University of Southern Denmark

Bachelor of Engineering in Robot Systems - thesis

Calibration of robot with vision camera for dynamic positioning

Authors:

Emil Reventlov Husted *emhus17@student.sdu.dk*

Anton Landtved Christensen *antch18@student.sdu.dk*

Supervisor:

Danish Shaikh *danish@ummmi.sdu.dk*

Company contact:

Thomas Sølund *ts@spin-robotics.com*

Deadline: 03-01-2022

Characters 102.997 ~ 54 pages

Abstract

This project aims to develop an implementation of a simple solution to pose estimation, based on the current available theories. The idea is proposed by a company called Spin Robotics ApS, that develops screwdriver grippers. The goal is to precisely position the robot arm so that the gripper interaction can engage precisely accordingly to a world frame. The setup includes a gripper, imitating a screwdriver and a vision camera. The focus of the idea is to find a way to move the robot arm and to manipulate with the image, with known methods and reach the most precise solution.

The technical challenge is to design a calibration technique so that the camera is prepared to estimate points in the surrounding workspace. In order for the controller program to know the camera placement, it is an important task to estimate the transformation from the end effector frame to the camera frame which is possible to achieve with the eye-in-hand calibration. The control of the robot holds a possible challenge, when the robot needs to move independently without the interaction and overview of a developer.

The implemented approach is able to detect a checkerboard and use it to do pose estimation for a predefined work tasks. In order to get a precise detection of the checkerboard, the position of the object must be transformed from the camera frame to the robot frame with the use of the eye-in-hand calibration. The detection of the checkerboard and the robot control was performed with a combined C++ program that initialize the usage of OpenCV for the image manipulation and RTDE control interface for the robotic control. Throughout the project various experiments was executed for camera calibration, the precision of the positioning system and simulation to test the robustness of the system. Eye-in-hand calibration was implemented using multiple approaches from OpenCV, ViSP and SDU UR-RTDE.

In cases where the user needs to interact across the program, the interactions is done through the C++ controller program to choose between calibrations with a user defined number of images and pose estimate the location of the camera in regards to the end effector.

Preface

This bachelor project was conducted at the University of Southern Denmark and the Maersk Mc-Kinney Moller Institute. The project focus on research, implementation and experimentation of a dynamic positioning system to a robot arm for the company Spin Robotics ApS. Although the requirements for this project is based upon Spin Robotics ApS idea for creating such a system, known research on the subject and guidelines were provided by the university. The work of this project is divided between the authors who studies robot technology at the University of Southern Denmark.

This project has been a useful learning experience to thoroughly research the topic and have the opportunity to get an insight in a robotic company. It has indeed been a good way to see what efforts are required to examine the theorem entirely. A big thanks should be given to the company, for giving us the opportunity to work on this project and for supplying insights into their company and work processes.

During this project we had the opportunity to receive the necessary support and feedback for our work. Therefore we wish to express our gratitude towards our supervisor Danish Shaikh for his feedback, advice, and continued support. It has been a good experience, and the professionalism and enthusiasm for this project has been very helpful towards reaching our goals. We would also like to give a big thanks to our external company contact Thomas Sølund for guiding us through the work and answering essential questions regarding the project. Without Thomas and the collaboration with Spin Robotics ApS, we would never have been able to complete this project. At last, a special thanks to Saeed D. Farahani, head of section of SDU Industry 4.0 lab, for providing the opportunity to build the groundwork for this thesis. Without him we would never have had the necessary equipment for this thesis, nor the encouragement for creating this project.

The student	The student
Date: 03/01-2022	*Date: 03/01-2022*
Anton L. Christensen	Emil R. Husted
Anton Landtved Christensen	Emil Reventlov Husted

Table of Content

List of Figures

List of Tables

Acronyms

CDC calibration and detection checkerboard. vi, vii, IV, 1–4, 9, 11–14, 17, 18, 20, 24–29, 31–33, 35, 37, 45, 46, 49, 51–53

DH Denavit–Hartenberg. ix, 16, 17, 34

EE end effector. 2, 34, 35, 43, 44, 51

EiH eye-in-hand calibration method. vii, 2–7, 11, 17, 18, 20–22, 25, 26, 33, 34, 36, 43–45, 48–54

FK forward kinematics. 16, 17, 48, 53

FOV field of view. viii, 5, 29–32, 45, 49

OpenCV Open Source Computer Vision Library. v, vi, ix, 3, 10–13, 17, 19, 21, 36, 37, 42–45, 50, 51, 53

RTDE Real-Time Data Exchange. iv, vii, 2, 3, 23–25, 27, 28

SDU UR-RTDE SDU UR-RTDE Interface. 23, 28, 53

TCP tool center point. vii–ix, 6, 15, 18, 19, 21, 22, 24, 25, 29, 33–35, 43–45, 48, 49, 52

UR Universal Robots. 23, 24, 26, 28

URCap universal robots capitulation plugin. 25, 51, 53

ViSP Visual Servoing Platform. v, ix, 19, 36, 43–45, 50, 51, 53

Chapter 1

Introduction

This project is carried out in collaboration with a company that develops screwdriver grippers for assembly work of automated production lines, to comply with a wide range of products using screws. The interesting task is to investigate if it is possible to create a simple solution of a dynamic adjustment estimation system. If it is able to prove that there is a high precision, it would be possible to use it on a system with an automatic screwdriver gripper attached to a robot. To specify a problem formulation, it needs to be describe properly how the motivation for this project arised, and how the project scope combine all the aspects.

1.1 Motivation

The importance for this project is to expand the flexibility of a robot arm and improve the overall automation setup by adding additional features. The task is to develop a dynamic positioning system using a camera and calibration and detection checkerboard (CDC). Throughout this project the dynamic operation from the system, is defined by the change in the movement of the robot. The change should happen every time a new detection of the CDC is verified at the default position. The robot arm then have an option with pose estimation to define a new position according to the data received from the CDC. The feature is important to companies that needs to have a new production line automated, because it gives an improvement for the customer to change certain parts of the system. During the project different experiments was done to increase the precision of the system, including various analysis to improve performance. This project has been acquired with a UR5e robot arm [6] available for use. The robot was equipped with a screwdriver imitation tool and a vision camera [8] mounted near the flange of the robot.

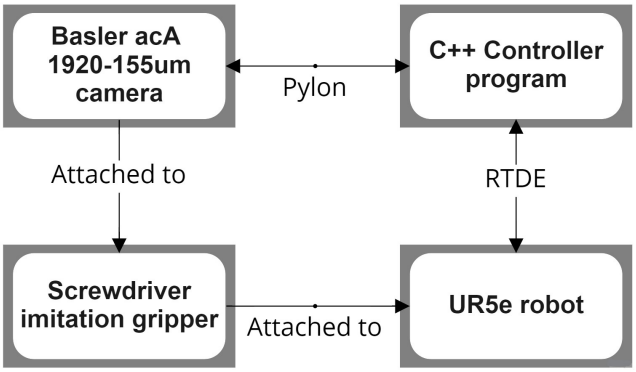


Figure 1.1: Overview of the project setup. Each box is representing a different section of the setup, while the arrows indicate the connections between the others.

A model of the project setup can be seen in figure 1.1, where it is shown how the different parts of the equipment was installed together. The gripper is attached to both the robot and camera, so the camera can have a fixed place near the end effector (EE). A cable is attached to a computer running Linux where the controller program is running, an Ethernet cable is going from the computer to the UR-controller connecting through the Real-Time Data Exchange (RTDE) interface [5]. To solve this task, it is necessary to define a problem statement to clarify the thesis requirements.

1.2 Problem statement

The goal is to develop a system that are able to detect a CDC fixed on a workspace and from there to plan a position using a camera and the eye-in-hand calibration method (EiH) which connects the camera to the robot through a homogeneous transformation matrix. The EiH method is defined by placing the camera on the robots EE, and calibrating the camera by different movements of the robot arm. The unknown homogeneous transformation matrix to where the camera is placed on the robot, is obtain with the use of multiple robot poses and the corresponding camera poses passed through the EiH calibration method. The following questions is considered from the above problem:

- How is it possible for the robot to perform dynamic positioning using a CDC?
- How precise can the system detect the CDC with vision sensor input?
- What possibilities is there to collect data from the robot arm and calibration process?

The project is part of a classical problem that estimate a position of a robot arm, accordingly to a world frame.

1.3 Project scope

To solve the problem, the project is to create the detection process where a camera is mounted on the robot arm which will act as the vision part, with Open Source Computer Vision Library (OpenCV) and the use of a CDC. The robot arm will be programmed, to moving within the proximity of the work board and check for the following job. The importance of the project is to create an simple solution for pose estimation and therefore the following requirements is listed:

- A vision camera will be used for detection done with the EiH method and the use of OpenCV for image manipulation.
- Robot control will be done using RTDE and external interface libraries.
- The system is able to engage and change according to a new dynamic environment.

The system will include a simple interface to the user which shows the process of the system. This project is also to verify the importance of different approaches of pose estimation and analyze what the benefits is of, either a large complex or a small frugal estimation system. The system consists of different scenarios, each with advantages and disadvantages, which should indicate the most suitable solution. During this project the workflow will be applied with a model-based approach which suggests that learning is assumed through planning with a pre-existing knowledge of the subject. The final project will examine the possibility to collect the relevant information from the camera and the robot arm throughout the process.

Figure 1.2: Equipment setup overview over the robot, the smaller checkerboard, screwdriver imitating gripper and the camera.

The robot arm is equipped what a pointing gripper where the camera can easily be attach. The robot arm is pointing the camera down on a 5×6-CDC, used in the calibration and is fixed on a target object. Later in the report, the importance of each part will be described. First research on the subjects of positioning systems and EiH calibration is done with multiple papers and approaches to investigate potential advantages and limitations to various methods. Also, to explore the existing state of the art solutions.

Chapter 2

State of the Art

Dynamic positioning of a robot arm can be solved in many ways and the problem is fairly old with theories going back decades. To the current day it is an important task across the industry to have a reliable pose estimation system, that automatic adjust the position of the robot in a safe way. The key problem is to do it fast and precise, in order to get the efficiency up in the more important parts of the automation production. Even though there are bigger tasks in automation, in regard to cost-effective and user friendly systems. Most companies still search for a unified easy system to attach to any robot arm, for during post estimation precisely. Looking at the wide varieties of closed systems, Omron tm landmark feature [21] comes close to an ideal system. Apart from the calibration procedure, the landmark system easily detects and gives the user a fixed or servo steering option in a user friendly programming environment. Not only Omron has this feature, but a Danish company called Enable Robots ApS [1] has also implemented a small, easy and precise system that does 6D calibration and pose estimation from the 5×6-checkerboard. The dynamic positioning is useful and easy for the users to accomplish more task to be automated, that most companies are implementing their own system and release it as a common feature. Even though the principle is old, we should include possible newer research for the last 5 years to see where the area has evolved on the subject.

A part of the precision of a dynamic positioning system depends on the physical connection between the camera and the robot arm described, with the transformation between the two. Many approaches have been investigated and developed form this problem e.g. the Kulpate et al. (2008) paper [11] which utilizes a single camera and a mirror to obtain 3D poses of the robot arms end effector for an EiH as shown in figure 2.1. The purpose of the mirror is to get the depth of field as in a stereo camera system and provides a wider field of view (FOV) while minimizing the cost of a stereo camera setup. The proposed visual servoing system include a 6-DOF robotic manipulator, but only the first 4-DOF in the robot is used for positioning. This paper uses an interesting method to achieve the

pose estimation through the depth of field with the mirror, but this setup estimates the EiH, where the camera is mounted stationary next to the robot and not fixed to the robot arm itself as the setup in this project.

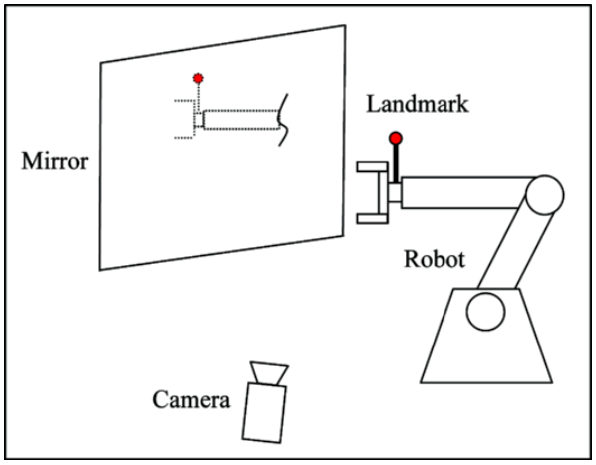
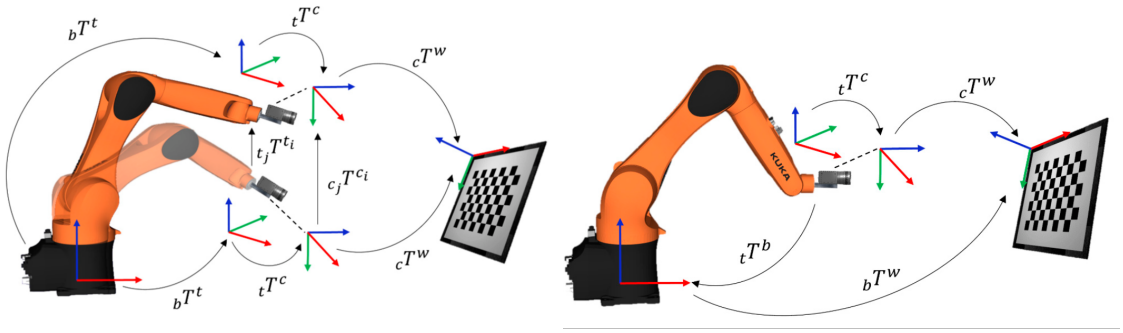


Figure 2.1: Proposed workspace consisting of a landmark mounted on the flange of a robot arm, a single camera, and a mirror placed to obtain the reflection of the robot from the view of the camera.

A more modern procedure is used in the Liu et al. (2019) paper [18] which proposes an image based visual servo approach with deep learning. The deep learning network used in the approach is convolutional neural network to obtain the learning capabilities. Training the network with images of the observed target and different poses of the manipulator, the network is able to estimate the nonlinear relationship between 2D image space and 3D Cartesian space. The image input is then transformed into desirable poses for guidance of the manipulator. The propose with deep learning uses a EiH setup similar to that of this project.

The Ali et al. (2019) paper [15] proposes the comparison between two novel geometrical interpretations of robot-world-hand-eye calibration as shown in figure 2.2. The main information used in the paper to obtain calibration is the tool center point (TCP) poses and the camera poses where the TCP pose can be extracted from the robot and the camera pose is gathered using for instance a calibration pattern which also can output the calibration parameters of the camera. The Ali et al. (2019) paper direct attention on solving the problems hand-eye and robot-world-hand-eye calibration formulated respectively as $AX = XB$ and $AX = ZB$. Where in the first formulation $AX = XB$, A is the transformation matrix from the robot base to the end effector ${}_bT^t$, B is the transformation matrix from calibration target to the camera ${}_cT^w$ and X is the transformation matrix from the end effector to the camera ${}_tT^c$. In the second formulation $AX = ZB$, A and B represents the same thing but X is the transformation matrix from the robot base to the calibration target ${}_bT^w$ and Z is the transformation matrix from end effector to the camera ${}_tT^c$.



(a) Eye-in-hand calibration setup (b) Robot-world-hand-eye calibration setup

Figure 2.2: Illustration regarding the setup of the two described problems of the geometrical transformation of a calibration.

The approach using a large number of different images with associated robot poses to solve the EiH will be examined further in this project. The Ali et al. (2019) paper mentions two different computation approaches where the first is described as the *separable*-solution where the problem formulation $AX = XB$ estimates the rotation and translation from the gripper to the camera separately in that specific order. The second is described as the *simultaneous*-solution where the rotation and translation are considered interdependent quantities and will not be computed separately but altogether.

The Shah (2013) paper [34] presents a different approach to solve the robot-world-hand-eye problem using an external sensor system as ground truth for the rest of the setup as shown in figure 2.3. The problem described throughout this paper is $AX = YB$ which differs from previous mentioned approaches to the robot-world-hand-eye problem there the focus is on the transformation from the robot base to the target this papers attention lies on the transformation from the ground truth sensor to the target.

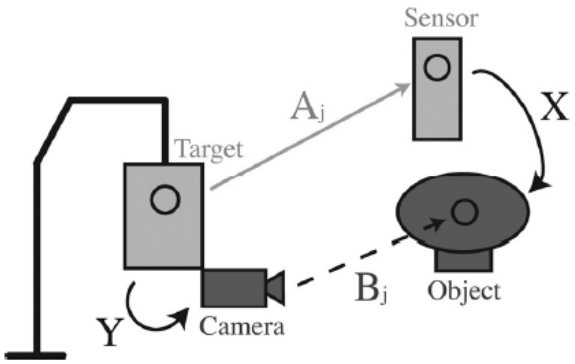


Figure 2.3: Experimental setup proposed to solve the $AX = YB$ problem. The setup consists the two systems, a computer vision- and a precise sensor-system considered ground truth.

In the experimental setup the camera system and the target for the sensor system are rigidly connected to a moving robot arm describing the fixed unknown transformation Y. The target is then being tracked by the system in different positions with the data from the stationary sensor represented by A while the camera is tracking the stationary object in

the positions with the data from the camera represented by B. Lastly the transformation between the sensor and object is described by X.
The Tabb et al. (2016) paper [39] proposes an implementation of the robot-world-hand-eye problem with three cameras mounted on a Denso VS-6577GM-B robot arm which later is converted to a larger Denso VM-60BIG robot arm with only one camera. The content of the paper involves three situations. The first part concerns a robot arm mounted with a camera for reconstruction of complicated objects when improving the outcome of the reconstruction using a low camera calibration error. The second part concerns the same setup with a robot mounted camera in laboratory and field conditions when non-experts are involved to perform calibration and data collection from a remote location and improve the robustness to non-ideal calibrations. The third part concerns robot-world-hand-eye calibration when multiple cameras are fixed to the same robot where one approach could be to calibrate one camera a then perform stereo camera calibration, but this would result in that an error from one calibration could be transferred to the next calibration. This leads to a desirable way to calibrate all components simultaneously. After the subjects of the state of the art systems that has been explained, the approaches and knowledge will lead to how the implementation of our solution lies in the research field. To further get the grasp of the project, the following chapters will explained the project implementation and what theories that are used, in regards to vision, transformation, and control.

Chapter 3

Theory and implementation

3.1 Vision

The vision input is created by the usages of a Basler camera of the model a2A1920-160umBAS [8] and a Basler camera lens of the model C125-1218-5M-P [9]. The use of this sensor system is to calibrate the camera in order to get the undistorted image stream, detect the reference CDC and pose estimation accordingly in a flexible workspace. To get access to the camera, it needs to be connected through an USB-port to the computer. The same computer is used to run the C++ `Controller` program made specifically for this task as seen in Appendix 2a. The end goal is to precisely transform the viewed point in the image frame to the corresponding point in reference to the base frame of the robot.

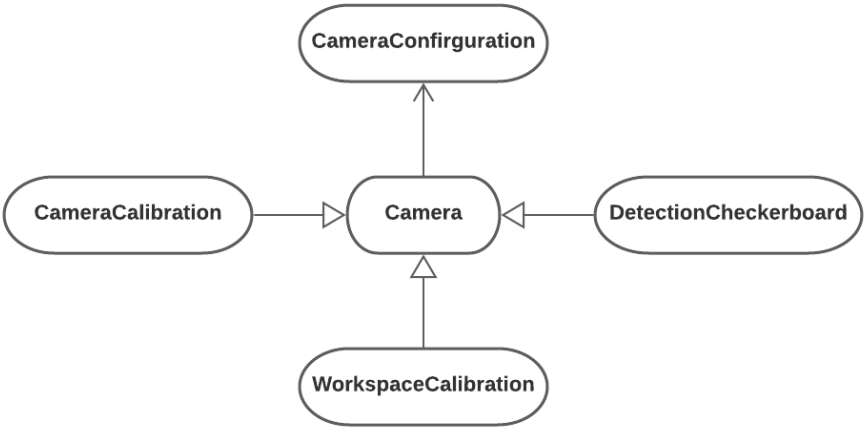


Figure 3.1: Design class diagram of the camera classes of the project. `CameraCalibration` is to process the calibration part, `Camera` are responsible for setup on the camera stream, `DetectionCheckerboard` does the detection from the camera scene, `WorkspaceCalibration` does most of the transformations of the system and `CameraConfirguration` is the settings definition of the camera.

The system uses libraries to perform as intended. To connect and use the Basler camera we implement a usage of libraries such as the Pylon library [10] and in every class to manipulate with the image frame we use OpenCV [27]. To convert between the Basler camera and OpenCV image objects `cv::Mat`, the Pylon wrapper library is necessary for the feasibility of implementing complex lines of code to the controller program and take images for the calibration. These libraries are exclusive to the vision implementation, and the performance of the system with the Basler camera.

3.1.1 Configuration

In order to use a camera, some initial work was done to configure the setup. A class called `cameraConfiguration` did this task, by creating an object from the class constructor. This is done so we have a suitable object, that defines the parameters for the camera. The parameters are unique to the camera brand and model and consist of the resolution, sensor size and focal length viewed from the Basler documentation [8]. The exposure is set in the class as well and helps to tune the setup environment according to light intensity.

Parameter	**Value**	**Unit**
Resolution	1920 × 1200	px
Sensor Size	6.6 × 4.1	mm
Focal length	12	mm

Table 3.1: Camera settings from the documentation of the Basler camera.

To use the camera to take pictures, the `camera` class was created and is working as the super class of the vision system. It is possible from the constructor to establish a connection to the Basler camera with help of the pylon wrapper library. In order for the sub class to start the stream, and calls the camera `initialize()` method. After the Pylon image object is created, it defines the value for the exposure time before calls the `action()` that executes the initialization code.

3.1.2 Calibration

A calibration is achieved by implementing the `cameraCalibration` class. The physical camera setup with a lens, can result in a distorted camera stream. In the scenario of a system that needs to transform and pose estimate anything according to the real world measuring, it is necessary to eliminate the distortion from the camera. The task is to calculate the camera distortion and provide a camera matrix and distortion coefficients for the provided lens. See the data output of the camera calibration in Appendix 1a.

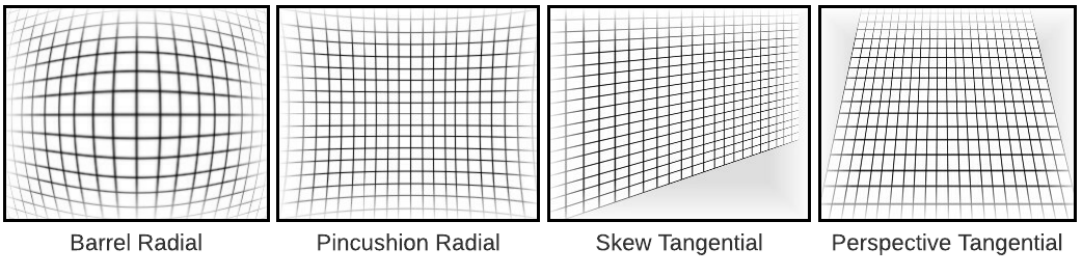


Figure 3.2: Examples of typical distortion experienced from a vision camera. It depends on the lens, the placement of the camera and the height of the image frame to target.

The radial distortion is an effect that occurs from the optical magnification of the lens. To detect anything on the workspace surface, the importance is to have a flat undistorted image. The result of these radial distortion makes the image sensor to create curves in the image. The tangential distortion is an effect of the camera lens that is not perfectly parallel to the plane of the image. The phenomenon occurs from different types and brands of camera lenses, but also in the event of a perspective view from different image poses from the camera on the robot arm in our setup. Due to the EiH calibration method, we have to calibrate the camera from the robot in different angles.
In order to rectify the image plane and create the linear map, OpenCV has a function that calibrates and provides the needed camera matrix and distortion coefficient of the camera. In our setup and technique we have a grid of intersection seen in a CDC. The widely used Zhang algorithm [40] is used where the grid is detected in different poses. This project implementation has included up to 65 picture poses, to include experiments useful to define the best number of images needed. The 65 picture poses can be seen in figure 3.3.

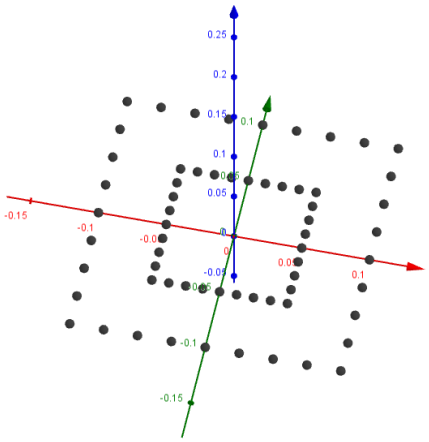


Figure 3.3: 65 different pictures poses used in the camera calibration process. All the poses is generated with respect to the default pose of the robot arm by 10-20 [cm] out from the sides.

The algorithm looks for the inner corners of the grid at a sub-pixel stage by using interpolation and solve a sequence of multiple radial and tangential equations. The calibration

technique is inexpensive and has few requirements for the calibration setup compared to more advanced approaches [13] with high precision of the geometric 3D space of the calibration object is used. Most high precision technique requires two or three calibration planes orthogonal to each other. To create the rectified maps the OpenCV provides functions that takes the calculated camera matrix and distortion coefficients and gives a reprojection error of the calibration. In the calibration class is a constructor that initialize the specific parameters, numbers of images, vertical intersections, horizontal intersections, and the size of the squares in the CDC. When the class initialize, several method is called, the video stream starts in a loop until the specified number of images is saved. The user has several options, which one of them is to grab individual images and that calls the `moveRobot` class and initialize `MoveArm::poseSwift()` that changes to the next position according certain parameters. The other option is to start the calibration option on auto mode and the loop will just keep going through every pose depending on chosen parameters and grab the pictures. Other features were also made to the experiments with added noise to the video stream, the option to quit an ongoing process or an option to regrab certain images if necessary. The user must verify that the number of images matched the files in the correct folder and the data is correctly saved.

3.1.3 Detection

For the detection of the CDC dynamic positioning in the workspace, the definition of calibration checkerboard and estimation checkerboard has to be defined. The dynamic position implies the need for detecting the CDC and finding its position with respect to the camera. In order to detect the CDC the function `findChessboardCorners()` [24] has been implemented. The method returns a boolean value of true if all corners was found and placed in correct order, in that case it outputs an array of the position of the corners. If the boolean value returns as false the function failed in either finding all corners or to reorder them correctly. This leads to the use of the function `cornerSubPix()` [23] to determine the position of the corners more accurately. The function uses the dot product methodology in order to increase the precision of the corners coordinate. This is done by iterating through different points in the neighborhood of the estimated corner from `findChessboardCorners()` and set the dot product of the gradient at that corner and the vector to the neighborhood point to 0. This generates a series of equations that form a linear system solved by inverting an auto-correlation matrix. This iteration is done until the termination criteria is reached specified by the user. The result of all corners found in the CDC is shown in figure 3.4.

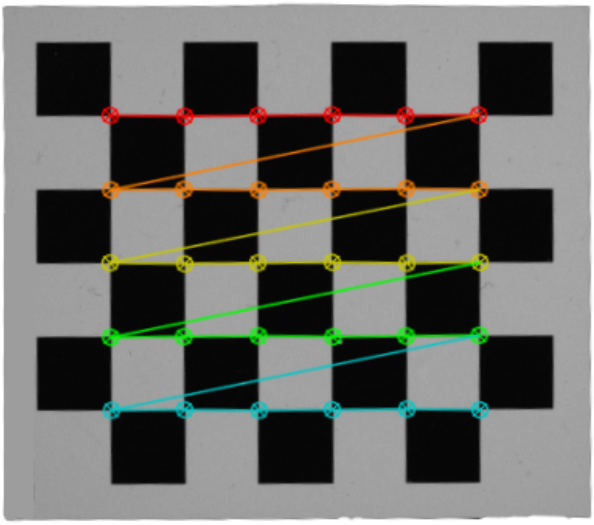
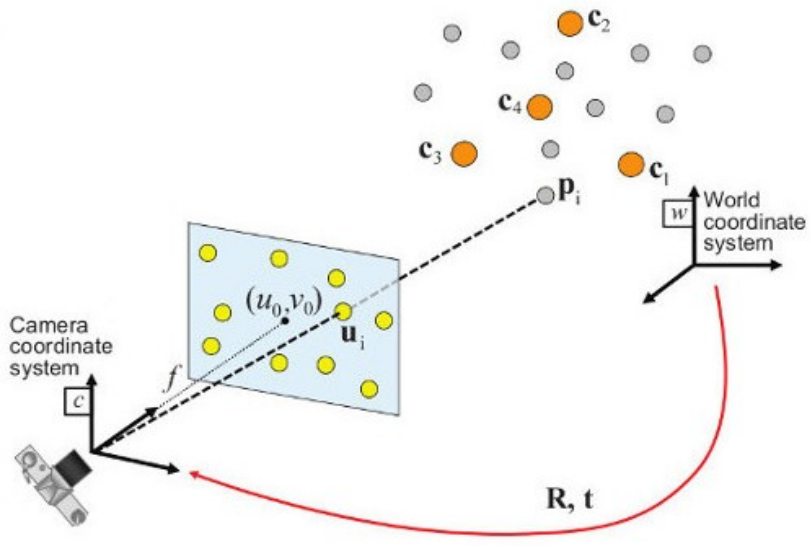


Figure 3.4: Detection of all inner corners on the CDC, using OpenCV method findChessboardCorners().

After detecting the CDC and finding the position of the corners in the CDC the function `solvePnP()` [26] is called. Using perspective projection and the intrinsic parameters of the camera, points from the world frame are projected into the camera frame. The projection lets the function determine the rotation and translation vectors which allows a 3D point expressed in the world frame to be transformed into the camera frame like shown in figure 3.5.



OpenCV

Figure 3.5: Simple pinholde model of projection to object point using the Perspective-n-Point computation.

The transformation ${}^{c}T_{w}$ from figure 3.7 in the following sections is composed of the rotation matrix and translation vector from `solvePnP()`. To achieve the pose estimation through projection, the function `projectPoints()` [25] is used in order to bring a given 3D point into the camera frame using the obtained rotation and translation from `solvePnP()`.

The transformation of the 3D point in the world frame to the 3D point p_c in camera frame is calculated as following:

$$p_c = \begin{bmatrix} {}^cR_w & {}^ct_w \\ 0_{1\times 3} & 1 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{3.1}$$

When the position of a point in the CDC is projected into the camera frame with respect to the optical center of the camera the CDC position can be described with respect to the robot base through further transformation as described throughout section 3.2. The vision implementation has focused on the camera calibration and detection, but the transformations data yield from the camera to scene, solves only part of the problem. The next chapter will describe how to use the detection data, so that it is possible to make a pose estimation from the robot arm manipulator through kinematic transformations.

3.2 Transformation

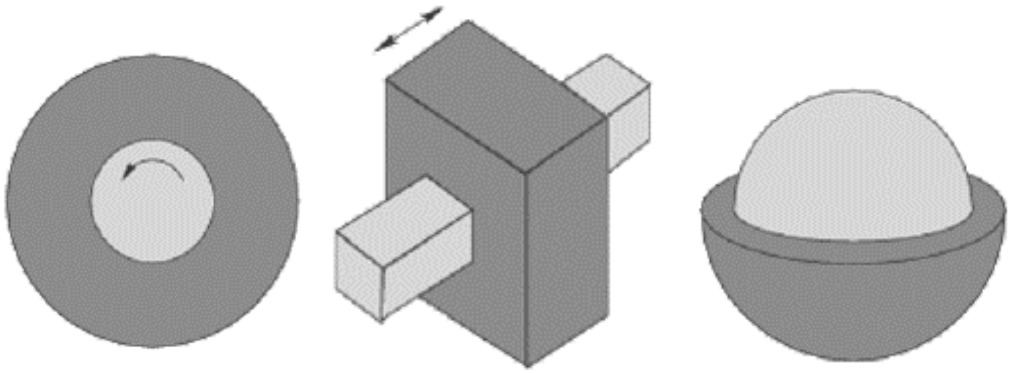
After the image information is extracted from the scene of the camera the next important step is to describe the system of bodies of the robot arm through kinematics and the individual relations between transformation matrices of the overall setup. All symbols used through the project can be seen in Appendix 7a.

3.2.1 Kinematics

Robot kinematics describes the relationship between the robot joints in configuration space and its tool center point (TCP) in cartesian space. The robot joints are often defined as a vector $[q]$ containing the value of each n-joints of the robot arm as shown in equation 3.2. The TCP pose is described as a homogeneous transformation matrix ${}^{B}_{T}T$ from the robot base to the tool where the matrix contains the 3×3 rotational matrix ${}^{B}_{T}R$ and the 3×1 translation vector ${}^{B}p_{T_{org}}$ also shown in equation 3.2

$$q = \begin{bmatrix} q_0 \\ \vdots \\ q_{n-1} \end{bmatrix}, \quad {}^{B}_{T}T = \begin{bmatrix} {}^{B}_{T}R & {}^{B}p_{T_{org}} \\ 0^{3\times1} & 1 \end{bmatrix} \tag{3.2}$$

The kinematics of the robot arm is described by a set of links often designed as rigid bodies and joints connecting them. Joints can constrain the links relative movements rotational or translational and can be implemented in different configurations depending on the requirements of the application. Three common types of joint configurations is shown in figure 3.6 with the revolute joint in figure 3.6a, the prismatic joint in figure 3.6b and the spherical joint in figure 3.6c.



(a) Revolute joint (b) Prismatic joint (c) Spherical joint

Figure 3.6: Three common types of joints use in different mechanical application.

The kinematic system can be divided into different subgroups regarding the combination of links and joints and their interconnection. Serial kinematics or chain kinematics describes a system where each link is connected by joints to form a single ordered chain.

This implies that each child link of a joint being the parent to the next without the system forming a closed loop which makes the number of joints important. Parallel kinematics describes a kinematic system where the links are allowed to form a closed loop. For this type of kinematics system joints that segregate or branches out without dividing the system into two separate halves exist [16].
The kinematic system of this project can be described with a chain of transformation as illustrated in figure 3.7 there ${}^{g}T_{b}$ is the serial kinematics of the robot arm consisting of the combined transformation for each link, ${}^{c}T_{g}$ is the transformation from the gripper of the robot arm to the camera fixed on the robot arm, ${}^{c}T_{w}$ is the transformation from the target to the camera, which was found through perspective projection as described in section 3.1.3.

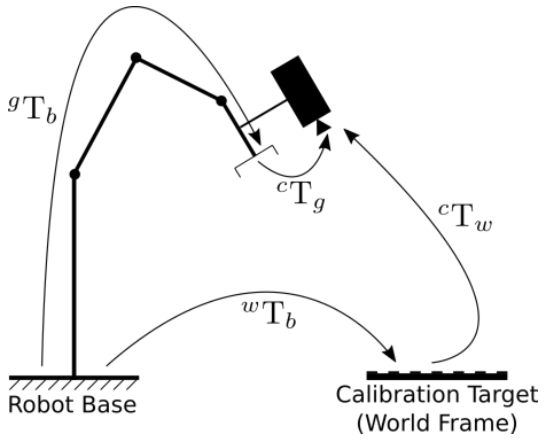


Figure 3.7: Visualization of this projects transformations, from the camera to the world frame.

The last transformation ${}^{w}T_{b}$ is the transformation from the base of the robot arm to the target which means the pose estimation is achieved. Throughout the rest of this section the transformation ${}^{g}T_{b}$ will be calculated and defined. The transformation ${}^{c}T_{g}$ will also be described in following sections. In order to achieve the transformation for the whole system the forward kinematics (FK) for the robot described as the transformation ${}^{g}T_{b}$ from base to gripper is calculated. First the transformation ${}^{f}T_{b}$ from the base to the flange of the robot arm is calculated using the Denavit–Hartenberg (DH) parameters for a UR5e robot [3] and is shown in table 3.2.

When calculating the FK of a robot using DH parameters θ_i describes the angle from $\widehat{X}_{i-1}$ to $\widehat{X}_i$ measured about $\widehat{Z}_i$, α_i describes the angle from $\widehat{Z}_i$ to $\widehat{Z}_{i+1}$ measured about $\widehat{X}_i$, a_i describes the distance from $\widehat{Z}_i$ to $\widehat{Z}_{i+1}$ measured along $\widehat{X}_i$ and d_i describes the distance from $\widehat{X}_{i-1}$ to $\widehat{X}_i$ measured along $\widehat{Z}_i$.
To compute the transformations the DH convention [38] is used with the definition of a homogeneous transformation matrix given as in equation 3.3.

Joint No.	θ_i	α_{i-1}	a_{i-1}	d_i
Joint 1	θ_1	$\frac{\pi}{2}$	0	0.1625
Joint 2	θ_2	0	-0.425	0
Joint 3	θ_3	0	-0.3922	0
Joint 4	θ_4	$\frac{\pi}{2}$	0	0.1333
Joint 5	θ_5	$-\frac{\pi}{2}$	0	0.0997
Joint 6	θ_6	0	0	0.0996

Table 3.2: Denavit–Hartenberg for the UR5e robot arm.

$$
{}^{i-1}\boldsymbol{T}_i = \left[\begin{array}{ccc|c} cos(\theta_i) & -sin(\theta_i)\cdot cos(\alpha_i) & sin(\theta_i)\cdot sin(\alpha_i) & a_i\cdot cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)\cdot cos(\alpha_i) & -cos(\theta_i)\cdot sin(\alpha_i) & a_i\cdot sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.3}
$$

See the calculated transformations for each link in Appendix 6a.
In order to obtain the whole robot transformation ${}^{g}T_b$ the transformation ${}^{f}T_b$ must be multiplied with a transformation ${}^{g}T_f$ from the robot flange to the gripper as following.

$$
{}^{g}T_b = {}^{f}T_b \cdot {}^{g}T_f \tag{3.4}
$$

The transformation ${}^{g}T_f$ depends on what gripper is mounted on the robot and as a consequence it is important to change this transformation if the gripper is replaced with another. In the system a method was programmed that takes the angel for each joint of the robot as argument and uses the predefined transformations to calculate the combined transformation for the whole robot which is later used to calculate the estimated pose of the CDC.

3.2.2 Eye-in-hand estimation

OpenCV

After the transformation ${}^{g}T_b$ is found through FK an estimation of the EiH transformation ${}^{c}T_g$ is desired. To find the transformation ${}^{c}T_g$ from gripper to camera as seen in figure 3.7 the OpenCV function `calibrateHandEye()` [22] was tested. The function can be called with methods that is implemented with respect to different papers regarding EiH calibration, but the main purpose of the methods is to estimate the rotation and translation for the transformation ${}^{c}T_g$ from gripper to camera. The methods associated with various papers is described in the following sections.
The EiH function requires the homogeneous transformation between the robot frame and

the gripper frame ${}^{g}T_{b}$ from several different poses and the corresponding estimated transformation between the target frame and the camera frame ${}^{c}T_{w}$. The EiH calibration will then return the homogeneous transformation:

$$p_g = \begin{bmatrix} {}^{g}R_c & {}^{g}t_c \\ 0_{1\times 3} & 1 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} X_g \\ Y_g \\ Z_g \\ 1 \end{bmatrix} \tag{3.5}$$

The problem is solved using the `calibrateHandEye()` function and is often described as $AX = XB$ where A and B are the pair of corresponding robot and camera poses and solving for X yields the transformation ${}^{c}T_{g}$.

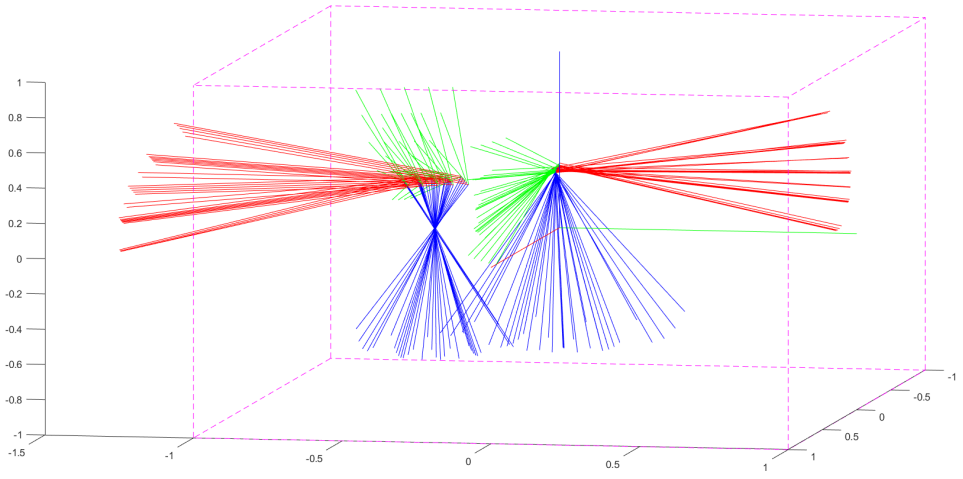


Figure 3.8: Robot (left) poses and camera (right) poses used from EiH calibration. The robot poses of the TCP is shown with respect to the robot base. The camera poses are shown with respect to the CDC.

The data set of 33 corresponding robot and camera poses can be seen in figure 3.8 and are visualized in 3D coordinates. The frames on the left of figure 3.8 shows the robot poses with respect to the robot base frame extracted from the program running the calibration. See the data output for the robot poses in Appendix 1b. The frames on the right of figure 3.8 shows the camera poses with respect to the CDC frame given from `solvePnP()` previously described in section 3.1.3. See the data output for the camera poses in Appendix 1d with the corresponding images of the CDC in Appendix 1c.

ViSP

The Visual Servoing Platform (ViSP) library was a solution we investigated close during the project. The library is useful in robotic application when it is appropriated to develop and test a project with vision capabilities. In robotic applications incorporating vision capabilities with the need for testing and further development the library is useful. It was time consuming to integrate a test environment inside the already prepared code. ViSP has like OpenCV capabilities of calibrating the camera in regard to a relative position. It usage the old TSAI algorithm [36] and is the same principle as the OpenCV function used with the TSAI-flag as parameter. Like the other methods mentioned, the class method `vpHandEyeCalibration::calibrate()` needs three parameters, a vector of 4×3 homogeneous transformation matrix representing the corresponding `solevePnP()` project points, an identical parameter representing the TCP position and a resulting homogeneous transformation matrix containing the calculated result. Experiments made for this technique were precise when providing the minimalism set of image calibration of 25 images.

3.2.3 Estimation

Pose estimation is a well-known technique in the field of computer vision mainly used for detection, tracking and augmentation of objects [35]. Many different solutions and methods have been developed to achieve pose estimation with each their different benefits and limitations. The methods continuously encounter optimization with the aim to increase speed and accuracy to the various applications using pose estimation. Pose estimation consists of various categories depending on the object or method used.
Human pose estimation is one of the categories where tracking key-points with focus on major links as the elbows, knees, and wrists. Due to the structure of the human body with arms and legs that can bend, causes the key-points to be in different positions relative to the rest of the body this also falls under a category of flexible objects as most animate things likewise most inanimate objects consists of a rigid structure. The estimation of inanimate objects with a rigid structure is known as rigid pose estimation which implies that the distance between key-points remain the same regardless of the object's orientation.
Other categories are 2D and 3D pose estimation where 2D focus on estimating the location of a key-point determined as X and Y coordinates in a frame relative to an image or video feed. Whether as 3D extends the estimation in the Z dimension when transforming an object in 2D to 3D allowing a spatial position of the desired object.
Lastly the category that consider the number of objects or humans during estimation which includes single and multi pose estimation with the single pose estimation approach

referring to the detection and tracking of one object or human and the multi pose estimation approach referring to the detection and tracking of multiple objects or humans [17]. As for this project the pose estimation falls under the 2D and 3D pose estimation where a point in the image frame is detected in 2D space relative to the optical center of the camera after which the transformation into 3D is performed. The implementation of pose estimation is when the transformations ${}^{g}T_b$ from equation 3.4, ${}^{c}T_g$ and ${}^{c}T_w$ is found or calculated and the pose estimation of the 3D point p_c in the camera frame from equation 3.1 can be achieved through the transformation ${}^{w}T_b$ describing the CDC position with respect to the robot base. The pose estimation of the CDC is then calculated:

$$ {}^{w}T_b = {}^{g}T_b \cdot {}^{c}T_g \cdot p_c \tag{3.6} $$

This sums up the transformation of the system, estimating the pose to the CDC with respect to the robot base.

3.2.4 Workspace Calibration

As mentioned before the camera is mounted on the flange of the robot in the system. This type of setup requires a calibration to find the transformation between the robot frame and the camera frame to bring detected object points from the camera frame into the robot base frame. The calibration is called hand eye calibration which consists of two types of setup as illustrated in figure 3.9 with each there own calibration method [41]. The two extensions of the calibration is called 'Eye to hand' and 'Eye in hand' - calibration. The eye-to-hand calibration is used when the camera is placed stationary next to the robot. The EiH calibration is used when the camera is fixed on the robot which is the case in this project.

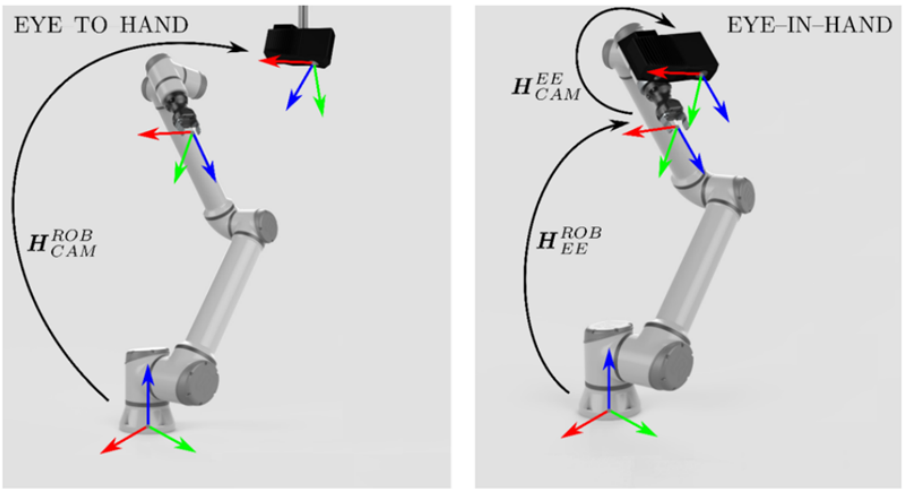


Figure 3.9: The two types of Hand eye calibration methods, eye-to-hand (left) and eye-in-hand (right).

To perform the EiH a large number of different robot arm poses and corresponding camera poses is needed. For each pose the robot TCP pose is stored and the camera pose is found using the rotation and translation vector from `solvePnP()` as mentioned in section 3.1.3. When a sufficient number of poses for the robot and camera is found the OpenCV function `calibrateHandEye()` [22] is used to compute the transformation from gripper to camera. The function has different methods to perform the calibration. The three methods estimating the rotation and translation with separate solutions are `CV::CALIB_HAND_EYE_TSAI`, `CV::CALIB_HAND_EYE_PARK` and `CV::CALIB_HAND_EYE_HORAUD` while the two other methods estimating the rotation and translation with simultaneous solutions are `CV::CALIB_HAND_EYE_ANDREFF` and `CV::CALIB_HAND_EYE_DANIILIDIS`.
The calibration method `CV::CALIB_HAND_EYE_TSAI` [36] derived from the study of Tsai from 1989 describing a new approach to use multiple poses of the robot and camera to compute an autonomous EiH calibration.
The calibration method `CV::CALIB_HAND_EYE_PARK` [28] derives from an approach using both closed-form exact and least squares solution. The closed-form exact approach determines a unique solution to $AX = XB$ with the assumption that no noise is present in the measured A and B. The closed-form least square approach determines a more practical solution to $AX = XB$ with noisy measurements.
The calibration method `CV::CALIB_HAND_EYE_HORAUD` [30] derives from an approach determining two potential formulations to the EiH calibration problem. The first formulation is the solution to the previously mention $AX = XB$ equation. The second formulation is the solution to the $MY = M'YB$ equation where M and M' are the 3×4 matrices related with two positions of the camera with reference to the calibration frame. The transformation matrix Y is equivalent to the X matrix describing the transformation from the gripper frame to the camera frame. B describes the transformation matrix related to two positions of the gripper with reference to the robot base frame. The approach states that the advantage of the second formulation is deeming the extrinsic and intrinsic parameters of the camera is unnecessary when computing the EiH calibration.
The calibration method `CV::CALIB_HAND_EYE_ANDREFF` [19] contrives from an approach deriving a new formulation to the EiH calibration problem and extend the new formulation to an on-line calibration. The approach states that classical formulations as the three above mentioned approaches are based on axis-angle representation of the rotation which is imprecise when the angle of rotation tend to zero in case of small adjustments between poses used for calibration. The new formulation is caused from the similarities between $AX = XB$ and the Sylvester equation $UV + VW = T$. The on-line calibration procedure incorporates the transformations between different robot poses with the transformations between different camera poses, but instead of using a calibration object to obtain the

camera pose, the camera observes an unknown rigid scene. Both the camera pose and the euclidean structure of the scene are then determined by motion processing of a structure. This approach causes the need of the calibration object with known geometry to be redundant.

The calibration method `CV::CALIB_HAND_EYE_DANIILIDIS` [12] derives from an approach estimating the EiH transformation in a more unified way unlike many other approaches by using dual quaternions. The reason the approach uses quaternions for interpolation of rigid body motions is the relationship between quaternions and screw theory that is an important tool in the mechanical engineering aspect of robots. The approach proves that if the camera and robot transformations are considered as screws the only relevance are the line coefficients of the screw axis regarding the EiH calibration. As mentioned previously in this section the `CV::CALIB_HAND_EYE_DANIILIDIS` method is a simultaneous estimation of the rotation and translation this is facilitated by the use of quaternions.

Now that the data for transformation of the camera to the scene and the kinematics of the robot is described, it is possible to go further to how the dynamic positioning of the robot arm can be made. The possibilities to create an estimated transformation matrix of the camera relative to the TCP, has now also been specified. The following part will explain the theories on how to control the robot arm, in order to calibrate the camera and to create the movement to a specific point in the workspace.

3.3 Robot

The application of the a UR5e robot arm [6] from Universal Robots (UR). The robot arm is used to perform and test the dynamic positioning and pose estimation from the vision system. Controlling the robot arm is done through an TCP/IP connection to the robot-controller box CB5. The communication is possible through UR script command documentation [4]. A library called SDU UR-RTDE Interface (SDU UR-RTDE) is used to have easy accessibility to the robot arm commands.

3.3.1 RTDE

When connected to the UR robot arm the RTDE interface [5] handles the synchronization between external applications and the UR-controller. This ensures any real-time properties of the UR-controller does not break. This functionality is also used when interacting with fieldbus drivers, utilizing the robot arms I/O ports or acquiring the robot arms status. When connecting with RTDE the connection is available on port number 30004. The RTDE interface is divided into three sub interfaces: RTDE Control interface, RTDE Receive interface and RTDE IO interface like shown on figure 3.10 [31].

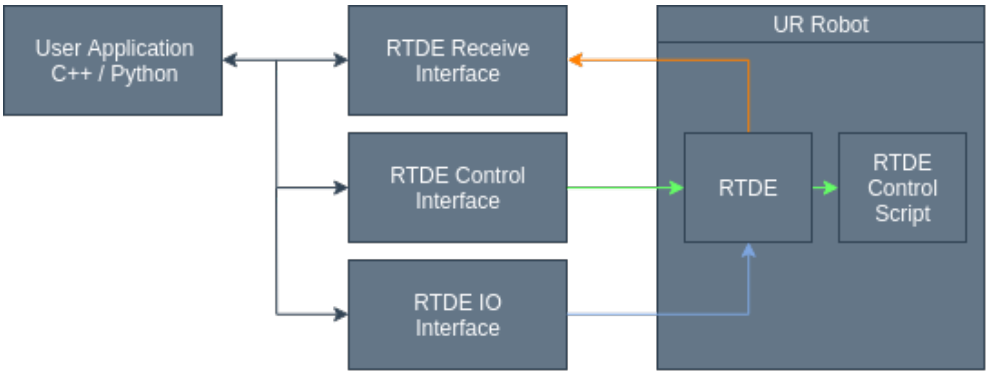


Figure 3.10: Real-Time Data Exchange interface flowchart where the RTDE Control interface uploads a control script automatically to the robot (green arrow), the RTDE Receive interface receives data from the robot (orange arrow) and the RTDE IO interface sets the digital and analog IO's to the robot (blue arrow).

When the RTDE is running the functionality is divided into two phases, a setup procedure and a synchronization loop. In the setup procedure the client is responsible for the variables to be synchronized upon connection with the RTDE interface and input and output registers from the client can be specified. To perform this the client has to send a setup list containing the named input and output fields for the actual data synchronization packages. The maximum limit of bytes representing the setup list is 2048 bytes. The format of a synchronization data package is described as a recipe and uses serialized binary data. The RTDE interface returns a list of the variable types or specifies a certain variable could not be found. If the input recipe was configured with success it will receive

a unique recipe id. When the setup procedure is finished the synchronization loop can start. When started the RTDE interface will send the requested data in the order the client specified. After the client receives the data the RTDE expects an updated input from the client upon change of values.
When the real-time synchronization is running the RTDE generates output messages of 125Hz. When the controller has higher priority than the RTDE it will skip a number of output packages if the controller lacks computational resources. The most recent data will always be sent as priority which means that skipped packages will not be sent afterwards. Regarding the input messages only one RTDE is able to change a certain variable at any time.
If the RTDE client is running on the UR-controller there is no network latency. This can however lead to the RTDE client and the UR-controller competing for the computational resources which is prevented when the RTDE client is running with standard operating system priority. Likewise, should other intensive processes like image processing be executed on an external computer.
Examples of used robot-controller outputs during this project is *actual_q* which return the robots joint positions and *actual_TCP_pose* which returns the cartesian coordinates of the tool represented as (x, y, z, rx, ru, rz) where (x, y, z) is the tool translation vector and (rx, ry, rz) is the tool rotation vector.

3.3.2 Control

The `moveRobot` class handles the interaction with the workspace and is used every time the calibration process needs a new image pose or if additionally testing was made. Once the teach pendant on the robot arm is set to remote control, the class is able to use the RTDE control interface. This utilizes the tools to use a movement in joint space to a default position. In the start the robot moves to a overview position in order to observe where the squared CDC is located. The method `getToCheckerboard()` travels to a fixed joint configuration by the `initialize()` method of `moveRobot` class and receive the TCP pose. To reliable distance the moveRobot class, the use of `rtde_controller::posetrans()`, method was used across most of the classes. The benefits are the ease of use and the return gives a reliable solution between to poses. At last, the movement to the CDC can be executed as a linear move in tool space, with the given change in parameters of the x-direction, y-direction, and our own implantation of the z-rotation. The robot is then directly over the CDC and that TCP pose is then saved as the temporarily base frame before the calibration is ready. To calibrate earlier chapters has mention the idea of the procedure. In relation to moveRobot, `poseSwift()` ensures the change in position according to chosen parameters, like the base frame, number of calibration images, progress,

velocity, and acceleration. Every time we utilize the method, the TCP-pose is saved in a document, so that later in the process it is possible to create an EiH estimation between the camera position and the robot flange. In order to use already prepared calibration points, the TCP pose is then saved so it matched the pose at the flange. In order to execute the calibration, every point above the CDC are transformed with a small variation in the position and orientation regarding the defined base frame.
After the calibration process, the detection of the CDC and the estimation of EiH, the next step is to move in relation to the found results. For that a method is created called `getToJob()` which keep track of the task at hand and gives a status of the process. The implantation uses the same movement principles used across the other methods and handles the coordination to the user specified task position. There is created room to three robot arm positions that needs to be specified at the beginning. Every time the robot arm returns to detect the checkerboard, it will look at the last place the CDC were located, the user can repeat the actions and the function will switch to the next task according to the latest detection.

3.3.3 URCaps

Instead of communicating over RTDE, is it possible to create script through the robot teach pendant. An universal robots capitulation plugin (URCap) plugin is a program especially made to add functionality to the user. To verify difference solutions, an initial idea was to include a user program via the robot teach pendant and send messages across our controller program in order to dynamical positioning the robot.

URCaps plugin

To solve the problem with user friendliness on the implemented system, a solution was to integrate a useful teach pendent program and with the right URCap functionality. Both in regard to the project simplicity and the request by of the company, we did examine the possibility to further work on the project in that direction. The features necessary was to verify the process in the static program, to connect through `IPv4` address and port to the certain main controller program. The relevant message that was appropriated in our setup was "Pose", "Close Server" and "Terminate server". For the pre-defined programs on the teach pendant, we created "System: Open", "System: Close", "Pose", "The program has been executed".

Figure 3.11: On the figure is shown what the feature implementing is capable of. The left side act as a client setup the controller program and on the right side is to specify a message.

The program is according to the URCap tutorial [7] created with respect to the described design procedure. The program is created in Java see Appendix 4a, and the activator class handles the bundles that makes the functions. According to the Universal Robots documentation, a URCap is created with the three bundle parts `ProgramNodeContribution`, `ProgramNodeService` and `ProgramNodeView`. Those bundles handle different parts of the functionality, and the names indicating what the part is specified in either contribution of a class, providing a service and defines a view layout for the features. To create the file needed, we combine this class and follow the steps created by Universal Robots that will let polyscope build the `.urcap` file. The main contribution for sending a command to the controller program, is to establish a connection were default IP address and port is defined in the variables class. The case with this implementation is that it is necessary to perform a client shutdown, due to a new connection cannot be made when a connection is continuing.

3.3.4 Movement

To be able to use the above implementation in a larger setup, it was intended to rely on predefined programs designed on the UR5e robots teach pendant. A program is included to present a solution for pose estimation of the robot to the CDC, in relation to static parameters. In the design it was a pre limitation that the EiH was defined statically in order to push the project to the last stage of the final system. After a detection on the CDC and definition of the rotation a feature frame was able to be extracted precisely to the reference point on the CDC. The program start by connecting to the client and sends a connection status message to the controller program. From that point the only other limitation is the feature frame has to be declared Beforehand. The steps of the tasks is to first move the robot to a predefined joint configuration, the calibration and detection before execution is responsible to deliver the robot in the already defined base position. The way the feature frame is represented is a change in direction and orientation, in regard to the detection of the CDC so the easy way was to use the Universal Robots predefined

function `pose_trans()` and then `moveL` could take the robot to the reference point. We experienced problems in the way the communication was to the teach pendant. The idea proposed was to declare an inline ur-script string to send to the controller, to define the feature frame as a global variable inside the running program. Due to the limitation in communication through the robot, when already establishing an RTDE connection and a socket connection gave us some errors. While a solution was integrated the compilation appeared to freeze the compiler, resulting in no respond to the running controller program. The decision later in the project was to work on the precision of the system, instead of creating a unified user friendly interface. The implementation are able to detect and calibrate from the CDC, create the communication and movement of the robot, transform the vital part of the system. Lastly, the system is missing the overall view to connect the different sections. The next is to address the combination of all the elements and performance, so that it is possible to presented an overview of all the main parts.

3.4 Controller

From the implementation, each section is individually explained but to see the project in the wider aspect this section focuses on the usability of the program. In addition, a flowchart is shown to get the idea of the flow of the project.

3.4.1 Program

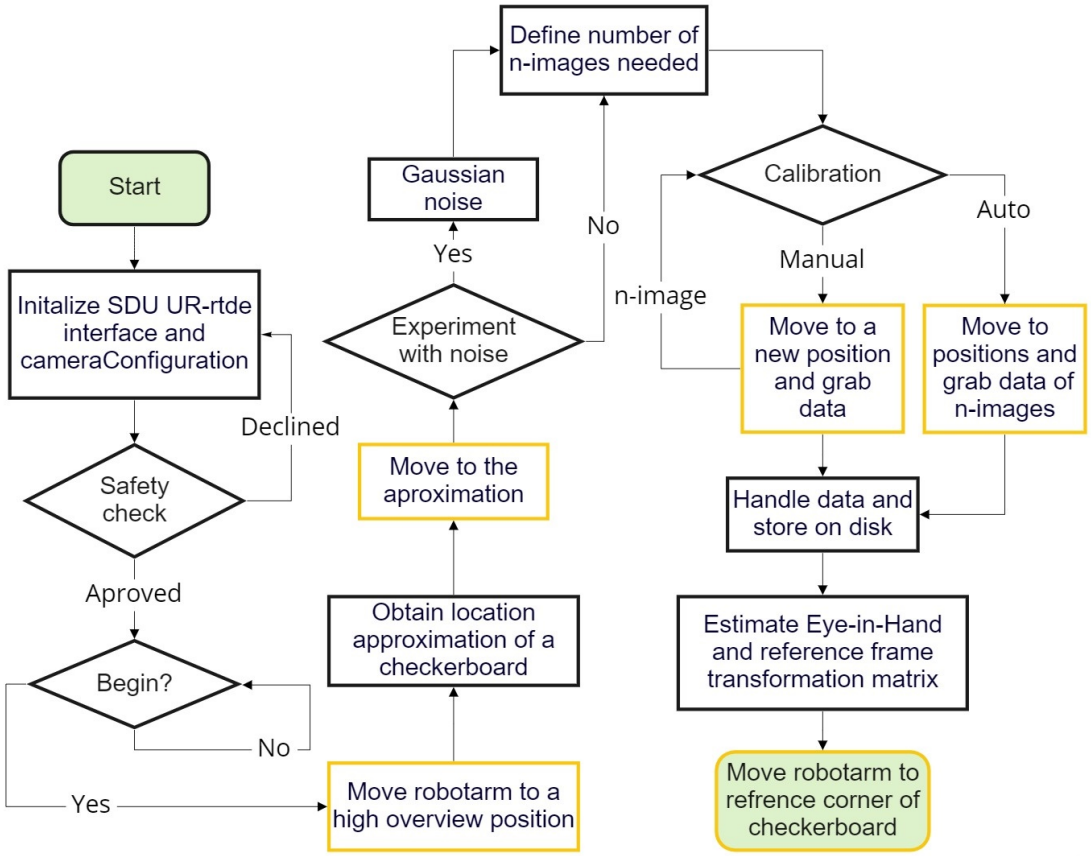


Figure 3.12: Flowchart showing one cycle of the main controller program of the implemented system.

The program is in the beginning trying to establish connection to the Universal Robots-controller by a RTDE connection with help of SDU UR-RTDE interface. It is at this state the objects from each class are constructed so that the configuration regarding the camera is specified in the cameraConfirguration. Safety features and user approval, was implemented as well in the end of this project. The idea for this project is separated by three core elements: Calibrate the camera, calibrate the workspace elements, and estimate the coordinate transformation. This project is not useful at this state to separate those section, so the overall cycle shown on figure 3.12 is the collaboration of each sequence. Despite the overall solution for this problem, it is possible to connect each individual segment for later experiment purpose. The first task is to find a possible CDC inside

the FOV and reach of the robot arm see Appendix 7b. The robot arm moves to a high predefined position to scan the area, it uses the same technique as for the detection of the inner corners of the CDC so the user needs to have in mind if any camera focus errors occurs. The reason for implementing the `findChessboardCorners()` is simple that the structure of the class `detectionCheckerboard` is implemented that way. The important task at this stage is to define a calibration movement route, and that can easily be solved if we have a known location over the CDC. The robot arm moves to the approximately location of the CDC marker, and tries to match the center of camera scene with the CDCs estimated center point. The user then go into the calibration sequence and is necessary to take some choices to complete the program. In regards to further code writing, this solution is in an experimental part, which include an experiment where noise can be added to the calibration process. The noise added to the images consists of numbers between 0-255 added to the gray scale value of each pixel. A function then distributes the numbers with a changeable standard deviation. There is not an option for the user to select number of images to calibrate with, as this is just defined before launching the controller program. The user does not need to choose the number of images that in the code can range from 1 to 65 images. The default is set to 25 images as it usually adds a good calibration. Some experiments were made of the inline calibration in figure 3.3 movement travel of this project and the results is defining the flow of the project. It will be discovered that the inline position has focused on keeping the CDC centered, which the manual mode can be used to overcome this problem. The user can choose either to take the auto mode which usage n-images at the predefined positions or choose to grab each image individually. This overcomes certain aspects as the user can then choose to run the calibration with inline positions or free drive the robot arm when during the calibration. If the user chooses to do the calibration manually the user needs to engage by pressing $'g'$ or $'G'$ to grab the data. The data is stored as `.txt` files in the appropriate folder on the drive, in different stages of the program cycle. The important files are calibration data, images, UR5 TCP pose, transformation data from the camera to the corner of the CDC and the joint configuration. Where the joint configuration or the joint value measured in [rad] is used to calculate the forward kinematics of the robot to obtain the transformation matrix of the robot poses. The last important thing is to validate the estimation. There are different approached implemented to try out what gives the best solution. Estimation is in the project involved in two things. The main purpose is to find a specific point on the CDC and precisely locate it. To do so the project has focused on estimating a transformation matrix to the cameras location in regard to a robot arms end effector. If the positioning is precisely know the robot arm will move down to the corner of the CDC, otherwise it will give an error message to the user.

Chapter 4

Analysis

To reflect on the project, the included chapter are to describe the choices that was made across the overall setup. The importance was to separate how and why from the above implementation of the system. And to include parts that may need to be experimented or concluded for future research.

4.1 System

The main parts of our setup are the camera, calibration and detection checkerboard (CDC) and the definition of the workspace area. The parts that are included are the changes we could make across the setup. Apart from the hardware setup the time has primarily been about experiments and discussions about the design of the implementation. From the beginning the parts of the project was to describe and preset a solution for pose estimation with a robot arm. Before it was possible to estimate, the calibration process became a big part eventually of this project. In the very beginning the focus was on creating the best calibration, and even though that other problems arised, the work focused still remained with improving the calibration.

4.1.1 Camera

The camera of the project is delivered by the company, and is typical used across different automation areas for the use of implementing vision. In the early state, we discussed the use of different kinds of cameras, and we was standing between a Raspberry Pi High Quality Camera [29] and the Basler camera [8]. The problems that occurred during this project with our choice of camera, was the lens working distance which defined the most possible working area on the table in general the FOV. Because of our limitation with the length of the robot arm and the fact it was necessary to get enough angle and variations of our calibration. Often the robot could not reach the symmetric poses we choose or the

CDC marker could not fit inside the FOV.
Another solution could be with the use of a 3D camera setup where the camera often consists of 2 or more lenses recording the view of multiple points. When using a 3D camera, the focus is to obtain the depth perception of a view be combining the multiple perspectives from the number of lenses contained in the device. In the case of a 3D camera with only one lens the device is able to move the lens to another position and from that obtain the same properties as a multi lens setup. Within the same category of 3D cameras also lies the 3D depth sensor that uses the technology known as time of flight. This technology detects the amount of time for a reflected beam of light to return to the sensor. From this measurement the device is able to calculate the distance to the object in view and the volume and size of the given object.
As for obtaining the depth of the image and calculate the distance to an object when using a 2D camera as done in this project. The used technique to measure the depth is the pinhole camera model describing the relationship between object in 3D space to the image plane. When implementing the pinhole camera model, the focus lies on calculating the focal length when used later for obtaining the distance to the object in view. To obtain the focal length of the camera the distance from the two diagonal intersections on the CDC in pixels is multiplied with the distance from the camera to the CDC measured in [mm] which is the divided with the distance between the two diagonal intersections measured in [mm].

4.1.2 Checkerboard

The CDC used in this project was a combination of the 5×6 checkerboard with a checker-size of 10 millimeter seen in figure 4.1 (right) and the 9×12 checkerboard with a checker-size of 15 millimeter seen in figure 4.1 (left).

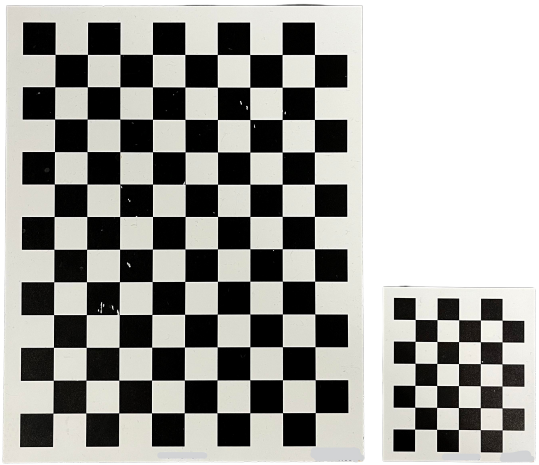


Figure 4.1: The checkerboard sizes used in this project, on the left is the 9×12 and on the right is the 5×6 shown.

The reason for the use of two different sized checkerboards was to see if the intrinsic parameters of the camera and the reprojection error had significant change when calibrating. Likewise, the big checkerboard forced the system to have a greater distance to the checkerboard during calibration which meant the robot had to operate in a different height due to the camera limitation of the FOV. The type of calibration board used in this project is a ordinary checkerboard with squares, that was chosen because of earlier experiences with camera calibration using a square checkerboard. However, a large range of different calibration targets exists e.g. checkerboards of different sized circles or checkerboards with squares covered with uniquely encoded fields see figure 4.2. Some extension research was made in regard to the different checkerboard, weather the project should use AprilTags [20], because of the extended advantages of AprilTags. Some other thoughts on the detection of the CDC was auto focus capabilities. Auto focus is not appropriate when dealing with pose estimation, because the image needs to be unaltered after the calibration and estimation to the reference. Ideally for our priority it would have been better if it was possible to have auto focus that was possible to control over the pylon library. If that was the case we would be able to detect the checkerboard at a higher location, move to just above the checkerboard and auto focus and than fixture the focus to define that as the default location.

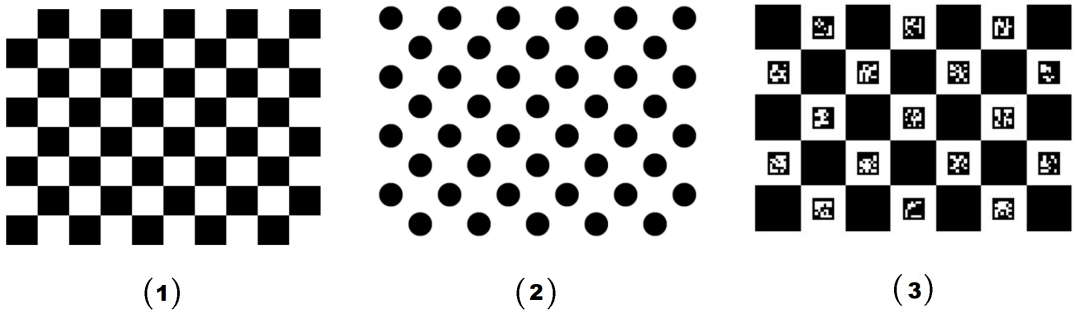


Figure 4.2: Three different calibration board types represented by squares (1), circles (2) and squares with unique encoding (3).

For the implementation of the camera calibration using the 5×6 checkerboard it was chosen that the checkerboard should have a static position during the calibration process. This way a default position of the robot where the checkerboard is in the center of the cameras FOV only needs to be programmed. The way the idea was implemented, the robot should move high to see a bigger picture, use a pose estimation technique from a undistorted image stream due to low distortion error, to be able to center the checkerboard at a fixed height of 0.45 meters. From there we could register that as the default position across the hole system and start the calibration so the camera was centering the CDC. The number of poses the robot arm should move to, corresponds to the number of images used in the calibration. From the default position the robot would move to the predefined

positions calculated with respect to the default position. However later in the project, errors occurred when using this calibration process due to the fact that in every image used for the calibration the checkerboard would always be in the center of the frame making the calibration less accurate when the checkerboard is not positioned in different areas of the plane.
A proposal from the company was to use an appropriated gripper like the pointing tool with a easy mount bracket for the camera. The gripper serves the purpose of interaction fairly easy with the workspace, for the task a piece of board were chosen, where certain circular forms are drawn. The meaning with the task, after the pose estimation is for the TCP to point the circles one by one. This kept the project in a simple manner, and therefore easy to focus on other important task, like the calibration and estimation of the pose as well as the EiH estimation. Other thoughts were discussed if we should use a two-finger classical gripper, to enhance the task at hand. Eventually the focus was narrow down to increase the precision instead of adding additional equipment.

4.1.3 Alternative approach

The alternative 6D pose estimation, could have been solved by exploring the AprilTag. If the small error of distortion by the lens is accepted, the proposed project could have leaned over to the choice of design of the detection marker that consist of an aprilTag format.

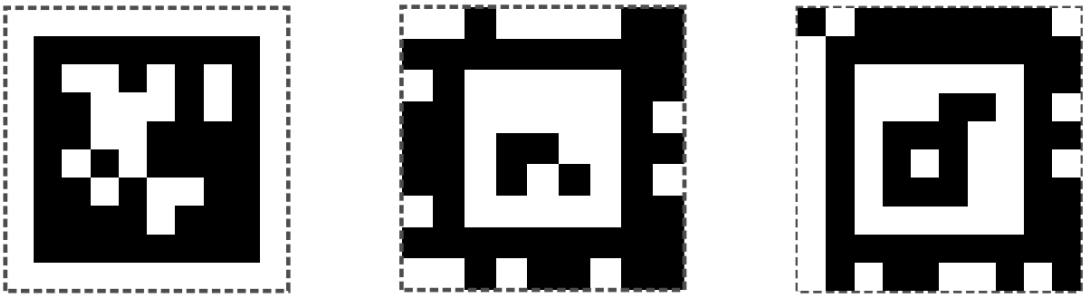


Figure 4.3: Three different AprilTag [20] style types represented by the default 2D bar code layout.

A proposal came for using AprilTags as they feature good capabilities of geometry estimation from a far distance. The first idea of the project handles the calibration of the camera, and therefore the use of fiducial marker e.g. AprilTags would not be an advantages when calibrating with our technique, so the calibration in this project would be redundant. Some later thoughts was that the CDC marker could just play a smaller role because the used calibration in real world scenarios, only are relevant when calibrating the camera scene and displacement from the robot. The research on fiducial markers are growing, and there is even now more data that suggest that the appropriated 6D

estimation could be better implemented by using these markers. A feature that could have been useful in the project are the data that can be stored inside the two dimensional AprilTag bar code which could have been an advantage in further implementation of the project. When we first discovered the possibilities of the tag, an idea was to incorporate useful information regarding the task of the specific AprilTag that were detected. This could bring a functionality for companies that usage detection-markers and bar codes in separation of each other. A useful example scenario is a company that most likely needs to interact with a rack of palettes in a production environment. On top of the each rack towers, is a checkerboard with software that does 6D pose estimation and a bar code for data storage of the specific rack type. The robot should first scan the checkerboard, read the bar code, and then travel to the estimated rack center. Lastly the robot arm should move down with a fixed distance and grab multiple palette. If our project have used AprilTag, the workcell could just have been integrated inside the customers production line and a technician could use the checkerboard to calibrate the camera if necessary. The idea proposal is then to sell extended AprilTag that could feature 6D pose estimation and included extended rack data, that can be placed on every production rack in the company. To unify the implementation, we discussed the example further. As the real companies that follow the above example, most likely also have Mobile industrial robots A/S [2] robots implemented in the compound of the company, to travel goods with different cart types. The detection of carts are done be scanning a QR Code the technician prints out in a specific size. A solution could be to combine useful features for the company, as the mobile robot could scan the apriltag and determine cart size, type, and 6D position from faraway.

4.1.4 Pose

In the final stages, the main concerns were to finish the EiH calculations and to prepare our technique of positions change accordingly. To prepare for the solutions, we could define the pose estimation steps and set a static frame for the camera relative placement by measuring with a ruler of precision about $\pm 0.03[mm]$. Even though there was struggle with the first part, the idea was to be prepared to finalize the overall project. The transformation up to the TCP was used e.g. the calculations of DH parameters and the detection of the work scene was made by loading the images that was saved in the earlier calibration.
To achieve the desired orientation of the EE when performing the pose estimation, the values of the rotation vector obtained from `solvePnP()` was manipulated. The change to the rotation vector was first, to convert the rotation vector to a 3×3 rotation matrix using the method `cv::Rodrigues()` followed by obtaining the transpose of that matrix using `cv::transpose()`. After returning the transposed matrix it was converted into

Euler angles before using the orientation for positioning the EE correctly with respect to the CDC. The reason that the matrix was converted into Euler angle was that when the program sends the function to move the robot arm with a given position and orientation the values of the orientation is with respect to the fixed TCP frame [32] uses the `poseTrans()` function to calculate the pose transformation between the two.

Because the Z-axis of EE was set to be in the direction towards the CDC and the Z-axis of the CDC was in the direction towards the end effector the orientation of the X- and Y-axis was manipulated to have the reverse sign of the values. When testing the manipulated orientation, the rotation around the Z-axis was a bit skewed. For this reason, the implementation of a method calculating the actual Z rotation of the CDC was manufactured. This implementation was built on simple trigonometry used when finding the angle of a vector formulated as.

$$\theta = tan^{-1}\left(\frac{y}{x}\right) \tag{4.1}$$

Where x and y are the coordinates of the vector and θ is the angle of the vector. When using the implementation, two corners on the edge of the CDC was used to obtain the vector from which the angle is calculated. The analysis has opened to further explanations on some parts of the system that, need further experiments.

Chapter 5

Experiments

To conclude on the project some experiments was made, to test how different parts of the system performed and if they work as expected. For a calibration to be precise, one must evaluate the parameters for the calibration to be able to correct the errors. The first experiment involves analyzing on the reprojection error of our calibration. To examine the calibration success rate, the reprojection error is not a complete verification, but it gives a little idea on how much the algorithm can correct the image. Several experiments are involved with this, as the importance of mapping coordinates between the frames can be complete off when the calibration is incomplete. In regard to the calibration an idea came to check the performance of a calibration if disturbance to the image plane was implemented. To test the overall system, the user can then add different amount of Gaussian noise before choosing to activate a calibration process. To sum up the overall performance, some initial experiment and analysis were necessary to determine the estimation of the unknown calculations of the precision. both estimation of the EiH with OpenCV and ViSP was needed to conclude on the project. To present the pose estimation performance, a static measured frame of the camera mount could in millimeters pinpoint our world frame.

5.1 Reprojection Error

From the OpenCV the `calibrateCamera()` can estimate a reprojection error, which is to define a calibrations performance. The result is specified as a double value, where the interest is to have a value inside the range of $[0; 0.5]$ which is evaluated to be a great calibration, and anything above 1 is considered bad. The estimation comes from the algorithm that build systems of equations to map between the image plane points and the projected points. The reprojection error is the sum of these equations of the observed points and the projected points. When calibrating the camera two tests have

been conducted. In the first test the number of images of the CDC used for the calibration was increased by 1 every time. This means that that the calibration first was done with one image then the calibration was done with two images etc. This iteration was done until the last calibration with 33 images. This test was done to see that effect the number of images had on the reprojection error obtained by the `calibrateCamera()` function from OpenCV. See data set in Appendix 5a. This will be referred to as *Reprojection error test 1.*

In the second test the number of used images for the calibration stayed the same during the test. All calibrations were done using 25 images and this was also done 33 times. This means that the calibration process was done 33 times with 25 new images every time. This test was done to see the variation of the reprojection error when the number of images used for the calibration was static. See data set in Appendix 5a. This will be referred to as *Reprojection error test 2.*

Now there will be made a statistic analysis of the data from the two tests starting with reprojection error test 1. All calculation can be seen in Appendix 5b together with a MATLAB file used for plots in Appendix 3a.

5.1.1 Statistical analysis of reprojection error test 1

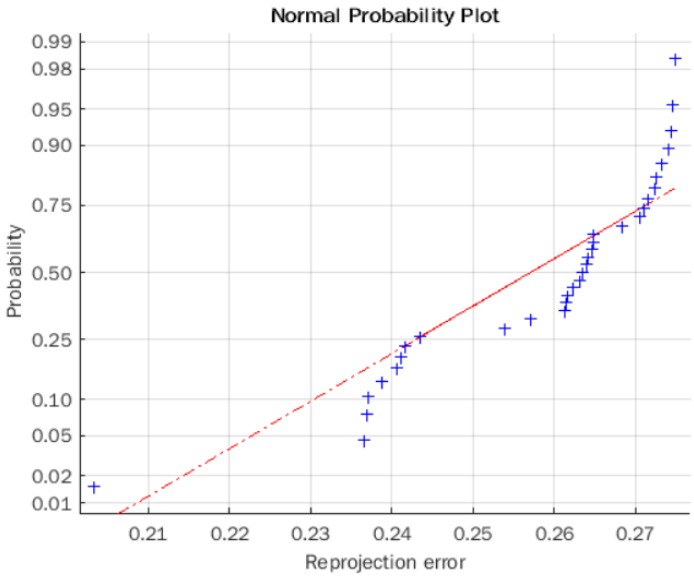


Figure 5.1: Normal Probability plot of the data set from test 1 with a increasing number of images used in the camera calibration process.

From figure 5.1 the data from reprojection error test 1 appears to not be normally distributed because the data set does not follow the red normal distribution line with several departures from the line throughout the data set and at each end of the distribution.

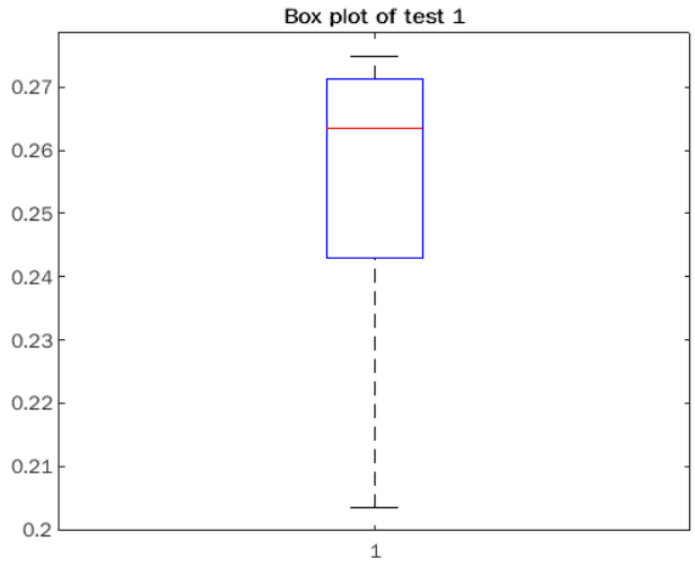


Figure 5.2: Boxplot of the data set from test 1 with the median as the red line and the two percentiles at 0,26 and 0,27.

On figure 5.2 a box plot of the data set from reprojection error test 1 is shown with the red line indicating the median between 0,26 and 0,27. The bottom and top of the box plot shows the 25th percentile at approximately 0,245 and 75th percentile at approximately 0,27. The difference between the minimum and the maximum value of the data set is 0,07155 as seen in Appendix 5a. The whiskers indicates the most extreme data point with respect to the main box. No outliers are indicated on this box plot.

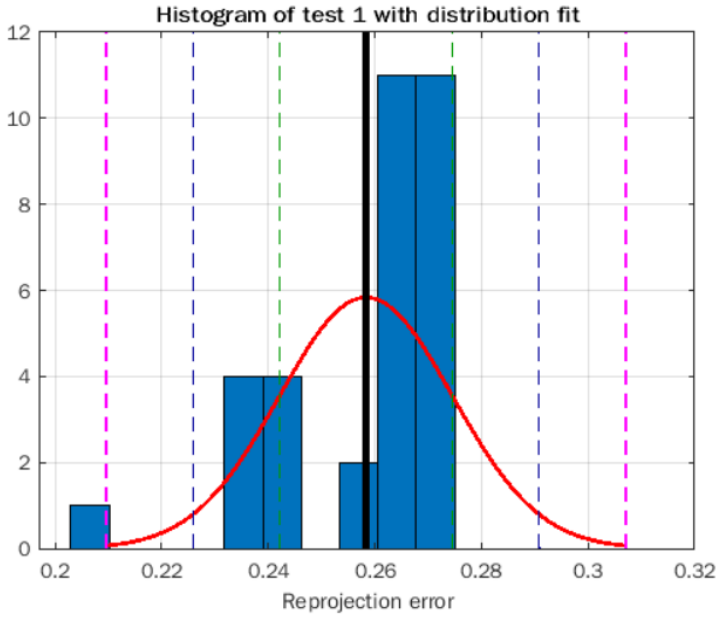


Figure 5.3: Normal distribution of the data set from test 1 shown with 10 bin histogram and the three levels of standard deviation in green, blue and pink.

Figure 5.3 shows the normal distribution of reprojection error test 1 with the black line as sample mean at 0,2584 and the red curve representing a normal distribution. The

green dotted lines represent the percentage level 68% from the empirical rule calculated in equation (6) in in Appendix 5b for the first standard deviation ($\bar{x} \pm 1\sigma$), the blue dotted lines represent the percentage level 95% from the empirical rule calculated in equation (7) in Appendix 5b for the second standard deviation ($\bar{x} \pm 2\sigma$) and the pink dotted lines represent the percentage level 99.7% from the empirical rule calculated in equation (8) in Appendix 5b for the third standard deviation ($\bar{x} \pm 3\sigma$). The empirical rule [14] also known as the *(68-95-99.7)*-rule is a statistical rule stating that for at normally distributed data set 68% of the data lies within one standard deviation, 95% lies within two standard deviations and 99.7% of the data lies within three standard deviations. Where the standard deviation is denoted as σ and the sample mean is denoted as $\bar{x}$. The empirical rule is often used if a prediction of a probable outcome from a data set is need. Furthermore, the empirical rule also indicates the normality of the distribution. This implies that the distribution may not be normal if a large number of data points lies outside the boundaries of the three standard deviation which could result in a skewed distribution.

The conclusions from the observations and the statistical analysis shown in figure 5.1, 5.2 and 5.3 was that the data set from reprojection error test 1 was not normal distributed, which means that the arrangement of most values from the data set doesn't cluster in the middle range of the distribution where the rest of the values taper off symmetrically toward the positive and negative extremes. The data set was also observed to have a left-skewed distribution or negatively skewed distribution as seen in figure 5.1 with a long tail rising above the red normal distribution line in the right side of the plot. This means that the sample mean will be to the left of most of the distribution for the data set as seen in figure 5.3 where the black line as the sample mean is to the left of the large blue bin.

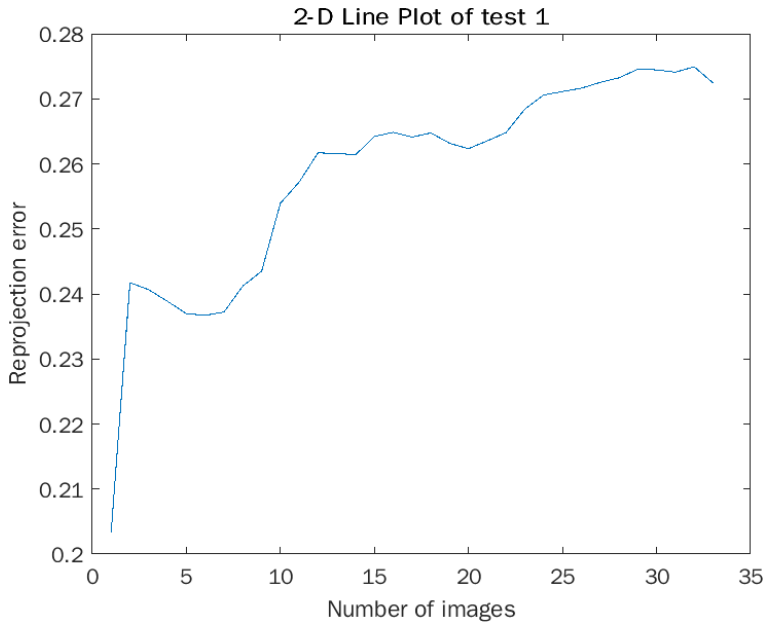


Figure 5.4: 2D plot of the data set from test 1

Furthermore, it can be observed that with an increasing number of images used for the calibration the reprojection error will stabilize around 0,274 after 29 images as shown in figure 5.4.

5.1.2 Statistical analysis of reprojection error test 2

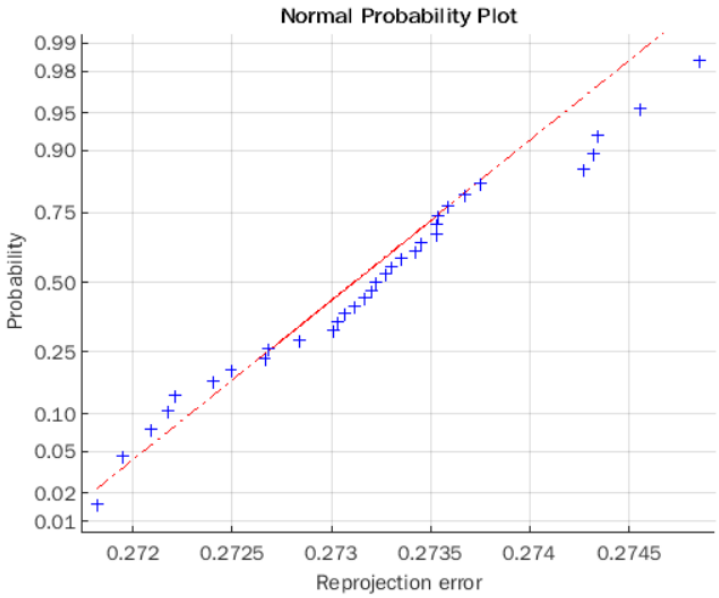


Figure 5.5: Normal Probability plot of data set from test 2 with the same number of images used in the calibration process.

From figure 5.5 the data from reprojection error test 2 appears to be normally distributed because the data set follows the red normal distribution line However, there are a few departures from the line at each end of the distribution.

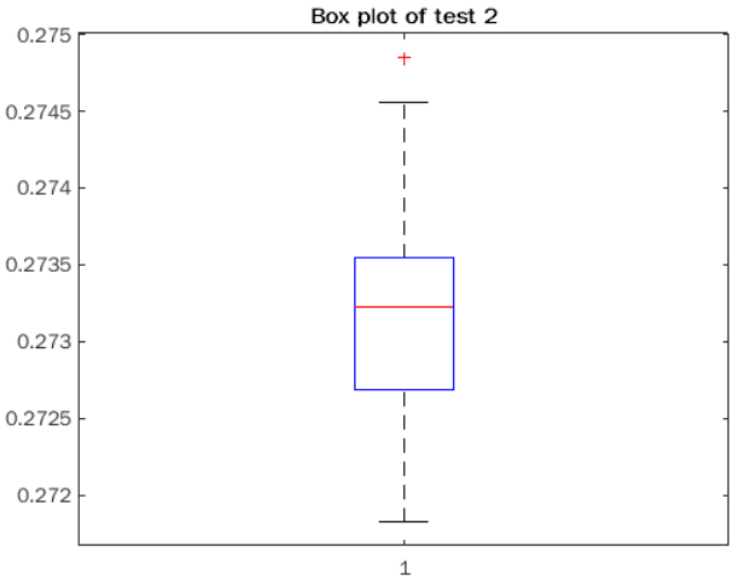


Figure 5.6: Boxplot of the data set from test 2 with the median as the red line and the two percentiles at 0,26 and 0,27.

On figure 5.6 a box plot of the data set from reprojection error test 2 is shown with the red line indicating the median between 0,273 and 0,2735. The bottom and top of the box plot shows the 25th percentile at approximately 0,2727 and 75th percentile at approximately 0,2735. The difference between the minimum and the maximum value of the data set is 0,003034 as seen in Appendix 5a. The whiskers indicate the most extreme data points with respect to the main box. One outlier at approximately 0,2748 is indicated with a red cross.

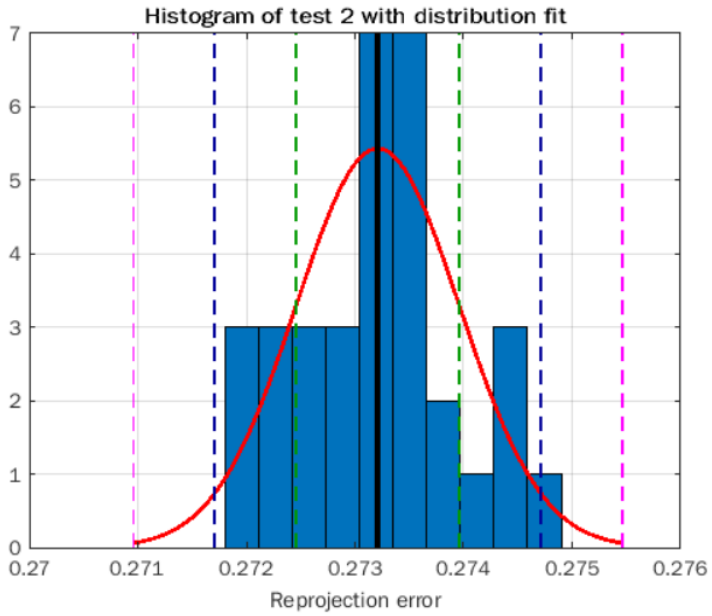


Figure 5.7: Normal distribution of the data set from test 2 shown with 10 bin histogram and the three levels of standard deviation in green, blue and pink.

Figure 5.7 shows the normal distribution of reprojection error test 2 with the black line as sample mean at 0,2732 and the red curve representing a normal distribution. The green dotted lines represent the percentage level 68% from the empirical rule calculated in equation (14) in Appendix 5b for the first standard deviation ($\bar{x} \pm 1\sigma$), the blue dotted lines represent the percentage level 95% from the empirical rule calculated in equation (15) in Appendix 5b for the second standard deviation ($\bar{x} \pm 2\sigma$) and the pink dotted lines represents the percentage level 99.7% from the empirical rule calculated in equation (16) in Appendix 5b for the third standard deviation ($\bar{x} \pm 3\sigma$). The conclusions from the observations and the statistical analysis shown in figure 5.5, 5.6 and 5.7 was that the data set from reprojection error test 1 was normal distributed, which means that most of the values are arranged so the data set cluster in the middle range of the distribution where the rest of the values taper off symmetrically toward the positive and negative extremes. As mentioned earlier and seen in figure 5.7 the difference between the minimum and maximum value is 0,003034 which shows that the variation of the reprojection error don't variate much when using the same number of images for the calibration process.

5.2 Gaussian noise

To test the robustness of the camera calibration process Gaussian noise was added to the images. When adding the Gaussian noise, the OpenCV function `randn(dst, mean, stddev)` was used where dst is the image on which the noise is added, mean is the mean value of the generated numbers and stddev is the standard deviation of the numbers. The noise is added to the grayscale value of each pixel using a number between 0-255.
During the test the mean was at 0 and the stddev value was increased until the calibration wouldn't compile anymore. The maximum value for stddev achieved in test with a successful calibration was $stddev = 65$. The reprojection error without Gaussian noise was 0,2728 which lies within the first standard deviation calculated in previous section 5.1 in equation (14) in Appendix 5b. The reprojection error with Gaussian noise with $stddev = 65$ was 0.3385 which lies beyond the standard deviation described in previous section in Reprojection error test 2.

Figure 5.8: On the left is calibration image without noise and on the right is calibration image with noise.

Above in figure 5.8 two images from the test with Gaussian noise can be seen. The experiment was made to test the robustness of the camera calibration where the images with noise was used. This iteration was done until the camera calibration process failed due to the amount of added noise. The added noise was to simulate real noise from the camera feed. The result of the Gaussian noise was useful to experience the robustness of the camera calibration and can be seen in the project code 2a with the intention for future work.

5.3 Precision

To test the precision of the system pose estimation, the EiH transformation must first be verified. This is done by comparing the result of a physical measuring of the change in translation (x, y, z) from EE to camera, to the result obtained through the OpenCV and ViSP function. First the distance in x, y and z was measured from the flange of the robot to the camera. When the distance x, y and z was measured from the EE to the camera. This measurements as seen in table 5.1 were later used to verify the results from EiH calibration from OpenCV and ViSP.

5.3.1 Static eye in hand

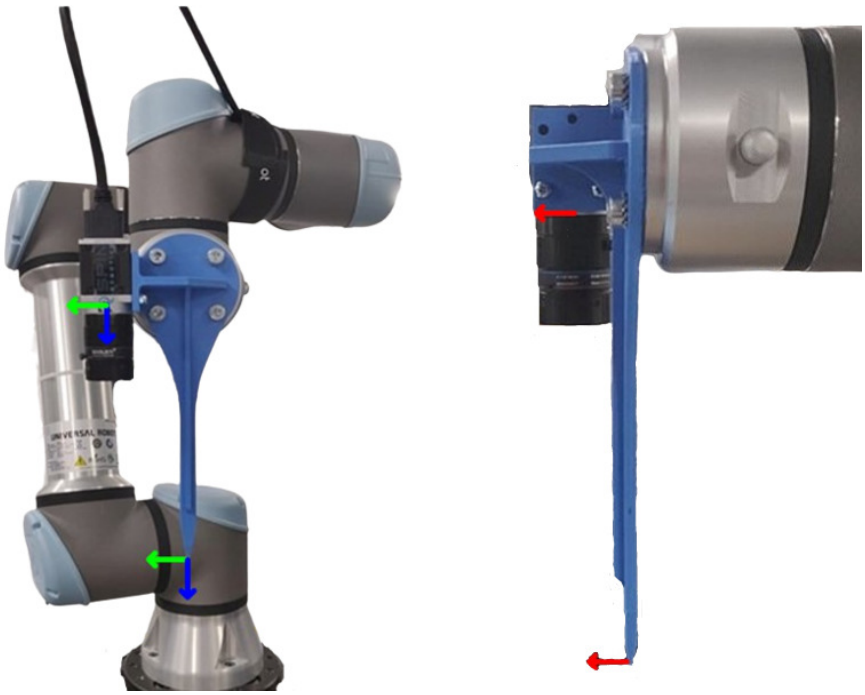


Figure 5.9: End effector from the front (left) and from the side (right) with drawn frame of the TCP and the camera frame.

In figure 5.9 the EE is shown with the coordinates represented as the X-axis with red, the Y-axis with green and the Z-axis with blue at the TCP. The translation is measured between both the flange of the robot and the camera and between the TCP and the camera. See the measurements in table 5.1.

	Translation			Rotation		
Measurements between frames	X [m]	Y [m]	Z [m]	rx	ry	rz
(1) Flange to camera approx.	-0.0225	0.0471	0.0118	0	0	0
(2) TCP to camera approx.	0.0225	0.0471	-0.1681	0	0	0

Table 5.1: Measurements for eye-in-hand with (1) as the transformation from flange to camera and (2) as the transformation from TCP to camera.

5.3.2 OpenCV eye in hand estimation

The EiH calibration estimation obtained from OpenCV is shown in table 5.2 where an estimation from each of the five calibration methods, as mentioned in section 3.2.4, is computed. The data used for the EiH calibration for the OpenCV function were 25 poses of the TCP and the corresponding 25 poses of the camera. The 25 TCP poses contain the position and orientation of the EE with respect to the robot base. The 25 camera poses contains the rotation and translation vector obtained from `solvePnP()`, as mentioned in section 3.1.3.
The estimation obtained from the OpenCV methods represents the transformation matrix from the EE to the camera which will be compared to the measured approximate transformation from TCP to camera seen in table 5.1 (2).

	Translation			Rotation		
Calibration method	X(m)	Y(m)	Z(m)	rx	ry	rz
Tsai	-0.022131	-0.06341	-0.01062	0	0	0
Park	0.0618783	-0.006878	0.0084970	0	0	0
Horaud	0.061888	-0.006847	0.0085011	0	0	0
Andreff	0.05027	0.01006	-0.13764	0	0	0
Daniilidis	0.0166	-0.01261	0.058060	0	0	0

Table 5.2: Estimated eye-in-hand transformation from the five OpenCV calibration methods.

5.3.3 ViSP eye in hand estimation

The EiH calibration estimation obtained from ViSP is shown in table 5.3. The data set used for the EiH calibration for the ViSP function were 25 poses of the robot flange and the corresponding 25 poses of the camera. The 25 robot flange poses contains the position and orientation of the robot flange with respect to the robot base. The 25 camera poses contain the rotation and translation vector obtained from `solvePnP()`.
The estimation obtained from the ViSP function represents the transformation from the robot flange to the camera which will be compared to the measured approximate transformation from flange to camera seen in table 5.1 (1).

	Translation			Rotation		
Transformation between frames	X (m)	Y (m)	Z (m)	rx	ry	rz
Flange to camera transformation	-0.0477	0.0627	0.02207	0	0	0

Table 5.3: Estimated eye-in-hand transformation from ViSP.

Verification of eye-in-hand estimation

To determine the accuracy of the estimated EiH transformation obtained from OpenCV and ViSP the results will be compared to the measured approximates transformation in the table 5.4. The estimated EiH obtained from OpenCV will be compared to the measured approximates transformation from TCP to camera shown in table 5.1 (2) and the EiH obtained from ViSP will be compared to the measured approximate transformation from flange to camera shown in table 5.1 (1).

	Translation			Rotation		
Measurements between frames	X [mm]	Y [mm]	Z [mm]	rx	ry	rz
(1) Flange to camera approx.	-22.5	47.1	11.8	0	0	0
(2) TCP to camera approx.	22.5	47.1	-168.1	0	0	0
Calibration method OpenCV						
Tsai	44.6	110.5	157.5	0	0	0
Park	39.4	54.0	159.6	0	0	0
Horaud	39.4	54.0	176.6	0	0	0
Andreff	27.8	37.0	30.5	0	0	0
Daniilidis	5.9	59.7	226.2	0	0	0
Calibration method VISP						
Flange to camera transformation	18.2	15.6	10.3	0	0	0

Table 5.4: Difference in the obtained estimations from OpenCV and ViSP compared to the measured estimation.

The overview of the difference in result can be seen on table 5.4. The table shows that some result are more promising than others. Among this are the `Andreff` method from OpenCV which has a precision of under 40$[m]$ in each axis. Likewise the result from ViSP keeps the difference in values under 20$[mm]$ for each axis. However, other results as the `Park` and `Horaud` methods from OpenCV seems to have a decent estimation in the X- and Y-axis, but have a much larger difference in the Z-axis. After the verification of the EiH estimation from OpenCV and ViSP an experiment of the static measured EiH is constructed.

5.3.4 Pose estimation static eye in hand

In this experiment the precision of the measured EiH transformation from section 5.3.1 is tested when performing pose estimation from the TCP to the CDC. To perform the experiment the robot is positioned in the default position with the CDC inside the FOV as shown in figure 5.10 to the left. From this position the CDC is detected and the pose

estimation is calculated.

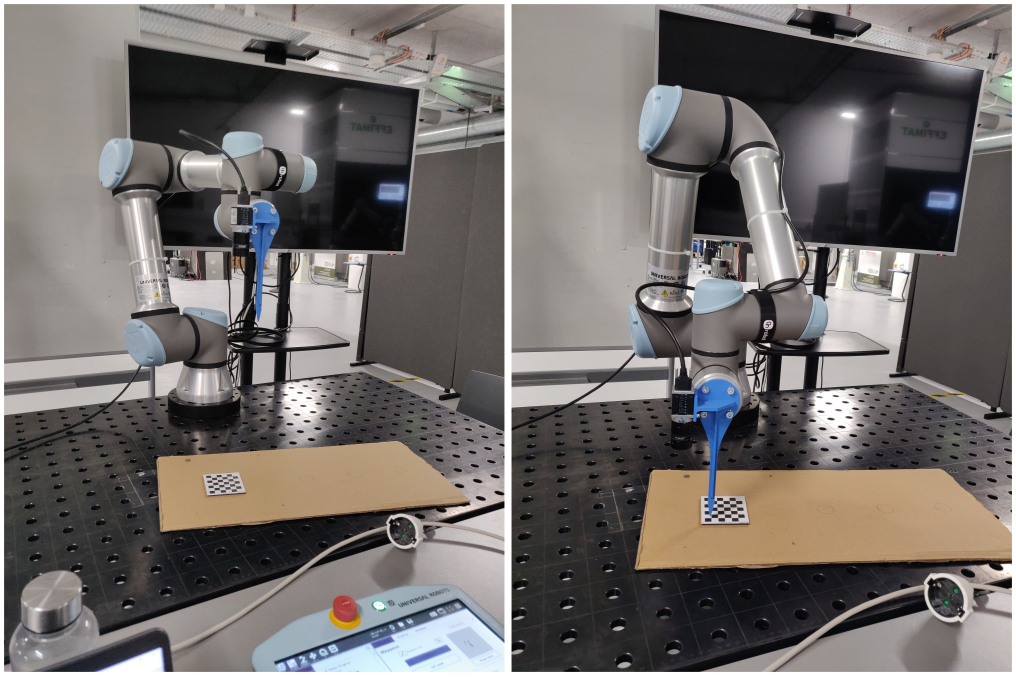


Figure 5.10: Robot at default position on the left and robot at estimated world frame pose on the right.

When the pose estimation is calculated the program sends a command to move the robot to the given estimated pose as shown in figure 5.10 to the right. To achieve the precision of the estimated pose the actual pose of the CDC is found by manually moving the robot through the teach pendant. The difference between the estimated pose and the actual pose is then able to be calculated. This test is done with 19 different positions and orientations of the CDC as seen in Appendix 5c where the precision of the 19 different positions and orientations are documented along with two images for each pose estimation in Appendix 5d. In table 5.5 the average difference between the pose estimation and the actual pose is shown.

	Translation			Rotation		
Precision of 6D pose estimation	X (m)	Y (m)	Z (m)	rx	ry	rz
Average difference of each axis	0.0011984	0.0020179	-0.0020968	0	0	0

Table 5.5: Pose estimation with the static eye-in-hand transformation. The experiment of the estimation is not limited only with the translation, it also features the rotational part. Due to the complexity of the experiment, the rotation vectors has been excluded.

The precision of the X-axis is off target with 1.1984 [mm] which means that the estimated pose is approximately 1.2 [mm] further away in the X-direction with respect to the actual pose. The Y-axis is off target with 2.0179 [mm], which means that the estimated pose is approximately 2.02 [mm] further away in the Y-direction with respect

to the actual pose. The Z-axis is off target with -2.0968 [mm], which means that the estimated pose is approximately -2.1 [mm] above the target with respect to the actual pose. The above experiments has verify different aspect e.g. calibration, precision and pose estimation. Before the conclusion of the project, it should be discussed what the result of the experiments mean for the project.

Chapter 6

Discussion

The following chapter is to discuss the most relevant results of the project to reflect on the chosen decisions. The structure continues the order from the implementation, analysis, and experiment. For this discussion the results of the calibration, estimation and implantation of the controller will be addressed.

6.1 Calibration

To be able to correctly map the coordinates between the image plane and world frame, it is essential to have a correct calibration to fix the distortion of the camera lens as mentioned earlier. The numbers of pictures required is difficult to determine, because it correlates with different reasons. There was a higher demand for differing poses from EiH method, than the calibration it self. In our technique everytime we create the pylon object for image grabbing, we decided on how we want to grab the pictures. The robot then moves to one or multiple poses, to get variations in the grabbed pictures. In order to estimate the pose, we needed the TCP poses of the robot while grabbing the image, the camera to object estimation of translation and rotation, and the robots joint configurations. The robot joint configurations is used when calculating the FK of the robot arm to get the homogeneous transformation from the robot base to the robot flange.
When working on the calibration, we often experience a phenomenon where the image stream appeared undistorted, even though it came from the stream with no changes to the input. In theory that is a good thing, but a lot of time was spend figuring out how our approach to this should be. In the final stage we overlooked a specific data sheet containing detail information of the lens mounted on the camera. The documentation present a geometric distortion of the lens to be approximately -0.7%, that explain why the unaltered stream seems to be calibrated. We focused on completing the calibration in most cases anyway, of the idea that if the image stream was indeed undistorted, we

would then just experience a calibration result that needs less correction. As we know the camera stream has a slight distortion of minimal -0.7%, that should correlate with a small distortion correction. The final controller program, uses in practical the estimation to CDC and `procjetPoints()`, before calibration with the last used calibration matrix. We discussed that even though that the estimation could work before the calibration, would not mean that there was not any distortion and therefore, possible complication when actual estimate a movement to the corner of the CDC which defined the world frame. The further verification could reach the conclusion that the implementation, could work without calibrating the camera, if the result comes extremely close the error percent should not matter.

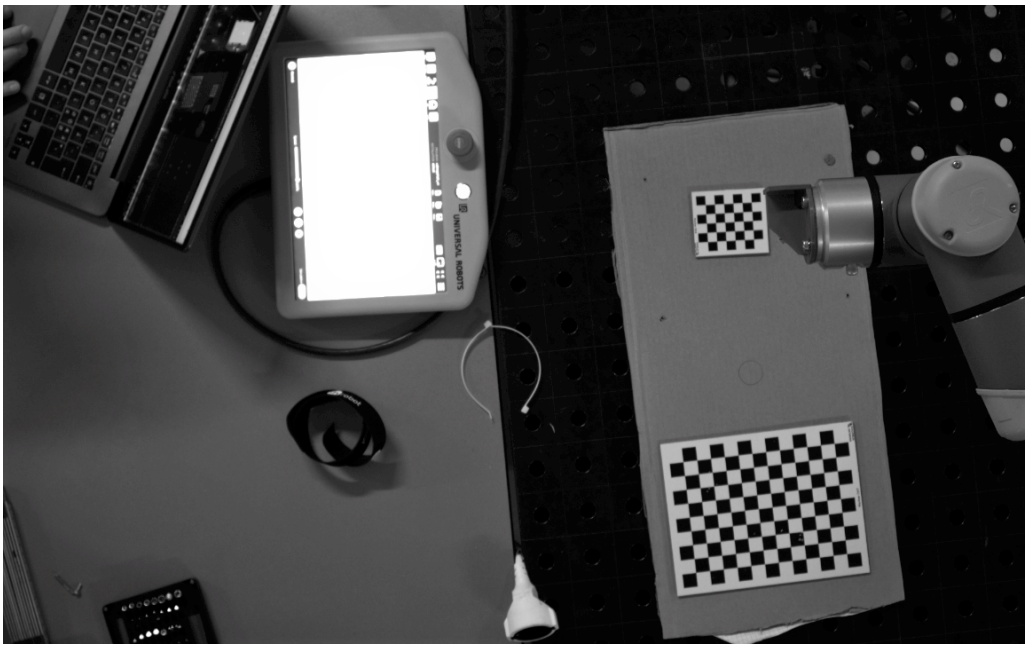


Figure 6.1: Illustration of the raw image output from the camera placement above the setup, this is with an increased height, so the FOV appear larger.

The overview was made 2 meters above our setup, and the image plane shows a clear, focused undistorted image where the lines are perpendicular with the environment objects and table sides. Besides the calibration the EiH was important to the project, and this is why it was important to get a clear and correct image representation of the CDC. The calibration is achieved by collecting images in a different set of robot arm positions. The amount of images is not limited to a certain number, but lies upon the approach that yields the best calibration. After the discovery of the mistaken assumption, the procedure for the implemented code holds the same structure. The procedure does rely on a higher number of TCP poses than calibration images, for the estimation of CDC. The essential for the discovering has been overcome and the understanding was discussed that the procedure only has to be run ones every time it needed to define a new CDC.

6.2 Estimation

For the implementation of the EiH calibration, the two libraries used was as mentioned in section 3.2 the OpenCV function `calibrateHandEye()` and the ViSP function vpHandEye-Calibration::calibrate(). The OpenCV function `calibrateHandEye()` was chosen as method to achieve the EiH transformation because OpenCV already was included in the program due to the decision of manipulate the pylon image input through the C++ controller program, it was a best practice to convert to the OpenCV. After experiments on the performance of the EiH from the OpenCV, was it clear that other possible solution needed to be found. The precision was not as expected and can be seen in the section 5.3, because of that we found ViSP and started implementing that approach inside the classes.
The initial course of action to project was as previously mentioned with a model-based approach. As opposite to the model-based, the model-free approach relies not on planning and knowledge but definitively on a trial and error approach when learning [33]. One of the advantages the model-free approach can have over the model-based approach is the difficulty of constructing an environment model with high enough accuracy for the model-based approach.

6.2.1 OpenCV

When implementing the OpenCV EiH method, one advantage is the function having the five different methods with each specific approach to the problem. Giving the user a possibility to obtain different result using the same data set.
During this project when testing the EiH from OpenCV it has been proven to be difficult for obtaining a correct data set for the method to compute an accurate transformation from the gripper to the camera. The problem of obtaining the right data set was iterated through multiple possible solution where different kind of conventions for the conversion to a rotation matrix was used. To present a solution for the EiH, the choice was to find a different approach during the project. The above functions from OpenCV gave inaccurate results, so the solution was to look elsewhere for a EiH solution.

6.2.2 ViSP

the alternative approach were to examine the ViSP library, to conclude that it was possible to create the estimation of the camera placement. Upon implementation of ViSP one disadvantage was the lack of information about the approach the method uses, which led to misunderstanding when manipulating the data set provided to the function. One advantage of the ViSP function was the easy implementation of the method and the existing environment suited for manipulating transformation matrices and methods for

converting the rotation between different conventions.

6.2.3 Static measurement

As stated the project focus was to implement a dynamic positioning system. This includes an automatic EiH transformation to obtain the complete kinematic transformation of the physical system. As documented in the experiment in section 5.3 the EiH transformation from OpenCV and ViSP was not accurate enough. This resulted in the pose estimation that was only tested with a measured approximate transformations between the end effector and the camera.
When testing the pose estimation with the static measurement of the transformation the advantage was to get the hole system of kinematic transformations verified and test the precision of the system with the static measured transformation as the substitute for the automated EiH transformation. A disadvantage was to not be able to test the pose estimation using the inaccurate automated EiH transformation fulfilling the dynamic positioning system.

6.3 Control

From the beginning of this project, the idea was to fully implement a working teach pendant program, with help of a basic controller program and URCap functionality. A helpful way of achieving the project, would be to incorporate ROS by wrapping the different parts of the system as an easy way to deliver the contribution. Even though parts the URCap has been examined, some further work should have been done to wrap our project in a ROS environment. The time was allocated for solving the EiH instead, and the result of that a full working system was not done. For control and interaction with the robot, an interface has been used to engage with the robot. The movements selected to calibrate the robot has been design in the idea of maintaining the height of the initial position above the CDC. The idea comes from the principle of during a calibration around a CDC in the shape of hemisphere from ViSP documentation [37].

Chapter 7

Conclusion

To conclude the project, a review of the problem statement is considered. The developed implementation are able to calibrate and detect a CDC marker fixed on a work board and perform pose estimation from the robot arm's TCP with detection data from the vision camera. The EiH process has been implemented to work automatically with the rest of the system. However, the result obtain from the EiH calibration process is confirmed to have a small inaccuracy compared to the measured, and that involves a complete pose estimation. Therefore a measured EiH transformation is used to test the whole system. The data collected from robot arm and the calibration process is stored in `.txt` files with easy access from other parts of the system to obtain if needed. Some other data management could have been further implemented, but a conclusion to this project were to rewrite a fairly simple solution, and the controller program can now be configured to work on different kinds of setup and hardware. As documentation of the achieved calibration process with 25 image poses see the video in Appendix 8a. The achieved pose estimation with transformation matrices can be seen on the video in Appendix 8b. Lastly, the documentation of the achieved detection of the CDC and continuous centering adjustment by the robot arm, can be seen on the video in Appendix 8c.
Regarding the workflow of the project it is concluded that the model-based approach works for this kind of development process. However, the model-free approach could have been preferred because of its agile nature and can easily react to new and unforeseen issue.
Each separate part of the project will be concluded below to summarize subproblems that occurred through the development process. Some reflections are discussed with usage of the camera, and the conclusion is that a smaller camera is more suitable for the task, than a larger vision camera. The change of the camera would mean that the calibration would play a larger role in the project, as the geometrical distortion likely would be more influencing. In that case, the project implementation is prepared if the future implementation decide to go with the smaller raspberry pi camera, but there is a need for more

experiments to conclude on the precision. Regarding the detection and projection of the CDC, the transformation of image points from the CDC to the camera frame was achieved with high accuracy according to the pose estimation when using the measured EiH transformation. The measured distance does have a error variance in millimeters, and later in the project the results for the EiH seems to have an equal precision. The kinematics of the project have been implemented with the ability to obtain the FK of the robot arm and the transformations needed to achieve pose estimation. In the implementation of an estimated EiH the experiments shows that neither the OpenCV or ViSP function returned precise results when taking the measurements into consideration. When testing the pose estimation using the measured EiH values, the precision achieved was about ± 1.5 [mm] in each directional axis. With the respect to the EiH, the precision of some the result shows a precision of ± 15 [mm] in each directional axis. Even though that some of the results from the EiH differs from the same precision of the measured, the result is very close and further experiments have to be made in future work. To conclude on the choices of the movement, the robot arm could have been created by a teach pendant program with extended use of URCap functionality. By using the SDU UR-RTDE interface complicated how the end result could have been, despite that the result gives an advantages when the user engage in the project, the key reflections is to keep the simplicity and therefore some future work should be created. The controller program could have included a better user interface, to simpler interaction with the program. The design is made in a closed environment, not suitable for practicality and user friendliness but rather experimental use only. Other features to use could have been a stronger database management system, to better store relevant data that is necessary in the process. It should be debated in future work where the advantages and disadvantages lies of data management, and what the frames of the implementation of a closed system should be.

7.1 Future work

The next step is to finalize the work and wrap the project classes from, see Appendix 2a to a ROS environment, so that the program can be simplified. There exist several useful guides and articles, explaining additional entries to wrap the classes and by continue using existing EiH estimation algorithms. Furthermore the work from the teach pendant and URCap program see Appendix 4a, can be further examined, to be able to control the robot arm and set a robot task with respect to the CDC. The project implementation manage to complete the part of a finished teach pendent program with extensions feature to communicate to the controller program. The future work lies inside a combination of change in URCap and the controller program, so the implementation not only receive robot

arm poses from the UR-controller, but also defines global variables from the controller program. Different approaches has been examined to solve the problem, but it is here that some future work needs to be implementing. To extend the user experience, there could have been a better UI to present the project flow. Additional work to the C++ controller program should have been to rewrite the code implementation so that the pose estimation fully utilize some of the better estimated results of EiH, as some feature a reasonable solution. A revision of the code could eliminate the static measured distance of the camera and a full estimation could have been incorporated. This is only possible because our result of EiH lies close to the precision error we have with physical measuring.

Bibliography

[1] Enabled Robotics ApS. *Landmarks: Product Overview*. URL: https://www.enabled-robotics.com/erflex. (accessed: 17-12-2021).

[2] Mobile industrial robots A/S.

[3] Universal Robots A/S. *DH PARAMETERS FOR CALCULATIONS OF KINEMATICS AND DYNAMICS*. URL: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/. (accessed: 14-12-2021).

[4] Universal Robots A/S. *Download*. URL: https://www.universal-robots.com/download/?option=62055#section61793. (accessed: 13-12-2021).

[5] Universal Robots A/S. *REAL-TIME DATA EXCHANGE (RTDE) GUIDE*. URL: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/. (accessed: 10-12-2021).

[6] Universal Robots A/S. *Universal Robot UR5e*. URL: https://www.universal-robots.com/products/ur5-robot/. (accessed: 09-12-2021).

[7] Universal Robots A/S. *URCap Software Development Tutorial Swing*. URL: https://plus.universal-robots.com/media/1810567/urcap_tutorial_swing.pdf. (accessed: 21-12-2021).

[8] Basler. *Basler ace 2*. URL: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace2/a2a1920-160umbas/. (accessed: 17-12-2021).

[9] Basler. *Basler Lens*. URL: https://www.baslerweb.com/en/products/vision-components/lenses/basler-lens-c125-1218-5m-p-f12mm/. (accessed: 22-12-2021).

[10] Basler. *Pylon 5.2,0*. URL: https://www.baslerweb.com/en/sales-support/downloads/software-downloads/pylon-5-2-0-linux-x86-64-bit/. (accessed: 28-12-2021).

[11] Raman Paranjape & Mehran Mehrandezh Chaiyapol Kulpate. *Precise 3D Positioning of a Robotic Arm Using a Single Camera and a Flat Mirror*. URL: https://www.tandfonline.com/doi/full/10.1080/15599610802301243. (accessed: 15-12-2021).

[12] Konstantinos Daniilidis. *Hand-Eye Calibration Usin Dual Quaternions*. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.366&rep=rep1&type=pdf. (accessed: 16-12-2021).

[13] Olivier Faugeras. *Three-Dimensional Computer Vision: a Geometric Viewpoint*. URL: https://www.researchgate.net/publication/30869787_Three-dimensional_computer_vision_a_geometric_viewpoint. (accessed: 29-12-2021).

[14] Adam Hayes. *Empirical Rule*. URL: https://www.investopedia.com/terms/e/empirical-rule.asp. (accessed: 02-01-2022).

[15] Atanas Gotchev & Emilio Ruiz Morales Ihtisham Ali Olli Suominen. *Methods for Simultaneous Robot-World-Hand–Eye Calibration: A Comparative Study*. URL: https://www.mdpi.com/1424-8220/19/12/2837. (accessed: 15-12-2021).

[16] University of Illinois Kris Hauser. *Robot kinematics*. URL: http://motion.cs.illinois.edu/RoboticSystems/Kinematics.html. (accessed: 16-12-2021).

[17] Fritz Labs. *Pose estimation guide*. URL: https://www.fritz.ai/pose-estimation/. (accessed: 19-12-2021).

[18] Jingshu Liu & Yuan Li. *An Image Based Visual Servo Approach with Deep Learning for Robotic Manipulation*. URL: https://arxiv.org/ftp/arxiv/papers/1909/1909.07727.pdf. (accessed: 15-12-2021).

[19] Bernard Espiau Nicolas Andreff Radu Horaud. *On-line Hand-Eye Calibration*. URL: https://hal.inria.fr/inria-00590109/document. (accessed: 16-12-2021).

[20] Edwin Olson. *AprilTags*. URL: https://april.eecs.umich.edu/software/apriltag.html. (accessed: 30-12-2021).

[21] Omron. *Landmarks: Product Overview*. URL: https://robotics.omron.com/collaborativerobots/main-features/. (accessed: 17-12-2021).

[22] OpenCV. *Calibrate hand eye OpenCV*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#gaebfc1c9f7434196a374c382abf43439b. (accessed: 16-12-2021).

[23] OpenCV. *OpenCV function cornerSubPix()*. URL: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e. (accessed: 17-12-2021).

[24] OpenCV. *OpenCV function findChessboardCorners()*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a. (accessed: 17-12-2021).

[25] OpenCV. *OpenCV function projectPoints()*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c. (accessed: 17-12-2021).

[26] OpenCV. *OpenCV function solvePnp()*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d. (accessed: 17-12-2021).

[27] OpenCV. *OpenCV Homepage*. URL: https://opencv.org/. (accessed: 28-12-2021).

[28] Frank C. Park and Bryan J. Martin. *Robot Sensor Calibration: Solving AX = XB on the Euclidean Group*. URL: https://leeway.tistory.com/attachment/cfile4.uf@2058100F49D1B6387C0ADD.pdf. (accessed: 16-12-2021).

[29] Raspberry Pi. *Raspberry Pi High Quality Camera*. URL: https://raspberrypi.dk/produkt/raspberry-pi-hq-camera/. (accessed: 15-12-2021).

[30] Fadi Dornaika Radu Horaud. *Hand-eye Calibration*. URL: https://hal.inria.fr/inria-00590039/document. (accessed: 16-12-2021).

[31] SDU Robotics. *Indroduction*. URL: https://sdurobotics.gitlab.io/ur_rtde/introduction/introduction.html. (accessed: 10-12-2021).

[32] *Robotics in UR Robots*. URL: https://aws1.discourse-cdn.com/business7/uploads/universal_robots/original/2X/d/d4588fd696b452e4b6566f4f44ada81d9f6e6fe6.pdf. (accessed: 28-12-2021).

[33] Ram Sagar. *What Is Model-Free Reinforcement Learning?* URL: https://analyticsindiamag.com/what-is-model-free-reinforcement-learning/. (accessed: 30-12-2021).

[34] Mili Shahs. *Solving the Robot-World/Hand-Eye Calibration Problem Using the Kronecker Product*. URL: https://www.researchgate.net/publication/275087810_Solving_the_Robot-WorldHand-Eye_Calibration_Problem_Using_the_Kronecker_Product. (accessed: 18-12-2021).

[35] Aalborg University Thomas Petersen. *A Comparison of 2D-3D Pose Estimation Methods*. URL: https://projekter.aau.dk/projekter/files/14427578/A_Comparison_of_2D-3P_Pose_Estimation_Methods.pdf. (accessed: 19-12-2021).

[36] Roger Y. Tsai and Reimar K. Lenz. *A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration*. URL: http://kmlee.gatech.edu/me6406/handeye.pdf. (accessed: 16-12-2021).

[37] VISP. *Camera extrinsic calibration*. URL: https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-calibration-extrinsic.html. (accessed: 21-12-2021).

[38] wikipedia. *Denavit–Hartenberg convention*. URL: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters. (accessed: 14-12-2021).

[39] Amy Tabb & Khalil M. Ahmad Yousef. *Solving the Robot-World Hand-Eye(s) Calibration Problem with Iterative Methods*. URL: https://arxiv.org/abs/1907.12425. (accessed: 18-12-2021).

[40] Zhengyou Zhang. *A Flexible New Technique for Camera Calibration*. URL: https://ieeexplore.ieee.org/document/888718. (accessed: 16-12-2021).

[41] Zivid. *Understanding the importance of 3D hand-eye calibration*. URL: https://blog.zivid.com/importance-of-3d-hand-eye-calibration. (accessed: 30-12-2021).

External Appendices

1. **Data**
   (a) Calibration output
   (b) Robot poses output
   (c) Image output
   (d) Detection output
2. **C++**
   (a) Full C++ program
3. **MATLAB**
   (a) Calculations
4. **Java**
   (a) Full Java program
5. **Experiments**
   (a) Reprojection error test
   (b) Calculations from reprojection error test
   (c) Measured pose estimation
   (d) Images output of pose estimation test
6. **Transformation calculations**
   (a) Calculation of the transformations using DH parameters
7. **Miscellaneous**
   (a) Symbol list for the calculations
   (b) Universal Robot UR5 data sheet
8. **Videos**
   (a) Video of calibration process
   (b) Video of pose estimation
   (c) Video of CDC detection