preparation phase		100%	**20.01.21**		**25.02.21**
Learn about rotation matrices and transformation matrices	Arvin, Erlend	100%	20.01.21	31	19.02.21
Learn about quaternions	Arvin, Erlend	100%	31.01.21	20	20.02.21
Learn about Kalman filter	All members	100%	05.02.21	20	25.02.21
Sensor research and documentation	Erik, Isak	100%	20.01.21	19	08.02.21
Evaluate different types of sensors and where to place them	Erik, Isak	100%	08.02.21	17	25.02.21
Phase 3 - Implementation phase		100%	**25.02.21**		**10.04.21**
Develop and test a system for orientation estimation	**Arvin, Erlend**	100%	**25.02.21**	**45**	**10.04.21**
Implement Kalman filter for sensor fusion	Arvin, Erlend	100%	25.02.21	10	06.03.21
Learn about and implement Madgwick filter for sensor fusion	Arvin, Erlend	100%	06.03.21	7	12.03.21
Create python classes for handling rotations and sensor processing	Arvin, Erlend	100%	25.02.21	45	10.04.21
Develop and test a system for position estimation	**Erik, Isak**	100%	**25.02.21**	**45**	**10.04.21**
Implement a stereo camera to estimate position	Erik, Isak	100%	01.03.21	40	10.04.21
		100%			
Write report	**All members**	100%	**01.02.21**	**108**	**20.05.21**
Create video presentation, including shooting and editing	All members	100%	05.05.21	13	18.05.21
Total Progression		**100%**	**11.01.21**	**129**	**20.05.21**
Days until DEADLINE		**99%**	**19.05.21**	**1**	**20.05.21**





Appendix C

Progress Reports 1-6

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 05.02.21

Main goal/purpose for this periods work

- Look into different types of sensor technologies
- Familiarize ourselves with the mathematics describing the motions of a ship
- Finish the pilot project report

Planned activities this period

- Find a way to mathematically describe the motions of a ship
- Evaluate different types of sensors
- Start creating a simulation of the mathematical models

Actually conducted activites this period

- We have written seven discussions about seven different sensor technologies:
- Camera
- Gyro
- Infrared
- LiDAR
- Magnetometer
- Radar
- Ultrasonic

These discussions will be used to decide which sensor technology/technologies to use in the project when the mathematical models are finished.

- We have gained a little more understanding of how to mathematically describe the different motions of a ship in relation to the earth, but we are not quite done with it yet.

Description of/ justification for potential deviation between planned and real activities

- We have not started to create a simulation of the mathematical models yet. This cannot be done before we have a clear understanding of the mathematics. In week 5 we have had lectures and assignments in Industry 4.0, which has slowed down our progress a little bit. We are also currently writing a scientific report in collaboration with Guoyuan Li. We aim to finish this report as soon as possible.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have learned a lot about different sensor technologies and will use this information further to decide which technologies to use.
- We have been discussed a lot about the issue the project is concentrating on and gained a greater understanding of what the issue is.
- We have gained more insight about how to find the mathematical models.

Main purpose/focus next period

- Mathematical models

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 05.02.21

- Simulation
- Discuss more about different types of sensor technologies

Planned activities next period

- Mathematically describe the motions of a ship.
- Create a simulation of the motions we can mathematically describe in this period.
- Look into the advantages/disadvantages of using a TOF Camera.
- Start the evaluation of which sensors to use.
- Learn about Kalman Filter.

Other

Wish/need for counceling

- Nothing in particular at the moment.

Approval/signature group leader	Signature other group participants
Erik Bjørnøy	Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 19.02.21

Main goal/purpose for this periods work

- Mathematical models
- Simulation
- Discuss more about different types of sensor technologies

Planned activities this period

- Mathematically describe the motions of a ship.
- Create a simulation of the motions we can mathematically describe in this period.
- Look into the advantages/disadvantages of using a TOF Camera.
- Start the evaluation of which sensors to use.
- Learn about Kalman Filter.

Actually conducted activites this period

- We have looked into the advantages and disadvantages of using a TOF Camera.
- We have started to look into the concept of quaternions, in order to better understand the mathematics.
- We have managed to visualize some motions using a 3D cube in Python.
- We have started to evaluate which sensors to use, but we need to be finished with the mathematical models first.
- We have started to write about the theoretical basis in the Bachelor report.
- Friday 19.02, we had a meeting with Daniel Bjørneseth at Seaonics' offices, and he showed us more about their simulator. We want to use this simulator to simulate our solution.

Description of/ justification for potential deviation between planned and real activities

- We have not learned about Kalman Filter yet. We will schedule a lecture with Aleksander on the next status meeting.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have learned about quaternions.
- We have managed to visualize some of the motions in Python.
- We have started to write the Bachelor report.
- We have gained better insight in Seaonics' simulator.

Main purpose/focus next period

- Simulation in Unity
- Learn Kalman Filter and sensor fusion
- Mathematical modeling

Planned activities next period

- Hopefully, we will get access to Seaonics' offices so that we can look further into their simulator and figure out how to test our solution in their simulator.
- Learn about Kalman Filter
- Learn about sensor fusion

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 19.02.21

- Combine Eular angles, quaternions and reference systems to figure out the best way to describe the motions of the gangway in relation to the ship

Other

Wish/need for counceling

- Lecture in Kalman Filter (this will be scheduled with Aleksander)

Approval/signature group leader	Signature other group participants
Erik Bjørnøy	Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 05.03.21

Main goal/purpose for this periods work

- Simulation in Unity
- Learn Kalman Filter and sensor fusion
- Mathematical modeling

Planned activities this period

- Get access to Seaonics' offices so that we can look further into their simulator and figure out how to test our solution in their simulator.
- Learn about Kalman Filter
- Learn about sensor fusion
- Combine Euler angles, quaternions and reference systems to figure out the best way to describe the motions of the gangway in relation to the ship

Actually conducted activites this period

- We have had a lecture in Kalman Filter and sensor fusion with Aleksander.
- We have been experimenting with euler angles, quaternions and reference systems and visualized it in Python using a 3D cube. We are currently working on making classes in Python to create representations of the orientation and position of different points on the ship in relation to another chosen point.
- We have been experimenting with an IMU and an Arduino. We have been reading data from the IMU and plotted a live graph showing the position of the IMU.
- We have been working on visualizing and converting measured data.

Description of/ justification for potential deviation between planned and real activities

- The last week has been occupied by the Industry 4.0 course. Therefore, we have only had one full work week the last two weeks. Next week will be the last module of the Industry 4.0 course.
- We have not started to look into simulation in Unity yet, but we hope to do this in the next couple of weeks.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have learned about Kalman Filter.
- We have learned about sensor fusion.
- We have started to create classes in Python for representing the orientation and position of different points on the ship.
- We feel that we have gained a clear understanding of how the mathematics should be represented.
- We have gained access to Seaonics' offices which has been very beneficial for our progress.
- We have gained more knowledge on how to process measured sensor data.

Main purpose/focus next period

- Finish the Industry 4.0 course, as well as the exam assignments.
- Figure out how to implement Kalman filter and sensor fusion.
- Look into simulation in Unity.

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 05.03.21

Planned activities next period

- Finish the Industry 4.0 course and all the assignments related to this.
- Figure out how to implement Kalman filter and sensor fusion, and how to combine it with our mathematical model.
- Explore the possibilities we have with simulation in Unity.

Other

Wish/need for counceling

- Nothing in particular at the moment.

Approval/signature group leader	Signature other group participants
Erik Bjørnøy	Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 19.03.21

Main goal/purpose for this periods work

- Finish the Industry 4.0 course, as well as the exam assignments.
- Figure out how to implement Kalman filter and sensor fusion.
- Look into simulation in Unity.

Planned activities this period

- Finish the Industry 4.0 course and all the assignments related to this.
- Figure out how to implement Kalman filter and sensor fusion, and how to combine it with our mathematical model.
- Explore the possibilities we have with simulation in Unity.

Actually conducted activites this period

- Finished the Industry 4.0 course and all assignments related to this.
- We have been experimenting with Kalman Filter and combined it with IMU readings on an Arduino.
- We have bought a microphone stand to use for physically simulating the motions of the gangway. We will use this to test sensors and programming solutions.
- Seaonics has provided us with a 3D camera
- We have started to experiment with ROS to create a SLAM with the 3D camera

Description of/ justification for potential deviation between planned and real activities

- We have decided to try to physically simulate the motions of the gangway using a microphone stand, rather than simulating it in Unity as we think it will be too time-consuming.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have gained a better understanding of the use of Kalman filter in combination with sensor readings.
- We have realized that it will be easier to test our solutions physically rather than in a simulation, because then it is easier to physically measure and observe that our output from our program is as correct as possible.
- We are currently working on the basis of a theory that combines the use of an IMU and a 3D camera.

Main purpose/focus next period

- Combine IMU measurements, Kalman Filter and orientation/position representation in different frames.
- Create a SLAM using the 3D camera.
- Look into which other sensor types that can be beneficial to use.

Planned activities next period

- Make a bracket to fasten the different sensors on the tip of the microphone stand.
- Create a Kalman Filter algorithm to “filter” the readings from an IMU.
- Represent the filtered IMU data in relation to different frames.
- Create a SLAM using the 3D camera and ROS.

Other

Wish/need for counceling

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 19.03.21

- Nothing in particular at the moment.	
Approval/signature group leader Erik Bjørnøy	Signature other group participants Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 08.04.21

Main goal/purpose for this periods work

- Combine IMU measurements, Kalman Filter and orientation/position representation in different frames.
- Create a SLAM using the 3D camera.
- Look into which other sensor types that can be beneficial to use.

Planned activities this period

- Make a bracket to fasten the different sensors on the tip of the microphone stand.
- Create a Kalman Filter algorithm to "filter" the readings from an IMU.
- Represent the filtered IMU data in relation to different frames.
- Create a SLAM using the 3D camera and ROS.

Actually conducted activites this period

- We have mounted an IMU sensor and an ultrasonic distance sensor on the tip of the microphone stand. These sensors are connected to an Arduino.
- We have created a Kalman Filter algorithm in Python which reads the serial output from the Arduino. Using Qt, we are now able to live-plot the sensor readings and the Kalman estimate. We have tried to use the distance sensor and the accelerometer in the IMU to estimate the distance and velocity of the microphone tip in relation to a given object. This works until we remove the distance sensor input; then the accelerometer drifts very fast and the estimate becomes unreliable until the distance sensor input is back.
- We have been working on recognizing an object using contours and color, to use as a reference to measure distance/velocity using the stereo camera.
- We have continued writing the project report.

Description of/ justification for potential deviation between planned and real activities

- Due to the covid19-situation, the Norwegian government has decided that everyone should work from home. That means we can no longer use Seaonics' offices. Therefore, Arvin and Erlend are now working together on the Kalman Filtering and microphone stand "prototype", while Erik and Isak are working together on using a stereo camera to observe and calculate distances and velocities in different directions.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have realized that using accelerometer and an ultrasonic distance sensor for fusing data to estimate position and velocity does not work very well when the distance sensor data is lost.
- We know that it is only a little bit over a month until the deadline of the project, which means that we soon will have to concentrate about the project report.

Main purpose/focus next period

- Continue to work on orientation estimation using IMU (accelerometer, magnetometer, gyroscope) and sensor fusion.
- Continue writing the project report.
- Continue to work on the stereo-camera distance measuring. Hopefully we will be able to combine this work with the orientation estimation using the IMU.

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 08.04.21

Planned activities next period

- Create a sensor fusion algorithm for estimating orientation using an IMU. Our initial plan is to use the gyroscope to measure/estimate the orientation, and the accelerometer and magnetometer for correcting the estimate.
- Write more on the project report.
- Continue to work on the stereo-camera distance measuring solution.

Other

Wish/need for counceling

- Nothing in particular at the moment.

Approval/signature group leader	Signature other group participants
Erik Bjørnøy	Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 1 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 30.04.21

Main goal/purpose for this periods work

- Continue to work on orientation estimation using IMU (accelerometer, magnetometer, gyroscope) and sensor fusion.
- Continue writing the project report.
- Continue to work on the stereo-camera distance measuring. Hopefully we will be able to combine this work with the orientation estimation using the IMU.

Planned activities this period

- Create a sensor fusion algorithm for estimating orientation using an IMU. Our initial plan is to use the gyroscope to measure/estimate the orientation, and the accelerometer and magnetometer for correcting the estimate.
- Write more on the project report.
- Continue to work on the stereo-camera distance measuring solution.

Actually conducted activites this period

- We have tested different combinations of filtering and sensor fusion to estimate orientation using raw data from the IMU, and have concluded with what combination we think is the best, with the time we have left of the project.
- We have managed to translate the orientation of the gangway to the ship. By estimating the orientation of the gangway, it is now possible to also calculate the orientation of the ship.
- We have managed to estimate distance and translational movement in surge and sway directions, using stereo-camera.
- We have created a graphical user interface using Qt. This GUI shows graphs representing the raw data from the IMU, as well as estimated orientation angles.
- We have been writing more on the project report.

Description of/ justification for potential deviation between planned and real activities

- The covid-19 situation in Ålesund prevents us from being able to be at Seaonics' offices. Also, a group member has been quarantined, making it impossible to work together physically. The group member is now out of quarantine.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- No changes at the moment.

Main experience from this period

- We have concluded with our opinion on how to estimate orientation.
- Project report is taking form.
- Stereo-camera works well to estimate distance.

Main purpose/focus next period

- Project report.
- Finish the estimation application.

Planned activities next period

- Continue to write the project report.
- Include the distance estimations in the GUI. Also, a camera feed from the stereo-camera is to be included in the GUI.

Other

Wish/need for counceling

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

IE303612 Bacheloroppgave	Project Position Estimation	Firma - Oppdragsgiver Seaonics	Side 2 av 2
Progress report	Period/week(s) 2	Prosjektgruppe (navn) Erik Bjørnøy, Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt	Dato 30.04.21

- Aleksander will look at our progress on the project rapport and give feedback. Aleksander will be invited to the Overleaf document.	
Approval/signature group leader Erik Bjørnøy	Signature other group participants Erlend Holseker, Arvin Khodabandeh, Isak Gamnes Sneltvedt

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

Appendix D

Meeting Reports 1-7

Oppstartsmøte 12.01.2021

Oppgaven går ut på å kunne beregne båtens posisjon i forhold til et objekt ved bruk av sensorer (og kalman filter)

- Valg av sensor
- Implementasjon (forslag/påbegynnelse av implementasjonen)
- Modell
    - 6 frihetsgrader
    - Hvilke målinger trenger vi?
- Data fra MRU:
    - MRU gir vinkler/posisjon
    - Akselerasjoner
    - Kommer posisjon estimert ut fra MRU
- Prøve å gjøre systemet kompatibelt med både gangbro og kran
- Stort sett standardisert på en type MRU
- Hvilke krav settes til nøyaktighet

To mål:

1. Spesifikk oppgave, hva skal gjøres
    1. Gruppen ønsker å kunne estimere posisjonen og orientere seg i forhold til et objekt.
2. Vise til hva som kan gjøres for å forbedre prosjektet og hvordan
    1. Systemet blir plug and play?

Forslag til gangbro:

- LiDAR?
- Radar?
- Kamera (2D/3D)?

Oppgavebeskrivelse:

- Det som ligger på BB er i utgangspunktet forslag. Seaonics ønsker ikke å låse oss til noe, og vi kan selv formulere oppgaven innenfor det utgangspunktet som har blitt diskutert. Forprosjektet vil være et utgangspunkt for oppgavebeskrivelsen. Eks: "Gangvei mot flytende vindmølle". Oppgaven kan gjøres større etter hvert. Tenk generelt.
- Stegvis kan det være:
    1. Bruke et faststående objekt som referanse
    2. Et annet bevegelig objekt som referanse

Posisjonsestimering

Møtereferat

25.01.2021

I. Start møtet

Erik Bjørnøy startet møtet med styringsgruppen kl. 11.00 den 25.01.21 på Teams.

II. Opprop

Følgende personer var til stede:

Erlend Holseker(sekretær), Erik Bjørnøy(prosjektleder), Isak Gamnes Sneltvedt(gruppemedlem), Arvin Khodabandeh(gruppemedlem), Aleksander Larsen Skrede(hovedveileder) og Daniel Bjørneseth(oppdragsgiver).

III. Saker

a) Gjennomgang av forprosjektrapport.

   Forprosjektrapporten er bra og har bra introduksjon. Før innlevering av den endelige forprosjektrapporten bør gruppen definere målene bedre. Her må gruppen ha et klart hovedmål, med eventuelle delmål knyttet til dette hovedmålet, med prioritet. Det kan også legges til bonusmål som kan bygge videre på hovedmålet. Problemstillingen bør spisses litt mer inn slik at gruppens hovedmål blir litt mer spesifikt. Gruppens hovedmål er posisjonsestimering / avstandsmåling for gangvei.

   I tillegg til dette så er det bare nødvendig å ta med navn til medlemmene i styringsgruppen.

   Aleksander sender standardavtale til gruppen slik at dette kan tas med i forprosjektrapporten.

b) Gjennomgang av Gantt-diagram

   Gantt-diagram ser greit ut. Bør nevne i forprosjektrapporten at oppgaven er delt opp i tre faser slik som det er gitt uttrykk for i Gantt-diagrammet. Gantt-diagramet vil bli oppdatert til hvert statusmøte, og det vil dermed

bli mer detaljert etter hvert som gruppen får se hvordan prosjektet utvikler seg. Dette kan også nevnes i forprosjektrapporten.

c) Avtale opplæring i Kalman filter

Aleksander sender et dokument med introduksjon om Kalman filter. Dette skal gruppen lese og gå gjennom før Aleksander gir gruppen en leksjon i Kalman filter. Denne leksjonen vil skje mot slutten av februar.

d) Øvrig diskusjon

Seaonics har en simulator. Den skal være grei å kjøre i gang. Vil kanskje ta en time eller to å sette opp på egen pc. Har tilgang til kildekoden til Unity så det skal være mulig å sette inn egne sensorer i simuleringen. Kan være lurt å sjekke ut denne simulatoren.

Daniel tipset også om å sjekke ut Webots. Dette er en open source simulator.

Aleksander sender lenker til relevante journaler og forskningsrapporter. Forskningsbasert oppgave er et «kriterie» for toppkarakter.

Når det gjelder grenser rundt værforhold, så ble det nevnt at man kjører ikke ut gangveien dersom det er mer en 25 m/s vindstyrke. En gangvei er normalt sett ute i 10-30 minutter før skipet går videre til neste jobb.

Signal Quest 2096 (IMU) blir testet på tuppen av gangveien nå. Den blir brukt for å måle bevegelsen til tuppen av gangveien i forhold til skipet.

Kongsberg MRU-H blir brukt i dag. Datablad til denne MRU-en finnes på Kongsberg sine nettsider. Seaonics planlegger å bytte til en MRU med litt bedre oppløsning.

Seaonics bruker 3D sensorer av typen IFM O3M161 på tuppen av gangveiene idag. Disse sensorene genererer en punktsky, men har ikke veldig god oppløsning. Det vurderes å bytte ut disse sensorene med en sensor som har bedre oppløsning; Basler Blaze-101.

Det er ingen spesifikke krav til oppdateringsfrekvens på sensorer, men vanligvis kjører komponentene på rundt 100 Hz. Gruppen vil drøfte

eventuelle egendefinerte krav rundt oppdateringsfrekvens i prosjektrapporten.

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 11.55.

Referat sendt av: Erlend Holseker

Referat godkjent av: Isak Gamnes Sneltvedt og Arvin Khodabandeh

Posisjonsestimering

Møtereferat

08.02.2021

I. Start møtet

Erik Bjørnøy startet møtet med styringsgruppen kl. 12.00 den 08.02.21 på Teams.

II. Opprop

Følgende personer var til stede:

Erlend Holseker(sekretær), Erik Bjørnøy(prosjektleder), Isak Gamnes Sneltvedt(gruppemedlem), Arvin Khodabandeh(gruppemedlem), Aleksander Larsen Skrede(hovedveileder) og Daniel Bjørneseth(oppdragsgiver).

III. Saker

a) Gjennomgang av fremdriftsrapport

   Fremdriftsrapporten fra 05.02 ble gjennomgått. Vi har den tidligere uken laget sju drøftinger av sju forskjellige sensorteknologier, som skal brukes til å rettferdiggjøre vårt fremtidige valg av sensorer til oppgaven. I tillegg har vi jobbet mye med å få kontroll på matematikken som ligger bak bevegelsene til et skip.

   Planen for den neste uken er å skrive en egen forklaring på matematikken som ligger bak bevegelsene til et skip, og prøve å simulere dette. I tillegg vil vi se nærmere på fordeler og ulemper ved å bruke TOF kamera. Vi vil også prøve å lære mer om Kalman filter, da dette er noe som høyst sannsynlig må brukes i oppgaven vår.

b) Øvrig diskusjon

   Gruppen ønsker å kunne benytte seg av Seaonics sine kontor, da vi tror dette vil føre til økt produktivitet, sammenlignet med å sitte hjemme eller på NTNU. På NTNU foregår det byggearbeid som kan bli noe forstyrrende. Daniel tar dette opp med Seaonics og har tro på at vi kan installere oss på kontorene der snart. Koronasituasjonen avgjør. Daniel

kommer kanskje til Ålesund i slutten av neste uke dersom situasjonen tilsier det, og vil da ta et møte med oss for å vise oss mer av Seaonics sin lab.

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 12.20.

Referat sendt av: Erlend Holseker

Referat godkjent av: Isak Gamnes Sneltvedt og Arvin Khodabandeh

Posisjonsestimering
Møtereferat

22.02.2021

I. Start møtet

Erik Bjørnøy stater møtet med styringsgruppen kl. 11.00 den 22.02.2021 på Teams.

II. Opprop

Følgende personer var tilstede:

Erik Bjørnøy(prosjektleder), Erlend Holseker(sekretær), Arvin Khodabandeh(gruppemedlem), Isak Gamnes(gruppemedlem) og Aleksander Larsen Skrede(hovedveileder).

III. Saker

a) Gjennomgang av Kalman-filter

Vi avtalte passende tidspunkt for gjennomgang av Kalman-filter. Aleksander foreslo torsdag 25.02.21 kl 09.00 eller kl 12.00 hvis det skulle gjennomføres på skolen, eller kl. 12.00 hos Seaonics. Gruppen og Aleksander ble enige om å ta det kl 12.00 hos Seaonics. Alle leser gjennom PDF-en om Kalman-filter Aleksander sendte på mail før gjennomgangen.

b) Matematisk modell

Gå mer i dybden på hvordan man kan bruke transformasjonsmatriser for å representere bevegelsene til gangbrua i forhold til massesenteret på båten.

c) Øvrig

Gruppen sender en epost til Aleksander snarest mulig med spesifikasjoner til en IMU-sensor(Benevninger/enheter til målingene).

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 11.40.

Referat sendt av: Arvin Khodabandeh

Referat godkjent av: Isak Gamnes Sneltvedt og Erlend Holseker

Posisjonsestimering

Møtereferat

22.03.2021

I. Start møtet

Erik Bjørnøy startet møtet med styringsgruppen kl. 12.00 den 22.03.21 på Teams.

II. Opprop

Følgende personer var til stede:

Erlend Holseker(sekretær), Erik Bjørnøy(prosjektleder), Arvin Khodabandeh(gruppemedlem), Aleksander Larsen Skrede(hovedveileder), Ottar L. Osen(assisterende veileder) og Daniel Bjørneseth(oppdragsgiver).

III. Saker

a) Gjennomgang av fremdriftsrapport

   Fremdriftsrapporten fra 19.03 ble gjennomgått. Vi har den tidligere uken jobbet med bruk av 3D kamera i kombinasjon med en SLAM algoritme. I tillegg har vi kjøpt et mikrofonstativ som vi vil bruke til å prøve å montere sensorer på og simulere bevegelsen til gangbroa.

   Planen for den neste uken er å fortsette med dette. Vi vil bruke mikrofonstativet og kombinere IMU målinger, Kalman filter og representere målingene relativt til forskjellige referansepunkt, og deretter måle fysisk om målingene våre er presise. Vi vil også se videre på andre sensortyper.

b) Øvrig diskusjon

   Koronasituasjonen i Ålesund har forverret seg, noe som har ført til at kommunen har satt i verk nye restriksjoner. Dette påvirker oss på den måten at vi ikke lenger kan benytte oss av Seaonics sine lokaler, og bruk av lab på skolen må gjennomføres ved at vi må spørre labansvarlig først. Ved bruk av lab bør vi ikke være mer enn to gruppemedlemmer til stede.

Ottar har sendt oss tre tidligere bacheloroppgaver som kan være relevante i forbindelse med å bruke kamera til å måle relativ posisjon. Daniel har gitt oss tilgang til Unity, så vi vil se litt på dette også i uken som kommer. Til neste møte må vi oppdatere fremdriftsplanen (Gantt-diagrammet).

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 12.55.

Referat sendt av: Erlend Holseker

Referat godkjent av: Isak Gamnes Sneltvedt og Arvin Khodabandeh

Posisjonsestimering

Møtereferat

09.04.2021

I. Start møtet

Erik Bjørnøy startet møtet med styringsgruppen kl. 13.30 den 09.04.21 på Teams.

II. Opprop

Følgende personer var til stede:

Erlend Holseker(sekretær), Erik Bjørnøy(prosjektleder), Arvin Khodabandeh(gruppemedlem), Isak Gamnes Sneltvedt(gruppemedlem) Aleksander Larsen Skrede(hovedveileder) og Daniel Bjørneseth(oppdragsgiver).

III. Saker

a) Gjennomgang av fremdriftsrapport

Fremdriftsrapporten fra 08.04 ble gjennomgått. Vi har den tidligere uken jobbet med bruk av stereokamera for å måle forflytning i forhold til et referansepunkt. I tillegg har vi montert en IMU sensor og en ultralydsensor på et mikrofonstativ der hensikten har vært å bruke mikrofonstativet til å simulere bevegelsene til en gangbro. Her har vi primært jobbet med å estimere avstand i en retning, der vi har brukt akselerometer- og ultralydsensordata til å estimere avstanden ved hjelp av Kalman filter og sensor fusion. Dette har fungert til en viss grad, men fortsatt en del å jobbe med her.

Planen for den neste uken er å fortsette på disse to områdene. Vi vil fokusere mer på å bruke gyroskopet, akselerometeret og magnetometeret i IMU-en til å estimere orienteringen til gangbroa (mikrofonstativet) ved hjelp av Kalman filter og sensor fusion. Vi vil også se på hvordan vi kan kombinere stereokameramålingene med orienteringsestimeringen.

Øvrig diskusjon

Koronasituasjonen er fortsatt slik at vi må jobbe hjemmefra, og det ser ut som at det vil bli slik for resten av bacheloroppgaven. Gruppen har delt seg i to, der Erik Bjørnøy og Isak Gamnes Sneltvedt jobber med måling av posisjon/forflytning ved hjelp av stereokamera, og Erlend Holseker og Arvin Khodabandeh primært jobber med estimering av orientering ved hjelp av IMU og sensor fusion ved hjelp av Kalman filter.

Det ble også nevnt på møtet at vi også kan se etter løsninger som går utenfor de begrensningene som Seaonics har satt for oppgaven.

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 14.00.

Referat sendt av: Erlend Holseker

Referat godkjent av: Isak Gamnes Sneltvedt og Arvin Khodabandeh

Posisjonsestimering

Møtereferat

03.05.2021

I. Start møtet

Erik Bjørnøy startet møtet med styringsgruppen kl. 12.00 den 03.05.21 på Teams.

II. Opprop

Følgende personer var til stede:

Erlend Holseker(sekretær), Erik Bjørnøy(prosjektleder), Arvin Khodabandeh(gruppemedlem), Isak Gamnes Sneltvedt(gruppemedlem) Aleksander Larsen Skrede(hovedveileder) og Daniel Bjørneseth(oppdragsgiver).

III. Saker

a) Gjennomgang av foreløpig prosjektrapport

Første utkast av bachelorrapporten ble gjennomgått. Hovedveileder ba gruppen være oppmerksom på følgende punkter:

- Finne en bedre og mer beskrivende tittel til rapporten.
- Være konsekvent på britisk eller amerikansk engelsk.
- Noen plasser i rapporten er det mange små avsnitt som med fordel kunne blitt samlet til større avsnitt.
- Passe på å ha god setningsbygning. Vær spesifikk, og ikke bruk for mye komma.
- Prosjektet er objektet vi skriver om, ikke rapporten.
    - Eksempel: “The aim of this report is to estimate orientation”, skal være “The aim of this project is to estimate”.
- Kilde/sitering skal ha mellomrom fra ordet foran.
- Ved referering til figurer, tabeller, seksjoner og likninger skal det skrives hva det henvises til.
    - Eksempel: “As seen in 2.3, x is calculated”, skal være “As seen in equation 2.3, x is calculated”.

- Gå over bildene i rapporten og pass på at bilder hentet fra nettet har kildehenvisning. Dette gjelder blandt annet figur 2.3.
- I likninger: Bruk \cdot i stedet for * når det skal multipliseres.
- Bruk bold/italic i stedet for å understreke tekst.
- Setningen "The filter can estimate both past, present, and future states, even when the precise behaviour of the modeled system is unknown" må skrives om da den ikke er helt sann.
- Beskrivelse av utstyret vi har brukt skal skrives under materials. For eksempel Arduino og Software skal flyttes til materials.
- «Choosing Design» flyttes til «Design Choices».
- Kodesnutter som skrives i "minted" skal ha highlighting.
- Vindmølle på engelsk er IKKE wind mill, men **wind turbine**. Steder i rapporten der det har blitt skrevet wind mill, må endres.
- Husk på akademisk skrivemåte. Bruk beskrivende temasetninger i starten av avsnitt. Sign posting.

Øvrig diskusjon

Det vil bli avholdt et nytt møte neste mandag, 10.05, for å gå gjennom neste utkast av rapporten. Tirsdag og torsdag denne uken vil gruppen jobbe med å sy sammen de siste delene av prosjektet på labben på skolen. Resten av tiden vil bli brukt til å skrive rapport.

IV. Møte hevet

Erik Bjørnøy hevet møtet kl. 12.35.

Referat sendt av: Erlend Holseker

Referat godkjent av: Isak Gamnes Sneltvedt og Arvin Khodabandeh

Appendix E

Source Code Orientation Application

GUIApplication.py

```
#This is an application made for a bachelor project.
#The application reads IMU data from an Arduino board and filters the data
#using Kalman filters and Madgwick filter to estimate orientation.
#The application also includes a graphical user interface to plot and visualize
#the data, using the Qt framework.
#
#
#Written by Erlend Holseker and Arvin Khodabandeh.

from PyQt5 import QtCore, QtGui, QtWidgets
from threading import Thread
import time
import cv2
import pyqtgraph as pg
import numpy as np
from random import randint
import math
import serial

from kalman import KalmanFilter
from madgwick import MadgwickFilter
import orientation_conversion
import quaternion
import homogeneous_transform as ht
import frame
import mainwindow

class ControlMainWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(ControlMainWindow, self).__init__(parent)
        self.ui = mainwindow.Ui_MainWindow()
        self.ui.setupUi(self)

        self.numberOfPlottingValues = 300

        self.listenThread = Thread(target=self.readSerialInputs, args=())
        self.listenThread.daemon = True

        # Connecting the menu buttons
        self.ui.RawAccBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(0))
        self.ui.RawGyroBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(1))
        self.ui.RawMagBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(2))
        self.ui.EstValBtn.clicked.connect(lambda : self.ui.WidgetPages.setCurrentIndex(3))

        self.ui.StartBtn.clicked.connect(lambda : self.listenThread.start())

        self.ui.SensorFrameXBtn.setChecked(True)
        self.ui.ShipFrameXBtn.setChecked(True)
        self.ui.SensorFrameYBtn.setChecked(True)
        self.ui.ShipFrameYBtn.setChecked(True)
        self.ui.SensorFrameZBtn.setChecked(True)
        self.ui.ShipFrameZBtn.setChecked(True)

        self.capture = cv2.VideoCapture('seaonics.mp4')

        # Create variables for holding the last 200 plotting values
        self.time_stamps = list(range(self.numberOfPlottingValues))
        ## ESTIMATE GRAPHS ##
        ## Roll ##
        self.est_roll_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.meas_roll_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.ship_pitch_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.est_roll_line = self.ui.EstValGraphX.plot(self.time_stamps, self.est_roll_list, name="Kalman Roll angle", pen=pg.mkPen(color='r'))
        self.ship_pitch_line = self.ui.EstValGraphX.plot(self.time_stamps, self.ship_pitch_list, name="Ship Pitch angle", pen=pg.mkPen(color='g'))
        ## Pitch ##

        self.est_pitch_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.meas_pitch_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.ship_roll_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.est_pitch_line = self.ui.EstValGraphY.plot(self.time_stamps, self.est_pitch_list, name="Kalman Pitch angle", pen=pg.mkPen(color='r'))
        self.ship_roll_line = self.ui.EstValGraphY.plot(self.time_stamps, self.ship_roll_list, name="Ship Roll angle", pen=pg.mkPen(color='g'))
        ## Yaw ##
        self.est_yaw_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.meas_yaw_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.ship_yaw_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.meas_yaw_line = self.ui.EstValGraphZ.plot(self.time_stamps, self.meas_yaw_list, name="Madgwick Yaw angle", pen=pg.mkPen(color='b'))
        self.ship_yaw_line = self.ui.EstValGraphZ.plot(self.time_stamps, self.ship_yaw_list, name="Ship Yaw angle", pen=pg.mkPen(color='g'))

        ## ACCELEROMETER GRAPHS ##
        ## X ##
        self.accX_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.accX_line = self.ui.AccValGraphX.plot(self.time_stamps, self.accX_list, name="Acc X", pen=pg.mkPen(color='b'))
        ## Y ##
        self.accY_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.accY_line = self.ui.AccValGraphY.plot(self.time_stamps, self.accY_list, name="Acc Y", pen=pg.mkPen(color='b'))
        ## Z ##
        self.accZ_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.accZ_line = self.ui.AccValGraphZ.plot(self.time_stamps, self.accZ_list, name="Acc Z", pen=pg.mkPen(color='b'))

        ## GYROSCOPE GRAPHS ##
        ## X ##
        self.gyroX_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.gyroX_line = self.ui.GyroValGraphX.plot(self.time_stamps, self.gyroX_list, name="Gyro X", pen=pg.mkPen(color='b'))
        ## Y ##
        self.gyroY_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.gyroY_line = self.ui.GyroValGraphY.plot(self.time_stamps, self.gyroY_list, name="Gyro Y", pen=pg.mkPen(color='b'))
        ## Z ##
        self.gyroZ_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.gyroZ_line = self.ui.GyroValGraphZ.plot(self.time_stamps, self.gyroZ_list, name="Gyro Z", pen=pg.mkPen(color='b'))

        ## MAGNETOMETER GRAPHS ##
        ## X ##
        self.magX_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.magX_line = self.ui.MagValGraphX.plot(self.time_stamps, self.magX_list, name="Mag X", pen=pg.mkPen(color='b'))
        ## Y ##
        self.magY_list = [0 for _ in range(self.numberOfPlottingValues)]
```

```
        self.magY_line = self.ui.MagValGraphY.plot(self.time_stamps, self.magY_list, name="Mag Y", pen=pg.mkPen(color='b'))
        ## Z ##
        self.magZ_list = [0 for _ in range(self.numberOfPlottingValues)]
        self.magZ_line = self.ui.MagValGraphZ.plot(self.time_stamps, self.magZ_list, name="Mag Z", pen=pg.mkPen(color='b'))

        ## KALMAN VARIABLES ##
        self.A = np.array([[0, 0],
                           [0, 0]], dtype='float')
        self.H = np.array([[1, 0],
                           [0, 1]], dtype='float')
        self.delta_t = 0.05
        ## Kalman Variables Roll ##
        self.x_roll = np.array([[0],                #Rotation angle (position)
                                [0]], dtype='float') #Angular velocity
        self.Q_roll = 0
        self.s2_roll = 0.2 ** 2
        self.est_state_roll = np.zeros(2)
        self.est_angularVel_roll = 0
        self.est_angle_roll = 0
        self.meas_roll = 0
        accelerometerX_variance = 0.00031141365254884923
        roll_variance = 0.01839461414087105
        gyroscopeX_variance = 0.007197688446362619
        self.R_roll = np.array([[roll_variance, 0],
                                [0, gyroscopeX_variance]])
        ## Kalman Variables Pitch ##

        self.x_pitch = np.array([[0],               #Rotation angle (position)
                                 [0]], dtype='float') #Angular velocity
        self.Q_pitch = 0
        self.s2_pitch = 0.2 ** 2
        self.est_state_pitch = np.zeros(2)
        self.est_angularVel_pitch = 0
        self.est_angle_pitch = 0
        self.meas_pitch = 0
        accelerometerY_variance = 0.0005460492817487228
        pitch_variance = 0.010618805958553145
        gyroscopeY_variance = 0.010014742586806145
        self.R_pitch = np.array([[pitch_variance, 0],
                                 [0, gyroscopeY_variance]])
        ## Kalman Variables Yaw ##
        self.x_yaw = np.array([[0],                 #Rotation angle (position)
                               [0]], dtype='float')    #Angular velocity
        self.Q_yaw = 0
        self.s2_yaw = 0.1 ** 2
        self.est_state_yaw = np.zeros(2)
        self.est_angularVel_yaw = 0
        self.est_angle_yaw = 0
        self.meas_yaw = 0
        mad_yaw_variance = 0.2296308583589739
        gyroscopeZ_variance = 0.009201665020393227
        self.R_yaw = np.array([[mad_yaw_variance, 0],
                               [0, gyroscopeY_variance]])
        self.yaw_calc = 0
        self.mad_yaw = 0
        self.mad_pitch = 0
        self.mad_roll = 0
        #Madgwick Filter Gain:  0.09114646853830789
        self.beta = 0.041
        #self.madgwick = MadgwickFilter(beta=self.beta)
        self.accel_meas_norm = np.array([[0], [0], [0]])
        self.q = np.array([0, 0, 0, 0]).transpose()
        # Create Kalman filters
        self.kalman_filter_roll= KalmanFilter(self.A, self.H, self.Q_roll, self.R_roll, self.x_roll)
        self.kalman_filter_pitch = KalmanFilter(self.A, self.H, self.Q_pitch, self.R_pitch, self.x_pitch)
        self.kalman_filter_yaw = KalmanFilter(self.A, self.H, self.Q_yaw, self.R_yaw, self.x_yaw)

        ## Variables to hold data from Arduino ##
        self.time_stamp = 0
        self.last_time_stamp = 0
        self.accelX_current = 0
        self.accelY_current = 0
        self.accelZ_current = 0
        self.magX_current = 0
        self.magY_current = 0
        self.magZ_current = 0
        self.gyroX_current = 0
        self.gyroY_current = 0
        self.gyroZ_current = 0

        self.shipRoll = 0
        self.shipPitch = 0
        self.shipYaw = 0

        # Connect a timer to the update GUI function to update the plots at a given interval
        self.timer = QtCore.QTimer()
        self.timer.setInterval(5)
        self.timer.timeout.connect(self.updateGUI)
        self.timer.start()

        # Starting a thread to capture camera video
        self.videoThread = Thread(target=self.showVideo, args=())
        self.videoThread.daemon = True

        self.videoThread.start()

    def readSerialInputs(self):
        # Initial Arduino reading
        self.madgwick = MadgwickFilter(beta=self.beta, initial_slew=self.ui.EncoderYaw.value())

        self.worldFrame = frame.Frame()
        self.sensorFrame = frame.Frame(parentFrame=self.worldFrame)
        self.shipFrame = frame.Frame(orientation=np.array([0, self.ui.EncoderPitch.value(), self.ui.EncoderYaw.value()]), parentFrame=self.sensorFrame)
        self.sensorFrame.set_child_frame(self.shipFrame)

        init_msg = arduino.readline()
        init_msg_vec = init_msg.decode('utf-8').split(',')
        while len(init_msg_vec) != 11:  # Wait for serial communication to stabilize
```

```
    while len(init_msg_vec) != 11:  # Wait for serial communication to stabilize
        init_msg = arduino.readline()
        init_msg_vec = init_msg.decode('utf-8').split(',')

    while True:
        msg = arduino.readline() #Read everything in the input buffer
        msgVec = msg.decode('utf-8').split(',')
        if len(msgVec) == 11 and msgVec[0] and msgVec[1] and msgVec[2] and msgVec[3] and msgVec[4] and msgVec[5] and msgVec[6] and msgVec[7] and msgVec[8] and msgVec[9] and msgVec[1
            #self.delta_t = float(msgVec[0])
            self.accelX_current = float(msgVec[1])
            self.accelY_current = float(msgVec[2])
            self.accelZ_current = float(msgVec[3])
            self.magX_current = float(msgVec[4])
            self.magY_current = float(msgVec[5])
            self.magZ_current = float(msgVec[6])
            self.gyroX_current = (float(msgVec[7]) + 0.042910521140609635)
            self.gyroY_current = (float(msgVec[8]) - 0.08874139626352016)
            self.gyroZ_current = (float(msgVec[9]) - 0.08517207472959686)

        arduino.reset_input_buffer()
        self.calculateEstimates()
        time.sleep(0.05)

def calculateEstimates(self):
    self.meas_roll = orientation_conversion.get_roll(np.array([[self.accelX_current, self.accelY_current, self.accelZ_current]]), degrees=True)
    self.meas_pitch = orientation_conversion.get_pitch(np.array([[self.accelX_current, self.accelY_current, self.accelZ_current]]), degrees=True)

    # Kalman Estimate Roll
    y_roll = np.array([[self.meas_roll],
                       [self.gyroX_current]])
    self.kalman_filter_roll.predict()
    self.kalman_filter_roll.update(y_roll)
    (x_r, P_r) = self.kalman_filter_roll.get_state()
    self.est_state_roll = x_r.transpose()
    self.est_angularVel_roll = self.est_state_roll.transpose()[1]
    self.est_angle_roll = self.est_state_roll.transpose()[0]

    # Kalman Estimate Pitch
    y_pitch = np.array([[self.meas_pitch],
                        [self.gyroY_current]])
    self.kalman_filter_pitch.predict()
    self.kalman_filter_pitch.update(y_pitch)
    (x_p, P_p) = self.kalman_filter_pitch.get_state()
    self.est_state_pitch = x_p.transpose()
    self.est_angularVel_pitch = self.est_state_pitch.transpose()[1]
    self.est_angle_pitch = self.est_state_pitch.transpose()[0]

    ## MADGWICK ##
    accel_meas = np.array([self.accelX_current, self.accelY_current, self.accelZ_current]).T
    gyro_meas = np.array([(self.gyroX_current*math.pi/180), (self.gyroY_current*math.pi/180), (self.gyroZ_current*math.pi/180)]).T
    mag_meas = np.array([0, self.magX_current, self.magY_current, self.magZ_current]).T

    self.q = self.madgwick.get_estimated_orientation(gyro=gyro_meas, acc=accel_meas, mag=None, delta_t=self.delta_t)

    ################################

    ## GET EULER ANGLES FROM MADGWICK QUATERNION ##
    self.mad_roll, self.mad_pitch, self.mad_yaw = quaternion.quaternion_to_euler(self.q, as_degrees=True)

    # Kalman Estimate Yaw
    y_yaw = np.array([[self.mad_yaw],
                      [self.gyroZ_current]])
    self.kalman_filter_yaw.predict()
    self.kalman_filter_yaw.update(y_yaw)
    (x_y, P_y) = self.kalman_filter_yaw.get_state()
    self.est_state_yaw = x_y.transpose()
    self.est_angularVel_yaw = self.est_state_yaw.transpose()[1]
    self.est_angle_yaw = self.est_state_yaw.transpose()[0]

    # Update parameters
    base_sigma = np.array([[self.delta_t ** 3 / 3, self.delta_t ** 2 / 2],
                           [self.delta_t ** 2 / 2, self.delta_t]])

    self.A = np.array([[1, -self.delta_t],
                       [0, 1]], dtype='float')
    self.Q_roll = self.s2_roll * base_sigma
    self.Q_pitch = self.s2_pitch * base_sigma
    self.Q_yaw = self.s2_yaw * base_sigma

    self.kalman_filter_roll.updateParameters(A=self.A, Q=self.Q_roll)
    self.kalman_filter_pitch.updateParameters(A=self.A, Q=self.Q_pitch)
    self.kalman_filter_yaw.updateParameters(A=self.A, Q=self.Q_yaw)

    self.sensorFrame.update_orientation([float(self.est_angle_roll), -float(self.est_angle_pitch), float(self.mad_yaw)])
    self.shipFrame.update_orientation([self.ui.EncoderPitch.value(), 0, -self.ui.EncoderYaw.value()])
    self.shipFrame.set_parent_frame(self.sensorFrame)
    self.sensorFrame.set_child_frame(self.shipFrame)

    shipAngles = ht.rot_matrix_to_euler(self.shipFrame.get_grandparent_representation())

    self.shipRoll = shipAngles[0]
    self.shipPitch = shipAngles[1]
    self.shipYaw = shipAngles[2]

def updateGUI(self):
    self.time_stamps = self.time_stamps[1:] #Remove first element
    self.time_stamps.append(self.time_stamps[-1] + 1)

    self.est_roll_list = self.est_roll_list[1:]
    self.est_roll_list.append(float(self.est_angle_roll))

    self.ship_pitch_list = self.ship_pitch_list[1:]
    self.ship_pitch_list.append(self.shipPitch)

    self.est_pitch_list = self.est_pitch_list[1:]
    self.est_pitch_list.append(float(self.est_angle_pitch))

    self.ship_roll_list = self.ship_roll_list[1:]
```

```
        self.ship_roll_list = self.ship_roll_list[1:]
        self.ship_roll_list.append(self.shipRoll)

        self.meas_yaw_list = self.meas_yaw_list[1:]
        self.meas_yaw_list.append(float(self.mad_yaw))

        self.ship_yaw_list = self.ship_yaw_list[1:]
        self.ship_yaw_list.append(self.shipYaw)


        self.accX_list = self.accX_list[1:]
        self.accX_list.append(float(self.accelX_current))
        self.accY_list = self.accY_list[1:]

        self.accY_list.append(float(self.accelY_current))
        self.accZ_list = self.accZ_list[1:]
        self.accZ_list.append(float(self.accelZ_current))

        self.gyroX_list = self.gyroX_list[1:]
        self.gyroX_list.append(float(self.gyroX_current))
        self.gyroY_list = self.gyroY_list[1:]
        self.gyroY_list.append(float(self.gyroY_current))
        self.gyroZ_list = self.gyroZ_list[1:]
        self.gyroZ_list.append(float(self.gyroZ_current))

        self.magX_list = self.magX_list[1:]
        self.magX_list.append(float(self.magX_current))
        self.magY_list = self.magY_list[1:]
        self.magY_list.append(float(self.magY_current))
        self.magZ_list = self.magZ_list[1:]
        self.magZ_list.append(float(self.magZ_current))

        self.ui.dialX.setValue(int(self.est_angle_roll))
        self.ui.dialY.setValue(int(self.est_angle_pitch))
        self.ui.dialZ.setValue(int(self.mad_yaw))

        roll_text = ('Estimated Roll: ' + "{:.2f}".format(float(self.est_angle_roll)) + '°' + '\n' +
                     'Ship Pitch: ' + "{:.2f}".format(float(self.shipPitch)) + '°')

        pitch_text = ('Estimated Pitch: ' + "{:.2f}".format(float(self.est_angle_pitch)) + '°' + '\n' +
                      'Ship Roll: ' + "{:.2f}".format(float(self.shipRoll)) + '°')

        yaw_text = ('Estimated Yaw: ' "{:.2f}".format(float(self.mad_yaw)) + '°' + '\n' +
                    'Ship Yaw: ' + "{:.2f}".format(float(self.shipYaw)) + '°')

        self.ui.label_7.setText(roll_text)
        self.ui.label_6.setText(pitch_text)
        self.ui.label_5.setText(yaw_text)

        self.est_roll_line.setData(self.time_stamps, self.est_roll_list)
        self.ship_pitch_line.setData(self.time_stamps, self.ship_pitch_list)
        self.est_pitch_line.setData(self.time_stamps, self.est_pitch_list)
        self.ship_roll_line.setData(self.time_stamps, self.ship_roll_list)
        self.meas_yaw_line.setData(self.time_stamps, self.meas_yaw_list)
        self.ship_yaw_line.setData(self.time_stamps, self.ship_yaw_list)

        self.accX_line.setData(self.time_stamps, self.accX_list)
        self.accY_line.setData(self.time_stamps, self.accY_list)
        self.accZ_line.setData(self.time_stamps, self.accZ_list)
        self.gyroX_line.setData(self.time_stamps, self.gyroX_list)
        self.gyroY_line.setData(self.time_stamps, self.gyroY_list)
        self.gyroZ_line.setData(self.time_stamps, self.gyroZ_list)
        self.magX_line.setData(self.time_stamps, self.magX_list)
        self.magY_line.setData(self.time_stamps, self.magY_list)
        self.magZ_line.setData(self.time_stamps, self.magZ_list)

        if self.ui.SensorFrameXBtn.isChecked():
            self.est_roll_line.show()
        else:
            self.est_roll_line.hide()
        if self.ui.ShipFrameXBtn.isChecked():
            self.ship_pitch_line.show()
        else:
            self.ship_pitch_line.hide()

        if self.ui.SensorFrameYBtn.isChecked():
            self.est_pitch_line.show()
        else:
            self.est_pitch_line.hide()
        if self.ui.ShipFrameYBtn.isChecked():

            self.ship_roll_line.show()
        else:
            self.ship_roll_line.hide()

        if self.ui.SensorFrameZBtn.isChecked():
            self.meas_yaw_line.show()
        else:
            self.meas_yaw_line.hide()
        if self.ui.ShipFrameZBtn.isChecked():
            self.ship_yaw_line.show()
        else:
            self.ship_yaw_line.hide()

    def showVideo(self):
        while True:
            self.ret, self.frame = self.capture.read()
            if self.frame is not None:
                self.rgbImage = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
                self.convertToQtFormat = QtGui.QImage(self.rgbImage.data, self.rgbImage.shape[1], self.rgbImage.shape[0],
                                                      QtGui.QImage.Format_RGB888)
                self.convertToQtFormat = QtGui.QPixmap.fromImage(self.convertToQtFormat)
                self.pixmap = QtGui.QPixmap(self.convertToQtFormat)
                self.resizeImage = self.pixmap.scaled(480,480, QtCore.Qt.KeepAspectRatio)
                QtWidgets.QApplication.processEvents()
                self.ui.CamFeedLabel.setPixmap(self.resizeImage)
                time.sleep(0.025)
            else:
```

```
            self.capture = cv2.VideoCapture('seaonics.mp4')
            #self.capture.release()


if __name__ == '__main__':
    arduino = serial.Serial(port='/dev/cu.usbserial-DN041PFR', baudrate=115200, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS, timeout=0)
    import sys
    app = QtWidgets.QApplication(sys.argv)
    mySW = ControlMainWindow()
    mySW.show()

    sys.exit(app.exec_())
```

```
#Written by Erlend Holseker and Arvin Khodabandeh

from PyQt5 import QtCore, QtGui, QtWidgets
from pyqtgraph import PlotWidget

"""
A class made for creating the GUI layout in an application made for a
bachelor project.

The class is made using the Qt framework.
"""
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1275, 808)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setStyleSheet("background-color: rgb(127, 127, 127);")
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        self.WidgetContents = QtWidgets.QWidget(self.centralwidget)
        self.WidgetContents.setStyleSheet("background-color:rgb(153, 153, 153)")
        self.WidgetContents.setObjectName("WidgetContents")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.WidgetContents)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.WidgetPages = QtWidgets.QStackedWidget(self.WidgetContents)
        self.WidgetPages.setObjectName("WidgetPages")
        self.RawAccPage = QtWidgets.QWidget()
        self.RawAccPage.setObjectName("RawAccPage")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.RawAccPage)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.label_2 = QtWidgets.QLabel(self.RawAccPage)
        self.label_2.setMaximumSize(QtCore.QSize(16777215, 50))
        font = QtGui.QFont()
        font.setFamily("Microsoft Sans Serif")
        font.setPointSize(20)
        self.label_2.setFont(font)
        self.label_2.setAlignment(QtCore.Qt.AlignCenter)
        self.label_2.setObjectName("label_2")
        self.verticalLayout_3.addWidget(self.label_2)
        self.AccValGraphX = PlotWidget(self.RawAccPage)
        self.AccValGraphX.setObjectName("AccValGraphX")
        self.verticalLayout_3.addWidget(self.AccValGraphX)
        self.AccValGraphY = PlotWidget(self.RawAccPage)
        self.AccValGraphY.setObjectName("AccValGraphY")
        self.verticalLayout_3.addWidget(self.AccValGraphY)
        self.AccValGraphZ = PlotWidget(self.RawAccPage)
        self.AccValGraphZ.setObjectName("AccValGraphZ")
        self.verticalLayout_3.addWidget(self.AccValGraphZ)
        self.WidgetPages.addWidget(self.RawAccPage)
        self.RawGyroPage = QtWidgets.QWidget()
        self.RawGyroPage.setObjectName("RawGyroPage")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.RawGyroPage)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.label_3 = QtWidgets.QLabel(self.RawGyroPage)
        self.label_3.setMaximumSize(QtCore.QSize(16777215, 50))
        font = QtGui.QFont()
        font.setFamily("Microsoft Sans Serif")
        font.setPointSize(20)
        self.label_3.setFont(font)
        self.label_3.setAlignment(QtCore.Qt.AlignCenter)
        self.label_3.setObjectName("label_3")
        self.verticalLayout_4.addWidget(self.label_3)
        self.GyroValGraphX = PlotWidget(self.RawGyroPage)
        self.GyroValGraphX.setObjectName("GyroValGraphX")
```

```
66          self.verticalLayout_4.addWidget(self.GyroValGraphX)
67          self.GyroValGraphY = PlotWidget(self.RawGyroPage)
68          self.GyroValGraphY.setObjectName("GyroValGraphY")
69          self.verticalLayout_4.addWidget(self.GyroValGraphY)
70          self.GyroValGraphZ = PlotWidget(self.RawGyroPage)
71          self.GyroValGraphZ.setObjectName("GyroValGraphZ")
72          self.verticalLayout_4.addWidget(self.GyroValGraphZ)
73          self.WidgetPages.addWidget(self.RawGyroPage)
74          self.RawMagPage = QtWidgets.QWidget()
75          self.RawMagPage.setObjectName("RawMagPage")
76          self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.RawMagPage)
77          self.verticalLayout_5.setObjectName("verticalLayout_5")
78          self.label_4 = QtWidgets.QLabel(self.RawMagPage)
79          self.label_4.setMaximumSize(QtCore.QSize(16777215, 50))
80          font = QtGui.QFont()
81          font.setFamily("Microsoft Sans Serif")
82          font.setPointSize(20)
83          self.label_4.setFont(font)
84          self.label_4.setAlignment(QtCore.Qt.AlignCenter)
85          self.label_4.setObjectName("label_4")
86          self.verticalLayout_5.addWidget(self.label_4)
87          self.MagValGraphX = PlotWidget(self.RawMagPage)
88          self.MagValGraphX.setObjectName("MagValGraphX")
89          self.verticalLayout_5.addWidget(self.MagValGraphX)
90          self.MagValGraphY = PlotWidget(self.RawMagPage)
91          self.MagValGraphY.setObjectName("MagValGraphY")
92          self.verticalLayout_5.addWidget(self.MagValGraphY)
93          self.MagValGraphZ = PlotWidget(self.RawMagPage)
94          self.MagValGraphZ.setObjectName("MagValGraphZ")
95          self.verticalLayout_5.addWidget(self.MagValGraphZ)
96          self.WidgetPages.addWidget(self.RawMagPage)
97          self.EstValPage = QtWidgets.QWidget()
98          self.EstValPage.setObjectName("EstValPage")
99          self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.EstValPage)
100         self.verticalLayout_2.setObjectName("verticalLayout_2")
101         self.label = QtWidgets.QLabel(self.EstValPage)
102         self.label.setMaximumSize(QtCore.QSize(16777215, 50))
103         font = QtGui.QFont()
104         font.setFamily("Microsoft Sans Serif")
105         font.setPointSize(20)
106         self.label.setFont(font)
107         self.label.setAlignment(QtCore.Qt.AlignCenter)
108         self.label.setObjectName("label")
109         self.verticalLayout_2.addWidget(self.label)
110         self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
111         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
112         self.verticalLayout_9 = QtWidgets.QVBoxLayout()
113         self.verticalLayout_9.setObjectName("verticalLayout_9")
114         self.label_8 = QtWidgets.QLabel(self.EstValPage)
115         self.label_8.setMinimumSize(QtCore.QSize(150, 0))
116         self.label_8.setMaximumSize(QtCore.QSize(16777215, 30))
117         font = QtGui.QFont()
118         font.setFamily("Microsoft Sans Serif")
119         self.label_8.setFont(font)
120         self.label_8.setAlignment(QtCore.Qt.AlignCenter)
121         self.label_8.setObjectName("label_8")
122         self.verticalLayout_9.addWidget(self.label_8)
123
124         self.SensorFrameXBtn = QtWidgets.QCheckBox(self.EstValPage)
125         self.SensorFrameXBtn.setObjectName("SensorFrameXBtn")
126         self.verticalLayout_9.addWidget(self.SensorFrameXBtn)
127         self.ShipFrameXBtn = QtWidgets.QCheckBox(self.EstValPage)
128         self.ShipFrameXBtn.setObjectName("ShipFrameXBtn")
129         self.verticalLayout_9.addWidget(self.ShipFrameXBtn)
130
131         self.horizontalLayout_2.addLayout(self.verticalLayout_9)
```

```
132         self.EstValGraphX = PlotWidget(self.EstValPage)
133         self.EstValGraphX.setMinimumSize(QtCore.QSize(500, 0))
134         self.EstValGraphX.setObjectName("EstValGraphX")
135         self.horizontalLayout_2.addWidget(self.EstValGraphX)
136         self.verticalLayout_8 = QtWidgets.QVBoxLayout()
137         self.verticalLayout_8.setObjectName("verticalLayout_8")
138         self.dialX = QtWidgets.QDial(self.EstValPage)
139         self.dialX.setStyleSheet("")
140         self.dialX.setMinimum(-90)
141         self.dialX.setMaximum(90)
142         self.dialX.setWrapping(True)
143         self.dialX.setNotchesVisible(True)
144         self.dialX.setObjectName("dialX")
145         self.verticalLayout_8.addWidget(self.dialX)
146         self.label_7 = QtWidgets.QLabel(self.EstValPage)
147         self.label_7.setMaximumSize(QtCore.QSize(16777215, 50))
148         font = QtGui.QFont()
149         font.setFamily("Microsoft Sans Serif")
150         self.label_7.setFont(font)
151         self.label_7.setAlignment(QtCore.Qt.AlignCenter)
152         self.label_7.setObjectName("label_7")
153         self.verticalLayout_8.addWidget(self.label_7)
154         self.horizontalLayout_2.addLayout(self.verticalLayout_8)
155         self.verticalLayout_2.addLayout(self.horizontalLayout_2)
156         self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
157         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
158         self.verticalLayout_10 = QtWidgets.QVBoxLayout()
159         self.verticalLayout_10.setObjectName("verticalLayout_10")
160         self.label_9 = QtWidgets.QLabel(self.EstValPage)
161         self.label_9.setMinimumSize(QtCore.QSize(150, 0))
162         self.label_9.setMaximumSize(QtCore.QSize(16777215, 30))
163         font = QtGui.QFont()
164         font.setFamily("Microsoft Sans Serif")
165         self.label_9.setFont(font)
166         self.label_9.setAlignment(QtCore.Qt.AlignCenter)
167         self.label_9.setObjectName("label_9")
168         self.verticalLayout_10.addWidget(self.label_9)
169         self.SensorFrameYBtn = QtWidgets.QCheckBox(self.EstValPage)
170         self.SensorFrameYBtn.setObjectName("SensorFrameYBtn")
171         self.verticalLayout_10.addWidget(self.SensorFrameYBtn)
172         self.ShipFrameYBtn = QtWidgets.QCheckBox(self.EstValPage)
173         self.ShipFrameYBtn.setObjectName("ShipFrameYBtn")
174         self.verticalLayout_10.addWidget(self.ShipFrameYBtn)
175         self.label_12 = QtWidgets.QLabel(self.EstValPage)
176         self.label_12.setMaximumSize(QtCore.QSize(16777215, 20))
177         font = QtGui.QFont()
178         font.setFamily("Microsoft Sans Serif")
179         self.label_12.setFont(font)
180         self.label_12.setAlignment(QtCore.Qt.AlignCenter)
181         self.label_12.setObjectName("label_12")
182         self.verticalLayout_10.addWidget(self.label_12)
183         self.EncoderPitch = QtWidgets.QDoubleSpinBox(self.EstValPage)
184         self.EncoderPitch.setMaximumSize(QtCore.QSize(200, 16777215))
185         self.EncoderPitch.setMinimum(-90)
186         self.EncoderPitch.setMaximum(90)
187         self.EncoderPitch.setObjectName("EncoderPitch")
188         self.verticalLayout_10.addWidget(self.EncoderPitch)
189         self.horizontalLayout_3.addLayout(self.verticalLayout_10)
190         self.EstValGraphY = PlotWidget(self.EstValPage)
191         self.EstValGraphY.setMinimumSize(QtCore.QSize(500, 0))
192         self.EstValGraphY.setObjectName("EstValGraphY")
193         self.horizontalLayout_3.addWidget(self.EstValGraphY)
194         self.verticalLayout_7 = QtWidgets.QVBoxLayout()
195         self.verticalLayout_7.setObjectName("verticalLayout_7")
196         self.dialY = QtWidgets.QDial(self.EstValPage)
197         self.dialY.setMinimum(-180)
```

```
198         self.dialY.setMaximum(180)
199         self.dialY.setWrapping(True)
200         self.dialY.setNotchesVisible(True)
201         self.dialY.setObjectName("dialY")
202         self.verticalLayout_7.addWidget(self.dialY)
203         self.label_6 = QtWidgets.QLabel(self.EstValPage)
204         self.label_6.setMaximumSize(QtCore.QSize(16777215, 50))
205         font = QtGui.QFont()
206         font.setFamily("Microsoft Sans Serif")
207         self.label_6.setFont(font)
208         self.label_6.setAlignment(QtCore.Qt.AlignCenter)
209         self.label_6.setObjectName("label_6")
210         self.verticalLayout_7.addWidget(self.label_6)
211         self.horizontalLayout_3.addLayout(self.verticalLayout_7)
212         self.verticalLayout_2.addLayout(self.horizontalLayout_3)
213         self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
214         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
215         self.verticalLayout_11 = QtWidgets.QVBoxLayout()
216         self.verticalLayout_11.setObjectName("verticalLayout_11")
217         self.label_10 = QtWidgets.QLabel(self.EstValPage)
218         self.label_10.setMinimumSize(QtCore.QSize(150, 0))
219         self.label_10.setMaximumSize(QtCore.QSize(16777215, 30))
220         self.label_10.setAlignment(QtCore.Qt.AlignCenter)
221         self.label_10.setObjectName("label_10")
222         self.verticalLayout_11.addWidget(self.label_10)
223         self.SensorFrameZBtn = QtWidgets.QCheckBox(self.EstValPage)
224         self.SensorFrameZBtn.setObjectName("SensorFrameZBtn")
225         self.verticalLayout_11.addWidget(self.SensorFrameZBtn)
226         self.ShipFrameZBtn = QtWidgets.QCheckBox(self.EstValPage)
227         self.ShipFrameZBtn.setObjectName("ShipFrameZBtn")
228         self.verticalLayout_11.addWidget(self.ShipFrameZBtn)
229         self.label_11 = QtWidgets.QLabel(self.EstValPage)
230         self.label_11.setMaximumSize(QtCore.QSize(16777215, 20))
231         font = QtGui.QFont()
232         font.setFamily("Microsoft Sans Serif")
233         self.label_11.setFont(font)
234         self.label_11.setAlignment(QtCore.Qt.AlignCenter)
235         self.label_11.setObjectName("label_11")
236         self.verticalLayout_11.addWidget(self.label_11)
237         self.EncoderYaw = QtWidgets.QDoubleSpinBox(self.EstValPage)
238         self.EncoderYaw.setMaximumSize(QtCore.QSize(200, 16777215))
239         self.EncoderYaw.setMinimum(0)
240         self.EncoderYaw.setMaximum(180)
241         self.EncoderYaw.setObjectName("EncoderYaw")
242         self.verticalLayout_11.addWidget(self.EncoderYaw)
243         self.horizontalLayout_4.addLayout(self.verticalLayout_11)
244         self.EstValGraphZ = PlotWidget(self.EstValPage)
245         self.EstValGraphZ.setMinimumSize(QtCore.QSize(500, 0))
246         self.EstValGraphZ.setObjectName("EstValGraphZ")
247         self.horizontalLayout_4.addWidget(self.EstValGraphZ)
248         self.verticalLayout_6 = QtWidgets.QVBoxLayout()
249         self.verticalLayout_6.setObjectName("verticalLayout_6")
250         self.dialZ = QtWidgets.QDial(self.EstValPage)
251         self.dialZ.setStyleSheet("")
252         self.dialZ.setMinimum(0)
253         self.dialZ.setMaximum(360)
254         self.dialZ.setWrapping(True)
255         self.dialZ.setNotchesVisible(True)
256         self.dialZ.setObjectName("dialZ")
257         self.verticalLayout_6.addWidget(self.dialZ)
258         self.label_5 = QtWidgets.QLabel(self.EstValPage)
259         self.label_5.setMaximumSize(QtCore.QSize(16777215, 50))
260         font = QtGui.QFont()
261         font.setFamily("Microsoft Sans Serif")
262         self.label_5.setFont(font)
263         self.label_5.setAlignment(QtCore.Qt.AlignCenter)
```

```
264         self.label_5.setObjectName("label_5")
265         self.verticalLayout_6.addWidget(self.label_5)
266         self.horizontalLayout_4.addLayout(self.verticalLayout_6)
267         self.verticalLayout_2.addLayout(self.horizontalLayout_4)
268         self.WidgetPages.addWidget(self.EstValPage)
269         self.horizontalLayout.addWidget(self.WidgetPages)
270         self.gridLayout.addWidget(self.WidgetContents, 0, 1, 1, 1)
271         self.WidgetMenu = QtWidgets.QWidget(self.centralwidget)
272         self.WidgetMenu.setMaximumSize(QtCore.QSize(500, 16777215))
273         self.WidgetMenu.setObjectName("WidgetMenu")
274         self.gridLayout_2 = QtWidgets.QGridLayout(self.WidgetMenu)
275         self.gridLayout_2.setObjectName("gridLayout_2")
276         self.FrameJustifier = QtWidgets.QFrame(self.WidgetMenu)
277         self.FrameJustifier.setFrameShape(QtWidgets.QFrame.WinPanel)
278         self.FrameJustifier.setFrameShadow(QtWidgets.QFrame.Raised)
279         self.FrameJustifier.setObjectName("FrameJustifier")
280         self.gridLayout_3 = QtWidgets.QGridLayout(self.FrameJustifier)
281         self.gridLayout_3.setObjectName("gridLayout_3")
282         self.CamFeedJustifier = QtWidgets.QWidget(self.FrameJustifier)
283         self.CamFeedJustifier.setObjectName("CamFeedJustifier")
284         self.gridLayout_4 = QtWidgets.QGridLayout(self.CamFeedJustifier)
285         self.gridLayout_4.setObjectName("gridLayout_4")
286         self.CamFeedLabel = QtWidgets.QLabel(self.CamFeedJustifier)
287         self.CamFeedLabel.setText("")
288         self.CamFeedLabel.setAlignment(QtCore.Qt.AlignCenter)
289         self.CamFeedLabel.setObjectName("CamFeedLabel")
290         self.gridLayout_4.addWidget(self.CamFeedLabel, 2, 0, 1, 1)
291         self.TitleLabel = QtWidgets.QLabel(self.CamFeedJustifier)
292         self.TitleLabel.setMaximumSize(QtCore.QSize(16777215, 50))
293         font = QtGui.QFont()
294         font.setFamily("Microsoft Sans Serif")
295         font.setPointSize(22)
296         self.TitleLabel.setFont(font)
297         self.TitleLabel.setAlignment(QtCore.Qt.AlignCenter)
298         self.TitleLabel.setObjectName("TitleLabel")
299         self.gridLayout_4.addWidget(self.TitleLabel, 1, 0, 1, 1)
300         self.gridLayout_3.addWidget(self.CamFeedJustifier, 0, 0, 1, 1)
301         self.PageSelectors = QtWidgets.QFrame(self.FrameJustifier)
302         self.PageSelectors.setStyleSheet("background-color:rgb(153, 153, 153);\n"
303                                          "border-style: outset;\n"
304                                          "border-width: 2px;\n"
305                                          "border-radius: 20px;\n"
306                                          "border-color: grey;\n"
307                                          "padding: 4px;")
308         self.PageSelectors.setFrameShape(QtWidgets.QFrame.WinPanel)
309         self.PageSelectors.setFrameShadow(QtWidgets.QFrame.Sunken)
310         self.PageSelectors.setObjectName("PageSelectors")
311         self.verticalLayout = QtWidgets.QVBoxLayout(self.PageSelectors)
312         self.verticalLayout.setObjectName("verticalLayout")
313         self.RawAccBtn = QtWidgets.QPushButton(self.PageSelectors)
314         self.RawAccBtn.setMinimumSize(QtCore.QSize(0, 70))
315         font = QtGui.QFont()
316         font.setFamily("Microsoft Sans Serif")
317         font.setPointSize(16)
318         self.RawAccBtn.setFont(font)
319         self.RawAccBtn.setStyleSheet("background-color: lightgrey;\n"
320                                      "border-style: outset;\n"
321                                      "border-color: grey;\n"
322                                      "border-width: 2px;\n"
323                                      "border-radius: 10px;\n"
324                                      "padding: 4px;\n"
325                                      "\n"
326                                      "QPushbutton::pressed\n"
327                                      "{\n"
328                                      "background-color: grey;\n"
329                                      "border-style: inset;\n"
```

```
330                         "border-color: grey;\n"
331                         "border-width: 2px;\n"
332                         "border-radius: 10px;\n"
333                         "padding: 4px;\n"
334                         "}")
335         self.RawAccBtn.setCheckable(False)
336         self.RawAccBtn.setChecked(False)
337         self.RawAccBtn.setFlat(False)
338         self.RawAccBtn.setObjectName("RawAccBtn")
339         self.verticalLayout.addWidget(self.RawAccBtn)
340         self.RawGyroBtn = QtWidgets.QPushButton(self.PageSelectors)
341         self.RawGyroBtn.setMinimumSize(QtCore.QSize(0, 70))
342         font = QtGui.QFont()
343         font.setFamily("Microsoft Sans Serif")
344         font.setPointSize(16)
345         self.RawGyroBtn.setFont(font)
346         self.RawGyroBtn.setStyleSheet("background-color: lightgrey;\n"
347                         "border-style: outset;\n"
348                         "border-color: grey;\n"
349                         "border-width: 2px;\n"
350                         "border-radius: 10px;\n"
351                         "padding: 4px;\n"
352                         "\n"
353                         "QPushbutton::pressed\n"
354                         "{\n"
355                         "background-color: grey;\n"
356                         "border-style: inset;\n"
357                         "border-color: grey;\n"
358                         "border-width: 2px;\n"
359                         "border-radius: 10px;\n"
360                         "padding: 4px;\n"
361                         "}")
362         self.RawGyroBtn.setObjectName("RawGyroBtn")
363         self.verticalLayout.addWidget(self.RawGyroBtn)
364         self.RawMagBtn = QtWidgets.QPushButton(self.PageSelectors)
365         self.RawMagBtn.setMinimumSize(QtCore.QSize(0, 70))
366         font = QtGui.QFont()
367         font.setFamily("Microsoft Sans Serif")
368         font.setPointSize(16)
369         self.RawMagBtn.setFont(font)
370         self.RawMagBtn.setStyleSheet("background-color: lightgrey;\n"
371                         "border-style: outset;\n"
372                         "border-color: grey;\n"
373                         "border-width: 2px;\n"
374                         "border-radius: 10px;\n"
375                         "padding: 4px;\n"
376                         "\n"
377                         "QPushbutton::pressed\n"
378                         "{\n"
379                         "background-color: grey;\n"
380                         "border-style: inset;\n"
381                         "border-color: grey;\n"
382                         "border-width: 2px;\n"
383                         "border-radius: 10px;\n"
384                         "padding: 4px;\n"
385                         "}")
386         self.RawMagBtn.setObjectName("RawMagBtn")
387         self.verticalLayout.addWidget(self.RawMagBtn)
388         self.EstValBtn = QtWidgets.QPushButton(self.PageSelectors)
389         self.EstValBtn.setMinimumSize(QtCore.QSize(0, 70))
390         font = QtGui.QFont()
391         font.setFamily("Microsoft Sans Serif")
392         font.setPointSize(16)
393         self.EstValBtn.setFont(font)
394         self.EstValBtn.setStyleSheet("background-color: lightgrey;\n"
395                         "border-style: outset;\n"
```

```
                        "border-color: grey;\n"
                        "border-width: 2px;\n"
                        "border-radius: 10px;\n"
                        "padding: 4px;\n"
                        "\n"
                        "QPushbutton::pressed\n"
                        "{\n"
                        "background-color: grey;\n"
                        "border-style: inset;\n"
                        "border-color: grey;\n"
                        "border-width: 2px;\n"
                        "border-radius: 10px;\n"
                        "padding: 4px;\n"
                        "}")
        self.EstValBtn.setObjectName("EstValBtn")
        self.verticalLayout.addWidget(self.EstValBtn)
        self.gridLayout_3.addWidget(self.PageSelectors, 2, 0, 1, 1)

        self.StartBtn = QtWidgets.QPushButton(self.FrameJustifier)
        #self.StartBtn.setMinimumSize(QtCore.QSize(0, 70))
        font = QtGui.QFont()
        font.setFamily("Microsoft Sans Serif")
        font.setPointSize(16)
        self.StartBtn.setFont(font)
        self.StartBtn.setStyleSheet("background-color: lightgrey;\n"
                        "border-style: outset;\n"
                        "border-color: grey;\n"
                        "border-width: 2px;\n"
                        "border-radius: 10px;\n"
                        "padding: 4px;\n"
                        "\n"
                        "QPushbutton::pressed\n"
                        "{\n"
                        "background-color: grey;\n"
                        "border-style: inset;\n"
                        "border-color: grey;\n"
                        "border-width: 2px;\n"
                        "border-radius: 10px;\n"
                        "padding: 4px;\n"
                        "}")
        self.StartBtn.setObjectName("StartBtn")
        self.gridLayout_3.addWidget(self.StartBtn)

        self.MenuLabel = QtWidgets.QLabel(self.FrameJustifier)
        self.MenuLabel.setMaximumSize(QtCore.QSize(16777215, 30))
        font = QtGui.QFont()
        font.setFamily("Microsoft Sans Serif")
        font.setPointSize(20)
        self.MenuLabel.setFont(font)
        self.MenuLabel.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.MenuLabel.setFrameShadow(QtWidgets.QFrame.Plain)
        self.MenuLabel.setAlignment(QtCore.Qt.AlignCenter)
        self.MenuLabel.setObjectName("MenuLabel")
        self.gridLayout_3.addWidget(self.MenuLabel, 1, 0, 1, 1)
        self.gridLayout_2.addWidget(self.FrameJustifier, 0, 0, 1, 1)
        self.gridLayout.addWidget(self.WidgetMenu, 0, 0, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        self.WidgetPages.setCurrentIndex(3)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label_2.setText(_translate("MainWindow", "Raw Accelerometer Values"))
```

```
462         self.label_3.setText(_translate("MainWindow", "Raw Gyroscope Values"))
463         self.label_4.setText(_translate("MainWindow", "Raw Magnetometer Values"))
464         self.label.setText(_translate("MainWindow", "Estimated Attitude Values For Gangway and Ship"))
465         self.label_8.setText(_translate("MainWindow", "Frame Representation"))
466         self.SensorFrameXBtn.setText(_translate("MainWindow", "Sensor Roll to World"))
467         self.ShipFrameXBtn.setText(_translate("MainWindow", "Ship Pitch to World"))
468         self.label_7.setText(_translate("MainWindow", "Estimated Roll:"))
469         self.label_9.setText(_translate("MainWindow", "Frame Representation"))
470         self.SensorFrameYBtn.setText(_translate("MainWindow", "Sensor Pitch to World"))
471         self.ShipFrameYBtn.setText(_translate("MainWindow", "Ship Roll to World"))
472         self.label_12.setText(_translate("MainWindow", "Encoder Value Boom"))
473         self.label_6.setText(_translate("MainWindow", "Estimated Pitch:"))
474         self.label_10.setText(_translate("MainWindow", "Frame Representation"))
475         self.SensorFrameZBtn.setText(_translate("MainWindow", "Sensor Yaw to Init Position"))
476         self.ShipFrameZBtn.setText(_translate("MainWindow", "Ship Yaw to Init Position"))
477         self.label_11.setText(_translate("MainWindow", "Encoder Value Slew"))
478         self.label_5.setText(_translate("MainWindow", "Estimated Yaw:"))
479         self.TitleLabel.setText(_translate("MainWindow", "Position and Orientation Estimation"))
480         self.RawAccBtn.setText(_translate("MainWindow", "Raw Accelerometer Values"))
481         self.RawGyroBtn.setText(_translate("MainWindow", "Raw Gyroscope Values"))
482         self.RawMagBtn.setText(_translate("MainWindow", "Raw Magnetometer Values"))
483         self.EstValBtn.setText(_translate("MainWindow", "Estimated Attitude Values"))
484         self.StartBtn.setText(_translate("MainWindow", "Start Measuring"))
485         self.MenuLabel.setText(_translate("MainWindow", "Menu"))
486 
487         # Graph showing estimated values around X-axis
488         self.EstValGraphX.setBackground('w')
489         self.EstValGraphX.setTitle("Angle X (Roll) in degrees", color="b", size="15pt")
490         styles = {"color": "#f00", "font-size": "15px"}
491         self.EstValGraphX.setLabel("left", "Roll Angle", **styles)
492         self.EstValGraphX.setLabel("bottom", "Time steps", **styles)
493         self.EstValGraphX.addLegend()
494         self.EstValGraphX.showGrid(x=True, y=True)
495         self.EstValGraphX.setYRange(-180, 180, padding=0)
496 
497         # Graph showing estimated values around Y-axis
498         self.EstValGraphY.setBackground('w')
499         self.EstValGraphY.setTitle("Angle Y (Pitch) in degrees", color="b", size="15pt")
500         styles = {"color": "#f00", "font-size": "15px"}
501         self.EstValGraphY.setLabel("left", "Pitch Angle", **styles)
502         self.EstValGraphY.setLabel("bottom", "Time steps", **styles)
503         self.EstValGraphY.addLegend()
504         self.EstValGraphY.showGrid(x=True, y=True)
505         self.EstValGraphY.setYRange(-90, 90, padding=0)
506 
507         # Graph showing estimated values around Z-axis
508         self.EstValGraphZ.setBackground('w')
509         self.EstValGraphZ.setTitle("Angle Z (Yaw) in degrees", color="b", size="15pt")
510         styles = {"color": "#f00", "font-size": "15px"}
511         self.EstValGraphZ.setLabel("left", "Yaw Angle", **styles)
512         self.EstValGraphZ.setLabel("bottom", "Time steps", **styles)
513         self.EstValGraphZ.addLegend()
514         self.EstValGraphZ.showGrid(x=True, y=True)
515         self.EstValGraphZ.setYRange(-180, 180, padding=0)
516 
517         # Graph showing raw accelerometer data X
518         self.AccValGraphX.setBackground('w')
519         self.AccValGraphX.setTitle("Accelerometer X", color="b", size="15pt")
520         styles = {"color": "#f00", "font-size": "15px"}
521         self.AccValGraphX.setLabel("left", "m/s<sup>2</sup>", **styles)
522         self.AccValGraphX.setLabel("bottom", "Time steps", **styles)
523         self.AccValGraphX.addLegend()
524         self.AccValGraphX.showGrid(x=True, y=True)
525 
526         # Graph showing raw accelerometer data Y
527         self.AccValGraphY.setBackground('w')
```

```
528         self.AccValGraphY.setTitle("Accelerometer Y", color="b", size="15pt")
529         styles = {"color": "#f00", "font-size": "15px"}
530         self.AccValGraphY.setLabel("left", "m/s<sup>2</sup>", **styles)
531         self.AccValGraphY.setLabel("bottom", "Time steps", **styles)
532         self.AccValGraphY.addLegend()
533         self.AccValGraphY.showGrid(x=True, y=True)
534 
535         # Graph showing raw accelerometer data Z
536         self.AccValGraphZ.setBackground('w')
537         self.AccValGraphZ.setTitle("Accelerometer Z", color="b", size="15pt")
538         styles = {"color": "#f00", "font-size": "15px"}
539         self.AccValGraphZ.setLabel("left", "m/s<sup>2</sup>", **styles)
540         self.AccValGraphZ.setLabel("bottom", "Time steps", **styles)
541         self.AccValGraphZ.addLegend()
542         self.AccValGraphZ.showGrid(x=True, y=True)
543 
544         # Graph showing raw gyroscope data X
545         self.GyroValGraphX.setBackground('w')
546         self.GyroValGraphX.setTitle("Gyroscope X", color="b", size="15pt")
547         styles = {"color": "#f00", "font-size": "15px"}
548         self.GyroValGraphX.setLabel("left", "degrees/second", **styles)
549         self.GyroValGraphX.setLabel("bottom", "Time steps", **styles)
550         self.GyroValGraphX.addLegend()
551         self.GyroValGraphX.showGrid(x=True, y=True)
552 
553         # Graph showing raw gyroscope data Y
554         self.GyroValGraphY.setBackground('w')
555         self.GyroValGraphY.setTitle("Gyroscope Y", color="b", size="15pt")
556         styles = {"color": "#f00", "font-size": "15px"}
557         self.GyroValGraphY.setLabel("left", "degrees/second", **styles)
558         self.GyroValGraphY.setLabel("bottom", "Time steps", **styles)
559         self.GyroValGraphY.addLegend()
560         self.GyroValGraphY.showGrid(x=True, y=True)
561 
562         # Graph showing raw gyroscope data Z
563         self.GyroValGraphZ.setBackground('w')
564         self.GyroValGraphZ.setTitle("Gyroscope Z", color="b", size="15pt")
565         styles = {"color": "#f00", "font-size": "15px"}
566         self.GyroValGraphZ.setLabel("left", "degrees/second", **styles)
567         self.GyroValGraphZ.setLabel("bottom", "Time steps", **styles)
568         self.GyroValGraphZ.addLegend()
569         self.GyroValGraphZ.showGrid(x=True, y=True)
570 
571         # Graph showing raw magnetometer data X
572         self.MagValGraphX.setBackground('w')
573         self.MagValGraphX.setTitle("Magnetometer X", color="b", size="15pt")
574         styles = {"color": "#f00", "font-size": "15px"}
575         self.MagValGraphX.setLabel("left", "Micro Tesla (μT)", **styles)
576         self.MagValGraphX.setLabel("bottom", "Time steps", **styles)
577         self.MagValGraphX.addLegend()
578         self.MagValGraphX.showGrid(x=True, y=True)
579 
580         # Graph showing raw magnetometer data Y
581         self.MagValGraphY.setBackground('w')
582         self.MagValGraphY.setTitle("Magnetometer Y", color="b", size="15pt")
583         styles = {"color": "#f00", "font-size": "15px"}
584         self.MagValGraphY.setLabel("left", "Micro Tesla (μT)", **styles)
585         self.MagValGraphY.setLabel("bottom", "Time steps", **styles)
586         self.MagValGraphY.addLegend()
587         self.MagValGraphY.showGrid(x=True, y=True)
588 
589         # Graph showing raw magnetometer data Z
590         self.MagValGraphZ.setBackground('w')
591         self.MagValGraphZ.setTitle("Magnetometer Z", color="b", size="15pt")
592         styles = {"color": "#f00", "font-size": "15px"}
593         self.MagValGraphZ.setLabel("left", "Micro Tesla (μT)", **styles)
```

```
594         self.MagValGraphZ.setLabel("bottom", "Time steps", **styles)
595         self.MagValGraphZ.addLegend()
596         self.MagValGraphZ.showGrid(x=True, y=True)
```

```python
## Written by Erlend Holseker and Arvin Khodabandeh

import numpy as np
import matplotlib.pyplot as plt


class KalmanFilter():
    """
    A class representing a Kalman filter.
    :param A: The A matrix (Representing the state-space model)
    :param H: The H matrix (Used to relate the state to the measurement)
    :param Q: The Q matrix (Process covariance matrix)
    :param R: The R matrix (Measurement noise covariance matrix)
    :param x_0: The initial state
    """
    def __init__(self, A, H, Q, R, x_0):
        # Model parameters
        self.A = A
        self.H = H
        self.Q = Q
        self.R = R

        # Initial state
        self._x = x_0
        self.n = len(self._x)
        self._P = np.zeros((self.n, self.n))

    def predict(self):
        self._x = self.A @ self._x
        self._P = self.A @ self._P @ self.A.transpose() + self.Q

    def update(self, z):
        self.S = self.H @ self._P @ self.H.transpose() + self.R
        self.V = z - self.H @ self._x
        self.K = self._P @ self.H.transpose() @ np.linalg.inv(self.S)

        self._x = self._x + self.K @ self.V
        self._P = self._P - self.K @ self.S @ self.K.transpose()

    def get_state(self):
        return self._x, self._P

    def updateParameters(self, A, Q):
        self.A = A
        self.Q = Q

class MotionModel():
    def __init__(self, A, Q):
        self.A = A
        self.Q = Q

        (m, _) = Q.shape
        self.zero_mean = np.zeros(m)

    def __call__(self, x):
        new_state = self.A @ x + np.random.multivariate_normal(self.zero_mean, self.Q)
        return new_state

class MeasurementModel():
    def __init__(self, H, R):
        self.H = H
        self.R = R

        (n, _) = R.shape
        self.zero_mean = np.zeros(n)
```

```
66
67      def __call__(self, x):
68          measurement = self.H @ x + np.random.multivariate_normal(self.zero_mean, self.R)
69          return measurement
70
71  def create_model_parameters(T=1, s2_x=0.1 ** 2, s2_y=0.1 ** 2, lambda2=0.3 ** 2):
72      # Motion model parameters
73      F = np.array([[1, T],
74                    [0, 1]])
75      base_sigma = np.array([[T ** 3 / 3, T ** 2 / 2],
76                             [T ** 2 / 2, T]])
77
78      sigma_x = s2_x * base_sigma
79      sigma_y = s2_y * base_sigma
80
81      zeros_2 = np.zeros((2, 2))
82      A = np.block([[F, zeros_2],
83                    [zeros_2, F]])
84      Q = np.block([[sigma_x, zeros_2],
85                    [zeros_2, sigma_y]])
86
87      # Measurement model parameters
88      H = np.array([[1, 0, 0, 0],
89                    [0, 0, 1, 0]])
90      R = lambda2 * np.eye(2)
91
92      return A, H, Q, R
93
94  def simulate_system(K, x0):
95      (A, H, Q, R) = create_model_parameters()
96
97      # Create models
98      motion_model = MotionModel(A, Q)
99      meas_model = MeasurementModel(H, R)
100
101     (m, _) = Q.shape
102     (n, _) = R.shape
103
104     state = np.zeros((K, m))
105     meas = np.zeros((K, n))
106
107     # Initial state
108     x = x0
109     for k in range(K):
110         x = motion_model(x)
111         z = meas_model(x)
112
113         state[k, :] = x
114         meas[k, :] = z
115
116     return state, meas
```

```python
## Written by Erlend Holseker and Arvin Khodabandeh

import numpy as np
import math

import quaternion

class MadgwickFilter():
    """
    A class representing a Madgwick filter.
    :param beta: The filter gain representing all mean zero gyroscope errors,
    expressed as the magnitude of a quaternion derivative. It is defined using the
    angular velocity: beta = sqrt(3/4) * omegab, where omegab is the estimated
    mean zero gyroscope measurement error of each axis.
    """
    def __init__(self, beta, initial_slew):
        self.beta = float(beta)
        self.q = quaternion.euler_to_quaternion([0, 0, initial_slew])
        self.update_term = 0
        self.acc_normalized = np.array([0, 0, 0]).T
        self.mag_normalized = np.array([0, 0, 0, 0]).T


    def get_estimated_orientation(self, delta_t, gyro, acc, mag=None):
        """
        Calculates the estimated quaternion representation of the orientation.
        :param delta_t: The time between each measurement
        :param gyro: Array containing the gyroscope measurement
        :param acc: Array containing the accelerometer measurement
        :param mag: Array containing the magnetometer measurement. This should
        be a 4x1 array where the first element is 0, and the rest is the
        magnetometer measurement.
        """
        qw = self.q[0]
        qx = self.q[1]
        qy = self.q[2]
        qz = self.q[3]

        acc_norm = np.linalg.norm(acc)
        if acc_norm > 0:
            self.acc_normalized = np.divide(acc, acc_norm)

        ##ORIENTATION INCREMENT FROM ACC ##
        f_g = np.array([(2*(qx*qz - qw*qy) - self.acc_normalized[0]),
                        (2*(qw*qx - qy*qz) - self.acc_normalized[1]),
                        (2*(0.5 - qx**2 - qy**2) - self.acc_normalized[2])])

        j_g = np.array([(-2*qy, 2*qz, -2*qw, 2*qx),
                        (2*qx, 2*qw, 2*qz, 2*qy),
                        (0, -4*qx, -4*qy, 0)])

        ## ORIENTATION INCREMENT FROM GYRO ##
        q_w_dot = 0.5*quaternion.multiply(self.q, np.array([0, gyro[0], gyro[1], gyro[2]]).transpose())

        if mag is None:
            grad_step = j_g.transpose() @ f_g
            grad_norm = np.linalg.norm(grad_step)

            if grad_norm != 0:
                self.update_term = -self.beta * np.divide(grad_step, grad_norm)
        else:
        ## CORRECT USING MAGNETOMETER ##
            mag_norm = np.linalg.norm(mag)
            if mag_norm > 0:
                self.mag_normalized = np.divide(mag, mag_norm)
```

```
66
67          h = quaternion.multiply(quaternion.multiply(self.q, self.mag_normalized), quaternion.conjugate(self.q))
68
69          bx = math.sqrt(h[1]**2 + h[2]**2)
70          bz = h[3]
71
72          f_b = np.array([(2*bx*(0.5-qy**2-qz**2) + 2*bz*(qx*qz-qw*qy) - self.mag_normalized[1]),
73                          (2*bx*(qx*qy-qw*qz) + 2*bz*(qw*qx+qy*qz) - self.mag_normalized[2]),
74                          (2*bx*(qw*qy+qx*qz) + 2*bz*(0.5-qx**2-qy**2) - self.mag_normalized[3])])
75
76          j_b = np.array([(-2*bz*qy, 2*bz*qz, -4*bx*qy-2*bz*qw, -4*bx*qz+2*bz*qx),
77                          (-2*bx*qz+2*bz*qx, 2*bx*qy+2*bz*qw, 2*bx*qx+2*bz*qz, -2*bx*qw+2*bz*qy),
78                          (2*bx*qy, 2*bx*qz-4*bz*qx, 2*bx*qw-4*bz*qy, 2*bx*qx)])
79
80          f_gb = np.block([f_g,
81                           f_b])
82
83          j_gb = np.block([[j_g],
84                           [j_b]])
85
86          grad_step = j_gb.T @ f_gb
87          grad_norm = np.linalg.norm(grad_step)
88
89          if grad_norm != 0:
90              self.update_term = -self.beta * np.divide(grad_step, grad_norm)
91
92      q_est_dot = q_w_dot + self.update_term
93      q_est = self.q + (q_est_dot * delta_t)
94      self.q = quaternion.normalize(q_est)
95      return self.q
```

```python
## A set of methods for quaternion calculations
#
# Written by Erlend Holseker and Arvin Khodabandeh

import numpy as np
import math

def multiply(q0, q1):
    """
    Multiplies two quaternions.
    BE AWARE OF THE ORDER, QUATERNION MULTIPLICATION IS NON-COMMUTATIVE!
    q0*q1 =/= q1*q0!
    :param q0: array containing the first quaternion
    :param q1: array containing the second quaternion
    """
    if q0.shape != (4, 1):
        q0 = q0.T
    if q1.shape != (4, 1):
        q1 = q1.T
    w0, x0, y0, z0 = q0[0], q0[1], q0[2], q0[3]
    w1, x1, y1, z1 = q1[0], q1[1], q1[2], q1[3]
    w = w0*w1 - x0*x1 - y0*y1 - z0*z1
    x = w0*x1 + x0*w1 + y0*z1 - z0*y1
    y = w0*y1 + y0*w1 + z0*x1 - x0*z1
    z = w0*z1 + z0*w1 + x0*y1 - y0*x1
    return np.array([w, x, y, z])

def conjugate(q0):
    """
    Calculates the quaternion conjugate
    :param q0: array containing the quaternion to be conjugated
    """
    if q0.shape != (4, 1):
        q0 = q0.T
    w0, x0, y0, z0 = q0[0], q0[1], q0[2], q0[3]
    x0, y0, z0 = -x0, -y0, -z0
    return np.array([w0, x0, y0, z0])

def euler_to_quaternion(r):
    """
    Converts from Euler angles(degrees) to Quaternions \n
    :param r: array containing Euler angles(degrees): Roll, Pitch and Yaw
    """
    (yaw, pitch, roll) = (r[2], r[1], r[0])
    (yaw, pitch, roll) = (math.radians(yaw), math.radians(pitch), math.radians(roll))
    qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
    qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)
    qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)
    qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
    return np.array([qw, qx, qy, qz])

def get_axis_angle(q):
    w, x, y, z = q[0], q[1], q[2], q[3]
    norm = math.sqrt(x**2 + y**2 + z**2)
    a = np.divide(([x, y, z]), norm)
    theta = 2 * math.atan2(norm, w)
    roll, pitch, yaw = a * theta
    return np.array(np.degrees([roll, pitch, yaw]))

def to_axis_angle(a):
    a = np.radians(a)
    x, y, z = a[0], a[1], a[2]
    norm = math.sqrt(x**2 + y**2 + z**2)
    x, y, z = np.divide([x, y, z], norm)
    theta = a[0]/x
```

```
66      return np.array([x, y, z, np.degrees(theta)])
67
68  def axis_angle_to_quat(a):
69      theta = np.radians(a[3])
70      s = math.sin(theta/2)
71      w = math.cos(theta/2)
72      x = a[0] * s
73      y = a[1] * s
74      z = a[2] * s
75      return np.array([w, x, y, z])
76
77  def quaternion_to_euler(q, as_degrees=True):
78      """
79      Converts quaternion to Euler angles \n
80      :param q: quaternion as a list: qw, qx, qy, qz
81      """
82      (w, x, y, z) = (q[0], q[1], q[2], q[3])
83      t0 = +2.0 * (w * x + y * z)
84      t1 = +1.0 - 2.0 * (x * x + y * y)
85      roll = math.atan2(t0, t1)
86      t2 = +2.0 * (w * y - z * x)
87      t2 = +1.0 if t2 > +1.0 else t2
88      t2 = -1.0 if t2 < -1.0 else t2
89      pitch = math.asin(t2)
90      t3 = +2.0 * (w * z + x * y)
91      t4 = +1.0 - 2.0 * (y * y + z * z)
92      yaw = math.atan2(t3, t4)
93      if as_degrees:
94          return [np.degrees(roll), np.degrees(pitch), np.degrees(yaw)]
95      else:
96          return [roll, pitch, yaw]
97
98  def quaternion_to_euler_2(q, as_degrees=True):
99      if q.shape != (4, 1):
100         q = q.T
101     q0, q1, q2, q3 = q[0], q[1], q[2], q[3]
102     t0 = 2*(q0*q1 + q2*q3)
103     t1 = 1 - 2*(q1**2 + q2**2)
104     roll = math.atan2(t0, t1)
105     t2 = 2*(q0*q2 - q3*q1)
106     pitch = math.asin(t2)
107     t3 = 2*(q0*q3 + q1*q2)
108     t4 = 1 - 2*(q2**2 + q3**2)
109     yaw = math.atan2(t3, t4)
110     if as_degrees:
111         return [np.degrees(roll), np.degrees(pitch), np.degrees(yaw)]
112     else:
113         return [roll, pitch, yaw]
114
115 def normalize(q):
116     norm = np.linalg.norm(q)
117     q_norm = 0
118     if norm > 0:
119         q_norm = np.divide(q, norm)
120     return q_norm
```

```python
## Written by Arvin Khodabandeh and Erlend Holseker

import numpy as np
import math
import matplotlib.pyplot as plt
import random
import homogeneous_transform as ht
import quaternion
from mpl_toolkits.mplot3d import Axes3D


class Frame(object):


    def __init__(self, position = np.array([0, 0, 0]), orientation = np.array([0, 0, 0]), parentFrame = None, childFrame = None):
        """
        Creates an instance of the Frame class.
        :param position: Position in relation to the parent frame
        :param orientation: Orientation in relation to the parent frame
        """
        self.position = position
        self.orientation = orientation
        self.parentFrame = parentFrame
        self.childFrame = childFrame
        self.representation = None

    def get_position(self):
        return self.position

    def get_orientation(self):
        return self.orientation

    def update_position(self, newPosition):
        """
        Updates the position in relation to the parent frame
        :param newPosition: New position
        """
        self.position = newPosition

    def update_orientation(self, newOrientation):
        """
        Updates the orientation in relation to the parent frame
        """
        self.orientation = newOrientation

    def set_parent_frame(self, parentFrame):
        self.parentFrame = parentFrame

    def set_child_frame(self, childFrame):
        self.childFrame = childFrame

    def get_parent_frame(self):
        return self.parentFrame

    def get_child_frame(self):
        return self.childFrame

    def get_rot_matrix(self):
        return ht.euler_to_rot_matrix(self.orientation)

    def get_trans_matrix(self):
        return ht.pos_to_trans_matrix(self.position)

    def get_representation(self):
        return np.matmul(ht.pos_to_trans_matrix(self.get_position()), ht.euler_to_rot_matrix(self.get_orientation()))
```

```
66
67      def get_quat(self):
68          return ht.euler_to_quaternion(self.get_orientation())
69
70      def get_parent_representation(self):
71          parent_representation = None
72          if self.parentFrame is not None:
73              parent_pos = self.parentFrame.get_position()
74              parent_orient = self.parentFrame.get_orientation()
75              parent_representation = np.matmul(ht.pos_to_trans_matrix(parent_pos), ht.euler_to_rot_matrix(parent_orient))
76          return parent_representation
77
78      def get_grandparent_representation(self):
79          grandparent_representation = None
80          if self.parentFrame.get_parent_frame() is not None:
81              parent_representation = self.get_parent_representation()
82              self.representation = self.get_representation()
83              grandparent_representation = np.matmul(parent_representation, self.representation)
84          return grandparent_representation
```

```python
## Written by Arvin Khodabandeh and Erlend Holseker


import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import time

from mpl_toolkits.mplot3d import Axes3D

def euler_to_quaternion(r):
    """
    Converts from Euler angles(degrees) to Quaternions \n
    :param r: array containing Euler angles(degrees): Roll, Pitch and Yaw
    """
    (yaw, pitch, roll) = (r[2], r[1], r[0])
    (yaw, pitch, roll) = (math.radians(yaw), math.radians(pitch), math.radians(roll))
    qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
    qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)
    qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)
    qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
    return np.array([qw, qx, qy, qz])


def quaternion_to_euler(q):
    """
    Converts quaternion to Euler angles \n
    :param q: quaternion as a list: qw, qx, qy, qz
    """
    (w, x, y, z) = (q[0], q[1], q[2], q[3])
    t0 = +2.0 * (w * x + y * z)
    t1 = +1.0 - 2.0 * (x * x + y * y)
    roll = math.atan2(t0, t1)
    t2 = +2.0 * (w * y - z * x)
    t2 = +1.0 if t2 > +1.0 else t2
    t2 = -1.0 if t2 < -1.0 else t2
    pitch = math.asin(t2)
    t3 = +2.0 * (w * z + x * y)
    t4 = +1.0 - 2.0 * (y * y + z * z)
    yaw = math.atan2(t3, t4)
    return [roll, pitch, yaw]

def quat_to_rot_matrix(q):
    """
    Converts quaternion to a 4x4 rotation matrix. \n
    :param q: quaternion as a list: qw, qx, qy, qz
    """
    R = np.array([[q[0]**2 + q[1]**2 - q[2]**2 - q[3]**2, 2*(q[1]*q[2]-q[0]*q[3]), 2*(q[0]*q[2] + q[1]*q[3]), 0],
                  [2*(q[1]*q[2]+q[0]*q[3]), q[0]**2 - q[1]**2 + q[2]**2 - q[3]**2, 2*(q[2]*q[3]-q[0]*q[1]), 0],
                  [2*(q[1]*q[3]-q[0]*q[2]), 2*(q[0]*q[1] + q[2]*q[3]), q[0]**2 - q[1]**2 - q[2]**2 +q[3]**2, 0],
                  [0, 0, 0, 1]])
    return R

def euler_to_rot_matrix(r, only_Z = False):
    (psi, theta, phi) = (r[2], r[1], r[0])
    c_phi = math.cos(math.radians(phi))
    s_phi = math.sin(math.radians(phi))
    c_theta = math.cos(math.radians(theta))
    s_theta = math.sin(math.radians(theta))
    c_psi = math.cos(math.radians(psi))
    s_psi = math.sin(math.radians(psi))

    Rx = np.array([[1, 0, 0, 0],
                   [0, c_phi, -s_phi, 0],
```

```
                [0, s_phi, c_phi, 0],
                [0, 0, 0, 1]])
    Ry = np.array([[c_theta, 0, s_theta, 0],
                [0, 1, 0, 0],
                [-s_theta, 0, c_theta, 0],
                [0, 0, 0, 1]])
    Rz = np.array([[c_psi, -s_psi, 0, 0],
                [s_psi, c_psi, 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1]])

    if only_Z:
        return Rz
    else:
        R = Rz @ Ry @ Rx
        return R

def to_3x3_rot_matrix(m):
    """
    Extracts a 3x3 matrix from a larger than 3x3 matrix \n
    :param m: matrix to be converted
    """
    R = np.array([[m[0, 0], m[0, 1], m[0, 2]],
                  [m[1, 0], m[1, 1], m[1, 2]],
                  [m[2, 0], m[2, 1], m[2, 2]]])
    return R


def get_rot_matrix_list(data):
    """
    Creates a list of rotation matrices. \n
    :param data: dataframe to extract values.
    """
    orientation = data.iloc[:, 1:4].values
    rot_matrices = []
    for i in range (len(orientation)):
        quat = euler_to_quaternion(orientation[i, :])
        rot_matrix_4x4 = quat_to_rot_matrix(quat)
        rot_matrix = to_3x3_rot_matrix(rot_matrix_4x4)
        rot_matrices.append(rot_matrix)
    return rot_matrices


def rot_matrix_to_euler(r):
    """
    Converts a rotation matrix to euler angles(degrees)\n
    :param r: rotation matrix to convert
    """
    sy = math.sqrt(r[0, 0] * r[0, 0]+ r[1, 0] * r[1, 0])
    singular = sy < 1e-6
    if not singular:
        roll = math.atan2(r[2, 1], r[2,2])
        pitch = math.atan2(-r[2, 0], sy)
        yaw = math.atan2(r[1, 0], r[0, 0])
    else:
        roll = math.atan2(-r[1, 2], r[1, 1])
        pitch = math.atan2(-r[2, 0], sy)
        yaw = 0
    return np.array([math.degrees(roll), math.degrees(pitch), math.degrees(yaw)])


def pos_to_trans_matrix(pos):
    """
    Creates a 4x4 translation matrix from given coordinates\n
    :param pos: how far the object should translate in x, y and z direction
    """
```

```
132     (new_x, new_y, new_z) = (pos[0], pos[1], pos[2])
133     trans = np.array([[1, 0, 0, new_x], [0, 1, 0, new_y], [0, 0, 1, new_z], [0, 0, 0, 1]])
134     return trans
135 
136 def get_pos(representation):
137     """
138     Get the position from a 4x4 homogeneous transformation matrix representation\n
139     :param representation: current representation
140     """
141     pos = representation[0:3, 3]
142     return pos
143 
144 def get_orientation(representation):
145     """
146     Get the orientation from a 4x4 homogeneous transformation matrix representation\n
147     :param representation: current representation
148 
149     """
150     rot_matrix = representation[0:3, 0:3]
151     orientation = rot_matrix_to_euler(rot_matrix)
152     return orientation
```

orientation_conversion.py

```python
## A set of methods to calculate orientation based on IMU data
#
# Written by Erlend Holseker and Arvin Khodabandeh

import numpy as np
import math

g_const = 9.80665

def get_pitch(a, degrees=True):
    norm = np.linalg.norm(a)
    phi = 0
    if norm > 0:
        a_norm = -np.divide(a.T, norm)
        phi = math.atan2(a_norm[0], math.sqrt(a_norm[1]**2 + a_norm[2]**2))
    if degrees:
        return math.degrees(phi)
    else:
        return phi

def get_roll(a, degrees=True):
    norm = np.linalg.norm(a)
    theta = 0
    if norm > 0:
        a_norm = np.divide(a.T, norm)
        theta = math.atan2(a_norm[1], math.sqrt(a_norm[0]**2 + a_norm[2]**2))
    if degrees:
        return math.degrees(theta)
    else:
        return theta

def get_yaw(pitch, roll, m, degrees=True):
    phi = pitch
    theta = roll
    c_phi = math.cos(math.radians(phi))
    s_phi = math.sin(math.radians(phi))
    c_theta = math.cos(math.radians(theta))
    s_theta = math.sin(math.radians(theta))
    mag = m.T
    # norm = np.linalg.norm(m)
    norm = math.sqrt((mag[0]**2) + (mag[1]**2) + (mag[2]**2))
    if norm > 0:
        m_norm = np.divide(mag, norm)

        XH = m_norm[0]*math.cos(math.radians(pitch)) + m_norm[1]*math.sin(math.radians(pitch))*math.sin(math.radians(roll)) + m_norm[2]*math.sin(math.radians(pitch))*math.cos(math.radians(roll))

        YH = m_norm[1]*math.cos(math.radians(roll)) + m_norm[2]*math.sin(math.radians(roll))

        psi = math.atan2(-YH, XH)

        if degrees:
            return math.degrees(psi)
        else:
            return psi
    else:
        return 0
```

Appendix F

Source Code Positioning Application

calibrate_rectify_undistort.py

```
"""
@AUTHORS Erik Bjørnøy, Isak Gamnes

_____________
Calibrates, rectifies and undistorts stereo camera images.
Saves cameras intrinsic and extrinsics.
"""

import cv2
assert cv2.__version__[0] >= '3', 'The fisheye module requires opencv version >= 3.0.0'
import numpy as np
import os
import glob
from tqdm import tqdm


# Set the path to the images captured by the left and right cameras
pathL = "myPath/left"
pathR = "myPath/rigth"

# Termination criteria for refining the detected corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)


objp = np.zeros((9*6,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)

img_ptsL = []
img_ptsR = []
obj_pts = []

for i in tqdm(range(1,22)):
 imgL = cv2.imread(pathL+"%s_left.png"%i)
 imgR = cv2.imread(pathR+"%s_right.png"%i)
 imgL_gray = cv2.imread(pathL+"%s_left.png"%i,0)
 imgR_gray = cv2.imread(pathR+"%s_rigth.png"%i,0)

 outputL = imgL.copy()
 outputR = imgR.copy()

 retR, cornersR =  cv2.findChessboardCorners(outputR,(9,6),cv2.CALIB_CB_ADAPTIVE_THRESH+cv2.CALIB_CB_FAST_CHECK+cv2.CALIB_CB_NORMALIZE_IMAGE)
 retL, cornersL = cv2.findChessboardCorners(outputL,(9,6),cv2.CALIB_CB_ADAPTIVE_THRESH+cv2.CALIB_CB_FAST_CHECK+cv2.CALIB_CB_NORMALIZE_IMAGE)

 if retR and retL:
  obj_pts.append(objp)
  cv2.cornerSubPix(imgR_gray,cornersR,(11,11),(-1,-1),criteria)
  cv2.cornerSubPix(imgL_gray,cornersL,(11,11),(-1,-1),criteria)
  cv2.drawChessboardCorners(outputR,(9,6),cornersR,retR)
  cv2.drawChessboardCorners(outputL,(9,6),cornersL,retL)
  cv2.imshow('cornersR',outputR)
  cv2.imshow('cornersL',outputL)
       cv2.imwrite("cornersR_%s.png"%i, outputR)
       cv2.imwrite("cornersL_%s.png"%i, outputL)
  cv2.waitKey(0)

  img_ptsL.append(cornersL)
  img_ptsR.append(cornersR)


K_l = np.zeros((3, 3))
D_l = np.zeros((4, 1))
K_r = np.zeros((3, 3))
D_r = np.zeros((4, 1))

# Calibrating left camera

retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(obj_pts,img_ptsL,imgL_gray.shape[::-1],K_l,D_l)
hL,wL= imgL_gray.shape[:2]
new_mtxL, roiL= cv2.getOptimalNewCameraMatrix(mtxL,distL,(wL,hL),1,(wL,hL))

# Calibrating right camera
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(obj_pts,img_ptsR,imgR_gray.shape[::-1],K_r,D_r)
hR,wR= imgR_gray.shape[:2]
new_mtxR, roiR= cv2.getOptimalNewCameraMatrix(mtxR,distR,(wR,hR),1,(wR,hR))


flags = 0
flags |= cv2.CALIB_FIX_INTRINSIC
# Here we fix the intrinsic camara matrixes so that only Rot, Trns, Emat and Fmat are calculated.
# Hence intrinsic parameters are the same

criteria_stereo= (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)


# This step is performed to transformation between the two cameras and calculate Essential and Fundamenatl matrix
retS, new_mtxL, distL, new_mtxR, distR, Rot, Trns, Emat, Fmat = cv2.stereoCalibrate(obj_pts, img_ptsL, img_ptsR, new_mtxL, distL, new_mtxR, distR, imgL_gray.shape[::-1], criteria_stereo, flags)


# Rectifying
rectify_scale= 1
rect_l, rect_r, proj_mat_l, proj_mat_r, Q, roiL, roiR= cv2.stereoRectify(new_mtxL, distL, new_mtxR, distR, imgL_gray.shape[::-1], Rot, Trns, rectify_scale,(0,0))


#Undistorting

Left_Stereo_Map= cv2.initUndistortRectifyMap(new_mtxL, distL, rect_l, proj_mat_l,
                                         imgL_gray.shape[::-1], cv2.CV_16SC2)
Right_Stereo_Map= cv2.initUndistortRectifyMap(new_mtxR, distR, rect_r, proj_mat_r,
                                          imgR_gray.shape[::-1], cv2.CV_16SC2)

#dispaly original image
cv2.imshow("Right bad", imgR)
bad_img = imgR
Right_nice= cv2.remap(imgR, Right_Stereo_Map[0], Right_Stereo_Map[1], cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
```

calibrate_rectify_undistort.py

```
cv2.imshow('Right nice', imgR)
cv2.waitKey(0)

#Save parameters to xml file
print("Saving paraeters ......")
cv_file = cv2.FileStorage("improved_params4.xml", cv2.FILE_STORAGE_WRITE)
cv_file.write("Left_Stereo_Map_x",Left_Stereo_Map[0])
cv_file.write("Left_Stereo_Map_y",Left_Stereo_Map[1])
cv_file.write("Right_Stereo_Map_x",Right_Stereo_Map[0])
cv_file.write("Right_Stereo_Map_y",Right_Stereo_Map[1])
cv_file.release()


#Rectify images
Left_nice= cv2.remap(imgL,Left_Stereo_Map[0],Left_Stereo_Map[1], cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
Right_nice= cv2.remap(imgR,Right_Stereo_Map[0],Right_Stereo_Map[1], cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)


# Show rectifyed images
cv2.imshow("Left image after rectification", Left_nice)
cv2.imshow("Right image after rectification", Right_nice)
cv2.waitKey(0)

out = Right_nice.copy()
out[:,:,0] = Right_nice[:,:,0]
out[:,:,1] = Right_nice[:,:,1]
out[:,:,2] = Left_nice[:,:,2]


cv2.imshow("Output image", out)
cv2.waitKey(0)
```

```
"""
@AUTHORS Isak Gamnes, Erik Bjornoy

_____
Takes input stream from stereocamera and IMU sensor and processes the data.
"""

import cv2
import numpy as np
import time
import math
import depthai as dai
import open3d as o3d
import SerialCom
from threading import Thread

first_run = True

fov = 68.8
fps = 60

# Yellow lower and upper bound in hsv
yellowLower = (15,60,50)
yellowUpper = (45,200,255)

cx_ref = 0
cy_ref = 0
z_depth_ref = 0
ref_radius = 0
current_contour_color = [0,255,0]

def get_contour_center(contour):
    # Get the moments in the contour
    M = cv2.moments(contour)
    cx=-1
    cy=-1
    # Make sure not to divide by 0
    if(M['m00']!=0):
        # Calculate x and y coordinates for the center of the contour
        cx=int(M['m10']/M['m00'])
        cy=int(M['m01']/M['m00'])
    return cx,cy

def convert_2_HSV(frame, show=1):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    if show:
        cv2.imshow('HSV frame', hsv)
    return hsv

def get_mask_from_hsv_frame(hsv_frame, lowerValues, upperValues, normalized_mask=0):
    # Define a mask using the lower and upper bound
    color_mask = cv2.inRange(hsv_frame, lowerValues, upperValues)
    if normalized_mask:
        # Create nparray to multiply with depth image to filter out depth values which are not in the color area
        color_mask_normalized = color_mask/255
        return color_mask, color_mask_normalized
    return color_mask

def get_grey_image_from_frame(frame):
    blurred_frame = cv2.GaussianBlur(frame, (5, 5), 0)
    grey_frame = cv2.cvtColor(blurred_frame, cv2.COLOR_BGR2GRAY)
    return grey_frame

def get_biggest_contour(mask, return_area=0):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    biggest_area = 0
    biggest_contour = 0
    for contour in contours:
        area = cv2.contourArea(contour)

        if area > biggest_area:
            biggest_area = area
```

```
            biggest_contour = contour

    if return_area:
        return biggest_contour, biggest_area
    return biggest_contour

# Init serial communication with arduino
ser = SerialCom.SerialCommunication('/dev/ttyUSB0')

# Start defining a pipeline
pipeline = dai.Pipeline()

"""
Defining the RGB camera
"""
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(1920, 1080)
camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
camRgb.setInterleaved(False)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)

# Create output
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
camRgb.preview.link(xoutRgb.input)

"""
Defining the Stereo camera
"""
# Define a source - two mono (grayscale) cameras
monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()
spatialLocationCalculator = pipeline.createSpatialLocationCalculator()

xoutDepth = pipeline.createXLinkOut()
xoutSpatialData = pipeline.createXLinkOut()
xinSpatialCalcConfig = pipeline.createXLinkIn()

xoutDepth.setStreamName("depth")
xoutSpatialData.setStreamName("spatialData")
xinSpatialCalcConfig.setStreamName("spatialCalcConfig")

# MonoCamera
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

outputDepth = True
outputRectified = False
lrcheck = False
subpixel = False

# StereoDepth
stereo.setOutputDepth(outputDepth)
stereo.setOutputRectified(outputRectified)
stereo.setConfidenceThreshold(255)


stereo.setLeftRightCheck(lrcheck)
stereo.setSubpixel(subpixel)

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

"""
Starting the spatial calculator and defining the ROI
"""
spatialLocationCalculator.passthroughDepth.link(xoutDepth.input)
stereo.depth.link(spatialLocationCalculator.inputDepth)

topLeft = dai.Point2f(0.0, 0.0)
bottomRight = dai.Point2f(0.0, 0.0)
```

```
147
148 spatialLocationCalculator.setWaitForConfigInput(False)
149 config = dai.SpatialLocationCalculatorConfigData()
150 config.depthThresholds.lowerThreshold = 100
151 config.depthThresholds.upperThreshold = 10000
152 config.roi = dai.Rect(topLeft, bottomRight)
153 spatialLocationCalculator.initialConfig.addROI(config)
154 spatialLocationCalculator.out.link(xoutSpatialData.input)
155 xinSpatialCalcConfig.out.link(spatialLocationCalculator.inputConfig)
156
157 """
158 Defining and starting the pipeline
159 """
160 device = dai.Device(pipeline)
161 device.startPipeline()
162
163 """
164 Initiating the output queues for both depth frame and RBG frame
165 """
166 depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
167 spatialCalcQueue = device.getOutputQueue(name="spatialData", maxSize=4, blocking=False)
168 spatialCalcConfigInQueue = device.getInputQueue("spatialCalcConfig")
169
170 qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
171
172 previous_cx = 0
173 z_ref_noted = False
174
175 while True:
176     #_, frame = cap.read()
177     ser.update_imu_data()
178     inDepth = depthQueue.get() # blocking call, will wait until a new data has arrived
179     inDepthAvg = spatialCalcQueue.get() # blocking call, will wait until a new data has arrived
180
181     # Get depth and RGB image. Merge images to RGBD image
182     depthFrame = inDepth.getFrame()
183     spatialData = inDepthAvg.getSpatialLocations()
184
185     inRgb = qRgb.get()  # blocking call, will wait until a new data has arrived
186     frame = inRgb.getCvFrame()
187     frame = cv2.flip(frame, -1)
188
189     if(first_run):
190         w = camRgb.getResolutionWidth()
191         h = camRgb.getResolutionHeight()
192         deg_per_pix = fov/w
193         topLeftX = 0.0
194         topLeftY = 0.0
195         bottomRightX = 1.0
196         bottomRightY = 1.0
197         imu_euler_x = 0.
198     else:
199         topLeftX = float((cx/w)-0.025)
200         topLeftY = float((cy/h)-0.025)
201         if topLeftX < 0.0:
202             topLeftX = 0.0
203         if topLeftY < 0.0:
204             topLeftY = 0.0
205
206         bottomRightX = float((cx/w)+0.025)
207         bottomRightY = float((cy/h)+0.025)
208         if bottomRightX > 1.0:
209             bottomRightX = 1.0
210         if bottomRightY > 1.0:
211             bottomRightY = 1.0
212
213         imu_euler_x, _, _ = ser.get_euler()
214         #imu_euler_x = -1 * imu_euler_x
215
216         topLeft_tuple = [topLeftX, topLeftY]
217         bottomRight_tuple = [bottomRightX, bottomRightY]
218         topLeft = dai.Point2f(topLeft_tuple[0], topLeft_tuple[1])
219         bottomRight = dai.Point2f(bottomRight_tuple[0], bottomRight_tuple[1])
```

```
220
221     config.roi = dai.Rect(topLeft, bottomRight)
222     cfg = dai.SpatialLocationCalculatorConfig()
223     cfg.addROI(config)
224     spatialCalcConfigInQueue.send(cfg)
225
226     hsv = convert_2_HSV(frame, 0)
227     yellow_mask = get_mask_from_hsv_frame(hsv, yellowLower, yellowUpper)
228
229     # Find the biggest contour and extract the center and radius of the contour
230     biggest_contour = get_biggest_contour(yellow_mask)
231     try:
232         ((x,y), radius) = cv2.minEnclosingCircle(biggest_contour)
233     except cv2.error:
234         print('Cannot find contour')
235     cx, cy = get_contour_center(biggest_contour)
236     # If this is the first run, the contour center and radius is stored as a reference for future images
237     if(first_run):
238         cx_ref = cx
239         cy_ref = cy
240         ref_radius = radius
241
242     # Calculate the current offset compared to the reference center point
243     pixel_offset_x = cx_ref - cx
244     pixel_offset_y = cy_ref - cy
245
246     # Convert the pixel offset to degrees by multiplying the pixel offset with degrees per pixel
247     camera_angle_offset = pixel_offset_x*deg_per_pix
248     x_angle_offset = camera_angle_offset - imu_euler_x
249     y_angle_offset = pixel_offset_y*deg_per_pix
250
251     abs_pixel_offset_x = math.sqrt(x_angle_offset**2)
252
253     if(abs_pixel_offset_x < 20):
254         current_contour_color = [0,255,0]
255     elif(abs_pixel_offset_x >= 20 and abs_pixel_offset_x < 45):
256         current_contour_color = [0,128,128]
257     else:
258         current_contour_color = [0,0,255]
259
260     for depthData in spatialData:
261         roi = depthData.config.roi
262         roi = roi.denormalize(width=depthFrame.shape[1], height=depthFrame.shape[0])
263         pt1 = (int(topLeftX*w), int(topLeftY*h))

264         pt2 = (int(bottomRightX*w), int(bottomRightY*h))
265         cv2.circle(yellow_mask, (cx,cy), 15, current_contour_color, -1)
266
267         color = (255,255,255)
268         cv2.rectangle(frame, pt1=pt1, pt2=pt2, color=color)
269         z_depth = int(depthData.spatialCoordinates.z)
270         cv2.putText(frame, f"Z: {z_depth} mm", (int(topLeftX) + 10, int(topLeftY) + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, (255,255,255))
271
272     if not z_ref_noted and (z_depth > 5 or z_depth < -5):
273         z_depth_ref = z_depth
274         z_translatoric_offset = 0.
275         z_ref_noted = True
276     else:
277         z_translatoric_offset = z_depth_ref - z_depth
278
279     x_translatoric_offset = z_depth * math.tan(math.radians(x_angle_offset))
280
281     cv2.imshow("Stream", frame)
282     print("Translatoric offset in x direction: {0:.2f} cm".format(x_translatoric_offset/10))
283     print("Translatoric offset in z direction: {0:.2f} cm".format(z_translatoric_offset/10))
284     print('Camera angle offset: {0:.2f} deg'.format(camera_angle_offset))
285     print("IMU Euler data: {0:.2f} deg".format(imu_euler_x))
286     print("Angle offset: {0:.2f} deg".format(x_angle_offset))
287
288     """if not first_run:
289         dt = time.time() - startTime
290         cx_dot = (cx - previous_cx)/dt
291     else:
292         dt = 0
293         cx_dot = 0
```

```
        cx_dot = 0
        first_run = False

    x_ang_dot = cx_dot * deg_per_pix"""

    first_run = False
    startTime = time.time()
    time.sleep((2/fps))

    key = cv2.waitKey(1)
    if key == 27:
        break

cv2.destroyAllWindows()
```

```
"""
@AUTHORS Isak Gamnes, Erik Bjornoy

_____________
Plots graph of different stereo camera baselines defined by it's parameters.
"""


import matplotlib.pyplot as plt
import numpy as np
import math


D = np.arange(0.5,40,0.1).tolist()
D = [round(num, 1) for num in D]

x_1 = []
x_2 = []
x_3 = []
x_4 = []
fov = 65
h_res = 2600
B_1 = 0.1
B_2 = 0.5
B_3 = 1.0
B_4 = 1.5

for i in range(len(D)):
    delta_x =  (B_1 * h_res)/(2*math.tan(fov/2)*D[i])
    x_1.append(round(delta_x))

for i in range(len(D)):
    delta_x =  (B_2 * h_res)/(2*math.tan(fov/2)*D[i])
    x_2.append(round(delta_x))

for i in range(len(D)):
    delta_x =  (B_3 * h_res)/(2*math.tan(fov/2)*D[i])
    x_3.append(round(delta_x))

for i in range(len(D)):
    delta_x =  (B_4 * h_res)/(2*math.tan(fov/2)*D[i])
    x_4.append(round(delta_x))

fig, ax = plt.subplots()
ax.plot(D, x_1)
ax.plot(D, x_2)
ax.plot(D, x_3)
ax.plot(D, x_4)
ax.legend(['Baseline = {}m'.format(B_1), 'Baseline = {}m'.format(B_2), 'Baseline = {}m'.format(B_3), 'Baseline = {}m'.format(B_4)])
ax.set(xlabel='Distance (m)', ylabel='x - x\' (pixels)',
    title='Detection range of stereo camera with varying Baseline')
ax.grid()

fig.savefig("plot-baselines.png")
plt.show()
```

```
"""
@AUTHORS Isak Gamnes, Erik Bjornoy

_____________
Serial communication
"""

# Importing libraries
import serial
import os
from time import sleep
import math

class SerialCommunication(object):
    """
    Serialconnection handler.
    """


    def __init__(self, port, baudrate=115200):
        """
        Establishes a connection to the given port.
        @port : where your device is connected
        @baudrate : the specified connection speed
                    (9600, 19200, 28800, 57600, 115200)
        """

        self.port = port
        self.baudrate = baudrate
        try:
            self.connection = serial.Serial(port, baudrate)
            sleep(2)
        except serial.SerialException as se:
            self.connection = None

        self.euler_x = 0.
        self.euler_y = 0.
        self.euler_z = 0.

    def isConnected(self):
        """
        Checks if the connection is established.
        @return False if not connected
                else True
        """

        result = False
        if self.connection is not None:
            result = True
        else:
         print("Not connected")
        return result

    def readInputStream(self):
        """
        Read data sent trough Serial.
        @return decoded message
        """
        if self.isConnected():
            raw = self.connection.readline()
            data = raw.decode('latin-1')
            return data.rstrip('\n')

    def sendOutputStream(self, data):
        """
```

```
66          Send data trough Serial.
67          """
68          addEndmaker = data + '\n'
69          self.connection.write(addEndmaker.encode())
70
71      def disconnect(self):
72          """
73          Disconnect the connection
74          @return True if sucsessfully disconnected
75          """
76
77          self.connection.close()
78          return True
79
80      def update_imu_data(self):
81          self.connection.flushInput()
82          sleep(0.01)
83          ser_input = self.readInputStream()
84          data_list = None
85          if ser_input is not None:
86              data_list = ser_input.split(sep=',', maxsplit=-1)
87          else:
88              print('Serial input = None')
89          if data_list is not None:
90              if len(data_list) > 0:
91                  #print('Length of data list: {}'.format(len(data_list)))
92                  try:
93                      self.euler_x = float(data_list[0])
94                      #self.euler_y = float(data_list[1])
95                      #self.euler_z = float(data_list[2])
96                  except ValueError as ve:
97                      print(ve)
98              else:
99                  self.euler_x = 0
100                 self.euler_y = 0
101                 self.euler_z = 0
102
103     def get_euler(self):
104         return self.euler_x, self.euler_y, self.euler_z
```

```
"""
@AUTHORS Erik Bjørnøy, Isak Gamnes

_____________
Draws SIFT keypoints in two images
"""

import numpy as np
import cv2

img1 = cv2.imread('myPath/left.png')
img2 = cv2.imread('myPath/right.png')

gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()

kp1 = sift.detect(gray1,None)
kp2 = sift.detect(gray2,None)

img1 = cv2.drawKeypoints(gray1,kp1,img1)
img2 = cv2.drawKeypoints(gray2,kp2,img2)

cv2.imwrite('left_features.png',img1)
cv2.imwrite('right_features.png',img2)
cv2.imshow("left",img1)
cv2.imshow("right",img2)

cv2.waitKey(0)
```

```
"""
@AUTHORS Isak Gamnes, Erik Bjornoy

_____
Calculates standard deviation, avrage offest, avrage and variance.
"""

import math
import sys


continue_calculating = True
n = 0
x_avg = 0
std_dev = 0
MSE = 0

x = []
y = float(input('Enter the expected value \n'))

while continue_calculating:
    usrInput = input('Enter the next number as float, or type \'done\' to calculate the results \n')
    if usrInput == str('done'):
        continue_calculating = False
    else:
        x.append(float(usrInput))
        x_avg += float(usrInput)
        n +=1

if n <= 0:
    print('Please enter more data values')
    print('Exiting...')
    sys.exit(0)

x_avg = x_avg/n

for i in range(len(x)):
    MSE += (x[i] - x_avg)**2

std_dev = math.sqrt(MSE / n)
variance = std_dev**2

print('-'*25)
if n > 1:
    print('Calculating a total of {} inputs'.format(n))
elif n == 1:
    print('Calculating 1 input')

print('Average: {}'.format(x_avg))
print('Average Offset: {}'.format(str(math.sqrt(float(x_avg)**2) - math.sqrt(float(y**2)))))
print('Standard deviation: {}'.format(std_dev))
print('Variance: {}'.format(variance))
```

```python
"""
@AUTHORS Erik Bjornoy, Isak Gamnes

_____________________
Python Module for unistorting a fisheye stero camera image pair.
Does not yet take live input stream.
"""

import cv2
import os

class Undistort:
    def __init__(self, improved_parameters, pathL, pathR):
        """
        Initiates the parameters for camera undistortion

        :param improved_parameters: path to xml file containing undistrortion parameters
        :param pathL: path to left image
        :param pathR: path to right image
        """
        self.values = cv2.FileStorage(improved_parameters, cv2.FILE_STORAGE_READ)
        self.left_x = self.values.getNode("Left_Stereo_Map_x").mat()
        self.left_y = self.values.getNode("Left_Stereo_Map_y").mat()
        self.right_x = self.values.getNode("Right_Stereo_Map_x").mat()
        self.right_y = self.values.getNode("Right_Stereo_Map_y").mat()

        self.imgL = cv2.imread(pathL)
        self.imgR = cv2.imread(pathR)




    def show_image(self):
        """
        View original image
        """
        cv2.imshow("Left image before rectification", self.imgL)
        cv2.imshow("Right image before rectification", self.imgR)
        cv2.waitKey(0)



    def undistort_and_rectify(self, new_name_left_img, new_name_right_img, save_img=False, show_image=True):
        """
        Undistorts, saves and showes images.s

        :param new_name_left_img: path and name to save left image
        :param new_name_right_img: path and name to save right image
        :param save_img (bool): save image as png, by defult False
        :param show_img (bool): show image, by defult True
        """

        Left_nice= cv2.remap(self.imgL, self.left_x, self.left_y, cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
        Right_nice= cv2.remap(self.imgR, self.right_x, self.right_y, cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)

        if save_img:
            cv2.imwrite(new_name_left_img, Left_nice)
            cv2.imwrite(new_name_right_img, Right_nice)

        if show_image:
            cv2.imshow("Left image after rectification", Left_nice)
            cv2.imshow("Right image after rectification", Right_nice)
            cv2.waitKey(0)



def main():
    images = Undistort(improved_parameters="improved_params.xml", pathL="myPath/left_img.png", pathR="myPath/right_img")
```

```
68      images.undistort_and_rectify(new_name_left_img="undistorted_left.png", new_name_right_img="undistorted_right.png")
69      images.show_image()
70
71
72  if __name__ == '__main__':
73      #main()
```

Appendix G

Source Code Arduino

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

/* This driver reads raw data from the BNO055

   Connections
   ===========
   Connect SCL to analog 5
   Connect SDA to analog 4
   Connect VDD to 3.3V DC
   Connect GROUND to common ground

   History
   =======
   2015/MAR/03  - First release (KTOWN)
*/

/* Set the delay between fresh samples */
#define BNO055_SAMPLERATE_DELAY_MS (50)
#define NED_AXIS_MAP (0x21)
#define NED_AXIS_SIGN (0x01)

/* Set the values of the configuration registers of the sensors */
#define GYRO_CONFIG0 (0x38)
#define GYRO_CONFIG1 (0x00)
#define ACCEL_CONFIG (0x0F)
#define MAG_CONFIG (0x7D)

const int trigPin = 11;           //connects to the trigger pin on the
distance sensor
const int echoPin = 12;           //connects to the echo pin on the distance
sensor

float distance = 0;               //stores the distance measured by the
distance sensor
float delta_t = 0.05;
float time_stamp = 0;
float last_time_stamp = 0;

boolean magnetometer = false;
boolean gyroscope = false;
boolean linear_acceleration = false;
boolean accelerometer = false;
boolean euler_meas = false;
boolean gravity_meas = false;
boolean acc_mag_gyro = true;
```

```
boolean display_offsets = false;

// Check I2C device address and correct line below (by default address is 0x29
or 0x28)
//                                   id, address
Adafruit_BNO055 bno = Adafruit_BNO055(-1, 0x28);

/**************************************************************************/
/*
    Arduino setup function (automatically called at startup)
*/
/**************************************************************************/
void setup(void)
{
  Serial.begin(115200);
  //Serial.println("Orientation Sensor Raw Data Test"); Serial.println("");
  //pinMode(trigPin, OUTPUT);   //the trigger pin will output pulses of
electricity
  //pinMode(echoPin, INPUT);    //the echo pin will measure the duration of
pulses coming back from the distance sensor
  //pinMode(13, OUTPUT);
  /* Initialise the sensor */
  if(!bno.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!
");
    while(1);
  }

  /* Update the sensor config registers */
  bno.setMode(0x00);
  delay(100);
  //bno.write8(Adafruit_BNO055::BNO055_UNIT_SEL_ADDR, 0x80);
  bno.setAxisRemap(0x24);
  bno.setAxisSign(0x02);
  //delay(25);
  //Serial.println(bno.read8(Adafruit_BNO055::BNO055_UNIT_SEL_ADDR));
  delay(25);
  bno.write8(Adafruit_BNO055::BNO055_PAGE_ID_ADDR, 0x01);
  delay(100);
  //bno.write8(Adafruit_BNO055::MAG_CONFIG_ADDR, MAG_CONFIG);
  bno.write8(Adafruit_BNO055::ACCEL_CONFIG_ADDR, ACCEL_CONFIG);
  bno.write8(Adafruit_BNO055::GYRO_CONFIG0_ADDR, GYRO_CONFIG0);
  bno.write8(Adafruit_BNO055::GYRO_CONFIG1_ADDR, GYRO_CONFIG1);
```

```
  adafruit_bno055_offsets_t calibrationData;
  sensor_t sensor;

  /* Set BNO055 to AMG mode (accel-mag-gyro) */
  bno.setMode(0x07);
  //bno.setMode(0x0C);

  delay(1000);

  /* Display the current temperature */
  int8_t temp = bno.getTemp();
//  Serial.print("Current Temperature: ");
//  Serial.print(temp);
//  Serial.println(" C");
//  Serial.println("");

  bno.setExtCrystalUse(false);

//  if(magnetometer)
//  {
//    Serial.println("time, Magnetometer X, Magnetometer Y, Magnetometer Z");
//  }
//
//  if(gyroscope)
//  {
//    Serial.println("time, Gyro X, Gyro Y, Gyro Z");
//  }
//
//  if(accelerometer)
//  {
//    Serial.println("time, Acceleration X, Acceleration Y, Acceleration Z");
//  }
//
//  if(linear_acceleration)
//  {
//    Serial.println("time, LinearAcc X, LinearAcc Y, LinearAcc Z, distance");
//  }
//
//  if(euler_meas)
//  {
//    Serial.println("time, Euler X, Euler Y, Euler Z");
//  }
//
//  if(gravity_meas)
//  {
//    Serial.println("time, Gravity X, Gravity Y, Gravity Z");
//  }
  //Serial.println("Calibration status values: 0=uncalibrated, 3=fully
```

```
calibrated");

  //Serial.println("time, Acceleration x, Acceleration y, acceleration z,
distance");
  //Serial.println("time, Acceleration x, Acceleration y, Acceleration z,
Magnetometer x, Magnetometer y, Magnetometer z, Euler x, Euler y, Euler z");

}

/**************************************************************************/
/*
    Arduino loop function, called once 'setup' is complete (your own code
    should go here)
*/
/**************************************************************************/
void loop(void)
{
//  time_stamp = millis();
//  delta_t = (time_stamp - last_time_stamp)/1000;
  // Possible vector values can be:
  // - VECTOR_ACCELEROMETER - m/s^2
  // - VECTOR_MAGNETOMETER  - uT
  // - VECTOR_GYROSCOPE     - degrees/s
  // - VECTOR_EULER         - degrees
  // - VECTOR_LINEARACCEL   - m/s^2
  // - VECTOR_GRAVITY       - m/s^2
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  imu::Vector<3> acc = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
  imu::Vector<3> acc_lin = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
  imu::Vector<3> gyro = bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  imu::Vector<3> magneto = bno.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
  imu::Vector<3> gravity = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY);
  double xm_off, ym_off, zm_off, xm_cal, ym_cal, zm_cal;


  xm_off = magneto.x() - 46.785;
  ym_off = magneto.y() + 98.595;
  zm_off = magneto.z() - 98.75;

  //Scale factor x:  1.0636341343
  //Scale factor y:  1.175314997967755
  //Scale factor z:  0.8271357742181541

  xm_cal = 0.643165*xm_off;
  ym_cal = 1.874837*ym_off;
  zm_cal = 1.096721*zm_off;

  //distance = getDistance();   //variable to store the distance measured by
```

the sensor

```
   if(magnetometer)
  {
    Serial.println(millis() + String(",") + (magneto.x()) + String(",") +
(magneto.y()) + String(",") + (magneto.z()) + acc.x() + String(",") + acc.y()
+ String(",") + acc.z());
  }

  if(gyroscope)
  {
    Serial.println(millis() + String(",") + gyro.x() + String(",") + gyro.y()
+ String(",") + gyro.z());
  }

  if(accelerometer)
  {
    Serial.println(millis() + String(",") + acc.x() + String(",") + acc.y() +
String(",") + acc.z());
  }

  if(linear_acceleration)
  {
    Serial.println(millis() + String(",") + acc_lin.x() + String(",") +
acc_lin.y() + String(",") + acc_lin.z() + String(",") + distance);
  }

  if(euler_meas)
  {
    Serial.println(millis() + String(",") + euler.x() + String(",") + euler.
y() + String(",") + euler.z());
  }

  if(gravity_meas)
  {
    Serial.println(millis() + String(",") + gravity.x() + String(",") +
gravity.y() + String(",") + gravity.z());
  }

  if(acc_mag_gyro)
  {
    Serial.println(delta_t + String(",") + (acc.x()) + String(",") + (acc.y())
+ String(",") + (acc.z()) + String(",") + (xm_cal) + String(",") + (ym_cal) +
String(",") + (zm_cal) + String(",") + (gyro.x()) + String(",") + (gyro.y()) +
String(",") + (gyro.z()) + String(",") + 10);
    //Serial.write(delta_t + String(",") + (acc.x()) + String(",") + (acc.y())
+ String(",") + (acc.z()) + String(",") + (xm_cal) + String(",") + (ym_cal) +
```

```
String(",") + (zm_cal) + String(",") + (gyro.x()) + String(",") + (gyro.y()) +
String(",") + (gyro.z()) + String(",") + 10);
  }

//  if(acc_mag_gyro)
//  {
//    Serial.println(millis() + String(",") + (acc.x()+0.34) + String(",") +
(acc.y()+0.46) + String(",") + (acc.z()+0.3) + String(",") + (magneto.x()+33.
25) + String(",") + (magneto.y()+2.9375) + String(",") + (magneto.z()+51.6875)
+ String(",") + (gyro.x()+0.125) + String(",") + (gyro.y()+0.125) + String(",
") + (gyro.z()+0.125) + String(",") + euler.z() + String(",") + euler.y() +
String(",") + euler.x());
//  }
  //Serial.println(millis() + String(",") + acc.x() + String(",") + acc.y() +
String(",") + acc.z() + String(",") + magneto.x() + String(",") + magneto.y()
+ String(",") + magneto.z() + String(",") + euler.x() + String(",") + euler.
y() + String(",") + euler.z());
  //Serial.println(gyro.x());
  //Linear Acceleration
  //Serial.println(millis() + String(",") + acc_lin.x() + String(",") +
acc_lin.y() + String(",") + acc_lin.z() + String(",") + distance);
  //Gyroscope
  //Serial.println(millis() + String(",") + gyro.x() + String(",") + gyro.y()
+ String(",") + gyro.z());
  //Magnetometer
  //Serial.println(millis() + String(",") + magneto.x() + String(",") +
magneto.y() + String(",") + magneto.z());
  //Gravity
  //Serial.println(millis() + String(",") + gravity.x() + String(",") +
gravity.y() + String(",") + gravity.z());

//  Serial.print("Distance: " + String(distance));     //print the distance
that was measured
//  Serial.println(" cm");      //print units after the distance

  /* Display the floating point data */
//  Serial.print("X: ");
//  Serial.print(acc.x());
//  Serial.print(" Y: ");
//  Serial.print(acc.y());
//  Serial.print(" Z: ");
//  Serial.print(acc.z());
//  Serial.print("\t\t");

  /* Display the floating point data */
//  Serial.print("X: ");
//  Serial.print(euler.x());
//  Serial.print(" Y: ");
```

```
//  Serial.print(euler.y());
//  Serial.print(" Z: ");
//  Serial.print(euler.z());
//  Serial.print("\t\t");

  /*
  // Quaternion data
  imu::Quaternion quat = bno.getQuat();
  Serial.print("qW: ");
  Serial.print(quat.w(), 4);
  Serial.print(" qX: ");
  Serial.print(quat.x(), 4);
  Serial.print(" qY: ");
  Serial.print(quat.y(), 4);
  Serial.print(" qZ: ");
  Serial.print(quat.z(), 4);
  Serial.print("\t\t");
  */

  /* Display calibration status for each sensor. */
//  uint8_t system, gyro, accel, mag = 0;
//  bno.getCalibration(&system, &gyro, &accel, &mag);
//  Serial.print("CALIBRATION: Sys=");
//  Serial.print(system, DEC);
//  Serial.print(" Gyro=");
//  Serial.print(gyro, DEC);
//  Serial.print(" Accel=");
//  Serial.print(accel, DEC);
//  Serial.print(" Mag=");
//  Serial.println(mag, DEC);
//  last_time_stamp = time_stamp;
  //Serial.flush();
  delay(BNO055_SAMPLERATE_DELAY_MS);
}

void displaySensorOffsets(const adafruit_bno055_offsets_t &calibData)
{
    Serial.print("Accelerometer: ");
    Serial.print(calibData.accel_offset_x); Serial.print(" ");
    Serial.print(calibData.accel_offset_y); Serial.print(" ");
    Serial.print(calibData.accel_offset_z); Serial.print(" ");

    Serial.print("\nGyro: ");
    Serial.print(calibData.gyro_offset_x); Serial.print(" ");
    Serial.print(calibData.gyro_offset_y); Serial.print(" ");
    Serial.print(calibData.gyro_offset_z); Serial.print(" ");

    Serial.print("\nMag: ");
```

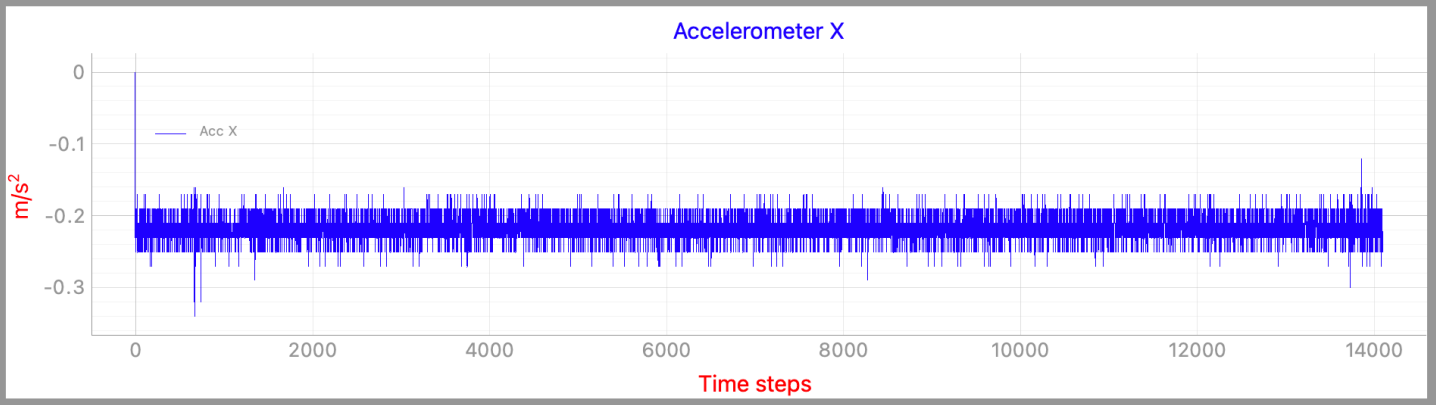
```
    Serial.print(calibData.mag_offset_x); Serial.print(" ");
    Serial.print(calibData.mag_offset_y); Serial.print(" ");
    Serial.print(calibData.mag_offset_z); Serial.print(" ");

    Serial.print("\nAccel Radius: ");
    Serial.print(calibData.accel_radius);

    Serial.print("\nMag Radius: ");
    Serial.print(calibData.mag_radius);
}

//RETURNS THE DISTANCE MEASURED BY THE HC-SR04 DISTANCE SENSOR
float getDistance()
{
  float echoTime;                   //variable to store the time it takes for
a ping to bounce off an object
  float calculatedDistance;         //variable to store the distance
calculated from the echo time

  //send out an ultrasonic pulse that's 10ms long
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  echoTime = pulseIn(echoPin, HIGH);      //use the pulsein command to see how
long it takes for the
                                          //pulse to bounce back to the sensor

  calculatedDistance = echoTime * 2.54 / 148.0;  //calculate the distance of
the object that reflected the pulse (half the bounce time multiplied by the
speed of sound)

  return calculatedDistance;              //send back the distance that was
calculated
}
```

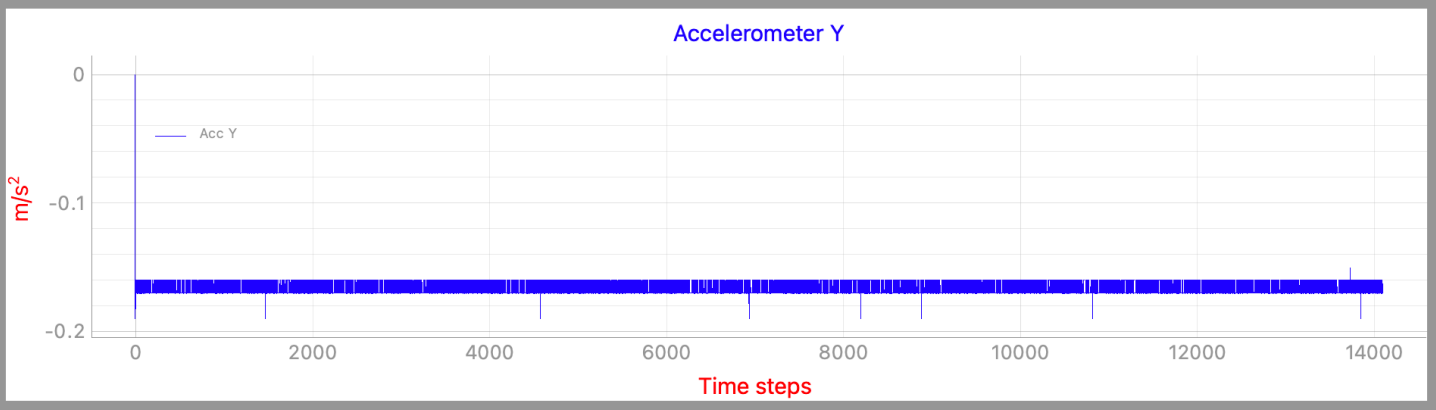
Appendix H

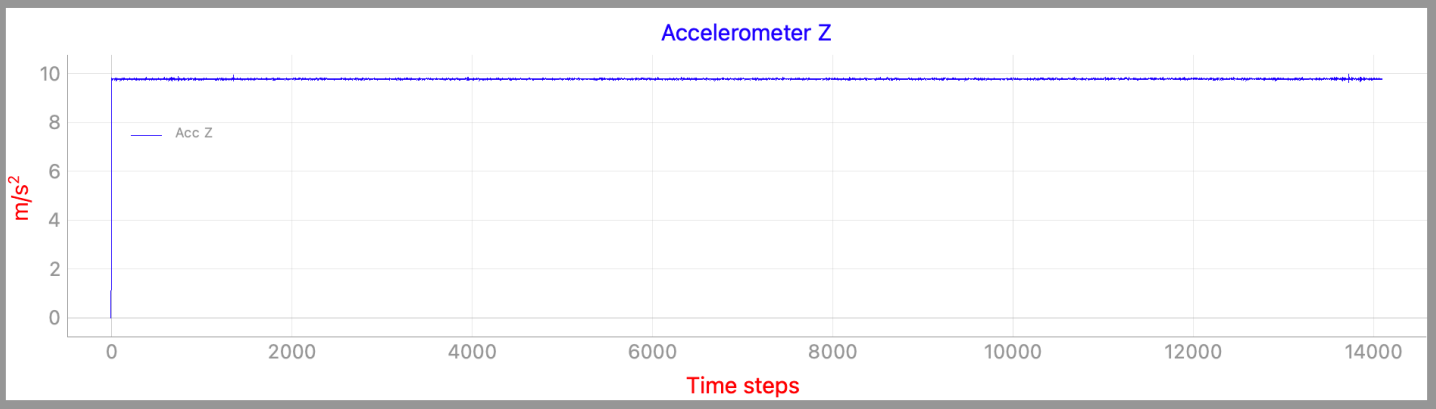
Plots of raw sensor data

Raw sensor values from Stationary test

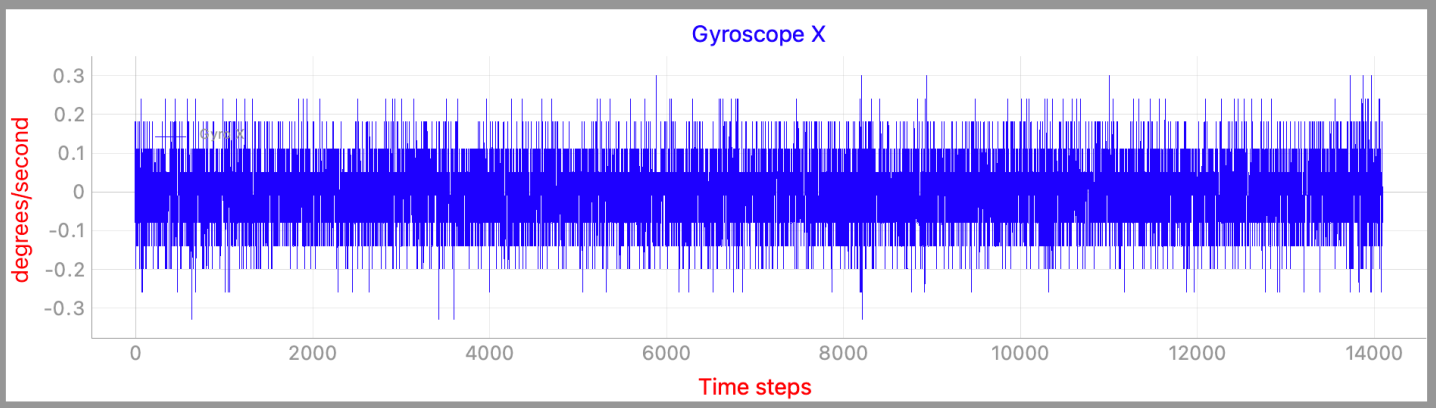
Raw Accelerometer Values

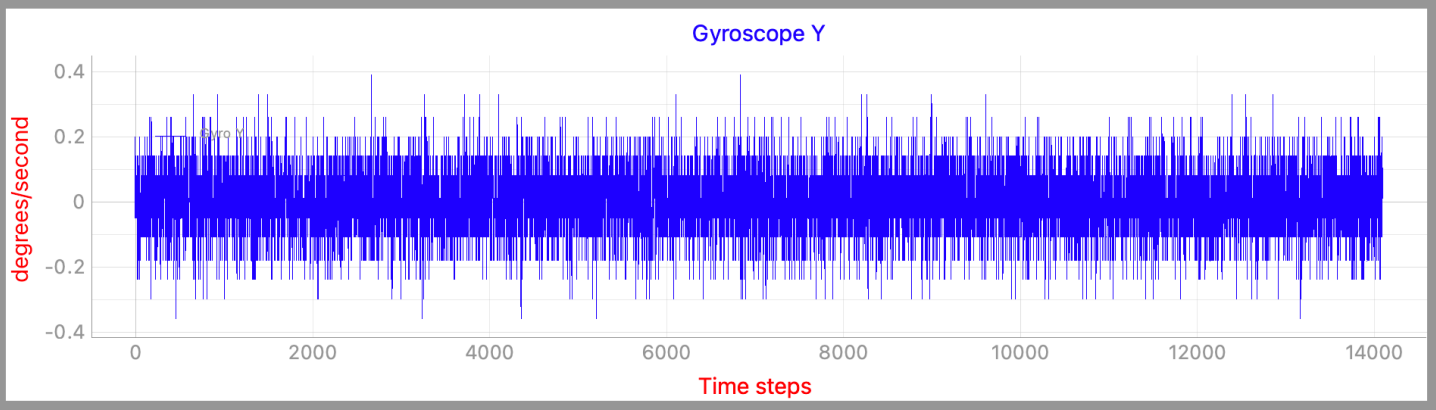


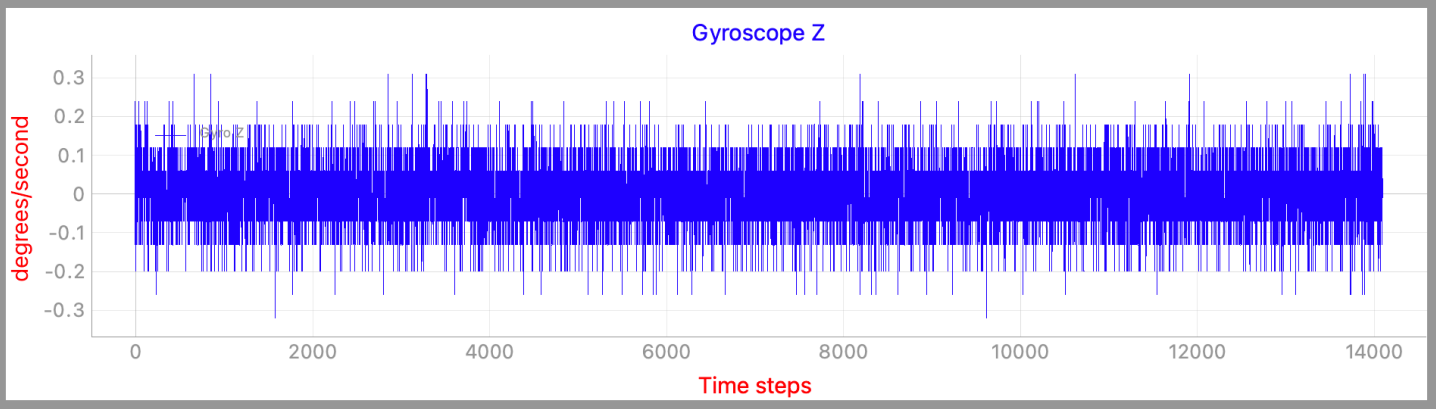




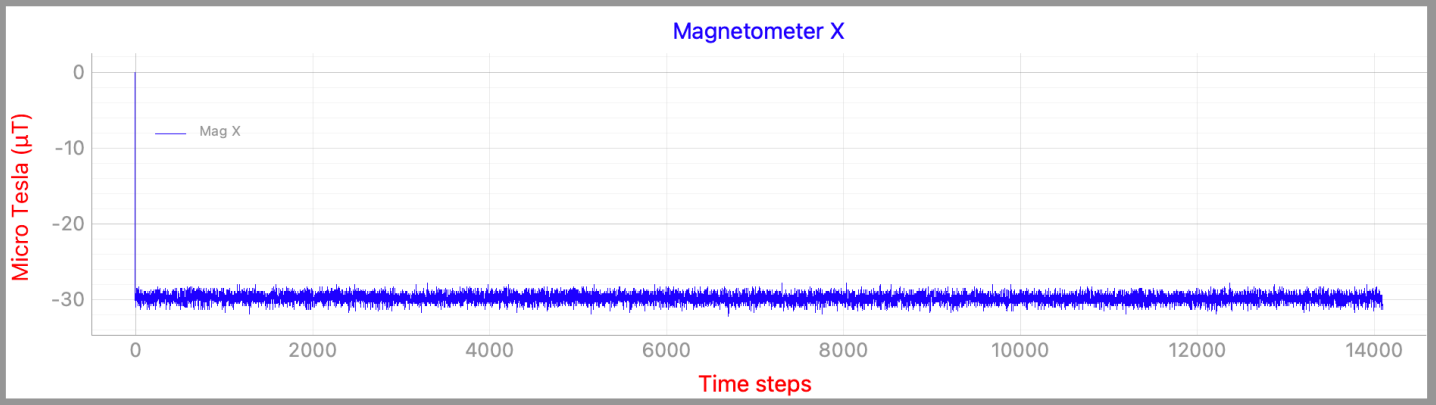
Raw Gyroscope Values

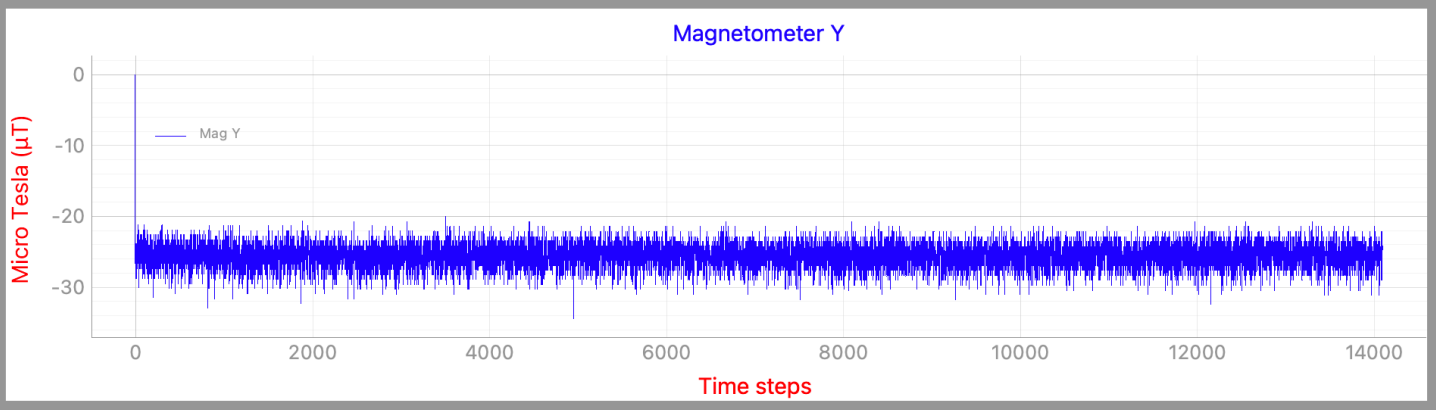


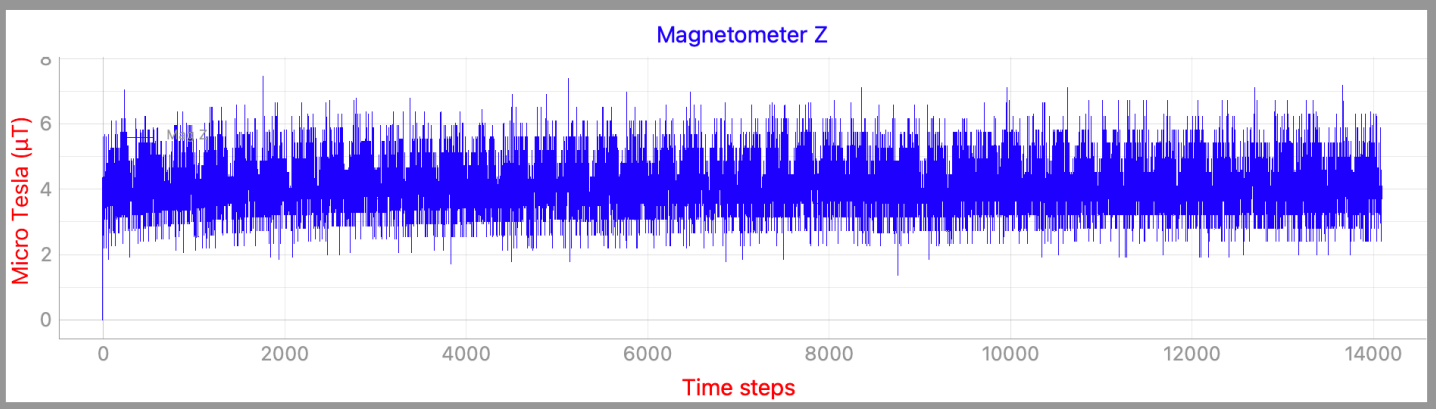




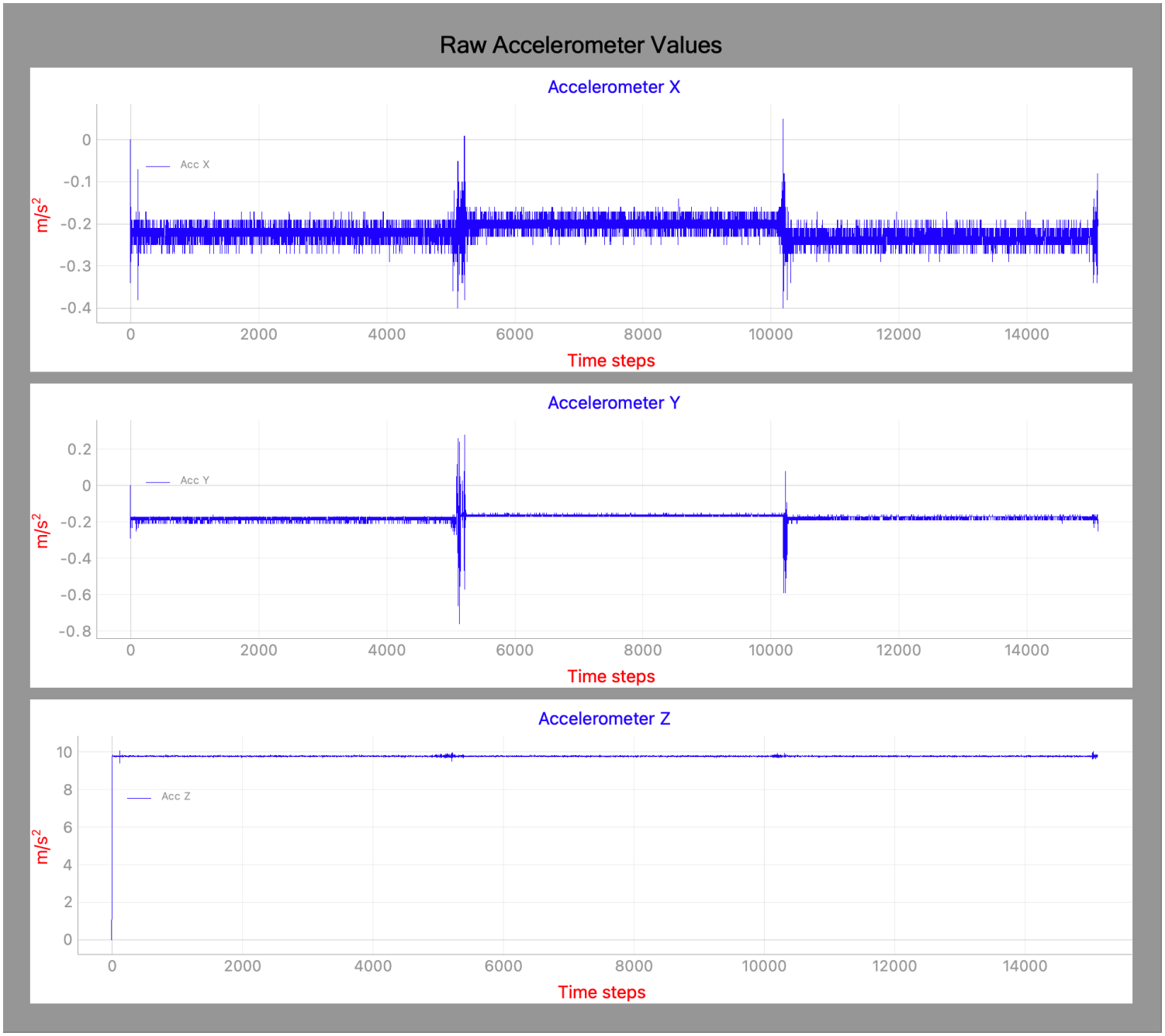
Raw Magnetometer Values



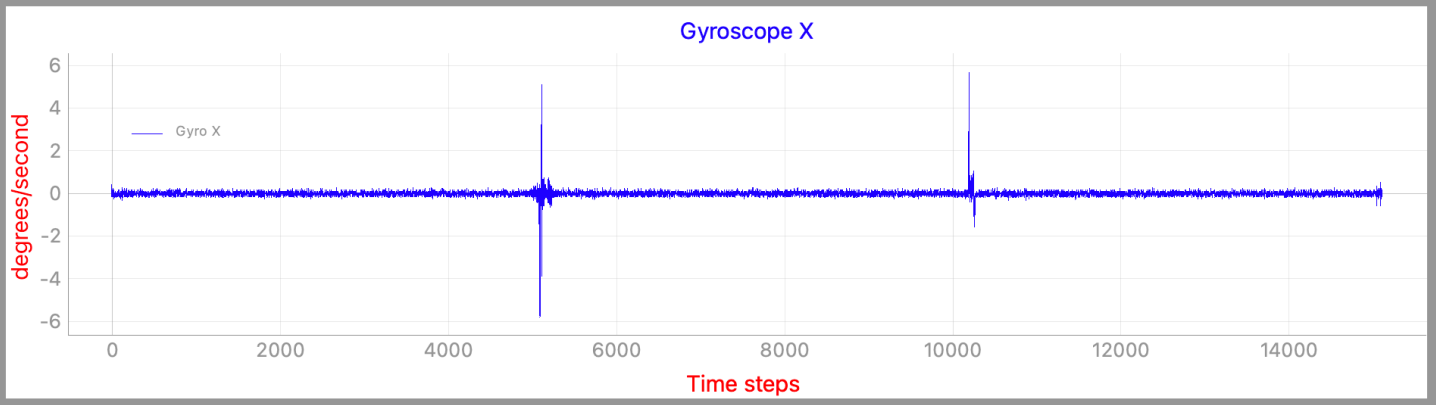


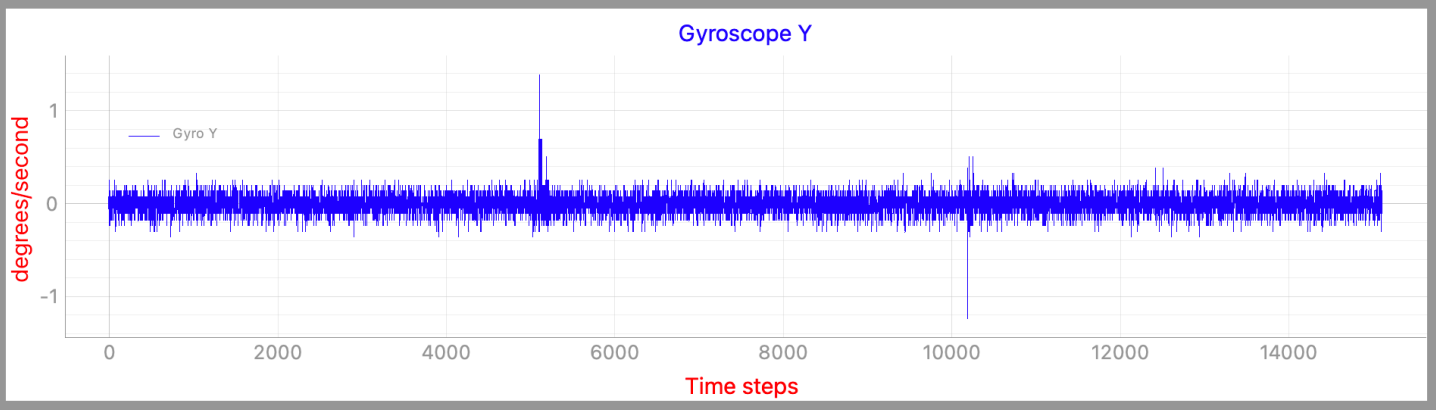


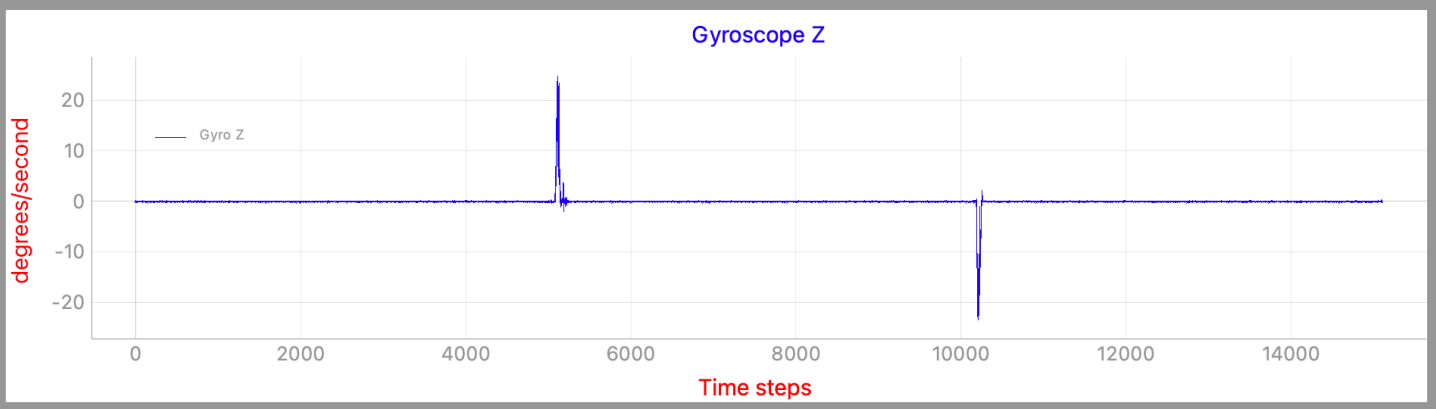
Raw sensor values from Yaw test


Raw Accelerometer Values
Accelerometer X
Acc X
m/s2
0
-0.1
-0.2
-0.3
-0.4
0
2000
4000
6000
8000
10000
12000
14000
Time steps
Accelerometer Y
Acc Y
m/s2
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2000
4000
6000
8000
10000
12000
14000
Time steps
Accelerometer Z
Acc Z
m/s2
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
12000
14000
Time steps

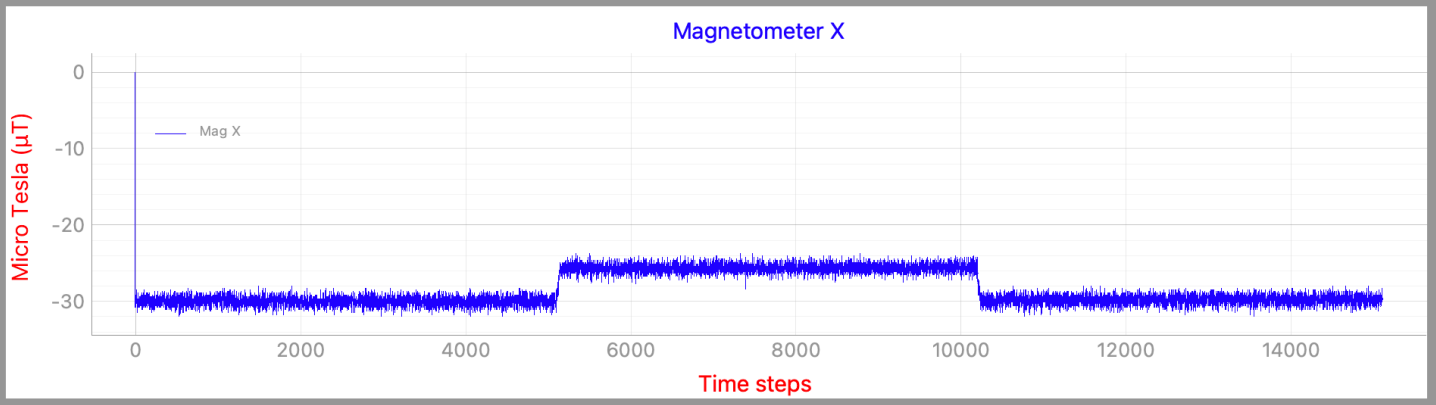
Raw Gyroscope Values

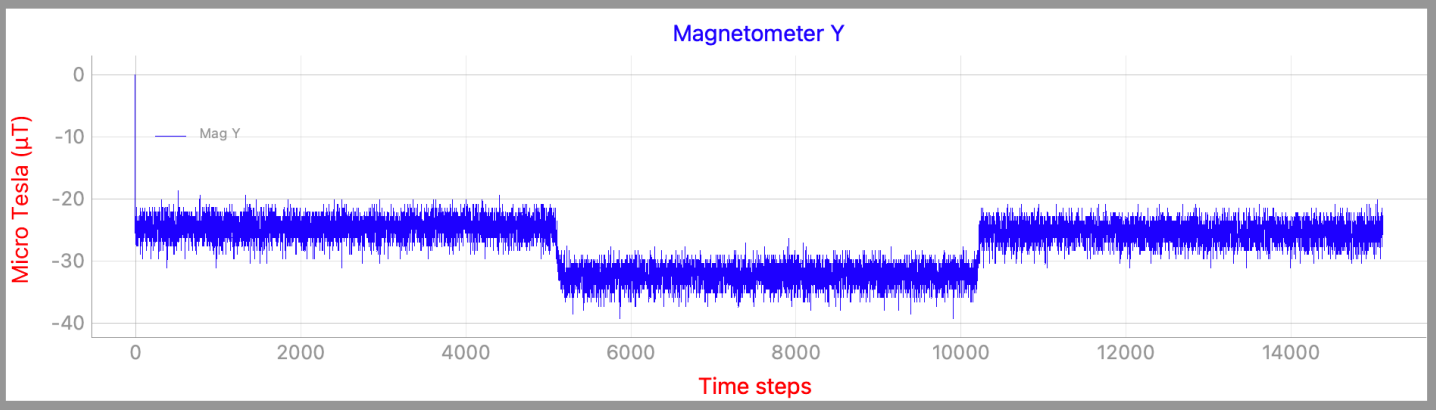


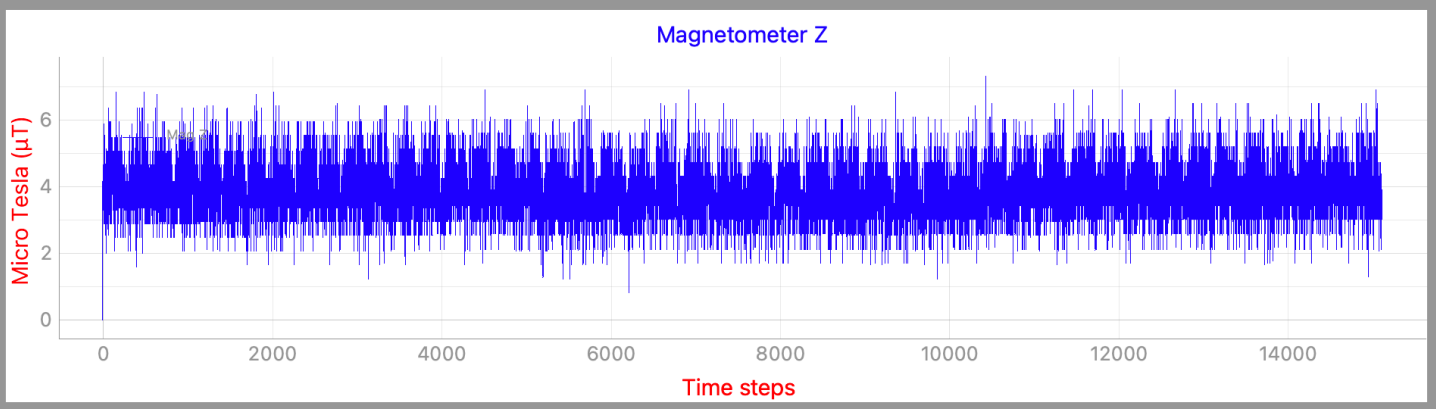




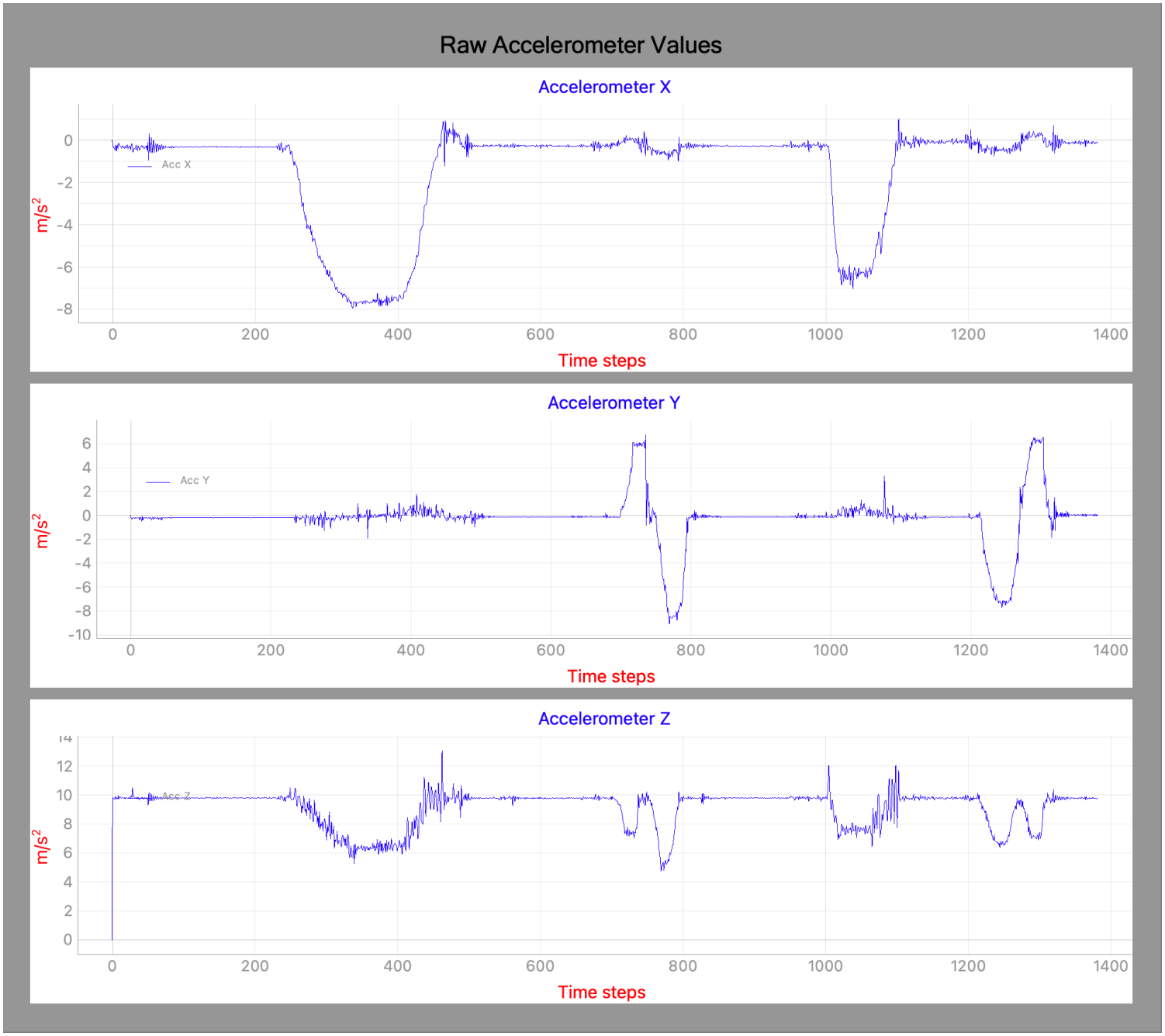
Raw Magnetometer Values



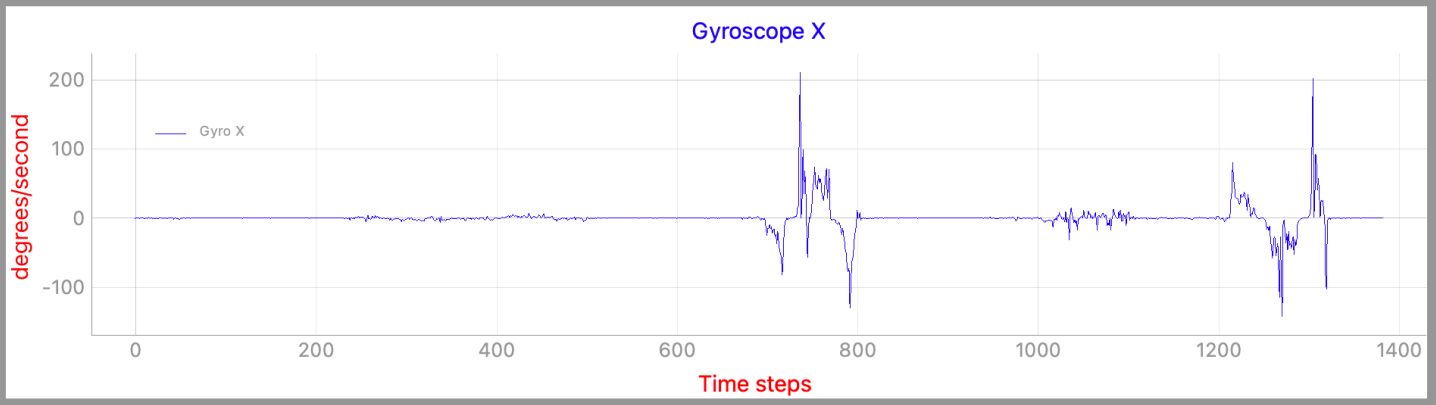


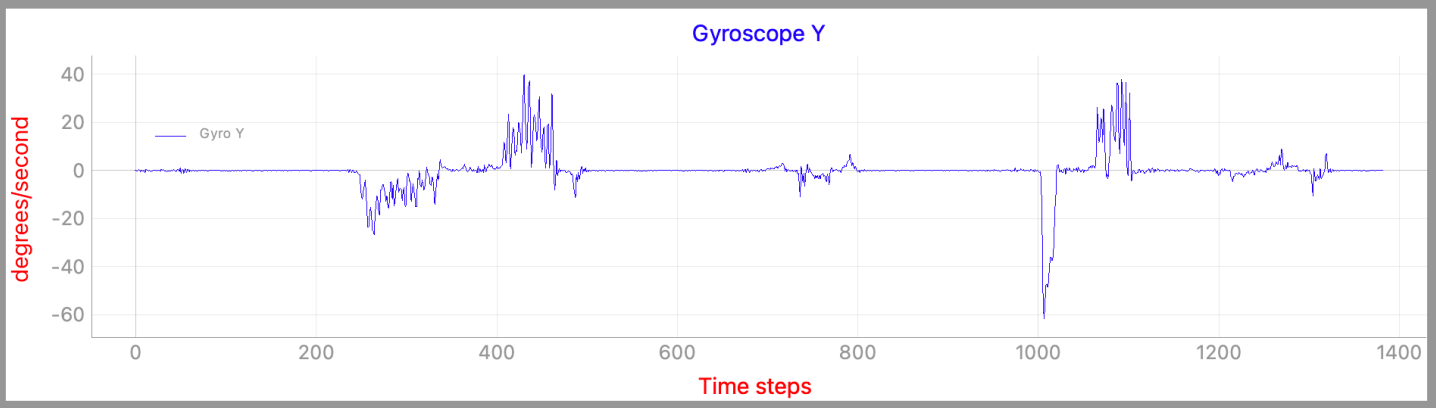


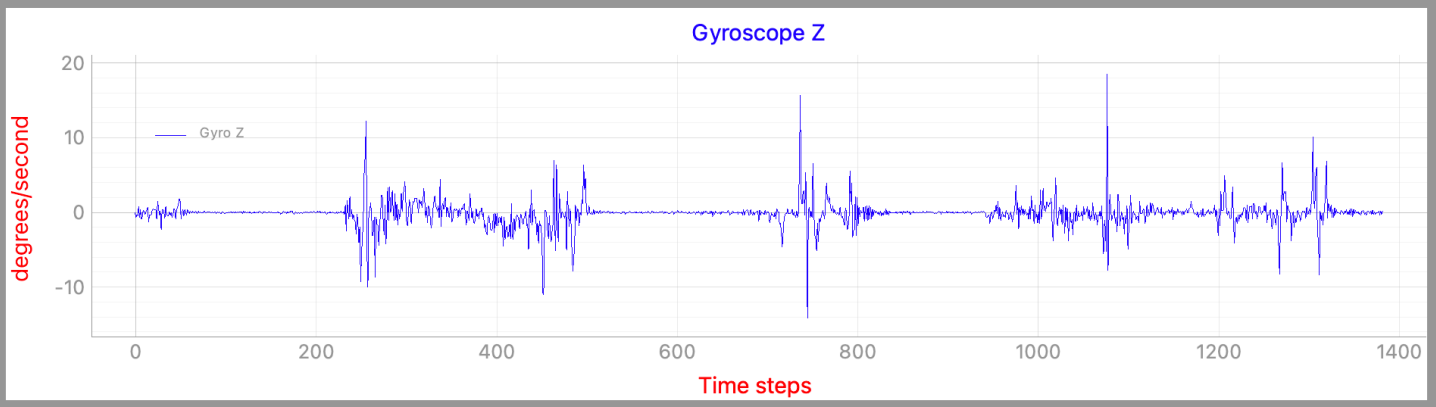
Raw sensor values from Pitch/Roll test


Raw Accelerometer Values
Accelerometer X
Acc X
m/s2
0
-2
-4
-6
-8
0
200
400
600
800
1000
1200
1400
Time steps
Accelerometer Y
Acc Y
m/s2
6
4
2
0
-2
-4
-6
-8
-10
0
200
400
600
800
1000
1200
1400
Time steps
Accelerometer Z
Acc Z
m/s2
12
10
8
6
4
2
0
0
200
400
600
800
1000
1200
1400
Time steps

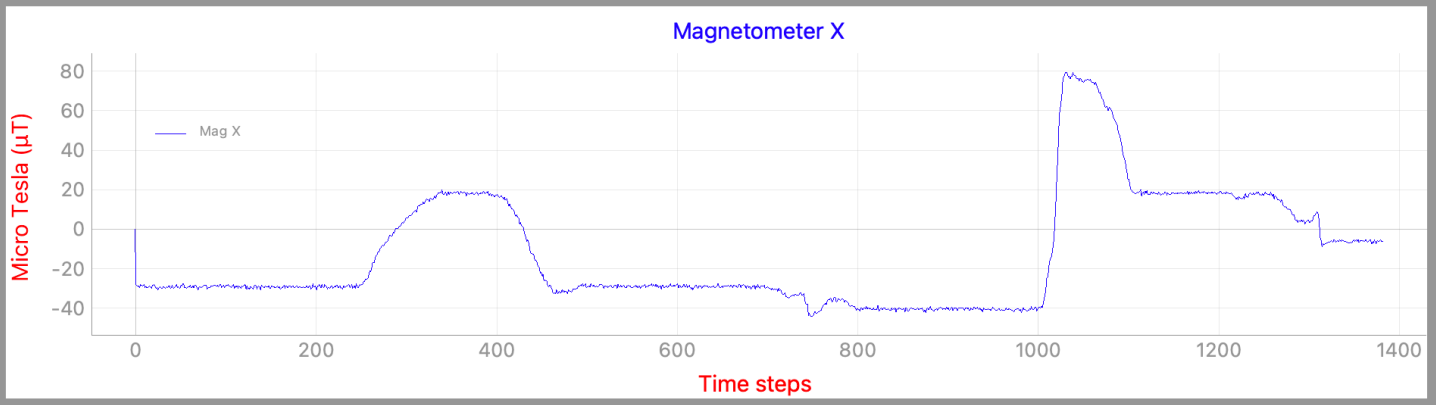
Raw Gyroscope Values

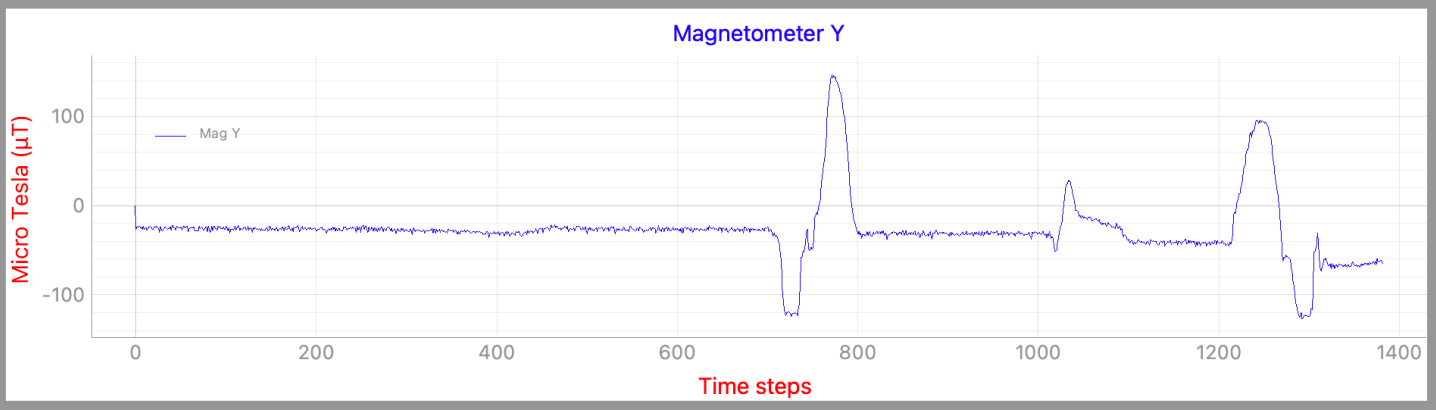


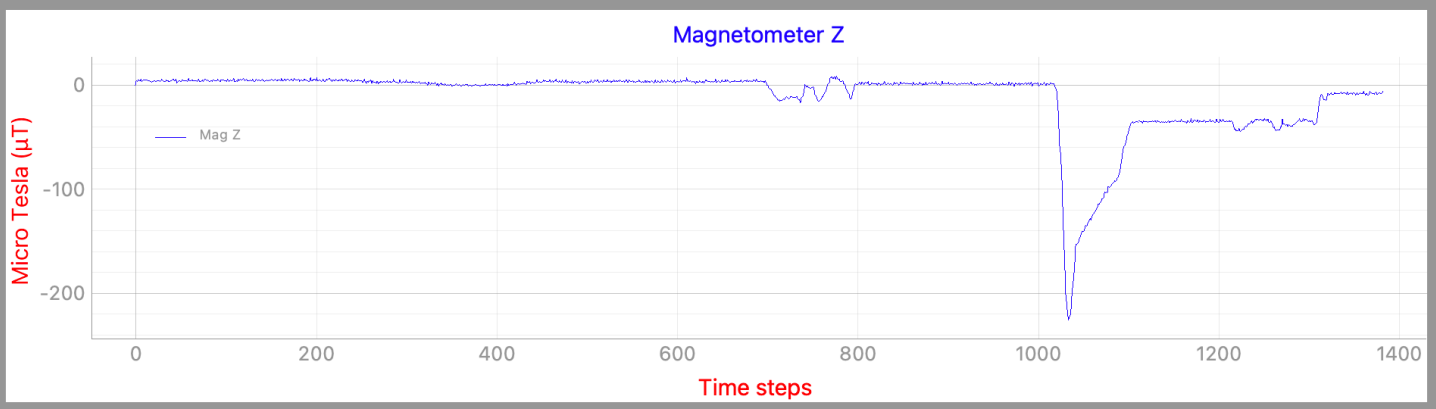


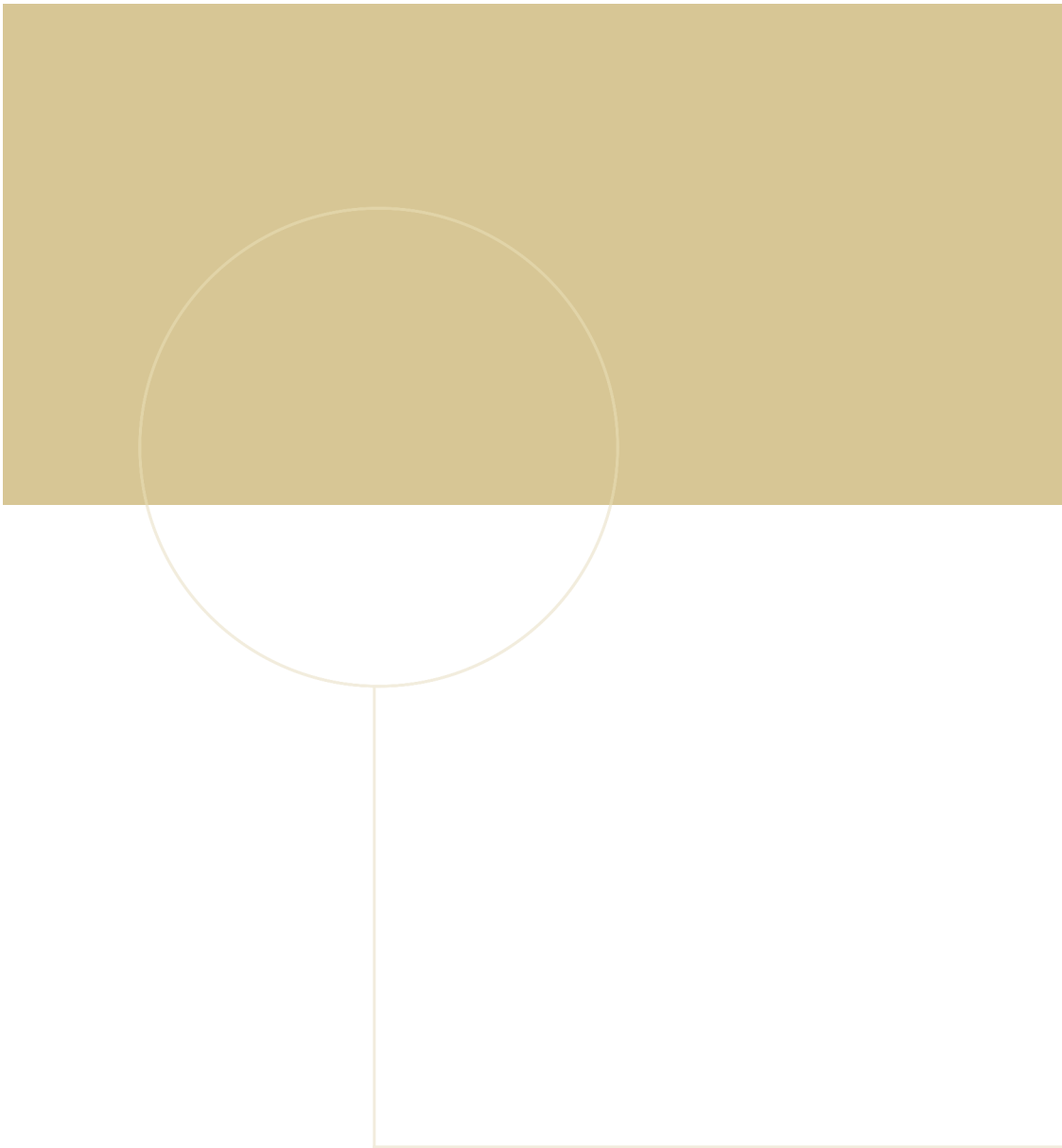


Raw Magnetometer Values









NTNU

Kunnskap for en bedre verden