Master's thesis

NTNU
Norwegian University of Science and Technology

Jakob Ingvar Utne Midtun

Autonomous Land-Based Seaweed Production

A Proof of Concept for Using a Vision-Guided Robotic System for Handling Substrate Tubes

Master's thesis in Mechanical Engineering
Supervisor: Lars Tingelstad
Co-supervisor: Aurora Tung Nilsen
June 2023

NTNU
Norwegian University of Science and Technology

SINTEF

Jakob Ingvar Utne Midtun

Autonomous Land-Based Seaweed Production

A Proof of Concept for Using a Vision-Guided Robotic System for Handling Substrate Tubes

Master's thesis in Mechanical Engineering
Supervisor: Lars Tingelstad
Co-supervisor: Aurora Tung Nilsen
June 2023

Norwegian University of Science and Technology

Preface

This report is for my master's project, concluding my master's degree in mechanical engineering. The project allowed me to work with product-, code- and systems development and robotics. It was a cool project, and I hope this report is useful if the project is continued.

I would like to thank Birk for getting Vegar and me in contact with SINTEF Ocean. This resulted in me and Vegar being allowed to work as we pleased with an actual robot. This was pretty cool after specializing in robotics for several semesters.

I would also like to thank Vegar for doing great work in the prestudy and early on in the master's period. Without the grippers, it would have been less fun and much more difficult to test all of my software. It was sad to lose you as a project partner, but thankfully the reason keeps you in Trondheim for several years.

Lastly, I would like to thank Aurora and Amund for always allowing me to focus on my own interests, and Lars and Elling for their guidance within robotics. The input all of you have given through the last year has been very valuable.

Sammendrag

Tare er en ressurs med stort potensial. Dagens produksjon er arbeidskrevende og tidkrevende. Hvis et høykostland som Norge ønsker å produsere tare i stor industriell skala, er automatisering essensielt. SINTEF Ocean har utviklet et system for landbasert produksjon av tareplanter som fungerer med manuelt arbeid, men som er ment å gradvis automatiseres ved å teste og implementere ny teknologi. Systemet baserer seg på sylindre som er viklet med tau der taren kan vokse. Disse sylinderene kalles substratrør, og vekstprosessen foregår i vannfylte inkubatorer.

Denne masteroppgaven tar for seg å utvikle og teste et robotisert system for håndtering av disse substratrørene. Prosjektets mål var å skape et "proof of concept" som viser at en robotarm med en spesialdesignet griper kan løse produksjonsoppgaver. Delmålene var å få på plass en fungerende kommunikasjon for å kontrollere en robot og samle inn data fra et 3D-kamera, utvikle et datasynssystem for å veilede roboten, utvikle en eller flere gripere for håndtering av substratrør og å verifisere systemet gjennom testing. Oppgaven med å lokalisere og plukke opp substratrørene fra en inkubator ble ansett som den mest utfordrende oppgaven, og ble derfor hovedfokuset.

De viktigste komponentene i det robotiserte systemet var en DENSO VS-087 robotarm med høy vanntetthet, et Intel RealSense D415 3D-kamera, griperprototyper og datamaskiner med programvare for både bildebehandling og styring av roboten. Et program for rørdeteksjon som benytter punktskyer og referansemarkører ble utviklet i Python. For å kunne bruke kamera-detekterte substratrør, var det også nødvendig med nøyaktig kalibrering av posisjoneringen av kameraet og roboten.

Resultatet var et system i stand til å plukke opp et tilfeldig plassert substratrør i en inkubator. Systemet viste også potensial for å telle og lokalisere flere substratrør i en inkubator. Prototypene av griperene og datasynssystemet beviser muligheten for å bruke et slikt system for håndtering av substratrør, og anbefales videreutviklet for å skape et effektivt system for tareproduksjon.

Abstract

Seaweed is a resource with a large potential. Today's production is labor-intensive and time-consuming. If a high-cost country like Norway wants to produce seaweed on a large industrial scale, automation is key. SINTEF Ocean has developed a system for land-based seaweed seedling production that is functional with manual labor but intended for gradual automation by testing and implementing new technology. The system revolves around cylinders, or tubes, wound with rope on which seaweed seedlings can grow. These cylinders are called substrate tubes, and the growing process occurs in water-filled incubators.

This master's thesis deals with the development and testing of a robotic system for manipulating these substrate tubes. The project's objective was to create a proof of concept that a robotic arm with a specially designed gripper could solve the tasks of the production system. The sub-objectives were to create a functioning signal pipeline to control a robot and gather data from a 3D camera, develop a computer vision system to guide the robot, develop grippers for handling the substrate tubes, and verify the system by conducting tests. The task of localizing and picking up substrate tubes from an incubator was considered the most challenging and therefore became the main focus.

The main components of the robotic system were a DENSO VS-087 robotic manipulator with a high waterproof rating, an Intel RealSense D415 3D camera, prototypes of gripper designs, and computers containing software for both manipulating images and controlling the robot. A tube detection program using 3D data and fiducial markers was created in Python. In order to use the camera-detected substrate tubes, accurate calibration of the positioning of the camera and robot was necessary.

The result was a system capable of picking up a tube arbitrarily placed in an incubator. The system also showed promise for counting and localizing multiple tubes in an incubator. The gripper prototypes and computer vision system both prove the possibility of using such a system for handling the substrate tubes and are recommended for further development to create an efficient system of seaweed production.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Norway has a long tradition of exploiting wild seaweed resources. Manure, feed, and food were some of the earliest uses, going all the way back to the first settlements in Norway [1]. Today, the list of potential usages also includes energy, cosmetics, and medicine. Research is also being done on the efficiency of capturing carbon on an industrial scale using seaweed[1]. The areas rich in wild seaweed are subject to regulations, and the growing demands of the industry reveal the need for production not solely reliant on wild seaweed [2]. This further industrialization of seaweed cultivation was also recommended by HAV21[2], a research and development strategy published by The Research Council Of Norway in 2013.

Automation is a key factor if Norway is to develop large-scale seaweed cultivation. Today's production is time and resource-demanding, and yields volumes in the 100-200 tons per year range. An increase in efficiency by automation and new cultivation methods could result in a potential volume in the megaton range [3].

SINTEF Ocean is one of the main institutions in Norway researching aquaculture. They are part of and have started multiple projects focusing on seaweed. Among these projects is the development of a land-based seedling production system. In an internal project note, researchers at SINTEF Ocean describe a goal of developing a system that is functional with manual labor, but with the opportunities to evolve it by testing and implementing new technology [4]. This system should include the phases of gathering seeds, seeding, growing, and harvesting.

Multiple companies focusing on seaweed cultivation and growing were interviewed in 2018 [5]. Based on this information, researchers of SINTEF have developed a concept for land-based seaweed seedling production. This project focuses on this production method, which includes the seeding and incubation phases.

[1] https://www.dnv.com/news/commencing-carbon-capture-with-seaweed-228139
[2] https://www.forskningsradet.no/siteassets/publikasjoner/1254002444065.pdf

1.1. SINTEF's Production Method

The system is a twine seeding system. Twine seeding is illustrated and compared to direct seeding in Figure 1.1. A thin rope is wound onto a tube, sprayed with seeds, and placed in an incubator for multiple weeks. After the incubation period, the thin rope is spun over to a larger rope. This larger rope is then set to sea, where the seaweed can grow freely while anchoring itself to the thicker rope before being harvested.

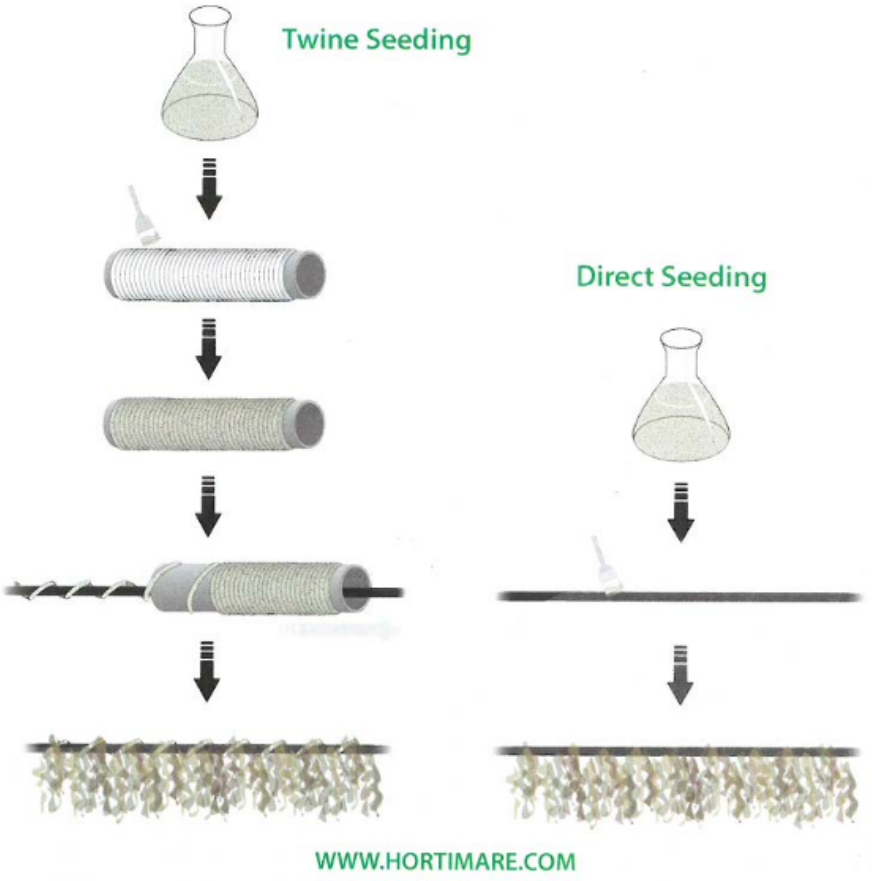


Figure 1.1.: Twine seeding versus direct seeding. Collected from [5] and originally found in a Hortimare flyer.

The incubators in SINTEF's method are rectangular plastic cases filled with water. The measurements of the incubator are listed in Table 1.1. When in use, the incubators are vertically stacked in racks for area efficiency. The incubators are placed in frames with drawer functionality. The drawer functionality eases access to the contents of the incubators.

Table 1.1.: Dimensions of the incubator used in SINTEF Ocean's production system

	Internal [mm]	**External** [mm]
Length	980	1000
Width	710	730
Height	230	240

Figure 1.2 shows an illustration of the substrate tube. The black cylinder is 630mm long, and with the end pieces, it is 704mm. Detailed drawings of the end pieces are included in section A.2. The end pieces consist of a gear and a gripping groove. The gears are essential to rotate the substrate tubes to provide light from a stationary source. The gripping grooves are intended for a robotic gripper to use for handling the tubes. When handling the tubes, only the end pieces must be touched in order to not lose any mass from the rope wound part. The fit of the substrate tubes in the incubator restricts their movement.

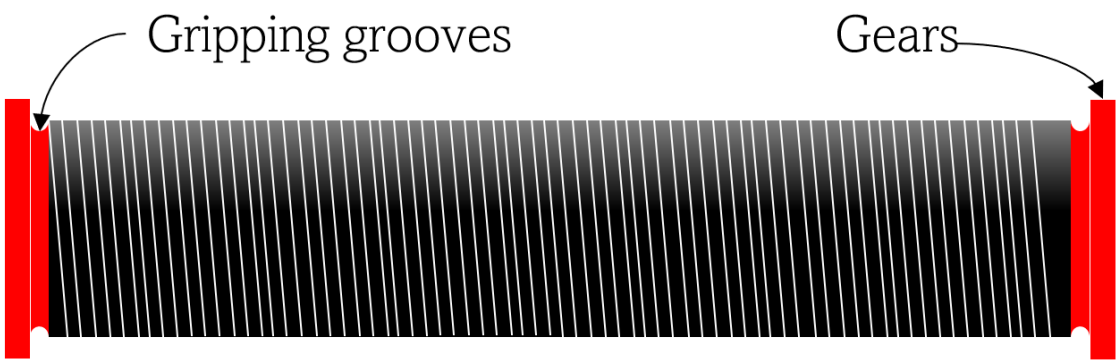


Figure 1.2.: Illustration of a substrate tube including the end pieces. The end pieces consist of a gripping groove and gears for translating rotation while in the incubator.

1.2. Project Objectives

The production method presented in section 1.1 is designed with the possibilities of automation in mind. In the initial project proposal, found in Appendix A.1, SINTEF Ocean proposes the development of a vision-guided robotic system as a way of automating the incubation process. The tasks revolve around the transportation of the substrate tubes between different parts of the incubation process.

A prestudy for this project [6] can be found in the digital appendix. It contains an analysis of the challenges of automating the handling of these substrate tubes and research on possible solutions. The conclusion of the prestudy was that a computer-vision-guided robotic system with a specially designed gripper could be a feasible method for automating the substrate tube handling. The prestudy also concludes on the task of finding and picking the substrate tubes from the incubators would be the most challenging.

The main objective of this project was therefore to develop a proof of concept for a vision-guided robotic system for handling substrate tubes. The system was to interact with the established system presented in section 1.1. The detection, localization, and gripping of the substrate tubes in a water-filled incubator was determined as the main challenge and has therefore been the main focus of this project.

The main objective was divided into the following sub-objectives:

1. Create a functional signal pipeline for controlling the robot and gripper
2. Develop a computer vision system for localizing tubes in the incubator
   (a) Develop a method for detecting a tube in an incubator
   (b) Develop a method for localizing the tubes relative to the robot
3. Develop a functioning gripper prototype
   (a) Prototype the gripper concepts decided in the prestudy [6]
   (b) Continue improvements on one or more of the concepts
      - Create automated control
4. Perform tests indicating the performance of the computer vision system and gripper

1.3. Structure of Report

Chapter 1 - Introduction: The introduction chapter provides the background and motivation for the project, presenting SINTEF's production method and outlining the project objectives.

Chapter 2 - Preliminaries: In this chapter, some background theory is presented to allow for understanding the subsequent chapters. Topics covered include spatial positioning, robotics, computer vision, and mechatronics.

Chapter 3 - Robot Testing Cell: This chapter presents the robot testing cell used in the project. It presents the signal pipeline, most importantly the DENSO VS-087 robotic manipulator, and the Intel RealSense D415 3D camera.

Chapter 4 - Prototyping Grippers: This chapter presents the different grippers prototyped in this project, as well as some initial reflections around the grippers.

Chapter 5 - Computer Vision System: Chapter 5 presents the computer vision system developed in the project. In addition to object detection and object placement, it discusses camera and hand-eye calibration methods.

Chapter 6 - Testing and Experiments: This chapter presents the results of the testing and experiments conducted during the project. It includes sections on verifying camera and hand-eye calibration, experiments with water, and testing of the function of the system as a whole.

Chapter 7 - Discussion: This chapter offers an in-depth discussion of the project's findings and outcomes. It covers the robot cell setup, the performance of the grippers, and the computer vision system, and presents how the system can be used.

Chapter 8 - Conclusion and Future Works: The final chapter concludes on how the project completed its objectives, and presents some recommendations for future work.

Appendices A and B: These appendices contain supplementary information, such as the initial project proposal, dimensions of substrate tube end pieces, and relevant code snippets developed during the project.

Appendix C: This digital appendix is a compressed folder containing the preliminary study, the Python code developed, images, and videos. The code is also found in a GitHub repository linked to in Appendix B.

Chapter 2.

Preliminaries

This chapter describes some necessary information to help understand the contents of this report. Sections 2.1, 2.2, 2.3 and 2.5 cover topics involving spatial positioning, robotics, and computer visioning. Sections 2.6 and 2.7 are relevant for the development and function of the grippers.

2.1. Describing Spatial Positioning

There are 6 degrees of freedom (DOF) to a rigid body's spatial whereabouts, meaning 6 pieces of information is necessary to describe its position and orientation in 3D. 3 DOF for translation, one for each axis, and 3 DOF for rotation, one for a rotation about each axis. The translation is referred to as position, and the rotation is referred to as orientation.

The placement can be described using a cartesian coordinate system, using a vector $p = [x, y, z]^T$ to describe the position of a point on the rigid body relative to an origin. Orientation can be described using different methods, such as Euler angles, and rotation matrices in the special orthogonal group SO(3).

2.1.1. Orientation

Euler angles are a set of three angles. They are typically represented as rotations around the x, y, and z-axes, respectively, and can be used to describe the orientation of an object relative to a fixed coordinate system. Euler angles are easy to understand and compute, but they suffer from a phenomenon known as gimbal lock, where two axes become aligned and the third axis loses its independence. This can make Euler angles unsuitable for certain applications, such as robotics.

The group of rotation matrices, or special orthogonal groups SO(2) and SO(3), also describe orientation. SO(2) describes planar rotations, and a rotation in the plane

is presented as in Equation 2.1. SO(3) is a bit more complex, but rotation about the different axes are as described in Equation 2.2 [7].

$$\text{Rot}(\theta) \in \text{SO}(2) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.1}$$

$$\begin{aligned} \text{Rot}(\hat{x}, \theta) \in \text{SO}(3) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \\ \text{Rot}(\hat{y}, \theta) \in \text{SO}(3) &= \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \\ \text{Rot}(\hat{z}, \theta) \in \text{SO}(3) &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.2}$$

A key property of SO(3) is shown in this example:

Consider 3 3D coordinate frames a, b and c. Make a the reference frame, and its rotation relative to itself presented by an identity matrix, I. Let the rotation from a to b and b to c be R_{ab} and R_{bc} respectively. The relationship between a and c can be presented as in Equation 2.3.

$$R_{ab}R_{bc} = R_{a\not{b}}R_{\not{b}c} = R_{ac} \tag{2.3}$$

R_{ab} denotes the rotation from frame a to b. A rotation from a to b would be ${R_{ab}}^{-1}$.

2.1.2. Transformation Matrices

In this project, homogeneous transformation matrices $\in$ SE(3) have been used for both robot control and the computer vision system. A transformation matrix $\in$ SE(3) is a 4×4 matrix that includes both a translation vector p and a rotation matrix R, and is shown in 2.4.

$$\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{2.4}$$

As with SO(3), homogeneous transformation matrices have the property that $T_{ab}T_{bc} = T_{ac}$, as well as that $T_{ba} = {T_{ab}}^{-1}$. 2.5 shows how one could invert a transformation matrix.

$$\begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix} \tag{2.5}$$

According to [7], the major uses of these transformation matrices are:

- to represent the position and orientation of a rigid body
- to change the reference frame in which a vector or frame is represented
- to displace a vector or frame

2.2. Robot Kinematics

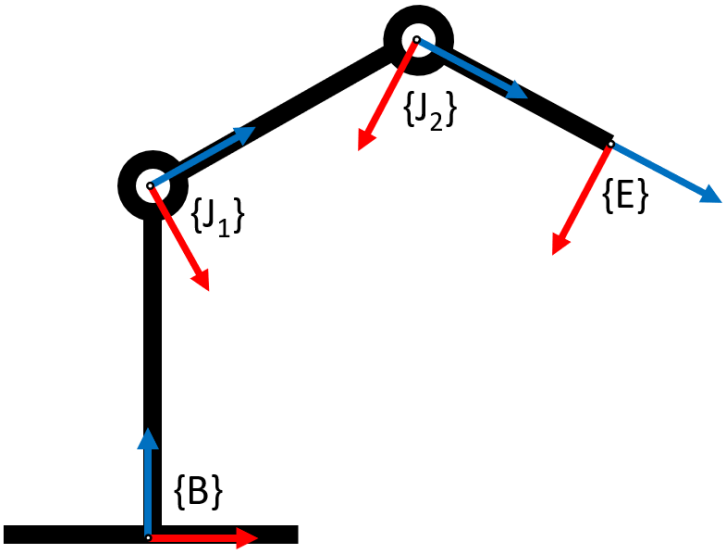


Figure 2.1.: Planar robot with frames. This planar has 2 revolute joints, and it's configuration is determined by the angles of the two joints.

A robot arm is a set of rigid bodies connected by joints, with one end open and one end fixed. The open end of a robotic arm is defined by a tool frame, and the fixed end is defined by a base frame. Figure 2.1 shows a planar robot with the baseframe B, joint frames J_i and end frame E. The relationship T_{BE} can be presented as in Equation 2.6, as stated in subsection 2.1.2.

$$T_{BJ_1}T_{J_1J_2}T_{J_2E} = T_{B\not{J}_1}T_{\not{J}_1\not{J}_2}T_{\not{J}_2E} = T_{BE} \tag{2.6}$$

2.2.1. Task and Configuration Space

Task space and configuration space are two fundamental concepts in robotics describing a robot's mobility and reach.

Task Space

A robot's task space is the sum of all possible positions of the robot's end frame in space. A visualization of this could be a 3D space filled with all possible end-effector frames. For an imaginary robotic arm with no joint constraints, this would be a sphere with $r = \sum d_i$, where d_i is the length of the individual links of the robot.

Constraints that limit this theoretical task space are joint limits, physical objects, and itself. Often a robot controller is equipped with soft joint limits, meaning it is programmed to stop prior to a mechanical joint limit being reached. Constraining movements based on physical objects often need to be determined for the given environment the robot operates. A robot could collide with itself, especially if a tool with larger dimensions is used.

Configuration Space

Configuration space, or C-space, is the space containing all vectors of possible configurations. In the case of the robot in Figure 2.1, the configuration space would include all possible vectors $[\theta_{J_1}, \theta_{J_2}]^T$. A robot's configuration describes the internal state of the robot, including the positions of its individual joints.

2.2.2. Forward and Inverse Kinematics

To control a n-jointed robotic arm, the relationship $T_{BE}(\theta_{J_1}, ..., \theta_{J_n})$ is essential, both to know where the end effector is, and how to configure the joints to reach the wanted position. The calculation of a robot's end frame based on the robot's configuration is called forward kinematics. Calculating the necessary configuration for a robot's end frame to reach a position and orientation is called inverse kinematics.

Both analytical and numerical methods are used for solving a robot's inverse kinematics. Analytical methods aim to find closed-form solutions using mathematical equations, but they may be limited to specific robot configurations. Numerical methods, such as gradient descent or Jacobian-based approaches, iteratively adjust the joint angles until the desired end effector pose is achieved. These methods are widely employed when closed-form solutions are not possible or for complex robotic systems. With today's computational possibilities, numerical solutions can be very efficient despite the iterative approach.

2.2.3. Singularities

Singularities in robotics refer to configurations where the robot arm loses one or more degrees of freedom, limiting its motion and control. An example of this is if two joints' axes align.

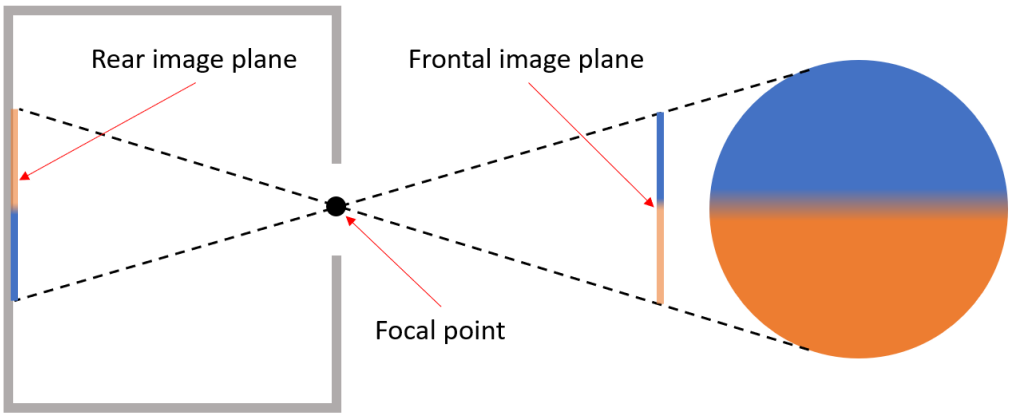
2.3. Computer Vision

The use of cameras and sensors can enable a robotic system to recognize its environment, as well as autonomously detect and locate objects for handling. Modern stereo cameras and technology allow for 3D imaging and could be a powerful tool to attain much information for an autonomous system to use.

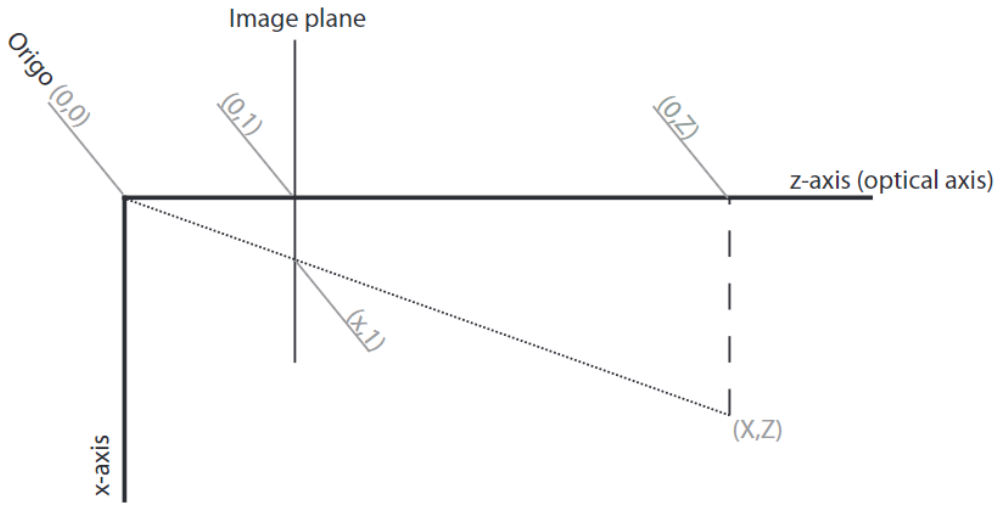
2.3.1. Pinhole Camera Model

The pinhole camera model is a simple model that assumes that light only passes through a small hole and is projected on a plane behind the hole. The image is flipped in the model, but it is no problem for modern digital cameras to "unflip" the images, so it can be equivalent to calculating using the frontal image plane shown in Figure 2.2a.

In Figure 2.2b the distance from the focal point to the image plane is set to 1, and one can see that the point (X, Z) is projected to the point $(x, 1)$. Here it is obvious that $\frac{x}{1} = \frac{X}{Z}$. Expanding this to 3D, it is seen that $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$ since the x and y axes are orthogonal. This can be written generally with the use of homogeneous coordinates. Homogeneous coordinates add a dimension to a coordinate and are especially used in projective geometry. The extra dimension works as a scale so that (x, y) can be represented as $[sx, sy, s]^T$. Making q_i homogeneous and Q_i regular coordinates, Equation 2.7 shows a case where the camera and world coordinate systems are the same. Since this is rarely the case, a translation, and rotation is introduced, making the equation shown in Equation 2.8. The rotation matrix and translation vector are equivalent to the ones introduced in subsection 2.1.2.



(a) Pinhole camera model. The image is flipped in the image plane.



(b) Pinhole camera model with axes and distance 1 from focal point to image plane

Figure 2.2.: Pinhole camera model illustrations

$$
\begin{aligned}
q_i &= Q_i \\
\begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} &= \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} \\
s_i &= Z_i \\
x_i &= \frac{X_i}{Z_i} \\
y_i &= \frac{Y_i}{Z_i}
\end{aligned}
\tag{2.7}
$$

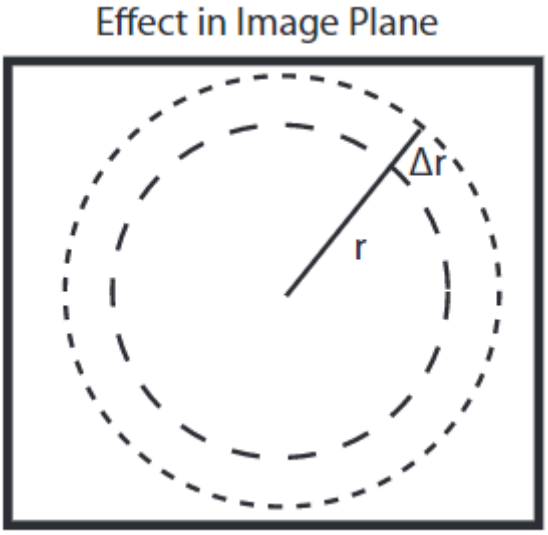


Figure 2.3.: Radial distortion model. The image pixels along the circle with radius r are equally distorted.

$$q_i = [R\ t]Q_i \tag{2.8}$$

One must also include the internal parameters of the camera, such as lenses and sensor size. These parameters are called the camera intrinsic parameters, and the sensor parameters are put in a matrix called the *camera matrix* or K. K is widely used as the linear model shown below, where f is the distance from the image plane to the focal point, called focal length, Δx and Δx are the image coordinates of the optical axis, and α and β describe the affine image deformation.

$$K = \begin{bmatrix} f & \beta f & \Delta x \\ 0 & \alpha f & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Introducing K to Equation 2.8, results in a way of calculating the projection of 3D points Q_i to image coordinates q_i, as shown in Equation 2.10. P is commonly referred to as the projection matrix.

$$q_i = K[R \mid t]Q_i = PQ_i \tag{2.10}$$

$$\Delta(r_i) = k_3 {r_i}^2 + k_5 {r_i}^4 + k_7 {r_i}^6 + ... \tag{2.11}$$

Another addition to the model would be lens distortion, often modeled as a radial

distortion. The image is assumed warped the same along each circle with radius r_i from the center, illustrated in Figure 2.3. Meaning for each point along the circle of r_i, the image points appear closer or further away than in the real world. This is sorted out by undistorting or *warping* the image by finding a polynomial of r_i that matches the distortion, and reversing it. This is commonly done by using a polynomial as shown in Equation 2.11.

[8][9]

2.3.2. Camera Calibration

Calibrating a pinhole camera involves estimating the intrinsic and extrinsic parameters of the camera. The intrinsic parameters are described in subsection 2.3.1, while the extrinsic parameters are the position and orientation of the camera relative to an object. The extrinsic parameters are often described using transformation matrices $\in \mathrm{SE}(3)$ as described in subsection 2.1.2. The intrinsic parameters can be estimated by taking multiple images of a known calibration object, such as a checkerboard, from different viewpoints. The extrinsic parameters can be obtained by solving Equation 2.10 using the same pictures.

In this project, the resources of OpenCV[1][2] have been utilized for the camera calibration. OpenCV is an open-source library for computer vision and image processing. It provides a comprehensive set of functions and tools for calibrating cameras, and there are many Python resources and tutorials available to help users get started with camera calibration in OpenCV.

To calibrate the camera intrinsic parameters using a checkerboard pattern and OpenCV, we can use the `findChessboardCorners()` function to automatically detect the corners of the checkerboard in each image. This function uses a modified Harris corner detection algorithm that is specifically designed for detecting the corners of a chessboard pattern.

We can use the `calibrateCamera()` function to estimate the intrinsic and extrinsic parameters of the camera once we have the pixel coordinates of the corners. This function implements Zhang's method [10]. The `calibrateCamera()` function takes the set of observed chessboard corners, the corresponding world coordinates of the corners, and the dimensions of the image frame as input. The world coordinates are often a model of a set of "perfect" checkerboard corners in the xy plane with one corner in origin. The output of the function is a camera matrix that contains the intrinsic parameters of the camera, a list containing the distortion parameters, and the extrinsic parameters, as presented in subsection 2.3.1. Note that the translation

[1] https://docs.opencv.org/4.7.0/
[2] https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

values and units of the extrinsic parameters are dependent on the dimensions of the world coordinate checkerboard.

The extrinsic parameters can explicitly be calculated by the OpenCV function `solvePnP()`[3], which takes in the pixels of the detected corners, as well as the intrinsic parameters and the "perfect" world coordinate checkerboard. The function uses a Perspective-N-Point method to determine the position of the camera relative to the checkerboard.

The `solvePnP()` and `calibrateCamera()` functions use a nonlinear optimization algorithm to minimize the reprojection error and estimate the intrinsic and extrinsic parameters of the camera.

2.3.3. Point Cloud Generation

To generate point clouds in computer vision applications, stereo imaging is a common method. By capturing two or more images of a scene from different viewpoints, the depth information can be inferred from the difference in the corresponding pixels in each image. Essentially triangulation uses the pixel/3D relationship of the pinhole camera model. This is done by stereo matching, and the resulting depth information can be used to create a point cloud of the scene. In many cases, stereo imaging can provide high-quality 3D information without the need for additional depth sensors.

Intel RealSense D415 is a depth sensor that is used to generate point clouds for various applications. It combines stereo imaging with an infrared projector to produce high-resolution depth maps. The sensor has two cameras that capture images and an infrared projector that emits a pattern of dots onto the scene. The depth information is computed using stereo matching between the two cameras, as well as the phase shift of the dots in the infrared image. The resulting depth map can then be converted to a point cloud using the camera's intrinsic and extrinsic parameters.

The Intel RealSense software development kit[4], with a Python library referred to as `pyrealsense2` or `rs2`, offer multiple solutions for generating point clouds as an `rs2.pointcloud()` object. For this project, Open3D[5] is used for point cloud manipulation. Open3D is an open-source 3D data resource with libraries available in both Python and C++ [11].

[3] https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
[4] https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html
[5] http://www.open3d.org/docs/release/tutorial/data/index.html#pointcloud

2.3.4. Fiducial Markers

Fiducial markers are visual markers that can be used for pose estimation, which involves determining the position and orientation of the camera relative to the marker. One of the advantages of these markers is that they can be designed to have unique identification patterns, which enables easy distinction and tracking of multiple markers simultaneously.

ArUco[6][7] and AprilTag[8] are examples of established fiducial marker systems. The ArUco marker is known for its robustness in challenging conditions, while AprilTag is designed to be robust to scaling and rotation. These markers are widely used in robotics applications such as pick-and-place operations, autonomous navigation, and object recognition. Overall, the use of fiducial markers in robotics provides an efficient and effective means of tracking objects in 3D space, enabling robots to accurately navigate and interact with their environments.

To detect and track markers, a camera is used to capture an image of the scene containing the markers. The image is then processed to identify the marker and extract its unique identifier as well as calculate its frame. This process involves several steps, including image preprocessing, thresholding, contour detection, and pattern matching.

2.4. Clustering

Clustering is a type of unsupervised machine learning technique used for grouping similar data points together based on some similarity criteria. Clustering can be used for various purposes such as data compression, data summarization, anomaly detection, and image segmentation.

The clustering of point clouds is a technique used in computer vision and 3D modeling for grouping points in a 3D space based on some similarity or proximity criteria. This can be done using various algorithms such as K-means, hierarchical clustering, and DBSCAN.

DBSCAN

Density-Based Spatial Clustering of Applications with Noise, or DBSCAN, is a density-based algorithm that groups together points that are close to each other in terms of distance and density. The algorithm defines a neighborhood around each

[6] https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html
[7] https://www.uco.es/investiga/grupos/ava/portfolio/aruco/
[8] https://april.eecs.umich.edu/software/apriltag

point based on a user-defined distance threshold and then groups all points that are within this neighborhood into clusters.

The advantage of using DBSCAN is that it can identify clusters of arbitrary shapes and can also handle noisy data points. The algorithm is defined by two parameters: ϵ, which is the distance threshold, and "minimum points" (minPts), which is the minimum number of points required to form a cluster. The algorithm starts by selecting a point and then expands the neighborhood around it until it reaches the minimum number of points required to form a cluster. The process is repeated until all points are assigned to a cluster or marked as noise.

The DBSCAN algorithm can be mathematically defined as follows [12]:

- Let PC be the set of data points and ϵ and minPts be the distance threshold and the minimum number of points required to form a cluster, respectively. The DBSCAN algorithm can be defined as follows:
- Randomly select a point p from PC.
- If there are at least minPts points within a distance of ϵ from p, then create a new cluster and add all these points to the cluster.
- Expand the cluster by adding all points within a distance of ϵ from any point in the cluster.
- Repeat steps 2 and 3 until the cluster cannot be expanded any further.
- Mark all points in the cluster as visited. Select a new unvisited point and repeat steps 2-5 until all points have been visited. Any unvisited points are marked as noise.

Many Python library resources, including Open3D which is used in this project, provide integrated DBSCAN functions for easy implementation in a system.

2.5. Hand-Eye Calibration

Hand-eye calibration is an important component of a robotic system that involves using sensors to perceive and manipulate objects. This is the calibration of the relationship between the robot's hand (end-effector) and its eye (sensor). By accurately calibrating the hand-eye relationship the robot can perform tasks based on sensor data with a high degree of accuracy.

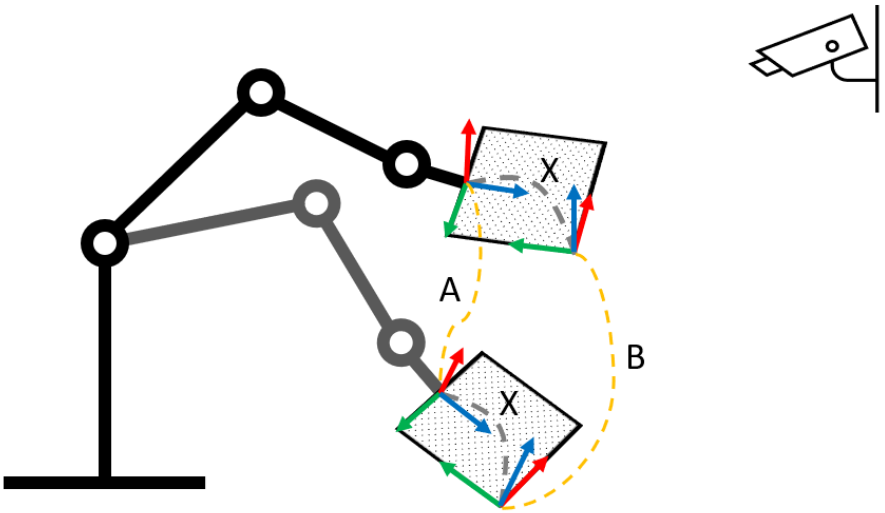


Figure 2.4.: Transformations A, B, and X. A is the transformation between the robot end from position 1 to position 2. B is the transformation between the camera-detected checkerboard frame in the two positions. X is the transformation between the robot end frame and the checkerboard frame and is constant.

In this section, we will present a method that solves $AX = XB$ on the Euclidean group, based on Park and Martin [13]. This is the method used in this project. This method can be used both when a camera is fixed to the robot tool frame or the robot base frame. In the case of this project, it is the latter, which includes fixing a detectable object, such as a checkerboard, to the robot tool frame. By having the robot perform some arbitrary movements and obtaining a set of tool frames, A_i, relative to the base frame, and the corresponding checkerboard frames, B_i relative to the camera, it is possible to calculate X. X is the relationship between the checkerboard and the tool frame. A_i, B_i and X are transformation matrices of SE(3), subsection 2.1.2. This method has several advantages, including its simplicity, robustness, and ability to handle noisy and incomplete data.

First, let us define A_i as the transformation of the tool frame from position $i-1$ to i, and B_i as the checkerboard frame transformation in camera coordinates, as illustrated in Figure 2.4. The transformations are 4×4 transformation matrices in SE(3), consisting of $R \in \mathrm{SO}(3)$ and position vector p.

There is a way of solving this equation exactly and analytically, but due to the relevance of noise, this project has focused on a least-squares method described in Park and Martin [13], revolving around reducing the error criterion on the form seen in Equation 2.12. The d is a metric for calculating the error, and is used as

seen in Equation 2.13. $||\cdot||$ denotes standard Euclidean norm, a way of measuring "length" or "magnitude" of a vector in Euclidean space.

$$\eta = \sum d(A_i, B_i) \tag{2.12}$$

$$d^2(A, B) = ||\log({R_A}^T R_B||^2 + ||p_B - p_A||^2 \tag{2.13}$$

Given the matrices in the $AX = XB \in \mathrm{SE}(3)$, then $R_A R_X = R_X R_B$. This can be recast via logarithmic mapping as shown in Equation 2.14. The logarithmic mapping takes the problem from a nonlinear least-squares minimization problem to a linear least-squares fitting.

$$\begin{aligned} R_X \log(R_B) &= \log(R_A) \\ R_X \beta &= \alpha \end{aligned} \tag{2.14}$$

The new error criterion can be written as Equation 2.15, where $x_1, x_2, ...x_n$ and $y_1, ...y_n$ are given vectors in Euclidean n-space.

$$\eta = \sum_{i=1}^{n} ||Rx_i + p - y_i||^2 \tag{2.15}$$

The data enter η only through the matrices N and M, and centroids $\bar{x}$ and $\bar{y}$

$$\begin{aligned} N &= \sum x_i {x_i}^T \\ M &= \sum x_i {y_i}^T \\ \bar{x} &= (x_1, ..., x_n)/n \\ \bar{y} &= (y_1, ..., y_n)/n \end{aligned} \tag{2.16}$$

The best values of R and p are given by M, and are

$$\begin{aligned} R &= (M^T M)^{-1/2} M^T \\ p &= \bar{y} - R\bar{x} \end{aligned} \tag{2.17}$$

The resulting method is to minimize Equation 2.18. Using the resulting R_X, minimize Equation 2.19 to find p_X. Optimal R_X is given by the Equation 2.17, where $M = \sum \beta_i {\alpha_i}^T$.

$$\eta = \sum_{i=1}^{n} ||R_X \beta_i - \alpha_i||^2 \tag{2.18}$$

$$\eta = \sum_{i=1}^{n} ||(R_{A_i} - I)p_X - R_X p_{B_i} + p_{A_i}||^2 \tag{2.19}$$

The value of p_x is $(C^T C)^{-1} C^T d$, with

$$\begin{aligned} C &= \begin{bmatrix} I - R_{A_1} \\ ... \\ I - R_{A_n} \end{bmatrix} \\ d &= \begin{bmatrix} p_{A_1} - R_X p_{B_1} \\ ... \\ p_{A_n} - R_X p_{B_n} \end{bmatrix} \end{aligned} \tag{2.20}$$

2.6. Mechatronics

Mechatronics is a term used for the development of systems integrating mechanical, electrical, and software components.

2.6.1. Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. With the use of an Arduino board flashed with code from the Arduino IDE, it is possible to rapidly create a system for using actuators and sensors. The programming of an Arduino board is done using a language similar to C/C++.

The Arduino board has a set of analog and digital pins that can be used as both signal input and signal output. The analog pins have a resolution of 1024 and the digital pins are binary by default. However, some digital pins have a resolution of 256 by use of PWM (pulse width modulation). PWM allows for modulation of a binary signal by having the signal "on" for the percentage of time in a period equivalent to the percentage of signal strength wanted. If continuously "on" is 1 and continuously off is 0, then sending an "on" signal for only 50% of the time in a period should result in an average signal of 0.5 [14]. This is given that the frequency of the signal is relatively high, otherwise, the signal could be too alternating. The pins with their respective resolutions can therefore read and write signals from 0V to 5V.

The Arduino Board as a power supply is quite limited, so the need for external power could be necessary, even if the components need less than 5V.

2.6.2. Hall Effect

If a small current is conducted across a thin plate of metal, one could measure a voltage of zero across the plate. The Hall effect occurs when a magnetic field is applied at a right angle to the current, and results in a voltage across the plate. What differentiates the Hall effect from a magnetic induction of a current is its persistence in steady-state conditions, meaning the magnetic field can be constant and the voltage across continues [15]. The example with the thin plate is illustrated in Figure 2.5.

A Hall effect proximity sensor outputs a different value under different magnetic field exposures. This can be used to measure if a magnet is within a certain distance.

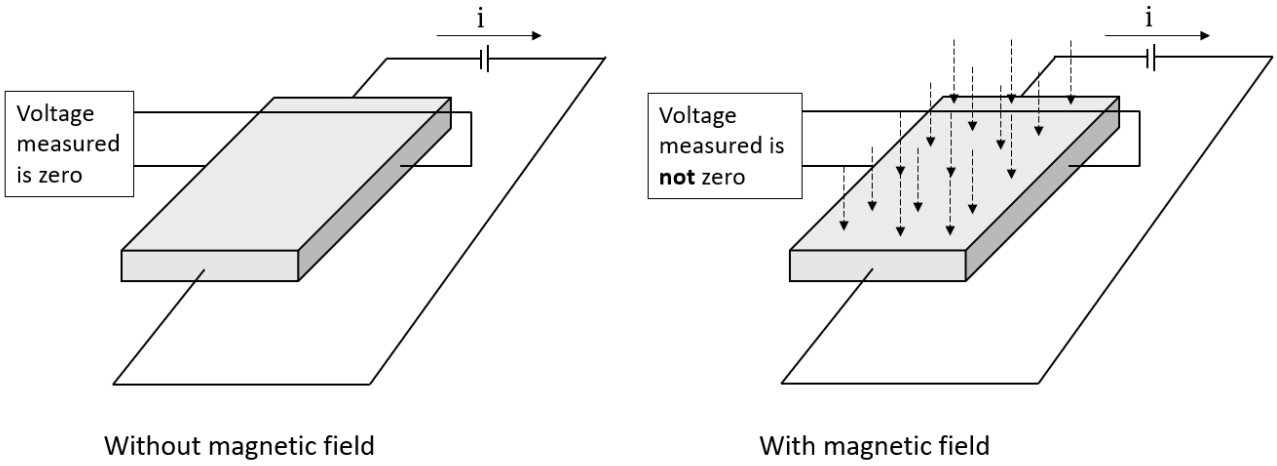


Figure 2.5.: Illustration of the Hall effect. The presence of the magnetic field results in a current orthogonal to the field.

2.7. Electromagnets

Electromagnetism is a fundamental branch of physics that explores the interplay between electricity and magnetism. One practical application of electromagnetism involves creating electromagnets by wrapping conducting wire around an iron core. According to Ampere's law, the magnetic field generated by a current-carrying wire is directly proportional to the current and inversely proportional to the distance from the wire. By coiling the wire into multiple turns or windings, the magnetic field strength can be amplified. This phenomenon is described by Equation 2.21.

$$B = \mu_0 \cdot \frac{N \cdot I}{L} \tag{2.21}$$

B represents the magnetic field strength, μ_0 is the permeability of free space ($\mu_0 =$

$4\pi \times 10^{-7}\,\mathrm{T \cdot m/A}$ for air), N is the number of turns or windings, I is the current flowing through the wire, and L is the length of the solenoid.

Using Equation 2.21, we can control the magnetic field strength by adjusting the number of windings N and the current I flowing through the wire.

Chapter 3.

Robot Testing Cell

The primary laboratory setup used for this project was a robotic testing cell set up with the help of SINTEF Ocean in their facilities, pictured in Figure 3.1. The setup was an attempt to mimic a scenario from the production method described in section 1.1. In this chapter, the robotic cell and its components are described, as well as the signal pipeline. The key components of the cell are listed below. The grippers used in this project are presented in chapter 4.

- DENSO VS-087 robotic arm
  - Teach pendant
  - Robot controller
- Windows computer with LabView
- Windows computer with Python and DENSO library
- Intel RealSense D415 3D camera
  - Camera Mount
- Gripper prototype
  - Arduino
  - Power supply
- Incubator
  - Substrate tube
  - Pillow to simulate flotation
  - Fiducial markers for calibration

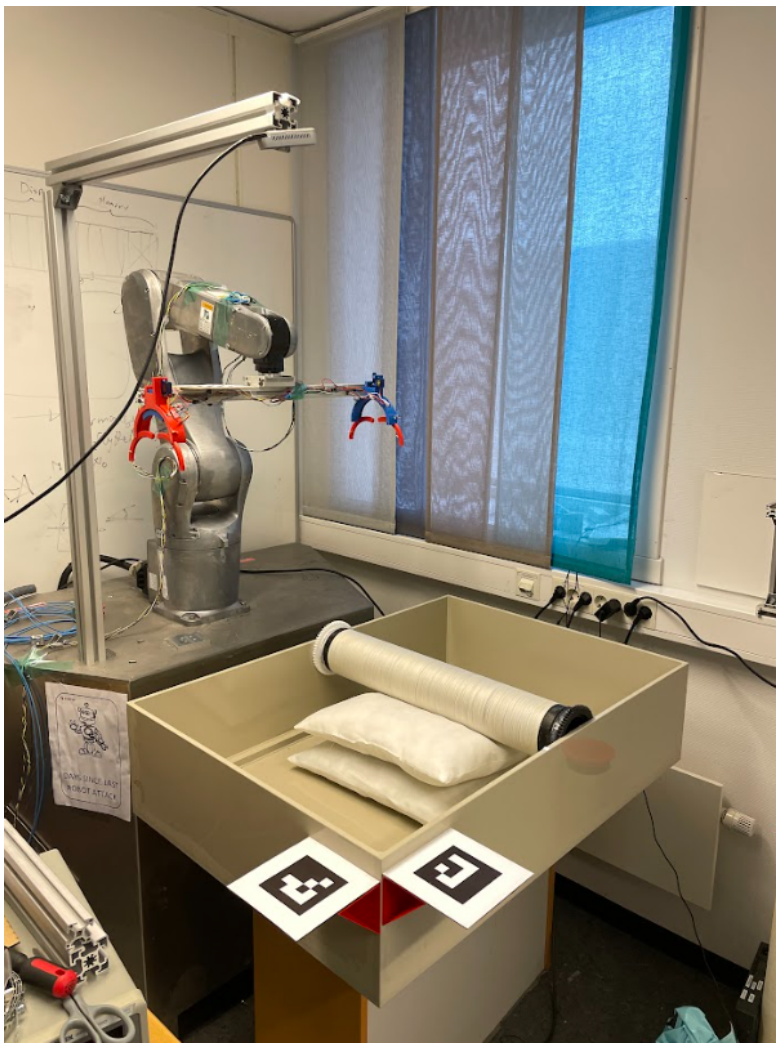


Figure 3.1.: Picture of robot testing cell. Here the robot is equipped with the folding fingers gripper.

3.1. Signal Pipeline

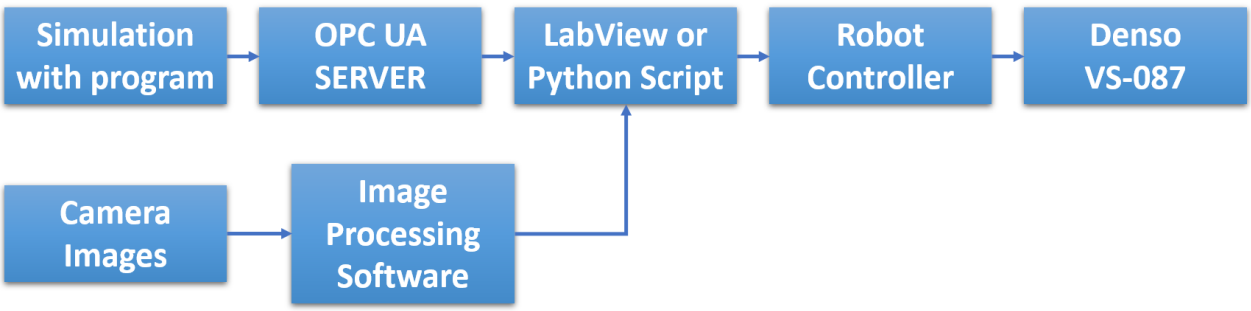


Figure 3.2.: Pipeline suggestion created in the prestudy. At this time, simulation and programming using Visual Components were considered.

In the prestudy [6] the signal pipeline was suggested as illustrated in Figure 3.2. The simulation was excluded as a robot unit was provided by SINTEF Ocean for the duration of the project period. The final pipeline for the robot testing cell is presented with components in Figure 3.3. The key differences between the pipeline suggestion from the prestudy and the final pipeline are:

- No simulation or OPC-UA
- LabView AND Python program on separate computers
- Image processing is done in a Python program on the same computer as the robot program
- Gripper and control of gripper included in the pipeline

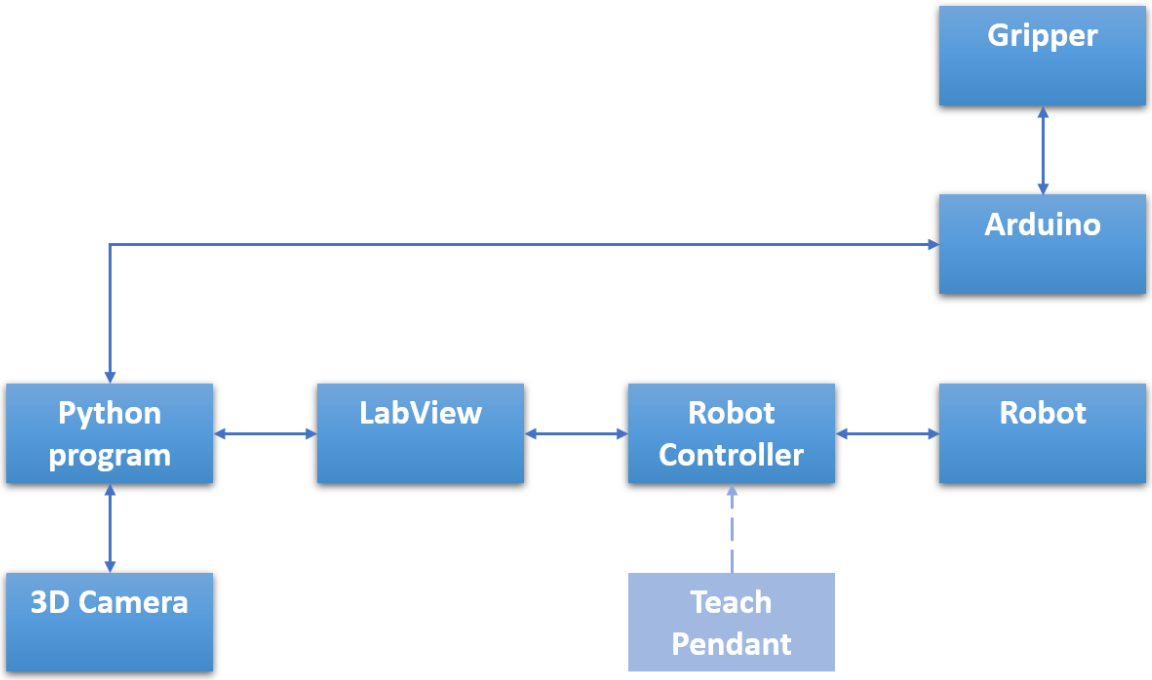


Figure 3.3.: Signal pipeline for the robotic system. The teach pendant is an additional tool for moving the robot manually.

3.2. Robotic Manipulator

This section presents the robot used in this project, the DENSO VS-087, as well as how the robot is controlled.

3.2.1. DENSO VS-087 Robotic Manipulator

The DENSO VS-087 robotic manipulator is a 6-axis robot arm with six revolute joints designed for use in industrial applications, such as assembly, packaging, and material handling. Due to its compact design and advanced control system, the VS-087 can work in tight spaces. Additionally, this robot is equipped with a variety of safety features, such as collision detection and emergency stop buttons. It has a producer-stated repeatability of ±0.03mm. The choice of this specific model of the robot is discussed in the prestudy [6] but the main factors were availability, payload, and its rating for waterproofness[1]. Unlike collaborative robots, this manipulator is not intended to work alongside human workers.

3.2.2. Robotic Control

The DENSO VS-087 robotic manipulator can be controlled using a Python library provided by SINTEF Ocean. The Python program allows users to define a desired position and motion. When executing the program, it communicates with a computer running LabView over an ethernet cable. LabView serves as an intermediary between the Python code and the robot controller[2].

Additionally, the robot can be controlled using a teach pendant that is directly connected to the controller. The teach pendant provides users with various options for controlling the robot's movements. Options are to move the robot's end-effector relative to the base frame or tool frame or to directly control each joint.

The controller is capable of numerically solving the robot's inverse kinematics, section 2.2. This allows users to send positions on the forms presented in Table 3.1. The controller is also capable of calculating linear paths, as used for this project. As many positions have multiple possible solutions, the DENSO controller allows users to add a "figure" parameter that determines what configuration should be used. By not having to solve the inverse kinematics, the workload is reduced considerably in terms of working with the robot.

Table 3.1.: Different types of poses to describe a robot pose for the DENSO unit. R_x, R_y, and R_z are Euler angles and the "figure" is an integer describing if the desired configuration is over/under-elbow, over/under-wrist, etc.

Pose type	**Parameters**
Joint	J_1, J_2, J_3, J_4, J_5, J_6
Cartesian	x, y, z, R_x, R_y, R_z, figure
Transformation matrix	$\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$, figure

[1] https://www.denso-wave.com/en/robot/product/five-six/vs068-087.html
[2] https://www.densorobotics.com/products/software/labview/

The Python library allows the user to establish a session with the computer containing LabView, and establish parameters for running the robot, such as movement speed and defining tool frames relative to the mounting plate at the end of the robot. The code used for establishing a connection is shown in Appendix B.1.

`move_to_cart(pose, interpolation=Interpolation.Move_L)` and `get_position_transformation_matrix()` are examples of how to use the different pose types using the DENSO controller library. `Interpolation.Move_L` is the parameter for moving linearly to the destination. The matrix in the last row of Table 3.1 is created and retrieved as a 4×4 NumPy array[3]. When establishing a position for the robot to move to, the type of motion is determined by the user. For this project, the linear interpolation movement is used, as it is easier to predict the movement in order to avoid collisions. When using linear movement, there is a risk of getting a joint configuration near a singularity. The robot controller is programmed to stop when a joint exceeds a certain angular velocity, as can occur when approaching a singularity.

3.3. Intel RealSense D415 3D Camera

The Intel RealSense D415[4] was used for both 2D and 3D imaging in this project. The camera is small and compact, measuring 99mm × 20mm × 23mm and weighing 72g. The current price tag for the model directly from Intel is $272. The camera was chosen mainly due to there being a unit available for the duration of the project period.

The camera allows for capturing both 2D color images and depth data making it capable of producing colored/textured 3D point clouds. The camera was connected to the computer running the Python programs, and the capturing of images and point clouds was done using RealSense's Python library resources. The process of creating point clouds using the camera is presented in subsection 2.3.3. The resolution of image capturing is limited if not using USB 3.0 or newer. The camera was mounted and fixed to the robot base frame for the final stages of the project period.

3.4. Incubator and Tube

The incubator and tube are as described in section 1.1. An incubator was placed on a table to provide a suitable work height. Fiducial markers, subsection 2.3.4,

[3] https://numpy.org/doc/stable/reference/generated/numpy.array.html
[4] https://www.intelrealsense.com/download/20289/?tmstv=1680149335

were mounted on the incubator as a way of calibrating the position of the incubator relative to the camera.

The room containing the testing cell did not allow for filling the incubator with water. The tubes were therefore placed on pillows inside the incubator to simulate flotation, reducing the risk of the gripper colliding with the floor of the incubator during gripping. When not filled with water, the incubator walls cave slightly in, making the middle of the incubator slightly more narrow.

The reach of the robot did not allow for the incubator to be oriented with the short side toward the robot. For the tubes to be reachable in all positions in the incubator, the incubator needs to be placed with its long side close to the robot base, as pictured in Figure 3.1.

Chapter 4.

Prototyping Grippers

As stated in the original project proposal by SINTEF Ocean, section A.1, a special gripper was necessary to perform the hatchery tasks using a robotic manipulator. Various gripper designs were thoroughly discussed in the prestudy [6]. There were two participants in the prestudy and at the beginning of this project and the gripper development and computer vision system development shared the same priority. Due to a reduction in participants, the focus of this project shifted away from gripper development. However, the need for a gripper was critical to further develop and test the robotic system as a detailed simulation was excluded in favor of physical testing.

Some gripper concepts were prototyped before the reduction in participants. Vegar Stubberud, a co-author of the prestudy, was the main researcher on the gripper design, as well as the main developer of the grippers presented in this chapter. However, the prototyping was a joint effort and therefore included in this report.

All the grippers in this project are based on two gripping units placed on an extruded aluminum profile to easily adjust the positioning of the units. The gripping units are designed and placed to fit in the gripping grooves of the substrate tubes presented in section 1.1.

4.1. Minimum Viable Product - Servo Electric Gripper

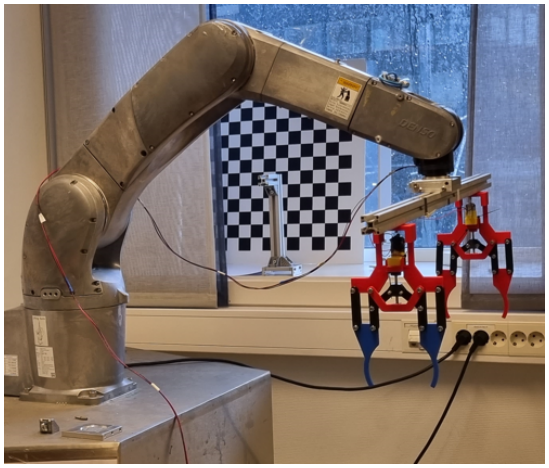
A feasible gripper type was a servo-electric gripper. In the prestudy [6] a type of servo-electric gripper pictured in Figure 4.1a was explored. This type of mechanism relies on the actuator maintaining a gripping force. A solution to this was inspired by the gripper in Figure 4.1b, where a leading screw was attached to an actuator, and as the screw rotates, a component with threads is forced up or down, resulting in the opening/closing of the gripper. When gripping, the force of friction in the threads would allow the gripper to stay in position without the force of the actuator.

(a) Early servo electric prototype

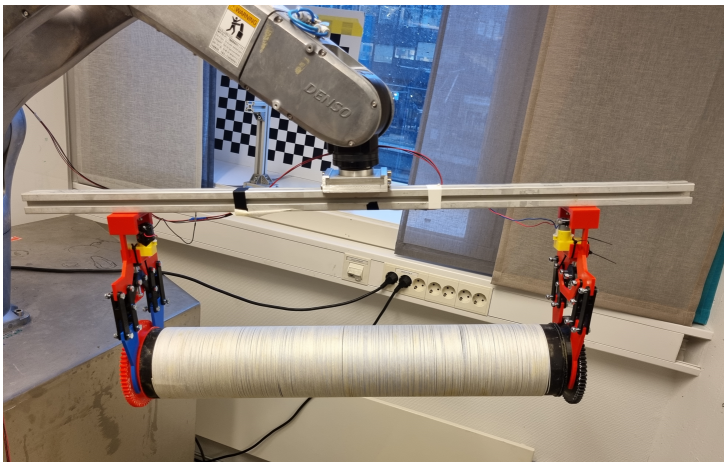
(b) Example of leading screw mechanism

Figure 4.1.: Servo electric grippers

With the objective of rapidly creating a full-scale functioning gripper for testing with the robot, the servo-electric gripper pictured in Figure 4.2 was built. This gripper allowed for early testing with the robot. The gripper was made using 3D printed components, using nuts and bolts as joints and a generic DC motor as an actuator. The gripper consists of two gripper units, mounted on an aluminum profile. The mechatronic control of the actuators is done using an Arduino board, presented in subsection 2.6.1. The simple programming of this setup is time-based, meaning the DC motors were set to rotate for a certain amount of time. The amount of time when "closing" was set higher than the amount of time "opening", resulting in the motor stalling and the grip being as tight as the DC motors would allow.



(a) Gripper mounted on robot



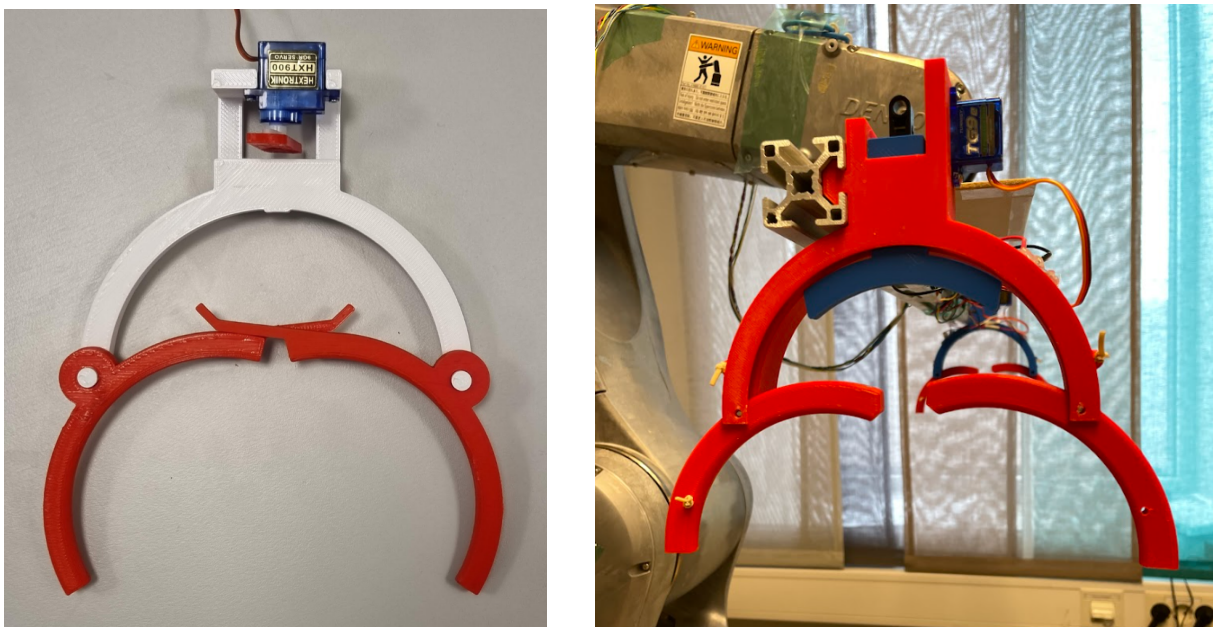
(b) Gripper gripping the tube

Figure 4.2.: First functioning gripper prototype

4.2. Folding Fingers Gripper

The folding fingers gripper is a design in which curved fingers are pushed to fold around the floating tube. An early prototype of the concept is pictured in Figure 4.3a. This option was favorable from the early stages, as the mechanism allowed for a sleek design without the need for any fast or powerful actuators. The rotating lock as seen in Figure 4.3a was developed already in the prestudy. When building the next prototype, the main change was the locking mechanism. A design with less risk of slipping was made that involved sliding a wedge between the fingers using a rack and pinion. A Lego gear and rack were used in the prototype. A feature to automatically place fingers in a maximum open position was added by using rubber bands to force them open. The new improved design is pictured in Figure 4.3.

The mounting of the gripping units was now done on the side of the aluminum profile, reducing the distance from the end of the robot to the gripping point. The result is quite compact and the rigidity of the folding fingers gripper is seemingly much better than the servo-electric gripper described in 4.1.



(a) Early prototype of the folding fingers gripper unit

(b) New iteration of the folding fingers gripper unit

Figure 4.3.: Pictures of the folding fingers gripper

4.2.1. Control of Folding Fingers Gripper

As with the servo-electric gripper, section 4.1, the actuators of the folding fingers gripper were controlled by an Arduino board, subsection 2.6.1. The actuators that control the rack and pinion are common 9g servo motors, connected to the Arduino

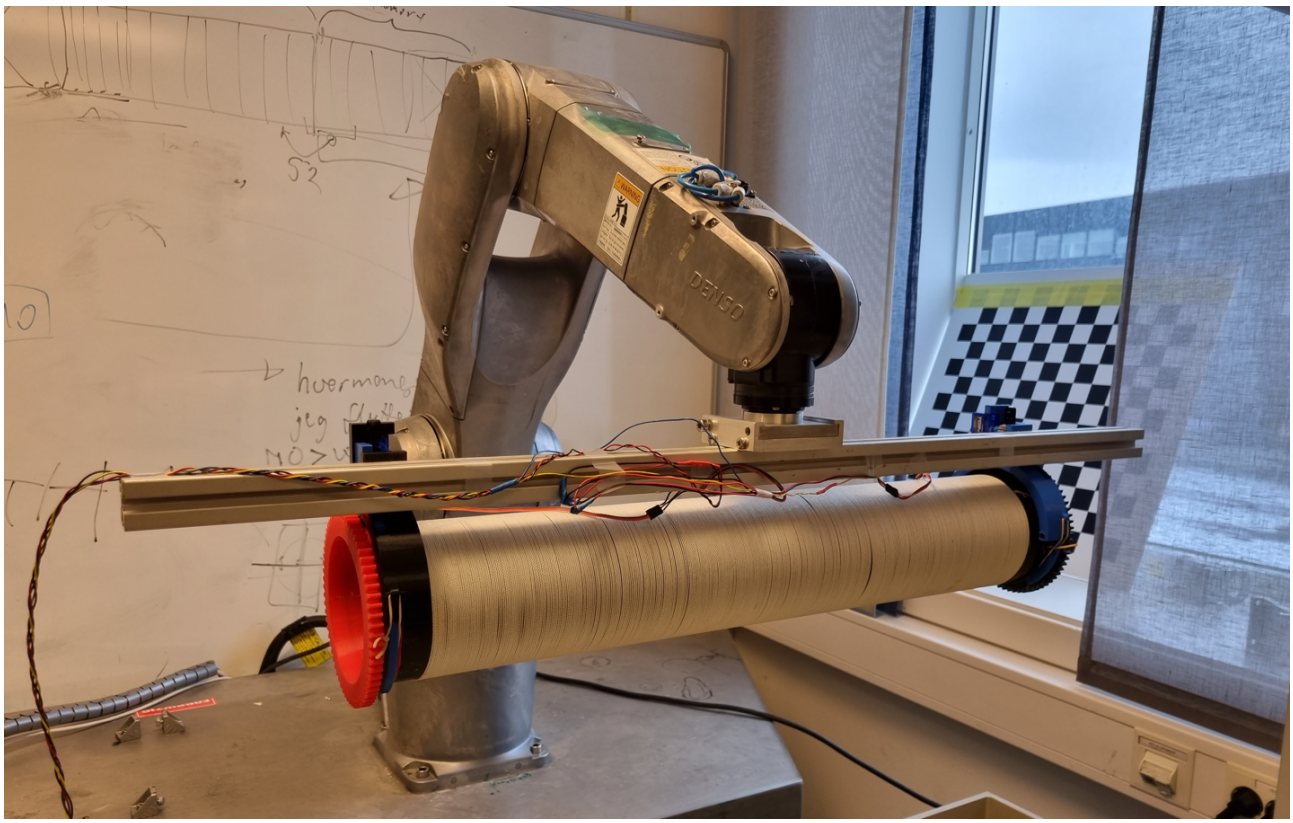


Figure 4.4.: Tube lifted using the folding fingers gripper.

by PWM pins, as well as to an external power supply providing $5V$. By using the "Servo" library in the Arduino IDE, the servo motors can be set to multiple repeatable positions between approximately 0 and 180 degrees. By finding a servo position for "open" and "closed" for each gripper unit, these positions can easily be called when wanting to open or lock the folding fingers. By having the Arduino board connected by USB to the computer running the program for the robotic system, the Arduino gets input for when to change the position of the servo motors.

The Arduino based gripper controller is connected by USB to the same computer that runs the robot commands. The communication is done serially.

Sensor Feedback

As mentioned in section 4.1, sensor feedback is desired. It is quite beneficial to have information on if the gripper is gripping an object or not. If the object is gripped, this could signal the system to move into the state of lifting. If the object is not gripped when it is supposed to, it could signal the system to try again. In the case of the folding fingers gripper in this project, where the fingers are closed when pushed onto a floating tube, sensor feedback could be used to push further on the floating tube in a continued attempt to grip it.

The folding fingers gripper prototype was equipped with Hall effect proximity sensors, described in subsection 2.6.2, as well as magnets. The magnets were placed in the fingers, and the sensor was on the main part. As the fingers are pushed closer to the sensors, the output of the sensor changes, and at a distance close enough for the gripper to initiate the locking mechanisms, the servo motors can be triggered. Figure 4.5 illustrates the placements of the sensors and magnets. This allows for knowing if all four fingers are in a close enough position to initiate the locking mechanism.

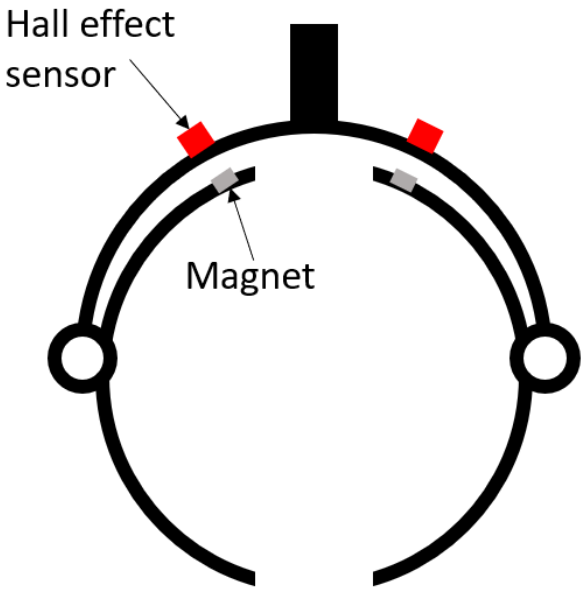


Figure 4.5.: An illustration of the placements of the Hall effects sensors on a folding fingers gripper unit.

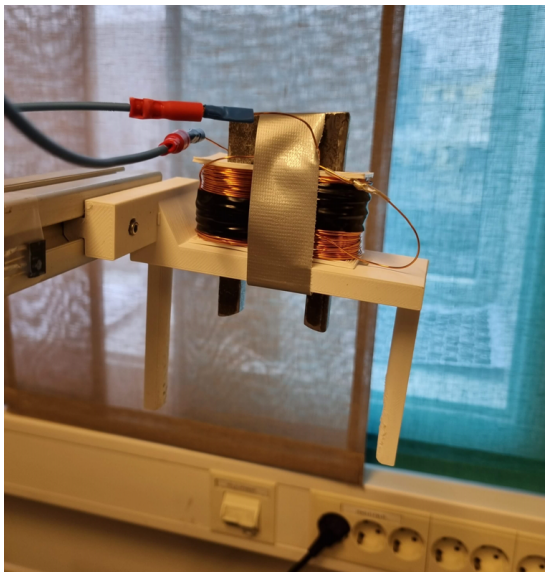
4.3. Electromagnetic Gripper

As explored in the prestudy [6] an electromagnetic gripper was also an option. One was prototyped by making two u-shaped electromagnets with a 3D-printed frame as pictured in Figure 4.6a. The electromagnets have sloped ends to increase the contact with the cylindrical gripping area of the tubes. To reduce the forces applied to the magnets, the gripper includes a frame, the 3D printed white plastic part seen in Figure 4.6a. The frame works as a guide to get in contact with the magnets. The frame also helps to keep the tube in place while the tube is being maneuvered.

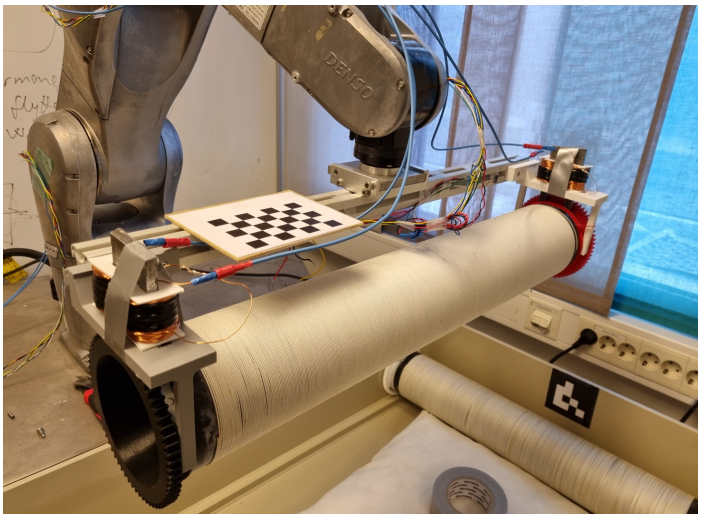
For the electromagnetic gripper to function, modifications would need to be done to the substrate tubes' end pieces. They would need to be equipped with a magnetic material. For the testing of this gripper, some generic patent tape was used. This was a cheap and efficient solution. Figure 4.7 shows the electromagnet in contact

with the patent tape.

No automatic control system nor sensor feedback was implemented. In order to automate the control of this gripper, a switch controlling the current could be implemented. A type of sensor feedback would also be beneficial. The idea of monitoring the current to see if a detectable change in current occurs when in contact with a magnetic surface was discussed in the prototyping phase.



(a) Electromagnetic gripper unit



(b) Lifting the tube using the electromagnetic gripper

Figure 4.6.: Electromagnetic gripper prototype.

Figure 4.8 shows how the electromagnet is put together. 60 meters of wire were spun on a magnet, resulting in ≈350 windings per 3D printed wire holder. The wires are isolated with a thin coating to make the current travel the full distance of the windings. As presented in section 2.7, the electromagnetic force can be altered by changing the current and the number of coiled turns of wire around the piece of iron. A current of $1.5 - 2$A was provided from an external power supply and provided a suitable lifting force. When attempting to manually pull the tube from the gripper, a considerable force was needed.

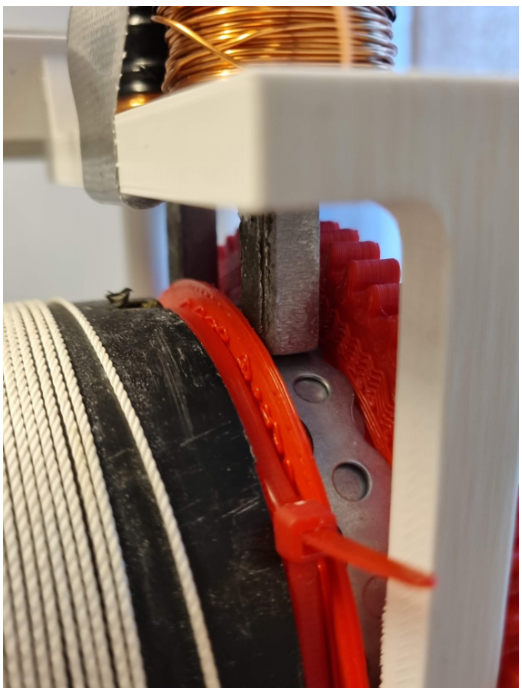


Figure 4.7.: The electromagnet in contact with the patent tape.

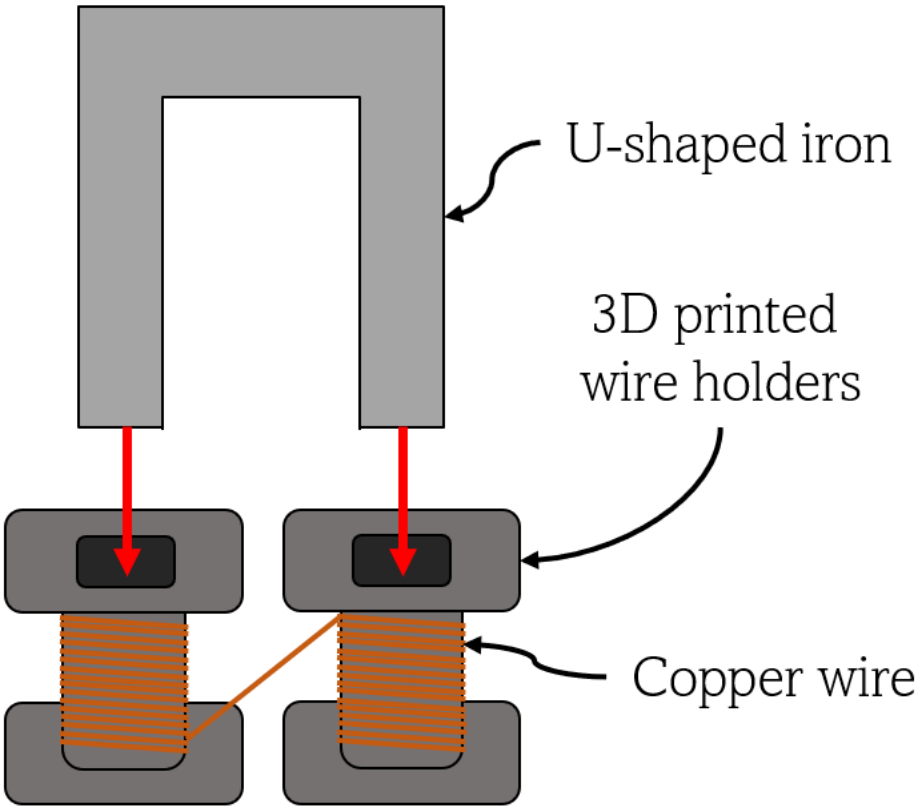


Figure 4.8.: The coiled 3D printed wire holders are placed on the U-shaped iron piece. When current travels through the wires, the iron will become magnetic.

Chapter 5.

Computer Vision System

As the tubes used in SINTEF's production method described in section 1.1 float inside the incubator, their position cannot be pre-programmed, and a detection system is required to pick them up. This chapter describes the development of such a system using fiducial markers and a 3D camera. This system detects and localizes the substrate tubes, and is one of the primary sub-objectives described in section 1.2.

The system contains the following steps:

1. Detect the position and orientation of the incubator using fiducial markers.
2. Create a point cloud of the incubator and use the information on the incubator to crop the point cloud to only include the contents of the incubator, i.e. the tubes.
3. Cluster the contents of the incubator using DBSCAN to isolate the individual tubes.
4. Use the information of the incubator and point clouds to determine a tube frame.

The resulting frame is visualized in Figure 5.1. The red part is the detected tube, and the coordinate system is used to position the robot tool to grip the substrate tube. In Appendix section B.4 a step-by-step visualization is provided.

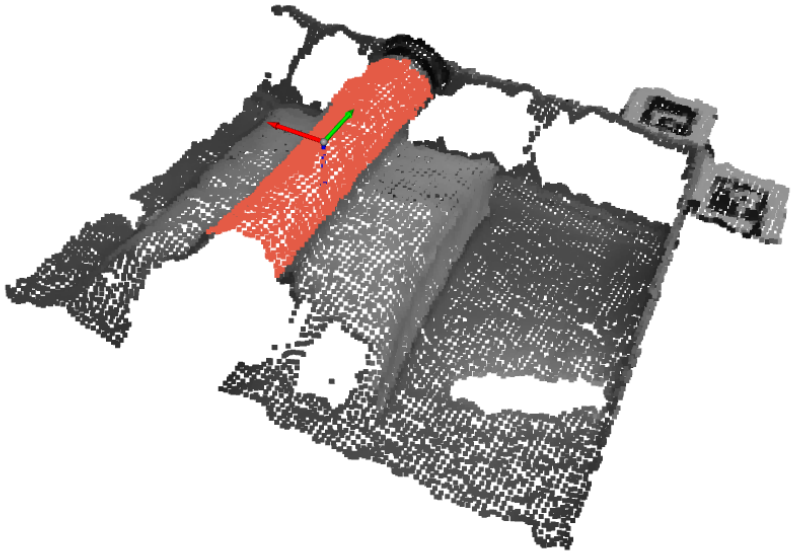


Figure 5.1.: A tube isolated by clustering and positioned by placing a coordinate system at the center top point. The coordinate system is orthogonal to the marker's orientation. The tube is laying on a pillow to simulate flotation and prevent collision.

In addition to the object detection algorithm, this chapter includes a section on the calibrating of the camera intrinsics parameters, as well as the hand-eye calibration. Both of these calibrations are important for the computer vision system to function.

5.1. Object Detection

The object detection method in this project revolves around a point cloud generated by an Intel RealSense D415 camera, the Intel RealSense SDK, Open3D, and OpenCV. This section is divided into subsections that describe the techniques used to detect and locate tubes in an incubator. The code developed for detecting the object and determining the positions is found in Appendix B.3.

5.1.1. Point Cloud Generation

By using the Intel RealSense SDK in a Python program, a stream of depth information from the stereo and infrared capabilities of the camera is opened. The camera software combines the data from the stereo and infrared emitter/sensor into a single depth image.

Simultaneous as the depth stream is opened, a color stream is opened. This color camera sensor is at a different location than the depth image origin. The RealSense library allows for aligning the depth and color streams, and due to the later use of color imaging in fiducial marker detection, the depth stream is aligned to the color stream, making the color stream origin the origin.

The color data and depth data are combined into an Open3D RGBD image (color and depth image), which is then used to create an Open3D point cloud object. The Open3D function `create_from_color_and_depth()` takes the RGBD and camera intrinsics as input to create an Open3d point cloud object. The code for generating an Open3D point cloud using an Intel RealSense snapshot is shown in Appendix B.2.

5.1.2. Cropping

The incubator is a rectangular prism-shaped box. This allows for isolating the contents by cropping everything outside the inner planes of the walls, as illustrated in Figure 5.2. By removing everything outside the red lines, the contents of the incubator would be the only remaining parts.

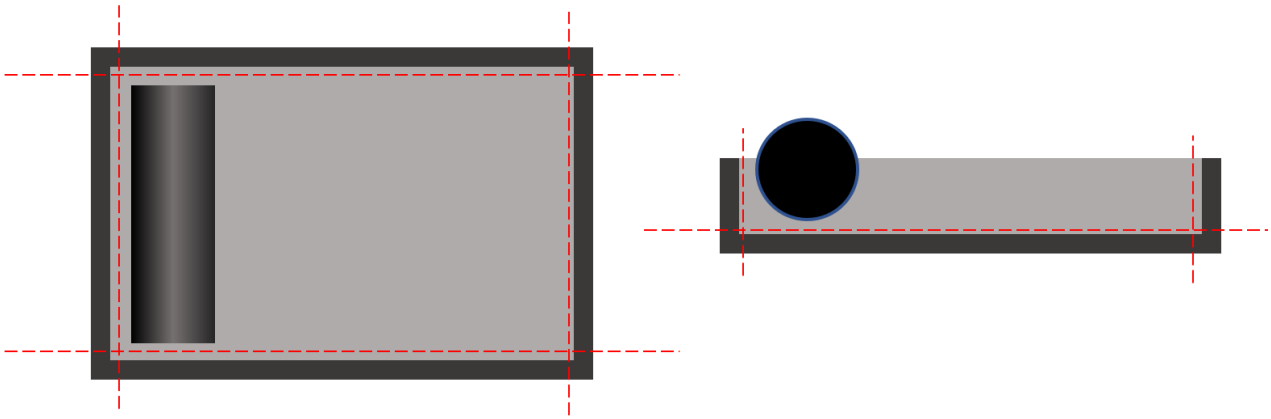


Figure 5.2.: Cropping of point cloud

In the early stages of development, the camera was mounted to a stand fixed to the incubator. The camera was attempted mounted orthogonal to the incubator floor and walls, with parts of all walls visible to the camera. This resulted in all walls having one coordinate approximately constant. These coordinates were found in the Intel RealSense Viewer software, as shown in figure Figure 5.3. These coordinates were then used to crop the point cloud to only include the contents of the incubator. This method revolved around an accurate mounting of the camera, which was difficult and not robust to movements. By introducing fiducial markers,

described in 2.3.4, to the incubator, the incubator position and orientation could be calibrated from an arbitrary camera position and orientation.

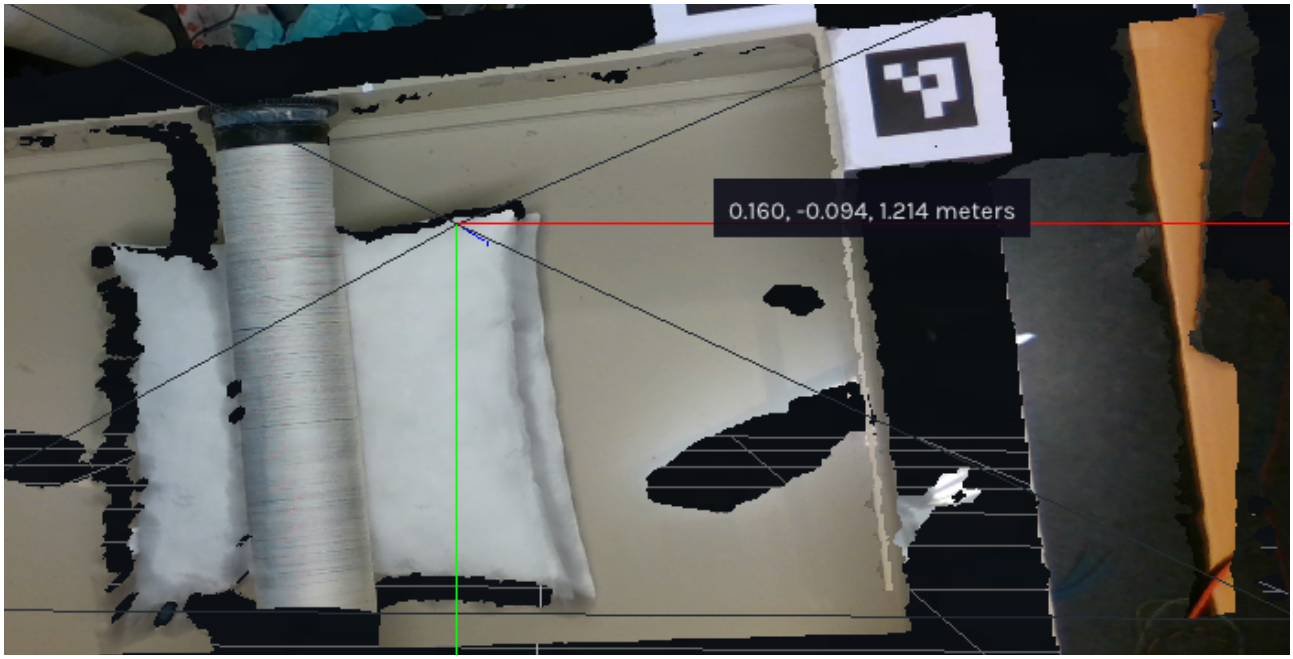


Figure 5.3.: Intel RealSense Viewer showing coordinate relative to camera frame

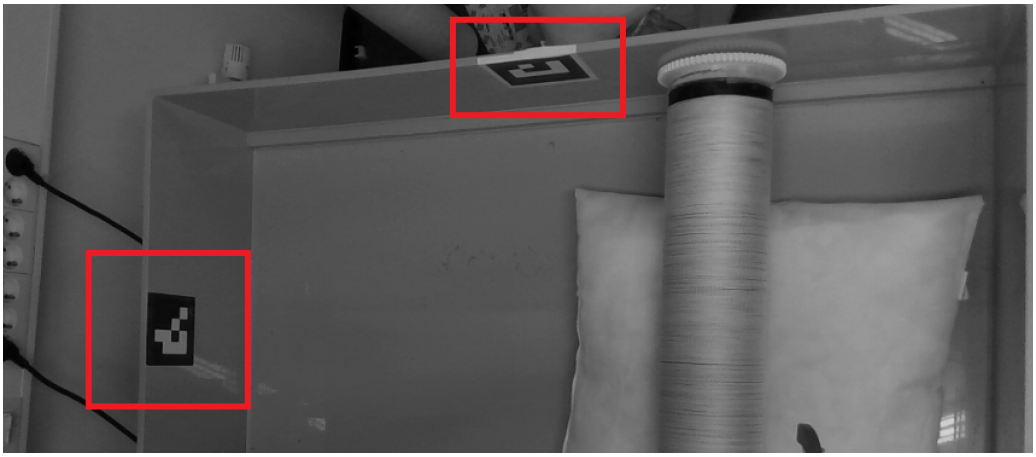
Two unique ArUco fiducial markers were created using OpenCV[1], and printed on paper. One was placed inside the short wall and one inside the long wall as shown in Figure 5.4a. By detecting the markers and retrieving their position and orientation, i.e. their frames, the cropping only relied on the markers being visible to the camera. The origin of each marker frame was assumed planar with its respective inside wall, and the z-axis of each marker was assumed normal to the wall. The point cloud was transformed into the frames of the markers, and cropped by the z-coordinates and internal dimensions of the incubator.

The use of fiducial markers would allow the system to work as long as both markers were visible to the camera. This possibility of determining the incubator position and orientation led to the camera being fixed to the robot base frame instead of the incubator. This new fixing was preferred due to the reduced amount of hand-eye calibrations, section 5.2, necessary.

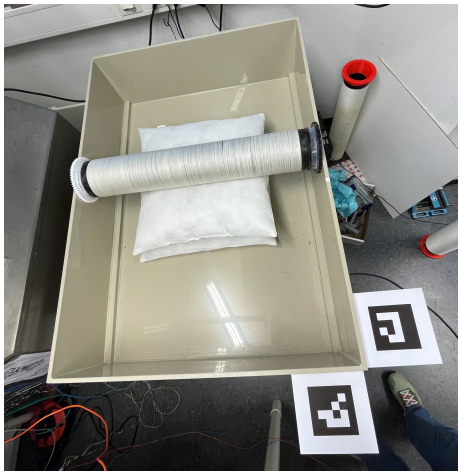
The fiducial markers were eventually placed on the outside of the incubator, as tubes occluded the markers in certain positions. The changing of the location of the markers was also done as the incubator will be filled with water and the light refraction on the potentially uneven water surface could present inaccuracies in reading the markers. The latest placement of the fiducial markers is shown in

[1] https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

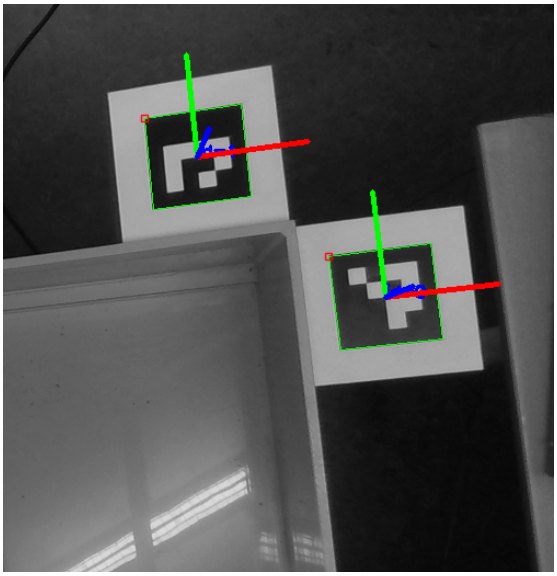
Figure 5.4b, and is an attempt to place the markers in the horizontal plane with the x and y axes orthogonal to the walls of the incubator.



(a) Markers placed in the incubator



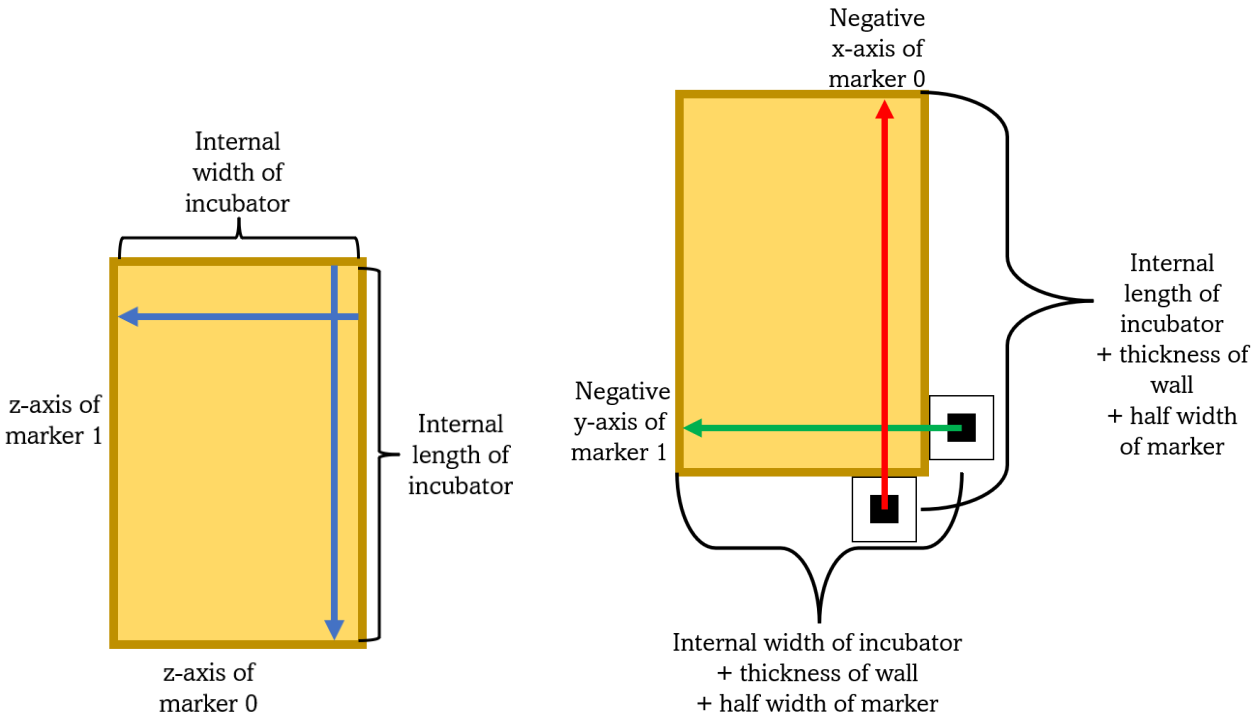
(b) Final placement of markers



(c) Markers with visualized coordinate systems

Figure 5.4.: Placement of ArUco markers

The new mounting of the markers demanded new parameters for the cropping, as the markers would no longer be in the same plane as the incubator walls. Figure 5.4c shows the new coordinate systems of the markers where x, y, and z are the red, green, and blue axes, respectively. Figure 5.5, shows how the cropping parameters in the original placement only relied on the internal dimensions of the incubator, while the new placements rely on the dimensions of the marker and surrounding frame.



(a) Necessary distances when the markers are placed inside

(b) Necessary distances when the markers are placed outside

Figure 5.5.: The different cropping parameters of different marker placements.

The vertical cropping parameters were made using the z-axis of one of the markers (y-axis when markers were inside the incubator). The cropping was initially done by removing all points below $z_{\text{avg}} + d$, z_{avg} is the average z-value of the points, and d is a distance in mm. If the incubator is empty, and the points representing the floor are noisy, the points above z_{avg} could still result in a detectable point cloud. The offset d was to account for this. Later testing revealed that a better approach could be to remove all points with z-value outside $[z_{\max} - d, z_{\max}]$.

Exaggerating the Cropping

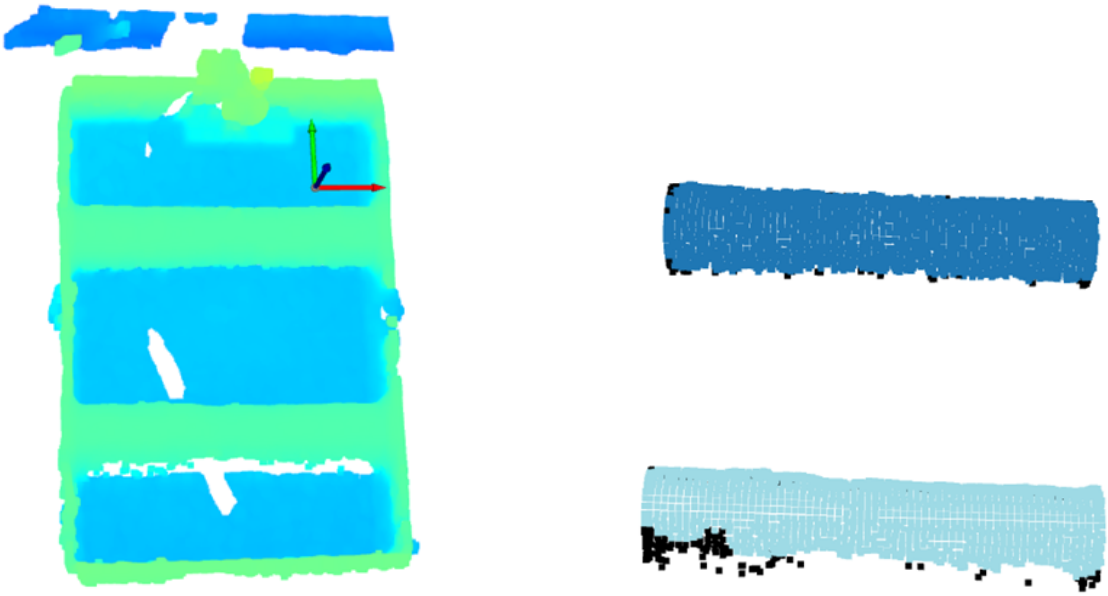
There is noise in the point cloud, and the points generated from the walls of the incubator are not smooth planes. The cropping is therefore done with some exaggeration, meaning the cropping parameters are set to be a bounding box even smaller than the internal measurements of the box. This removes the gears and gripping area of the tubes, leaving only a part of the cylinder.

This exaggeration does not affect the positioning of the tube frame, as the positioning of the frame is not reliant on data of the whole tube. Only two translation parameters of the tube frame are reliant on the point cloud of the tube, and neither of these relies on the full length of the tube. Asymmetric cropping should not affect

the positioning either for the same reason.

5.1.3. Clustering

To both count and isolate tubes in the incubator, the clustering algorithm DBSCAN is used, described in section 2.4. This method of clustering is used as it is not reliant on pre-defining the number of clusters, and it is also capable of excluding points as noise. The output of running DBSCAN on the cropped point clouds is a list containing point clouds of individual tubes, or an empty list if no clusters are found. The main parameters of the algorithm are the minimum amount of points in a cluster and the radius around each point that will be considered. Figure 5.6 visualizes the before and after cropping and clustering a point cloud of the incubator. The black points in Figure 5.6b are outliers who are not part of a cluster.



(a) Point cloud of incubator **(b)** Clustered point cloud

Figure 5.6.: Point cloud of incubator with two tubes. The tubes were successfully clustered and isolated.

5.1.4. Positioning

The position of the tube is defined by a coordinate frame. The tube frame is thought as shown in Figure 5.8. The goal of this object detection is to place the tube frame relative to the camera, and then the robot. Since the marker frame has multiple shared parameters as the tubes, and is well-defined relative to the camera, it is used in the tube positioning. Equation 5.1 shows the final computation of the

tube frame relative to the base frame. $T_{\text{CameraMarker}}$ is already obtained prior to the cropping process, and $T_{\text{BaseCamera}}$ is obtained in the hand-eye calibration presented in section 5.2.

$$T_{\text{BaseTube}} = T_{\text{BaseCamera}} \cdot T_{\text{CameraMarker}} \cdot T_{\text{MarkerTube}} \tag{5.1}$$

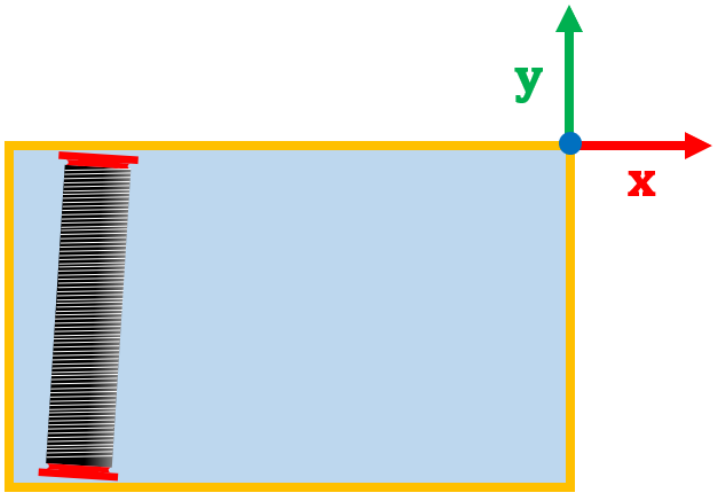


Figure 5.7.: Coordinate system of the incubator

Before the process of positioning, the cropping and clustering isolate the tubes as individual point clouds. Looking at the coordinate system in Figure 5.7, the incubator constrains the movement of the tube along its y-axis and the rotation about its z-axis. This allows for defining these parameters using marker detections. The y-coordinate of the tube frame is placed at $\frac{1}{2}(w_{\text{tot}} + w_{\text{marker}})$, where w_{tot} is the width of the incubator including walls and w_{marker} is the width of the marker. For the old placements, this coordinate was determined using the z-axis of the long side markers $\frac{1}{2}w_{\text{int}}$, where w_{int} was the internal width of the incubator.

The flotation and assumed evenly distributed weight of the tube results in no rotation about the x-axis. The rotation about the y-axis is not relevant for grasping, due to the cylindrical shape. The resulting missing parameters are the positions along the x-axis and z-axis.

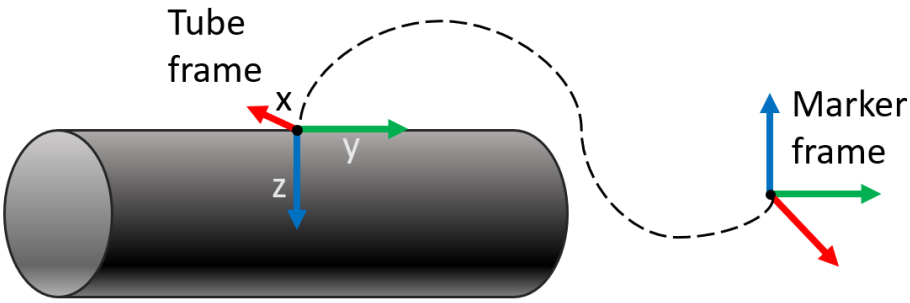


Figure 5.8.: Tube frame visualization

The z-coordinate can be found by transforming the point cloud to the coordinate system of one of the fiducial markers. Note that both markers are assumed to have the same orientation. The markers are placed with their z-axes pointing upward close to parallel with the vertical axis. Assuming the clustering has removed all noise, and the remaining points are points found on the tube, the point with the highest z-value can be used to describe the z-position. Due to the high possibility of noise, the 10^{th} and 20^{th} highest points have been used, as these points are assumed likely to be close to the top of the tube.

The x-coordinate can be found using the mean x coordinate of the clustered tube point cloud while in the frame of a marker. The coordinate system of the markers xy-plane is close to parallel to the horizontal plane, and their x-axis is close to parallel to the x-axis defined for the tube. The coordinate is found by taking the numeric average of all x-coordinates of the tube point cloud. Note that this could lead to a bias based on the angle of the camera, as illustrated in Figure 5.9.

Having found all coordinates of the tube frame, the point cloud is transformed into the coordinate system of the marker on the long wall. A new frame of the same orientation is placed in the coordinates of the newly found tube frame positions and rotated 180° to fit the orientation of the gripper. This preserves the orientation of the vertical axis.

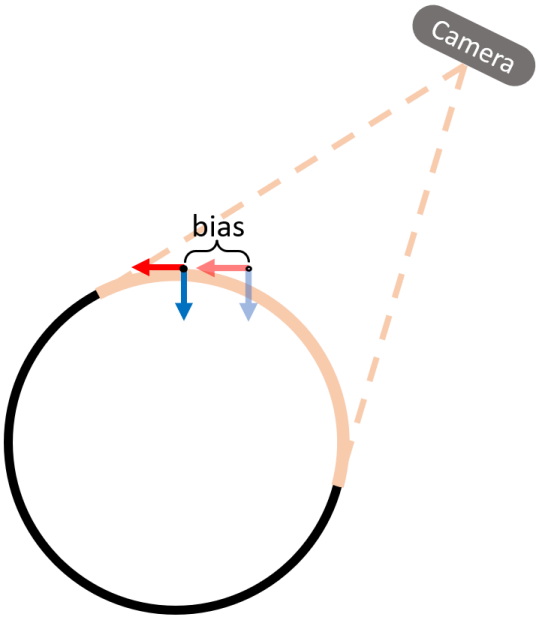


Figure 5.9.: The position of the tube relative to the camera determines where the mean x-coordinate will be.

5.2. Camera and Hand-Eye Calibration

In order for the robot to act on the data generated by the computer vision system, the camera and robot positions must be found relative to each other. This is one of the objectives of the computer vision system, and highly relevant in order for the objective of testing the system to be possible.

The calibration of the camera intrinsics and the hand-eye relationship is done by fixing a checkerboard to the robot end, placing the robot end in N different positions, and taking a snapshot of the checkerboard for each position using a camera fixed to the robot base frame. These snapshots are used to both calculate the camera intrinsics and to calculate the relationship between the robot end frame and checkerboard.

5.2.1. Method for Calibration

The hand-eye calibration is performed using the $AX = XB$ method described in section 2.5. The Python code used to solve the calibration problem was found in a GitHub repository created by Torstein A. Myhre[2], and modified to fit this project.

During the development process, different combinations of calibration parameters were applied and tested to some degree. In terms of checkerboards, a smaller one

[2] https://github.com/torstem/robcam-calibration/blob/master/README.md

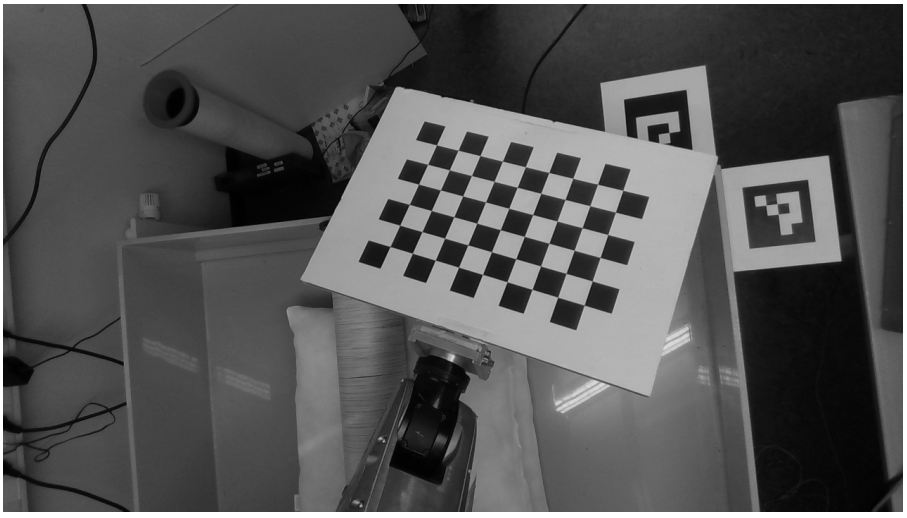
fastened to the gripper was used, as well as a larger checkerboard mounted instead of a gripper, shown in Figure 5.10. When it comes to the intrinsic parameters, the Intel RealSense comes with precalibrated intrinsics callable from the camera software, with the option to re-calibrate using a white wall. However, the best results came from the checkerboard calibration of intrinsic parameters presented in subsection 2.3.2.

Using Park and Martin [13] described in section 2.5, the algorithm after saving N different robot positions and snapshots was as follows:

- Detect checkerboard corner pixels for all images
  - Remove image and the corresponding robot pose from lists if no checkerboard detected
- Retrieve camera intrinsics using OpenCV (Alternatively retrieve pre-calibrated intrinsics from the RealSense camera using the RealSense Python library)
- Get the object pose for the checkerboard in all images using OpenCV
- Calculate $X = \begin{bmatrix} R_X & p_X \\ 0 & 1 \end{bmatrix} \in SE(3)$ as described in section 2.5.
  - Create lists A and B (A_i is the transformation between robot pose i and $i+1$, and B_i is equivalent for detected checkerboard positions)
  - Get R_{A_i} and R_{B_i} and calculate M as sum of all outer products of $\log(R_{A_i})$ and $\log(R_{B_i})$
  - Calculate $R_X = (M^T M)^{-\frac{1}{2}} M^T$
  - Calculate $p_X = (C^T C)^{-1} C^T d$ where C and d are as described in Equation 2.20
- Use the newly found $X = T_{\text{EndCheckerboard}}$ with a robot pose and corresponding checkerboard image to calculate $T_{\text{BaseCamera}} = T_{\text{BaseEnd}} \cdot T_{\text{EndCheckerboard}} \cdot T_{\text{CheckerboardCamera}}$

The creation of the lists A and B was done by finding the transformation between the pairs i and $i+1$. Resulting in a list length of $N-1$. Using more permutations results in more input to the least squares method, which could increase the accuracy. As the number of samples increases, the least squares method has more data points to work with, which should result in a more accurate estimation. When using this method, the shape of the graph representing the error tends to converge toward a minimum as more samples are included.

(a) Small checkerboard



(b) Large checkerboard

Figure 5.10.: Different checkerboards for calibration

Chapter 6.

Testing and Experiments

In this chapter, the data gathering methods and data are presented. Some sections present testing that occurred throughout one or more periods in the project period, while some sections present single-session experiments. The different sections present the following testing:

- 6.1 Verifying the accuracy and quality of the camera intrinsic- and hand-eye calibrations.
- 6.2 Verifying the feasibility of the object detection algorithm in a scenario with water and multiple substrate tubes in an incubator. Also testing the folding fingers gripper with floating tubes.
- 6.3 Testing the consistency of the system by picking up a tube randomly placed in an incubator.

6.1. Verifying Camera and Hand-Eye Calibration

After establishing $AX = XB$ [13] as a viable calibration method for hand-eye calibration during preliminary testing, some data was gathered when conducting calibrations. As mentioned in section 5.2, both a large and a small checkerboard was tested during the project period, and both pre-calibrated and calibrated intrinsics were used while performing hand-eye calibration.

For calibration, $N \approx 20$ robot poses in which the checkerboard was visible, were found using the teach pendant. By moving between these positions and taking snapshots N checkerboard images with corresponding poses were stored. The poses stored were retrieved by using the `get_position_transformation_matrix()` function in the DENSO controller library.

The lists of As and Bs, the relationship between positions, were calculated. Since the X calculated in $AX = XB$ is the relationship between the robot end frame and the checkerboard frame, there are N sets of "camera $\rightarrow$ checkerboard $\rightarrow$ robot

end → robot base" relationships to calculate the relationship between the camera frame and the robot base frame, T_{BC}. The relationship is calculated as seen in Equation 6.1, where C, E, Ch denote the camera frame, robot end frame, and checkerboard frame respectively.

$$T_{\mathrm{BC}} = T_{\mathrm{BE}} \cdot X \cdot T_{\mathrm{ChC}} = T_{\mathrm{BE}} \cdot T_{\mathrm{ECh}} \cdot T_{\mathrm{ChC}} \tag{6.1}$$

As a way of indicating the accuracy in the calculation of X, a T_{BC} was calculated for all N sets of images and robot poses. As $T_{\mathrm{BC}} \in \mathrm{SE}(3)$ it contained the vector p, as presented in subsection 2.1.2]. Looking at p for all N calculations, the biggest gap in the x, y, and z coordinates was calculated, as well as the index of the ones with the largest gap to see if it was an individual image and pose combination that created an outlier. A large gap was assumed a poor calibration.

This method of verifying the calibration was done as the translation vector was possible to visualize, while the rotation matrix is more difficult to visualize by looking at it. However, an error in the rotation could be revealed, but not necessarily as a rotation error. If a rotation error is present on the right side of Equation 6.1, it should result in a translation error.

Other notes on the method:

- At one point a re-calibration of the stored intrinsic parameters of the Intel RealSense camera was conducted following the steps provided by the Intel RealSense Viewer software.
- The data gathering of the hand-eye calibration revealed that the small checkerboard had an obviously larger gap between translation vectors when performing the N calculations of T_{BC}, leading to it being excluded from further testing.
- Set 4 of calibration positions had smaller movements between each position than the previous testing sets.
- When establishing robot poses in which the checkerboard was visible to the camera, the movement between these positions resulted in near singularity situations. Two joint axes were close to parallel, and the joints would accelerate to a point where the robot controller would stop due to join speed limits being exceeded.

6.1.1. Results

Table 6.1 presents data from some of the calibrations performed during the project period. A small explanation of the columns are:

- No.: Number on the list
- Precalibrated: Whether or not the stored camera intrinsic parameters of the Intel RealSense camera are used or OpenCV's checkerboard calibration.
- Large/Small Checkerboard: Says which checkerboard was used. Both are Pictured in Figure 5.10.
- Biggest Gap in Translation [mm]: The gap between the largest and smallest x, y, and z coordinates of p in mm.
- Biggest Gap in Translation [%]: The gap between the largest and smallest x, y and z coordinates of p, as a percentage of the mean x, y and z value.
- Before/After Recalibrating: Whether or not the hand-eye calibration occurred before or after the re-calibration of the Intel RealSense internal camera intrinsics.
- Images/Permutations: How many images/poses were used, and how many permutations/combinations of these images/poses were used, section 5.2.
- Set of Calibration Positions: As the camera was moved somewhat throughout the project, $<$ 0.3m, new sets of calibration positions were used to get the checkerboard in the image.

Table 6.1.: Data gathered from several hand-eye calibrations

No.	Precalibrated	Large/ Small Checkerboard	Biggest Gap in Translation [mm]	Biggest Gap in Translation [%]	Before/ After Recalibrating	Images/ Permutations	Set of Calibration Positions
1	Yes	Small	176,00	24,34	Before	18/17	1
			77,14	24,65			
			83,40	8,60			
2	No	Small	270,87	38,58	NA	18/17	1
			112,74	37,19			
			76,81	7,81			
3	Yes	Large	41,71	5,66	Before	19/18	2
			45,28	14,51			
			30,77	3,12			
4	No	Large	7,92	1,09	NA	19/18	2
			8,76	2,85			
			7,15	0,72			
5	Yes	Large	39,11	5,39	After	18/17	2
			35,84	11,50			
			26,15	2,64			
6	No	Large	6,63	0,92	NA	19/18	3
			8,09	2,65			
			6,19	0,63			
7	No	Large	7,15	9,93	NA	19/53	3
			8,09	2,65			
			6,20	0,63			
8	No	Large	2,46	0,28	NA	19/18	3
			4,81	1,46			
			2,77	0,28			
9	No	Large	3,35	0,39	NA	18/17	4
			2,69	0,82			
			2,27	0,23			
10	No	Large	3,37	0,39	NA	18/17	4
			2,86	0,87			
			2,53	0,25			
11	Yes	Large	7,67	0,88	After	18/46	4
			10,05	3,01			
			8,80	0,89			
12	Yes	Large	7,52	0,87	After	18/17	4
			9,93	2,98			
			8,95	0,90			
13	No	Large	3,66	0,42	NA	18/46	4
			3,07	0,93			
			2,78	0,28			
14	No	Large	3,95	0,46	NA	19/53	4
			3,21	0,97			
			1,91	0,19			

The procedure of hand-eye calibration also included checkerboard calibration on many occasions. This yielded a camera matrix and a set of distortion coefficients. The internally calibrated distortion parameters were all zero, indicating no distortion, while the checkerboard calibration detected some distortion.

6.2. Experiment: Verification of Function with Water

As described in the section 1.2, the objective is to develop a robotic system for manipulating substrate tubes. This manipulation of the tubes includes picking and placing them in seawater. The majority of development of both the gripper and the computer vision system described in chapter 4 and chapter 5 respectively, has been done without water in the incubator. However, with the goal of developing a system for solving tasks involving water, requirements have been discussed in the prestudy [6], and functionality with water has been an underlying factor for decisions made throughout the development process.

The robot cell is located in a laboratory in which the use of water is quite difficult. Therefore an experiment was conducted in the lab used for seaweed growth. This lab uses the same incubator as in the robot cell. The experiment was divided into two parts. One test for the gripper and one for the computer vision system.

Part one of the experiment was to verify the functionality of the object detection part of the computer vision system. The hypothesis is that the computer vision system should perform similarly to the testing performed without water. Due to buoyancy, the top of the tube will be close to the water surface or slightly above, and determining the vertical coordinate of the tube frame should be possible without factoring in water refraction. The refraction of light will probably distort the creation of the point cloud, but the spacing between them should still allow the clustering to isolate each tube. There is also a possibility to change the parameters of the clustering to accommodate this.

The refraction might have an effect on the one coordinate of the tube frame calculated by the mean value of the clustered point cloud, but the design of the gripper should allow for inaccuracies along this axis. The last coordinate needed to define the tube position, as well as the orientation, are defined by readings of the fiducial markers. Due to the placement of the markers, the water should not affect this reading.

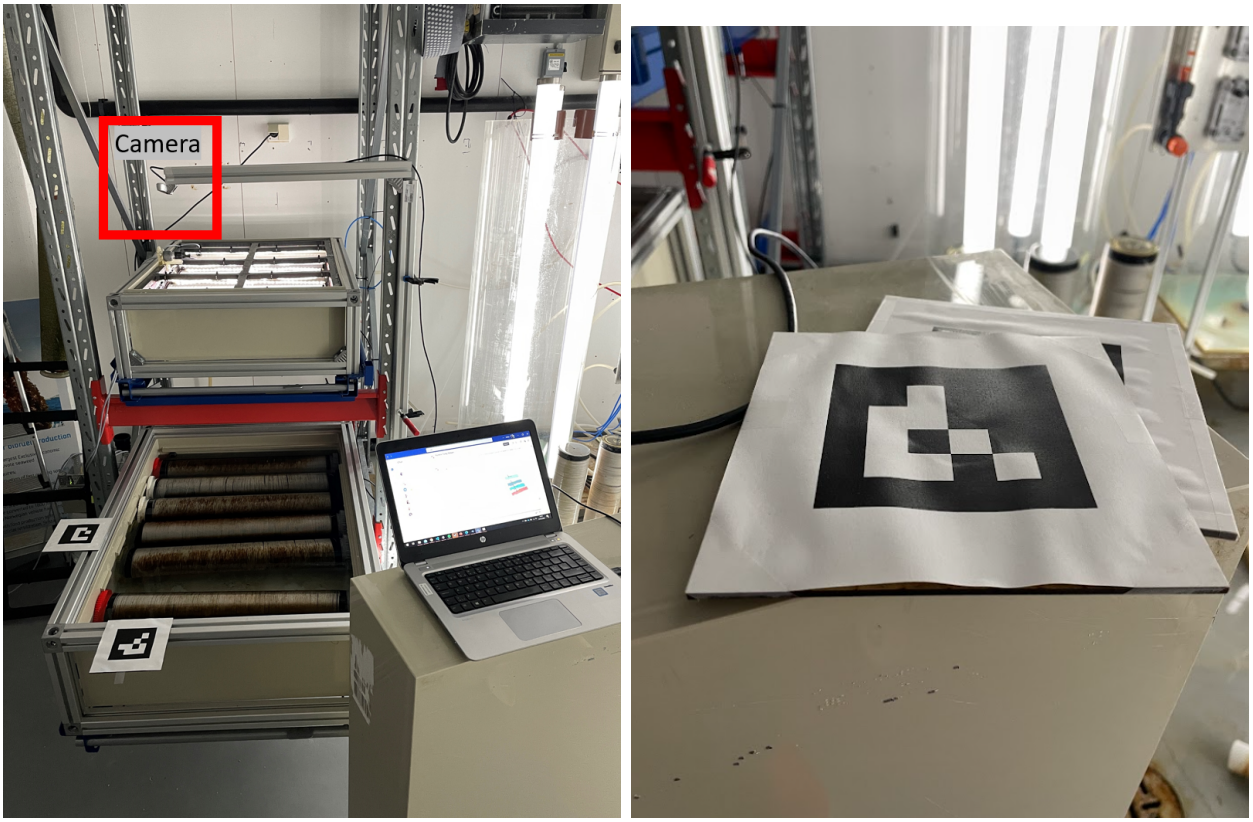
Part two of the experiment was to verify the functionality of the folding gripper described in section 4.2. As the gripper consists of fingers forced to their open position by rubber bands, pushing the gripper down on the floating tube will force the fingers around the tube. The tubes are assumed to have a symmetric weight

distribution, which will make the tube top line horizontal. The gripper should therefore be horizontal when gripping the tube. The fit of the tube in the incubator is assumed to keep the tube from rotating about the vertical axis. If placing the gripper at or slightly above the tube frame, a vertical movement downwards should be sufficient to get the tube in position to lock the gripper, without pushing into the floor of the incubator.

The verification of tube positioning in the gripper is normally done by the sensor feedback described in subsection 4.2.1, but for this test a visual confirmation was used.

6.2.1. Method

Part One



(a) Setup for experimenting with water **(b)** Wavy marker

Figure 6.1.: Images from experiment setup

The experiment was conducted in SINTEF Ocean's seaweed laboratory, with the Intel RealSense D415 camera mounted as pictured in Figure 6.1a. The camera was connected to a Windows computer with the Python program for object detection. The fiducial markers were used to crop and position the incubator. The incubator

used was different from the one used for developing the system, as it had notches keeping some of the tubes submerged.

As neither the incubator nor the camera was moved between the tests, the detection of the fiducial markers was performed once before the tests. The frames of the markers were visually confirmed to be satisfactory. The markers were produced using paper on a piece of fiberboard, and the paper lost its planar form as seen in Figure 6.1b. This was assumed due to the humidity in the seaweed laboratory.

12 different tests were conducted. The parameters changed between tests were:

- Cropping height on the vertical axis, subsection 5.1.2
- The ϵ and "minimum points" of the DBSCAN, subsection 5.1.3
- Ceiling light
- Number of point clouds

The tests were performed by creating a number of point clouds of the incubator and surroundings, and running the object detection program, section 5.1, on each point cloud. The primary verification of correct detection was done by counting the number of clustered point clouds, and these were again verified visually. For the tests done without ceiling light, there was still an amount of light emitted from light sources in the lab. At first, only 3 point clouds were created for each test. As the results showed promise, the amount of point clouds was increased for the resulting tests.

Part Two

Using gripper units similar to the ones used for the development and testing in the robot cell were operated by hand to get a sense of functionality when used on floating tubes. The testing was conducted by pushing the gripper units down on the tube, getting a sense of how often the tube touched the floor before the gripper units were in a position to lock.

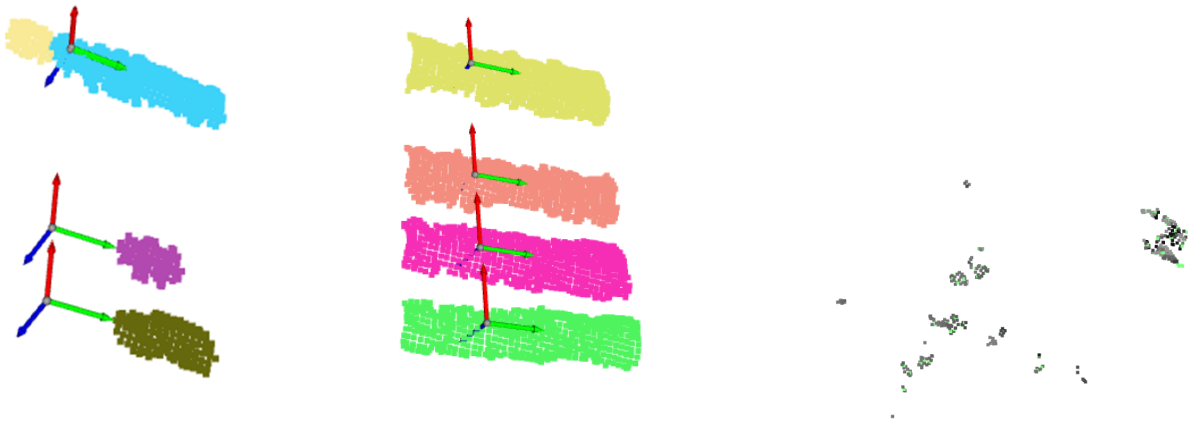
6.2.2. Results

Part One

The parameters and results of each test are presented in Table 6.2. The first 3 tests were to provide some evidence of the need for turning the ceiling light off and adjusting the cropping parameter. Some key notes on the results are listed below:

- The instances where the count was 4, but the visual verification proved the detection wrong, were all similar to Figure 6.2a.

- In the instances of a successful running of the detection program, the results were similar to Figure 6.2b.
- The "unusable point cloud" section of the table is used to describe when the camera only produced an indistinguishable point cloud of sparsely placed points, similar to Figure 6.2c.
- Cells marked "NA" are in columns that were added during the testing session, meaning the column was not added for that specific test.



(a) Wrongly detected tubes **(b)** Correctly detected tubes **(c)** Unusable point cloud

Figure 6.2.: Wrong and correct clustering

The 2 innermost tubes were occluded due to the camera placement. Therefore the correct detection of the 4 outermost tubes is set to be a successful result. The ceiling light turned on resulted in a light reflection on the water surface, reducing the quality of the point cloud of the incubator. As seen, the submerged tubes were also discovered by the algorithm.

The issue of unusable point clouds led to the abortion of multiple tests. After reconnecting the camera USB, the issue stopped for a while. Sometimes the problem would fix itself without reconnecting.

Table 6.2.: Results from testing with water

N.o. point clouds	**Ceiling light**	**Cropping parameters [mm]**	ϵ	**Minimum points**	**Count =4**	**<4**	**>4**	**Verified**	**Unusable point cloud**	**Success rate [%]**
3	On	$>z_{\max}$ - 70, $>z_{\text{avg}}$ + 70	40	60	0	NA	NA	0	NA	0
3	Off	$>z_{\max}$ - 70	40	60	0	NA	NA	0	NA	0
3	Off	$>z_{\max}$ - 30	40	60	3	NA	NA	3	0	100
100	Off	$>z_{\max}$ - 30	40	60	60	NA	NA	59	7	59
100	Off	$>z_{\max}$ - 30	40	60	70	2	22	70	0	70
50	Off	$>z_{\max}$ - 30	40	70	2	1	32	2	15	4
50	Off	$>z_{\max}$ - 30	45	60	18	29	1	18	2	36
50	Off	$>z_{\max}$ - 30	40	60	38	3	9	38	0	76
50	Off	$>z_{\max}$ - 30	42	60	41	5	3	40	1	80
50	Off	$>z_{\max}$ - 30	42	60	33	12	4	33	1	66
50	Off	$>z_{\max}$ - 30	43	60	41	5	3	41	1	82
50	Off	$>z_{\max}$ - 30	43	60	34	12	1	34	1	68

Part Two

The buoyancy of the tubes was less than initially thought. The gripper units brought to the seaweed lab had folding fingers with various "looseness" to the fingers, meaning some fingers needed more force to fold around the tube, not including the rubber band. The issue was seemingly mechanical friction in the joint. This was particularly the case for one of the gripper units used for the test.

The gripper unit without loose joints worked as expected, even with the buoyancy

being less than assumed. When putting it in position above the tube and pushing is softly down on the tube, it folded around the tube without pushing the tube down towards the floor of the incubator.

Much of the gripper was submerged during gripping, as pictured in Figure 6.3, with the surface line drawn on the right side to better show the waterline roughly marked by white lines. One could argue that this image shows a "best case scenario" if little submersion is optimal, as no force is pushing the tube down except the relatively low weight of the gripper unit.

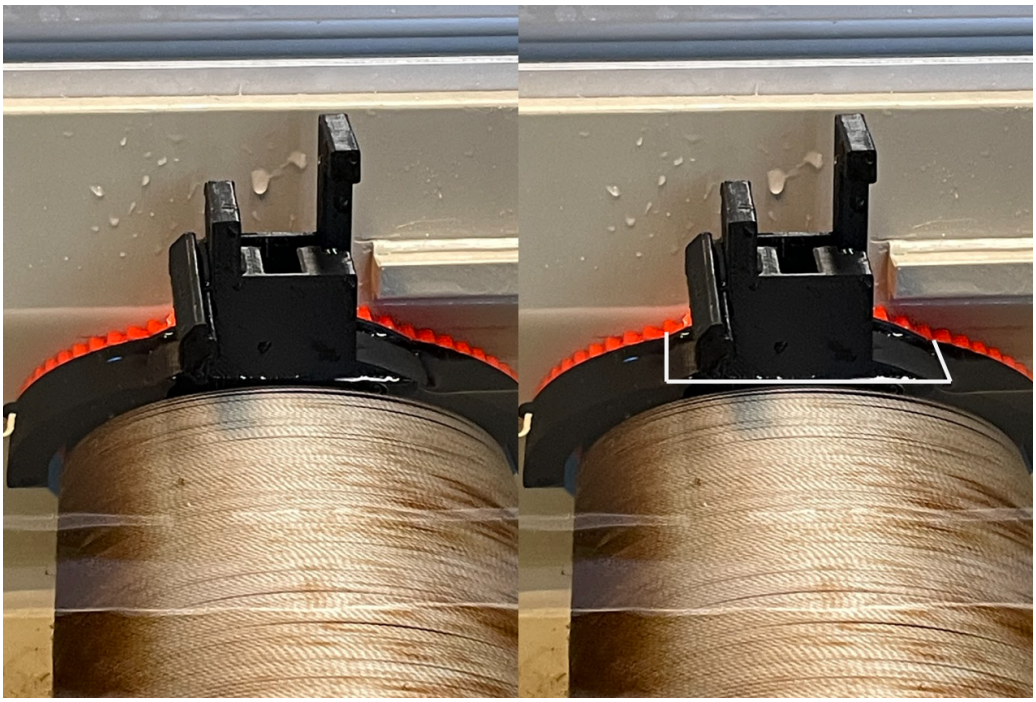


Figure 6.3.: Gripper unit in gripping position while the tube is in water

6.3. Picking Up the Tube

As written in section 1.2, the objective of this project is to create a proof of concept for manipulating the substrate tubes. The prestudy [6] concluded that the task of finding and picking the tube up from the incubator was the most challenging. To have the robotic system autonomously pick a tube up from the incubator the sum of accuracy in the hand-eye calibration, fiducial marker detection, and object detection algorithm must be sufficient. With a computer vision system developed, it was possible to test the system's capabilities of picking up tubes from the incubator.

6.3.1. General Method

The computer vision system described in chapter 5, involves a hand-eye calibration, as well as an object detection algorithm that returns a set of tube frames. Prior to each pick-up testing session, a hand-eye calibration was performed. Some preliminary testing was conducted to ensure the calibration was seemingly adequate. As the OpenCV calculated camera intrinsics yielded the most promising results in section 6.1, these parameters were used.

The folding finger gripper introduced in section 4.2 was used, as this was developed the furthest and included sensor feedback. A set of pillows were used to simulate flotation. The pillows would also reduce the risk of the robot colliding with the incubator floor.

The pick-up process was performed as listed below:

1. Move the robot to the home position to not occlude the camera.
2. Take RGB snapshot to detect markers
3. Take a depth snapshot to create a point cloud
4. Detect tubes using the algorithm described in section 5.1
5. Move the robot to a position vertically above a tube frame
6. Rotate gripper so rotation about x and y is zero
7. Move robot vertically down to almost align gripper frame to tool frame
    - If visible risk of collision with the incubator stop the robot and reset
8. Move robot end downwards a small distance a given amount of times to get the tube in position for the gripper to lock
9. Lock the gripper when the sensors signal all 4 fingers are in the gripping position
10. Lift the tube vertically to a safe position

During some testing of the gripper when prototyping, tool frame parameters relative to the robot end frame was established, Table 6.3. The parameters were set using the `set_tool_params()` function in the Python library provided by SINTEF Ocean. The original parameters were used with minor changes throughout the testing. A slight offset in rotation about the z axis was discovered at the very start of the familiarizing period, resulting in the -2.5° in r_z, and a slight offset in the y parameter has also been used.

Table 6.3.: Relationship between the robot end mount and the defined gripper frame

x[mm]	y[mm]	z[mm]	r_x[°]	r_y[°]	r_z[°]
-50	0	110	0	0	-2.5

While testing, the incubator was placed close to the robot to try to ensure the tubes were within reach of the robot. Looking at Figure 6.4, the different incubator zones are defined roughly relative to the robot base frame and camera mounting. The angle α is moved arbitrarily between each test to verify the robustness of the incubator calibration.

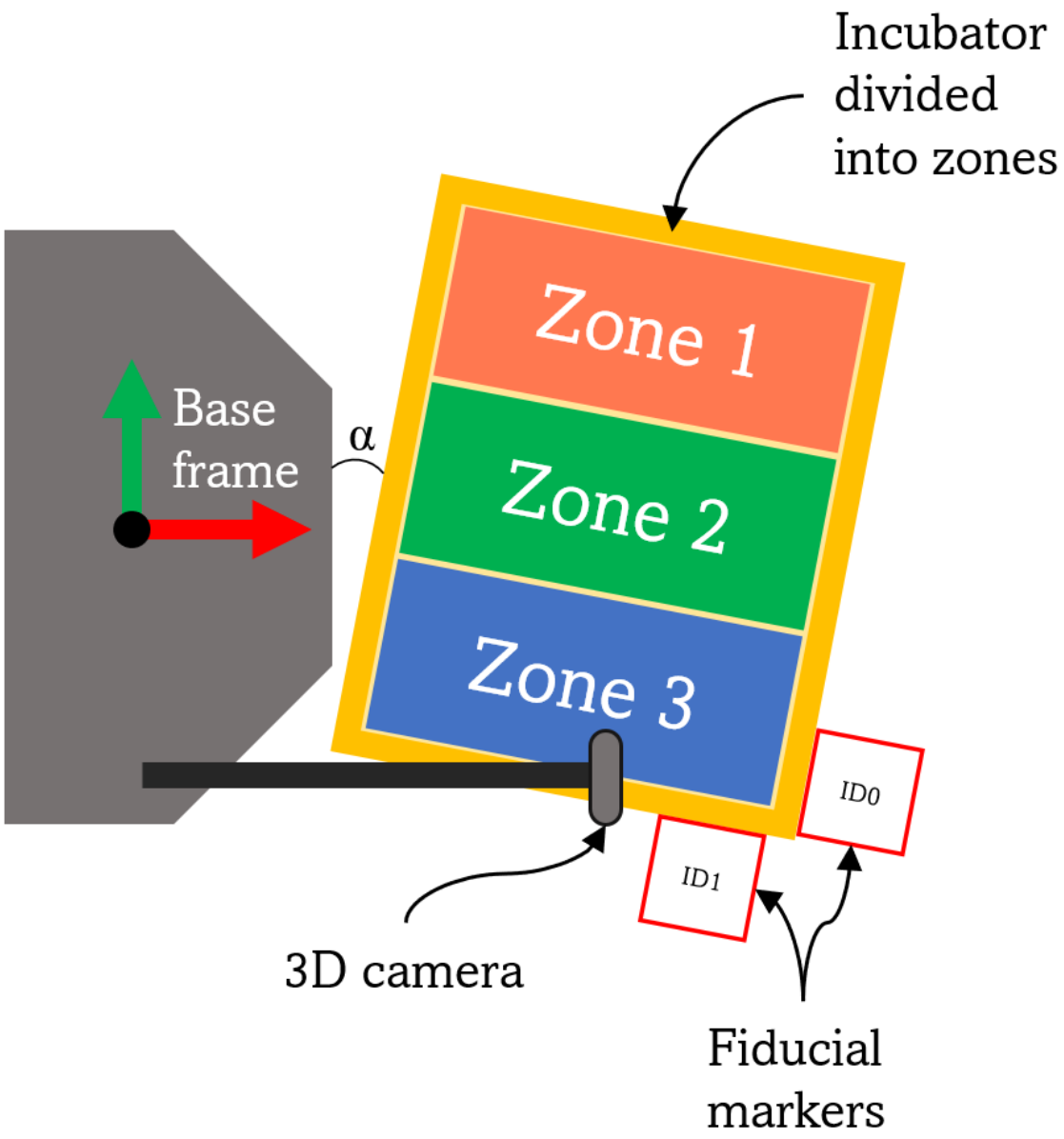


Figure 6.4.: Set up for pick up testing including different incubator zone definitions

6.3.2. Pick-Up Testing With Markers Inside the Incubator

This testing session was conducted prior to the moving of the fiducial markers from inside the incubator. Only Zone 1 and 3 were relevant, as placement in Zone 2 would occlude a marker. The main reason for the test was to get a reference of how functional the hand-eye calibration and object detection were at this stage in the project, as well as look for obvious points of improvement.

The testing session consisted of 18 tests, 9 in Zone 1 and 9 in Zone 3. The hand-eye calibration performed prior to the testing was the first calibration with the largest gap for all 3 coordinates being less than 5mm, calibration number 8 in Table 6.1.

Results

The results of the testing session are visualized in Figure 6.5. In 4 out of the 5 tests that needed more than one attempt, the attempts were aborted due to the robot seemingly heading towards a collision with the incubator, while the last of the 5 was just not able to lock in. The main reason for failure in the 4 collision bound attempts was seemingly an error in the tube frame coordinate determined by the fiducial marker inside the longest wall of the incubator.

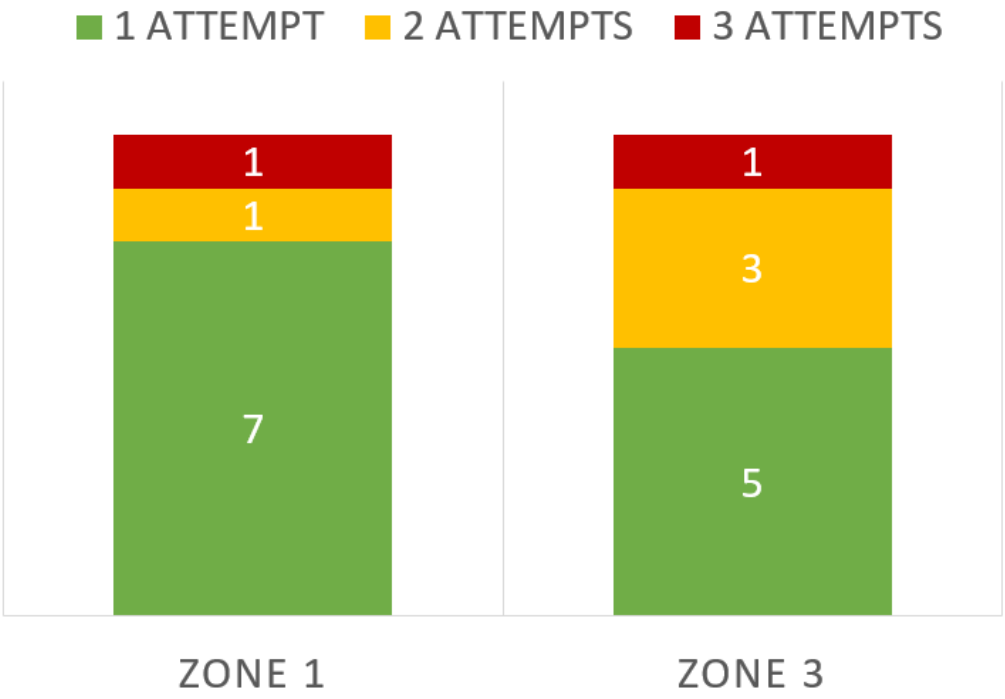
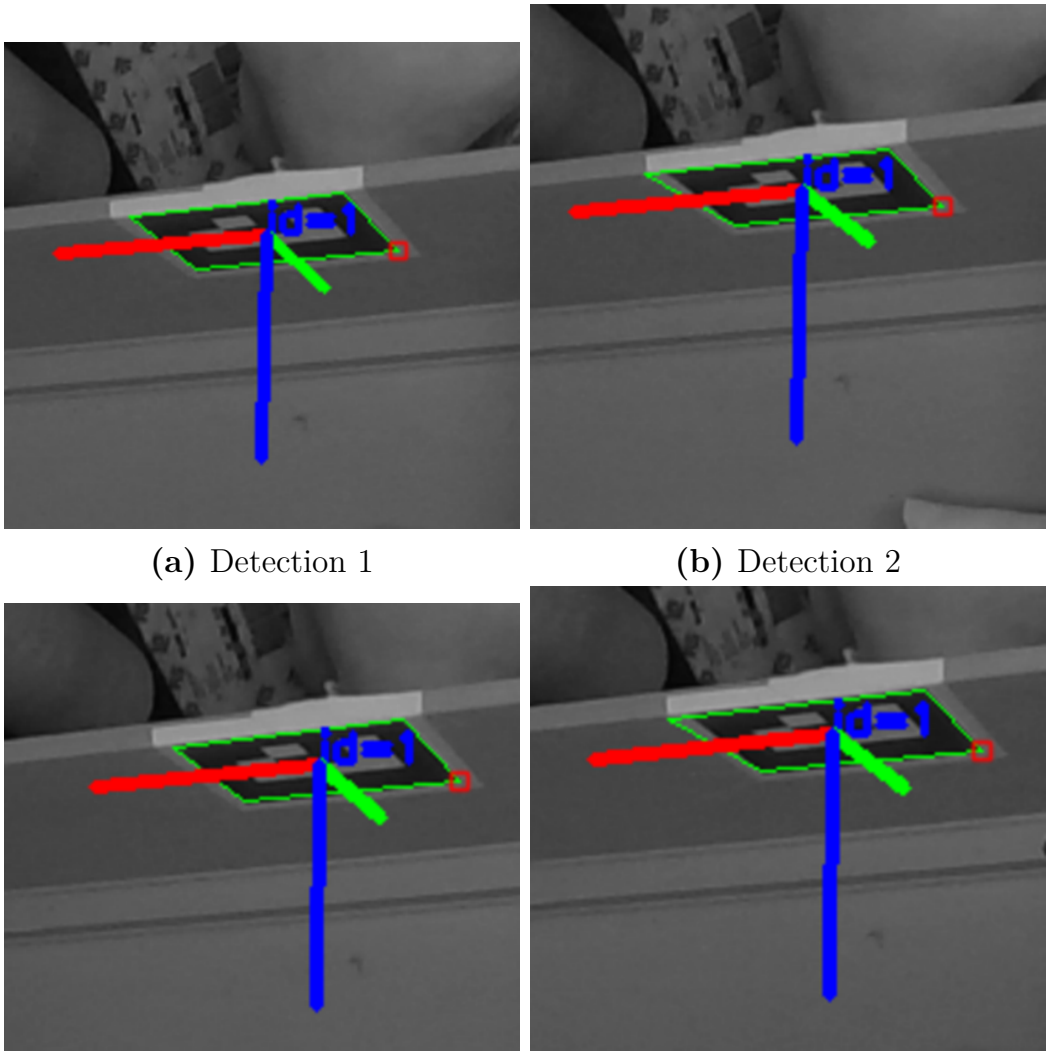


Figure 6.5.: Number of attempts to grip the tube

6.3.3. Marker Positioning

The markers were placed inside the incubator, as seen in Figure 5.4a, early in the project period as it was the most convenient solution at the time. However, due to the possible effects of reflection and refraction of light at the water surface, and the fact that some tube placements occlude the markers, it was established early they needed to be moved.

Error in Marker Readings



(a) Detection 1 **(b)** Detection 2

(c) Detection 3 **(d)** Detection 4

Figure 6.6.: 4 detections of the long side marker. Detections 2 and 4 have a wrongly placed corner.

A small investigation of the marker readings used in subsection 6.3.2 revealed a reoccurring error. 4 detections of the same marker are visualized in Figure 6.6 with no movements or changes in parameters between each detection. The marker

detection provides a rotation and translation vector, and the translation vectors are presented in Table 6.4.

Table 6.4.: Translation vectors of 4 long side marker detections

Detection No.	Translation Vector [mm]	Length [mm]
1	[119.2, -303.8, 1085.8]	1142.4
2	[109.6, -283.7, 1016.1]	979.5
3	[118.9, -302.2, 1081.1]	1121.0
4	[110.7, -285.6, 1023.9]	948.2

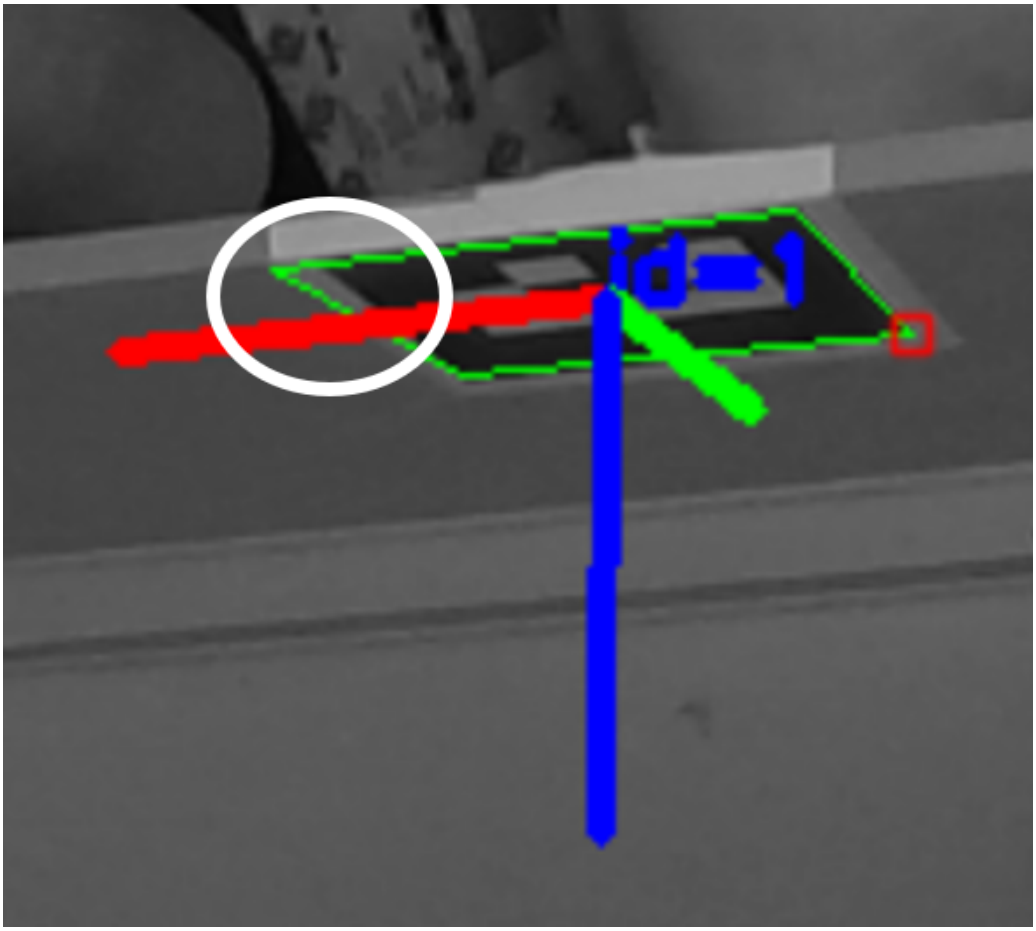


Figure 6.7.: Wrong detection of the corner on ArUco marker. The white circle shows the corner that is placed at the corner of the white surrounding paper frame.

The vector lengths show a $>$ 100mm gap between detections 1 and 3 compared to 2 and 4. Looking at detections 2 and 4, the visualizations pictured in Figure 6.6b and Figure 6.6d show a wrong detection of the upper right corner. Figure 6.7 shows this error emphasized. The corner of the white paper background seems to be detected as the corner of the marker.

The z-coordinate of this marker determines one of the tube frames coordinates directly. This makes this reading critical for the system's functionality. Table 6.4 reveals a >50 mm difference in the z-coordinates from one reading to another.

Simultaneous as the detections of the long side marker, the short side marker was also detected, visualized in Figure 6.8 with their translation vectors presented in Table 6.5. These readings show the short side markers being more consistent. Figure 6.8 show no obvious error in detecting the corners of the markers. This marker was cut flush to the marker, leaving no white paper frame. However, the inconsistencies in the detections might be non-negligible.

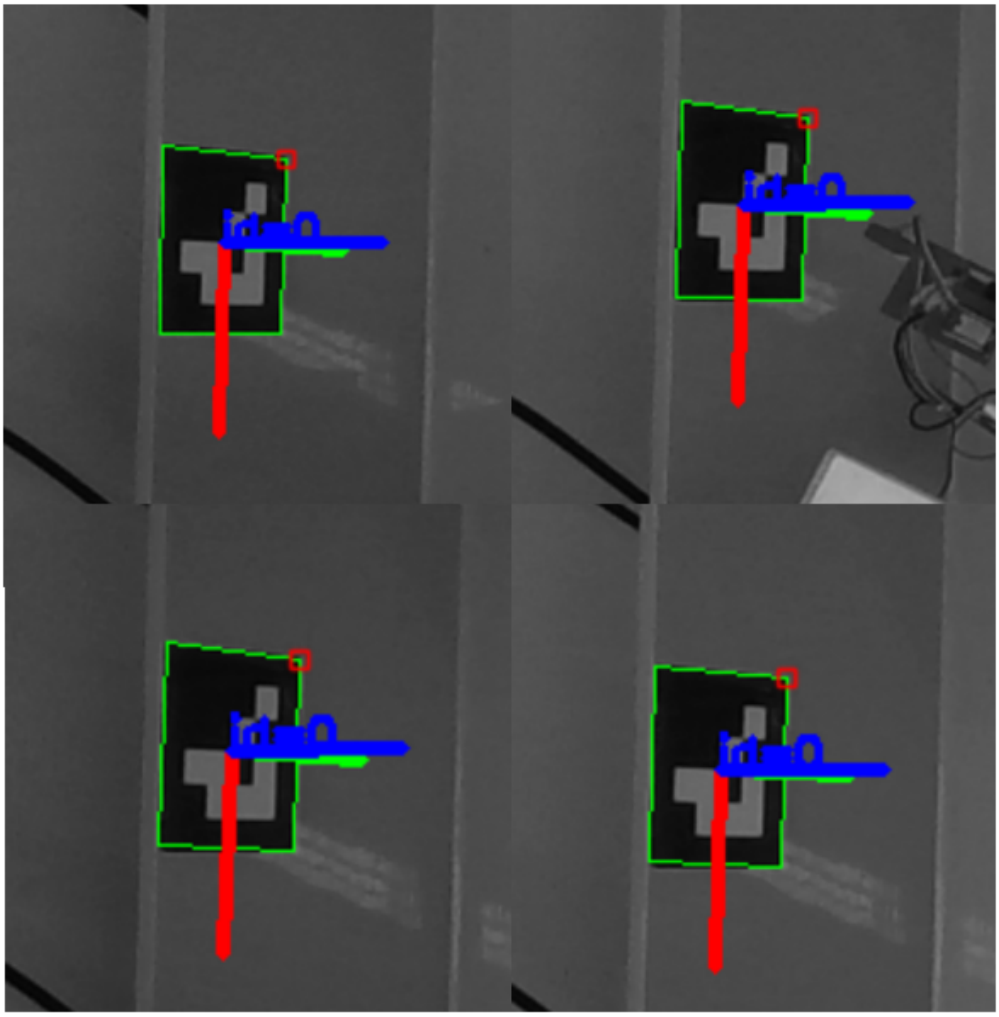


Figure 6.8.: 4 detections of the short side marker. There are some small but visible differences between each reading.

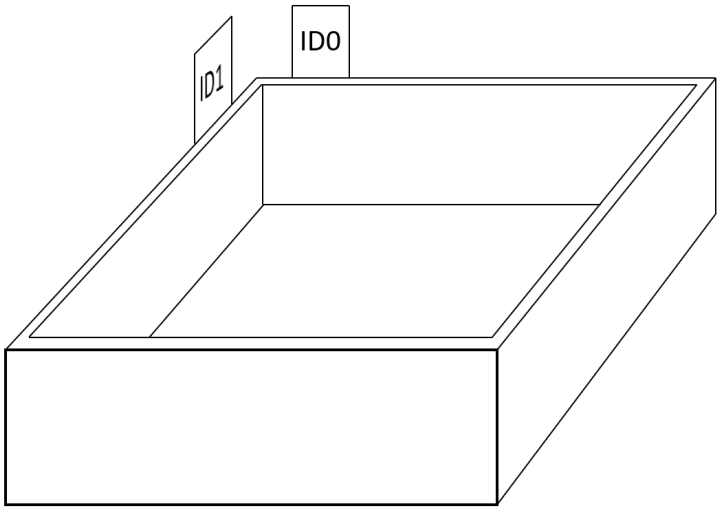
Table 6.5.: Translation vectors of 4 short side marker detections

Detection No.	Translation Vector $[x, y, z]$	Length
Detection 1	[-364.8, -7.1, 1310.6]	1311.1
Detection 2	[-364.2, -7.8, 1309.8]	1310.4
Detection 3	[-367.3, -8.3, 1319.6]	1320.0
Detection 4	[-368.4, -8.3, 1322.2]	1322.7

New Marker Positions

As the markers were moved, one of the options was as illustrated in Figure 6.9a. These positions would result in a very minimal change in all software parameters regarding the markers due to the similarity to the original positions. However, the suspicion that the steep angles of this orientation would result in less precise readings led to the new positions being as presented in subsection 5.1.2, and pictured in Figure 6.9b.

New markers in different sizes were printed with a relatively larger white surrounding frame, hoping to reduce the risk of the errors presented in Figure 6.6. Mounts were 3D-printed to allow for placing the marker plates orthogonal to the incubator walls, as well as close to parallel to the horizontal plane. The less steep angles relative to the camera would make the markers larger in the image, potentially making the detection more accurate due to more pixels representing the markers. This argument also favored the largest of the markers created. They measured 89mm×89mm, while the smaller measured 45mm×45mm.



(a) Alternative new positions of markers

(b) Intel RealSense image of incubator with new marker positions and their frames visualized

Figure 6.9.: Alternative and final new marker positions. The final new position has a much less angle towards the camera.

Data on the new positions were gathered by capturing 1000 images of the markers. Table 6.6 shows the standard deviation of all 3 coordinates from the translation vector from each frame. The standard deviation, σ, and relative standard deviation, σ_{rel}, are calculated as seen in Equation 6.2, where $\overline{x}$ is the average measurement.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{6.2a}$$

$$\sigma_{\text{rel}} = \sigma/\overline{x} \tag{6.2b}$$

Table 6.7 shows the relationship between the standard deviation of measurement of the small and large markers. The relationship is calculated by dividing the standard deviation of the smaller marker reading by the standard deviation of the reading of the larger marker, $\sigma_{\text{small}}/\sigma_{\text{large}}$.

Table 6.6.: Data of precision of new markers in new positions

Small/Large	**ID**	**Coordinate**	**σ for each coordinate [mm]**	**σ_{rel} for each coordinate**
Small	0	x	2,01	0,00426
		y	0,45	0,00572
		z	4,05	0,00434
Large	0	x	0,45	0,00095
		y	0,04	0,00050
		z	0,78	0,00084
Small	1	x	0,49	0,00161
		y	0,26	0,00123
		z	1,69	0,00175
Large	1	x	0,33	0,00111
		y	0,27	0,00128
		z	1,26	0,00132

Table 6.7.: Comparing the standard deviations of the small and large marker.

ID	**Coordinate**	**Relationship std.dev.**	**Relationship relative std.dev.**
0	x	4,50	4,47
	y	11,51	11,43
	z	5,20	5,18
1	x	1,48	1,45
	y	0,99	0,96
	z	1,35	1,33

6.3.4. Pick-Up Testing With New Marker Positions

The new markers and marker placements having seemingly small standard deviations motivated further testing for picking up the tube in the incubator. This subsection presents 2 testing sessions. These sessions were performed after the testing with water, section 6.2.

Each testing session consisted of 15 gripping attempts, 5 in each zone illustrated in Figure 6.4. A hand-eye calibration was performed prior to each testing session. Some preliminary testing was done to verify the calibration of each session. Both calibrations had similar results as the best results in Table 6.1.

The preliminary testing to the first session revealed a bias in the positioning of the gripper, and a 5mm change in the y-parameter of the gripper-robot relationship was added. The results in testing session 1 noted "Tube frame too high" resulted in the z-coordinate of the tube frame in testing session 2 being the 20^{th} highest point instead of the highest and 10^{th} used in testing session 1.

Results

The results of the sessions are presented in Table 6.8 and Table 6.9, with some notes.

- "Tube not horizontal" means that the tube itself was laying crooked on the pillow in the incubator, in a position in which the robot could not pick it up even when jerking downward and hitting the grooves. Sometimes it would start crooked, and then "pop" into position to lock. Figure 6.10 shows an illustration of a non-horizontal tube in the incubator seen from the side.
- "Tube frame too high" means the vertical position of the tube frame is higher than the actual top of the tube. This can result in the gripper never reaching a position to lock.
- "Collision bound" means the robot positioning in y was poor and continuing the attempt would result in a collision with the incubator.

Figure 6.10.: Illustration of a tube not laying horizontally in the incubator seen from the short side

Table 6.8.: Results Session 1

Zone	Well positioned	Successful	Notes
1	5	3	Attempt 3 - Tube not horizontal Attempt 5 - Tube frame too high
2	5	3	Attempt 1 - Tube frame too high Attempt 2 - Tube frame too high
3	4	3	Attempt 2 - Tube not horizontal Attempt 3 - Collision bound

Table 6.9.: Results Session 2

Zone	Well positioned	Successful	Notes
1	5	5	
2	5	4	Attempt 2 - Tube not horizontal
3	5	5	

During the preliminary testing before conducting the second session, the tube frame was placed out of reach of the robot. This resulted in the robot controller stopping the robot due to a soft, or programmed, joint limit being reached. The position of

the incubator and tube was not extreme, and the attempt is pictured in Figure 6.11 where the robot is in the position in which it stopped.

Some videos of lift attempts are found in the digital appendix C.3.

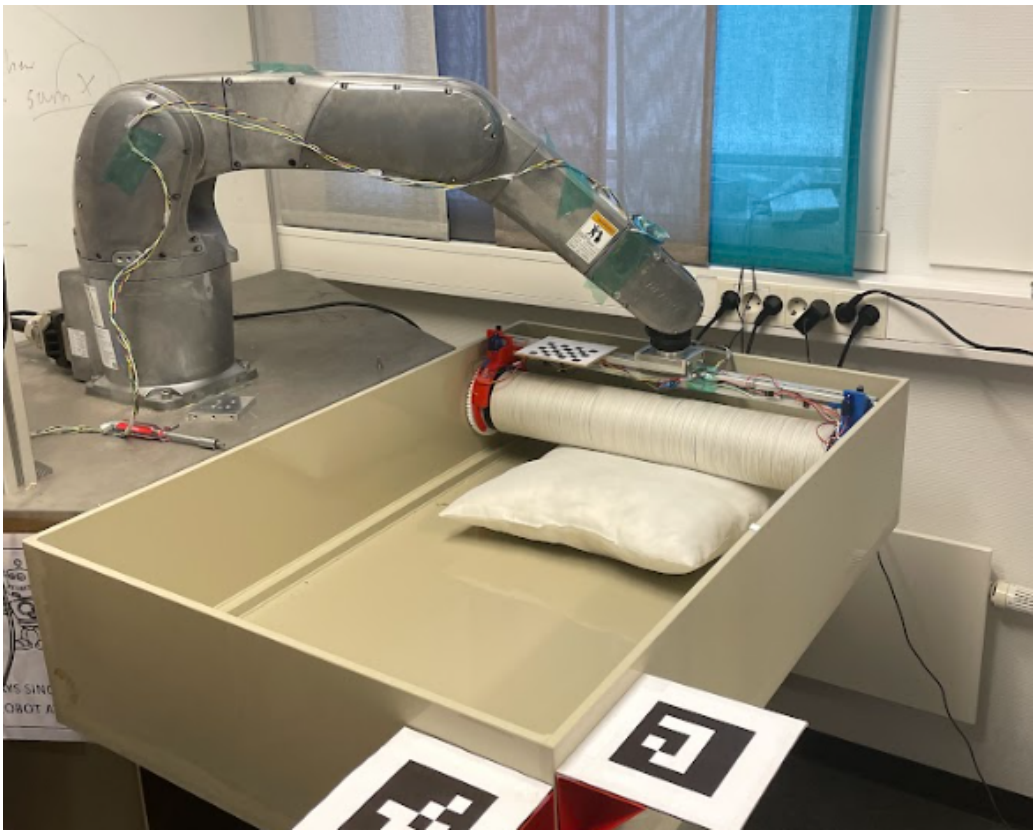


Figure 6.11.: Joint 2 reaching its soft motion limit.

Chapter 7.

Discussion

This chapter contains discussions regarding the function of the developed system. The discussion is mainly based on the findings in chapter 6. This chapter is divided into 4 sections. The first 3 sections cover the topics presented in chapters 3, 4, and 5 respectively. Potential implementations of the system for task solving are discussed in section 7.4.

7.1. Discussion of Robot Cell Setup

The robot cell was provided by SINTEF Ocean. Elling Ruud Øye, a researcher at SINTEF, provided a tutorial on the robot including the teach pendant as well as the "Python library → LabView → controller → robot" pipeline. His tutorial and experience-based advice provided confidence in the choice of having the robot cell and signal pipeline as presented in chapter 3. This section discusses the functionality of the robot testing cell in terms of the project objectives and the testing conducted.

7.1.1. DENSO VS-087 Robotic Arm

This robot was chosen in the prestudy due to its availability, payload, and water-proofness. The environment it is supposed to work in is likely very humid, and the risk of contact with liquid salt water is very high.

This robotic arm model was capable of lifting the substrate tube given a suitable gripper. This was proven by every successful lift performed in this project period, like the ones presented in section 6.3. The control of the robot, both through the teach pendant and the Python Library, was highly functional.

The robot is capable of moving very fast. It is not a co-bot, meaning it should be a safe distance between the robot and humans when in action. This makes it less applicable in a potential scenario where manual and robotic labor is used. When testing the robot using higher speeds, the camera stand started oscillating, meaning

a more rigid camera stand and mount would be necessary if the robot is to work at high speeds.

Reach

The robot has a given reach limiting its task space. As presented in the pick-up testing results in 6.3.4, the dimensions of the incubator could present reach issues, even when the incubator is placed seemingly close. If the robot is mobile at some stage, this mobility should include systems to place the robot within reach of the tubes or be used actively to increase the reach. Some reach could also be added by having a height-adjustable robot base or redesigning the gripper.

Collisions and Singularities

When the robot and controller detected a collision it stopped the movement and required a manual reset. A form of collision detection would be a great improvement to the system in terms of both efficiency and safety. The payload of the robot model is 7 kg and has the potential to damage or push items if they are in its path. The gripper also has relatively large dimensions, and the camera mount is close to the robot. This could make self-collisions and collisions with the camera mount possible.

Utilizing the robot's own torque measurements could be a way of avoiding damaging collisions. The DENSO VS-087 is not marketed as having these capabilities, but more research on the matter or communication with DENSO might reveal the capabilities of the robot and controller not known in this project. If the gripper is in a collision and this is detected before any damaging force is applied, this could increase the robustness of the system. In the case of a potentially damaging force being applied, the system should probably be manually inspected and reset. The gripper could also be redesigned to include some form of collision detection.

When approaching a singularity while moving, the system also stops due to joint speed limits being reached. This was especially the case when moving linearly between positions in the hand-eye calibration process. The reason for linear movement was predictability by visually confirming collision-free movements. A thorough design of a task space accounting for both the gripper, camera mount, and incubator, combined with careful path planning should help eliminate some risk of collisions and singularities.

7.1.2. Intel RealSense D415 3D Camera

The Intel RealSense D415 camera is not a state-of-the-art 3D camera for industrial settings, but it has proved sufficient for the pick-up task focused on in this project. The D415 unit combined with the RealSense software development kit

(SDK) proved capable of creating detailed enough point clouds. Through cropping and clustering the point cloud, combined with marker detection, it was possible to create a tube frame with high enough accuracy for the system to pick it up from an arbitrary position in the incubator.

The unusable point clouds mentioned in subsection 6.2.2 were an issue. This seemed to happen when multiple point clouds were created within a short time. This could be a hardware or software issue on the RealSense camera itself, occurring when many point clouds are generated within a short time. It also happened during the development phase of the object detection system, the circumstances were similar. It did not occur during the pick-up testing where more time was given between each point cloud generation.

Calibration

Looking at the results in Table 6.1, the hand-eye calibration performed using checkerboard-calibrated intrinsic parameters yielded better results than when using the internal calibration parameters. The point clouds were generated using the internal intrinsic parameters. As the accuracy of the point cloud was not as critical as other aspects of the computer vision system, these parameters were not changed and compared.

7.1.3. Absence of Water

The fact that no pick-up testing was performed with a water-filled incubator leaves a gap in data regarding the performance of the system for the intended tasks. The main reason for no such testing being conducted was the grippers not being waterproofed. The robot's capabilities of lifting the tube when wet are assumed adequate with good margins based on the weighing in the prestudy [6] and the payload provided in the robot's documentation.

The pillow used to simulate flotation and prevent collisions would occasionally result in the tube not being horizontal, as presented in the pick-up testing in section 6.3.

The testing of the computer vision system with water present, presented in section 6.2, provides some indication of the performance of the system with water in the incubator. This is discussed further in section 7.2 and section 7.3.

7.2. Discussion of Grippers

This section presents some discussion on the impressions and data gathered on the different gripper prototypes. Due to the grippers not being the main focus

of this project, this discussion is brief but relevant for the overall discussion and recommendations of future works.

As mentioned in section 4.1, this gripper was primarily constructed in order to test the robot and prove the tubes could be picked up by the DENSO VS-087 unit. The structural integrity was lacking, and compared to the other grippers, the compactness of the gripper carrying a tube was not satisfactory. Some redesigning with structural rigidity and an increase of compactness in mind would drastically improve this concept. A simple modification such as mounting to the aluminum profile side and not the bottom could be an improvement. Sensor feedback would also improve this gripper. An option for this could be to monitor the change of current as the motor stalls, i.e. when the gripper is pushing in on the tube. The gripper was also quite slow when changing from "open" to "closed" with the current DC motor.

The compactness is drastically improved with the folding fingers concept compared to the servo-electric gripper prototype. Comparing the gap between the extruded aluminum profile and the tube when gripping in Figure 4.2b and Figure 4.4, the improvements are very clear. The implementation of the hall-effect sensors made the folding fingers gripper very applicable for the pick-up testing. The Arduino-based control of the locking mechanism also performed well.

The electromagnetic gripper, being made shortly before the change of focus in the project, was also promising. It was only tested briefly after completing the prototype, but the results were satisfactory. A considerable force was needed to pull the tube from the gripper. The compactness and the gripping force were both satisfactory. Implementing functioning feedback and automatic control would make this gripper a serious contender to the folding fingers gripper.

7.2.1. Function With Water

The results in part two of the experiment in section 6.2 revealed that the buoyancy was less than anticipated. This could be an issue if it forces the robot to push the tube to the floor to achieve a locking position. A push against the floor could register as a collision, stopping the system. If the buoyancy of the tube is not enough to push the tube into a position to lock, this might rule out the folding fingers gripper from the discussion. However, the unit with the loosest joints proved that the buoyancy might be enough to push the tube in position to lock.

The electromagnetic gripper is not reliant on the buoyancy to exert a certain amount of force to grip the tube. However, the gripper is reliant on both gripper units coming in contact with the gripping area. The gripping procedure described in subsection 6.3.1 orients the gripper horizontally before gripping, and therefore depends on the tube floating horizontally.

In terms of submersion, the servo-electric gripper might have an advantage. The length of the gripper units could keep the electronic components at a safe distance from the water. This could reduce the need for waterproofing the gripper. The fact that only the fingers, and not the aluminum profile, are likely to touch the water would also reduce the amount of the gripper to clean to prevent contamination when performing tasks in different incubators.

7.2.2. Accuracy Dependance

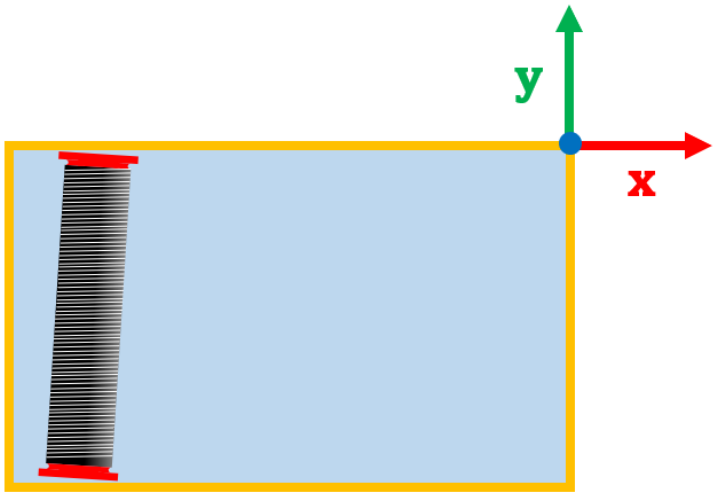


Figure 7.1.: Coordinate system of incubator

All gripper concepts are comparable when it comes to the need for accuracy in positioning to grip. Looking at the coordinate system defined in Figure 7.1, the grippers are all capable of picking up the tube despite an error along the x-axis. This is mainly due to flotation and the circular shape of the tubes. The folding fingers gripper has the greatest width in the x direction, making it more robust for this error. However, some cases involving an error in the x-axis might result in a collision.

- If the incubator is full of tubes, or the tubes are close and one is close to the wall, an error in x could result in the tube being pushed into the wall directly or through a chain collision, as illustrated in Figure 7.2.
- If the tube is lying close to the short walls, an error in x might result in a collision with the short wall if the gripper is placed directly above the wall before its vertical movement to grip.

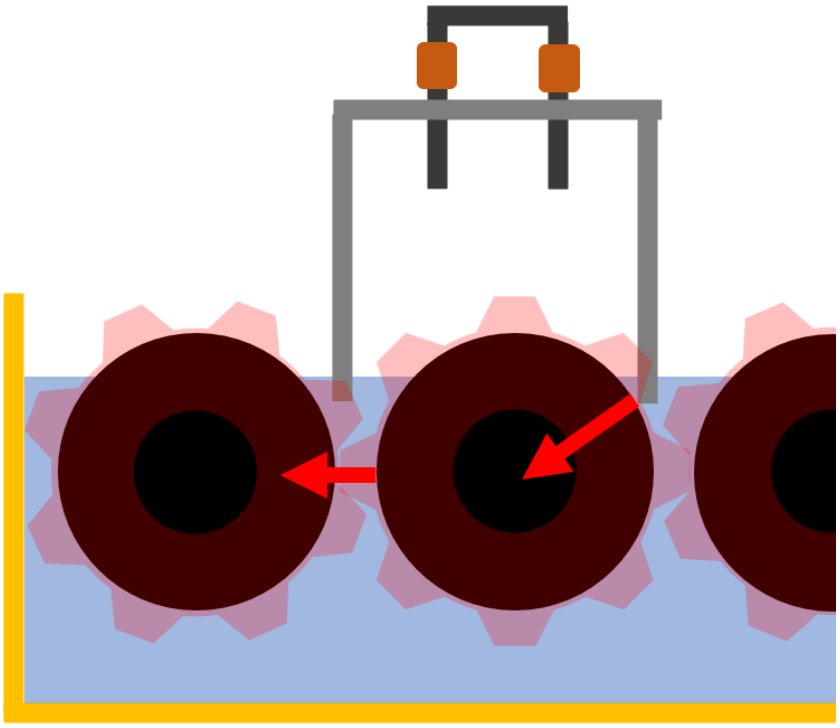


Figure 7.2.: Collision scenario if the incubator is full and there is an error in x.

As no pick-up testing has been performed with an incubator full of tubes, an error in x has not been an issue. An error in y on the other hand has been the cause of multiple failed attempts, as the gripper is reliant on a quite accurate positioning in y to hit the grooves.

7.2.3. Mounting

As presented in the pick-up testing results, 6.3.4, the robot reached a soft motion limit while attempting to pick up a tube. Neither the incubator nor the tube was placed in any extreme position. This revealed how the window for placing the incubator relative to the robot is quite small. A solution to increase the reach could be to redesign the gripper. An option could be to mount it asymmetrically as pictured in Figure 7.3. This mounting would introduce a torque exerted on the robot end, as the center of mass would be shifted. This would need to be researched and tested before a commitment to changing the mounting.

Figure 7.3.: An asymmetric gripper mount of the gripper could increase the reach of the robot, but torque would be introduced due to the center of mass being shifted.

7.3. Discussion of Computer Vision System

The computer vision system was functional in terms of locating the substrate tubes relative to the robot, which was one of the main objectives of this project. This section discusses the systems presented in chapter 5 based on the testing and results in chapter 6.

7.3.1. Object Detection

The object detection algorithm presented in section 5.1 was capable of both determining amount of tubes in the incubator and their position relative to the camera. The results of the pick-up testing in section 6.3 prove it is capable of detecting and positioning one tube with high enough accuracy to pick it up. However, the results of the testing with water, section 6.1, revealed some inconsistencies.

The idea of using more than a snapshot, maybe a continued stream of point cloud and RGB images was visited in the development process. The occlusion of the incubator and substrate tubes, while the robot is moving, would likely reduce the quality of that data. The RealSense camera was also not consistent in the point cloud generating when attempting to run a continuous stream of point clouds using the RealSense Viewer software.

Cropping and Clustering

The technique of cropping and clustering the point cloud of the incubator in order to isolate the tubes in the point cloud shows promise.

Looking at the test results of the testing with water in Table 6.2, there are inconsistencies to address. Some of the inconsistencies can be blamed on the waviness of the markers as seen in Figure 6.1b. This was likely due to the high level of humidity in the air at the seaweed laboratory. Another case is a single tube being clustered into multiple detections. This could be avoided in the algorithm by only allowing one tube within a certain space, as it is physically impossible for two tubes to be within a certain space.

As for the results excluding tubes, one could argue that this is not a critical error. The flotation of the tubes will likely lead to drifting when removing a tube, and a new snapshot and detection should be made before any lifts. As long as a tube intended for lifting is localized accurately enough, the system will continue.

Drifting could cause problems if the distance drifted is too great between the camera snapshot and the lifting attempt. The drifting only occurs when the incubator is not full and will happen along one axis. This is the one axis the grippers are capable of handling inaccuracies, as discussed in section 7.2. The robot is also capable of moving very fast, and the drifting might not be relevant.

Some improvements to the clustering could be to explore other values of the parameters and even other clustering algorithms. The ideal "ϵ" and "minimum points" might change if the camera distance or the voxel down-sampling of the point clouds differ from the testing done in this project. Changes could affect the resolution of the point cloud, i.e. the density, and DBSCAN is dependent on the distance between points.

For the testing performed in section 6.2, ϵ in the range of 40-43 yielded the best results with z-cropping being at $z_{\max} - 30$. These results were 68-82% accurate in terms of isolating and counting. Improvements could be to produce more solid markers and mount them more planar.

Other cropping techniques could also be introduced, such as plane detection. This could be introduced to recognize the incubator. Especially the cropping about the vertical axis is vulnerable and very critical. If the detected z-axis of the marker is not close enough to vertical, the cropping about this axis that spaces the tubes before clustering would be inadequate. This is likely the case for multiple unsuccessful detections.

Other improvements to the isolation of tubes could be to utilize the geometric shape of the tube point clouds to implement some form of cylinder fitting. Another option could be to use the 3D and/or RGB data to train a machine-learned model. This would require gathering a large set of data. This could work as a new stand-alone method, or as a supply to increase the robustness of the already existing cropping and clustering method. Using RGB data, or even black and white images, for machine learning would need to account for the color and texture changes as

the seedlings grow on the incubator. The tube end-pieces, being 3D-printed could potentially be made in high-contrast colors. This data would also be possible to use in further improving the of the detection and placement of the tubes.

Positioning of the Tube Frame

Using the reference frame of Figure 5.7, the most critical part of the tube positioning is the placement of the tube frame along the y-axis, as an inaccuracy would result in a collision. This is highly dependent on the detection of the markers. The standard deviation of the marker detection in the new positions, listed in Table 6.6, are all under 1.3mm for detections of the larger marker. Given the results from the pick-up testing, this is sufficiently accurate.

The accuracy of the marker detection is also attributed to the camera calibration. Looking at the pinhole camera model, subsection 2.3.1, the solution to the pixel-3D relationship is highly dependent on the intrinsic parameters. The calibration using the calibration functions of OpenCV with a suitable checkerboard has proved sufficient enough for the system to perform its task.

Comparing the standard deviations of the larger and smaller markers in the new positions, Table 6.7 shows a correlation between the size of the marker and the standard deviation of the position detection. For the y-coordinate of Marker 0, the standard deviation is as much as 11 times higher when using a smaller marker. For Marker 1 the difference is not as extreme, and y is marginally better. The takeaway, however, is that a larger marker should and does provide a more precise reading. Changing the size of the marker is easily accounted for in the code, and the decision for the final marker size could be tested and determined based on the space available in the future.

Having positions with a less steep angle to the camera, as well as a larger white surrounding frame has eliminated some of the issues experienced in the testing with the markers inside the incubator, as presented in subsection 6.3.3. These issues were a misreading of the marker's corner, resulting in a considerable error.

The positioning of the tube frame along the vertical axis using the highest point of the tube has shown some inconsistencies, as seen in Table 6.8, where the tube frame was too high for the gripping to be successful. Later pick-up testing used the 10^{th} and 20^{th} highest points to account for noise. This was functional.

The positioning along the x-axis is not as critical, with the exception of the two cases discussed with the grippers in section 7.2. Below the cases are listed with possible solutions.

- Case of collision between the gripper and the short wall:
  - The exaggerated cropping presented in subsection 5.1.2 could decrease

the risk as the possible positions of the tube frames would be further from the wall.

- Case of a collision or a chain collision between tubes and the short wall with a full incubator:
  - The risk could be decreased by having the lifts always be the tube most vertically below the camera. This would reduce the camera bias issue presented in subsection 5.1.4.

7.3.2. Hand-Eye Calibration

The relationship between the camera and the robot must be sufficiently accurate for the robot to interact with the camera-detected environment. The pick-up testing section 6.3 reveals that an adequate calibration is possible with the method used.

The results in Table 6.1 also reveal some inconsistencies in calibrations performed with seemingly identical parameters. The two testing sessions presented in subsection 6.3.4 had very similar calibration results in terms of translation, but the preliminary testing before Session 1 performed better after an adjustment in the tool frame parameters.

The method of determining the accuracy of the hand-eye calibration is just an indication, as it does not directly measure the rotational errors. This could be the reason for two seemingly similar calibrations to yield different results when performing pick-up testing.

Another reason could be changes in lighting. The robot cell was close to a window, and the variation of lighting could result in different results in the corner detections. This could explain why in all calibrations 10-14, only calibration 14 detected a checkerboard in 19 images. All calibrations had 20 identical poses for taking 20 snapshots, and all except 14 found checkerboards for only 18.

Small vs. Large Checkerboards for Calibration

Table 6.1 shows that the small checkerboard was not sufficient for an accurate hand-eye calibration. Calibration 1 in Table 6.1 was the first calibration to be measured using this metric and was performed after multiple poor hand-eye calibrations with the small checkerboard. Calibration 2 was done to compare how using the precalibrated parameters would affect the result. The calculation of the transformation between the camera and robot base differs by more than 7cm at the most accurate coordinate and an extreme 27cm at its worst. Unusable for this system.

A reason for the smaller checkerboard yielding less accurate results could be due to the reduced size resulting in less distinct and recognizable corner features, making

it more challenging to precisely identify and locate them. The smaller dimensions may also have led to a higher pixel-to-corner ratio, increasing the likelihood of errors caused by image distortions or noise.

The large checkerboard calibrations yielded much better results. Ultimately, the gap would reach a level of less than a 5mm gap, less than 1% for all coordinates. This is a strong indication of accurate hand-eye calibration, also proven by the successful pick-up testing.

An issue with checkerboard calibration is the symmetric properties of the detected corners. There is a possibility of the checkerboard being detected as rotated 180° about two different axes. Using a square checkerboard would also introduce the possibility of detecting it as rotated 90° and 180° about the last axis. This was assumably not an issue, as the most extreme calibration results did not have an individual pose+image as the source of the largest gap.

To reduce the risk of this occurring, monitoring the direction of the normal axis of the plane of the detected corners. Fiducial markers, or sets of fiducial markers like ChArUco boards[1], could also be a way of detecting a known pattern. These markers account for orientation, as opposed to a checkerboard.

Precalibrated vs. Checkerboard Calibrated Intrinsic Parameters

An interesting finding in the hand-eye calibration data presented in Table 6.1 is the fact that performing a hand-eye calibration using the internal intrinsic parameters of the RealSense Camera yielded less accurate results than the intrinsic parameters calculated using OpenCV. A noticeable difference in the parameters is that the RealSense states the distortion coefficients are zero while the distortion coefficients from the checkerboard calibration are not. If there is a significant lens distortion this should be included in the intrinsic parameters and might be relevant to why the checkerboard calibration outperforms the internal intrinsic parameters.

Comparing calibrations 10, 11, 12, and 13 in Table 6.1, it is clear that 10 and 13 yielded better results. 11 and 12 yielded the best results using precalibrated intrinsic parameters, but still resulted in gaps more than twice as large as 10 and 13 in both millimeters and percentage.

For the calibrations performed using a small checkerboard, calibrations 1 and 2, the precalibrated parameters yielded better results. This is likely due to a checkerboard calibration using the small checkerboard being very poor.

[1] https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html

Using More Permutations

Comparing calibrations 9 and 10 with 13 and 14 in Table 6.1, all resulted in maximum gaps less than 4mm. 9 and 10 both used 17 permutations, while 13 and 14 used 46 and 53 respectively. There are marginal differences in these results when comparing the gaps for all coordinates, both in percentage and millimeters.

The least squares method tends to yield better results with a larger number of samples because it minimizes the overall error between the observed data points and the fitted model. However, the utilization of more permutations between the different positions in the calibration yielded little improvements according to Table 6.1. This could indicate that the ≈ 20 permutations of images and poses and images are enough for the trend to converge on a minimum error with the given camera, camera intrinsics, and checkerboard.

7.4. Implementation of System

The system developed and tested in this project deals with a specific task of the production method described in section 1.1. Specifically, the task of picking up the tube from the incubator. This was considered the most challenging task as the tube is floating and must be located. The concepts developed and researched in this project should translate to other tasks of production.

Assume the robot is placed in position in front of a full incubator and the camera is placed with both markers and all tubes sufficiently visible. A hand-eye calibration has been performed and no movement has been done to the camera. A storage container is also within reach, and a number of tubes are known due to the incubator being full. The process could be to perform object detection and pick up the tube most vertically below the camera until the incubator is empty. Even though the results of the object detection testing with multiple incubators in water were inconsistent, the system should know how many tubes are left in the incubator by counting each successful lift, as the total amount is known.

Placing substrate tubes in the incubator should also be easily done by using the same method of determining the center of the incubator using the fiducial markers. The need for placing the tubes evenly spread out should not be necessary, due to their cylindrical shape and flotation. If they are placed on top of each other they roll over and space out naturally.

If the storage container is equipped with ArUco markers with different IDs, a placing method could be established easily by determining the center of the container using the same method as for the pick-up algorithm. Figure 7.4 shows an illustration of how the storage could be placed on a potential future mobile robot platform.

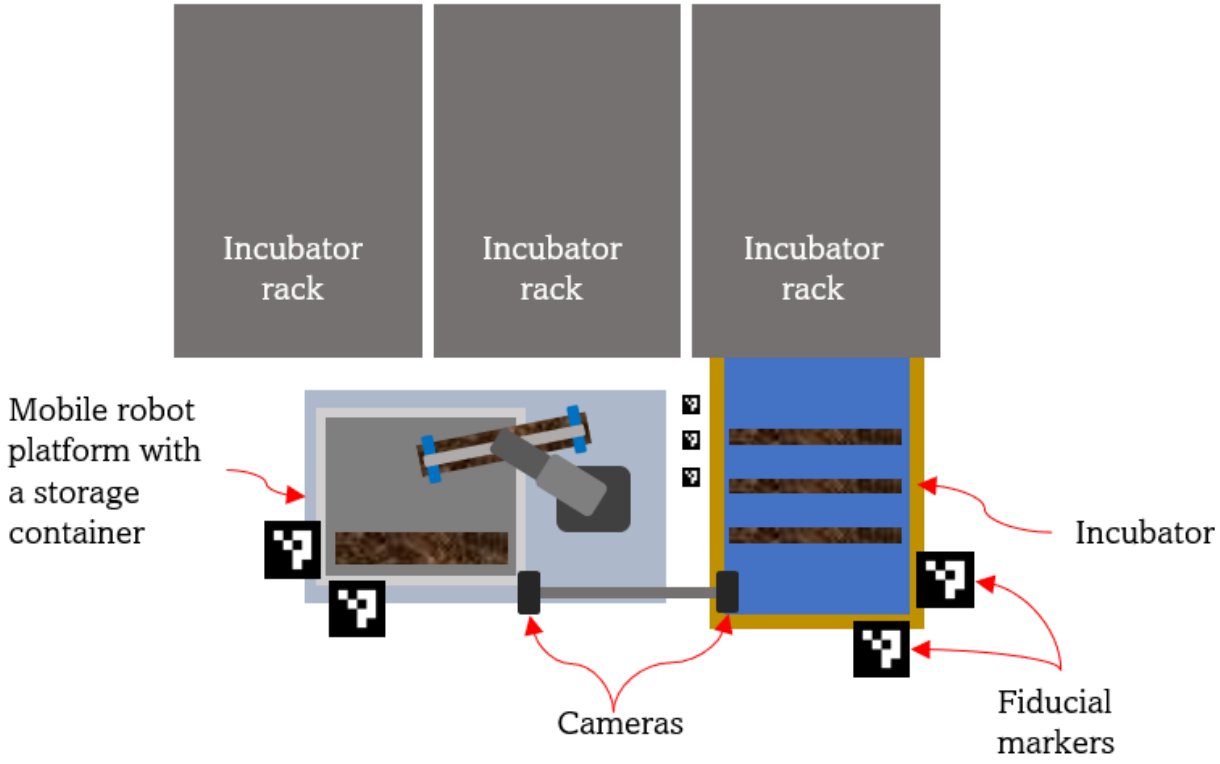


Figure 7.4.: A future scenario with the robot mounted on a mobile platform equipped with a storage container. The amount and placements of cameras would need to be determined based on needs.

Chapter 8.

Conclusion and Future Works

As stated in the project objectives in section 1.2, the main objective of this project was to create a proof of concept for a vision-guided robotic system for handling substrate tubes. The system would include the incubator and substrate tube used in the production method presented in section 1.1, DENSO VS-087 robotic arm, RealSense D415 camera, and a specially designed gripper. The results of the pick-up testing presented in 6.3 combined with the promising results of the testing with water, 6.2.2, prove the system's feasibility.

Assuming the robot's accuracy is as good as its stated repeatability of ±0.03mm, the functionality of the system relies on the accuracy of the computer vision system. The marker detections, hand-eye calibration, and point cloud generation all have room for improvement, but the fact that the system is capable of picking up a randomly placed tube from the incubator is undeniable, proving the sum of accuracy sufficient. The computer vision system for detecting tubes is applicable already, as long as an accurate hand-eye calibration is performed.

The electromagnetic and folding finger gripper concepts have proven functional and could both be developed further. They are both capable of sufficiently lifting the tube. Further testing with water would be important to ensure functionality in the actual environment the system is supposed to work in.

8.1. Future Works

Going forward, the issues and solutions presented in the discussions should be explored, especially regarding collision avoidance. Path planning and movement procedures should be developed carefully, and the use of camera data could be implemented in this. For one robot to tend to more incubators it should be made

mobile. If a DENSO VS-087 robotic arm is not used, the range, waterproofness, and payload should be equivalent or better.

In order to solve more tasks, the application of fiducial markers has proven reliable. Using fiducial markers for locating other parts of the process should be considered in further development. They could also serve as positioning tools if the robot is made mobile in a future stage.

Both the readings of the fiducial markers and the hand-eye calibration are highly dependent on accurately calibrated camera intrinsics. The current calibration is sufficient, but a systematic approach with more quality images of checkerboards, and maybe a more accurate and planar checkerboard could help on increasing the accuracy. An option could also be to use a camera with better pre-calibrated intrinsic parameters.

The gripper is critical if going forward with a robotic arm system. If proceeding with the folding or the electromagnetic gripper with the current dimensions, they would need some extensive waterproofing of electronics. This waterproofing would be important for cleaning and performing tasks. Other recommendations would be to explore a non-symmetric mount to increase the reach, improve the control system, and implement a more suitable power supply.

References

1. Meland M and Rebours C. Short description of the Norwegian seaweed industry. *Bioforsk-konferansen 2012*. Ed. by Fløistad E and Günther M. Bioforsk Fokus 7(2). Bioforsk, 2012 :275–7
2. Stévant P, Rebours C, and Chapman A. Seaweed aquaculture in Norway: recent industrial developments and future perspectives. Aquaculture International 2017; 25:1373–90. DOI: https://doi.org/10.1007/s10499-017-0120-7
3. Solvang T, Bale ES, Broch OJ, Handå A, and Alver MO. Automation Concepts for Industrial-Scale Production of Seaweed. Frontiers in Marine Science 2021; 8. DOI: https://doi.org/10.3389/fmars.2021.613093
4. Lillienskiold A, Nilsen AT, Kvæstad B, Michelsen FA, Stefanakos C, Ole Jacob Broch HD, and Silje Forbord SU og. Fremtidens tareindustri. Tech. rep. SINTEF Ocean, 2021
5. Alver MO, Solvang T, and Dybvik H. State of the art. Tech. rep. D5.4. SINTEF Ocean, 2018
6. Stubberud V and Midtun J. Automation of Land Based Production of Seaweed (Appendix C.1). 2022
7. Lynch KM and Park FC. Modern Robotics. Cambridge University Press, 2017
8. Hartley R and Zisserman A. Camera Models and Computation of the Camera Matrix P. *Multiple View Geometry in Computer Vision*. 2nd. Cambridge University Press, 2003 :153–92
9. Aanæs H. Lecture Notes on Computer Vision. Lecture Notes, 02504, DTU. 2018
10. Burger W. Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation. Tech. rep. HGB16-05. University of Applied Sciences Upper Austria, 2016 May. DOI: 10.13140/RG.2.1.1166.1688/1
11. Zhou QY, Park J, and Koltun V. Open3D: A Modern Library for 3D Data Processing. arXiv:1801.09847 2018
12. Ester M, Kriegel HP, Sander J, and Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Knowledge Discovery and Data Mining*. 1996

13. Park F and Martin B. Robot sensor calibration: solving AX=XB on the Euclidean group. IEEE Transactions on Robotics and Automation 1994; 10:717–21. DOI: 10.1109/70.326576

14. Toulson R and Wilmshurst T. Analog Output. *Fast and Effective Embedded Systems Design*. Newnes, 2012 :69–89. DOI: https://doi.org/10.1016/C2015-0-00101-0

15. Ramsden E. Hall Effect Physics. *Hall Effect Sensors*. 2nd ed. Newnes, 2006 Jan :1–10. DOI: 10.1016/B978-075067934-3/50002-8

Appendix A.

Appendix A - From SINTEF

A.1. Initial Project Proposal

To:

Whom it may concern

From:

Aurora Nilsen
at SINTEF Ocean
Brattørkaia 17C
7010 Trondheim
aurora.nilsen@sintef.no

Your ref.	Our ref.	Project No. / File code	Date
Aurora Nilsen	AutoHatchery for Seaweeds, Aurora Nilsen	Pr.nr. / File code	2022-08-25

Topic: *Specialization project with attached masters.*

Auto-hatchery for Seaweeds: robotic use with vision interactions, and tool design.

Seaweed hatcheries are currently a fully manual operation. However, with the increasing demands and high future production volumes this cannot prevail. Automation is going to be essential when Norwegian production increases thousand-folds. Today we are able to produce around 40.000 km for habitual-rope (finished spun substrate-rope), however in a mere eight years this production is supposed to be 800.000 km. At SINTEF we run multiple projects focused on making this possible. Our current challenge is scale up. In this project we want to make a robotic system that can load and unload substrate-tubes – both for winding of rope and placed in hatcheries.

In these assignments, the candidate will work with researchers at SINTEF Ocean to improve on a current design concept, develop and build the mechanical tool for interaction with substrate-tubs. Program and deploy a vision-guided robotic control for placement of substrate-tubs in a winding-machine and a hatchery-pod. The tasks are divided into increments, and can be edited upon our choice of candidate and or heading of current research.

Tasks:

1. *Specialization project*

   In the specialization project the focus will be establishing working designs for mechanical parts for a gripper. Mounting sensors, simulate mechanical strengths and weakness, building function for interaction.
   - Slim, simple design.
   - Simple assembly.
   - Parts and connections.
   - Sensor integration/fusion.
   - Strength analyses of structure and load capacities.

2. *Master's thesis*

   Building a rig with a robotic arm, automated. The goal will be a setup a 7-DOF-robotic-arm. Establish a system ready for implementation interaction with substrate-tubs. Placing substrate-tubs in hatchery-pods and in winding machines – with the use of vision-tracking-control.
   - Create a setup for use of robotic to automated handling of substrate-tubs.
   - Validate design and setup.
   - Execute a demonstration of handling substrate-tubs.

Prerequisites:

- Knowledge in robot gripper design.
- Drawing in 3D, preferably SoildWorks – for use in control of the gripper.
- Knowledge and interest in 3D-printing and building DIY projects.

Other:

The candidate will be provided access to SINTEF Ocean's 3D-printing facilities and robotic laboratory along with other need be equipment.

Supervisors	Names	Affiliation
Main Supervisor:	xxxx	Department of xxxxx, NTNU
Co-supervisor:	Aleksander Lillienskiold	SINTEF Ocean
Co-supervisor:	Aurora Nilsen	SINTEF Ocean

A.2. Dimensions of Substrate Tube End Pieces

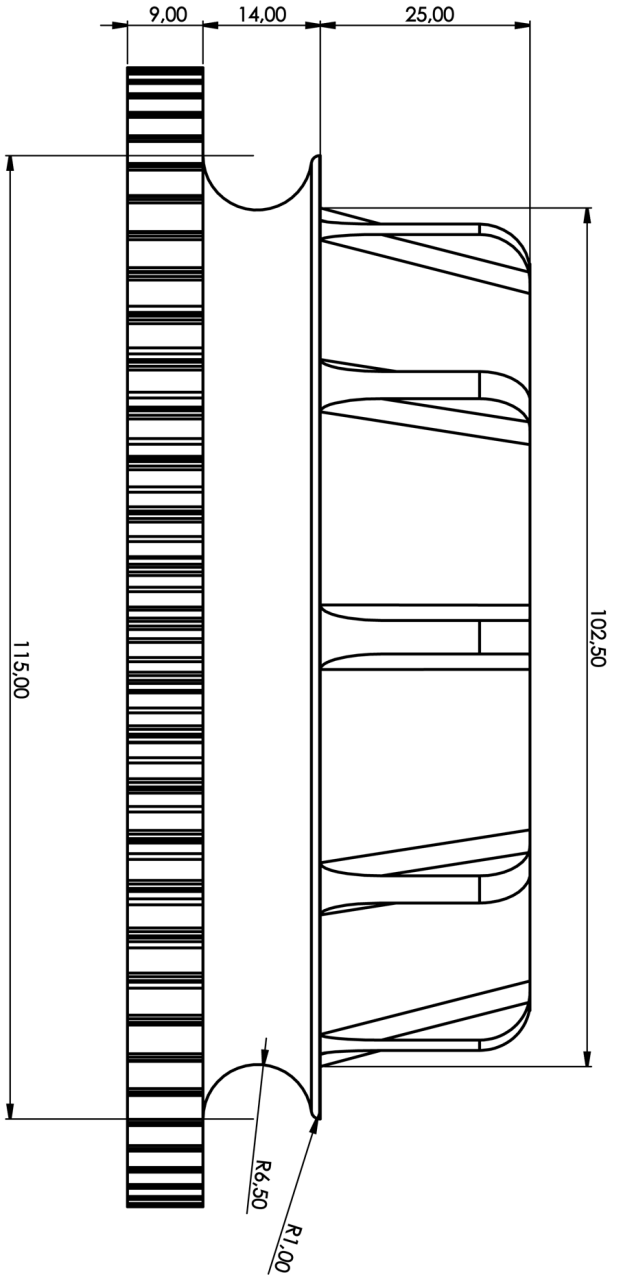


Figure A.1.: Tube end piece without thread gap.

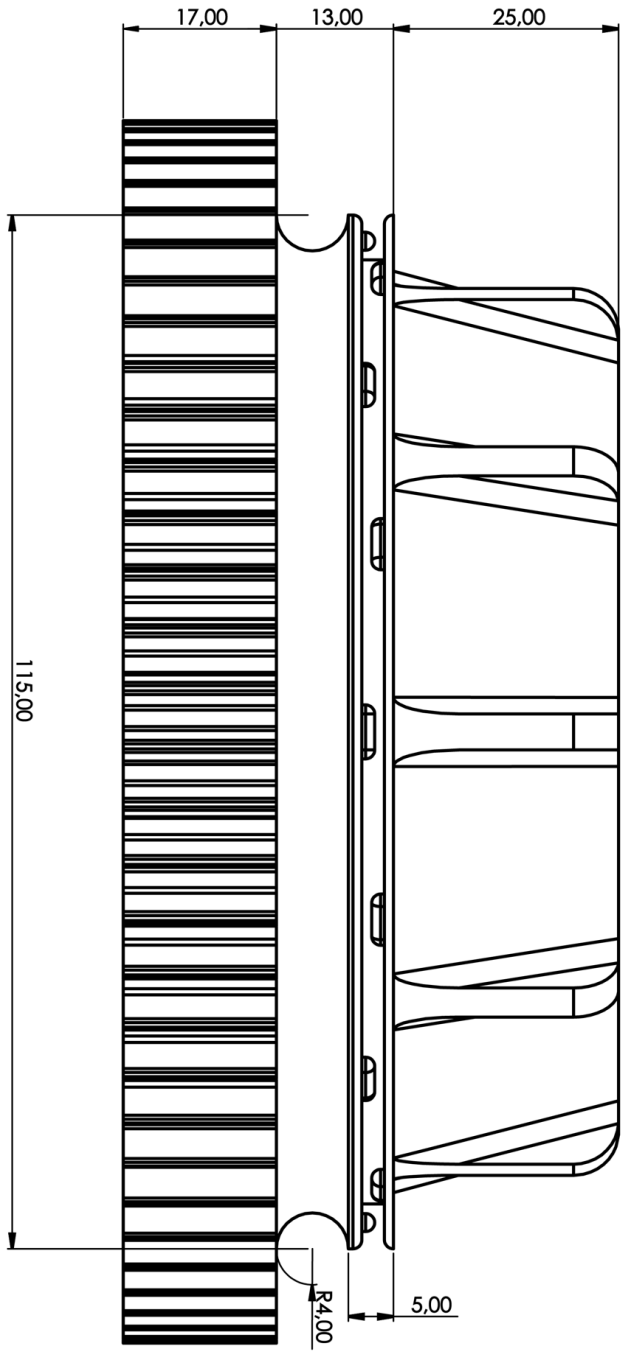


Figure A.2.: Tube end piece with thread gap. The gap is designed to anchor the thread for winding the thread on the cylinder.

Appendix B.

Appendix B - Code

The code developed in this project is available in a GitHub repository https://github.com/jakoboss69/DensoSubstrateTubeHandling. Below are some snippets developed for this project that can be of interest to get a better picture of the object detection algorithm, point cloud generation and establishing a session to perform tasks with the DENSO VS-087.

B.1. Connecting to DENSO Unit

Below is a code that shows how to start and end a session for controlling the DENSO robot. This is based on an example in the DENSO controller repository. The DENSO controller file is found in the GitHub repository for this project[1]. First, the code connects to the computer containing LabView, then sets the robot end as the tool frame. It then creates a home position using cartesian and Euler angles with a specific figure, as seen in Table 3.1, and sends the robot to this position in a linear movement. After tasks are performed, the session is ended safely by the commands at the bottom of the code.

```
    # Connect to LabVIEW program controlling the robot:
    denso_controller = DensoController()
    connected = denso_controller.connect("tcp://192.168.100.100", "
5555")
    print(f"{connected = }")

    if connected:
        # Start a DENSO robot session:
        session = denso_controller.open_session()
        print(f"{session = }")


        # Activate the RoboSlave.pac program in the robot controller
:
```

[1] https://github.com/jakoboss69/DensoSubstrateTubeHandling

```
13          toolkit_status = denso_controller.set_toolkit(True)
14          print(f"{toolkit_status = }")
15
16
17          # Set the speed of the robot, maximum
18          10 external when testing
19          internal = denso_controller.set_internal_speed(100)
20          print(f"{internal = }")
21          external = denso_controller.set_external_speed(10)
22          print(f"{external = }")
23
24          # Turn on the motors:
25          servo_status = denso_controller.set_servo(True)
26          print(f"{servo_status = }")
27
28          # set and move to home position. using tool
29          0 to ensure same every time
30          set_active_tool = denso_controller.set_active_tool(0)
31          print(f"{set_active_tool = }")
32
33          home_position_cart = CartesianPose(320, 0, 650, 180, 0, 180,
     5)
34          denso_controller.move_to_cart(home_position_cart,
    interpolation=Interpolation.Move_L)
35          print(f"{home_position_cart = }")
36
37
38          # Set the speed of the robot befoe ending session.
39          This speed will be used when robot is started again.
40          internal = denso_controller.set_internal_speed(100)
41          print(f"{internal = }")
42
43          external = denso_controller.set_external_speed(10)
44          print(f"{external = }")
45
46          # Set the active tool to tool 0:
47          set_active_tool = denso_controller.set_active_tool(0)
48          print(f"{set_active_tool = }")
49
50          # Move to home position:
51          denso_controller.move_to_cart(home_position_cart,
52                                        interpolation=Interpolation.
    Move_L)
53          print(f"{home_position_cart = }")
54
55
56          ##############################################
57          ########  PERFORM TASKS WITH ROBOT ########
58          ##############################################
59
60
```

```
        # Turn off the motors:
        servo_status = denso_controller.set_servo(False)
        print(f"{servo_status = }")

        # Deactivate the RoboSlave.pac
        program in the robot controller:
        toolkit_status = denso_controller.set_toolkit(False)
        print(f"{toolkit_status = }")


        # Close the DENSO robot session:
        session = denso_controller.close_session()
        print(f"{session = }")
```

Listing B.1: Opening and closing a session for controlling the DENSO unit though Python code via LabView.

B.2. Generating a Point Cloud

```

def getSnapShotPointCloud2(log = False, timestamp = ""):
        """
        parameters:
        log: if True, the point cloud is saved to a file
        timestamp: if log is True, the point cloud is
        saved to a file with the given timestamp
        returns:
        open3d point cloud
        """
        # Create a RealSense pipeline object
        pipeline = rs.pipeline()

        # Create a configuration object for the pipeline
        config = rs.config()
        config.enable_stream(rs.stream.color, 1280,720,
        rs.format.bgr8, 30)
        config.enable_stream(rs.stream.depth, 1280,720,
        rs.format.z16, 30)

        # Enable depth alignment to color frames
        align_to = rs.stream.color
        align = rs.align(align_to)

        # Start the pipeline
        pipeline.start(config)

        try:

                # Wait for a new set of frames from the camera
```

```
            frames = pipeline.wait_for_frames()

            # Align the depth frame to the color frame
            aligned_frames = align.process(frames)
            color_frame = aligned_frames.get_color_frame()
            depth_frame = aligned_frames.get_depth_frame()

            # Get the intrinsics of the color camera
            color_intrinsics = color_frame.profile.
            as_video_stream_profile().intrinsics

            # Convert the color and depth frames to Open3D format
            color = o3d.geometry.Image(np.array(color_frame.
            get_data()))
            depth = o3d.geometry.Image(np.array(depth_frame.
            get_data()))

            # Create an Open3D RGBD image
            rgbd = o3d.geometry.RGBDImage.
            create_from_color_and_depth(
                  color, depth, depth_scale=1/float(depth_frame.
                  get_units()))

            # Create an Open3D point cloud
            pcd = o3d.geometry.PointCloud.
            create_from_rgbd_image(
                  rgbd, o3d.camera.PinholeCameraIntrinsic(
                        width=color_intrinsics.width,
                        height=color_intrinsics.height,
                        fx=color_intrinsics.fx,
                        fy=color_intrinsics.fy,
                        cx=color_intrinsics.ppx,
                        cy=color_intrinsics.ppy))

            if log:

                  # Save the point cloud to a file

                  o3d.io.write_point_cloud("Log\\
                  PointClouds\\pointcloud" + timestamp
                  + ".pcd", pcd)
                  print("Point cloud saved to pointcloud" +
                  timestamp + ".pcd")

            o3d.io.write_point_cloud("Log\\PointClouds\\
            latestPointCloud.pcd", pcd)
            print("Point cloud saved to latestPointCloud.pcd")

      finally:
            # Stop the pipeline when done
            pipeline.stop()
```

```
82         return pcd
```

Listing B.2: Generating a point cloud using the Intel RealSense Camera with SDK and Open3D.

B.3. Detecting and Positioning of Tube

Here is the function detecting and determining the position of tubes in the incbuator. Below are the helping functions.

```
1
2  def determinePosition(preShot = False, visualize = False, log =
       False, timestamp = '', store_pointcloud = False):
3      """
4      parameters:
5      preShot: if True, the point cloud is read from a file, if False,
       the point cloud is taken from the camera
6      visualize: if True, the point cloud is visualized before and
       after cropping
7      returns:
8      positions and frames of the tubes in the camera coordinate
       system
9      """
10
11     if preShot:
12         path = "Log\\PointClouds\\latestPointCloud.pcd"
13         pcd = o3d.io.read_point_cloud(path)
14
15     else:
16
17         pcd = gss.getSnapShotPointCloud2(log = log, timestamp =
       timestamp)
18
19     if log:
20         o3d.io.write_point_cloud("Log\\PointClouds\\"+timestamp+"
       pointcloud.ply", pcd)
21     if store_pointcloud:
22         o3d.io.write_point_cloud("Log\\PointClouds\\latestPointCloud
       .ply", pcd)
23
24     pcd = voxelDownsample(pcd, voxel_size=0.008)
25
26     # convert pointcloud to mm from meters
27     pcd.scale(1000, center=(0, 0, 0))
28
29     # Do a rough crop of the point cloud
30
31     pcd = pcd.crop(o3d.geometry.AxisAlignedBoundingBox(min_bound
       =[-500, -500, -1500], max_bound=[500, 500, 1500]))
```

```
    pcd_for_visualization = deepcopy(pcd)

    # Get the latest AruCo positions as transformation matrices
    T_A0C = np.loadtxt("Log\\TransformationMatrices\\latestNEWT_A0C.csv", delimiter=",")
    T_A1C = np.loadtxt("Log\\TransformationMatrices\\latestNEWT_A1C.csv", delimiter=",")

    if visualize:
        pcd_for_visualization = voxelDownsample(pcd_for_visualization, voxel_size=1)
        # visualize
        o3d.visualization.draw_geometries([pcd_for_visualization, o3d.geometry.TriangleMesh.create_coordinate_frame(size=100, origin=[0, 0, 0])])



    if visualize:
        print("T_A0C: ", T_A0C)
        print("T_A1C: ", T_A1C)

    # crop the point cloud to only include items within the incubator
    pcd = crop(pcd, T_A0C, T_A1C, visualize=visualize)

    # find clusters
    pcd_list = cluster_dbscan(pcd, eps=43, min_points=70)

    tube_frames = False

    if pcd_list!=False:
        tube_frames = []
        center_points_and_angles = []
        for cluster in pcd_list:
            # give the cluster a random color
            cluster.paint_uniform_color([random.random(), random.random(), random.random()])
            tube_frame = findPoint(cluster, T_A1C)

            tube_frames.append(tube_frame)
            if visualize:

                print("Tube frame in camera: ", tube_frame)

    if visualize:
        # visualize
        tube_frames_visualize = []
        if tube_frames!=False:
            for tube_frame in tube_frames:
```

```
                tube_frame_visualize = o3d.geometry.TriangleMesh.
    create_coordinate_frame(size=100, origin=tube_frame[:3,3]).rotate
    (tube_frame[:3,:3], center=tube_frame[:3,3])
                tube_frames_visualize.append(tube_frame_visualize)
            o3d.visualization.draw_geometries([*
    tube_frames_visualize,*pcd_list,  o3d.geometry.TriangleMesh.
    create_coordinate_frame(300, origin=[0, 0, 0])])
            o3d.visualization.draw_geometries([pcd_for_visualization
    ,*tube_frames_visualize,*pcd_list,  o3d.geometry.TriangleMesh.
    create_coordinate_frame(300, origin=[0, 0, 0])])
            print("Frame of tube: ",tube_frame)

     return tube_frames
```

Listing B.3: Determining position for substrate tubes.

```

def rotate(pcd, angle_x, angle_y, angle_z):

    """
    Rotating pointcloud or array around the origin
    """

    if isinstance(pcd, o3d.geometry.PointCloud):
        pcd.rotate(np.array([[1, 0, 0], [0, np.cos(angle_x), -np.sin
    (angle_x)], [0, np.sin(angle_x), np.cos(angle_x)]]), center=(0,
    0, 0))
        pcd.rotate(np.array([[np.cos(angle_y), 0, np.sin(angle_y)],
    [0, 1, 0], [-np.sin(angle_y), 0, np.cos(angle_y)]]), center=(0,
    0, 0))
        pcd.rotate(np.array([[np.cos(angle_z), -np.sin(angle_z), 0],
     [np.sin(angle_z), np.cos(angle_z), 0], [0, 0, 1]]), center=(0,
    0, 0))

    if isinstance(pcd, np.ndarray):
        # take only the first 3 columns and rows of the array
        R = pcd[:3, :3]
        R = np.dot(np.array([[1, 0, 0], [0, np.cos(angle_x), -np.sin
    (angle_x)], [0, np.sin(angle_x), np.cos(angle_x)]]), R)
        R = np.dot(np.array([[np.cos(angle_y), 0, np.sin(angle_y)],
    [0, 1, 0], [-np.sin(angle_y), 0, np.cos(angle_y)]]),R)
        R = np.dot(np.array([[np.cos(angle_z), -np.sin(angle_z), 0],
     [np.sin(angle_z), np.cos(angle_z), 0], [0, 0, 1]]),R)
        pcd[:3, :3] = R
    return pcd


def voxelDownsample(pcd, voxel_size = 0.01):
    """
    Downsamples the point cloud to reduce the number of points
    """
    pcd = pcd.voxel_down_sample(voxel_size)
```

```
    return pcd


def cluster_dbscan(pcd, eps, min_points):
    """
    Clusters the point cloud using DBSCAN and returns a list of the clustered objects as pointclouds
    """

    labels = np.array(pcd.cluster_dbscan(eps=eps, min_points=min_points, print_progress=True))
    max_label = labels.max()
    print(f"Point cloud has {max_label + 1} clusters")


    # if there are no clusters, clusters = False
    if max_label == -1:
        print("No clusters found")
        return []
    else:
        # return a list of the clustered objects as pointclouds
        return [pcd.select_by_index(np.where(labels == i)[0]) for i in range(max_label + 1)]




def crop(pcd, T_A0C, T_A1C, visualize = False):
    """
    parameters:
    pcd: point cloud
    T_A0C: transformation matrix from ArUco marker 0 to camera
    TA1C: transformation matrix from ArUco marker 1 to camera
    visualize: if True, the point cloud is visualized before and after cropping
    returns:
    pcd: cropped point cloud
    """
    if visualize:
        pcd_for_visualization = deepcopy(pcd)
        pcd_for_visualization = voxelDownsample(pcd_for_visualization, voxel_size = 20)
        # make pcd_for_visualization light green
        pcd_for_visualization.paint_uniform_color([0.5, 1, 0.5])



    # Transform the point cloud to the coordinate system of the ArUco marker 0 and crop the length of the incubator
    pcd.transform(np.linalg.inv(T_A0C))
    pcd = pcd.crop(o3d.geometry.AxisAlignedBoundingBox(min_bound=[-
```

```
    length_incubator_with_walls, -1200, -900], max_bound=[-
    length_marker, 900, 150]))
73      if visualize:
74          pcd_for_visualization.transform(np.linalg.inv(T_A0C))
75          o3d.visualization.draw_geometries([pcd_for_visualization,
    pcd, o3d.geometry.TriangleMesh.create_coordinate_frame(size=100,
    origin=[0, 0, 0])])
76
77
78      # Transform the point cloud back to the coordinate system of the
     ArUco marker 1 and crop the width of the incubator
79      pcd.transform(T_A0C)
80      pcd.transform(np.linalg.inv(T_A1C))
81      pcd = pcd.crop(o3d.geometry.AxisAlignedBoundingBox(min_bound
    =[-1200, -width_incubator_with_walls, -200], max_bound=[900, -
    length_marker, 150]))
82
83      if visualize:
84          pcd_for_visualization.transform(T_A0C)
85          pcd_for_visualization.transform(np.linalg.inv(T_A1C))
86          o3d.visualization.draw_geometries([pcd_for_visualization,
    pcd, o3d.geometry.TriangleMesh.create_coordinate_frame(size=100,
    origin=[0, 0, 0])])
87
88      # Find average z value of the pointcloud still in T_A1C
89      pcd_points = np.asarray(pcd.points)
90      z = pcd_points[:, 2]
91      # z_avg = np.average(z)
92
93      #Find 20th highest z value of the pointcloud
94      z = np.sort(z)
95      z_max = z[-20]
96
97
98      mask =  (pcd_points[:, 2] > z_max-30)
99
100     # Transform the point cloud back to the camera coordinate system
101     pcd.transform(T_A1C)
102     pcd.points = o3d.utility.Vector3dVector(pcd_points[mask])
103
104     if visualize:
105         pcd_for_visualization.transform(T_A1C)
106         o3d.visualization.draw_geometries([pcd_for_visualization,
    pcd, o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.1,
    origin=[0, 0, 0])])
107
108     return pcd
109
110 def findPoint(cluster, T_A1C):
111     """
112     Simple way of finding the top points of the point cloud by
```

```
    slicing the
113     point cloud along the z axis of A1 and finding the point in the
    slice
114     with the lowest y value
115     """
116 
117     # transform to A1
118     cluster1 = deepcopy(cluster)
119 
120     cluster1.transform(np.linalg.inv(T_A1C))
121 
122     pcd_points = np.asarray(cluster1.points)
123 
124     # find the highest z value
125     z = pcd_points[:, 2]
126     z_max = np.max(z)
127 
128     # make all points' z value equal to the highest value
129     pcd_points[:, 2] = z_max
130 
131     # find average point of pcd_points
132     tube_point = np.average(pcd_points, axis=0)
133 
134     # make y_value negative half of width of incubator
135     tube_point[1] = -(width_incubator_with_walls+length_marker)/2
136 
137     # translate a frame to the center point and create T_A1_center
138     T_A1tube = np.eye(4)
139     T_A1tube[:3,3] = tube_point
140 
141     # rotate to match tool frame
142     T_A1tube = rotate(T_A1tube, 0, np.pi, 0)
143 
144     # create T_CTube
145     tube_frame_in_camera = np.matmul(T_A1C, T_A1tube)
146 
147     return tube_frame_in_camera
```

Listing B.4: Helping functions for manipulating point cloud

B.4. Visualization of Object Detection

The green points are the excluded points of the point cloud. Their sparsity is to reduce computational load as it is only for visualization.

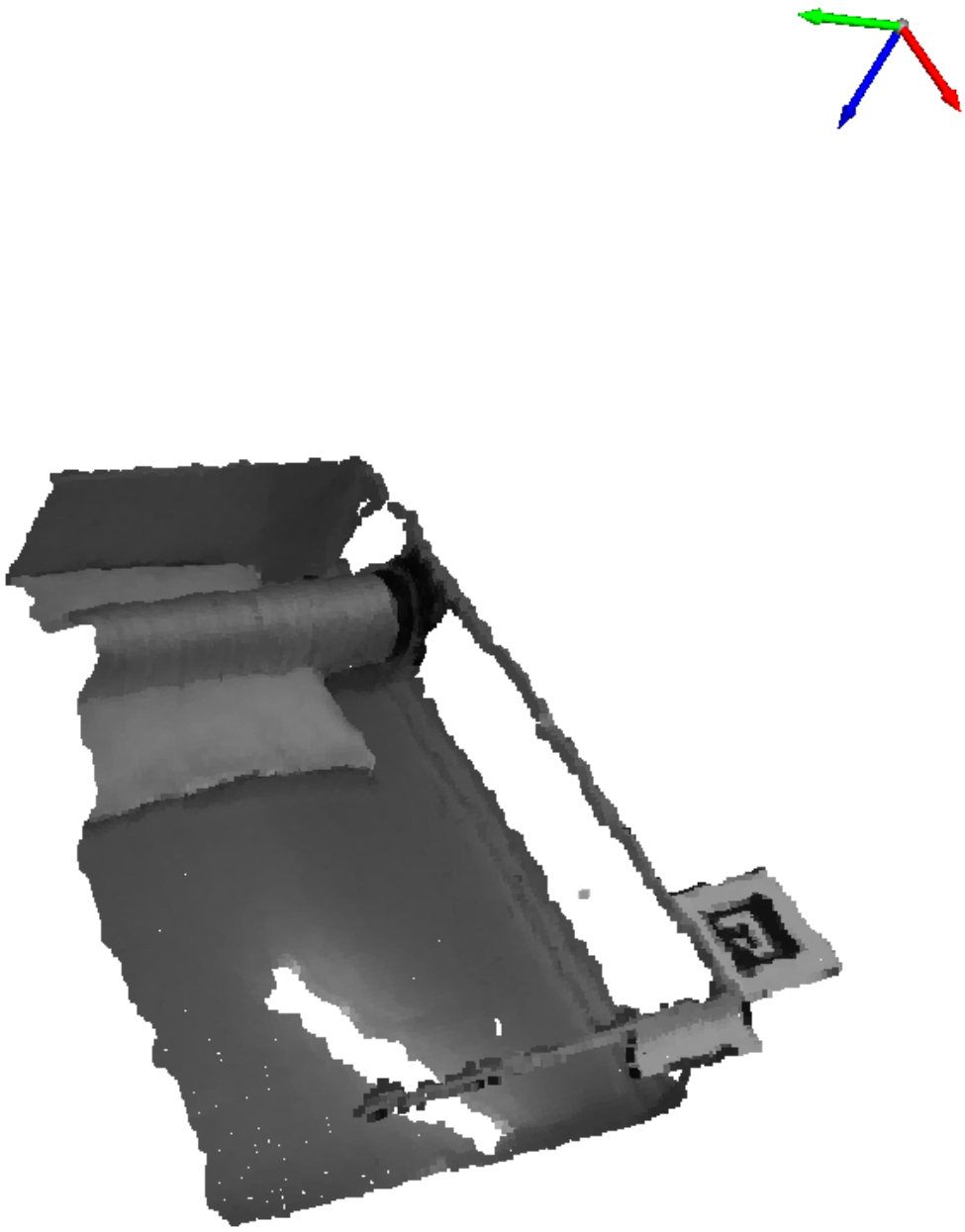


Figure B.1.: Original point cloud. Not all of the incubator is visible, but this is not an issue as stated in subsubsection 5.1.2. The coordinate system shows the camera frame. The white "hole" in the incubator reflects light from the ceiling on the reflective incubator surface.

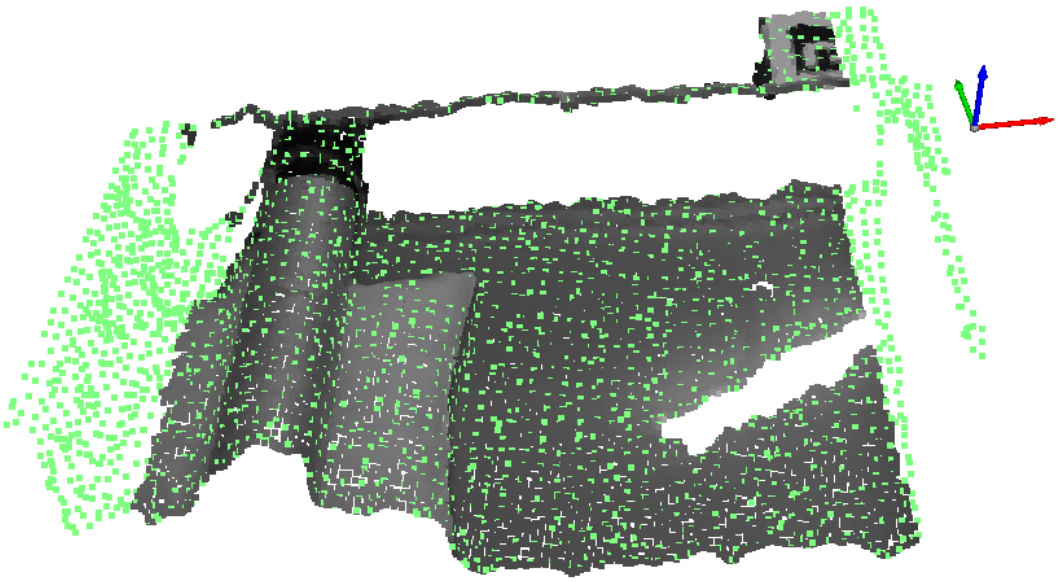


Figure B.2.: The point cloud is transformed to the coordinate system of the marker on the short side and cropped according to the parameters in Figure 5.5

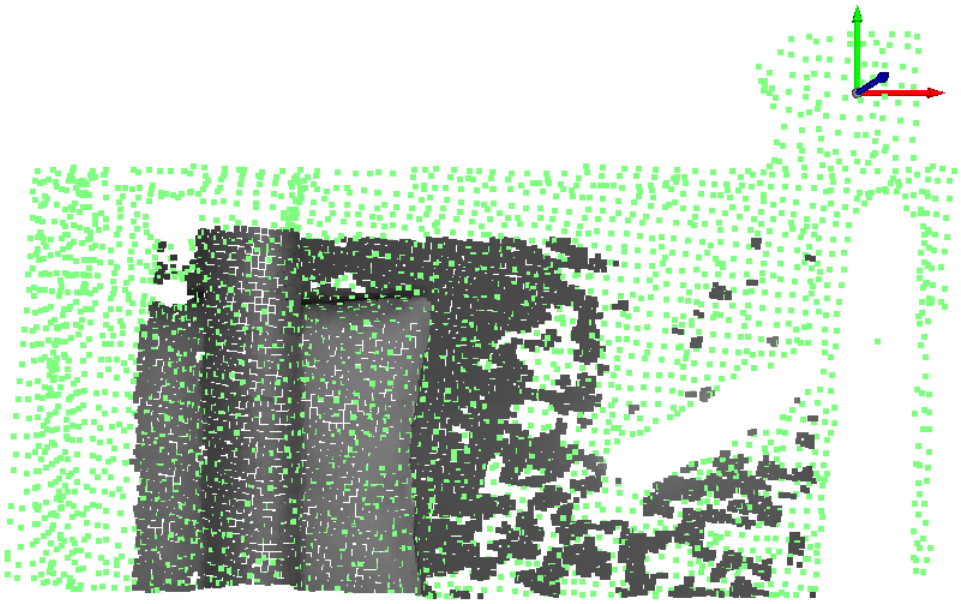


Figure B.3.: The point cloud is transformed to the coordinate system of the marker on the long side and cropped according to the parameters in Figure 5.5

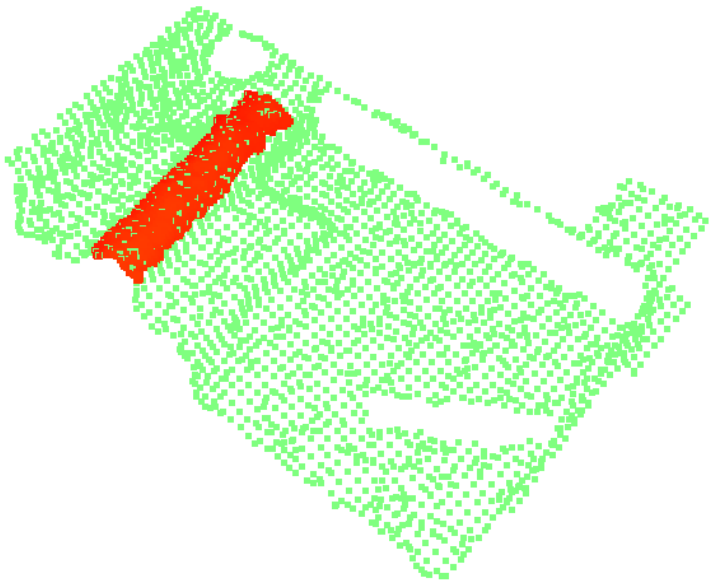


Figure B.4.: The point cloud is cropped on the vertical axis and the remaining point cloud is clustered. One cluster is found.

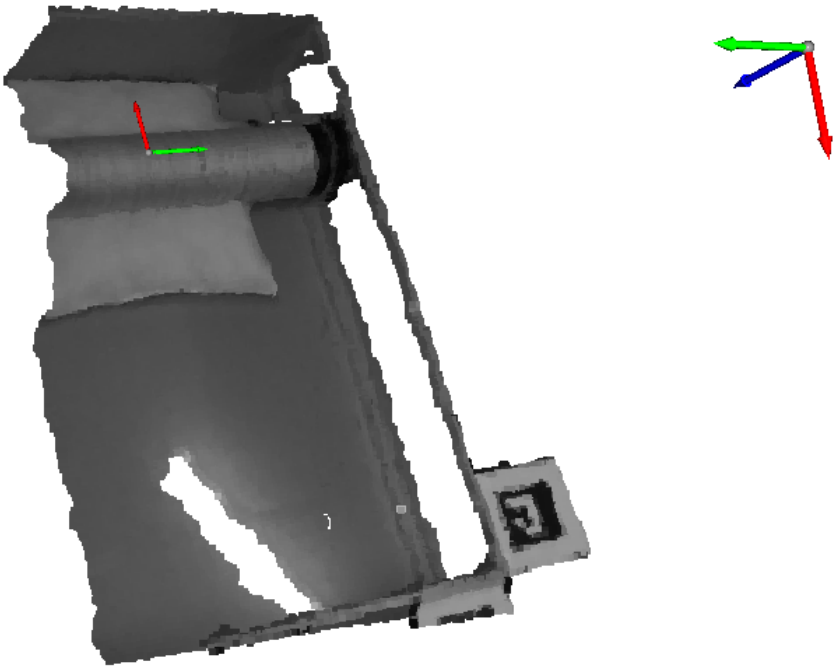
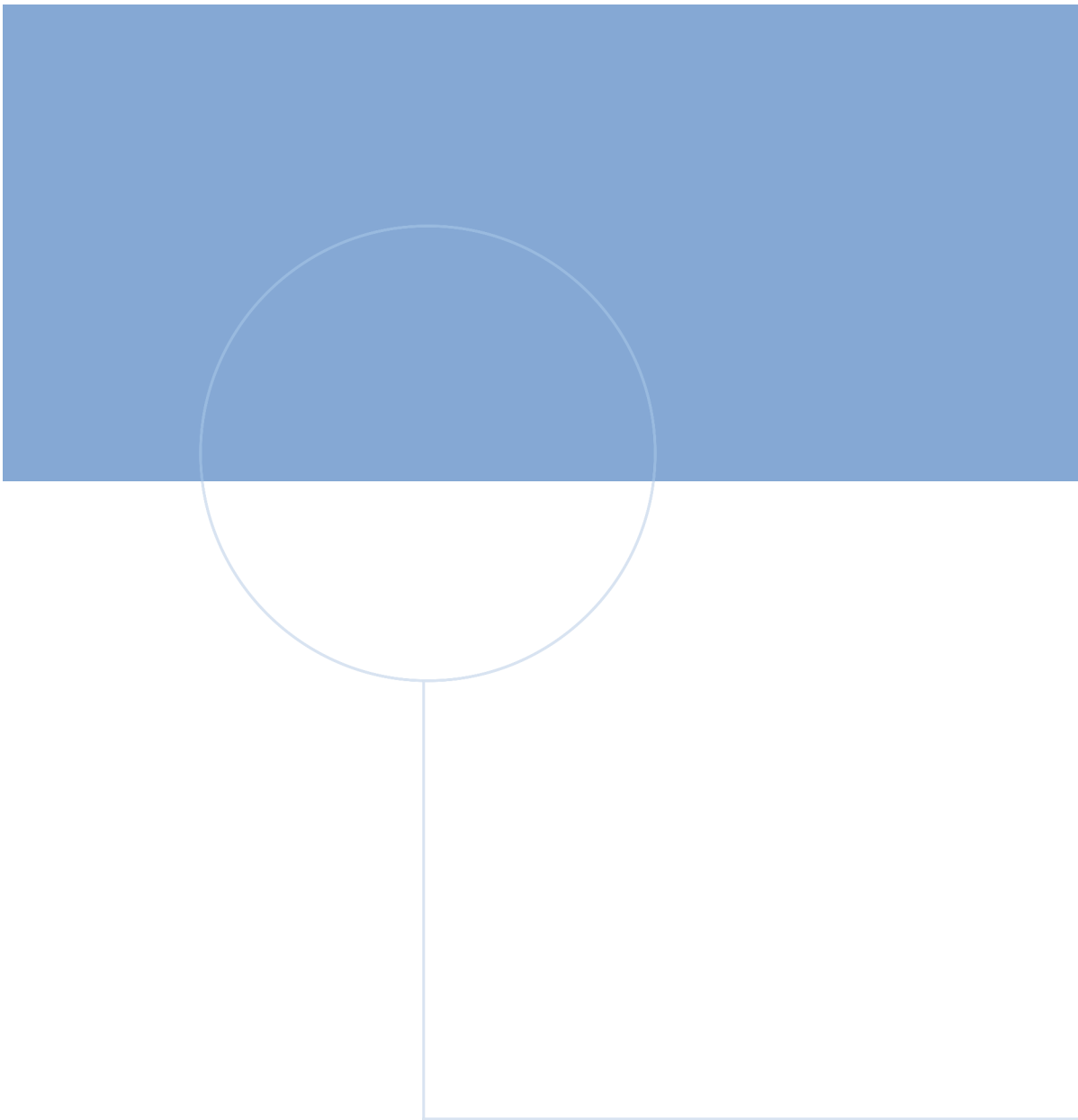


Figure B.5.: A tube frame of the same orientation is placed at the detected center of the tube frame and rotated orthogonally to match the gripper frame. The coordinate system above is the camera frame.



NTNU
Norwegian University of
Science and Technology

SINTEF