NTNU—Norwegian University of Science and Technology

Department of ICT and Natural Sciences

Bachelor Thesis

Relative Motion Tracking of Vessels using Multi-Camera Handover and Aruco Markers

Kai Hagseth
Even Drugli
Vegard Fjørtoft
Ole Kristian Sande

Supervisors
Ottar L. Osen
Robin T. Bye

May 19, 2019

Summary

The towing tank at NTNU campus Ålesund is used by students in ship-design and other disciplines to test designed ships, ROVs and other vessels.

When doing research on ship designs, small scale testing with models is used to verify theoretical results. For analysis, it's very useful to get exact results of the movement of the ship. As of today, a mechanical solution is installed above the tank. A trolley runs along the length of the tank. An arm from the trolley runs down in the middle of the tank to the ship model. This system provides information about the ship's movement in 4 degrees of freedom as movement to the sides and rotation in yaw is restricted by the arm.

NTNU Department of Ocean Operations and Civil Engineering (IHB) wants to investigate the possibilities of developing a new system to track the movement and rotation of their ship-models.

In this paper, we present a computer vision solution using multiple cameras and ArUco markers for tracking of vessels in 6 degrees of freedom. The system has adequate accuracy in the initial camera views but suffers from a compounding error when calculating the absolute position of the vessel after switching camera frames. Despite this, the data the system acquires should still be useful to see how a vessel behaves in the tank.

The developed application can be refitted for many purposes. The system is designed to easily be taken down and re-deployed somewhere else since it automatically calibrates the camera positions with regard to the object for each individual run. As long as one can provide camera coverage of the area where the object should be tracked, and there are possibilities to attach a marker on the object, one should be able to get a live pose estimation of the tracked object.

We have implemented an user-friendly GUI with accompanying user manual. The system should be usable by anyone and requires no prior knowledge of vision systems.

Preface

We would like to thank everyone who has helped us with this project, especially:

- Supervisors Ottar L. Osen and Robin T. Bye for guidance throughout the project.
- Karl Henning Halse for providing us with an interesting bachelor thesis at short notice.
- Anders Sætersmoen and Øivind Hanken with help to order parts.
- André Tranvåg for organizing access to the water tank and helping with creating parts for the project.
- Michal Malisz and Oskar Sunde for organizing and help with the water tank systems.
- Arne Styve for sharing his thoughts about programming problems.

If you are a developer who wishes to do further development to our system you are welcome to contact us with any questions you may have regarding our code or our implementations. Citation with source code on Github and E-mail: Drugli et al. (2019)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

NTNU Department of Ocean Operations and Civil Engineering (IHB) wants to investigate the possibilities of developing a new system to track the movement and rotation of their ship-models. The models are tested in a towing tank at campus.

The system should be able to find the right position and rotation relative to a set reference point. The pose data should be collected so it can later be used to analyze and improve the vessels.

Neither IHB nor our supervisors had any specific accuracy requirement. Given the dimensions of models used, we considered a accuracy of $\pm$ 10 mm for position and $\pm$ 2 degrees rotation relative to the starting-pose to be an achievable goal which would result in a useful product.

An important requirement was to have a robust system that is usable for students and teachers. Therefore the system should have a good GUI and be usable on the PC that already controls the other systems in the towing tank.

The department of ICT and natural sciences (IIR) are setting up Computer Vision as a area of interest for the years to come. It's therefore desirable from IIR to focus our work around using computer vision.

Please note this report is not a manual for the system. A user manual is found in appendix A.

1.1 Background

The towing tank at campus is used by students in ship-design and other disciplines to test designed ships, ROVs and other vessels.

When doing research on ship designs, small scale testing with models is used to verify theoretical results. For analysis it's very useful to get exact results of the movement of the ship. As of today, a mechanical solution is installed above the tank. A trolley runs along the length of the tank. An arm from the trolley runs down in the middle of the tank to the ship model, see figure 1.1. The vessel model is fastened to the arm. In total this system

can provide information in 4 degrees of freedom as movement to the sides and rotation in yaw is restricted by the arm.

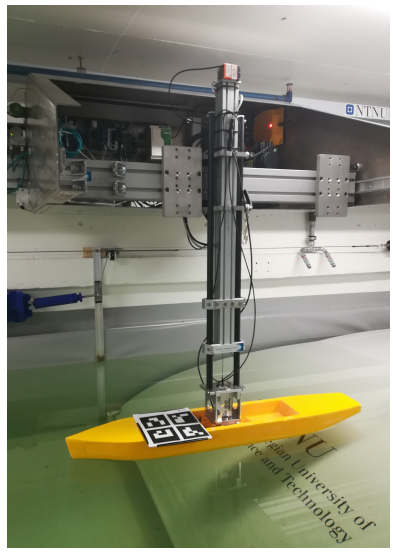


Figure 1.1: Tank trolley and arm connected to model.

This system is according to the contractor not very user-friendly and tends to give noisy measurements.

1.2 Project Objectives

The main objective of the project is to develop a system that can detect and track vessels. It should improve the user experience and the accuracy should be better than the older solution.

A solution for wave analysis has also been requested. This is considered a secondary objective.

1.3 Project Requirements

- No definite accuracy requirement was specified from our contractor or from our supervisors, but we considered ± 10mm and $\pm 2^\circ$ relative to the starting pose to be an achievable goal.
- Create a robust system that is usable by students and teachers.
- Get the results of the pose estimation and video feed from the cameras in real-time.
- Give the same amount or more data than the current system.

Chapter 2

Literature Review

In this chapter, we will go over some of the literature that has been used in the research phase and also later in the project to make comparisons. It is important to note that we have not done an intensive literature search through databases to find all relevant research related to our task, reviewed, critiqued and compared it like you would in a proper literature review. We also make no claim that the texts listed and used are the most comprehensive or most accurate within the field. We have instead listed the textbooks and articles that have been very helpful for us to gain understanding and inspiration and that we feel would be helpful to others undertaking a project like this.

The notations and definitions used for spatial descriptions and transformation are taken from chapter 1 & 2 of "Introduction to Robotics: Mechanics and Control" by Craig (2005). While this is a textbook covering robotics, the first two chapters, and their examples are easy to understand while comprehensive enough to cover everything we needed regarding these topics in the project.

For image processing and computer vision we have primarily used the textbooks "Multiple View Geometry in Computer Vision" by Hartley and Zisserman (2004) and "Learning OpenCV" by Bradski and Kaehler (2013) to educate ourselves on the topics while supplementing with information from the online documentation of OpenCV (Open Source Computer Vision Library) and lectures by Hoff (2014). Together these gave us a comprehensive view of core concepts and terminology. The ArUco library for OpenCV is well documented by its creators in articles Garrido-Jurado et al. (2014), Garrido-Jurado et al. (2016) and Romero-Ramirez et al. (2018).

In this paper, we have used Pentenrieder et al. (2019) and López-Cerón and Cañas (2016) for comparison of our accuracy results. Both of these articles use marker-based position tracking but not ArUco markers.

In chapter 3 pieces of information from sources mentioned in this chapter and numerous others will be put into more context.

Chapter 3

Theory

In this chapter, we will go over some of the concepts, definitions and information we found during research that we feel are important. Both for the reader to have knowledge of to better understand the information presented in later chapters and for us to refer to when explaining our decision making and implementations.

3.1 Description and Transformation of Positions and Orientations

There are several ways to describe where a point is and how it is oriented in three-dimensional space. Some methods give clarity to the end user, while others offer arithmetical advantages when performing transformations. The following methods are chosen with both readability and usability in mind. The definitions used in this paper can be found in Craig (2005).

3.1.1 Position

After we have defined a coordinate system we can describe any point in 3D space using a $3x1$ position vector. The position vector will have a preceding superscript to indicate what system it is defined in. For example, a position P in system A will be written as ${}^{A}P$. When working with transformation between systems, a vector indicating position will some times be called a translation vector as it represents the difference in XYZ coordinates between the two systems.

3.1.2 Orientation

To describe orientation we attach a coordinate system to a point and describe this coordinate system relative to a reference system. A system A relative to system B will be described in a $3x3$ rotation matrix written as ${}^{B}_{A}R$. The orientation of system B relative to

system A can be found by taking the inverse of ${}^{B}_{A}R$. The inverse of a rotation matrix is the rotation matrix transposed so ${}^{A}_{B}R = {}^{B}_{A}R^{T}$

A detailed explanation of how you compute the rotation matrix and its inverse can be found in chapter 2.2 of Craig (2005).

3.1.3 Transformation Between Systems

Transforming from multiple relative coordinate systems into a single absolute coordinate system has been important in order to keep a common reference in our data.

If you have a position and/or orientation in system A and you want to represent it in terms of system B you will have to remap the values.

Translation

When system A and system B have the same orientation the only difference between the systems is a translation and you can solve the mapping with pure vector addition. If ${}^{B}P_{Aorg}$ is the origin of system A represented in system B then ${}^{B}P = {}^{A}P + {}^{B}P_{Aorg}$

Rotation

When a point in system A and system B have the same position but different orientation you can show ${}^{B}P$ as ${}^{A}P$ multiplied by the rotation matrix ${}^{B}_{A}R$ (The orientation of system A relative to system B). This gives us ${}^{B}P = {}^{B}_{A}R\,{}^{A}P$

Homogeneous Transformation

Combining the two previous solutions we get the general solution ${}^{B}P = {}^{B}_{A}R\,{}^{A}P + {}^{B}P_{Aorg}$ we can rewrite this as a homogeneous transformation on the form

$$\begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{B}_{A}R & {}^{B}P_{Aorg} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & 1 \end{bmatrix} \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} \tag{3.1}$$

This lets us do the mapping in a single matrix operation. We call the combined rotation and translation of system A with regard to B the transformation ${}^{B}_{A}T$

Compound Transformations

If you know the transformation matrices ${}^{A}_{B}T$ and ${}^{B}_{C}T$ the transformation of system C with regard to system A can be shown as ${}^{A}_{C}T = {}^{A}_{B}T\,{}^{B}_{C}T$

A more detailed explanation of transformation and mapping can be found in chapter 2.3 of Craig (2005)

3.1.4 Pose (Z-Y-X Euler angles)

While rotational matrices are useful and have good clarity during calculations, the most common way to describe the orientation of a vessel in layman's terms is to use Euler angles, more commonly known as roll, pitch and yaw. Instead of using the 3x3 rotation

matrix to show orientation we can present it by three consecutive rotations around the principal axes attached to the moving object.

The convention we use is Z-Y-X Euler angles, also known as Z-Y-X Tait-Bryan angles. The angles of rotation around these axes are commonly called ψ (Yaw) for the rotation around Z, θ (Pitch) around Y and ϕ (Roll) around X. We can write the set of rotations as:

$$R_\psi = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} R_\theta = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{3.2}$$

$$R_\psi R_\theta R_\phi = \begin{bmatrix} cos\theta cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ cos\theta sin\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\psi sin\theta sin\phi - sin\psi cos\phi \\ -sin\theta & sin\psi cos\theta & cos\psi cos\theta \end{bmatrix} \tag{3.3}$$

The general representation of a rotation matrix is:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.4}$$

Using this we can get the Euler angle representations from any rotation matrix where $\theta \neq \pm 90$:

$$\begin{aligned} &r_{32} = sin\psi cos\theta, \;\; r_{33} = cos\psi cos\theta \rightarrow \tfrac{r_{32}}{r_{33}} = \tfrac{sin\psi}{cos\psi} = tan\psi \\ &\psi = arctan2(r_{32}, r_{33}) \end{aligned} \tag{3.5}$$

$$\begin{aligned} &r_{21} = cos\theta sin\phi, \;\; r_{11} = cos\theta cos\phi \rightarrow \tfrac{r_{21}}{r_{11}} = \tfrac{sin\phi}{cos\phi} = tan\phi \\ &\phi = arctan2(r_{21}, r_{11}) \end{aligned} \tag{3.6}$$

$$\begin{aligned} &r_{32} = sin\psi cos\theta, \;\; r_{33} = cos\psi cos\theta, \;\; r_{31} = -sin\theta \\ &r_{32}^2 = sin\psi^2 cos\theta^2, \;\; r_{33}^2 = cos\psi^2 cos\theta^2 \\ &r_{32}^2 + r_{33}^2 = (sin\psi^2 + cos\psi^2)cos\theta^2 = cos\theta^2 \rightarrow cos\theta = \sqrt{r_{32}^2 + r_{33}^2} \\ &\tfrac{-r_{31}}{\sqrt{r_{32}^2 + r_{33}^2}} = \tfrac{sin\theta}{cos\theta} = tan\theta \rightarrow \theta = arctan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \end{aligned} \tag{3.7}$$

These equations can not be used if θ is ± 90 as $cos\theta = 0$. When this happens we will have to calculate the rotations differently as seen in Slabaugh (1999). The trigonometric identities used are not listed in the paper but can be found in Adams and Essex (2014).

For $\theta = \frac{\pi}{2}$:

$$\begin{aligned} &R_{12} = sin\psi cos\phi - cos\psi sin\phi = sin(\psi - \phi) \\ &R_{13} = cos\psi cos\phi + sin\psi sin\phi = cos(\psi - \phi) \\ &R_{22} = sin\psi sin\phi + cos\psi cos\phi = cos(\psi - \phi) = R_{13} \\ &R_{23} = cos\psi sin\phi - sin\psi cos\phi = -sin(\psi - \phi) = -R_{12} \\ &(\psi - \phi) = atan2(R_{12}, R_{13}) \\ &\psi = \phi - atan2(R_{12}, R_{13}) \end{aligned} \tag{3.8}$$

For $\theta = -\frac{\pi}{2}$:

$$\begin{aligned} R_{12} &= -sin\psi cos\phi - cos\psi sin\phi = -sin(\psi + \phi) \\ R_{13} &= -cos\psi cos\phi + sin\psi sin\phi = -cos(\psi + \phi) \\ R_{22} &= -sin\psi sin\phi + cos\psi cos\phi = cos(\psi + \phi) = -R_{13} \\ R_{23} &= -cos\psi sin\phi - sin\psi cos\phi = -sin(\psi + \phi) = R_{12} \\ (\psi + \phi) &= atan2(-R_{12}, -R_{13}) \\ \psi &= -\phi - atan2(-R_{12}, -R_{13}) \end{aligned} \tag{3.9}$$

We can see that for both cases roll ϕ and yaw ψ are linked and we have lost 1 DOF. This is called Gimbal lock or singularity and we can see that there are infinite solutions for ϕ and ψ. For us, it is convenient to pick one solution so we set $\phi = 0$ and solve for ψ.

3.2 Camera Properties

The definitions used in this paper for describing the camera properties mathematically are the same as you will find used in most papers and places online. One text using these definitions in the context of single and multiple view geometry is Hartley and Zisserman (2004). By using a mathematical camera model, in this case, the pinhole model, we are able to calculate where a point in three-dimensional space will get projected to a two-dimensional image frame.

3.2.1 Extrinsic Camera Matrix

The camera's extrinsic matrix describes the location and orientation between the camera and the world. It is used in transformations from 3D world coordinates to 3D camera coordinates. The matrix is often on the form

$$\begin{bmatrix} {}^{C}_{W}R_{3x3} & {}^{C}P_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \tag{3.10}$$

Where ${}^{C}_{W}R$ is a $3x3$ rotation matrix and ${}^{C}P$ is a $3x1$ translation vector indicating the position of the world origin expressed in the camera coordinate system where the camera origin is in the camera focal point.

To find the camera pose relative to the world coordinate system you take the inverse of the extrinsic matrix. You can simplify this by transposing the rotation and subtracting the rotated translation. This means ${}^{W}_{C}R = {}^{C}_{W}R^{T}$ and $C = -{}^{W}_{C}R{}^{C}P$ where C is the camera center position in world coordinates

3.2.2 Intrinsic Camera Matrix

The intrinsic matrix describes the characteristics of a camera and is a perspective transformation of 3D camera coordinates to 2D homogeneous image coordinates. The intrinsic

camera matrix is on the form

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Focal Length f_x, f_y

The focal length is the distance between the focal point of the camera (where all the light rays converge) and the film/sensor. For a pinhole camera, the focal length is the distance from the pinhole to the image plane. For the intrinsic camera matrix used to suit our needs, the focal length is measured in millimetres but pixel units are also used.

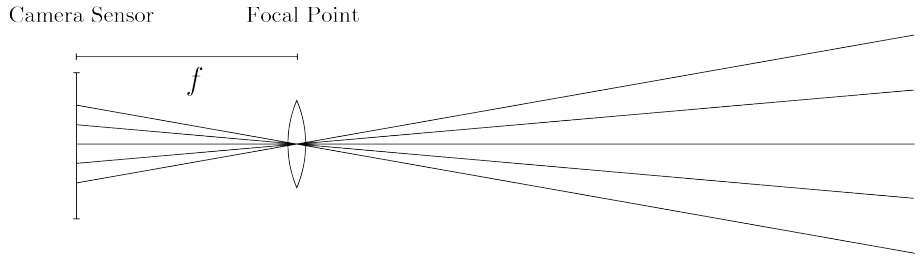


Figure 3.1: Illustration of how focal length, f, is defined.

For a true pinhole camera model f_x and f_y are the same but from Simek (2013) the reasons they can be different are:

- Flaws in digital camera sensor
- The image used to calibrate has been non-uniformly scaled in post-processing
- The camera's lens introduces unintentional distortion
- The camera uses an anamorphic format, where the lens compresses a widescreen scene into a standard-sized sensor.
- Errors in camera calibration.

In all these instances the image has non-square pixels. Some texts use a single focal length and aspect ratio to describe the deviation from a perfectly square pixel.

Principal Point Offset x_0, y_0

The point where a line perpendicular to the image plane passes through the center of the lens and intersects the image plane/sensor/film. The coordinates x_0 and y_0 are relative to the image plane origin (for us this is in the top left corner of the image).

Axis skew s

The axis skew is the shear distortion of the image. In most cases, this will be zero.

3.2.3 Perspective Transform

Perspective transform is the method used to map homogeneous world coordinates in 3D space to homogeneous 2D image coordinates.

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} {}^{C}_{W}R_{3x3} & {}^{C}P_{3x1} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.12}$$

Where s is a scaling factor. The product of the camera intrinsic matrix and the extrinsic matrix is some times called the camera projection matrix.

3.3 Camera Calibration

The purpose of the camera calibration functions in OpenCV is to determine the camera distortion coefficients $(k_1\ k_2\ p_1\ p_2\ k_3)$ and intrinsic camera matrix K. In Bradski and Kaehler (2013) the entire camera calibration process is explained in detail in chapter 11 with example code using OpenCV. From there we also see that the camera distortion coefficients are used to describe the difference between the distorted and ideal (corrected) image coordinates. For radial distortion:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.13}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.14}$$

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2} \tag{3.15}$$

where x_c and y_c is the coordinates of the distortion center. For tangential distortion:

$$x_{corrected} = x + (2p_1 y + P_2(r^2 + 2x^2)) \tag{3.16}$$

$$y_{corrected} = y + (p_1(r^2 + 2y^2) + 2p_2 x) \tag{3.17}$$

Also according to Bradski and Kaehler (2013) the other forms of distortion have less of an effect than the radial and tangential distortion and this is why OpenCV does not account for them.

Using a chessboard as our calibration object at least 10 images using a 7×8 or larger board is recommended for high quality calibration results (page 388). This is only if you move the chessboard enough between images to get a "rich" view.

3.4 Image analysis

3.4.1 HSV

HSV is an alternative representation of the RGB color model as seen in figure 3.2. Color segmentation for individual colors can easily be done by creating a mask with upper and lower bounds for the three parameters hue, saturation and value.

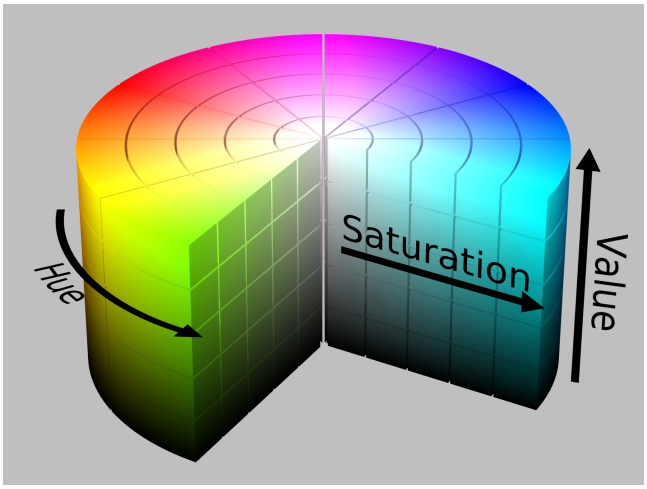


Figure 3.2: HSV cylinder color chart from Wikimedia (2010)

3.4.2 Contours

A contour is defined as an outline representing or bounding the shape or form of something. Contours in image processing are explained in detail in chapter 8 of Bradski and Kaehler (2013). In this paper we use the OpenCV function **findContours()** that implements the algorithm for contour detection found in Suzuki and Abe (1985).

3.4.3 Shape Factor

Shape factors are used in image processing to describe the shape of an object regardless of its size. For a circle the shape factor is described in equation 3.18 found in Friel (2000) Where A is the area of the blob created by the contour and P is the arc length/circumference of the contour. A perfect circle has a shape factor $f_{circle} = 1$ and any other shape has a factor $f_{circle} > 1$.

$$f_{circle} = \frac{4\pi A}{P^2} \tag{3.18}$$

3.5 ArUco-Markers

ArUco markers is a type of binary square fiducial markers. Each of the markers have a distinguishable bit pattern and this makes them ideal for use in vision applications such as ours.

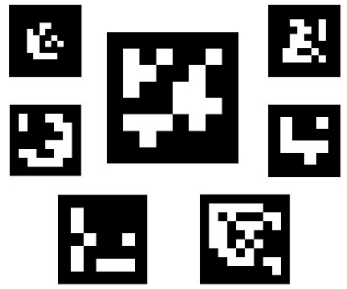


Figure 3.3: ArUco marker examples from OpenCV documentation

Marker configurations varies as seen in figure 3.3. The ones used in this paper have 6x6 squares (bits) where the outer squares are used to create the border leaving us 16 bits to create distinguishable markers.

To minimize the amount of inter marker confusion errors i.e. a marker being mistakenly identified as a different marker, special considerations have to be taken when generating marker libraries. One method for doing this effectively is outlined in Garrido-Jurado et al. (2016) and is the method used in the ArUco library for OpenCV.

3.5.1 ArUco Marker Detection

The ArUco marker detection algorithm is developed and works as described by Garrido-Jurado et al. (2014). The algorithm can be briefly summarized by the following steps:

- Convert input image to gray scale
- Perform adaptive thresholding
- Find contours
- Find rectangles
- Check if each rectangle is a marker by:
  - Warping the marker to a square
  - slice into (n+2) x (n+2) squares
  - check if edges are black
- Calculate ID for detected marker

The markers have built in error-correction that prevents false negatives even if one of the bits in the marker is flipped.

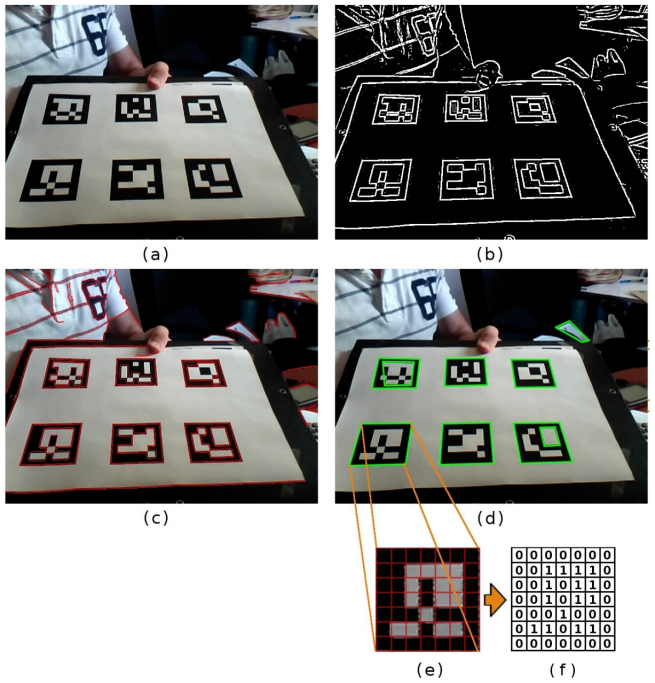


Figure 3.4: From Garrido-Jurado et al. (2014): ArUco algorithm steps: (a) Original Image. (b) Adaptive thresholding. (c) Contour detection. (d) Polygonal approximation and removal of irrelevant contours. (e) Example of marker after perspective transformation. (f) Bit assignment for each cell.

3.6 Perspective n Points

Perspective n Point (PnP) is a method for finding the six degrees of freedom of a 3D object, given a 2D image. The PnP problem solutions can be divided into two camps, iterative and analytic. According to Lucchese (2005) iterative methods are usually more computationally costly but are less sensitive to errors while analytic methods are faster and can therefore be more suited for real-time applications if you have high-quality image processing. In this paper we use iterative methods but we will mention both for context.

To use any PnP method n distinguishable points in 2D image coordinates need to be known together with the camera intrinsic matrix. All the points need to be on a rigid object and their position relative to each other in 3D space need to be known. For iterative PnP methods, an initial guess pose with depth $z \neq 0$ is also needed. It is preferable that the guess pose is as close to the solution as possible. This is to avoid errors caused by the algorithm finding local minimums.

Analytic solutions can be found with $n \geq 3$ but P3P can have up to 8 different solutions where 4 have positive z-axis (appear in front of the camera) Fischler and Bolles (1981).

For $n \geq 4$ a single analytic solution can be found but some configuration criteria have to be met and some exceptions exist as seen in Quan and Lan (1999) and Hu and Wu (2002).

Iterative solutions for the PnP problem work by projecting a guess pose x as 2D image coordinates y and compare these to the measured image point coordinates y_0 and minimizing the squared re-projection error $||y - y_0||^2$ using various algorithms. One such method is shown in the university lecture on pose estimation Hoff (2014) and is used in the minimum viable product solution in section 4.2.3

One of the more common algorithms for minimizing re-projection error in iterative PnP is the Levenberg-Marquardt algorithm (for solving non-linear least squares problems as described by Marquardt (1963)). This is also the algorithm used in the OpenCV function **solvePnP()** used by the ArUco library to find the pose of the markers.

3.7 Stereo-Vision

By comparing common points in images taken from different locations, it is possible to estimate the points 3D location. A more In-depth analysis of multiple view geometry systems in computer vision can be found in Hartley and Zisserman (2004). The two most relevant concepts for this paper will be triangulation and stereo image rectification as these are used in the function **triangulatePoints()** from the OpenCV library used in our simple implementation of 6DOF pose estimation using stereovision.

3.7.1 Triangulation

From Mussabayev et al. (2018) we see that by using a stereo image pair from two cameras with parallel optical axes we can calculate the position of a point in 3D space.

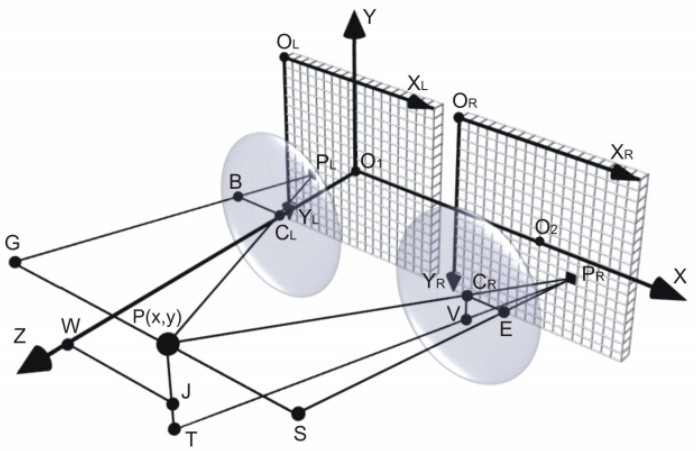


Figure 3.5: View of a parallel stereoscopic system. From Mussabayev et al. (2018)

$$x = \frac{b \cdot BC_L}{BC_L + CR_E}, \;\; y = \frac{b \cdot C_L V}{BC_L + CR_E}, \;\; z = \frac{h \cdot b}{BC_L + CR_E} + h \tag{3.19}$$

Where b is the baseline distance O_1O_2 and $h = P_RE = P_LB$ and is the same as the focal length f. It is important to note that in this figure the origin is set in the principal point and not in the focal point where we set the origin in the camera extrinsic matrix. If we take this into account and also see that $BC_L + CR_E$ is the same as the disparity in x coordinates between the two images $x_L - x_R$ we can rewrite the depth z from the focal point as:

$$z = \frac{f \cdot b}{x_L - x_R} \tag{3.20}$$

3.7.2 Stereo Image Rectification

The image planes being parallel simplifies several functions including triangulation and finding corresponding image points in both images. When a parallel configuration is not viable in the physical installation it is possible to re-project both the image planes to the same plane parallel to the baseline between the two camera focal points. This is called stereo image rectification and one method of doing this is detailed in Fusiello et al. (2000) and should showcase the concept. We have been unable to find documentation specifying the method used in OpenCV.

3.8 Finding angle between camera image plane and marker plane

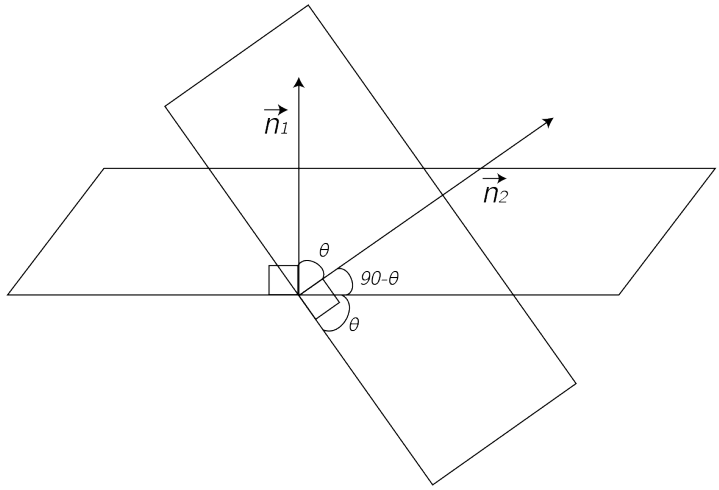


Figure 3.6: Sketch of angle between planes. Traced from Byju's (2019)

Generic Case

As shown in figure 3.6 the angle between the two planes is the same as the angle between the two normals (n_1 and n_2) of the planes. The dot product of two vectors is defined as:

$$\vec{n_1} \cdot \vec{n_2} = \|\vec{n_1}\|\|\vec{n_2}\| \, cos\theta \tag{3.21}$$

where $\|n_1\|$ indicates the Euclidean norm of the vector and θ is the angle between them. This gives us the formula for the acute angle between the planes as:

$$\theta = acos(\frac{|\vec{n_1} \cdot \vec{n_2}|}{\|\vec{n_1}\|\|\vec{n_2}\|}) \tag{3.22}$$

Special Case

For us, it was only relevant to know the angle between the camera image plane and the plane of the ArUco markers so we simplified the equation for our need.

Every column in a rotation matrix represents one of the principal axes of the object coordinate system with length 1. For our object system o defined with regard to the camera system c this can be written as:

$${}^{c}_{o}R = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \tag{3.23}$$

The camera rotation matrix with regard to itself will just be a $3x3$ identity matrix. For both our camera and object the normal on the plane created by the first two principal axes will be the same as the last column of the rotation matrix, n_1 for the object normal and n_2 for the camera normal.

$$n_1 = \begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}, n_2 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

Because of both vector n_1 and n_2 being of length 1 the product of their euclidean norms will also be 1. Following this we get

$$\theta = acos|n_1 \cdot n_2| = acos|z_3| \tag{3.25}$$

3.9 Moore-Penrose Inverse

In this project, we used the Moore-Penrose Inverse, commonly called pseudo inverse to solve the least squares problem of our initial PnP implementation.

The Moore-Penrose inverse is a generalization of the inverse matrix described by Moore (1920). The pseudoinverse is defined and unique for all matrices whose entries are real or complex numbers. It can be computed using the singular value decomposition and has been implemented in the Numpy library for Python. It is used when a matrix is degenerate/singular and the inverse does not exist, or when determining if the matrix is invertible is impractical. The most common use of the Moore-Penrose inverse is to solve the least squares problem, according to MacAusland (2014).

3.10 Norms(mathematical)

One of the norms used in this paper is the Eucledian norm, see equation 3.26. It is used to find the length of a vector. The other norm used is the Frobenius norm, see equation 3.27. This is an extension of the Eucledian norm to $m \times n$ matrices.

$$||\vec{v}|| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \tag{3.26}$$

$$||A||_F = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \tag{3.27}$$

3.11 Estimated Derivatives and the Jacobian

To implement our initial PnP solution for pose estimation we need to know how we can estimate the partial derivatives of a vector function.
In this paper, a two-point method is used to numerically estimate the first order partial derivative of a vector function at a given point. Where ϵ is a very small number and $\hat{u}_i$ is unit component i of x.

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \epsilon \hat{u}_i) - f(x)}{\epsilon} \tag{3.28}$$

The Jacobian J is a matrix of all first order partial derivatives of a vector function

$$J = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_i} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_i} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_i} \end{bmatrix} \tag{3.29}$$

3.12 Programming Principles

For our project Python and object-oriented programming (OOP) is used. In OOP there are some principles defining how good code should be written. This is mentioned in Martin (2008). Good code implements all required functionality while being *easy to read*, *reusable* and as *simple as possible*. Measurements for these things are coupling and cohesion. Coupling is how connected the classes in the project are to each other. High coupling means a change in one class forces changes in the other classes as well, which is something to avoid. Cohesion is based on how many roles a single class has. A good class should as a rule of thumb not have more than one role. Good code has high cohesion and low coupling.

Chapter 4

Method

In this chapter, we discuss the development process, explain how we have implemented our solutions and provide justification for the choices we have made along the way. We will also list all software and hardware used during development.

4.1 Work Preparations

Before starting on the development, plans were made for how the development should be carried out. There were made plans regarding work hours, responsibilities, formalities and development. See appendix D for the preliminary report.

As a starting point for development, we wanted to create a minimum viable product (MVP). We wanted a system that was as simple as possible, while still implementing all the core functionality needed for relative pose estimation. The goal was to test our hypothesizes and chosen estimation-method to learn fast and see whether our core ideas were right.

4.1.1 Hardware and Software

Software			
Name	Detail	Version	Licensing
Python 3	Programming Language	3.6.7	GPL
Jetbrains PyCharm	IDE	2.4	Apache 2
Git	Version control	2.16.1	-
Autodesk Fusion 360	CAD tool	Cloud based	Education Licence
Blender	3D creation suite	2.79b	Open Source
Ultimaker Cura	3D printer slicing application	4.0.0	Open Source
GitKraken	Version control program	5.0.4	-
GanttProject	Project time management	2.9	Open Source
NumPy	Math lib	1.15.4	MIT
Tkinter	GUI lib	8.6	Python license
matplotlib	Graphing lib	3.0.1	SD
OpenCV	Computer vision lib	4.0.0.21	Open Source
ArUco	Marker/pattern detection	3.4	GPL v3
ttkthemes	Visual GUI enhancement	2.2.3	Open Source
Quaternion	Quaternion dtype to NumPy	3.21.14.22.55	Open Source

Table 4.1: Software used for this project

Hardware			
Parts	Information	Material	Amount
Axis-cross	Used to mount markers on ship	Steel	1
Ship model	Test model for visual representation	Styrofoam	1
Logitech C920 HD	Six cameras used in final implementation	NA	6
Odroid USB-CAM 720P	Only used for testing	NA	2
Axis P3225-V MKII	IP-camera tested functionality	NA	4
Markers	Painted markers for blob analysis	Wood and Steel	1
Ultimaker 3D-Printer	Used to create parts	NA	2
Camera Stand	Stand that is adjustable and flexible	Alu and PLA filament	6
Test Rig	Used to test camera angles and different ideas	Alu	1

Table 4.2: Hardware used for this project

4.2 Minimum Viable Product

4.2.1 Initial Ideas

At the start of the project we set out with 3 main ideas.

1. Marker-based tracking. Multi-camera (stereo vision) solution with at least 3 distinguishable points in the frame. After calibration, the points can by using triangulation (see section 3.7.1) give 6 degrees of freedom (6DOF) pose.
2. Model-based pose estimation (Perspective n Point). Solves for all 6DOF with only one camera. This can be expanded with multiple cameras to achieve large enough coverage and precision.
3. Stereo vision for hull recognition. Using edge detection and possibly artificial intelligence to replicate the hull of the ship model in 3D and compare it to known models. Here only cameras are needed so you are not dependent on markers on the ship. This idea demands a higher level of knowledge from the group than the other ideas. Seeing as we are working in an environment with water reflections this can also make the solution difficult.

The first and the third option utilizes stereo vision triangulation to find depth information in the images. In configurations where the two cameras are spaced with adequate distance, we could, in theory, get high depth resolution. An obvious downside with this approach is that we would need at least two cameras to cover all parts of our entire workspace.

The third option appeared to be the most complex solution to our problem. We struggled to see a clear point of entry to how we would start solving this problem, and figured we would rather implement a working solution before we made an attempt at this implementation. An obvious upside would be that it would not require any markers on the vessel.

The model-based approach still requires dual camera coverage in the sections of the tank where a handover from one camera to the next would take place, but outside this area one camera is sufficient.

Model-based pose estimation using a single also appears to be a solved problem (4.2.3), which would allow us to focus our attention on figuring out to how to increase our coverage with multiple cameras and implementing a user-friendly GUI.

After these considerations, we decided to implement a working solution for option three first, and experiment with the other options after the first implementation was working.

4.2.2 Image Processing

Preparatory Work

Our initial idea was to use HSV segmentation to separate the red color of the spheres of the axis-cross shown in figure 4.1 from the rest of the image and then use contour detection to find the exact positions of the sphere centers. We researched this and ways to do this using the OpenCV library. We also noted that Hough circle transform and SIFT would be ways to improve our function if needed.

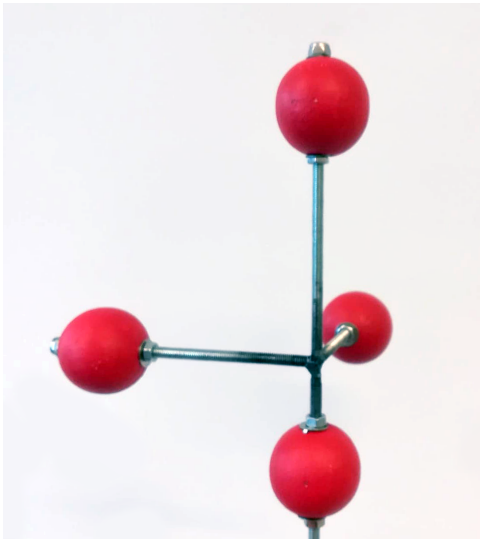


Figure 4.1: Axis cross used as reference model

Implementation

Initially, we applied a low pass Gaussian blur filter kernel to smooth the image and remove high-frequency noise. We then applied our HSV mask to separate our spheres from the rest of the image before running two iterations of erode, followed by two iterations of dilate, to perform an opening operation on the image to remove lone outliers in the segmented image that are higher than their neighbours. The opening operation is explained in pages 120-121 of Bradski and Kaehler (2013). To find the contours of our spheres we used the OpenCV function **findContours()** (see section 3.4.2) on our filtered image frame.

We checked the circularity of our contours using equation 3.18 where we get the area A using **cv2.contourArea()** and our circumference P using **cv2.arclength()**. If our blobs meet the threshold set for circularity we use the function **cv2.minEnclosingCircles()** to find the center of the sphere in (x, y) coordinates and the radius. This function is explained on pages 249-250 of Bradski and Kaehler (2013) and checks what minimum size of circle would fit around our contour. We would keep the values for the 3 largest circles and use these in our PnP algorithm.

Testing

During testing of our image processing function, we had issues with false detection of spheres due to poor selection of values for our HSV mask. To fix this we implemented functionality for manual calibration of the values using simple GUI with sliders for the different values that would continuously apply the new masks to our image frame so we could see when our spheres were being detected and nothing else.

(a) Before applying mask

(b) After applying mask

Figure 4.2: HSV masking tool

Hough circle transform was tested, but we did not achieve robust measurements. We were unable to tune the parameters of the algorithm to a point where we would get consistent readouts for each marker without also getting false positives, as seen in figure 4.3.

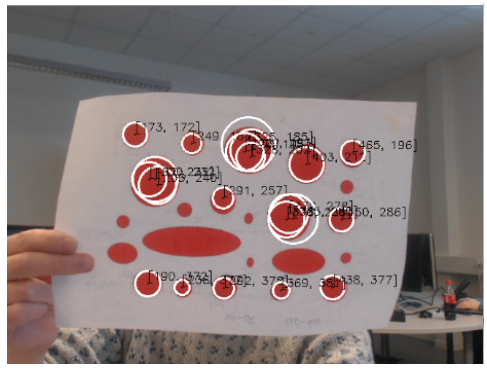


Figure 4.3: Hough circle transform experiment

4.2.3 Perspective n' Point Algorithm

Preparatory Work

During initial research on the project, we came across some lectures by William Hoff from Colorado School of Mines for their computer vision class Hoff (2014). One of these lectures was on pose estimation proposing an iterative method for solving the PnP problem. This was a very low-level solution for showing off the concept and we decided this would suit us well for gaining more understanding of the field through following the lectures while implementing and testing the solution.

Implementation

The lecture by Hoff (2014) showed us an iterative method for solving the perspective n point problem. Let $y = f(x)$ where $f(x)$ is a function that projects the image points y in camera coordinates given a 6DOF pose in ZYX Euler angles x and let y_0 be the observed image coordinates of our axis-cross markers. We want to find a pose x that minimizes the squared re-projection error $E = ||f(x) - y_0||^2$.

$$y = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 & \cdots & x_n & y_n \end{bmatrix}^T \tag{4.1}$$

$$x = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T \tag{4.2}$$

To do this we start with an initial guess pose for x where the depth $z \neq 0$. We call this guess pose x_0. We also need the camera intrinsic matrix K and the 3D position of the object points relative to the object coordinate system. For our algorithm, we had $n = 3$ points from the 3 spheres attached to our axis-cross and we knew their position relative to the origin of the axis-cross. We defined these points in a 4x3 matrix of homogeneous model coordinates P.

$$P = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

To project our 3D points in the model coordinate system as 2D image points we use the formula for perspective transform outlined in section 3.2.3.

$$sY = K \begin{bmatrix} {}^{C}_{M}R_{3x3} & {}^{C}P_{3x1} \end{bmatrix} P \tag{4.4}$$

Where ${}^{C}_{M}R_{3x3}$ can be calculated using equation 3.3 with the x rotations and ${}^{C}P_{3x1}$ is the same as the position variables of x. To remove the scaling s we divide the first two rows of the output matrix Y by the last row before removing the last row and reshaping the matrix to a 6x1 vector y on the form shown in equation 4.1. We can now find the error $\Delta y = y - y_0$.

To find the step distance Δx of our next iteration we need to estimate the partial derivatives of $f(x)$, create the Jacobian J and evaluate the derivative at the current guess pose x.

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \epsilon \hat{u}_i) - f(x)}{\epsilon} \tag{4.5}$$

$$J = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \dots & \frac{\partial f_1(x)}{\partial x_i} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \dots & \frac{\partial f_2(x)}{\partial x_i} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \dots & \frac{\partial f_n(x)}{\partial x_i} \end{bmatrix} \tag{4.6}$$

Once we have the Jacobian at x we can solve for Δx using the Moore–Penrose inverse.

$$\Delta x = J^{-1} \Delta y \tag{4.7}$$

We check if the change in x is significant by comparing the euclidean norms of x and Δx.

$$\frac{||\Delta x||}{||x||} \tag{4.8}$$

if the result of equation 4.8 is sufficiently small we stop the algorithm here. If not we update the guess pose to $x = x + \Delta x$ and run another iteration. The algorithm keeps going until the result is acceptable or the cap on iterations is reached.

For the next set of image points, we assume that the current pose estimation is correct and set the initial guess pose for this new estimation to the current pose of the camera. We do this because we run up to 20 estimations per second and the object can not realistically have moved very far away from the previous position in that time.

Testing

When doing initial testing of the algorithm we mounted the camera on the test rig shown in figure 4.7. We mounted the axis-cross depicted in figure 4.1 on one of the ship models and placed it under the rig. After calibrating the HSV values so we could distinguish the red color of the axis cross spheres from the background, we were able to estimate the pose of the ship model.

Under testing we some times had problems with the algorithm where the projected guess pose was not converging against the measured image points and the algorithm timed

out from hitting the iteration cap. This turned out to be because we were unable to distinguish the spheres on the axis-cross from each other the order of the measure image points in y_0 was not constant and would not always match the order of the 3D model points in P. To combat this we discussed using different colored spheres or objects of different shapes on the axis-cross before learning about the ArUco marker library for OpenCV that we would use in the next iterations of the project.

4.2.4 Defining a World Coordinate System

The PnP algorithm returns the pose of the axis-cross relative to the camera. Using the transformation matrix ${}^{C}_{M}T$ from the first pose generated by the algorithm as our world coordinate system renaming it ${}^{C}_{W}T$ we could use this as a reference point for all subsequent transformation matrices. The transformation of the model with regard to world coordinates would be as shown below.

$$ {}^{C}_{W}T^{-1}{}^{C}_{M}T = {}^{W}_{C}T{}^{C}_{M}T = {}^{W}_{M}T \tag{4.9} $$

Using the definition of the transformation matrix from section 3.1.3 we can use the equations for finding the ZYX Euler angles in section 3.1.4 on the rotation matrix ${}^{W}_{M}R$ and take the x, y and z coordinates form the position vector ${}^{W}P$ giving us the full 6DOF pose of the axis-cross with regard to the defined world coordinate system.

4.2.5 Camera Calibration

To use the Perspective n Point algorithm mentioned earlier we need to know the camera intrinsic matrix K. To do this we need to perform calibration of the camera. The size of the chessboard used as our calibration object is chosen to meet the recommendations mentioned in section 3.3.

Preparatory Work

The camera calibration functionality in OpenCV is well documented and numerous examples of how this can be implemented using a chessboard as the calibration object exist online in the OpenCV documentation and in the book by Bradski and Kaehler (2013).

Implementation

We first had to define our calibration object in 3D space. In our case, this was a 7×9 chessboard found in figure 4.4. The board would be defined in a $7 \times 9 \times n$ by 3 matrix P where every row represented a chessboard square corners position relative to the board origin. After 7×9 rows we copy the board n times where n represents the number of chessboard images we have where all the image points have been found. It is important that the coordinates are in millimetres as this is what we want to use as units in this project.

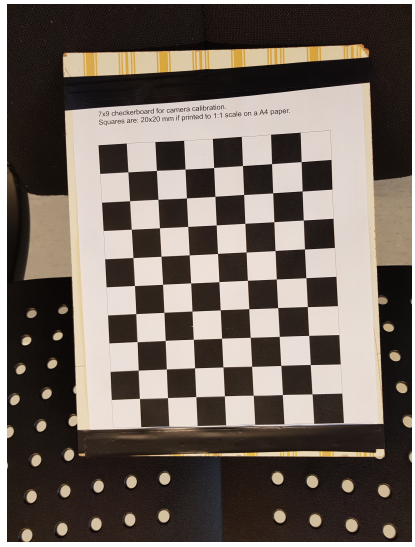


Figure 4.4: Image of calibration chessboard

To find the 2D image coordinates of the chessboard corners OpenCV has a function called **findChessboardCorners()** that takes an image frame and the size of the board in $m \times n$ squares and returns the image coordinates in a $m \times n$ by 2 matrix I. These image points are then refined using the function **cornerSubPix()** to find the sub-pixel accurate location of the corners. The image points from the next image frames are in turn added to the end of the matrix I. The images where the corners cannot all be found are discarded.

After all images have been processed and if a suitable number of calibration images have been accepted (≥ 10) the function **calibrateCamera()** is used taking the object points P, image points I and also the size of the image to initialize the intrinsic camera matrix. This function will return the intrinsic matrix K and the distortion coefficients explained in section 3.3.

To capture images for calibration we created functionality for grabbing a frame from the video feed at certain intervals once we started the calibration so we could just stand in place with the board gently moving it around. This made it a lot simpler to perform calibration alone.

4.2.6 Choice of Camera

Camera Information				
Camera Type	Resolution	FOV	Adjustable Focus	Price
Logitech C920 HD	1920x1080	71°	Yes	599 NOK
Odroid USB-CAM 720p	1280x720	68°	No	150 NOK
Axis P3225-V MKII	1920x1080	92°	Yes	4500 NOK

The Odroid camera provided decent sharpness but its field of view was narrower than the other cameras and therefore it results in the need for more cameras to cover the entire towing tank. The model we tested was taken out of production, so we did not have the option to buy more units. The camera also lacked a tripod mount, and designing a solution for holding the camera in place would demand more hours being used to create a more complex design - Therefore it was not usable for our project and was only used for small scale testing.

The Axis camera provided sharp images, and since it is possible to pan and tilt it, it could, in theory, give us the advantage of using fewer cameras. It has a horizontal field of view which ranges from 92° to 34° when zoomed in. The function of panning and tilting the camera gives an extra movement of ±180° panning, -35 to +75° tilting and ±95° rotation.

The Axis camera is more expensive and implementation in Python would require additional time to create a functional communication protocol. We decided that for this would become a to great of a task for us to finish. Also, since we got the cameras quite late in the project, it would require to much time for us to deliver a usable product.

The Logitech C920 is a web camera mostly used for video feed transfer in locations such as offices. It has a wide angle lens and gives a horizontal FOV of 70.42° and vertical FOV 78°. Because of this, it can cover larger areas with fewer units. It has better sharpness than the Odroid and the Axis camera. It is easy to set up, and has its own software for adjustments.

From usage we found that the cameras we had could have been damaged or have wrong factory settings. When taking pictures at the same distance, we got different results from the two cameras we had. The focus at a working distance of 1.2 meter did not meet our needed requirements.

After researching how to improve the focus, we found a guide that in detail explained how to open up the camera and adjusting the focus ring without damaging it. This modification increased the performance. This is shown when comparing figure 5.1b to figure 5.1c and figure 5.1d.

Logitech C920 HD is of good quality relative to its price. The Axis P3225-V MKII is a better camera but at a far more expensive price range. It also needs to be setup in a more comprehensive way. The camera would need a static IP address and controlling it is done by an own protocol that would demand that the user has far more knowledge than the other cameras.

Our tests show that the Logitech C920 HD gives good quality compared to the price. It is also easy fixed in the tank because it has a tripod mount that can be used to fasten it to a stand.

After choosing a camera type, it was placed on the side of the tank. It gave a smaller field of view than wanted. This was improved by mounting the cameras to a track on the ceiling. It gave a larger field of view than original, which also resulted in the need of fewer cameras.

4.2.7 Simulation

in order to test our image processing- and tracking algorithms, we needed images of the axis cross. We wanted to test the system without the added noise and distortion we could

expect from a camera, so we created a simulator for exporting pictures of a 3D modeled vessel with the attached markers from different camera angles.

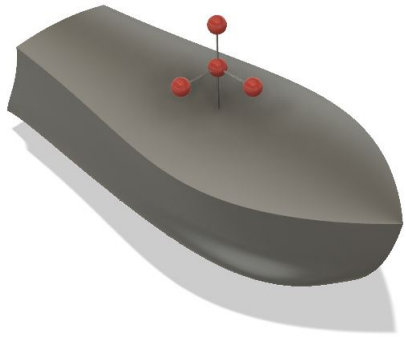


Figure 4.5: 3D drawing of possible solution for markers on boat

The 3D-model was initially drawn in Autodesk Fusion 360 and later exported to Blender for rendering. Blender provided us with easy to use rendering options, where we could recreate our exported images with good control over the location, orientation, camera settings and lighting.

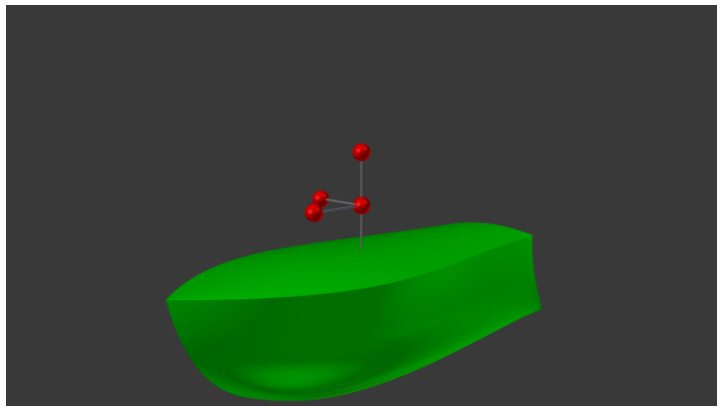


Figure 4.6: Generated camera angle from Blender

4.2.8 Camera frame for small scale testing

In order to get consistent setups for our cameras under testing, we created a test rig on which we would attach our cameras. The vessel would then be moved under the frame

during testing as we checked if our results were consistent with the actual movement of the vessel.

Figure 4.7: Photo of test rig: The rigs measurement is 1x1x0.5 meters.

4.2.9 Software Architecture

Preparatory Work

To start with the coding a software language and method was chosen. We considered Java, C++ and Python. Everyone was familiar with Java from earlier courses, but the OpenCV documentation is not very thorough for this language. When considering C++, we found out it had the potential to run more effectively in the final product. An obvious downside was that none of the team members had used it before, and it is not known to be beginner friendly. Python was well known for one of the team members who had used it in a project the past year. In the end, Python was chosen due to our past experience with it, and because it appeared to work well with OpenCV.

Python supports both object-oriented (OOP) and functional programming (FP) architectures. FP can often provide more compact than OOP, which could, in theory, mean that we as developers could write less code. Object-Oriented code is known to be well suited for coding in teams since it offers high modularity in each code block.

The entire team already had experience with OOP, but only limited experience with FP. We would have to consider relearning programming patterns specific for functional programming if we decided to go for that route. Instead we decided to go for the Object-Oriented Programming approach, as it seemed to fit our skills and needs well.

MVP processes

The primary goal for the architecture for MVP was to separate the processes into different classes, with the goal that a design change in one part shouldn't require a complete refactoring in the rest. The planned architecture is shown in figure 4.8.

The fundamental processes needed to be done:

1. All cameras takes a picture
2. Find model points in images
3. Estimate the position of the axis-cross with regard to the world coordinate system for each image.
4. Sensor fusion and comparison and validation of the results.
5. The results are given to GUI, visualization and logging section.

Figure 4.10a visualizes how the processes is solved. Each camera would run their own thread and send their image frame and PnP-solution to the PoseEstimator, where a final pose would be found. This would be sent through the Connector to the UI, csv-file and graph plotting.

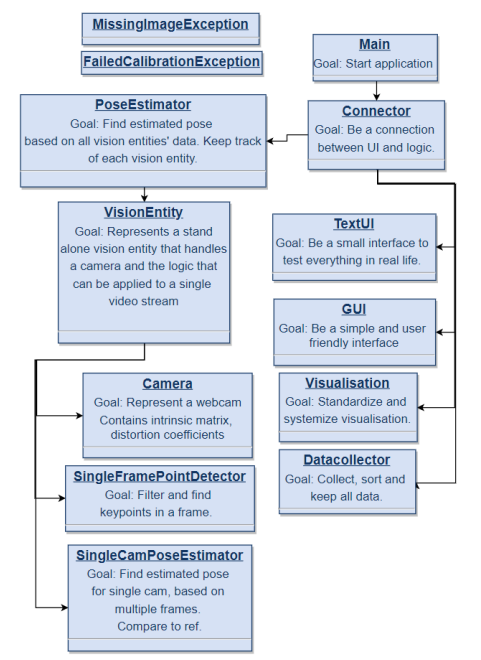
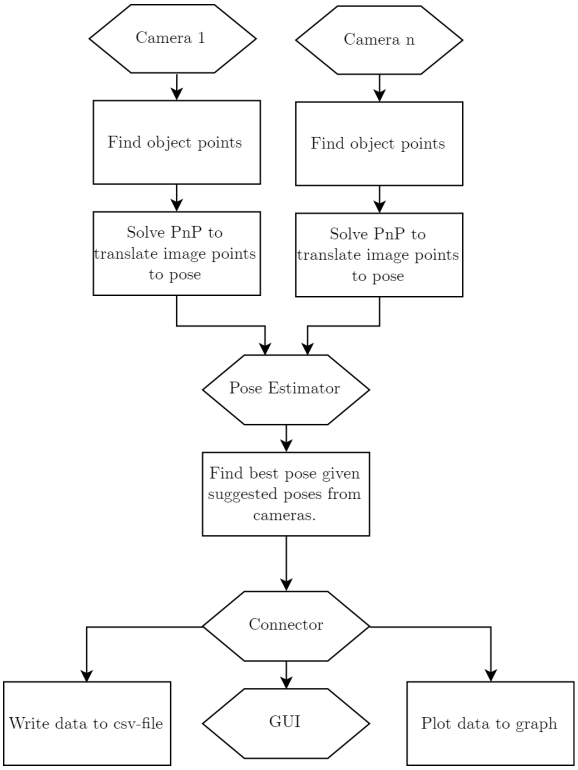
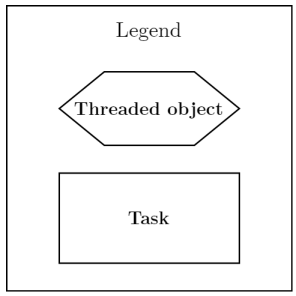


Figure 4.8: Class architecture for MVP.



(a) This diagram shows how the system processes information from the data collection in the camera until it is displayed to the user.



(b) Legend for flow diagram

Figure 4.9: Data flow diagram

As the purpose of MVP is to create a useful prototype, the only interface created was a Text UI that runs in the command line. TextUI is explained in section 4.2.10.

Implementation

The first step of implementation was to write, test and validate the classes for a single camera. None of the classes was used in the main program before they were working well alone. The image processing was described in section 4.2.2, and the pose estimation in section 4.2.3.

Since the pose estimation from several cameras would run in parallel, threading was used. Each created *VisionEntity* is run in a separate thread, as well as the *PoseEstimator*. It required some planning for how variables were accessed but presented no problem.

To debug and report errors in the code, the inbuilt logging-library was used instead of the normal print function. It can immediately see where the message was written from and see the most critical errors easily. We also write all detected errors to an error-file for ease debug.

In order to make a robust application, you need to keep track of which cameras are where.

Camera indexing

When using multiple USB-cameras with OpenCV, there is no protocol for identifying which camera is connected on what port.

Webcam access in OpenCV is based on indexing. OpenCV makes a secret list of available cameras connected and when asked returns the camera found in the position that corresponds to the index asked for. This means, the list used one day, isn't necessarily the same the next day. No info about what camera is connected where is stored. This means we can't automatically assign calibration files and eventual extrinsic camera parameters to cameras directly as we don't know which cameras are connected. Instead the user must check the stream of each camera and assign a calibration file corresponding to the given camera name.

With library win32com ("Windows Python 3.2 Communication") we are able to get a list of IDs, where one ID is one USB-connection. From the ID list we can read off the ID of the webcam driver and the USB port-number. But since we don't have camera-unique numbers (i.e. serial numbers), this is not useful if we use several cameras of the same type. Unfortunately, as OpenCV is written in C++ the OpenCV camera-list is not accessible in Python. OpenCV supports using USB port-number to access a camera, but only for Linux and not for Windows.

Since we were unable to address cameras directly, we created a solution that would let the user preview each connected camera before adding it to the tracking system. This would let them choose the relevant cameras and decide the calibration file that should be assigned for each camera.

4.2.10 Text User Interface

Preparatory Work

To test the system, an interface was needed. A graphical user interface is often very complex and time-consuming, which means the interface itself needs many hours of work to function properly. Early in the project, a bug-free and simple interface is much more important than a nice looking one. We, therefore, created a command line interface.

Implementation

The focus was to make things simple and functional. We wanted the interface to have the same functionality as the GUI would have in the future. Implementing the TUI took less than two working days.

```
 SHIP POSE ESTIMATOR @ NTNU 2019
Please make your choice:
1. Start application
2. Change model parameters
3. Configure cameras
4. Exit
Type: 3

Please make your choice:
1. Initialise connected cameras
2. Initialise connected cameras except screencam
3. List cameras
4. Import cameras
5. Calibrate cameras
6. Test cameras
7. Videotest cameras
8. Show current calib params, index 0
9. Fix HSV-calibration
10. Load HSV-values
11. Go back.
Type:
```

Figure 4.11: Main menus in TextUI

Usage

The TUI did its job well as a debugging tool before we got to the point where a working GUI was implemented. It made it easy to add, remove and edit functionality in the code without spending much time in rewriting the user interface.

4.3 Implementing ArUco

Due to our not so successful attempts to achieve a robust system based on the spherical marker approach, we were on the lookout for alternative approaches for a tracking and pose estimation engine. Square planar markers, normally associated with QR-codes, could provide us with solutions to our most obvious problem - namely getting a unique identifier associated with each point of interest.

After a dive into available tech - the ArUco library based on OpenCV stood out to us as an excellent candidate, promising high performance, robustness, and free to use software.

4.3.1 Tracking ArUco Boards

Preparatory work

A coarse implementation with a single camera detecting the pose of a single marker was implemented as a proof of concept, and both the performance and resolution of this demo program was better than our spherical marker based implementation. In addition, we had no problems with false readouts. Due to the performance, compatibility with our existing solution written in Python and open source accessibility, we decided to switch to an ArUco maker based approach. We also saw value in the fact that the ArUco Markers could be printed on paper, and thus be virtually massless compared to our spherical marker rig made from wooden spheres and steel rods.

Implementation

The ArUco Library conveniently implemented a lot of our previous work. The library implemented point detection and the solving of the perspective N-point problem neatly into its own functions, so the goal for our first iteration in the development of our system was to express these solutions as Euler Coordinates in a coordinate system where we could decide the origin.

The OpenCV solution to the SolvePnP-algorithm provides an answer in the format of two vectors, rvec - the rotation vector, and tvec - the translation vector. Our initial plan was to stick with this regime since most of the OpenCV- and ArUco-library uses the translations and rotations in this format. This did, however, pose some challenges.

A Rodrigues' rotation vector is a convenient and compact way of representing a rotation, but in order to do actual calculations where rotations are combined, or inverted, we would have to transform the vector into a matrix first, or even use quaternions.

If we were dealing with rotations only, quaternions would be an obvious choice, since they are compact, more readable and are less computationally demanding than rotation matrices. They do however lose some of their desired properties when our pose contains both rotations and translations. Dealing with a translation vector on the side of quaternions would not be impossible, but it seemed less readable and convenient than the alternative we eventually went for.

A 4x4 Homogeneous Matrix combines a 3x3 rotational matrix, a 1x3 translation column vector and fills the last row with three trailing zeros followed by a one. The property that makes this matrix so desirable for us is that they can be combined with simple matrix multiplication, and they can easily be decomposed back to their rotational and translational form (see section 3.1.3). By building a small class of helper functions we were able to supplement these matrices with functions for converting them to the OpenCV rvec/tvec regime when needed.

The final implementation of the ArUco Board Tracker runs in each of our so-called "Vision Entity"-objects, which are threaded objects within the software that is responsible for the data collection and processing of a single camera.

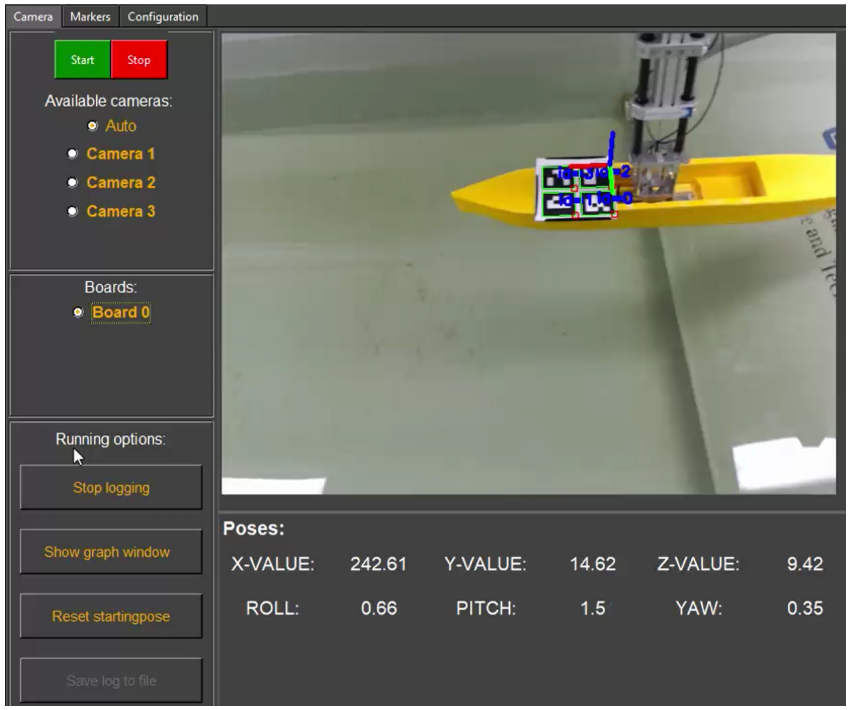


Figure 4.12: Live tracking a Vessel

Each Vision Entity is continually reading frames from their respective cameras and analyzes the frames looking for ArUco Boards. When a board is found, the Vision Entity records the position of the board relative to the camera. This transformation will be written as ${}^{C}_{M}T$ in the next section 4.3.2.

4.3.2 Multi Camera Tracking

Calculating Relative Position of each Camera

When a tracked board enters the image frame of one of the cameras for the first time, the world coordinate system is defined to set the boards position and rotation to the origin. How we define the world origin is the same as explained in section 4.2.4 When a camera that has not yet calculated its pose sees a board that has an affiliated position related to the world coordinate system, the camera will use the boards' position to calculate its own pose in relation to the world coordinate system.

$$ {}^{W}_{C}T = {}^{W}_{M}T\,{}^{C}_{M}T^{-1} = {}^{W}_{M}T\,{}^{M}_{C}T \tag{4.10} $$

This transformation can then be used to find the board position relative to the world for all subsequent poses found by the new camera.

$$ {}^{W}_{M}T = {}^{W}_{C}T\,{}^{C}_{M}T \tag{4.11} $$

Improving Camera Position

When calculating the subsequent cameras positions we run into the same issue as when merging boards. A new camera position is initially calculated when the board is in the edge of the camera frame, and while most of the boards' markers are still outside the image frame. This means that we, in theory, should be able to calculate the position of the subsequent camera at a later time when the entire board is visible in order to improve upon the initial estimate of the cameras pose.

This was solved by implementing a certainty measurement algorithm that would amend the subsequent cameras positions if a higher certainty estimate of the position was available. The algorithm works as follows:

The camera pose certainty of the first camera is always set to 1, as the quality of the initial camera position can never be improved upon, as it is the first static variable in or world coordinates.

As the board is being tracked, the tracking camera is updating the Board Pose Certainty. The certainty of the poses of the different objects are measured on a scale from zero to one, and this scale is defined by the fraction of visible markers on the board.

$$\text{Board Pose Certainty} = \text{Camera Pose Certainty} * \frac{\text{Visible Board Markers}}{\text{Total Board Markers}} \tag{4.12}$$

Similarly, when the subsequent cameras have a board in sight that is still being tracked by another camera, they will compare their current pose quality with their potential pose, and if they believe that their pose can get a higher quality, they will make a new estimate for their positions, and update heir pose quality to the new value.

$$\text{Potential Camera Pose Certainty} = \text{Board Pose Certainty} * \frac{\text{Visible Board Markers}}{\text{Total Board Markers}} \tag{4.13}$$

Calculating Board Position when Several Cameras has Board in Sight

When more than one camera has the same board in sight at the same time, only the camera that can offer the highest board quality will write its estimated pose to the board.

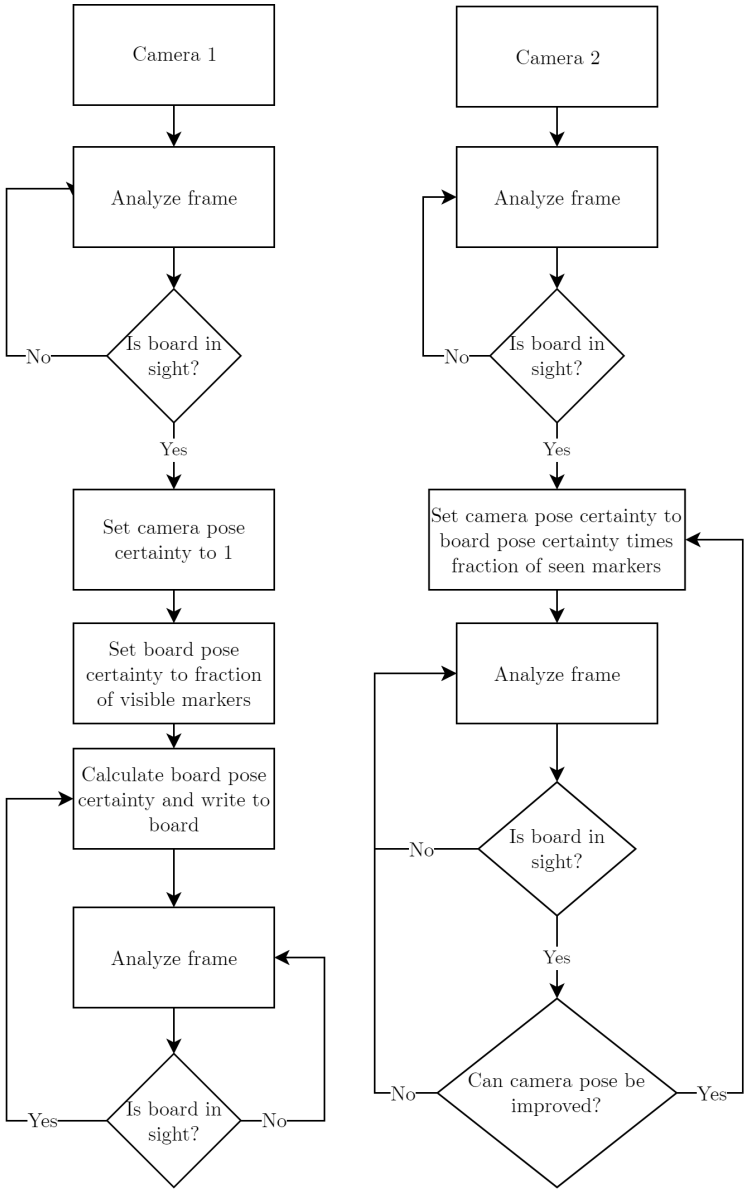


Figure 4.13: Algorithm for calculating camera pose certainty

Automatic Switching in Camera Views

The GUI offers an "automatic" setting that always shows the camera that is currently writing poses to the board. This function relies on the same algorithm as in the previous sub-chapter.

4.3.3 Multi-Object Tracking

The system has built-in functionality to track several boards simultaneously. When more than one board is being tracked, the cameras detect all the visible markers in the frame and calculates and saves the position of each respective board.

The multi-object tracking was one of the simpler implementations of the project, as most of the programming of this functionality was done by abstracting the tracking classes to accept lists of boards and repeating some of the tracking steps for each board.

4.3.4 Software Architecture

Switching to ArUco markers naturally meant some changes in the code, but the class structure mostly stayed the same. See figure 4.14. The old classes for image filtering and pose estimation wasn't used anymore and removed. To keep track of markers, a class *ArUcoBoard* was added. The class-object would hold a single board, with a distinct ID according to the ArUco dictionary. The boards would be created and managed from GUI but will be sent down PoseEstimator as soon as tracking starts.

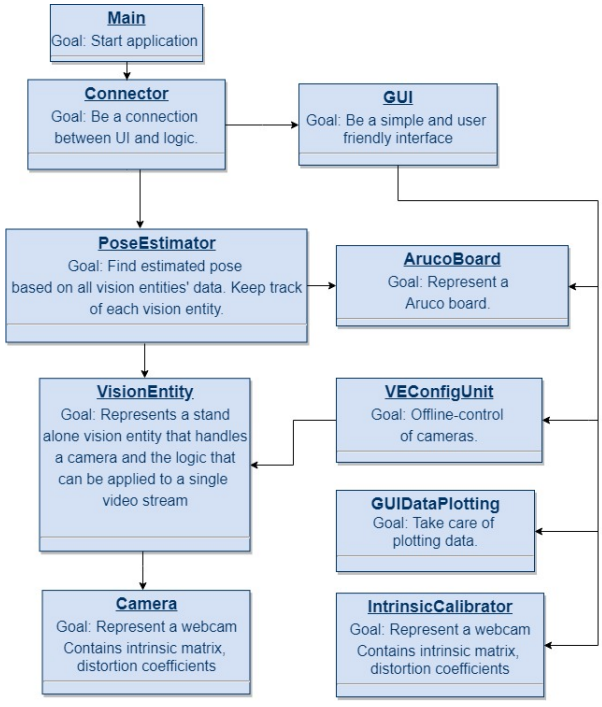


Figure 4.14: First SW architecture with ArUco markers

4.3.5 GUI

Preparatory Work

For the GUI, a library had to be chosen. The most important features except the basic things were the possibility of good live video streaming and live graph plotting. The choices are shown in table 4.3. From earlier, one team member had much experience with Kivy [1] but that was mostly specialized for touch and tablet hardware. Qt [2] is the largest and possibly most feature-rich GUI library for Python. But from our earlier experience we knew setting it up was not trouble-free, and the price was high if we wouldn't go Open Source. After some research, we decided to go with Tkinter [3]. This mainly because Tkinter would do everything we wanted, like visualizing graphs and integrating video streams. Based on previous experience we knew that Tkinter would be able to solve our problems regarding the GUI.

GUI Library				
Feature	**Priority**	**Kivy**	**Tkinter**	**Qt**
Live video	High	Yes	Yes	Yes
Live graphing	High	Not directly	Yes	Yes
Earlier experience	Medium-High	Yes	Yes	No
GUI Builder	Medium	No	No	Yes
Price	Medium	Free	Free	Free if Open Source $459/mo. Commercial

Table 4.3: Features of considered GUI libraries

Implementation

It was early decided to separate the different parts of the GUI by tabs. That would make things easier to program and more user-friendly. We started out by creating five sections, or tabs:

- Live screen - camera stream and overview
- Calibration - intrinsic calibration
- ArUco - to create ArUco markers
- Graphing - display graphs
- Configuration - for cameras

[1] www.Kivy.org
[2] www.Qt.io
[3] www.wiki.python.org/moin/TkInter

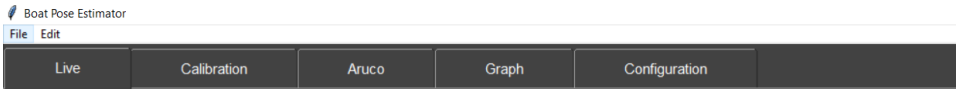


Figure 4.15: Initial GUI tabs

The final result were 3 tabs, where 'Calibration' has been moved under Configuration and 'Graphing' is in a separate window.

Configuration-tab

In the configuration-tab, the user can select which cameras to use in the system. Figure 4.16 show camera options in the GUI.

The camera can be connected to the system, the camera-stream can be shown and calibrations-parameters can be loaded. These options are a necessity because the user must identify and connect each camera individually by themselves. This necessity is given by the problem described in section 4.2.9, where OpenCV or Windows doesn't recognize USB-cameras individually themselves.

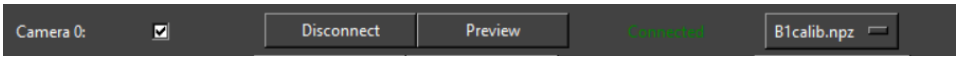


Figure 4.16: GUI: Camera options.

Creating ArUco Boards

ArUco boards can be customized in the GUI, see figure 4.17. A number of markers and their size can be set. The markers are not randomly created. By default, a number of markers are created, in our program 50. They are listed with an index. No matter the length of the list, the first marker will always be id 0 and have the same bit-mapping every time. Same with second, third, etc. This means we can use the board placed on the model again and again, as long as the creation order is the same. When a board is customized, it can be saved and a pdf-document will be created with the board. The user can then print it out and it's ready to use.

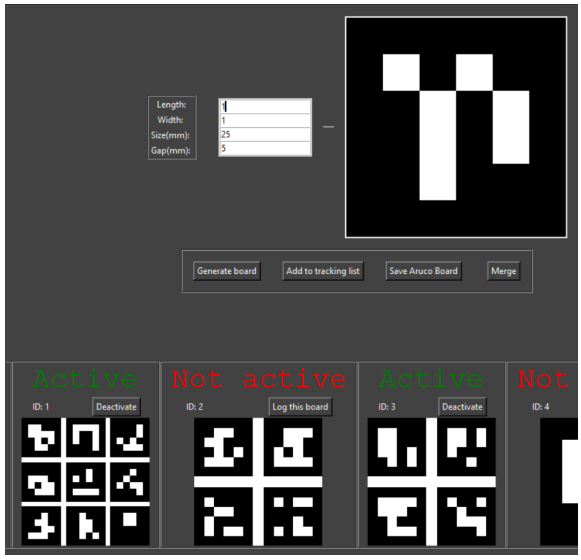


Figure 4.17: GUI: Creation of markers in Marker-tab.

4.4 Final Development ArUco

Here we describe the final development of the software. The main functionality added was the possibility to 'merge' boards, meaning setting up so several markers can act as a single one. This was seen useful when a model had several markers added since we aren't interested in several poses for the same boat.

4.4.1 ArUco Merger

In addition to creating boards, we also wanted functionality for merging several boards together. An ArUco board can be created either by creating a grid board, which is a collection of markers placed in a square grid, or by defining the position of the corners of each marker in 3D space.

The first option makes for easily defined boards but limits our marker placement to a single plane, and to a grid pattern. If we could place markers outside the square grid, we would be able to track our model from more angles, and we would also imagine that we could get more accurate measurements since we could read more depth information from a frame with markers placed in more than a single plane.

The latter option for creating boards, where we would define the position of each individual marker is not suited for manual creation, since it is unpractical to measure and enter all the positions by hand.

We, therefore, proposed a solution where we would create two or more grid boards, define one of them as the main board, and the others as sub-boards. Then we would estimate the pose of the sub-boards relative to the main board, and get a transformation matrix we could use to transform the corner points of the sub-boards into the coordinate

system of the main board. By appending these new corner points to the list of corner points in the main board, the main board and the sub-boards would be merged into a single board.

Preparatory Work

When we prepared this functionality, we had to answer the question of how the transformation between the sub-boards and the main board would be calculated.

In the simplest case, where we had one main board and a single sub-board, the only way to find the relative pose between the sub-board and the main board would be to calculate the pose for both from a single image frame. See figure: 4.18

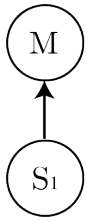


Figure 4.18: ArUco Merger: Single sub-board graph

However, when there are more than two boards, it should be possible to know the relative position between the two sub-boards, and between one of the sub-boards and another sub-board. By chaining the relative poses together, we could find the relative position between the second sub-board and the main board without observing this relation directly. see figure: 4.19

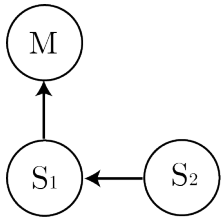


Figure 4.19: ArUco Merger: Chained graph

Considering more complex situations, we quickly run into the next problem to consider: What do we do in a case where we have more than one way from a sub-board to the main board? See figure: 4.20

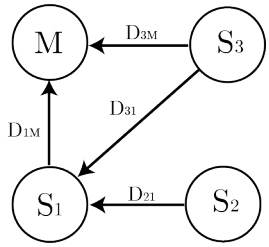


Figure 4.20: ArUco Merger: Weighted graph

By defining some cost between each transformation, it is possible to search for the path with the lowest cost. By implementing a shortest path algorithm, like Dijkstra's algorithm or the A* search algorithm, we could calculate the lowest cost transformations for a sub-board to the main board.

Before starting the work with implementing this solution, we decided to seek advice from Arne Styve, who after a fruitful conversation advised us not to go for a solution this complex, and instead only consider direct transformations between each sub-board and the main board.

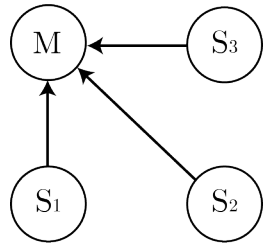


Figure 4.21: ArUco Merger: Graph using only direct transformations

This advice was easily justifiable since each transformation in practice would have a cost so high that a direct transformation would almost always be preferable to an indirect one. By completing the board merging prior to starting the actual logging of the model pose, we would get high-quality transformations for all sub-boards, and we would not worry about having "down time" where some of the sub-boards were unconnected to the main board.

The downside to this approach is that we would need at least one measurement where each sub-board and the main board would be in the same image frame during the calibration, but we considered this as easily solvable in most cases.

Implementation

As this part of the project was implemented at a late stage, the implementation was initially done in a separate environment. As a general test for this functionality, we decided to use a cube with a marker placed on each side, with the goal of merging five faces of the cube to a single board. See figure: 4.22

Figure 4.22: Aruco Merger: Photo of marker cube

We observed that when we rotated the cube to reveal the next face, the first measurement we got was weak, due to the steep angle between the newly revealed marker and the camera. This made us consider how we would calculate the transformation between the boards since the first measurement surely would not be the best.

By finding the angle between the camera plane and the observed marker as described in chapter 3.8, we found that the cosine of the angle between the camera and the marker would increase with the visibility of the markers. This variable was easily obtainable from the camera to model rotation matrix, (3.4) as r_{33}, as described in equation 3.22.

We considered averaging several measurements, where we weighted each measurement as the minimum of the cosines of the angle between each board and the camera. The problem that arose from this weighting was a need to calculate an average between the rotations of each transformation. There are several implementations of averaging rotations in the quaternion space, but we decided against using it and decided to use our single strongest measurement, due to the complexity of these operations, and that the expected payoff seemed low.

Testing

Unfortunately, the current implementation of our merger is not consistent in giving good result for the merged boards. Small errors in the relative pose of the boards results in large errors when calculating the pose in the merged board. The PnP-algorithm seems to give us some strange answers when the object points are incompatible with the observations.

We believe that the merger can be improved by using more accurate methods of measurement, like a stereo camera solution, or by using a more robust method of considering the pose of the merged board. One can imagine measuring the pose each board separately, and only considering the main board when more than one board is visible, and only using the most visible sub-board when more than one sub-board is visible.

4.4.2 Pose Quality

We decided that having a numerical value representing the quality of the pose estimation would be helpful seeing as we use multiple cameras to provide images for our estimations

and some can be more suited than others depending on multiple factors.

Preparatory work

We hypothesized that the accuracy would be most heavily impacted by three factors; The number of markers recognized by the algorithm (marker occlusion), the distance from the camera to the object and the angle of the marker plane with regard to the camera plane. During research, we found information that corroborates with our hypothesis about angle and distance in Pentenrieder et al. (2019) and López-Cerón and Cañas (2016).

Implementation

The number of identified markers was found by taking the length of the list of marker Ids returned by the ArUco function **detectMarkers()**.

To find the distance D from the camera to the marker corner we defined as origin we took the Eucledian norm of the position vector ${}^{C}_{M}P$ of the transformation of the marker object M with regard to the camera C, ${}^{C}_{M}T$.

$$D = ||{}^{C}_{M}P|| \tag{4.14}$$

To find the angle of the object plane with regard to the camera plane we use the equation found in section 3.8. We set our accuracy value gained from the angle between the object and the camera as the cosine of the angle as this will be 0 when the board is perpendicular on the camera and 1 when they are facing each other.

$$cos\theta = |n_1 \cdot n_2| = |z_3| \tag{4.15}$$

We assumed that marker occlusion, distance to marker and angle of the camera would not impact the result the same so we needed to add weights to the coefficients. These weights would be decided by our own testing done in section 4.7 and by analyzing results from other similar projects like Pentenrieder et al. (2019) and López-Cerón and Cañas (2016). We never got to implementing these weights because of time limitations and this not being of high priority.

4.4.3 Software Architecture

The software architecture wasn't affected much by the latest development. The biggest change was GUI becoming the starting class. It was some discussion about whether GUI or Connector should run the main loop for the program. In earlier versions, the main loop was initiated in the Connector. This meant the Connector would have the GUI about changes in the GUI. We decided to move the main loop to GUI.This cleaned up the data flow between Connector and GUI, although the GUI got a bit more responsibility.

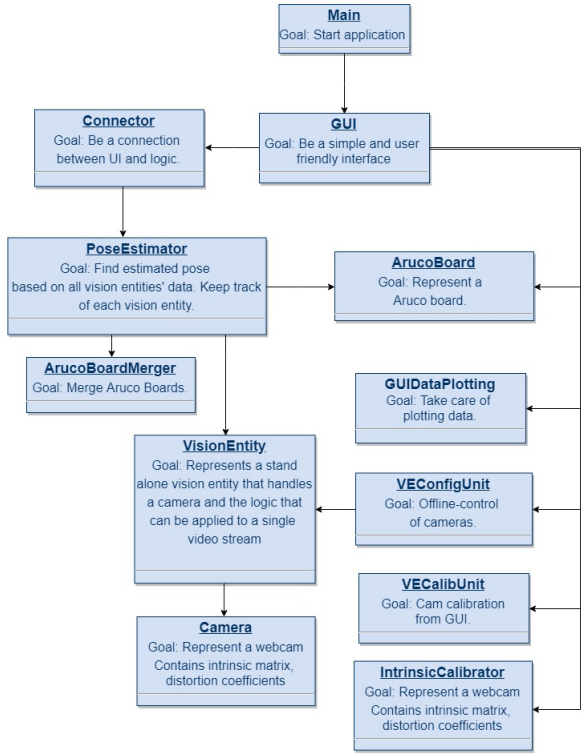


Figure 4.23: Software Architecture Final Version

4.4.4 GUI

Functionality for doing calibration was added. The user could now calibrate all cameras from the GUI. It was refactored functionality so a VE could be connected and previewed in the calibration tab, the same way as in the camera config tab. After the calibration, which now takes between 15-60 seconds for each camera, the created file with parameters becomes available as a choice in the camera-config tab and is directly inserted to the camera when chosen.

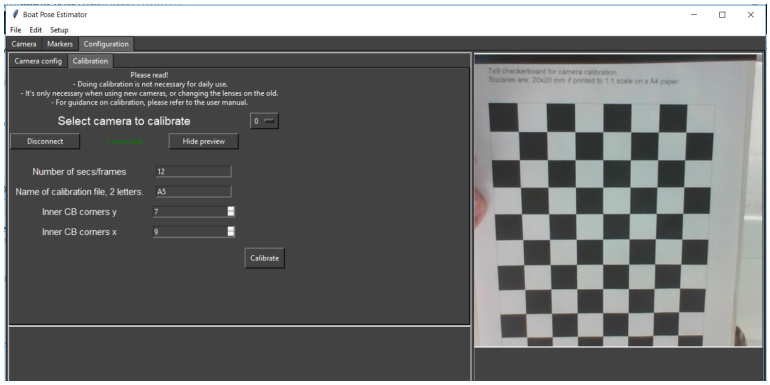


Figure 4.24: GUI Calibration Tab

Functionality for the merging was also added. Main marker and which markers to merge with can be set. During merging, live feed is given and bars showing the quality of each merge is displayed, see figure 4.25.

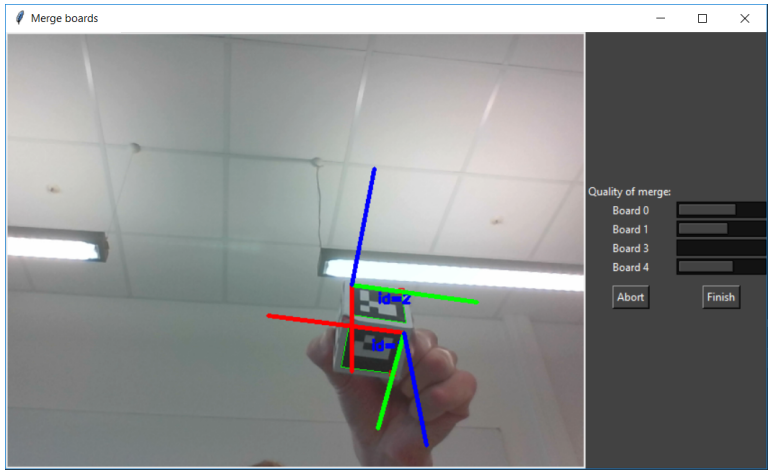


Figure 4.25: Screenshot taken during merging process

4.4.5 User Manual

Since the system was meant used for students and teachers, a user manual was created. The manual is found in appendix A. It goes through all necessary steps to get every feature working.

4.5 Installation

Before mounting the camera setup in the towing tank, preliminary work had to be done. Measuring distance between cameras and testing handover from one camera feed to another at a given distance. This was done at the operational distance of the trolley. After this, more camera stands were put on the side of the tank. A testing zone for the computer was also setup, so that we could use a smaller area of the tank. This was to make sure we were of no hindrance to other people that also used the tank for testing other projects.

This gave an indication of how many cameras were needed to cover the entire operational length of the towing tank. We needed to use at least 5 cameras with a 90 cm distance between them.

Also when testing, we have to check if the first alternative of putting the cameras on the side of the tank is the best solution. The second alternative is to use a track that is mounted in the ceiling so that the camera stand can be fastened to this and the camera would have to be rotated upside down. Also, an extra bracket is made so that we can connect the camera stand to the track in the ceiling. Drawing of this can be found in appendix B page 7. This is made so that it can be slid into the track and connected to the already made camera stand. An image of how this was done is show in figure 6.4b

If we were to have the cameras on the side of the tank, it would be a much larger operation to implement them. We would have to either create holes to fasten them by drilling in the frame of the tank. This is not ideal since we do not have the equipment to do it. And also it could also be needed to drain the tank to be able to do this.

We used the program that already exists to control the towing tank. It has the possibility to specify where the trolley should move to, and also generating waves with a given height and interval. This was used to test if the handover from camera to camera worked. We fixed the boat to the trolley and towed it at a given distance with a given length of towing specified in the program.

4.5.1 Camera Stand

A camera stand was made to place the camera in the right position on the side of the towing tank. This was done by creating different drawings of solutions which later on was 3D-printed, figure 6.4a. The first version, drawing shown at appendix B page 1, did not produce a good result. It is based on adding multiple ball joints together with a locking ring to prevent it from loosening.

Because the design was complex, the parts were also sensitive to change of heat when it was printed. The result was not up to standard and a new solution had to be made. A more robust design principal with fewer parts and less complexity provided a good result. It is also mountable on the side of the tank or the ceiling. And the height is adjustable.

Appendix B, page 3 shows drawing of this. Each camera stand should be placed with a distance of 90 cm between them to ensure that the cameras are overlapping so that the handover from camera to camera works when using the trolley on the towing tank.

4.6 Logging and Storing Data

4.6.1 Displaying Real-Time Data in GUI

In order to analyze the vessels movement over time, we decided to implement a system for displaying this data in a clear manner. We considered two options for the real-time logging:

Our first option was to display the boat position as a line in a 3D graph. This approach would be intuitive and would show the vessels path through the towing tank. Displaying the orientation of the vessel would not be as easy in a set up like this, and we would also lose the time dimension.

Therefore, we decided to go for a six window set up, where we would plot each degree of freedom in separate subplots, see figure 4.26. This is also similar to how the data is displayed by the sensor currently implemented in the tank, and would, therefore, be easy to understand for the user.

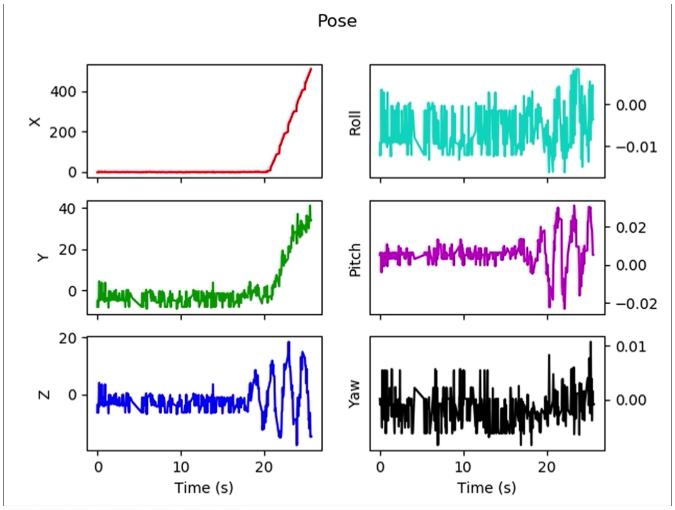


Figure 4.26: Logging window from GUI. Orientations are displayed in radians, but are changed to degrees in the final product. One can see the tow starting at 20 seconds in the x-plot. Also observe the z-plot, where waves from the wave generator hits the vessel at about 18 seconds.

4.6.2 Saving Logs to CSV-Files

In order to save a session for the future or do further analysis on the data gathered, we have implemented functionality for saving the logged data to CSV-files.

We decided to do this since both our client and our supervisors advised us to, and we believe that it drastically increases the systems utility value. Storing the data is critical in order to compare how different vessels behave in the tank and even if this is a small part of our system, it is an important one.

The logger stores 20 data points each second for each tracked board, and all the data is written to a CSV-file on the users request.

4.7 Position Accuracy Testing

We wanted to determine the accuracy in 3D position of the object at different angles of the camera to see how this would affect the result. To do this we mounted a 3x3 ArUco board with 40x40mm markers to a UR3 robot as shown below in figure 4.27.

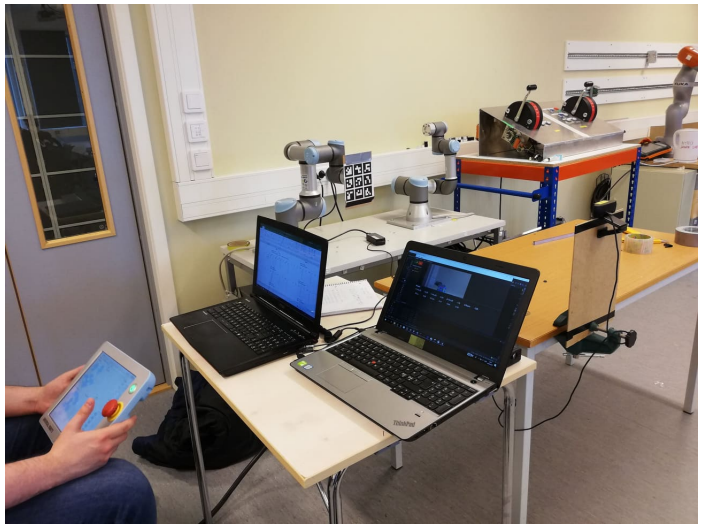


Figure 4.27: Photo of accuracy test setup

We made a square board out of wood with known dimensions with accurate markings for where it would be mounted on the robot and how we would have to tape the paper sheet with the markers to the board. We would know from doing this what offset to add to the tool center point (TCP) to get accurate readings.

We mounted the camera 100cm away directly opposite of the robot using a table vice and a wooden board so that the angle between the camera and the board would be 180 degrees (facing each other). The camera was mounted on a height such that the board, when placed in origin, would be at the center of the image. We wrote a short script to broadcast the calculated angle between the board and the camera so we would not have to measure it. To simplify the reading of the robot TCP position we set this current position as origin in the teach pendant so we could directly compare the readings from the pendant and our systems GUI.

When testing the accuracy we would move the robot along a single axis in 10mm intervals from 0 up to and including 50mm before moving back to origin and repeating for the remaining axes. When we finished testing along all 3 axes we would rotate the board around the x-axis (roll) to 150 degrees followed by 135 degrees and finally 120 degrees. This would give us 6 measurements for each axis at all 4 angles. To confirm the angle between the board and the camera we created a script using the formula from section 3.8.

when evaluating the results we saw some discrepancies in the trends of our average errors. We decided to do another series of tests. This time we would repeat the same experiment, but rotate the board around the y-axis (pitch) instead. This was to see if it would give us significantly different results. While doing this we realized the discrepancies between our initial test results, and the ones we have found recorded in other papers such as Pentenrieder et al. (2019) and López-Cerón and Cañas (2016), are likely due to inaccurate definitions of the robot TCP. We have explained this further in section 6.3.1. Due to this we redefined the TCP and redid the tests but this time using smaller increments in rotation to make the results more comparable to the ones found in the other papers.

4.8 Testing of Accuracy in Roll, Pitch and Yaw

To test the accuracy of the orientation estimation we had to be able to accurately define the TCP of the robot as the origin in our ArUco board. If this was not done, the orientation shown in the teach pendant would not match the orientation of the coordinate system defined by the board. We saw some of the problems that could occur by not doing this correctly in section 4.7. We used the same mounting of the board as used for the second sets of tests done in that section.

We would rotate the board around a single axis at a time in intervals of 5° from 0° to 40° and note the display value of all 3 axes.

4.9 Experimental implementation of stereo vision solution for 6DOF estimation

We hypothesized that by using stereo-vision techniques we would be able to increase accuracy in orientation tracking by identifying points on the model we were tracking as far apart as possible and calculating the orientation using the knowledge of how these points relate to each other. When the tracked points are further apart a small error in position between two points would not result in a large error in the angle between them. We also knew that some problems we had with compounding errors could be reduced by using stereo calibration to find two cameras position in relation to each other and a common reference point. To get a feel of how difficult a stereo vision implementation could be we decided to do a small scale test using some functionality in OpenCV we had come across during research.

One method for getting orientation using stereo-vision was by having 3 distinguishable points each pointing a known distance away from a common known origin in the direction of the principal axes as done with the axis cross in the MVP. This puts a lot of limitations

on the distance between the points and also adds non-negligible weight to the model that will have to be accounted for.

A different solution is to have two distinguishable points p_1 and p_2 along the first principal axis x where one of them is assigned as the origin and a third distinguishable point p_3 anywhere on the side of the positive second principal axis y on the plane created by the x and y-axes. Doing this we can find the principal axis z using the cross-product of the normalized vector created by the two points on the x-axis and the normalized vector created by the origin point on the x-axis and the third point.

$$\vec{x} = \overrightarrow{p_1 p_2} \tag{4.16}$$

$$\vec{z} = \overrightarrow{p_1 p_3} \times \vec{x} \tag{4.17}$$

We can then find the direction of the y axis by taking the cross product of the normalized vectors representing the x and z axes.

$$\vec{y} = \vec{x} \times \vec{z} \tag{4.18}$$

This solution is preferable if and only if the model has a flat surface plane that lies parallel to the plane created by the models x and y-axes. It makes it very simple to place two markers along the model x-axis and the third marker is very flexible letting us maximize distance. The weight added by ArUco markers on the flat surface is also considerably less than an axis cross. Because the ship models have a flat surface plane like mentioned this is what we felt would be the best solution.

Implementation

To find the 2D coordinates corresponding to each point we used the same method as the PnP ArUco marker solution as this was already implemented. We had 3 ArUco markers placed on the object and used the image coordinates corresponding to the point in the top left corner of each marker.

For image rectification and triangulation we used the OpenCV function **triangulatePoints()**. This function takes pairs of corresponding image points from two images together with the camera projection matrix for each camera and returns the 3D position of the points with regard to the world coordinate system used in the projection matrix.

Using the second solution outlined in the previous section we can then use these 3 points to calculate the orientation of the object. Together with the position of the point defined as origin we then have a 6DOF description of the object in 3d space.

Chapter 5

Results

5.1 Code

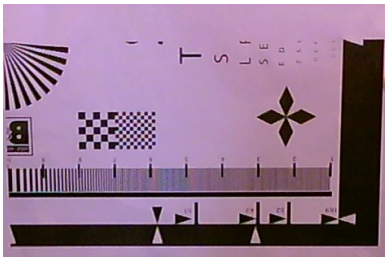
The complete code is published as open source code under the GNU General Public License V3.0 on Github, link in reference; Drugli et al. (2019).

5.2 User Manual

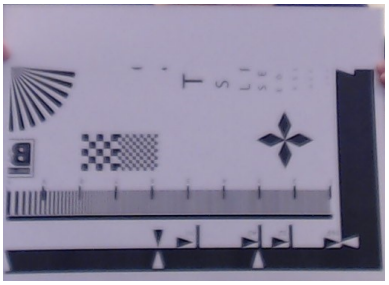
A user manual for the system can be found in Appendix A.

5.3 Result of Camera Sharpness Testing

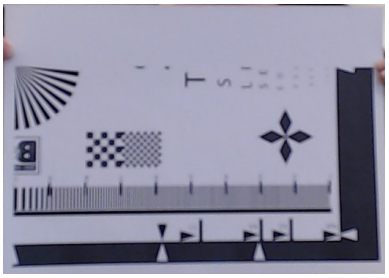
Testing of camera sharpness was done by taking a picture at a given distance of an object that shows the focus capacity of the camera. The more vertical lines that are clear, the higher the sharpness. Also, the images is taken under the same light conditions, to show if the white balance is off. See figure 5.1



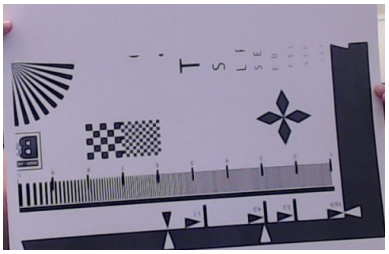
(a) Odroid Camera



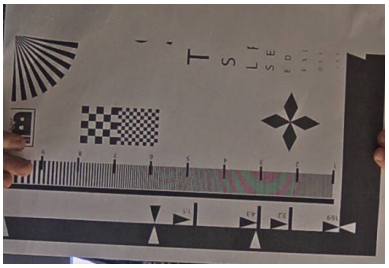
(b) C920 Unmodified



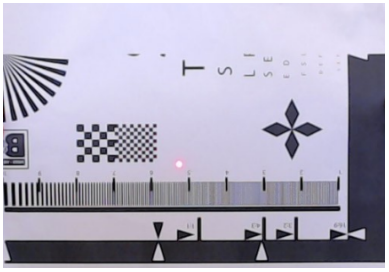
(c) C920 nr 1 modified



(d) C920 nr 2 modified



(e) Axis P3225-V MKII



(f) New C920 cameras unmodified

Figure 5.1: Results from Focus Test of Cameras

Figure 5.1a is the Odroid web camera. Figure 5.1b is the Logitech C920 web camera, before manually adjusting the focus of the lens. Figure 5.1c and 5.1d is two different c920 cameras with modified lens. Figure 5.1e is the IP camera. Figure 5.1f is the newly bought c920 web camera. This is also not modified.

5.4 Results of Position Accuracy Testing

The setup used for these tests is outlined in section 4.7. Each table represents tests done at a certain angle between the board of markers and the camera image plane. The tables are divided into three sections one section for movement along each axis separated by double lines. The values x-y-z are the reference values from the robot and the values $\hat{x}$-$\hat{y}$-$\hat{z}$ are the measured values from our camera system. Some values in the tables can be highlighted using bold text with an explanation given under the table. The measurements are in millimetres (mm).

5.4.1 Series 1:

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0	0.1	0.1	0	0.5	0.5	0.2	0.97	0.77
9.99	10.1	0.11	10	9.85	-0.15	10	9	-1
19.99	20	0.01	20	18.9	-1.1	19.98	21.5	1.52
30.01	30.5	0.49	30.1	28.7	-1.4	30.01	35	4.99
40.01	40.6	0.59	40	38.17	-1.83	40.03	45	4.97
50.01	50.7	0.69	49.99	48.07	-1.92	50	52.2	2.2

Table 5.1: Position accuracy testing series 1. Distance to camera: 100cm, Angle: 175°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.01	1	0.99	0.03	0.1	0.07	0.02	1	0.98
9.98	9.3	-0.68	10.01	9	-1.01	10	8	-2
20.02	19.5	-0.52	20.02	18.7	-1.32	20.01	23	2.99
30.01	29.7	-0.31	29.99	28.5	-1.49	30.01	32	1.99
39.98	39.9	-0.08	40.02	38.5	-1.52	40.02	43	2.98
50	49.9	-0.1	50.02	48.3	-1.72	49.99	53.5	3.51

Table 5.2: Position accuracy testing series 1. Distance to camera: 100cm, Angle: 161°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.01	2.5	2.49	0.01	0.3	0.29	0	4.5	4.5
10.1	7.5	-2.6	9.98	9.1	-0.88	9.98	10	0.02
19.98	17.8	-2.18	20	18.7	-1.3	19.99	23	3.01
29.99	28.3	-1.69	30.02	29	-1.02	30.02	31.5	1.48
40	38.3	-1.7	39.97	38.8	-1.17	40.01	44	3.99
50.01	48.5	-1.51	49.98	48.2	-1.78	50.01	51.5	1.49

Table 5.3: Position accuracy testing series 1. Distance to camera: 100cm, Angle: 132°

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0.01	3	2.99	0.03	0	-0.03	0.01	4	3.99
9.98	7.2	-2.78	10.01	7.2	-2.81	9.98	3.5	-6.48
20	17.2	-2.8	20.01	19.9	-0.11	19.98	25.5	5.52
29.99	27.1	-2.89	30.03	29.7	-0.33	29.99	28	-1.99
40.01	37.3	-2.71	39.99	39.3	-0.69	39.99	37.5	-2.49
50	47.8	-2.2	50.03	48.3	-1.73	50	46.5	-3.5

Table 5.4: Position accuracy testing series 1. Distance to camera: 100cm, Angle: 117°

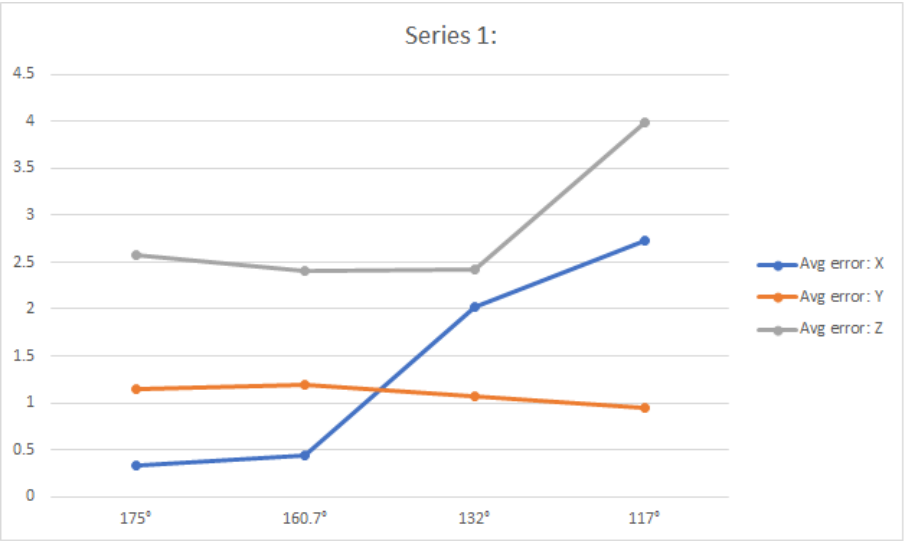


Figure 5.2: Average error comparison to angle of camera series 1

The graph shows the average error of each of the axes x-y-z at every recorded angle of the board. It is important to note the trend of the average errors where the average error of Y seems to be constant or decreasing as the angle changes. The error of X seems to have a sharp increase while the error of Z seems more like a convex curve.

5.4.2 Series 2:

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.01	0.1	0.09	0.02	0.3	0.28	0	0.5	0.5
9.99	9.7	-0.29	10.03	9.7	-0.33	10	9.5	-0.5
20	20.3	0.3	20.01	19.5	-0.51	20.01	22	1.99
30.01	30.6	0.59	29.97	29.1	-0.87	29.9	37.5	7.6
40.01	40.5	0.49	39.99	37.6	-2.39	40.02	40.25	0.23
50	50.4	0.4	50.02	47.9	-2.12	49.99	49.99	0

Table 5.5: Position accuracy testing series 2. Distance to camera: 100cm, Angle: 173°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.01	0.7	0.69	0.01	0.2	0.19	0.02	1	0.98
10.01	9.3	-0.71	10	10	0	10.02	7.2	-2.82
19.99	19.7	-0.29	20.01	19.4	-0.61	19.99	20	0.01
30.01	29.7	-0.31	29.99	29.4	-0.59	29.97	31	1.03
40.01	40.1	0.09	40	39.5	-0.5	40.02	40	-0.02
50.03	49.9	-0.13	50.02	49.2	-0.82	49.99	46	-3.99

Table 5.6: Position accuracy testing series 2. Distance to camera: 100cm, Angle: 161°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.01	2.5	2.49	0.02	0.3	0.28	0	-2	-2
10.02	7.8	-2.22	9.98	10.2	0.22	9.98	7.5	-2.48
19.98	17.9	-2.08	19.98	19.9	-0.08	20.01	21	0.99
30.02	28.7	-1.32	30	29.8	-0.2	30.01	29	-1.01
39.99	38.5	-1.49	39.98	39.8	-0.18	39.99	40.5	0.51
50.01	48.4	-1.61	50.01	49.7	-0.31	49.98	49.5	-0.48

Table 5.7: Position accuracy testing series 2. Distance to camera: 100cm, Angle: 133°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0	2	2	0.01	0.3	0.29	0	-2	-2
10.01	6.7	-3.31	10	9.8	-0.2	10.02	3.5	-6.52
20	17.2	-2.8	20.01	19.9	-0.11	20	20	0
29.98	27.1	-2.88	30.02	29.7	-0.32	29.87	26	-3.87
40	37.9	-2.1	39.99	39	-0.99	40	35	-5
50.02	47.8	-2.22	50.01	49.2	-0.81	49.84	50.5	0.66

Table 5.8: Position accuracy testing series 2. Distance to camera: 100cm, Angle: 117°

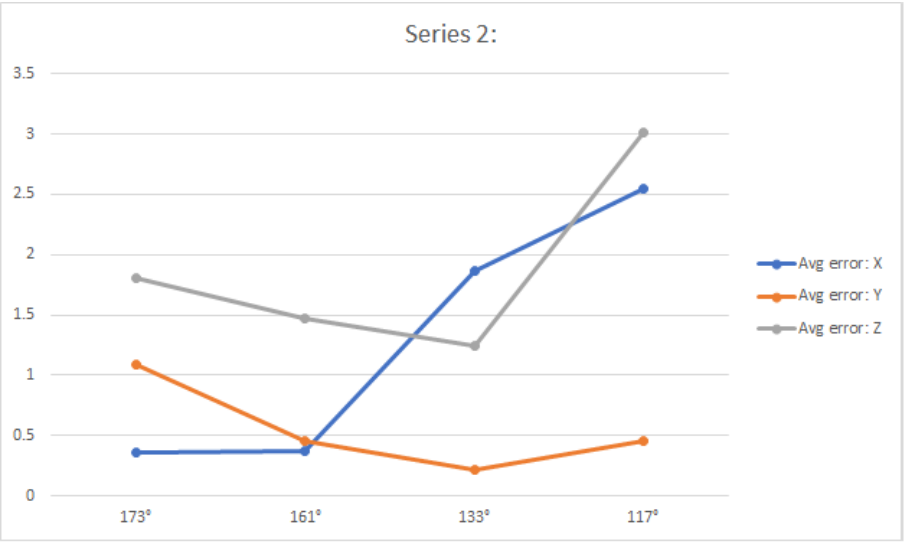


Figure 5.3: Average error comparison to angle of camera series 2

Again we see the errors following the same trends as in graph **??**, but this time with more of a decline in the average error of Y and the error of Z seems to be lower overall.

5.5 Results of Position Accuracy Testing after Redefined TCP

These tests are done after redefining the robot tool center point (TCP). The data follow the same structure as in section 5.4. The measurements are in millimetres (mm)

5.5.1 Series 3:

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0	0.16	**0.16**	0.03	0.18	**0.15**	0	-2	-2
10.01	10.2	**0.19**	10.02	9.9	**-0.12**	10.01	12.5	2.49
19.99	20.56	**0.57**	20.01	20.7	**0.69**	20.01	20.1	0.09
29.99	30.9	**0.91**	30	31.5	**1.5**	30	32.4	2.4
39.99	41.6	**1.61**	40	42.06	**2.06**	40	43.7	3.7
50.01	52.3	**2.29**	49.99	52.2	**2.21**	50.01	52.2	2.19

Table 5.9: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 176°

If you look at the highlighted column $\hat{x} - x$ you will see that the error is larger the further from origin you move. This happens for column $\hat{y} - y$ also and seems with few exceptions to hold true for all angles

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0	0.53	0.53	0.05	0.12	0.07	0	-3.8	-3.8
10	10.2	0.2	9.99	9.8	-0.19	10	7	-3
19.99	20.5	0.51	20.02	20.03	0.01	20.01	14.5	-5.51
30.01	31.5	1.49	30.02	30.7	0.68	30.01	26.4	-3.61
40.01	42.2	2.19	40.01	41.3	1.29	40.01	36.6	-3.41
50.05	52.3	2.25	50.01	51.7	1.69	50.01	49	-1.01

Table 5.10: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 166°

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0.02	-0.7	-0.72	0.01	-0.3	-0.31	0.01	-4.5	-4.51
10.01	10.18	0.17	10.02	9.7	-0.32	10.02	5	-5.02
20	21.2	1.2	19.97	20.5	0.53	20.01	15	-5.01
30.02	31.5	1.48	30.8	31.2	0.4	30.01	18.6	**-11.41**
40.01	42.3	2.29	40.02	41.6	1.58	40.01	34.4	-5.61
50	52.3	2.3	50.98	51.8	0.82	50.01	43.4	-6.61

Table 5.11: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 156°

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0.02	-0.3	-0.32	-0.06	-0.9	-0.84	0.01	-8	**-8.01**
9.98	10.3	0.32	10.03	9.8	-0.23	10.01	3.2	-6.81
20.01	20.7	0.69	20.05	20.9	0.85	20.03	15.1	-4.93
29.99	30.9	0.91	30	30.8	0.8	30.01	26.4	-3.61
40.01	41.7	1.69	40.01	41.9	1.89	40.01	31.6	**-8.41**
50	52.6	2.6	49.97	52.4	2.43	49.98	44.2	-5.78

Table 5.12: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 146°

x	x̂	x̂-x	y	ŷ	ŷ-y	z	ẑ	ẑ-z
0.03	-0.2	-0.23	0.02	-0.8	-0.82	0.01	-7.5	-7.51
10.01	9.9	-0.11	10.02	9.4	-0.62	10	4.5	-5.5
19.98	20.7	0.72	20.01	20.3	0.29	19.99	14.9	-5.09
30.01	31.4	1.39	30.01	30.9	0.89	30	20.9	-9.1
40	42.4	2.4	40.02	41.7	1.68	40.01	35.5	-4.51
49.99	52.1	2.11	49.99	52.3	2.31	50.01	45.7	-4.31

Table 5.13: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 136°

x	$\hat{x}$	$\hat{x}$-x	y	$\hat{y}$	$\hat{y}$-y	z	$\hat{z}$	$\hat{z}$-z
0.05	-0.7	-0.75	-0.02	-0.9	-0.88	0.01	-7.7	-7.71
9.97	10.1	0.13	9.99	9.7	-0.29	10	4.3	-5.7
20.02	20.7	0.68	20	20.6	0.6	20.01	13.1	-6.91
30.02	31.2	1.18	30.04	31.06	1.02	29.99	22.4	-7.59
39.99	41.2	1.21	39.99	41.7	1.71	40.02	32.6	-7.42
50.02	52.2	2.18	50.02	52.4	2.38	50.02	45.4	-4.62

Table 5.14: Position accuracy testing series 3. Distance to camera: 100cm, Angle: 126°

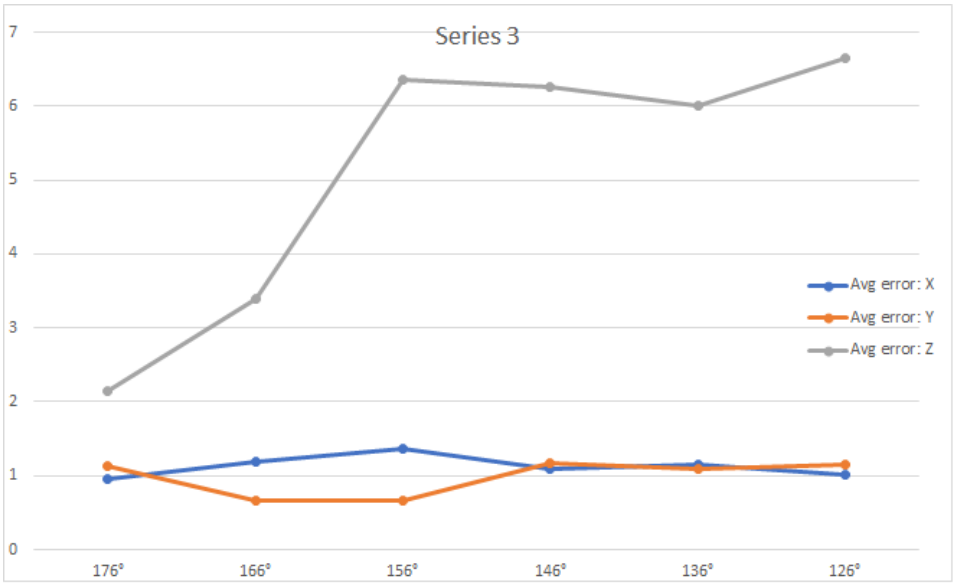


Figure 5.4: Average error comparison to angle of camera series 3

After the adjustment of the robot tool center point, we now see the average errors of X and Y following each other much more closely and being somewhat constant while the average error of Z has a sharp rise. There is a high average error in Z at 156° and 146° and we have highlighted some outlier values of $\hat{z} - z$ in tables 5.11 and 5.12 that contributes to this.

5.6 Results of Roll, Pitch and Yaw Accuracy Testing

ϕ	$\hat{\phi}$	$\hat{\phi}-\phi$	θ	$\hat{\theta}$	$\hat{\theta}-\theta$	ψ	$\hat{\psi}$	$\hat{\psi}-\psi$
0	0.1	0.1	0	0	0	0	0	0
5	5.2	0.2	0	0.6	0.6	0	0	0
10	9.8	-0.2	0	0.8	0.8	0	0.1	0.1
15	14.3	-0.7	0	0.8	0.8	0	0	0
20	18.7	-1.3	0	0.3	0.3	0	0.1	0.1
25	23.6	-1.4	0	0.8	0.8	0	0.3	0.3
30	28.7	-1.3	0	0.2	0.2	0	0.1	0.1
35	34.1	-0.9	0	0.5	0.5	0	0.3	0.3
40	38.5	-1.5	0	0.5	0.5	0	0.3	0.3
0	1	1	**5**	3.9	-1.1	0	0.1	0.1
0	0.5	0.5	**10**	7	-3	0	0.1	0.1
0	1.7	1.7	**15**	13.1	-1.9	0	0.5	0.5
0	0.7	0.7	**20**	18.7	-1.3	0	0.4	0.4
0	1.2	1.2	**25**	23.1	-1.9	0	0.8	0.8
0	1.1	1.1	**30**	28.5	-1.5	0	1	1
0	2	2	**35**	34.2	-0.8	0	1.5	1.5
0	1.3	1.3	**40**	38.4	-1.6	0	1.8	1.8
0	0.2	0.2	0	-2.2	-2.2	**5**	5	0
0	0.2	0.2	0	-1.9	-1.9	**10**	10	0
0	0.5	0.5	0	-1.2	-1.2	**15**	14.9	-0.1
0	0.5	0.5	0	-1.3	-1.3	**20**	19.9	-0.1
0	0.7	0.7	0	-1.6	-1.6	**25**	24.7	-0.3
0	-1.3	-1.3	0	-0.5	-0.5	**30**	29.8	-0.2
0	-1.9	-1.9	0	0.2	0.2	**35**	34.8	-0.2
0	0.2	0.2	0	0.5	0.5	**40**	40.1	0.1

Table 5.15: Roll, Pitch and Yaw accuracy testing series 1. Distance to camera: 100cm

How this test was performed is outlined in section 4.8. It is important to note that the measurements are taken with the z-axis pointing perpendicular to the board directly towards the camera when it is placed in origin. This will cause the angle between the camera and the markers to change when the angles of roll ϕ and pitch θ are changed, but not when we change the angle of yaw ψ. This means how we have defined the world coordinate system with regard to the camera will likely affect the accuracy of the system and we should not read too much into how accurate roll and pitch is compared to yaw. The values from our camera system are denoted $\hat{\phi}-\hat{\theta}-\hat{\psi}$ and the values from the robot $\phi-\theta-\psi$. The measurements are in degrees (°).

Chapter 6

Discussion

6.1 Minimum Viable Product

The goal of the Minimum Viable Product (MVP) was to create a system that could tell us whether a solution with axis cross was an idea to work on further. At this point, we wanted to prioritize creating a working tracking algorithm, and wait for later versions to implement more complex functionality that could eventually be discarded because of the bad results of another subsystem.

Spherical markers mounted in a cross configuration (figure 6.1a), was our first attempt to create a model based pose estimation tracking system. When deployed in a carefully controlled environment, we were able to obtain accurate data, but the system proved to be unstable and unreliable. Some of the instability was due to the axis cross spheres overlapping each other for the camera or they were indistinguishable from the background. This could have been solved with more markers and better image processing algorithms. Since we did not have a clear way to distinguish each marker from each other, the PnP-algorithm stopped working when the markers were read in the wrong order.

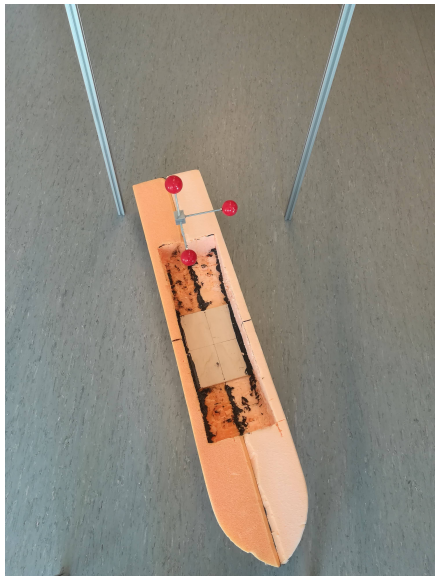
These problems were most likely solvable, and we could have improved this system to be usable. But this was discarded since early testing with ArUco markers gave good results.

The test rig proved to be useful since our first implementation of markers - an axis cross made with steel, proved to be too heavy to use on the ship models. It could affect the buoyancy and movement of the ship model.

The text user interface we created was adequate for its use. It let us control our system, and its quick implementation gave us more time working with other things.

The ArUco solution (figure 6.1b), was the last thing we implemented in the MVP since we were satisfied with the performance of the algorithm. We also considered the weight of the paper ArUco markers to be light enough to not impact the vessel models performance.

Summarized, the MVP provided an environment for experimentation, but also worked as a stepping stone for further improvements of the system with more functionality and more streamlined user experience.



(a) Axis cross marker

(b) ArUco markers

Figure 6.1: Markers mounted on vessel

6.2 Sharpness Testing of Cameras

From the result from sharpness testing in section 5.1 we can see that adjusting the lens of the C920 camera gave a greater focus distance. But after we bought new C920 cameras, it became clear to us that the old C920 were not functioning as well as they should, even after the lens adjustment. You can see the difference between the old and new cameras in figures 5.1c, 5.1d and 5.1f. This can be the result of many things. These cameras have been used before by other students and have probably not been taken good care of.

The Odroid camera has a narrower field of view than the C920s. When taking two snapshots from the same distance, the target in the Odroid frame takes up more of the image frame, and this results in an advantage for the Odroid when comparing sharpness. This means that we should not compare the Odroid image directly to the c920 in figure 5.1a.

The Axis camera had a variable zoom function, but we did not adjust the field of view to match the other cameras. Because of this, we do not compare the sharpness test results between the different types of cameras.

6.3 ArUco Marker Solution

6.3.1 Accuracy of Estimated Pose

After comparing the results in section 5.4 and 5.5 we concluded that we should disregard the results in section 5.4 as we don't think they are valid and instead use only the results in section 5.5.

The reason for this is that we found some discrepancies when comparing the result of the average error in test series 1 in figure 5.2 to the results in section 5.2 of Pentenrieder et al. (2019). There the change in error as the angle between the marker and camera increases is not only very low but also the same for x and y while the error in depth z increases sharply as the angle increases. In our results, x and y don't seem to follow each other at all and the average depth error z looks more like a convex curve than a linear increase. We did another series of tests and it did not seem like the discrepancies in series 1 was a one-off occurrence as can be seen in figure 5.3.

We had a few different ideas about what could do this. The most likely one seemed for us to be that the robot tool center point (TCP) was defined wrong or the robot was poorly calibrated. When we rotated about what we thought would be the origin of the board we instead rotated about a different point close to it and this would create a discrepancy in the values shown in the robot teach pendant and the values from our camera system. To verify this we tried to change the board pitch instead of the roll when increasing the angle between the board and the camera view because if our theory was correct we would see a different trend in the errors. While doing this we quickly realized our theory was correct and remade the board mounting for the robot tool and swapped to a different UR3 robot before redoing the tests. You can see the results of these new tests in figure 5.4 in section 5.5. These results more closely resemble the results in Pentenrieder et al. (2019).

The goals for accuracy we set ourselves of $\pm$10mm in position $(x\ y\ z)$ and $\pm 2^\circ$ in rotation $(\phi\ \theta\ \psi)$ have been achieved when looking at the results in section 5.5 for position and section 5.6 for rotation. Both of these tests have been done at a distance of 100cm. Assuming the results in Pentenrieder et al. (2019) and López-Cerón and Cañas (2016) regarding accuracy getting worse with increased distance to the markers holds true for us also it is hard to make a definite claim regarding accuracy in the actual installation. Our system is designed to be very flexible in how the cameras are set up so the distance to the tracked object can vary. Accuracy at different distances is also very much impacted by the camera focus adjustment. We have not performed a test to see how distance in excess of 100cm impacts the accuracy.

The tests done here also don't take into account the decrease in accuracy that can happen when we use multi-camera tracking. This is covered more in section 6.3.4.

6.3.2 Refresh rate

From our log files, we found that the system could find a new pose for the board faster than the cameras could provide us with new frames. Because of this we decided to limit the refresh rate to 20Hz, which should be adequate for most applications.

Since the system provided us with up to a 60Hz refresh rate before it was limited, we reason that we could add even more cameras or use images with a higher resolution and still maintain an adequate refresh rate.

6.3.3 ArUco Merger

The ArUco merger covered in section 4.4.1 did not give us satisfying results and we recommend not using it without first making improvements. The pose of the subsequent boards in relation to the primary board is not accurate enough and will cause large errors in pose estimation of the vessel. One way to solve this would be to physically measure the location of the markers in relation to each other but this would go against having a simple user-friendly solution.

6.3.4 Multi Camera Tracking and Pose Quality

The tracking of the vessel using multiple cameras is working as intended. There are still improvements to be made when it comes to the handover between the different cameras. Choosing what camera that would probably give us the most accurate pose estimation when multiple cameras can see the vessel at the same time is one of these. A solution for this was started in section 4.4.2 but not completed. A different issue we had was that even small errors in the orientation of a camera lead to growing errors in the vessels location. This relationship is explained in figure 6.2, and in equation 6.1.

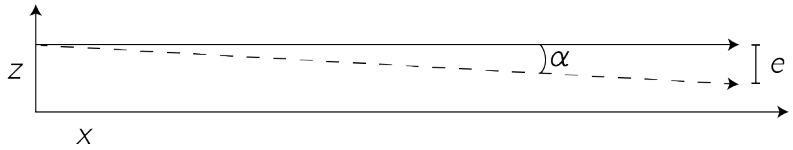


Figure 6.2: Errors from camera orientation estimation compounds over the distance travelled. Solid line: Actual position. Dashed line: Estimated position. α: Error in camera orientation. e: Error in vessels location.

$$\hat{z} - z = x \sin \alpha \tag{6.1}$$

The error in the z-position increases as the vessel moves in the x-direction, where the slope of this error is given by the sine of the error of the angle. To solve this problem we need to be able to more accurately define the position of the cameras.

6.4 Implementation in tank

The physical implementation of the system provides coverage over most of the tank. The cameras are mounted in such a way that they are both easily accessible for modifications or maintenance. Since they are attached to a rail in the ceiling, they have the added benefit

of not being in the way when adding or removing vessels for testing. This is an important point, as the towing tank also is used for larger vessels such as ROVs.

The current implementation can be improved by adding better cable management for the cameras, since we don't consider the current set up to satisfy the needs for a permanent solution.

We have five cameras installed, but we have not been able to use more than three of them at the same time with our personal computers, due to a limitation of one camera per USB port on the computer. This can be solved by using a computer with more USB-ports. This was apparently not an unknown problem, but we could not find anyone who was successful in circumventing this with USB-webcams. We believe the camera failure is rooted in a limitation in the USB-interface, and that we can avoid the problem by using IP-cameras instead. These cameras are running as stand-alone entities on the network, and should always be able to provide a frame on request.

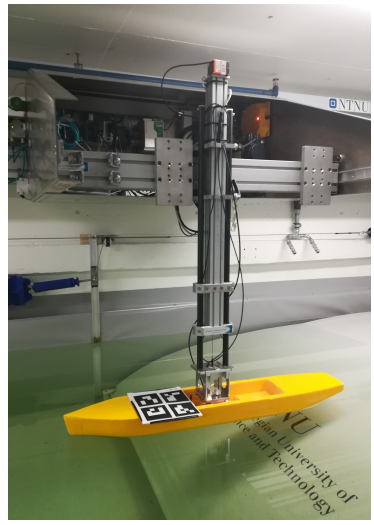
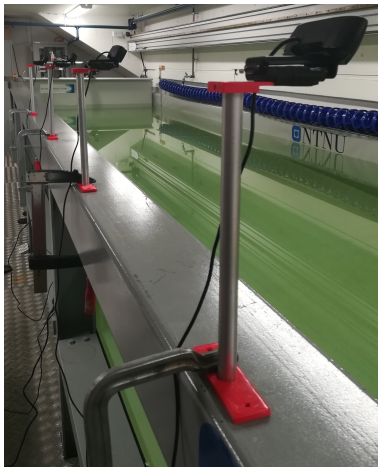


Figure 6.3: Photo from towing tank: Vessel with ArUco marker attached to trolley



(a) Initial: On side of the Tank

(b) Final: On the track in roof

Figure 6.4: Camera mountings in tank

To improve the system, it can be better to create the 3D printed parts in stainless steel or steel with painting on to stop corrosion. This will add weight to the parts but it prevents water damage caused by a highly humid environment.

6.4.1 Software and GUI

The current GUI provides an easy to use interface that should be usable for anyone, even without technical knowledge of vision systems. We have put effort into preventing the system from crashing from obvious user errors, and have provided in depth help for the users when doing more complex tasks such as camera calibration.

The system is not completely self explanatory, but coupled with the appended user manual the system is ready to be released for anyone to use.

The current state of the code is that most of it is thoroughly commented, and we have had readable variable and function names in mind while coding. The code is released under a open source license, which means that anyone can further develop our system.

We do have some complexity creep in our GUI-classes which should have been refactored, but we assess the back end of our system to be most interesting for future developers.

Chapter 7

Conclusion

Our objective was to develop a vision system that can detect and track vessel position and orientation in the Department of Ocean Operations and Civil Engineering's towing tank. We considered accuracy of ± 10mm in position and $\pm 2^{\circ}$ in orientation or better to be achievable. We should create an easy to use robust system that would give the results of the pose estimation and video feed in real-time and provide the same amount or more data than the existing system. If time permitted we should also create a tool for wave analysis.

We have developed a vision system that tracks an object in six degrees of freedom and implemented this system in the Department of Ocean Operations and Civil Engineering's towing tank. The system has adequate accuracy in the initial camera views but suffers from a compounding error when calculating the absolute position of the vessel after switching camera frames. Despite this, the data the system acquires should still be useful to see how a vessel behaves in the tank.

The developed application can be refitted for many purposes. The system is designed to easily be taken down and re-deployed somewhere else since it automatically calibrates the camera positions with regard to the object for each individual run. As long as one can provide camera coverage of the area where the object should be tracked, and there are possibilities to attach a marker on the object, one should be able to get a live pose estimation of the tracked object.

We have implemented a user-friendly GUI with accompanying user manual. The system should be usable by anyone and requires no prior knowledge of vision systems.

7.1 Recommendations

The camera setup itself needs improvement. With the solution of having the cameras in the ceiling above the tank will need an addition to the track that is already there. This is to be able to cover the entire operational length of the towing tank. Also the stand itself needs better locking of parts so that the risk of it falling down after a long time usage and adjustments diminishes. Now the parts have the possibility to be screwed together, but it is

not implemented. Now it is only using a interference fit between the rod and the brackets. This can over time loosen if the part is moved and should be secured.

7.2 Further Work

7.2.1 Improving Camera Position to Increase Accuracy

The camera pose accuracy is critical to avoid compounding errors in the vessels absolute location. If a cameras orientation is off by a few degrees, the error in the vessels position will increase with the sine of this angles error over the distance traveled.

Stereo Vision

OpenCV has a calibration function for finding the relative poses between two cameras using a set of common points in each image. These common points could be the corner points of the board, and an amendment to our current code to test this could be done with a only a small amount of coding. We believe that this function could give more accurate poses for the subsequent cameras than the algorithm that is currently implemented.

Implementing Additional Markers as Reference Points

By adding markers to known locations in the tank, it would be possible to calibrate the camera poses directly from these markers. By utilizing these fixed points, it would be possible to eliminate the compounding errors we get by doing subsequent measurements of the vessel, and the error of each cameras pose would only be reliant on it's own measurement rather than every measurement done before it.

Calibration Using Towing

A method that is limited to calibration in the tank, but would result in a coordinate system that is in line with the water surface, is to perform a calibration run, towing the vessel through the tank, and calibrating from this data. This method would use the nature of growing errors in of position as a result of small errors in orientation (see figure 6.2 and equation 6.1) as a mean to calibrate each camera pose.

When towing, the vessels y-axis is locked, the x-position is always increasing, while the z-position is oscillating in the waves. When the vessel has reached its set speed, the speed in the x-direction should stay constant. Given these assumptions, it should be possible to calculate the orientation of each camera relative to the path the vessel is traveling along.

The towing tank also allows for setting specific start and end points for the tow, and these positions can be used as a reference when estimating camera position. We see a clear potential for using this as a way forwards, but we have not yet thought up a clear implementation of the solution.

7.2.2 Integration with Towing Tank

By letting the system control the towing trolley the calibration function could be tuned even more precisely by using the trolley position as a controllable reference.

7.2.3 Saving and exporting video files

Coupling the numerical data with a video of the vessel in the tank could possibly put the data in context. Reviewing videos of different vessels could grant useful insights when reviewing different hull designs. It would be especially useful for reviewers who did not see the vessel in the water.

7.2.4 DP-simulation

Since the system can output a vessels current pose over six degrees of freedom, it could be used as the sensor in a closed loop system for dynamic positioning simulations.

7.2.5 3D-simulation with ArUco-markers

Simulating a 3D-space with multiple cameras and a movable ArUco-board, it would be possible to test our algorithms under ideal conditions. This would be valuable when comparing tracking and camera handover algorithms, as it would present an environment absent of outside factors that could impact the pose estimation for the algorithms to be tested in. It would also be a lot easier to set up experiments in a system like this than it has been for us using the UR3 robots and other physical setups.

7.2.6 Using the System in Other Environments

Since the system has been built to be set up anywhere, it is not limited to use in the towing tank. It could be used for tracking other vehicles, like cars or drones, or even robot arms.

Bibliography

Adams, R. A., Essex, C., 2014. Calculus, a complete course, 8th Edition. Pearson Canada Inc, ISBN: 9780321781079.

Bradski, G., Kaehler, A., 2013. Learning OpenCV: Computer Vision in C++ with the OpenCV Library, 2nd Edition. O'Reilly Media, Inc.

Byju's, T. L. A., 2019. Calculation of angle between two planes.
URL https://byjus.com/maths/angle-between-two-planes/

Craig, J. J., 2005. Introduction to Robotics, Mechanics and Control, 3rd Edition. Pearson Prentice Hall, ISBN: 0131236296.

Drugli, E., Fjørtoft, V., Hagseth, K., Sande, O. K., 2019. Source code of project: Relative motion tracking of vessels using multi-camera handover and aruco markers. E-mail: kai.hagseth@gmail.com.
URL https://github.com/kaihagseth/arucoTracker

Fischler, M. A., Bolles, R. C., 1981. Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography. Communications of the ACM.

Friel, J. J., 2000. Practical Guide to Image Analysis. ASM International.

Fusiello, A., Trucco, E., Verri, A., Jul 2000. A compact algorithm for rectification of stereo pairs. Machine Vision and Applications 12 (1), 16–22.

Garrido-Jurado, S., Munoz-Salinas, R., Madrid-Cuevas, F., Medina-Carnicer, R., 2016. Generation of fiducial marker dictionaries using mixed integer linear programming. Pattern Recognition 51, 481–491.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M., 06 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition 47, 2280–2292.

Hartley, R. I., Zisserman, A., 2004. Multiple View Geometry in Computer Vision, 2nd Edition. Cambridge University Press, ISBN: 0521540518.

Hoff, W., 2014. Eeng 512 - computer vision, university lecture Colorado School of Mines. URL https://www.youtube.com/playlist?list=PL4B3F8D4A5CAD8DA3

Hu, Z., Wu, F., 2002. A note on the number of solutions of the noncoplanar p4p problem. IEEE Transactions on Pattern Analysis and Machine Intelligence.

López-Cerón, A., Cañas, J. M., 2016. Accuracy analysis of marker-based 3d visual localization.

Lucchese, L., 2005. Closed-form pose estimation from metric rectification of coplanar points. IEE Proceedings.

MacAusland, R., 2014. The moore-penrose inverse and least squares, note from University of Puget Sound, MATH 420: Advanced Topics in Linear Algebra.

Marquardt, D. W., 1963. A note on the number of solutions of the noncoplanar p4p problem. Journal of the Society for Industrial and Applied Mathematics.

Martin, R. C., 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

Moore, E., 1920. On the reciprocal of the general algebraic matrix. Bulletin of the American Mathematical Society 26, 394–395.

Mussabayev, R., Kalimoldayev, M., Amirgaliyev, Y., Tairova, A., Mussabayev, T., 2018. Calculation of 3d coordinates of a point on the basis of a stereoscopic system. De Gruyter.

Pentenrieder, K., Meier, P., Klinker, G., 2019. Analysis of tracking accuracy for single-camera square-marker-based tracking.

Quan, L., Lan, Z., 1999. Linear n $\geq$ 4-point pose determination. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Romero-Ramirez, F., Muñoz-Salinas, R., Medina-Carnicer, R., 06 2018. Speeded up detection of squared fiducial markers. Image and Vision Computing 76.

Simek, K., 2013. Dissecting the camera matrix, part 3: The intrinsic matrix. URL http://ksimek.github.io/2013/08/13/intrinsic/

Slabaugh, G. G., 1999. Computing euler angles from a rotation matrix. URL http://www.gregslabaugh.net/publications/euler.pdf

Suzuki, S., Abe, K., 1985. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing 30 (1), 32 – 46.

Wikimedia, 2010. Hsv color solid cylinder saturation gray.png. URL https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png

Appendix

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Estimering av modellposisjon og bølgebevegelse i bølgetank

KANDIDATNUMMER(E):			
DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
	IE303612	**Bacheloroppgave**	- Åpen
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
AUTOMATISERINGSTEKNIKK		/	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):

Oppdrag gitt av NTNU Institutt for havromsoperasjoner og byggteknikk/Institutt for IKT og realfag
Veiledere: Ottar L. Osen og Robin Bye

OPPGAVE/SAMMENDRAG:

Ved forsøk med skipsmodeller og andre modeller vanntank, er det ønskelig å kunne hente ut data om modellens posisjon og orientering, samt beregne bølgeform og -høyde i tanken.

Oppgaven går ut på å finne løsninger for dette. Det er planlagt å bruke flere kamera til å løse oppgaven, hvor punkter på fartøyet sammenlignes fra flere vinkler for å få en 3D representasjon av fartøyet/objektet.

Prosjektet er en avsluttende bacheloroppgave i Automatiseringsteknikk, gitt av NTNU IHB og IIR.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Ved forsøk med skipsmodeller og andre modeller vanntank, er det ønskelig å kunne hente ut data om modellens posisjon og orientering, samt beregne bølgeform og -høyde i tanken.

Oppgaven går ut på å finne løsninger for dette. Det er planlagt å bruke flere kamera til å løse oppgaven, hvor punkter på fartøyet sammenlignes fra flere vinkler for å få en 3D representasjon av fartøyet/objektet.

Prosjektet er en avsluttende bacheloroppgave i Automatiseringsteknikk, gitt av NTNU IHB og IIR.

2 BEGREPER

IHB - Institutt for havromsoperasjoner og byggteknikk

IIR - Institutt for IKT og realfag

Marker based motion Capture - Teknikk for å fange bevegelse i et objekt ved hjelp av flere kameraer som logger referansepunkter.

Model based pose estimation - Teknikk for logging av posisjon og rotasjon av objekt ved hjelp av ett kamera og en fysisk modell med kjente dimensjoner.

MVP - "Minimum Viable Produkt", enkleste brukbare produkt

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)	
Even Drugli	**476122**
Vegard Fjørtoft	**460011**
Kai Hagseth	**997480**
Ole Kristian Sande	**476134**

3.1.1 Oppgaver for prosjektgruppen - organisering

Gruppen vil bestå av en prosjektleder, sekretær, kodeansvarlig og en ansvarlig for hardware og innkjøp. Når en deloppgave blir tildelt vil det være en person som som er ansvarlig og en person som er hjelper.

3.1.2 Oppgaver for prosjektleder

- Skal være kontaktperson for oppdragsgiver og veiledere.
- Skal holde oversikt over fremdrift og utfordringer i prosjektet, og rapportere disse til veiledere og evt. oppdragsgiver når det sees nødvendig eller hensiktsmessig.
- Skal sørge for at alle i gruppen har oppgaver til enhver tid.
- Sørge for god trivsel i gruppa.

3.1.3 Oppgaver for sekretær

- Skal ha hovedansvar for at rapport blir skrevet og at denne blir av høy kvalitet.
- Ansvar for utarbeiding av møtereferat og fremdriftsrapporter.
- Ansvar for dokumentasjon av systemet.

3.1.4 Oppgaver for kodeansvarlig

- Ansvarlig for den underliggende arkitekturen i programvareutviklingen
- Har ansvar for god kodestruktur
- Skal holde oversikt over hva som må gjøres i kode
- Har ansvar for kompatibilitet og ryddig versjonskontroll
- Ansvar for dokumentasjon av kode (gode kommentarer i kode og i sourcetree).

3.1.5 Oppgaver for Hardware og Innkjøpsansvarlig

- Ansvar for innkjøp av nytt utstyr
- Budsjettansvarlig
- Ansvar for fremgang i implementering av fysisk utstyr

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Veiledere: Ottar L. Osen og Robin Bye
Oppdragsgiver IHB: Karl Henning Halse

4 AVTALER

4.1 Avtale med oppdragsgiver

Oppdragsgiver er Karl Henning Halse fra NTNU IHB. Det foreligger ingen skriftlig avtale mellom studenter og oppdragsgiver per nå.

4.2 Arbeidssted og ressurser

Fast arbeidssted blir på campus Ålesund, fortrinnsvis L167 eller Tunglab (L044) på NTNU.

Møter med oppdragsgiver/veiledere vil skje på NTNU.

Oppdateringsmøte med veiledere vil skje hver andre uke. Disse møtene blir berammet til 30 minutter. Trengs det mer tid ifm. tekniske spørsmål eller lignende skal dette avtales før møtet. Oppdragsgiver kan delta om det sees behov. Gruppen skal lage ukesrapport siste arbeidsdag i uken.

Prosjektet inneholder ingen informasjon som er taushetsbelagt. Derfor er det ikke et krav om at prosjektet skal være skjult fra offentligheten.

4.3 Gruppenormer – samarbeidsregler – holdninger

Gruppen er blitt enig om:

- Alle i gruppen skal møte opp på avtalt klasserom alle hverdager innen klokken 09:00.
- Helgearbeid avtales fortløpende, minst 24 timer på forhånd.
- Dagen avsluttes på eget skjønn, men det forventes at ingen går før klokken 16:00 dersom man ikke har god grunn og koordinerer med resten av gruppa.
- Avspasering skal avklares fortrinnsvis minimum 1 uke på forhånd. Sykdom e.l. skal informeres om så snart som mulig.
- God kommunikasjon gruppen imellom når det kommer til fremgang i forhold til tidsfrister på delmål slik at gruppen kan diskutere eventuelle utsettelser eller tildeling av ekstra arbeidskraft.
- Deloppgave ansvarlig skal levere inn en oppsummering av ukas fremgang (hva som har blitt gjort, eventuelle problemer og hva som skal gjøres neste uke) til sekretær innen klokka 11:00 hver fredag slik at en rapport kan utformes før gruppemøte klokka 13:00

Prosjektet skal gjenspeile et arbeid som er gjort av automasjonsingeniører. Det vil si at arbeidet skal ha høy kvalitet. Det betyr god kildekode, gode systemmanualer og at alle opptrer profesjonelt. Det betyr at enhver står ansvarlig for å levere innenfor frister, med en god kvalitet som en selv kan være stolt av.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Problemstilling:

- Er det mulig gjennom et kamerasystem å registrere et skipsmodells posisjon og retning gjennom kamerasyn?
- Er det mulig å innhente data om bølgehøyde og -bevegelser gjennom samme kamerasystem, eller tilhørende system?
- Kan dette presenteres på en måte som gjør det aktivt nyttig i testing og forskning av skips- og havmodeller i bølgevanntank?

Et godt resultatmål vil være å få et ja på problemstillingene ovenfor, eventuelt få et godt begrunnet nei. I tillegg har man prosess- og effektmål. Hva ønskes å oppnås som følge av jobbingen under prosessen og hva er det langsiktige målet hvor prosjektet kun er et delmål. Et stort prosessmål er å kunne bruke dette prosjektet som en overgang fra student til arbeidsliv. Stor del av arbeidslivet er prosjektbasert, og derfor kan et slikt prosjekt sees på en prøvelse før det virkelig arbeidsliv, hvor mye mer står på spill tidsmessig og økonomisk.
Et faglig stort prosessmål er å kunne utvide kompetanse, særlig innen vision, og få en effekt av dette i et fremtidig arbeidsliv.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Prosjektet sikter på høyest mulig presisjon i målinger. Et matematisk begrunnet anslag for nøyaktigheten
Et mål er å få pose-estimasjon på en nøyaktighet på +-1 mm. Hva dette utgjør i rotasjon kommer an på dimensjonene i skipet og utstyret.

Akser:	Maks avvik
Posisjon	+- 1 mm
Rotasjon	+- 0.5 grader

Prosjektresultatet skal være en ferdig oppkoblet løsning ved vanntanken som kan brukes av andre studenter for datainnsamling. Det skal være utarbeidet manual for styring og bruk av systemet.

5.3 Planlagt framgangsmåter for utviklingsarbeidet - metoder

Gruppen har blitt enig om at vi skal bruke en iterativ og inkrementell modell for systemutviklingen. Fokuset for denne modellen er å utvikle systemet gjennom gjentakende faser, (iterativ), og på den måten kunne justere systemet etter hvert som vi legger til ny funksjonalitet(inkrementell).

For å komme i gang med denne utviklingsmodellen er det gitt at man har en plan for hvilken funksjonalitet man ønsker at systemet skal ha når prosjektet er ferdig, hvilken funksjonalitet man trenger for å lage en fungerende prototype, og hvordan man kan implementere fungerende iterasjoner mellom disse.

Etter at prosjektplanen er lagt starter den initielle fasen av utviklingen hvor en enkel modell av systemet utvikles, testes, og evalueres. Denne fasen etterfølges av repeterende iterasjoner hvor mer funksjonalitet legges til, og hvor eksisterende funksjonalitet forbedres og refaktoreres.

Fordelen med en iterativ og inkrementell prosess at man får et fungerende system på et tidlig stadium, og at man hele tiden har et klart scope for hva som skal gjøres i prosjektet. I evalueringsfasen mellom iterasjonene kan man finne feil og mangler ved arbeidet man har gjort, og kan reagere og reparere kort tid etter.

Når det kommer til ulemper er det lett å få problemer med systemarkitekturen dersom man gjør feilaktige antakelser om fremtiden. Det vil si at hvis systemet bygges på grunnlag av at neste iterasjon skal kunne fungere tilfredsstillende, uten at man tar hensyn lenger frem i tid, vil man få problemer. Dette kan forebygges gjennom god planlegging, og at man tar seg tid til å snakke sammen om hvordan iterasjonen vi begynner på er fremtidssikret i planleggingsfasene.

5.4 Informasjonsinnsamling – utført og planlagt

Det skal utredes hvilke metoder som gir best presisjon og nytte ved plassering av datapunkt på modellen. Vi har drøftet tre metoder.

Alternativ 1: Markørbasert posisjonssporing. Dette er en flerkameraløsning som gir oss stor frihet i hvordan markørene plasseres, og hvor mange markører vi ønsker å bruke. Etter en kalibrering av alle kameraene kan et markørpunkts posisjon estimeres i tre dimensjoner ved hjelp av triangulering. Dersom man har minst tre punkter i bildet og antar at objektet er rigid kan man estimere alle de seks frihetsaksene til objektet. Denne metoden bør i teorien også kunne estimere fordreining i objektet hvis man bruker mer enn tre punkter per objekt. Ved å bruke mer enn to kameraer kan man oppnå høyere presisjon, dekningsområde og redundans, på bekostning av tregere prosessering og høyere materialkostnader. For å øke ytelsen i denne metoden er det mulig å bruke IR-dioder som markørpunkt, og filtrere ut lys som ikke er infrarødt. Da vil man kunne finne alle punktene i bildet med en ressursvennlig terskel-operasjon. I våre undersøkelser har vi ikke funnet noen implementeringer av denne metoden for vår use-case, og løsninger med åpen kildekode er vanskelig å komme over. Dette kan bety at implementeringen er ressurskrevende, eller at dette ikke er en hensiktsmessig metode for vårt scenario.

Alternativ 2: Modellbasert positurestimering. Dette alternativets største fordel er at man kan lese av informasjon om alle de seks frihetsaksene til objektet med kun ett kamera, og at det ser ut til å ha mange implementasjoner på lignende caser som vår egen fra før. Det går ut på at det plasseres et objekt med kjente dimensjoner, for eksempel en figur med tre linjer som peker i x, y og z-retning med en kule på enden av hver linje. Ut fra denne modellen er det mulig å både kalibrere kamera og spore objektet. Denne metoden har tidligere blitt delvis utredet her på instituttet av Ø. Gjelseth og I I. Flatval så i oppgaven "Posisjons- og avstandsmåling med ett enkelt kamera for maritime løfteoperasjoner" (2015). Vi ser for oss at denne metoden kan utvides med flere kamera for å kunne oppnå bedre presisjon og større dekningsområde. Ved bruk av flere modeller bør vil også kunne øke presisjonen orientering om de pares, og man åpner mulighet for flerobjektssporing. Vi ser for oss at også denne metoden kan pares med IR-dioder for bedre ytelse, hvor endepunktene på figuren vår byttes ut med dioder.

Alternativ 3: Stereosyn for skroggjenkjenning. Her er fordelen at man i teorien ikke ville trenge markører av noe slag for å estimere de seks frihetsaksene. Gjennom en utvidelse av stereo-syn algoritmer kan man lage en 3d-modell av skroget, og beregne modellens positur ut fra et satt utgangspunkt. Dette alternativet er det mest ressurskrevende, og vil neppe fungere i sanntid. I tillegg fungerer ikke disse algoritmene med refleksjon og refraksjon, noe som ikke er ideelt når vi jobber i og rundt vann. Vi tror ikke dette er en god løsning for vår case.

Litteratur:
Multiple View Geometry in Computer Vision, *R. Hartley, A. Zisserman, Mars 2004, ISBN: 9780521540513*
Programming Computer Vision, *Jan Erik Solem, Juni 2012 ISBN: 9781449316549*

5.5 Vurdering – analyse av risiko

Matrise for risikovurderinger ved NTNU

KONSEKVENS	Svært Alvorlig	E1	E2	E3	E4	E5
	Alvorlig	D1	D2	D3	D4	D5
	Moderat	C1	C2	C3	C4	C5
	Liten	B1	B2	B3	B4	B5
	Svært liten	A1	A2	A3	A4	A5
		Svært liten	Liten	Middels	Stor	Svært stor
		SANNSYNLIGHET				

Farge		Beskrivelse
Rød		Uakseptabel risiko. Tiltak skal gjennomføres for å redusere risikoen
Gul		Vurderingsområde. Tiltak skal vurderes
Grønn		Akseptabel risiko. Tiltak kan vurderes ut fra andre hensyn.

Vi anser prosjektet som vellykket hvis vi møter gitte krav til presisjon og ender med et brukervennlig sluttprodukt.

For å realisere prosjektet er vi nødt til å få presisert en del krav. Dette inkluderer:
- Presisjonskrav i posisjon og pose i mm og grader.
- Om vi skal ha statiske kamera eller disse skal følge rigg
- Skal data/video feed være i real-time
- Budsjett

Mulige problemer som kan oppstå underveis i arbeidet kan være:
Risiko B2 - Liten konsekvens, liten sannsynlighet:

- Tapt arbeidstid på grunn av sykdom eller annet fravær. Det er ikke mye vi kan gjøre når sykdom oppstår men vi kan legge planer rundt ferie ol. hvis det blir informert om i forkant (se 4.3). Vi er også en relativt stor gruppe med fire medlemmer, og vi tror det skal være mulig å fylle inn for hverandre dersom et medlem får et illebefinnende.
  Tiltak: Vi kommer ikke til å innføre flere tiltak.

Risiko B3 - Liten konsekvens, middels sannsynlighet.

- Leveringsforsinkelser for utstyr. Dersom vi får en forsinkelse i bestilling kameraer eller i utstyret vi trenger for å sette opp riggen vår kan det påvirke kvaliteten på prosjektet.
  Tiltak: Vi har fått tilgang på det vi trenger av utstyr for å bygge og teste et fungerende produkt, så selv med forsinkelser vil det ikke påvirke fremgangen vår i stor grad. Vi har satt opp en plan for å bestille produktene vi trenger tidlig. Vi anser konsekvens redusert til svært liten og sannsynligheten til liten, risiko A2.
- Feil i hardware. Noe av dette kan oppdages ved å teste utstyr tidlig slik at vi kan returnere defekt utstyr å få tilsendt nytt. Ellers har dette punktet mye til felles med forrige punkt.
  Tiltak:Vi har fått tilgang på fungerende utstyr som er tilstrekkelig for progresjon i utviklingen. Som nevnt i forrige punkt skal vi bestille utstyr i god tid, slik at vi har mulighet til å reparere eller erstatte eventuelle feilvarer. Vi anser risikoen etter tiltak som A2. Liten sannsynlighet, og svært liten konsekvens.

Risiko C3 - Middels konsekvens, middels sannsynlighet

- Scope creep. Dette oppstår når vi itererer for mye på en problemstilling og legger til tilleggsfunksjoner som ikke var definerte i oppgaven og ender med et større scope enn det som var planlagt.
  Tiltak: Vi har satt opp en rigid fremdriftsplan, og gjennom prosjektmetoden vår har vi bestemt oss for å gjennomføre fungerende iterasjoner av produktet før vi legger til mer funksjonalitet. I tillegg har vi daglige stand-up-møter for å rapportere og drøfte fremgangen til alle gruppemedlemmene, som bidrar til at vi får oversikt om noen prøver å overprodusere funksjonalitet underveis. Vi anser risikoen redusert til liten gjennom møtene, og konsekvensen redusert til liten gjennom prosessmetoden. Risiko B2.

Risiko E2 Svært alvorlig konsekvens, liten sannsynlighet.

- Datatap. Dersom vi skulle miste større mengder med data, enten i form av kode eller rapporter står vi i fare for at vi må gjøre store deler av prosjektet på nytt. Dette kan forebygges med gode rutiner for lagring og backup.
  Tiltak: For koden har alle medlemmer lokale kopier av nåværende kode, og alle bruker versjonskontroll i Source Tree. Vi har også både lokale kopier, og kopier i googles sky av de største rapportene våre. Gjennom disse tiltakene anser vi sannsynligheten redusert til svært liten, mens konsekvensen fortsatt er alvorlig. Risikogruppe E1. Gjennom diskusjon i gruppen har vi vurdert denne risikoen til å være akseptabel, siden sannsynligheten er minimal.

5.6 Hovedaktiviteter i videre arbeid

Se Vedlegg A: Gantt-diagram for bachelorgruppe Estimering av modellposisjon og bølgebevegelse i bølgetank

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Kapittel 5.6 er ganske detaljert og beskriver planlagt fremgang.
Første tunge avgjørelse er å avklare hvilken metode man skal bruke. Om man skal bruke "motion capture" eller "model based" gir store føringer på hva som må gjøre i geometri-delen av prosjektet, som er den største delen av prosjektet.

Planen er å ha et MVP ferdig 8. februar. Dette vil være en versjon hvor systemet har et minimum av features. Tanken er å tidlig kartlegge eventuelle utfordringer ift nøyaktighet og praktisk bruk som vi ikke har tatt høyde for.

Videre er det planlagt å ha det meste av software klart for fullstendig testing i starten av mars. Deretter vil det bli tre uker med testing og optimalisering av software, parallelt med at GUI blir utviklet. Å utfylle Metodedel og resterende teoridel vil gjøres mens man arbeider med dette, hovedsakelig til en fastsatt dag i uken.

5.7.2 Styringshjelpemidler

- Gantt-Project
- Google Drive
- Office 365

5.7.3 Utviklingshjelpemidler

- MATLAB
- AutoDesk Fusion 360
- Blender
- Bitbucket
- Sourcetree
- Git
- Overleaf
- LaTeX
- PyCharm
- Python 3
- Numpy
- OpenCV
- Matplotlib

5.7.4 Intern kontroll – evaluering

Intern kontroll i prosjektet med hensyn på fremdrift vil gjennomføres ved bruk av daglige stand-up møter hvor det blir gjennomgått hva arbeid som forventes gjennomført den dagen og fremgang i forhold til gårsdagens mål. I slutten av hver uke vil det være et gruppemøte hvor fremgangen for alle deloppgaver diskuteres og eventuelle problemstillinger blir tatt opp slik at vi kan evaluere om det må gjøres endringer i fremgangsplan eller om det må legges til flere deloppgaver for å løse problemene.

Et delmål ansees som gjennomført når det det foreligger en implementerbar løsning på gitt problemstilling. Løsningen skal bli presentert til en uavhengig tredjepart internt i gruppen for evaluering slik at eventuelle feil kan lukes ut eller forbedringer kan gjøres.

Det vil også være møter med veiledere annenhver uke hvor fremgang blir diskutert og råd blir gitt.

5.8 Beslutninger – beslutningsprosess

Alle viktige avgjørelser skal bli tatt ved avstemning etter diskusjon i plenum. Dersom konflikt oppstår (2 mot 2 stemmer) vil gruppeleder ta avgjørelsen. Alle beslutninger skal dokumenteres med begrunnelse (argumenter for og imot) i ukesrapporten.

Ved arbeid med forprosjekt har alle avgjørelser blitt tatt etter diskusjon i gruppa og det har ikke oppstått noen konflikter.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Bacheloroppgaven skal skrives etter mal gitt på Blackboard IE303612 Bacheloroppgave (2019 VÅR)\Undervisningsmateriell\Rapportmaler\. Det kommer til å bli lagt stor vekt på teori, metode og utførelse.

Gruppen vil dokumentere fremgang hver uke ved bruk av ukesrapporter som inneholder fremgang, avgjørelser som er tatt og mulige problemer eller uforutsette hendelser som har oppstått. En kopi av denne rapporten vil bli sendt til styringsgruppen. Det vil bli holdt en timeliste med tid for oppmøte og arbeidsslutt for alle på gruppen.

Alle rapporter, dokumenter, timeliste og tekniske datablad som følger utstyr vi har brukt skal lagres på gruppens delte rom på Google Drive. Det er tenkt at dette rommet bare skal være tilgjengelig for gruppen og eventuelt veiledere.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Oppdateringsmøte med veiledere vil skje hver andre uke. Disse møtene blir berammet til 30 minutter. Trengs det mer tid ifm. tekniske spørsmål eller lignende skal dette avtales før møtet. Oppdragsgiver kan delta om det sees behov.

7.1.2 Prosjektmøter

Prosjektmøter skal skje hver fredag klokken 13:00. Før klokken 11:00 samme dag skal det leveres inn oppsummering av ukas fremgang av deloppgave ansvarlige slik at en ukesrapport kan fremstilles før møtet. I dette møtet skal eventuelle endringer i fremgangsplan drøftes i forhold til problemstillinger eller uforutsette hendelser som har oppstått underveis i arbeidet, se kap 8.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter

En fremdriftsrapport (også kalt ukesrapport i dette dokumentet) skal leveres til styringsgruppen hver fredag. Denne skal være på formen gitt eksempelvis i Blackboard IE303612 Bacheloroppgave (2019 VÅR)\Undervisningsmateriell\Rapportmaler\Framdriftsrapport (progress report) - eksempel (example)

8 PLANLAGT AVVIKSBEHANDLING

Dersom prosjektets framdrift ikke blir holdt er alternativene å arbeide ut over vanlig arbeidstid i hverdagene eller eventuelt å arbeide i helger for å komme ajour.

For hver arbeidsoppgave er det ansvarlig for denne oppgaven sitt ansvar og gjennomføre innenfor tidsfristen. Vanlig prosedyre dersom dette ikke skulle skje er å gjennomføre en diskusjon i plenum om tiltak for å igjen kunne følge fremdriftsplanen. Deretter er det prosjektleders ansvar om en diskusjon ikke fører til løsning å avgjøre hvilke tiltak som er nødvendig. Der ett eksempel er å utsette sluttdato på deloppgaven.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

For gjennomføring av prosjektet kreves det innkjøp av kamera og markører for sporing av objekter i bølgetanken. Innkjøp av materiale for å lage en testrigg for skalerbar testing av løsningen og feste for kamera på bølgetanken er også nødvendig.

Det er også nødvendig med opplæring i bruk av bølgetanken og vognen for å fjerne risiko for feil bruk eller skade på utstyr eller personell.

10 REFERANSER

Gjelseth, Ørjan; Flatval, Ivan; (2015); "Posisjons- og avstandsmåling med ett enkelt kamera for maritime løfteoperasjoner", NTNU Ålesund
Hartley, R; Zisserman, A; (2004); "Multiple View Geometry in Computer Vision", Cambridge University Press
Solem, Jan Erik; (2012); "Programming Computer Vision with Python", O'Reilly Media inc
Tutorialspoint: Software development life cycles

11 VEDLEGG

Vedlegg A: Gantt-diagram for bachelorgruppe Estimering av modellposisjon og bølgebevegelse i bølgetank.

Appendix A

Appendix: System User Manual

Manual for the users.

Norwegian University of Science and Technology

Vessel Motion Tracking System

User Manual

Even Drugli
Kai Hagseth
Ole Kristian Sande
Vegard Fjørtoft

16. mai 2019

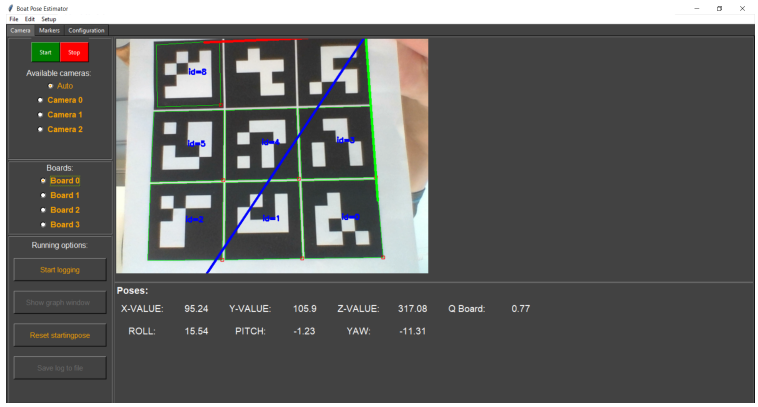
Innhold

Sammendrag

This guide explains how a user can interact with the system tracking system installed in the water tank at NTNU Ålesund. The system was a bachelor thesis in 2019.

1 Use the system

- If you haven't done it, do section 2 and 3 before starting.
- When done just press 'Start' on the home screen and the program should start and show a videofeed.
- To reset the starting point for the boat in program, press 'Reset starting-pose'.
- To start logging to a file, press 'Start logging'. To access the file, press 'Save log to file' and it can be found in the program root folder as a csv-file.
- To access the graphs open the graph window with 'Show graph window'.



Figur 1: The main screen.

2 Choosing cameras

1. Start up the application.

2. Make sure all cameras you want to use is connected to USB, either directly or through a USB-hub.

3. Go in Configuration Camera config tab.

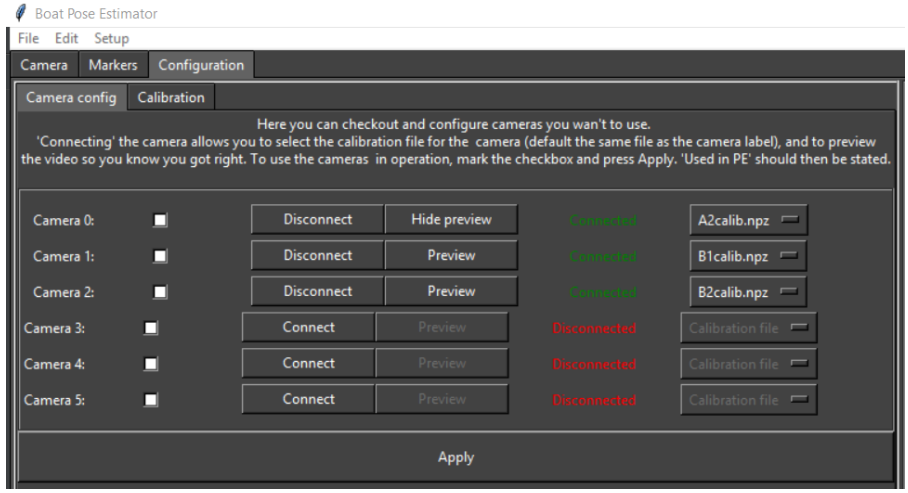
4. No you can select what cameras to use. The list starts at 0 and go upwards (if no camera on 3, no camera on 4 or 5 either). If you don't know which cameras to use:

5. If the PC in use has a in-built camera, it is *most often* on index 0. But for a few PCs its on a other index. The best is to 'Connect' camera 0 and see whether its the inbuilt camera or not. If it is, you can 'Disconnect' it again, and leave it.

6. Then you can use the checkbox to the left and mark the cameras you want to use. Whether they are already 'Connected' or not yet, doesn't matter.

7. Press 'Apply' and wait. It can take a small while (normally 15-20sec, up to 1 minute.)

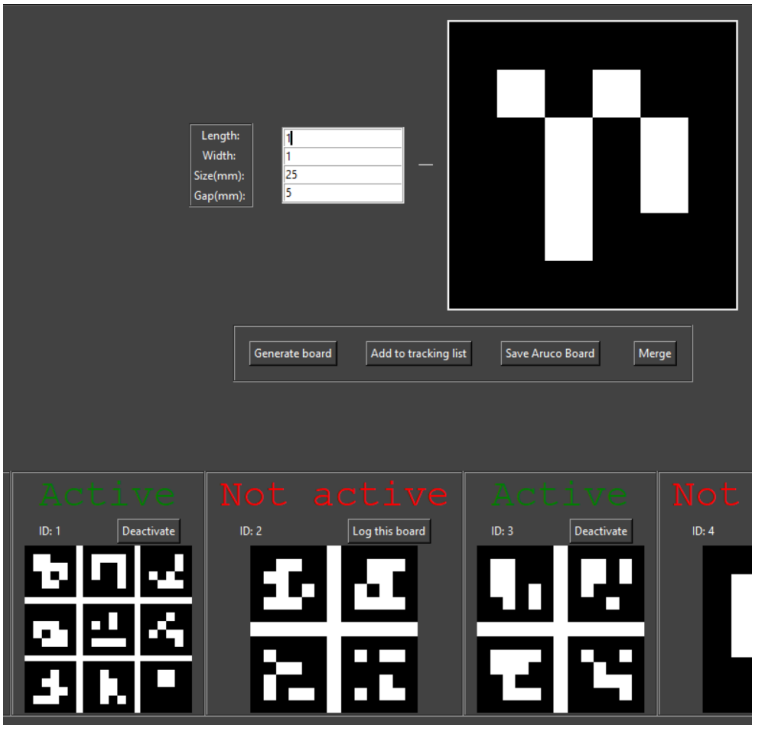
8. In the end all cameras should have 'Used in PE' stated in green. Otherwise 'Failed' in red will come up, then connection was unsuccessful.



Figur 2: Camera setup page.

3 Adding Aruco-board

1. Go to the Marker tab.
2. Here you can create markers.
3. Choose how many markers you want to use. This depends on how big your area on tracking object. A marker shouldn't be less than 40x40mm if the camera is 1-1.5m from the tracking object. If you have three markers, the total are taken will be about 40mm*3 + 25mm spacing = 145mm across. If you have less space available, reduce number of markers to less than 3x3. 3x2 or 2x1 is also fine etc.
4. Set the square size (as said at least 40mm) and the gap (at least 5mm).
5. You can then 'Generate board' to see how it would become. The marker displayed in the window is not a relative scale!
6. If you want to use it, do 'Add to tracking list'. The marker will now be used in the estimation.
7. If you want to print out the marker:
8. Press 'Save Aruco board'.
9. The board can now be found as a PDF in the root folder of the program, called 'arucoBoard.pdf'. Where the program folder lies, depends on installation.
10. When printing out, be sure to use landscape mode and set scaling to 100% or None! Otherwise the markers will be printed in wrong size.
11. You can now repeat the process if you want to use more boards. The program can easy track 4-5 boards or even more. But it doesn't support more than a total of 50 markers in total.



Figur 3: Aruco marker page.

4 Calibrating camera

Camera calibration is done so the physical properties of the camera can be known to the system.

NOTE! Calibration is only needed once for each camera type, as long as the lens on the cameras hasn't been tampered with.

1. Go in Configuration → Calibration tab.
2. Select the camera you want to calibrate.
3. If it hasn't been Connected yet, please do so. You can also preview so you know you got right camera.

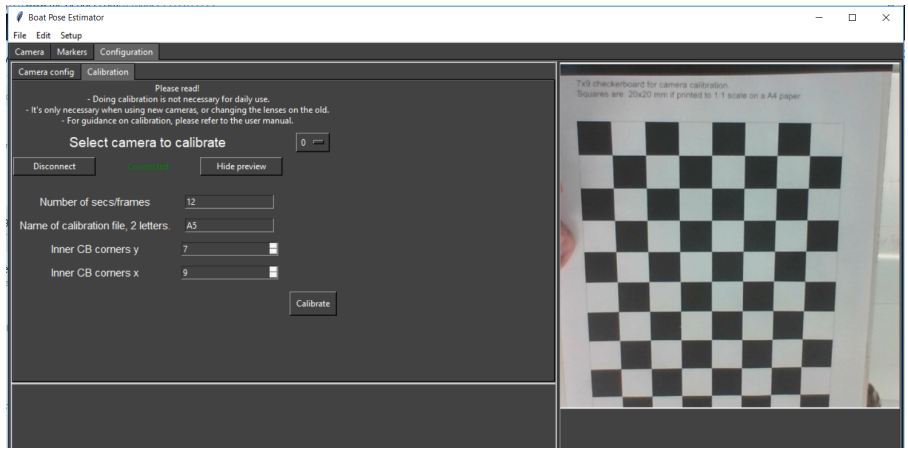
4. You can choose between calibrating with a film or with a set of images. Generally images is recommended, but video is a bit easier.

5. Then choose number of images or seconds of film you want to take. The higher the better, but we recommend at least 10 images or 15 seconds of film.

6. Set the size of thew chessboard. Note it's the inner corners that are counted. A normal 8x8 chessboard, have 7 vertical and 7 horizontal innercorners.

7. When ready press 'Calibrate'. The calibration starts right away. Remember to turn the checkboard as many ways as possible, and in total fill the whole frame (some images down and some far up). If Images where choosen, you need to press 'Space' on the keyboard between each image. The image is displayed. The image is taken right away. With video, it takes 3 images per second. Remember to move slowly when using film, otherwise it will become blurry and useless.

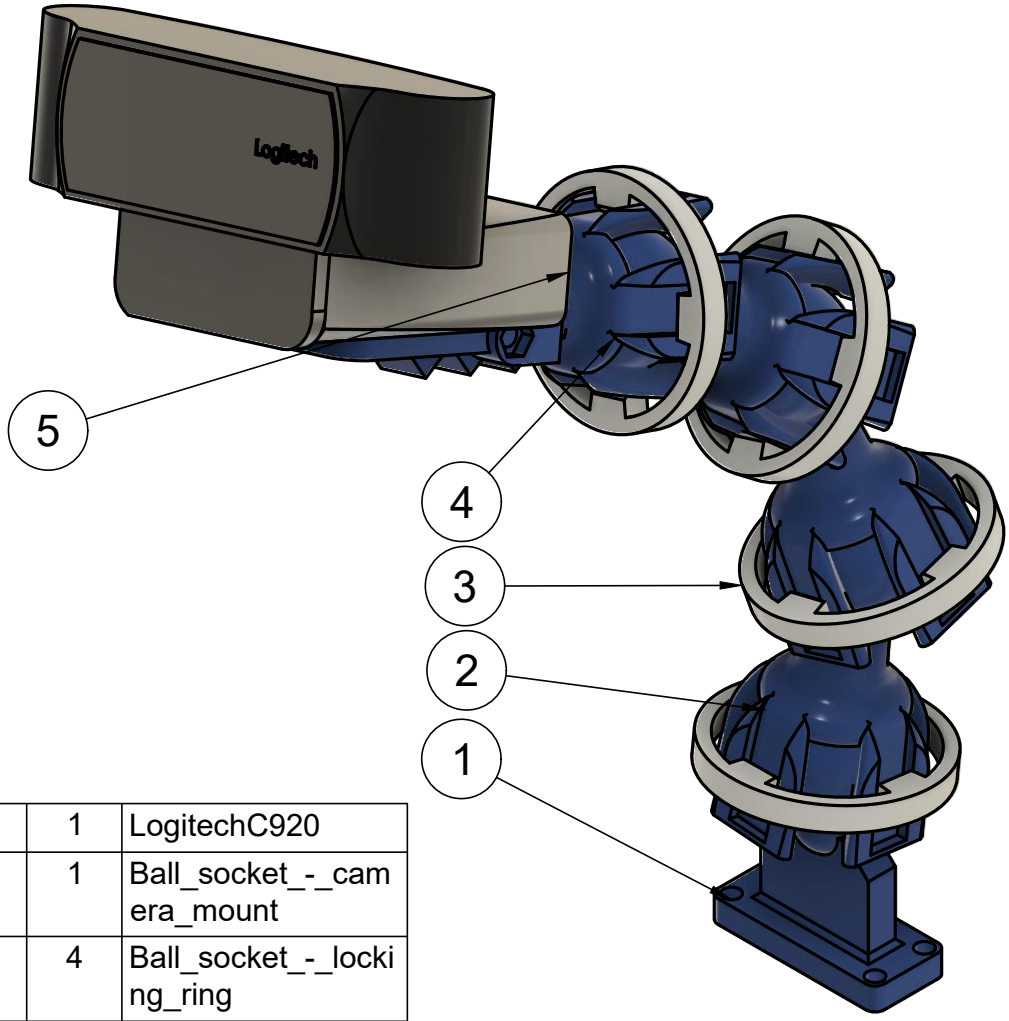
8. When finished, the program will show the result, before it closes all calibration windows. The file created should now be visible in the dropdown-menu for the cameras.



Figur 4: Calibration tab.

Appendix B

Appendix: Mechanical Drawings

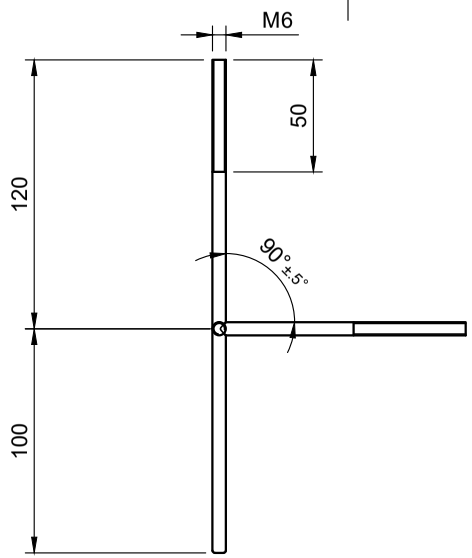


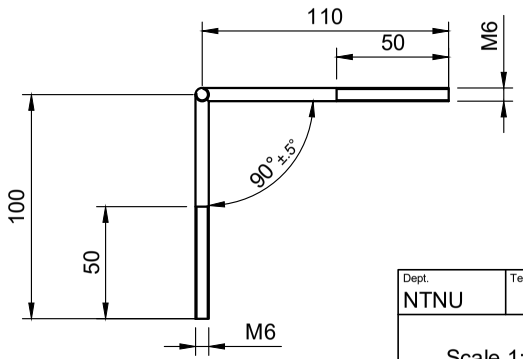
Item	Qty	Part Number
5	1	LogitechC920
4	1	Ball_socket_-_camera_mount
3	4	Ball_socket_-_locking_ring
2	3	Ball_socket_-_link
1	1	Ball_socket_-_generic_holder

Parts List

Consept drawing for holding camera.

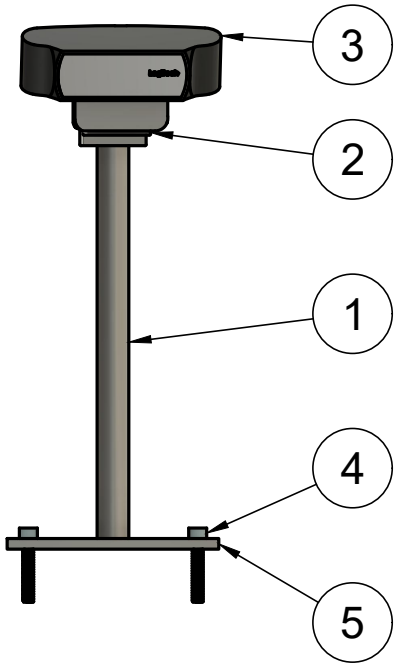
Dept. NTNU	Technical reference	Created by Ole Kristian Sande 2.04.2019	Approved by
		Document type	Document status
		Title adjustable_stand	DWG No. 1
			Rev. A / Date of issue 02.03.2019 / Sheet 1/1

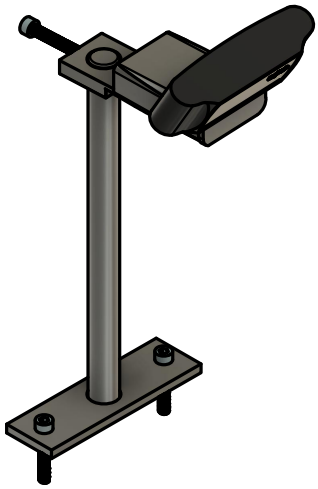




Alle gjenger er M6x1x50. Bruk enten gjengestang eller vanlig stang. Alt etter hva materiale dere har liggende.

Dept.	Technical reference	Created by	Approved by	
NTNU		Ole Kristian Sande 05.02.2019	OKS	05.02.19
Scale 1:2		Document type	Document status	
		Title: Axis Cross	DWG No.: 1	
			Rev.: 1	Date of issue: 05.02.19 / Sheet: 1/1

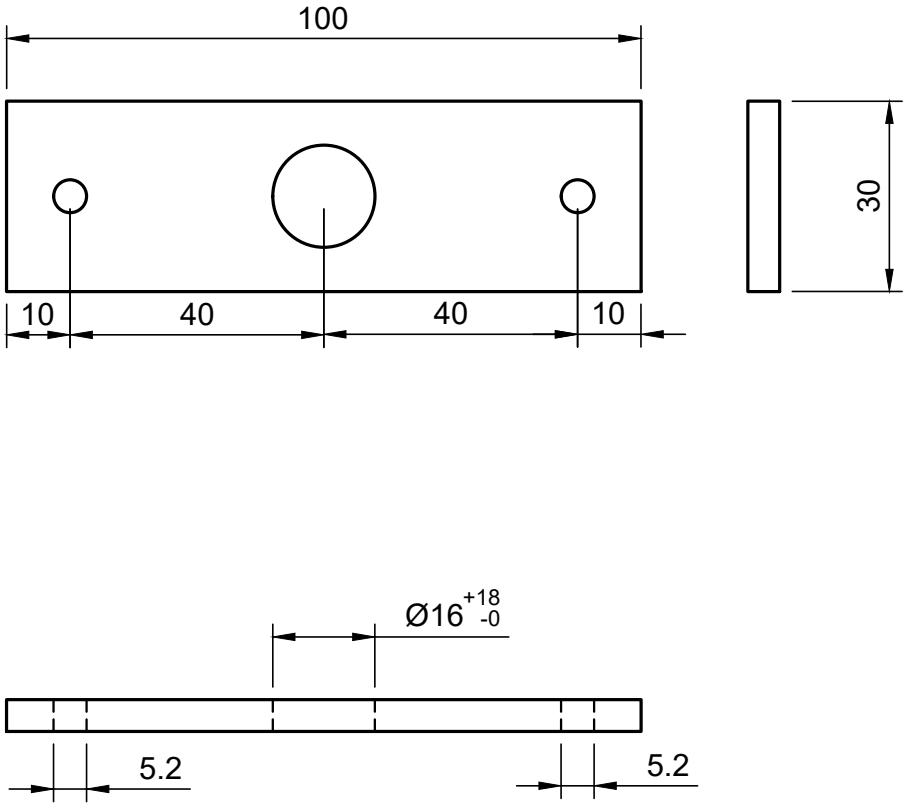




Item	Qty	Part Number	Description
5	1	Bracket	Bracket for mounting camera stand to towing tank
4	3	M5x30	Umbraco screws for fastening
3	1	LogitechC920	Web Camera
2	1	Camera_holder	Connector between camera and holder
1	1	Shaft	Aluminium shaft Ø16 mm

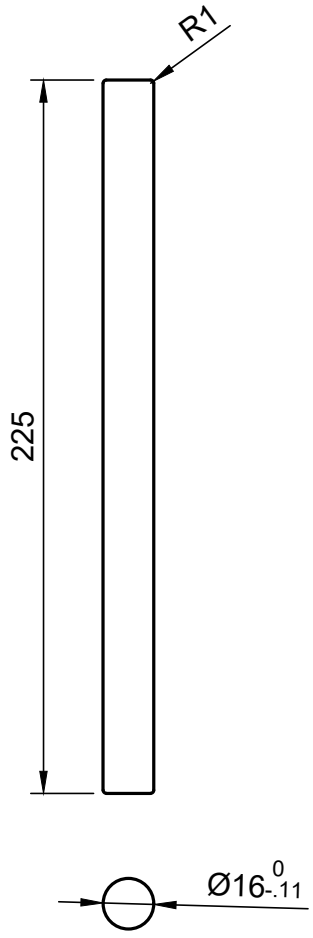
Parts List

Dept.	Technical reference	Created by	Approved by
NTNU	NA	Ole Kristian Sande 08.05.2019	
		Document type: Assembly	Document status: Complete
		Title: adjustable_stand	DWG No.: 1
			Rev.: A; Date of issue: 08.05.2019; Sheet: 1/1



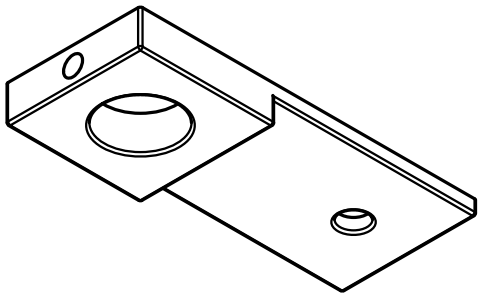
3D Print with MakerBot 2+

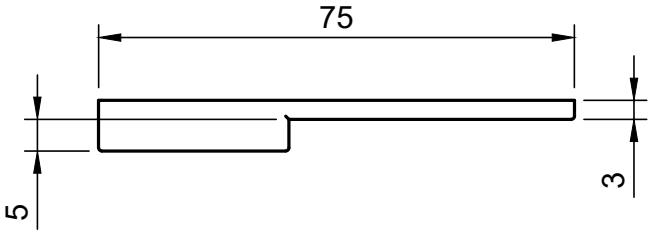
Dept.	Technical reference	Created by	Approved by	
NTNU	NA	Ole Kristian Sande 08.05.2019		
		Document type	Document status	
		Title: bracket	DWG No.: 2	
			Rev.: A	Date of issue: 08.05.2019 / Sheet: 1/1

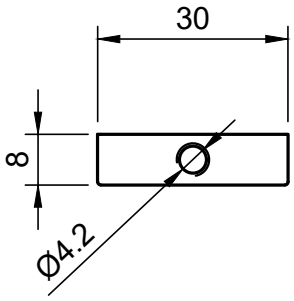


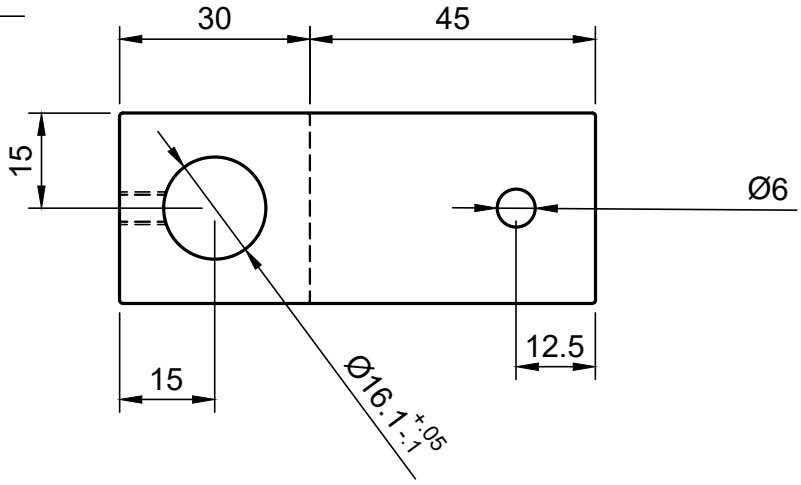
Can use pipes if shaft is not found. Do not use steel because water contact can occur.

Dept.	Technical reference	Created by	Approved by		
NTNU		Ole Kristian Sande 08.05.2019			
		Document type	Document status		
		Title	DWG No.		
		shaft_16mm	3		
			Rev.	Date of issue	Sheet
			A	08.05.2019	1/1



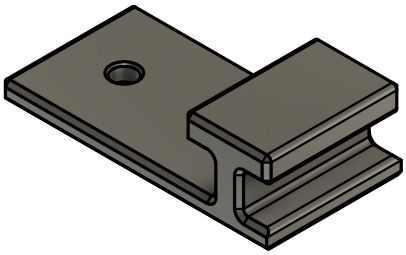




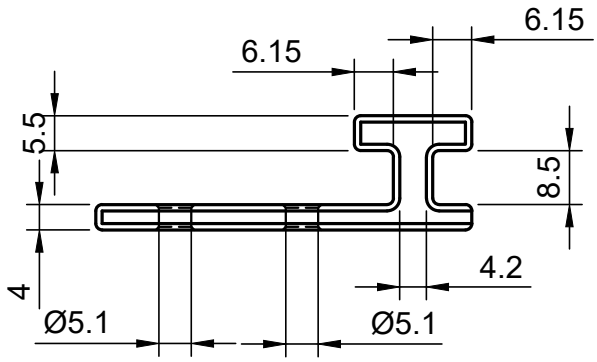


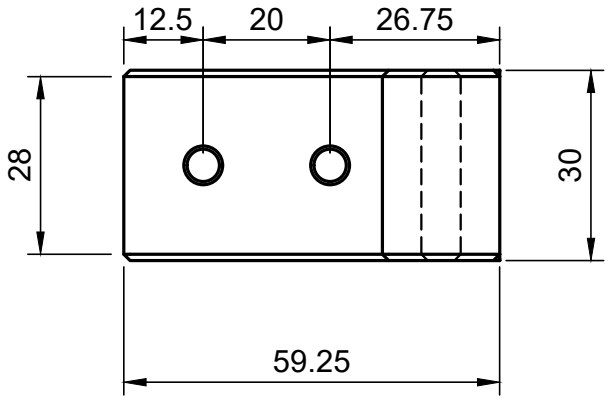
3D Print part

Dept.	Technical reference	Created by	Approved by	
NTNU	NA	Ole Kristian Sande 08.05.2019		
		Document type	Document status	
		Title: camera_holder	DWG No.: 4	
			Rev.: A	Date of issue: 08.05.2019 / Sheet: 1/1



Edges have 1mm chamfer.





Dept.	Technical reference	Created by	Approved by	
NTNU		Ole Kristian Sande 15.05.2019	OKS	14.05.2019
		Document type	Document status	
			Complete	
		Title	DWG No.	
		bracket_track	1	
			Rev.	Date of issue / Sheet
			A	13.05.2019 / 1/1

Appendix C

Appendix: GUI Images

Figure C.1: Live screen

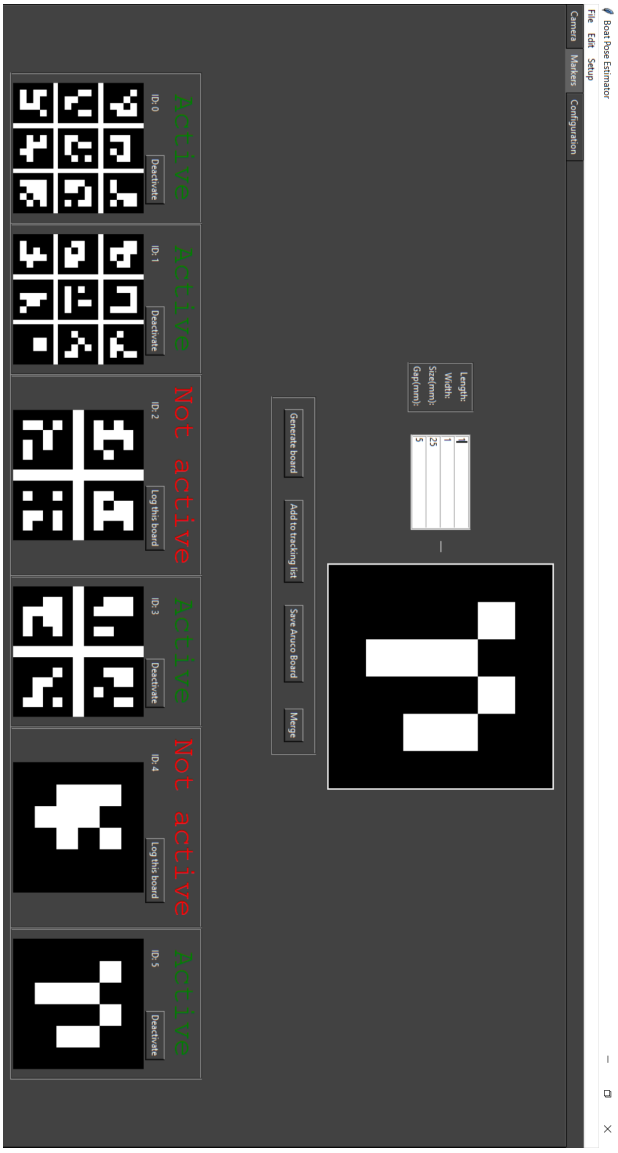


Figure C.2: Aruco marker tab

Figure C.3: Calibration.

Figure C.4: Setting up to merge.

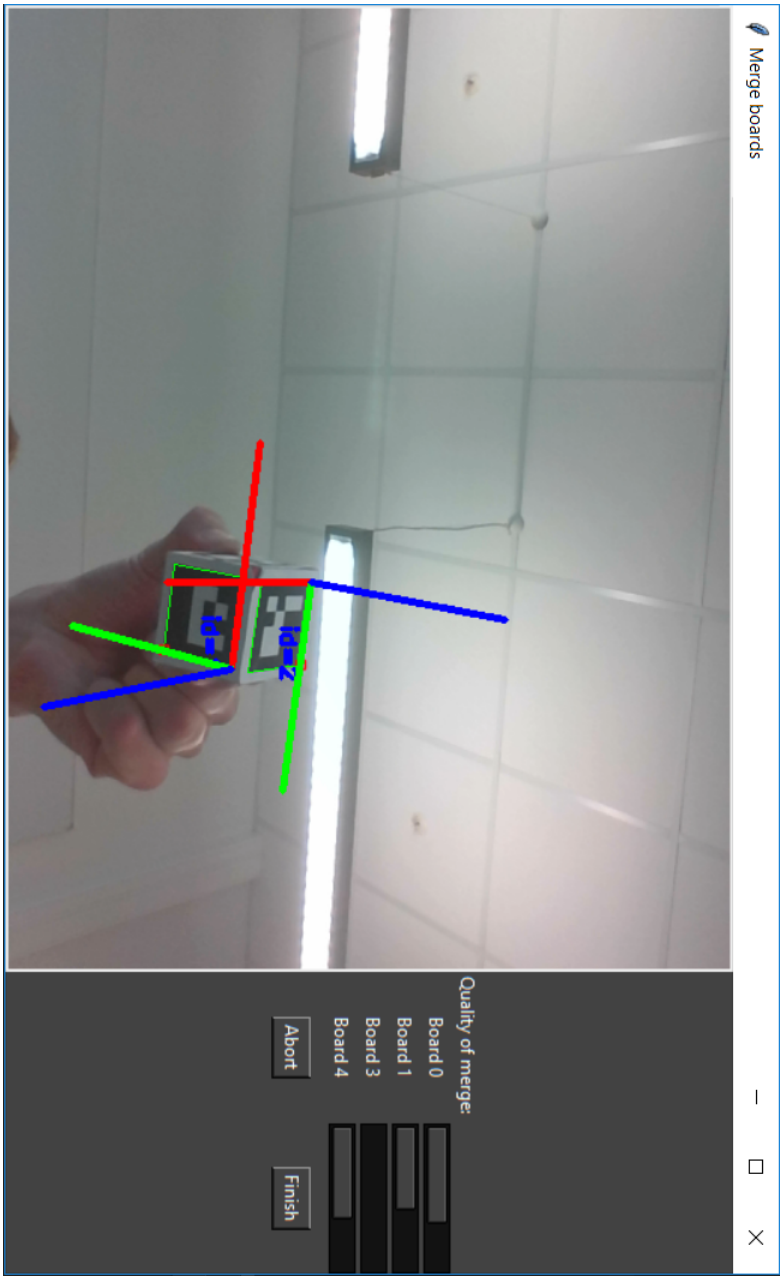


Figure C.5: Merging boards.

Appendix D

Appendix: Preproject report

(In norwegian)

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Estimering av modellposisjon og bølgebevegelse i bølgetank

KANDIDATNUMMER(E):			
DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
	IE303612	**Bacheloroppgave**	- Åpen
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
AUTOMATISERINGSTEKNIKK		/	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):

Oppdrag gitt av NTNU Institutt for havromsoperasjoner og byggteknikk/Institutt for IKT og realfag
Veiledere: Ottar L. Osen og Robin Bye

OPPGAVE/SAMMENDRAG:

Ved forsøk med skipsmodeller og andre modeller vanntank, er det ønskelig å kunne hente ut data om modellens posisjon og orientering, samt beregne bølgeform og -høyde i tanken.

Oppgaven går ut på å finne løsninger for dette. Det er planlagt å bruke flere kamera til å løse oppgaven, hvor punkter på fartøyet sammenlignes fra flere vinkler for å få en 3D representasjon av fartøyet/objektet.

Prosjektet er en avsluttende bacheloroppgave i Automatiseringsteknikk, gitt av NTNU IHB og IIR.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Ved forsøk med skipsmodeller og andre modeller vanntank, er det ønskelig å kunne hente ut data om modellens posisjon og orientering, samt beregne bølgeform og -høyde i tanken.

Oppgaven går ut på å finne løsninger for dette. Det er planlagt å bruke flere kamera til å løse oppgaven, hvor punkter på fartøyet sammenlignes fra flere vinkler for å få en 3D representasjon av fartøyet/objektet.

Prosjektet er en avsluttende bacheloroppgave i Automatiseringsteknikk, gitt av NTNU IHB og IIR.

2 BEGREPER

IHB - Institutt for havromsoperasjoner og byggteknikk

IIR - Institutt for IKT og realfag

Marker based motion Capture - Teknikk for å fange bevegelse i et objekt ved hjelp av flere kameraer som logger referansepunkter.

Model based pose estimation - Teknikk for logging av posisjon og rotasjon av objekt ved hjelp av ett kamera og en fysisk modell med kjente dimensjoner.

MVP - "Minimum Viable Produkt", enkleste brukbare produkt

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)	
Even Drugli	**476122**
Vegard Fjørtoft	**460011**
Kai Hagseth	**997480**
Ole Kristian Sande	**476134**

3.1.1 Oppgaver for prosjektgruppen - organisering

Gruppen vil bestå av en prosjektleder, sekretær, kodeansvarlig og en ansvarlig for hardware og innkjøp. Når en deloppgave blir tildelt vil det være en person som som er ansvarlig og en person som er hjelper.

3.1.2 Oppgaver for prosjektleder

- Skal være kontaktperson for oppdragsgiver og veiledere.
- Skal holde oversikt over fremdrift og utfordringer i prosjektet, og rapportere disse til veiledere og evt. oppdragsgiver når det sees nødvendig eller hensiktsmessig.
- Skal sørge for at alle i gruppen har oppgaver til enhver tid.
- Sørge for god trivsel i gruppa.

3.1.3 Oppgaver for sekretær

- Skal ha hovedansvar for at rapport blir skrevet og at denne blir av høy kvalitet.
- Ansvar for utarbeiding av møtereferat og fremdriftsrapporter.
- Ansvar for dokumentasjon av systemet.

3.1.4 Oppgaver for kodeansvarlig

- Ansvarlig for den underliggende arkitekturen i programvareutviklingen
- Har ansvar for god kodestruktur
- Skal holde oversikt over hva som må gjøres i kode
- Har ansvar for kompatibilitet og ryddig versjonskontroll
- Ansvar for dokumentasjon av kode (gode kommentarer i kode og i sourcetree).

3.1.5 Oppgaver for Hardware og Innkjøpsansvarlig

- Ansvar for innkjøp av nytt utstyr
- Budsjettansvarlig
- Ansvar for fremgang i implementering av fysisk utstyr

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Veiledere: Ottar L. Osen og Robin Bye
Oppdragsgiver IHB: Karl Henning Halse

4 AVTALER

4.1 Avtale med oppdragsgiver

Oppdragsgiver er Karl Henning Halse fra NTNU IHB. Det foreligger ingen skriftlig avtale mellom studenter og oppdragsgiver per nå.

4.2 Arbeidssted og ressurser

Fast arbeidssted blir på campus Ålesund, fortrinnsvis L167 eller Tunglab (L044) på NTNU.

Møter med oppdragsgiver/veiledere vil skje på NTNU.

Oppdateringsmøte med veiledere vil skje hver andre uke. Disse møtene blir berammet til 30 minutter. Trengs det mer tid ifm. tekniske spørsmål eller lignende skal dette avtales før møtet. Oppdragsgiver kan delta om det sees behov. Gruppen skal lage ukesrapport siste arbeidsdag i uken.

Prosjektet inneholder ingen informasjon som er taushetsbelagt. Derfor er det ikke et krav om at prosjektet skal være skjult fra offentligheten.

4.3 Gruppenormer – samarbeidsregler – holdninger

Gruppen er blitt enig om:

- Alle i gruppen skal møte opp på avtalt klasserom alle hverdager innen klokken 09:00.
- Helgearbeid avtales fortløpende, minst 24 timer på forhånd.
- Dagen avsluttes på eget skjønn, men det forventes at ingen går før klokken 16:00 dersom man ikke har god grunn og koordinerer med resten av gruppa.
- Avspasering skal avklares fortrinnsvis minimum 1 uke på forhånd. Sykdom e.l. skal informeres om så snart som mulig.
- God kommunikasjon gruppen imellom når det kommer til fremgang i forhold til tidsfrister på delmål slik at gruppen kan diskutere eventuelle utsettelser eller tildeling av ekstra arbeidskraft.
- Deloppgave ansvarlig skal levere inn en oppsummering av ukas fremgang (hva som har blitt gjort, eventuelle problemer og hva som skal gjøres neste uke) til sekretær innen klokka 11:00 hver fredag slik at en rapport kan utformes før gruppemøte klokka 13:00

Prosjektet skal gjenspeile et arbeid som er gjort av automasjonsingeniører. Det vil si at arbeidet skal ha høy kvalitet. Det betyr god kildekode, gode systemmanualer og at alle opptrer profesjonelt. Det betyr at enhver står ansvarlig for å levere innenfor frister, med en god kvalitet som en selv kan være stolt av.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Problemstilling:

- Er det mulig gjennom et kamerasystem å registrere et skipsmodells posisjon og retning gjennom kamerasyn?
- Er det mulig å innhente data om bølgehøyde og -bevegelser gjennom samme kamerasystem, eller tilhørende system?
- Kan dette presenteres på en måte som gjør det aktivt nyttig i testing og forskning av skips- og havmodeller i bølgevanntank?

Et godt resultatmål vil være å få et ja på problemstillingene ovenfor, eventuelt få et godt begrunnet nei. I tillegg har man prosess- og effektmål. Hva ønskes å oppnås som følge av jobbingen under prosessen og hva er det langsiktige målet hvor prosjektet kun er et delmål. Et stort prosessmål er å kunne bruke dette prosjektet som en overgang fra student til arbeidsliv. Stor del av arbeidslivet er prosjektbasert, og derfor kan et slikt prosjekt sees på en prøvelse før det virkelig arbeidsliv, hvor mye mer står på spill tidsmessig og økonomisk.
Et faglig stort prosessmål er å kunne utvide kompetanse, særlig innen vision, og få en effekt av dette i et fremtidig arbeidsliv.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Prosjektet sikter på høyest mulig presisjon i målinger. Et matematisk begrunnet anslag for nøyaktigheten
Et mål er å få pose-estimasjon på en nøyaktighet på +-1 mm. Hva dette utgjør i rotasjon kommer an på dimensjonene i skipet og utstyret.

Akser:	Maks avvik
Posisjon	+- 1 mm
Rotasjon	+- 0.5 grader

Prosjektresultatet skal være en ferdig oppkoblet løsning ved vanntanken som kan brukes av andre studenter for datainnsamling. Det skal være utarbeidet manual for styring og bruk av systemet.

5.3 Planlagt framgangsmåter for utviklingsarbeidet - metoder

Gruppen har blitt enig om at vi skal bruke en iterativ og inkrementell modell for systemutviklingen. Fokuset for denne modellen er å utvikle systemet gjennom gjentakende faser, (iterativ), og på den måten kunne justere systemet etter hvert som vi legger til ny funksjonalitet(inkrementell).

For å komme i gang med denne utviklingsmodellen er det gitt at man har en plan for hvilken funksjonalitet man ønsker at systemet skal ha når prosjektet er ferdig, hvilken funksjonalitet man trenger for å lage en fungerende prototype, og hvordan man kan implementere fungerende iterasjoner mellom disse.

Etter at prosjektplanen er lagt starter den initielle fasen av utviklingen hvor en enkel modell av systemet utvikles, testes, og evalueres. Denne fasen etterfølges av repeterende iterasjoner hvor mer funksjonalitet legges til, og hvor eksisterende funksjonalitet forbedres og refaktoreres.

Fordelen med en iterativ og inkrementell prosess at man får et fungerende system på et tidlig stadium, og at man hele tiden har et klart scope for hva som skal gjøres i prosjektet. I evalueringsfasen mellom iterasjonene kan man finne feil og mangler ved arbeidet man har gjort, og kan reagere og reparere kort tid etter.

Når det kommer til ulemper er det lett å få problemer med systemarkitekturen dersom man gjør feilaktige antakelser om fremtiden. Det vil si at hvis systemet bygges på grunnlag av at neste iterasjon skal kunne fungere tilfredsstillende, uten at man tar hensyn lenger frem i tid, vil man få problemer. Dette kan forebygges gjennom god planlegging, og at man tar seg tid til å snakke sammen om hvordan iterasjonen vi begynner på er fremtidssikret i planleggingsfasene.

5.4 Informasjonsinnsamling – utført og planlagt

Det skal utredes hvilke metoder som gir best presisjon og nytte ved plassering av datapunkt på modellen. Vi har drøftet tre metoder.

Alternativ 1: Markørbasert posisjonssporing. Dette er en flerkameraløsning som gir oss stor frihet i hvordan markørene plasseres, og hvor mange markører vi ønsker å bruke. Etter en kalibrering av alle kameraene kan et markørpunkts posisjon estimeres i tre dimensjoner ved hjelp av triangulering. Dersom man har minst tre punkter i bildet og antar at objektet er rigid kan man estimere alle de seks frihetsaksene til objektet. Denne metoden bør i teorien også kunne estimere fordreining i objektet hvis man bruker mer enn tre punkter per objekt. Ved å bruke mer enn to kameraer kan man oppnå høyere presisjon, dekningsområde og redundans, på bekostning av tregere prosessering og høyere materialkostnader. For å øke ytelsen i denne metoden er det mulig å bruke IR-dioder som markørpunkt, og filtrere ut lys som ikke er infrarødt. Da vil man kunne finne alle punktene i bildet med en ressursvennlig terskel-operasjon. I våre undersøkelser har vi ikke funnet noen implementeringer av denne metoden for vår use-case, og løsninger med åpen kildekode er vanskelig å komme over. Dette kan bety at implementeringen er ressurskrevende, eller at dette ikke er en hensiktsmessig metode for vårt scenario.

Alternativ 2: Modellbasert positurestimering. Dette alternativets største fordel er at man kan lese av informasjon om alle de seks frihetsaksene til objektet med kun ett kamera, og at det ser ut til å ha mange implementasjoner på lignende caser som vår egen fra før. Det går ut på at det plasseres et objekt med kjente dimensjoner, for eksempel en figur med tre linjer som peker i x, y og z-retning med en kule på enden av hver linje. Ut fra denne modellen er det mulig å både kalibrere kamera og spore objektet. Denne metoden har tidligere blitt delvis utredet her på instituttet av Ø. Gjelseth og I I. Flatval så i oppgaven "Posisjons- og avstandsmåling med ett enkelt kamera for maritime løfteoperasjoner" (2015). Vi ser for oss at denne metoden kan utvides med flere kamera for å kunne oppnå bedre presisjon og større dekningsområde. Ved bruk av flere modeller bør vil også kunne øke presisjonen orientering om de pares, og man åpner mulighet for flerobjektssporing. Vi ser for oss at også denne metoden kan pares med IR-dioder for bedre ytelse, hvor endepunktene på figuren vår byttes ut med dioder.

Alternativ 3: Stereosyn for skroggjenkjenning. Her er fordelen at man i teorien ikke ville trenge markører av noe slag for å estimere de seks frihetsaksene. Gjennom en utvidelse av stereo-syn algoritmer kan man lage en 3d-modell av skroget, og beregne modellens positur ut fra et satt utgangspunkt. Dette alternativet er det mest ressurskrevende, og vil neppe fungere i sanntid. I tillegg fungerer ikke disse algoritmene med refleksjon og refraksjon, noe som ikke er ideelt når vi jobber i og rundt vann. Vi tror ikke dette er en god løsning for vår case.

Litteratur:
Multiple View Geometry in Computer Vision, *R. Hartley, A. Zisserman, Mars 2004, ISBN: 9780521540513*
Programming Computer Vision, *Jan Erik Solem, Juni 2012 ISBN: 9781449316549*

5.5 Vurdering – analyse av risiko

Matrise for risikovurderinger ved NTNU

KONSEKVENS	Svært Alvorlig	E1	E2	E3	E4	E5
	Alvorlig	D1	D2	D3	D4	D5
	Moderat	C1	C2	C3	C4	C5
	Liten	B1	B2	B3	B4	B5
	Svært liten	A1	A2	A3	A4	A5
		Svært liten	Liten	Middels	Stor	Svært stor
		SANNSYNLIGHET				

Farge		Beskrivelse
Rød		Uakseptabel risiko. Tiltak skal gjennomføres for å redusere risikoen
Gul		Vurderingsområde. Tiltak skal vurderes
Grønn		Akseptabel risiko. Tiltak kan vurderes ut fra andre hensyn.

Vi anser prosjektet som vellykket hvis vi møter gitte krav til presisjon og ender med et brukervennlig sluttprodukt.

For å realisere prosjektet er vi nødt til å få presisert en del krav. Dette inkluderer:
- Presisjonskrav i posisjon og pose i mm og grader.
- Om vi skal ha statiske kamera eller disse skal følge rigg
- Skal data/video feed være i real-time
- Budsjett

Mulige problemer som kan oppstå underveis i arbeidet kan være:
Risiko B2 - Liten konsekvens, liten sannsynlighet:

- Tapt arbeidstid på grunn av sykdom eller annet fravær. Det er ikke mye vi kan gjøre når sykdom oppstår men vi kan legge planer rundt ferie ol. hvis det blir informert om i forkant (se 4.3). Vi er også en relativt stor gruppe med fire medlemmer, og vi tror det skal være mulig å fylle inn for hverandre dersom et medlem får et illebefinnende.
  Tiltak: Vi kommer ikke til å innføre flere tiltak.

Risiko B3 - Liten konsekvens, middels sannsynlighet.

- Leveringsforsinkelser for utstyr. Dersom vi får en forsinkelse i bestilling kameraer eller i utstyret vi trenger for å sette opp riggen vår kan det påvirke kvaliteten på prosjektet.
  Tiltak: Vi har fått tilgang på det vi trenger av utstyr for å bygge og teste et fungerende produkt, så selv med forsinkelser vil det ikke påvirke fremgangen vår i stor grad. Vi har satt opp en plan for å bestille produktene vi trenger tidlig. Vi anser konsekvens redusert til svært liten og sannsynligheten til liten, risiko A2.
- Feil i hardware. Noe av dette kan oppdages ved å teste utstyr tidlig slik at vi kan returnere defekt utstyr å få tilsendt nytt. Ellers har dette punktet mye til felles med forrige punkt.
  Tiltak:Vi har fått tilgang på fungerende utstyr som er tilstrekkelig for progresjon i utviklingen. Som nevnt i forrige punkt skal vi bestille utstyr i god tid, slik at vi har mulighet til å reparere eller erstatte eventuelle feilvarer. Vi anser risikoen etter tiltak som A2. Liten sannsynlighet, og svært liten konsekvens.

Risiko C3 - Middels konsekvens, middels sannsynlighet

- Scope creep. Dette oppstår når vi itererer for mye på en problemstilling og legger til tilleggsfunksjoner som ikke var definerte i oppgaven og ender med et større scope enn det som var planlagt.
  Tiltak: Vi har satt opp en rigid fremdriftsplan, og gjennom prosjektmetoden vår har vi bestemt oss for å gjennomføre fungerende iterasjoner av produktet før vi legger til mer funksjonalitet. I tillegg har vi daglige stand-up-møter for å rapportere og drøfte fremgangen til alle gruppemedlemmene, som bidrar til at vi får oversikt om noen prøver å overprodusere funksjonalitet underveis. Vi anser risikoen redusert til liten gjennom møtene, og konsekvensen redusert til liten gjennom prosessmetoden. Risiko B2.

Risiko E2 Svært alvorlig konsekvens, liten sannsynlighet.

- Datatap. Dersom vi skulle miste større mengder med data, enten i form av kode eller rapporter står vi i fare for at vi må gjøre store deler av prosjektet på nytt. Dette kan forebygges med gode rutiner for lagring og backup.
  Tiltak: For koden har alle medlemmer lokale kopier av nåværende kode, og alle bruker versjonskontroll i Source Tree. Vi har også både lokale kopier, og kopier i googles sky av de største rapportene våre. Gjennom disse tiltakene anser vi sannsynligheten redusert til svært liten, mens konsekvensen fortsatt er alvorlig. Risikogruppe E1. Gjennom diskusjon i gruppen har vi vurdert denne risikoen til å være akseptabel, siden sannsynligheten er minimal.

5.6 Hovedaktiviteter i videre arbeid

Se Vedlegg A: Gantt-diagram for bachelorgruppe Estimering av modellposisjon og bølgebevegelse i bølgetank

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Kapittel 5.6 er ganske detaljert og beskriver planlagt fremgang.
Første tunge avgjørelse er å avklare hvilken metode man skal bruke. Om man skal bruke "motion capture" eller "model based" gir store føringer på hva som må gjøre i geometri-delen av prosjektet, som er den største delen av prosjektet.

Planen er å ha et MVP ferdig 8. februar. Dette vil være en versjon hvor systemet har et minimum av features. Tanken er å tidlig kartlegge eventuelle utfordringer ift nøyaktighet og praktisk bruk som vi ikke har tatt høyde for.

Videre er det planlagt å ha det meste av software klart for fullstendig testing i starten av mars. Deretter vil det bli tre uker med testing og optimalisering av software, parallelt med at GUI blir utviklet. Å utfylle Metodedel og resterende teoridel vil gjøres mens man arbeider med dette, hovedsakelig til en fastsatt dag i uken.

5.7.2 Styringshjelpemidler

- Gantt-Project
- Google Drive
- Office 365

5.7.3 Utviklingshjelpemidler

- MATLAB
- AutoDesk Fusion 360
- Blender
- Bitbucket
- Sourcetree
- Git
- Overleaf
- LaTeX
- PyCharm
- Python 3
- Numpy
- OpenCV
- Matplotlib

5.7.4 Intern kontroll – evaluering

Intern kontroll i prosjektet med hensyn på fremdrift vil gjennomføres ved bruk av daglige stand-up møter hvor det blir gjennomgått hva arbeid som forventes gjennomført den dagen og fremgang i forhold til gårsdagens mål. I slutten av hver uke vil det være et gruppemøte hvor fremgangen for alle deloppgaver diskuteres og eventuelle problemstillinger blir tatt opp slik at vi kan evaluere om det må gjøres endringer i fremgangsplan eller om det må legges til flere deloppgaver for å løse problemene.

Et delmål ansees som gjennomført når det det foreligger en implementerbar løsning på gitt problemstilling. Løsningen skal bli presentert til en uavhengig tredjepart internt i gruppen for evaluering slik at eventuelle feil kan lukes ut eller forbedringer kan gjøres.

Det vil også være møter med veiledere annenhver uke hvor fremgang blir diskutert og råd blir gitt.

5.8 Beslutninger – beslutningsprosess

Alle viktige avgjørelser skal bli tatt ved avstemning etter diskusjon i plenum. Dersom konflikt oppstår (2 mot 2 stemmer) vil gruppeleder ta avgjørelsen. Alle beslutninger skal dokumenteres med begrunnelse (argumenter for og imot) i ukesrapporten.

Ved arbeid med forprosjekt har alle avgjørelser blitt tatt etter diskusjon i gruppa og det har ikke oppstått noen konflikter.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Bacheloroppgaven skal skrives etter mal gitt på Blackboard IE303612 Bacheloroppgave (2019 VÅR)\Undervisningsmateriell\Rapportmaler\. Det kommer til å bli lagt stor vekt på teori, metode og utførelse.

Gruppen vil dokumentere fremgang hver uke ved bruk av ukesrapporter som inneholder fremgang, avgjørelser som er tatt og mulige problemer eller uforutsette hendelser som har oppstått. En kopi av denne rapporten vil bli sendt til styringsgruppen. Det vil bli holdt en timeliste med tid for oppmøte og arbeidsslutt for alle på gruppen.

Alle rapporter, dokumenter, timeliste og tekniske datablad som følger utstyr vi har brukt skal lagres på gruppens delte rom på Google Drive. Det er tenkt at dette rommet bare skal være tilgjengelig for gruppen og eventuelt veiledere.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Oppdateringsmøte med veiledere vil skje hver andre uke. Disse møtene blir berammet til 30 minutter. Trengs det mer tid ifm. tekniske spørsmål eller lignende skal dette avtales før møtet. Oppdragsgiver kan delta om det sees behov.

7.1.2 Prosjektmøter

Prosjektmøter skal skje hver fredag klokken 13:00. Før klokken 11:00 samme dag skal det leveres inn oppsummering av ukas fremgang av deloppgave ansvarlige slik at en ukesrapport kan fremstilles før møtet. I dette møtet skal eventuelle endringer i fremgangsplan drøftes i forhold til problemstillinger eller uforutsette hendelser som har oppstått underveis i arbeidet, se kap 8.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter

En fremdriftsrapport (også kalt ukesrapport i dette dokumentet) skal leveres til styringsgruppen hver fredag. Denne skal være på formen gitt eksempelvis i Blackboard IE303612 Bacheloroppgave (2019 VÅR)\Undervisningsmateriell\Rapportmaler\Framdriftsrapport (progress report) - eksempel (example)

8 PLANLAGT AVVIKSBEHANDLING

Dersom prosjektets framdrift ikke blir holdt er alternativene å arbeide ut over vanlig arbeidstid i hverdagene eller eventuelt å arbeide i helger for å komme ajour.

For hver arbeidsoppgave er det ansvarlig for denne oppgaven sitt ansvar og gjennomføre innenfor tidsfristen. Vanlig prosedyre dersom dette ikke skulle skje er å gjennomføre en diskusjon i plenum om tiltak for å igjen kunne følge fremdriftsplanen. Deretter er det prosjektleders ansvar om en diskusjon ikke fører til løsning å avgjøre hvilke tiltak som er nødvendig. Der ett eksempel er å utsette sluttdato på deloppgaven.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

For gjennomføring av prosjektet kreves det innkjøp av kamera og markører for sporing av objekter i bølgetanken. Innkjøp av materiale for å lage en testrigg for skalerbar testing av løsningen og feste for kamera på bølgetanken er også nødvendig.

Det er også nødvendig med opplæring i bruk av bølgetanken og vognen for å fjerne risiko for feil bruk eller skade på utstyr eller personell.

10 REFERANSER

Gjelseth, Ørjan; Flatval, Ivan; (2015); "Posisjons- og avstandsmåling med ett enkelt kamera for maritime løfteoperasjoner", NTNU Ålesund
Hartley, R; Zisserman, A; (2004); "Multiple View Geometry in Computer Vision", Cambridge University Press
Solem, Jan Erik; (2012); "Programming Computer Vision with Python", O'Reilly Media inc
Tutorialspoint: Software development life cycles

11 VEDLEGG

Vedlegg A: Gantt-diagram for bachelorgruppe Estimering av modellposisjon og bølgebevegelse i bølgetank.

Appendix E

Appendix: Gantt diagram

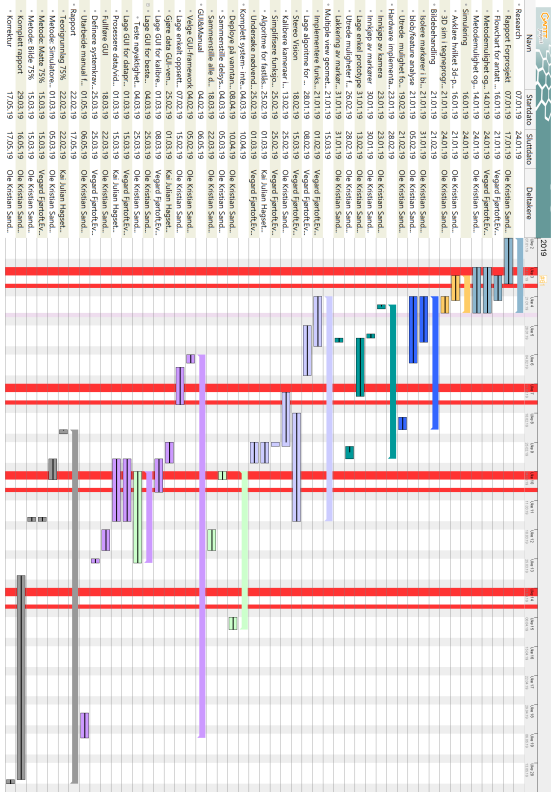


Figure E.1: Gantt diagram for project progress.