MASTER'S THESIS 2024

Camera Calibration for Alignment of a Real World Setup to a Simulated Environment

Ludwig Jakobsson, Oskar Larsson

ISSN 1650-2884
LU-CS-EX: 2024-01

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-01

Camera Calibration for Alignment of a Real World Setup to a Simulated Environment

Kamerakalibrering för justering av en verklig miljö till en simulerad miljö

Ludwig Jakobsson, Oskar Larsson

Camera Calibration for Alignment of a Real World Setup to a Simulated Environment

(Ensuring Consistency and Accuracy in Sim2Real Alignment)

Ludwig Jakobsson
ludwig.jakobsson.560@student.lu.se

Oskar Larsson
oskar.larsson.671@student.lu.se

January 8, 2024

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors:
Main Supervisor, Alexander Dürr, alexander.durr@cs.lth.se
Co-Supervisor, Volker Krueger, volker.krueger@cs.lth.se

Examiner: Elin Anna Topp, elin anna.topp@cs.lth.se

Abstract

This thesis investigates camera calibration for use in computer vision systems. The intended use case is camera alignment between a simulated environment and a real-world setup to facilitate sim2real learning. This is a two-step process, intrinsic and extrinsic calibration. In both cases, the effects of several factors are investigated. These include image/pose count, rotational variation, and distance. Included is also a proposal for a method to verify the calibration results in the form of a pick-and-place task with HSV object detection. The camera calibration is performed using a ChArUco calibration board, a camera attached to the robot's end-effector, and two external cameras, one ceiling-mounted and one in front of the robot, facing the workbench.

Keywords: MSc, Sim2Real Alignment, Computer Vision, Reinforcement Learning, Robotics, ArUco, ROS, Image Analysis

Acknowledgements

First, we would like to thank our supervisor Alexander for the project, the regular feedback, and the consistent support provided throughout this project. We would also like to thank our examiner Elin for their continued patience throughout the project. For his valuable input on report writing, we would like to thank Volker. Lastly, a thank you to Josefin and Pontus for their early input on camera calibration and ROS.

Contents

Workload Distribution: The work of this thesis was split equally and has been conducted through a hybrid approach, combining remote collaboration and on-site testing of the robot. The code was divided into segments, which were distributed evenly and implemented both individually and together. These segments were then applied, tested, and merged into the system together, ensuring all the code was reviewed. The results of the merged segments were analysed and discussed and future design choices were taken. The same approach was taken when preparing the report, with segments written individually and then reviewed by the other author. Graphs and tables were produced by Oskar and illustrations by Ludwig.

Chapter 1
Introduction

1.1 Motivation / The Problem

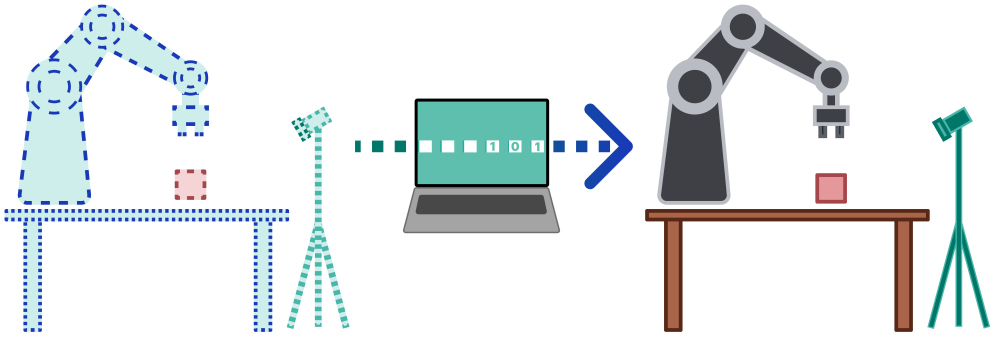


Figure 1.1: Aligning a real setup environment to simulated environment

Aligning simulation to reality (sim2real)

To make robots useful they need to know how to do things. One way of teaching them is to use reinforcement learning (RL), a method that involves simulating an environment and learning from the actions taken within it. While humans might need just a few attempts to master a task, an RL Model often requires up to thousands of trials to achieve the same level of proficiency [36]. Performing these learning attempts using the real robot would take a couple of seconds or minutes for each try. Such a large amount of learning trials required for a task would result in a lot of wear and tear on the machine and, depending on the task, weeks, months or years of learning.

To avoid this problem one could move the learning process to a simulated environment, a place where each try could be sped up and performed in a fraction of the real time, while also removing mechanical damage. For this to work, the real-world setup and the simulated environment need to be accurately aligned with one another to be able to successfully transfer the learned behaviour from simulation to reality.

General Calibration Problem

A more general problem exists in the field of robotics, where camera feedback plays a crucial role in solving tasks. One of the main reasons for using cameras is to accurately determine the position of the objects required to solve the task. This can be achieved using various image analysis techniques and depth sensors. However, these methods only provide the position with respect to the camera's coordinate system.

Therefore, it is essential to translate the position from the camera's coordinate frame to the robot's coordinate frame to successfully solve tasks, for example, assembly, opening doors or picking up milk cartridges.

In this report, we will explore how to perform both of these by determining the camera's position in relation to the robot in a real-world setting as a way of mimicking a simulated setup.

1.2 Related Work

The field of computer vision (CV) and calibration has been an active area of research for several decades. The ability to extract meaningful information from images and videos to control robots is crucial for a wide range of applications, including manufacturing, assembly, and inspection.

In recent years, there has been a growing interest in using robots in collaborative environments where they work alongside humans. This has led to the development of new robot platforms such as the Franka Emika Panda [7], and KUKA iiwa [14], which are designed to be lightweight, safe, and easy to use. However, to make these robots more autonomous, more sensors are added. One of the challenges of using these robots in collaborative environments is the need for precisely calibrated cameras used for visual perception. Calibration is required to get an accurate description and feedback from the world that the robot bases its behaviour on.

A Deep Learning-Based Hand-eye Calibration

A possible approach for calibrating the camera in relation to the robot is a deep learning-based approach to find the unknown geometric transformation between a robot's base and an external camera [2]. This can be done by training a deep learning-based regression model to estimate the unknown transformation assuming that the robot and intrinsic camera calibration are known. The RGB and depth images of a camera are fed into a neural network which outputs the estimated extrinsic parameters of the camera.

Continuous hand-eye calibration using 3D points

Two of the usual approaches for solving the calibration problem are a mathematical formulation of the problem which provides high accuracy and a self-calibration approach that uses feature matching which reduces accuracy but removes the need for a calibration object. A third method is to base the calibration on translation only which results in that the calibration object can be reduced to a single 3D point making it a flexible solution for continuous hand-eye calibration [11].

1.3 Contribution

- **Development of a platform-agnostic camera calibration system :** We present the design and implementation of a versatile camera-robot calibration system for performing intrinsic and extrinsic calibration across different robotic platforms and cameras. The main goal of the system is to facilitate sim2real alignment while also providing a tool to help solve more general tasks, such as object picking.
- **Enhancement of calibration quality:** The research gives insight into what contributes to good results and explores ways to improve the quality, performance and robustness of the camera calibration.
- **Guide for camera calibration:** The thesis also aims to be a guide for understanding and using the built system and use it. It also provides a real-world test application of "pick and place" with coloured cubes and a "pinpoint" test to showcase the use of the calibration system

1.4 Research Questions

At the onset of the project, the goal was to facilitate sim2real learning by aligning a real-world setup to a simulated environment and assessing the quality with a reinforcement learning model. During the project, this was scaled back to more closely investigate camera calibration and how to verify such estimates.

Research Questions (RQ)

- RQ1: What would be a suitable method for camera calibration in the context of sim2real alignment?
- RQ2: How can the results of such a calibration be verified?
- RQ3: Which factors affect the quality of the calibration process?

Constraints

- Re-implementation of calibration algorithms, we will apply existing algorithms to build a working system for calibration.

- For this project, alignment will consist of being able to position cameras in specified locations without addressing the alignment of the camera lens or field of view.
- The system will be developed to be platform agnostic but will only be tested on the Franka Emika Panda robot [7] and the Intel Real-Sense cameras [20] due to time limitations.

1.4.1 Method

The methodology used for this project is inspired by the design science principles of problem conceptualisation, solution design, and validation [26]. These are principles that adhere well to our development of a camera calibration system and a broad overview of how they are applied is presented below.

- Problem conceptualisation by research.
  - Literature and background study about robotics and computer vision.
  - Explore alternatives for intrinsic and extrinsic camera calibration.
- Solution design by building a camera calibration system with the following aims
  - Facilitate sim2real alignment
  - Support general camera calibration
  - Be platform agnostic
  - Easy to use
- Validation by evaluation of the built system by.
  - Verify calibration with a "pick and place" and "pinpoint" test, together with sim2real alignment.
  - Evaluate it by finding methods that determine the accuracy and robustness of the calibration by sim2real alignment.

1.4.2 Outline

Chapter 2 presents the theory, concepts, and tools needed to understand the implementation and results of this thesis. In chapter 3, we present a three-step approach for building the system and verifying it, along with an overview of the implementation. Chapter 4 explains the specific experiments that were carried out for evaluation, and chapter 5 presents the results of those experiments. In chapter 6, the conclusions and reflections of those results are presented, along with answers to the research questions, limitations of the project and ideas for future work.

Chapter 2
Theory

2.1 Representing poses in 3D space

The goal of this section is to understand the transformations of points between different frames of reference. This includes navigating in three-dimensional space, finding the translation and rotation of a pose, understanding their representations, and performing transformations between various reference frames.

2.1.1 Coordinate System

In order to describe the position and orientation, also known as the pose of an object, we define a set of axes and a reference point. The reference point describes the pose of an object in relation to another, with the world usually being treated as the primary frame of reference and providing a global coordinate system for the whole scene.

2.1.2 Translation

Translation is described as a three-dimensional vector (x, y, z), representing the position of the object along each corresponding axis in the three-dimensional coordinate system.

In figure 2.1 we observe the position of bench B from the world frame W as the row vector $B_W = (x, y, z)$. Alternatively, the world could be seen from the reference frame of B as W_B. Assume $B_W = (1, 2, 3)$, then W_B would be the inverse, resulting in $W_B = (-1, -2, -3)$.

Homogeneous Coordinates

To be able to move an object, a solution would be to add the desired translation to its coordinate vector. For example, if we want to move the bench in figure 2.1 to the world origin, we could calculate the new position as $B_W - (x, y, z) = (0, 0, 0)$. However, direct addition

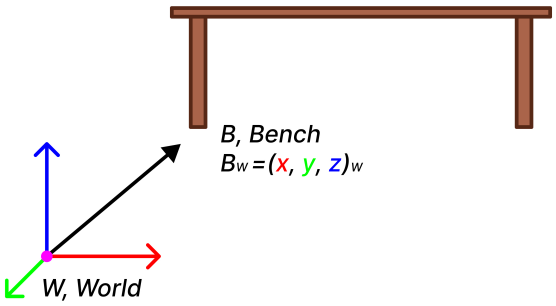


Figure 2.1: The figure shows a three-dimensional coordinate system with World W being the frame of reference, Bench B being represented as the coordinate vector $B_W = (x, y, z)_{world}$.

or subtraction for translation would lead to the loss of linearity. This non-linearity makes us loose the convenient property of being able to chain multiple transformations together using multiplication.

To address this problem, we introduce the concept of homogeneous coordinates. To convert our Cartesian coordinate to homogeneous coordinates, we simply append a "1" at the end of the vector, adding an extra dimension. Applying this conversion to B_w will result in the new vector $\bar{B}_w = (x, y, z, 1)$. This new coordinate representation allows us to utilise matrix multiplication for translating our objects. This is done by taking the identity matrix I and appending the desired translation to the last column of I, resulting in the transformation matrix T. The transformation matrix is then multiplied with the homogeneous coordinate to obtain the new position of the object.

$$\begin{pmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.1}$$

To convert the new position back to Cartesian coordinates the vector is normalised by dividing each element by the last element in the vector and then omitting it. Since T preserves linearity, we can then chain multiple transforms together by matrix multiplication to get one transform that represents them all [23].

So far we have looked at a way of describing the **translation** of an object in a three-dimensional coordinate system. Now we will look at how to describe the **rotation** of an object.

2.1.3 Rotation

There are multiple ways of describing the rotation of an object in a three-dimensional coordinate system, all with their advantages and drawbacks.

Euler Angles

$$\text{Euler Angles} = [\text{roll}, \text{pitch}, \text{yaw}] = [r_x, r_y, r_z] \tag{2.2}$$

Euler angles are usually represented as the three-dimensional vector 2.2. The rotation of the object is described by first doing a rotation on the x axis (roll), followed by a rotation on the y axis (pitch) and lastly a rotation on the z axis (yaw). These three rotations together define the object's final orientation. This notation offers a simple and intuitive representation of the orientation, making it easy to use and interpret [3].

Rotation Matrix

$$\text{Rotation Matrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

The rows in the matrix describe the rotation of each axis separately relative to the original orientation. It poses several important properties: all rows and columns are orthogonal unit vectors and the volume is always preserved during rotation since the determinant is always 1. Additionally, the transpose of the matrix is equal to its inverse, simplifying multiplication operations. Rotations can also be combined by multiplying the matrices, allowing for a sequence of rotations [22].

Axis-Angle Rotation Vector

$$\text{Rotation Vector} = [r_x, r_y, r_z]$$

Each element in the vector describes the angle of rotation around each respective axis. This notation is also known as the axis-angle representation. The rotation of the angles is performed at the same time, compared to Euler angles (roll, pitch, yaw) which are performed sequentially. The main benefit of this rotation vector format is that it is very intuitive to understand and interpret each value in the vector since it is a single rotation along the axis. Other benefits are its compactness and being more straightforward for interpolating than a matrix, making it useful in motion planning.

Quaternion

$$\text{Quaternion} = a + bi + cj + dk \tag{2.3}$$

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.4}$$

Compared to the other representations of rotation, quaternions take advantage of complex numbers to explain the rotation of an object. It is a four dimensional representation defined by equation 2.3, where a, b, c, and d are real numbers, and i, j, k are complex numbers that satisfy equation 2.4.

One key benefit of quaternions is their ability to preserve the order of rotations, similar to Euler angles. However, Euler angles are susceptible to Gimbal lock, a phenomenon where two axes align, causing a loss of one degree of freedom (DOF). This is something quaternions

inherently avoid with their representation, providing smooth interpolation between states at all times.

These properties, together with its compact representation in comparison the rotation matrix and its numerical stability make it widely used in robotics. The quaternion is commonly represented as the vector `[x, y, z, w]`, with w being the scalar and x, y, z as the imaginary components [33][10].

All results and findings are presented as quaternions, but every representation was used at some point during the project due to specific requirements from third-party packages.

2.1.4 Transform Matrix

Now that we have ways to describe both the translation and rotation of an object, we can put them together to describe the pose of an object with a transformation matrix.

$$\begin{pmatrix} \boldsymbol{R} & t \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

With both a translation and rotation part, the transformation matrix is capable of describing changes in both translation and rotation for a point as seen in figure 2.2. With the use of homogeneous coordinates, like in section 2.1.2, the vector can simply be multiplied with the transformation matrix to obtain the new position. Like the transform matrix described in 2.1.2, the linearity is kept with matrix multiplication giving it the ability to chain multiple transformations together, resulting in a new transformation matrix representing all the transforms.

As seen in equation 2.5 the pose of an object can be defined as a $4x4$ matrix consisting of a rotation part $\boldsymbol{R}$ and a translation part t. With $\boldsymbol{R}$ being the $3x3$ rotation matrix, see section 2.1.3, and t the $3x1$ translation vector, see section 2.1.2, then padded with 0s and a 1.

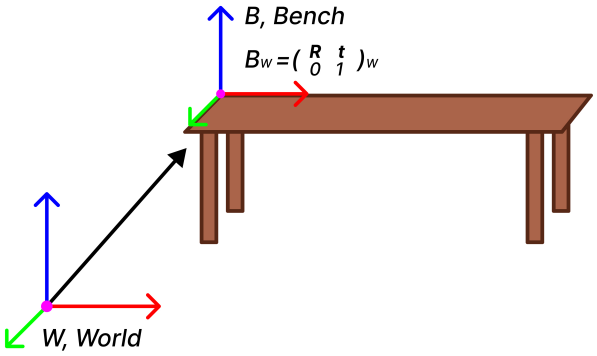


Figure 2.2: The figure shows Bench B with R being a $3x3$ matrix describing the **rotation** and t being a $3x1$ vector describing the **translation**. Together with padding, they make a $4x4$ transformation matrix describing the pose of B with World W as a reference point.

2.1.5 Transforming Coordinate Systems

An important part of understanding and navigating in three-dimensional space is moving between different frames of reference. From figure 2.3, assume the following transformations are known:

$$\text{World to Robot} = {}^{R}T_{W}$$

$$\text{Robot to Marker} = {}^{M}T_{R}$$

Then by applying the properties of transformation matrices, see section 2.1.4, we can express the marker with the world as frame of reference.

$$\text{World to Marker} = {}^{M}T_{W}.$$

By using matrix multiplication and making sure the order of transforms follows the path, the resulting transformation will look like

$${}^{R}T_{W}{}^{M}T_{R} = {}^{M}T_{W}.$$

Using these operations we are finally able to traverse the three-dimensional coordinate system.

2.2 Robotics

The goal of this section is to find the transform ${}^{ee}T_{b}$ between the robot base b and the end-effector ee. For this we need to know the kinematics of the arm, how they move and how to operate and receive feedback from them.

2.2.1 Robot Arms

The most common type of robotic arms are very similar to the ones of humans, comprising solid links l_n connected by joints j_n that can freely rotate. The number of joints is determined

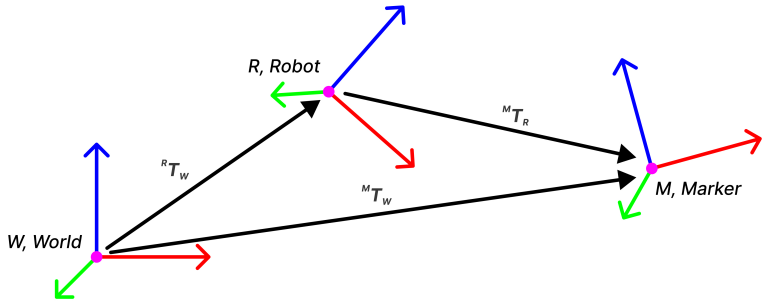


Figure 2.3: Coordinate system with World and Robot as a possible frame of reference for Marker M. The transforms between the points are represented as: ${}^{R}T_{W}$, ${}^{M}T_{R}$ and ${}^{M}T_{W}$

by the degrees of freedom needed for the task, with revolute joints providing rotational freedom, prismatic joints providing linear freedom and spherical joints providing multiple degrees of freedom [24]. At the end of each arm, there is usually an end-effector ee or gripper allowing the robot to interact with the task environment. Like the human arm, a robotic arm needs a solid base to operate from, typically defined as base b and can have the same frame of reference as world w. The lifting power and precision of the arm are determined by the stiffness of the links, the strength of the motors and the control algorithms [25].

2.2.2 Robot Operating System (ROS)

To avoid reinventing the wheel one can use the Robot Operating System (ROS), to control robots and provide a shared platform to make it easier for them to work together. It is an open-source framework, with a large collection of software libraries and tools to ease development of robots [4]. Here are some of the most important features of ROS [5]:

- **Nodes**, the framework of ROS is built around nodes. Nodes are modular units of computation, that represent the various components of the robotic system. The components are things like sensors, actuators or algorithms, each to control a single task, much like functions in programming.
- `roscore` is the main system used to connect the different nodes and allow for communication between them.
- `rospackage`, ROS projects are structured by an array of packages, each package usually containing scripts for nodes and launch files with parameters.
- **Topics** are open channels that are accessible by all the nodes, allowing transfer of data. A node can **publish** to a topic to send out data and **subscribe** to a topic to receive data.
- **Actions & Services** are used to request a node to do something. Services are used to perform the request synchronously with feedback on the request. Actions are used to

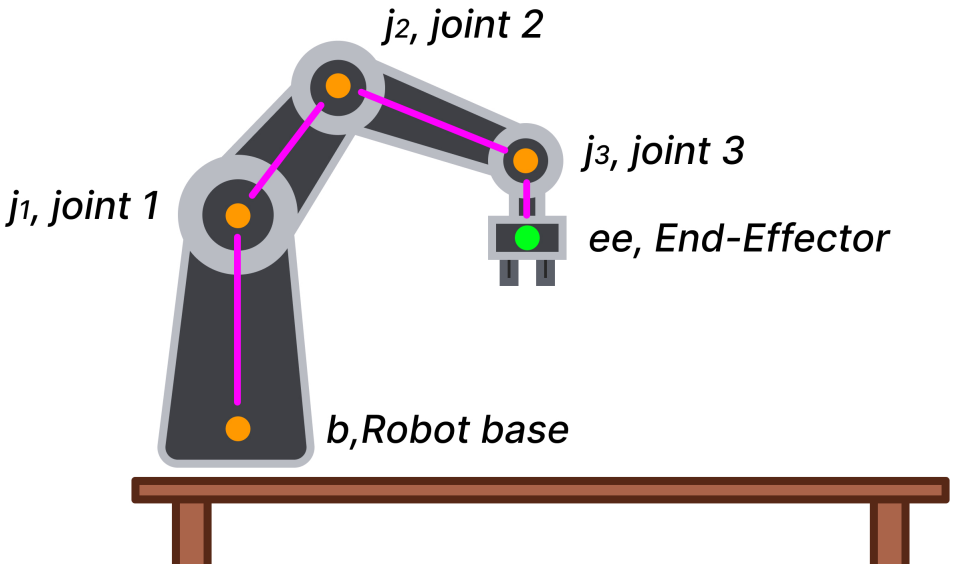


Figure 2.4: A depiction of a robotic arm with its joints, links and end-effector.

perform the request asynchronously, while sending progress of the request and allowing for preemption.

- **ROS CLI** is the interface used to control all parts of ROS. It is used to run nodes, view available topics, services and actions and manage all the packages.
- **RViz** is a visualisation tool for viewing everything from the robot system in 3D to available messages and topics.
- **Programming languages**, ROS provides support for both C++ and Python

2.2.3 Transform Frames

As seen in section 2.1.1, keeping track of poses with different frames of reference is a messy process. To ease this process, a transform frame (tf) library was developed [8].

To be able to explain the movement of the robot arms, each actuator is represented as a transform frame. A transform frame consists of:

- **Timestamp**, the time at which the frame was in that pose.
- **Frame name**, name of the frame.
- **Parent frame**, the reference frame for the pose.
- **Translation vector**, the three-dimensional space coordinates of the frame in relation to its parent frame.
- **Rotation vector**, described as a quaternion, see section 2.3, in relation to its parent frame.

[8]

Transform Frame Tree

Connecting multiple Transform Frames together, (as shown in Figure 2.5) creates a Transform Frame tree (TF Tree). By following a path in the TF tree, it is possible to determine the pose of a frame with respect to the starting frame of the path by utilising the coordinate transformation as shown in section 2.1.5. The transform frame system has become a convenient and essential component of the ROS system to access and view frames from different frames of reference [8].

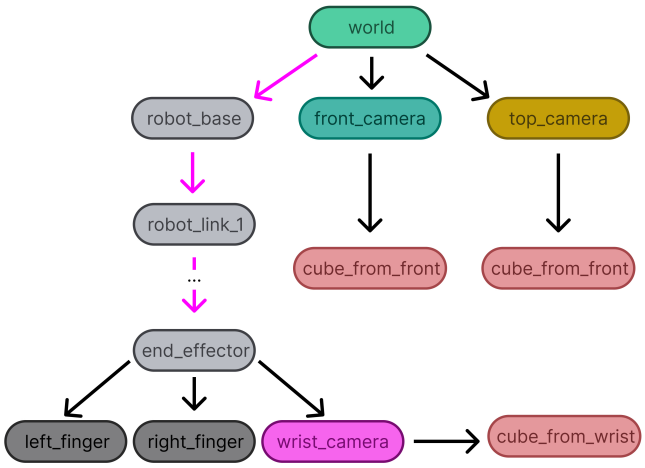


Figure 2.5: A depiction of a transform tree, representing a robot arm from figure 2.4 with its links, joints, end-effector and an attached wrist camera, as well as external cameras (front and top). Each camera also detects a cube from its point of view.

End-Effector Pose

Finding the pose of the end-effector then results in utilising the coordinate transformations in section 2.1.5 and traversing a TF tree.

$$ {}^{j_1}T_b * {}^{j_2}T_{j_1} * \ldots * {}^{j_n}T_{j_{n-1}} * {}^{ee}T_{j_n} = {}^{ee}T_b \tag{2.6} $$

2.2.4 Unified Robot Description Format

The Unified Robot Description Format (URDF), describes the geometry of a robot, including the physical components, joints and links, and their movement about another. Like the TF tree, see section 2.2.3, and also partly being the source of building the TF tree, it describes the robot system using a tree structure to explain the physical parameters, how each part of the robot is connected and what is attached (to) where. Additionally, URDF utilises tags to specify things like visual parameters for simulation, collision behaviour, and inertia for physics calculations. The URDF helps facilitate collision checking and dynamic path planning as well as translating the joint parameters to transform frames, see section 2.2.3 [21].

2.2.5 Inverse Kinematics

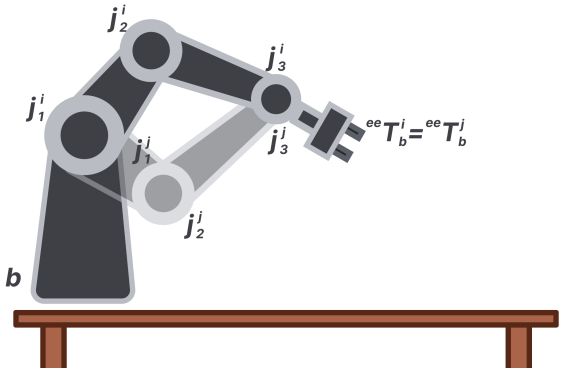


Figure 2.6: A depiction of a robot arm with the same transform between base and end-effector ${}^{ee}\boldsymbol{T}_b^i = {}^{ee}\boldsymbol{T}_b^j$ but with different joint states $(j_1^i, j_2^i, ..., j_n^i) \neq (j_1^j, j_2^j, ..., j_n^j)$.

There are two ways to view the movement of the end-effector of a robotic arm to a desired pose: 1) by following a path (*forward kinematics*) and 2) by setting a goal and figuring out a path to reach that pose (*inverse kinematics*).

Forward kinematics involves setting each joint of the arm $j_1, j_2, ..., j_n$ to specific values, translating it to transform frames, see section 2.2.4, and then chaining the transformations from the base to the end-effector together, as shown in section 2.2.3.

Inverse kinematics (IK), on the other hand, works the other way around. It starts with setting a goal pose for the end-effector and calculates the joint combination required to reach that pose.

The space of available joint states for each pose is determined by the DOF of the manipulator. To be able to reach an arbitrary pose in a three-dimensional workspace, 6-DOF is needed, i.e. translation in x, y, z and rotation around the axes roll, pitch, yaw. Fewer DOF could result in the pose being unreachable, and more DOF will result in an infinite number of joint states to represent the desired pose, while a non-redundant amount of DOF results in a unique solution [15].

There are two ways to solve for the joint parameters: analytical and approximate. For robotic systems with more than 6 DOF, approximate is the preferred method. The most common approach for solving it approximately is using the Jacobian inverse technique [16].

2.2.6 MoveIt

To avoid reinventing the wheel and solving the inverse kinematic problem ourselves, we used the open-source framework MoveIt [17] for easy manipulations in robotics. It provides support for collision detection, inverse kinematics, see section 2.2.5, motion planning and manipulation of individual actuators and grippers by utilising the URDF information, see section 2.2.4. It is widely adopted in the robotics community and includes support for many different robots, see section 2.2.1, and integration with ROS, see section 2.2.2.

2.3 Image Analysis

The goal of this section is to explore various markers used to find the pixel coordinates (u, v) representative of the marker in an image.

2.3.1 ArUco Markers

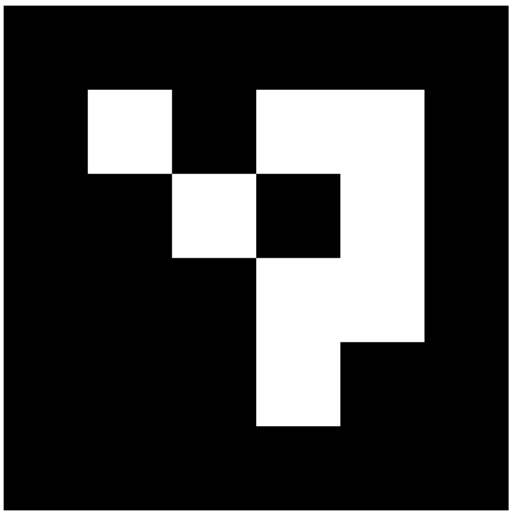


Figure 2.7: ArUco Marker representing the id 0 in the set DICT_4x4, meaning that the marker is 4x4 tiles excluding the border.

ArUco (Augmented Reality University of Cordoba) [18] markers are fiducial markers belonging to predetermined sets with unique patterns and known size properties, see figure 2.7 [9]. The unique patterns allow for the identification of a marker's pixel coordinates, $p = (u, v)$, via image segmentation. Having acquired said coordinates, they can be used along with the camera intrinsics, see section 2.4.2, and the known size properties to determine the transform between the camera and ArUco.

2.3.2 ChArUco board

A ChArUco (Chess ArUco) board consists of ArUco markers embedded within a checkerboard, see figure 2.8. This setup allows for more points with known properties and easier detection due to each ArUco marker being surrounded by black squares [28]. Additionally, each ArUco being detected individually within the board means that the board can still be detected while partially occluded.

Finding the ChArUco board is done by interpolating the corners between the found ArUco markers and calculating the position of the upper left corner from the known parameters of the board.

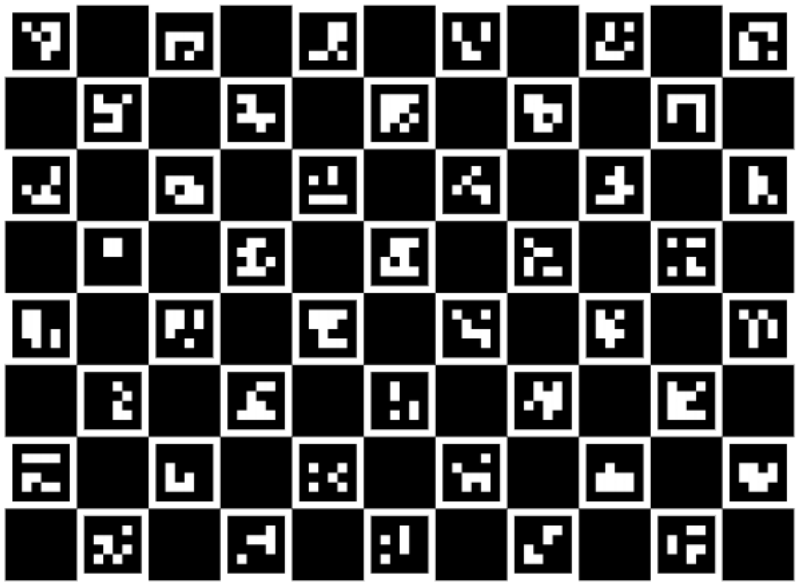


Figure 2.8: ChArUco, 8x11, ArUco DICT_4x4

2.3.3 HSV Object Detection

HSV Colour Space

The HSV colour space is a way to represent colours using a set of three quantities called hue, saturation and value (HSV) [34]. As can be seen in figure 2.9, the hue is cyclical, meaning that passing over the maximum value is equivalent to starting over from zero. Saturation governs how much white is mixed into the colour, with lower values being more white. Value performs the same role but for brightness.

Image Masking

An image mask is a two-dimensional matrix, sharing the same shape as the original image, with values constrained to being one or zero. Each such value represents a pixel in the original image, which when combined will yield a new image. In this image, only the pixels which corresponded to a one in the mask will be kept from the original image, and the pixels with a zero will be black.

This technique can be utilised in combination with HSV colour values to create a mask which only keeps pixels which correspond to specific HSV values. This space can be widened by providing a range of HSV values to keep, rather than just a single value.

Merely applying an HSV filter to an image can result in a noisy image if there are individual pixels which correspond to the provided HSV range. To combat this, morphological transformations [29] can be applied. This is done by convolving the binary image mask with a kernel.

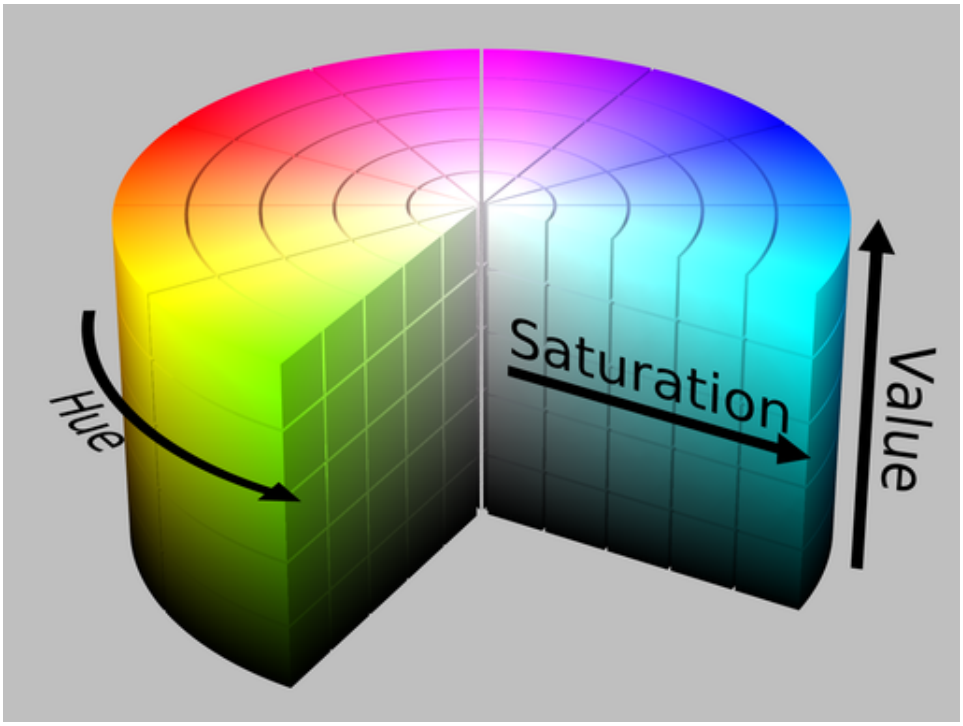


Figure 2.9: The HSV colour space. Image source: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png

2.4 Computer Vision

The goal of this section is to find the transform ${}^{t}\boldsymbol{T}_{C}$ between the camera $\boldsymbol{C}$ and the target t. Using the pixel coordinate of the target (u, v) found in section 2.3. This is done by using computer vision techniques to extract three-dimensional data from images and finding the intrinsic and extrinsic calibration parameters of the camera.

2.4.1 Coordinate Systems

In computer vision, there are four commonly used coordinate systems. How the systems relate to each other can be seen in figure 2.10.

1. **World coordinates** $(x, y, z)_W$, a three-dimensional Cartesian coordinate system, usually used as the main frame of reference in a scene.

2. **Camera coordinates** $(x, y, z)_C$, a three-dimensional Cartesian coordinate system representing points from the camera's frame of reference. The z axis of the camera usually points straight out of the camera lens (camera principal axis).

3. **Image coordinates** $(x, y)_i$, a two-dimensional coordinate system representing a projection of the camera coordinates onto a plane, as seen in figure 2.10. The centre point $(0, 0)$ of the image plane is usually on the camera's principal axis.

4. **Pixel coordinates**, (u, v), a discrete two-dimensional coordinate system representing the pixel values of the image, usually with $(0, 0)$ in the upper left corner of the image.

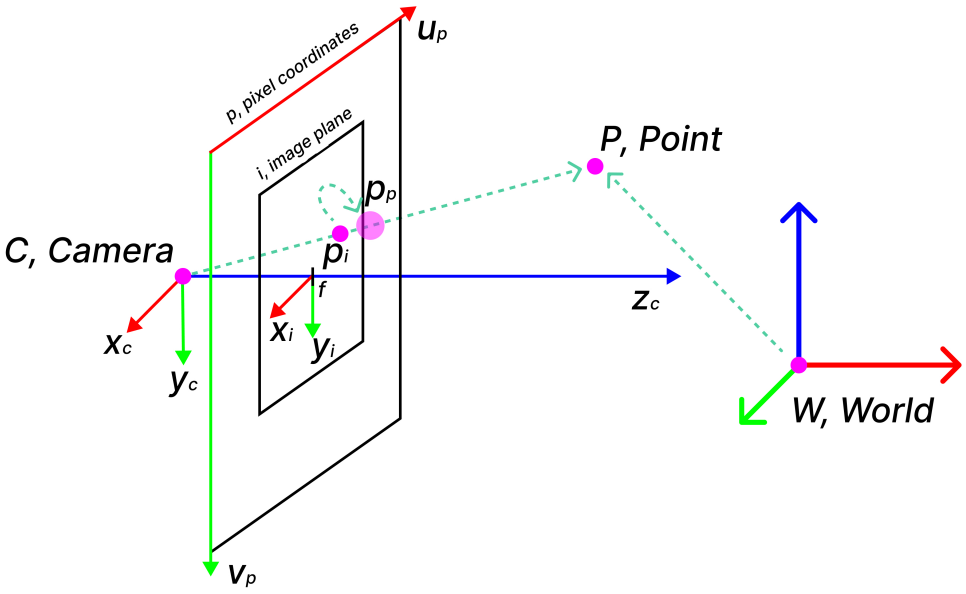


Figure 2.10: Shows target point P from the World coordinate system W and Camera coordinate system C, and the projection onto the point p_i in the image plane i, as well as the corresponding point p_p in the pixel coordinate system p and f representing the distance from the camera origin to the centre of the image plane on the cameras z axis.

2.4.2 Intrinsic Matrix

The intrinsic matrix explains the internal parameters of the camera. With the matrix K, mapping the camera coordinate to the image plane and then transforms it to a pixel coordinate as seen in figure 2.11. The intrinsic matrix is defined by equation 2.7 and the remapping is done by multiplying K with the camera point $P = (x, y, z)_C$ to receive the homogeneous pixel point $p_{p_h} = (u, v, w)$, which can then be converted to the pixel point $p_p = (u, v)$ like described in the end of section 2.1.2 about homogeneous coordinates.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

The intrinsic values are represented by a matrix K and vector D. Equation 2.7 describes K, where f_x and f_y are the focal lengths along the x- and y-axes while c_x and c_y are the coordinates of the centre point along selfsame axes.

Parameters:

1. **Focal length** (f_x, f_y), in a perfect camera f_x and f_y are the same but lens distortion, none-square pixels in sensors and variability in manufacturing always result in some error margin between the two axes. If one were to look at figure 2.11 and imagine the image plane sliding along the z_c axis, keeping the image plane size the same, one can see that the further away the plane is (increased focal length) the bigger the object will

be in the picture and the closer the plane is to the centre (decrease focal length) the smaller the object will be.

2. **Principal centre point** (c_x, c_y), describes the offset between the pixel coordinate origin $(u, v) = (0, 0)$ and the pixel coordinates that represent where the camera coordinate system's principal axis (z-axis) intersects the image plane at a distance of f.

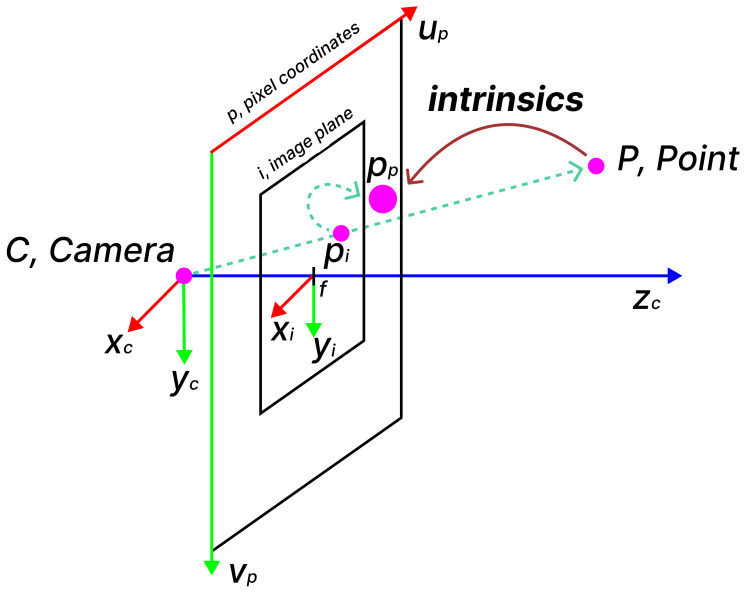


Figure 2.11: A depiction of the intrinsic matrix transform, transforming a camera point to a pixel point.

Together with the intrinsic camera matrix, a distortion vector, equation 2.8, is often used to combat lens distortion. The distortion will be individual to each camera lens. [12]

$$D = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix} \tag{2.8}$$

Equation 2.8 describes the vector D, representing the distortion coefficients of the camera lens. The distortion coefficients represent two types of distortion.

1. **Radial Distortion Coefficients** (k_1, k_2, k_3), parameters that represent the amount of radial distortion made by the camera lens, causing straight lines to look bent (fish-eye effect).

2. **Tangential Distortion Coefficients** (p_1, p_2), parameters that represent the amount of misalignment between the lens and the image plane, causing images to look skewed, tilted, closer or further away than they are.

There are different ways of removing distortion from an image, like iterating over each pixel and remapping it, cropping the image or interpolating. Finding the intrinsic matrix and distortion parameters will be explored in section 2.5.1 [37] [31]

2.4.3 Extrinsic Matrix

The extrinsic matrix, equation 2.9, consists of the rotation matrix R and translation matrix t for the camera, which describe the pose of the camera in relation to the World coordinate system as seen in figure 2.12. Whenever the position of the camera changes, the extrinsic matrix also changes [12].

$$[Rt] = \begin{pmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{pmatrix} \tag{2.9}$$

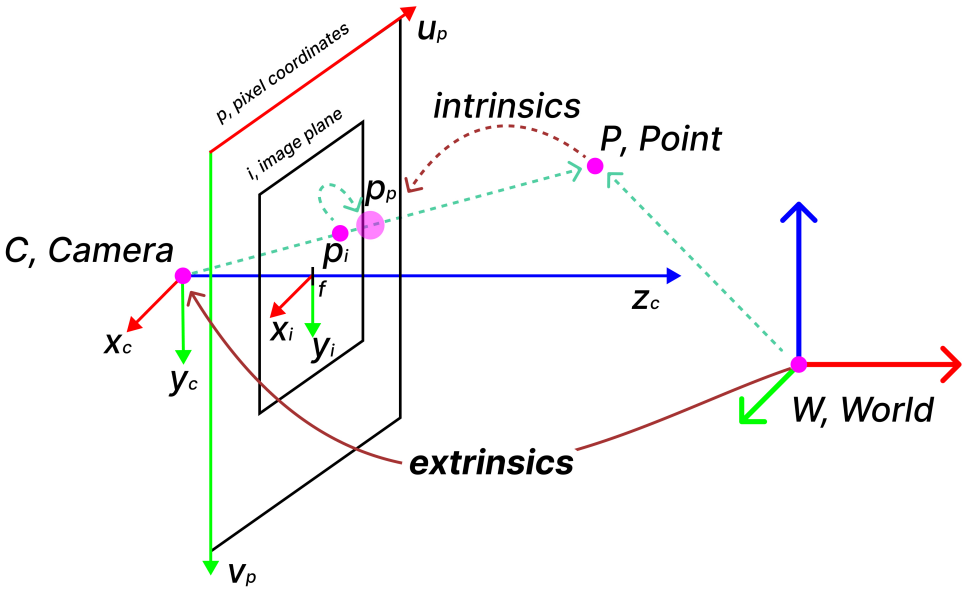


Figure 2.12: A depiction of the extrinsic matrix transform, transforming a world point to a camera point.

To convert a point $P = (x, y, z)_W$ from the world frame to the camera frame, the extrinsic matrix is simply converted to a transformation matrix, like in section 2.1.4, and multiplied with the homogeneous coordinate of P. Finding the extrinsic matrix will be explored in section 2.5.2

2.4.4 Pixel to Camera Coordinate

Since the camera-to-image plane transformation loses the depth data of the point, we can not go the other way around using the same technique. To solve this we need another way to get the depth together with the pixel coordinate. The depth of a point can be computed using a reference point with a known size, like the ArUco marker, see section 2.3.1, or using a depth sensor.

Parameters

f_x, f_y = intrinsic focal lengths
c_x, c_y = intrinsic image centre points
z_C = depth
(u, v) = pixel coordinate

Equations

$$x_C = \frac{u - c_x}{f_x} \cdot z_C \tag{2.10}$$

$$y_C = \frac{v - c_y}{f_y} \cdot z_C \tag{2.11}$$

$$z_C = z_C \tag{2.12}$$

2.5 Calibration Techniques

In this section, we will look at techniques for finding the intrinsic parameters and for applying the transformation ${}^{ee}\boldsymbol{T}_b$ from the section about robotics (section 2.2) and the transformation ${}^{t}\boldsymbol{T}_C$ from the section about computer vision (section 2.4) to perform both intrinsic and extrinsic camera calibration, thus finding the transform ${}^{C}\boldsymbol{T}_b$ between the robot base $\boldsymbol{b}$ and the camera $\boldsymbol{C}$.

There are several possible ways to estimate the values in equation 2.7-2.9. Below, we give a brief overview of the methods used in this project.

2.5.1 Intrinsic Calibration

The intrinsic calibration used in this project is known as Zhang's method [37], and can be performed by the following process:

- **Collect calibration images**. Images are collected with a calibration board with known properties, like the ChArUco board, see section 2.3.2.
- **Detect corners.** The internal corner pixel coordinates of the calibration board are collected.
- **Transform to Camera coordinates.** Detected corners are then transformed into three-dimensional camera coordinates.
- **Parameter estimation.** The intrinsic parameters are then estimated with numerical techniques that match the pixels to camera coordinates.

2.5.2 Extrinsic Calibration

There are two cases of extrinsic calibration, eye-in-hand where we want find the camera $\boldsymbol{C}$ to end-effector $\boldsymbol{ee}$ transform ${}^{ee}\boldsymbol{T}_C$ and eye-to-hand where we find the camera $\boldsymbol{C}$ to robot

base b transform ${}^{b}T_{C}$. In both of the cases the goal is to find a fixed transform, that doesn't change whenever the robot moves, either in relation to the end-effector as in eye-in-hand or in relation to world as in eye-to-hand.

Eye-in-hand

- Known transforms
  - $A = {}^{ee}T_{b}$, base b to end-effector ee coordinate frame, found in section 2.2.3.
  - $B = {}^{C}T_{t}$, target t to camera C coordinate frame, found in section 2.4.4.
- Unknown transforms
  - $X = {}^{C}T_{ee}$, end-effector ee to camera C coordinate frame.
  - $Y = {}^{t}T_{b}$, base b to target t coordinate frame.

We can describe the system as $AX = YB$ which is shown in figure 2.13. The equation $AX = YB$ shows two paths from the base to the camera, either through the end-effector as in AX or through the target as in YB. The four transforms form a closed loop and solving for either X or Y will result in finding the subsequent one by combining the transforms in the right order. It is worth noting that swapping the origin and destination of all transforms is allowed, and will result in the resulting transform also changing the origin and destination.

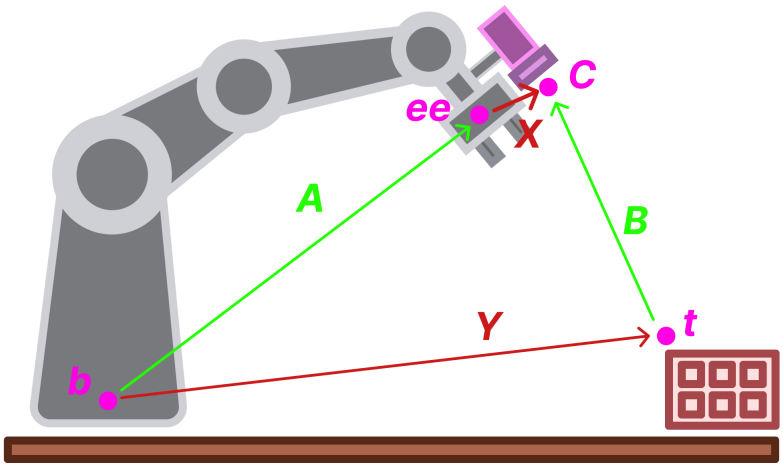


Figure 2.13: The eye-in-hand calibration problem setup of with known transforms A and B and the unknown X and Y.

To find X we can observe the arm in multiple poses as is shown in figure 2.14 and 2.15. We choose two different poses i and j and let the different poses of the end-effector between state i and j be the transform A_1 while the transform of the camera between the states is defined as B_1.

Assuming that the camera is screwed on tightly to the end-effector and the base of the robot and the target do not move, we can set up the following equations 2.13 - 2.24 from the corresponding figure 2.14

$$A_i X B_i = Y \tag{2.13}$$

$$A_j X B_j = Y \tag{2.14}$$

$$A_i X B_i = A_j X B_j \tag{2.15}$$

$$A_j^{-1} A_i X B_i = A_j X B_j B_i^{-1} \tag{2.16}$$

$$A_1 X = X B_1 \tag{2.17}$$

$$\dots$$

$$A_n X = X B_n \tag{2.18}$$

By collecting multiple transform pairs of A and B we can then create an equation system of $A_n X = X B_n$ and then solve for X to find the target calibration.

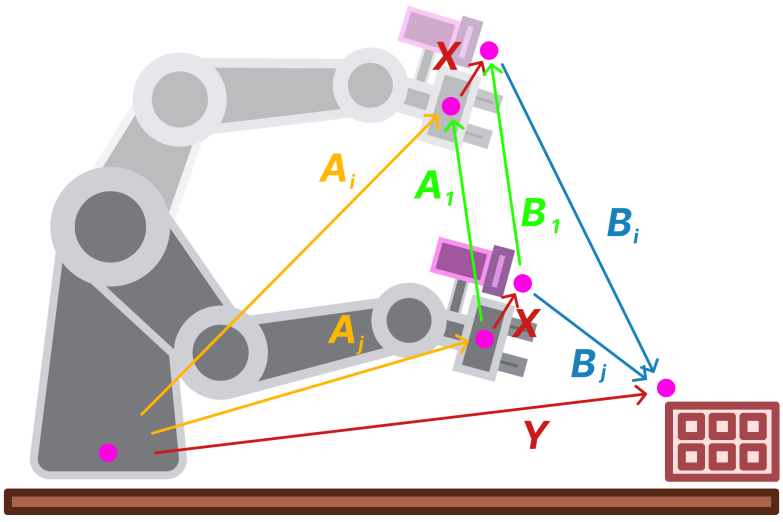


Figure 2.14: The robot arm observed in the two states i (faded) and j with the transforms required to solve for the eye-in-hand calibration.

Eye-to-hand

To solve for the eye-to-hand problem we can reorder the transformations to the ones in figure 2.15 and solve for Y instead and find the camera to base transform.

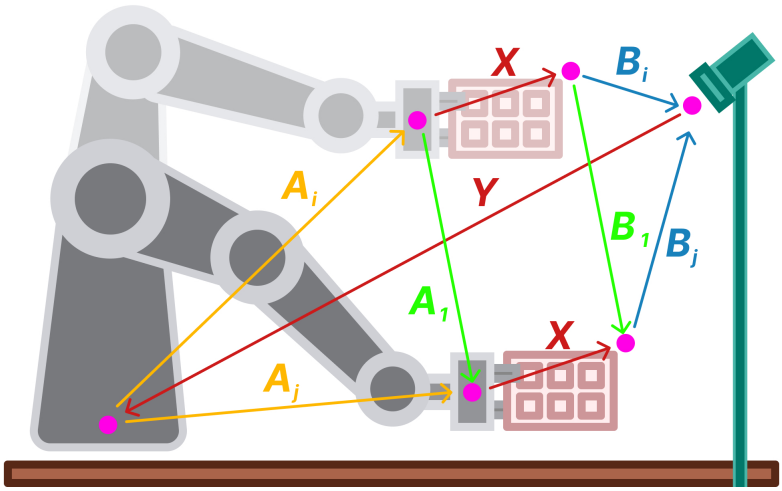


Figure 2.15: The robot arm observed in the two states i (faded) and j with the transforms required to solve for the eye-to-hand calibration.

$$B_i Y A_i = X \tag{2.19}$$

$$B_j Y A_j = X \tag{2.20}$$

$$B_i Y A_i = B_j Y A_j \tag{2.21}$$

$$B_j^{-1} B_i Y = Y A_j A_i^{-1} \tag{2.22}$$

$$B_1 Y = Y A_1 \tag{2.23}$$

$$\dots$$

$$B_n Y = Y A_n \tag{2.24}$$

Results in $BY = YA$ which is the same as solving $AX = XB$.

Solving AX=XB

For this problem, there exist two main categories of solutions. Separable solutions, where the rotation is estimated before the translation and simultaneous solutions, where they are estimated at the same time. In this project, three separable solutions, herein referred to as Tsai [32], Park[19] and Horaud [13], and two simultaneous solutions, herein referred to as Andreff [1] and Daniilidis[6], were used off the shelf via their OpenCV implementations [30].

2.6 Open CV

Open Computer Vision (Open CV) is an open-source framework that provides implementations of common computer vision, see section 2.4, and image analysis algorithms, see section 2.3, as well as support for ArUco markers, see section 2.3.1, and ChArUco boards, see section 2.3.2, and calibration, see section 2.5. OpenCV is supported on most platforms and is regularly updated [30].

2.7 Aligning the real environment to the simulated environment

Normally one would align the simulated environment with the real environment, instead of the other way around like we aim to do. The reason for doing so is to provide an option to move standardised simulated learning environments to reality. The standardised environments would make machine learning models easier to explore, benchmark and tune, before deploying them to reality.

Aligning a real-world workspace with a simulated environment is also a fundamental part of transferring knowledge from a reinforcement learning model to a physical robot. Having a good alignment eases the knowledge the model needs about error correction of the tasks being performed. The scope of aligning the set-ups includes having the actuators in the workspace positioned correctly in both the real set-up and the simulated environment.

2.7.1 Robot Learning Benchmark (RLBench)

RLBench is an open-source platform for facilitating and evaluating robot learning algorithms. It provides a diverse set of tasks, covering challenges like manipulation, perception and control. The platform offers a standardised and accessible framework for checking the performance and robustness of various learning techniques making it a valuable resource for researchers and practitioners [27].

Chapter 3

Method / Implementation

In order to align the real-world setup with the simulated environment, several calibration steps are required, see figure 3.1. In this section, we will outline each step and alternative approaches.

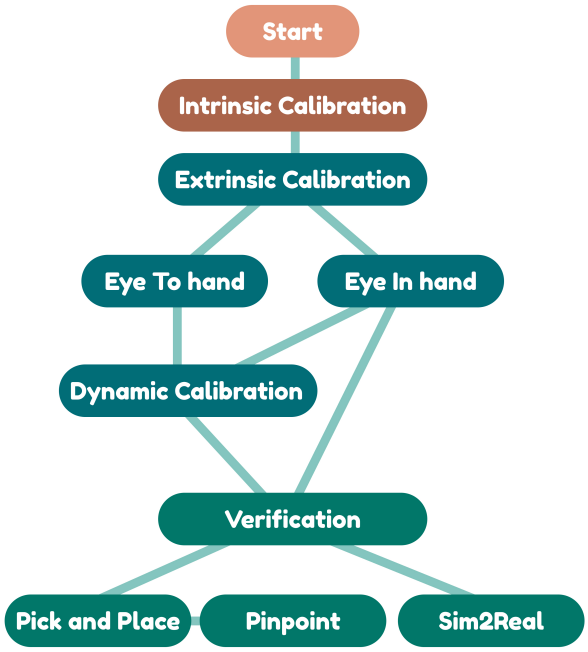


Figure 3.1: Figure shows the work process in a flowchart. The process consists of three parts. 1) Intrinsic calibration, 2) Extrinsic calibration and 3) Verification

3.1 Intrinsic Camera Calibration

When performing intrinsic calibration we utilise Zhang's method, see section 2.5.1, as it is implemented in OpenCV, see section 2.6. We do this by collecting several images containing a calibration target with known properties, see section 2.3.2, in varying angles and positions. Having calculated the camera matrix, see section 2.5.1, this is then stored along with the image resolution and a name as a JSON file [35] for later use in extrinsic calibration and verification steps.

3.2 Extrinsic Camera Calibration

3.2.1 Eye In hand

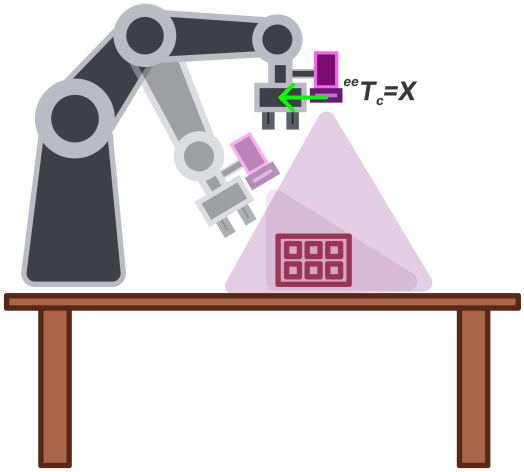


Figure 3.2: Figure shows the calibration target transform ${}^{ee}T_c = X$, the transform between the camera c and robot end-effector ee.

Eye-in-hand extrinsic calibration is performed by relying on the OpenCV implementations solving the $AX = XB$ matrix equation, see section 2.5.2. For eye-in-hand specifically, this is done by having a wrist-mounted camera on the robot move around a stationary calibration target, see figure 3.2. During this process, the world-to-end-effector transform along with the respective camera-to-calibration target transform are collected for each pose. These poses are then fed through the OpenCV implementations. The results of each implementation and the mean value thereof are stored for verification.

3.2.2 Eye To hand

AX=XB

The first method for performing eye-to-hand calibration relies on the OpenCV implementations solving the $AX = XB$ matrix equation, see section 2.5.2. A calibration target is fixed

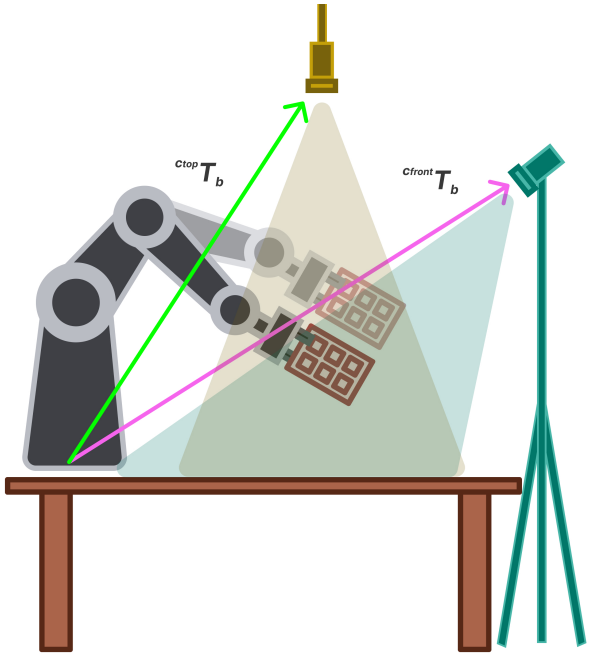


Figure 3.3: Figure shows the calibration target ${}^{c_{top}}\boldsymbol{T}_b$, the transform between the robot base $\boldsymbol{b}$ and the camera attached in the ceiling $\boldsymbol{c}_{top}$ and ${}^{c_{front}}\boldsymbol{T}_b$, the transform between the robot base $\boldsymbol{b}$ and the front camera $\boldsymbol{c}_{front}$.

to the robot arm, which proceeds to take several poses with the calibration target in view of a static external camera, see figure 3.3. As was the case in section 3.2.1, the results of each implementation listed in section 2.5.2 and the mean value thereof are saved and stored for verification.

Dynamic Calibration

We suggest a dynamic approach as an alternative method for eye-to-hand calibration. This requires a calibrated wrist-mounted camera, with which a calibration target is located in the world coordinate frame. Selfsame calibration target is then also located by the static external camera to be calibrated, thus allowing the location of the static camera to be found in the world frame via the TF-Tree, see section 2.2.3. This approach provides instant calibration results and ongoing feedback on any changes made to the camera's placement.

3.3 Verification

3.3.1 Pick and Place

To verify the extrinsic and intrinsic calibration, a pick-and-place implementation was applied. Coloured cubes are detected in the camera images, see section 2.3.3, and transferred to the world coordinate system by means of the camera matrix and the estimated pose of the

camera. One cube is then picked up and placed on top of the other. This is done with two different setups, one with and one without taking the cube measurements into account.

This provides a binary quality metric (success or failure), the granularity of which is dependent on the size of the object being picked and placed. Provided the extrinsic calibration estimate is the only source of error, being able to successfully pick and place an object in the correct positions would mean that the extrinsic calibration estimate is wrong by at most the size of the object being picked and placed.

3.3.2 ArUco Centre Pinpoint

This is an alternate method for verifying the estimates generated by camera calibration. Similar to the approach detailed in section 3.3.1, an object is identified in the camera's image and the robot arm is directed to move towards it. However, in this case, the object is an ArUco marker (see section 2.3.1), which eliminates the need for HSV detection to determine the target's position.

Once the centre point of the ArUco marker is determined by traversing the TF tree (see section 2.2.3), the robot arm is directed to move to those coordinates. The difference between the estimated centre point provided by the camera and the actual centre point can then be measured to calculate the error. This error can be used to manually adjust the calibration results. With the adjusted values, the verification process can be repeated until satisfactory results are obtained.

3.4 How to use the System

Below we have listed the intended steps to follow when using the system. The main tools used are the OpenCV library `cv2`, see section 2.6, and the ROS libraries `tf` (Transform) and MoveIt, see section 2.2.2. A full rundown can be found in the flowchart depicted in figure 3.4.

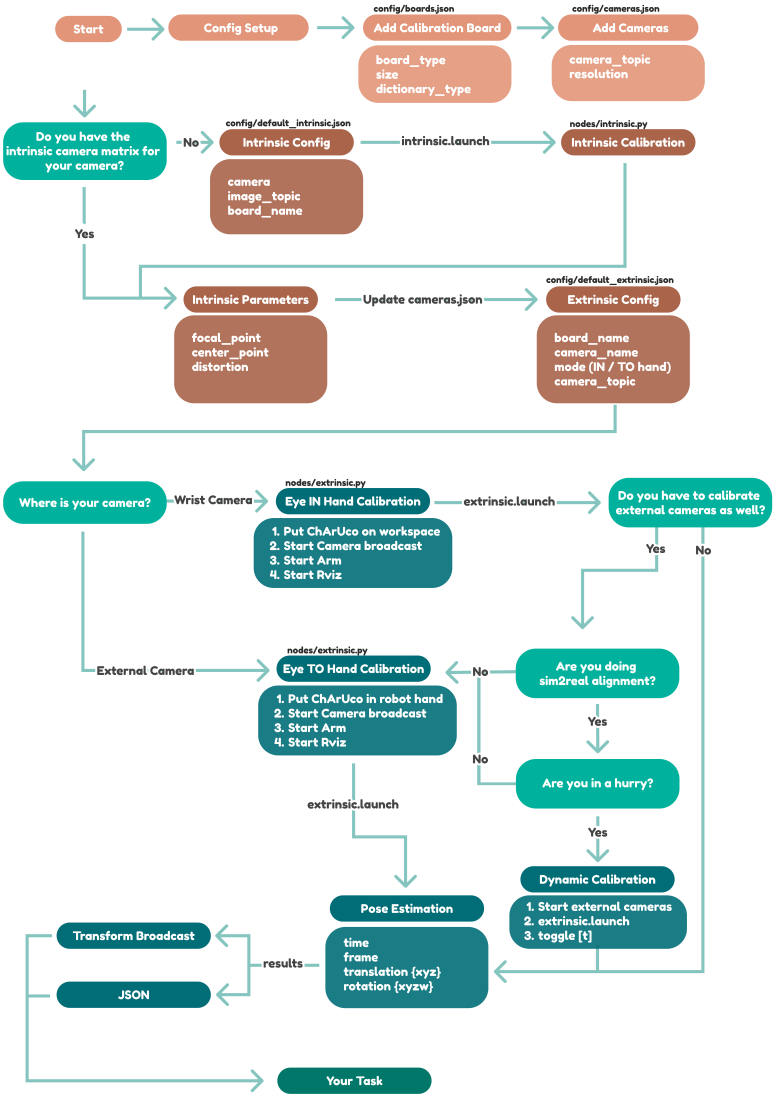


Figure 3.4: Figure shows a brief overview of the intended system usage, with the larger boxes under a topic describing essential parameters, outputs and steps for the topic.

3.4.1 Intrinsic Calibration

An overview of the intrinsic calibration implementation can be seen in figure 3.5. In essence, the method follows the steps explained below:

1. **Collect images.** Collecting n number of images of the calibration board, using the GUI.
2. **Intrinsic calibration.** The intrinsic calibration, see section 4.2, is done by using the method `cv2.calibrateCameraCharuco()`.
3. **Save parameters.** The resulting intrinsic matrix is saved in a JSON file.

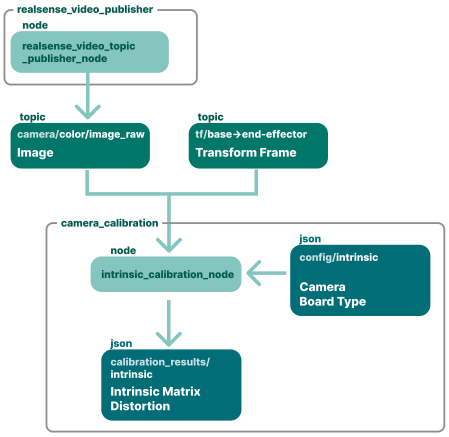


Figure 3.5: Figure shows an overview of the ROS structure of the intrinsic calibration. The grey boxes represent `rospackages`, the headline of each box is the source and the larger bottom text is the main content type

3.4.2 Extrinsic Calibration

An overview of the extrinsic calibration implementation structure can be seen in figure 3.6. The implementation of the extrinsic camera calibration can be broken down into six steps:

1. **Find Camera to Target Transform, ${}^{t}T_{C}$.**
2. (a) Finding pixel coordinates of ChArUco board , see section 2.3.2, using `cv2.detectMarkers()`
   (b) Transform pixel coordinate to 3D pose, see section 2.4.4, using `cv2.estimatePoseCharucoBoard()`
3. **Find Robot Base to End-Effector Transform, ${}^{ee}T_{b}$.** Find the base to end-effector transform, see section 2.2.3, using the TF Tree, see section 2.2.3, and `tf.lookUpTransform()`
4. **Collect transforms.** Collecting n number of sample transforms of the ${}^{t}T_{C}^{i}$ and ${}^{ee}T_{b}^{i}$ pair, using the GUI, with $n \geq 3$.

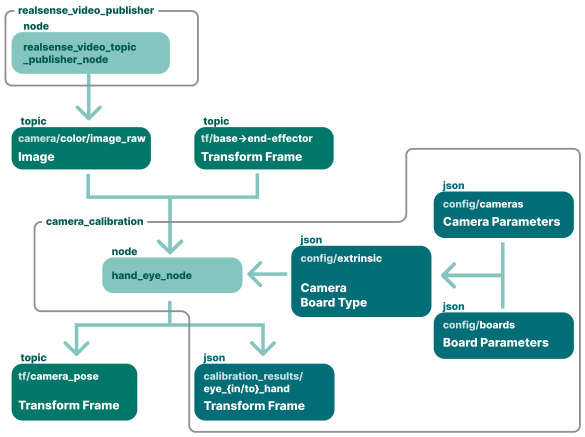


Figure 3.6: Figure shows an overview of the ROS structure of the extrinsic calibration. The grey boxes represent `rospackages`, the headline of each box the source and the larger bottom text the main content type

5. **Extrinsic Calibration, ${}^{C}T_{W}$.** The external camera calibration, see section 2.5.2, is performed with the n transforms of ${}^{t}T_{C}^{i}$ and ${}^{ee}T_{b}^{i}$ using `cv2.calibrateHandEye()` to get the transform ${}^{C}T_{W}$

6. **Publish and Save Result.** The resulting camera pose ${}^{C}T_{W}$ is then broadcasted as a transform frame, see section 2.2.3, onto the TF topic and saved in a JSON file.

7. **Verify results** using either pick & place or ArUco centre pinpoint verification.

3.4.3 Pick and Place Verification

An overview of the Pick and Place verification implementation can be seen in figure 3.7. Below is an outline for the workflow:

1. **Select a colour** by clicking on the object in the GUI and adjusting the selected colour segment with the sliders until a stable mask is produced. The HSV detection, see section 2.3.3, is implemented with `cv2` by masking the image with upper and lower values, fill and noise reduction functions. After that, the centre point of the segment is selected as the pixel coordinate $p = (u, v)$ for the object.

2. **Save pick-up** cube colour by pressing `[u]pp`.

3. **Redo step 1** for the intended placement cube.

4. **Save put-down** cube colour by pressing `[d]own`.

5. **Pixel to Camera coordinate**, the pixel point p is then converted to a camera coordinate P_C for both cubes by using equations 2.10-2.12, found in section 2.4.4. The value for z_c is gained by reading the depth camera value for p.

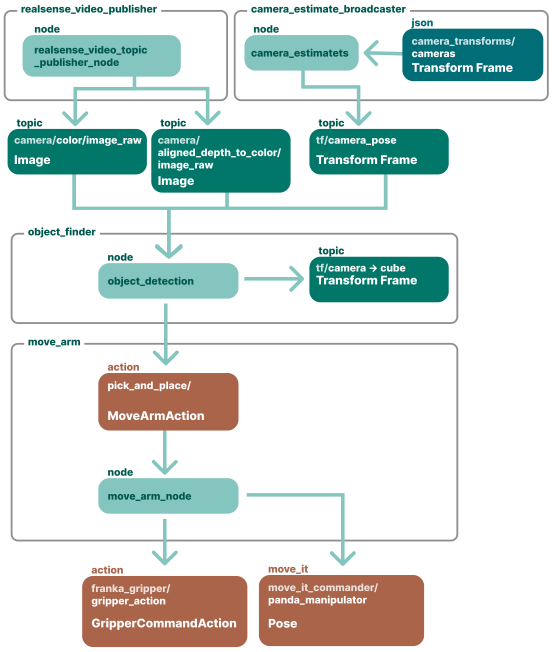


Figure 3.7: Figure shows an overview of the ROS structure of the Pick and Place implementation. The grey boxes represent `rospackages`, the headline of each box is the source and the larger bottom text is the main content type

6. **The pick and place**, is then performed by using inverse kinematics, see section 2.2.5, implemented by the `moveit` package. The arm is moved to the pick-up cube, grasps it, and moves to the put-down cube where it is released.

7. **Observe** whether or not the cube was correctly picked and placed.

3.4.4 ArUco Centre Pinpoint Verification

1. **Place** an ArUco marker well in view of the calibrated camera.

2. **Grip** a pointy object (e.g. a pen) with the robot gripper and measure the offset between the robot tool-centre-point and the tip. Supply this offset for the z-value when starting the `object_finder` node.

3. **Press** [a] to toggle ArUco mode.

4. **Optionally** check ${}^{ArUco}\boldsymbol{T}_{world}$ by pressing [w] to ensure the coordinate is safe.

5. **Move to coordinates** by pressing [m].

6. **Measure** pinpointed location offset from ArUco centre.

Chapter 4
Experiments

We apply our method to a concrete setup. We have RLBench, see section 2.7.1, as a simulator and since this is a benchmark, it is better to align the real setup to the simulation, than the simulation to the real setup. In this case, we want to align the camera streams, which in turn means accurate positioning of the external cameras and the wrist-mounted camera. In this section, we describe the setup and the experiments we conducted to provide answers to our research questions.

4.1 Experimental Setup

4.1.1 Hardware and Software

The following hardware and software specification was used throughout all experiments.

- **Hardware**
  - Franka Emika Panda Robot Arm
  - Cameras
    * Intel® RealSense™ Depth Camera D435
    * Intel® RealSense™ Depth Camera D455
  - ChArUco Boards
    * Large (9x14), square size 40mm, ArUco_size 31mm, DICT_5x5_1000
    * Medium (18x29), square size 10mm, ArUco_size 8mm, DICT_5x5_1000
    * Small (7x10), square size 12mm, ArUco_size 8mm, DICT_4x4_50
- **Software Infrastructure**
  - ROS

  - RViz
  - OpenCV 4.6
  - RLBench
  - Python 3.8

- **Python Libraries**
  - `pandas v2.0.1`
  - `numpy v1.24`
  - `opencv-contrib-python v4.6`
  - `seaborn v0.12.2`
  - `pyrealsense2`
  - `scipy v1.10.1`

4.1.2 Workspace

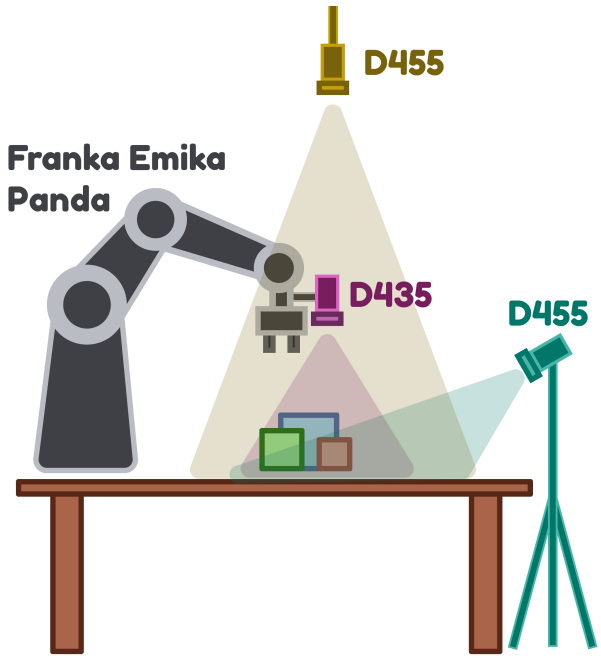


Figure 4.1: Figure shows a depiction of the workspace setup with the "RealSense" cameras, coloured cubes and the "Franka Emika Panda" robot arm.

The workspace consists of a Franka Emika Panda robotic arm installed on a table. A D435 camera is attached to the robot end effector, and two D455 cameras are situated directly in front of the camera at an elevated position and above the robot base mount. An illustration of the workspace is provided in figure 4.1 and a photograph in figure 4.2.

When conducting a pick-and-place verification session, differently coloured cubes are placed arbitrarily in view of the camera being tested.

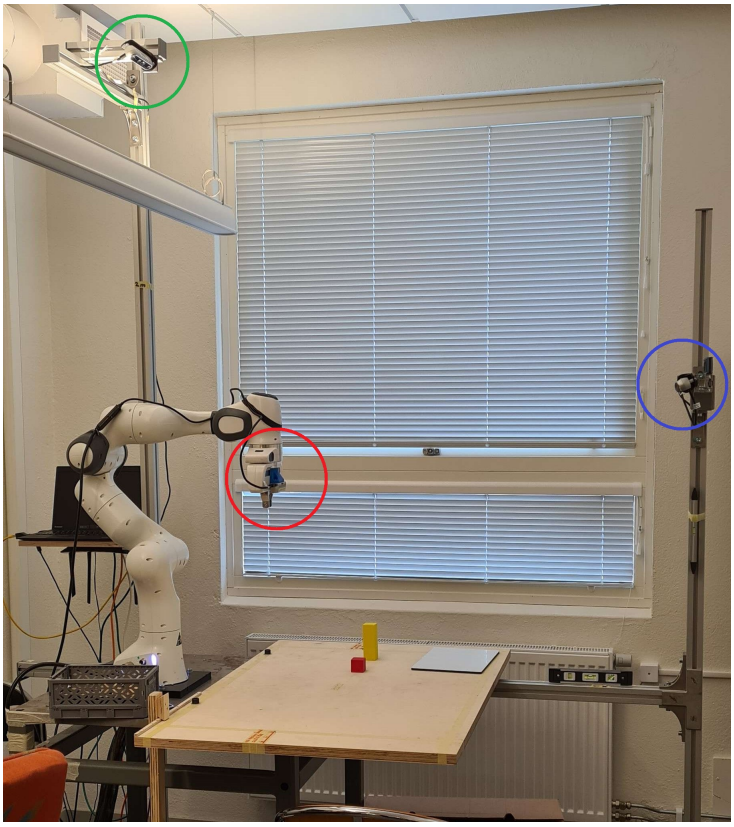


Figure 4.2: Figure shows the actual workspace setup with the "RealSense" cameras (Wrist, Front, Top), coloured cubes and the "Franka Emika Panda" robot arm.

4.2 Experiments: Intrinsic Calibration

Following the method in section 3.1, we conducted experiments where the following parameters were varied.

- Image Count
- Factory settings as an initial guess
- Distance to the calibration target

In an effort to provide a meaningful interpretation of the data and assess the quality of the obtained intrinsic calibration parameters, they were compared with the factory settings, see table 4.1. The intention is that examining how the computed parameters deviate from the factory settings will provide an indication of how a specific experimental parameter affects the calibration process.

Requirements for experiments:

- `roscore` running

- A publishing camera topic
- Large ChArUco board
- Wrist-mounted D435 camera

Table 4.1: The factory settings for the wrist mounted D435 camera.

Parameter	Value (pixels)
fx	616.68
fy	617.031
cx	314.182
cy	249.473
$k1$	0.0
$k2$	0.0
$p1$	0.0
$p2$	0.0
$k3$	0.0

4.2.1 Image Count and Initial Guess

The OpenCV implementation for intrinsic calibration allows for the utilisation of an initial guess for the camera matrix and distortion values. By collecting an image set consisting of 24 varied images we can use the same data set to investigate both the effect of image count and how the inclusion of an initial guess affects the calibration results. The values for intrinsic parameters provided by the manufacturer serve as the initial guess.

- **Steps:**

1. The intrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    intrinsic.launch
    config:=intrinsic_experiment
```

2. The calibration target was placed well in view of the camera, and an image was collected.
3. The relation between the calibration target and the camera was changed by moving the end effector.
4. The above step was repeated 24 times while ensuring that the calibration target appeared in different areas of the camera image and at different angles.
5. The calibration results for calibration while using and not using an initial guess were calculated and compared.

4.2.2 Distance to Target

In this experiment, the effect of distance to the calibration marker was investigated. A set of five images were captured for long (70cm) and short (40cm) distances respectively to see how it affects the resulting camera matrix.

- **Steps:**

  1. The intrinsic calibration node was launched like so:

     ```
     roslaunch camera_calibration
         intrinsic.launch
         config:=intrinsic_experiment
     ```

  2. The robot arm end effector was placed at a distance approximately 70 centimetres above the table, with the camera pointing straight down where it was kept static.
  3. The calibration target was placed well in view of the camera, and an image was collected.
  4. The calibration target was moved to a new pose and a new image was collected.
  5. The above step was repeated nine times while ensuring that the calibration target appeared in different areas of the camera image and at different angles.
  6. The above process was repeated with the height of the end effector adjusted to approximately 40 centimetres.
  7. The calibration results for high and low positions were calculated and compared.

4.3 Experiments: Eye In Hand Calibration

Following the methods in section 3.2.1, we conducted experiments where the following factors were investigated.

- Pose Count
- Static End Effector
- Sliding Window Mean of the transform ${}^{t}\boldsymbol{T}_{c}$
- Reproducibility

Requirements for experiments:

- `roscore` running
- A publishing camera topic
- Large ChArUco board
- Wrist-mounted D435 camera

4.3.1 Pose Count

In this experiment, we look at calibration results for a different number of sample sizes. This is done by collecting 20 transformations and comparing the calibration results for each new sample added.

We seek to investigate whether or not the calibration converges to a specific value over time, and how many poses such convergence would require.

- **Steps:**

  1. The extrinsic calibration node was launched like so:

     ```
     roslaunch camera_calibration
         extrinsic.launch
         config=cam_wrist
     ```

  2. The calibration target was placed on the workspace, well in view of the camera, and an image was collected.
  3. The arm with the wrist camera was moved to a new pose and a new image was collected.
  4. Above step was repeated 20 times while ensuring that the images covered different views of the calibration target and poses of the end-effector.

4.3.2 Static End Effector

In this experiment, we look at the difference between changing the translation of the wrist camera, while trying to keep the rotation fixed in comparison to changing the rotation of every collected transform. The resulting translation values are compared to a hand-measured baseline value as a reference value.

- **Steps:**

  1. The extrinsic calibration node was launched like so:

     ```
     roslaunch camera_calibration
         extrinsic.launch
         config=cam_wrist
     ```

  2. The calibration target was placed on the workspace, well in view of the camera, and an image was collected.
  3. The arm with the wrist camera was moved to a new pose, ensuring the robot end effector was not rotated, and a new image was collected.
  4. Above step was repeated 10 times while ensuring that the images covered different views of the calibration target.
  5. A set of 10 images was collected while ensuring variation in robot end effector rotation.

6. The target pose for the wrist-mounted camera was calculated for the set with and without rotation for comparison.

4.3.3 Sliding Window Mean

In this experiment, we investigate how the calibration result is affected by introducing additional stability to the $camera \rightarrow target$ transform by using the mean value of the last 30 pose estimations contra only using the latest one. The resulting translation values are compared to a hand-measured baseline value as a reference value.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_wrist_window_30


roslaunch camera_calibration
    extrinsic.launch
    config=cam_wrist_window_1
```

2. The calibration target was placed on the workspace, well in view of the camera, and an image was collected.
3. The arm with the wrist camera was moved to a new pose and a new image was collected while averaging the last 30 poses.
4. Above step was repeated 10 times while ensuring that the images covered different views of the calibration target and poses of the end-effector.
5. Another set was collected, using the same poses but without the averaging.
6. The target pose for the wrist-mounted camera was calculated with and without the smoothing effect and results were compared.

4.3.4 Reproducibility

In this experiment, we look at the stability of the result by reproducing the same calibration three times. This is done twice, once with identical robot poses and once with varied robot poses.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_wrist
```

2. The calibration target was placed on the workspace.
3. A set of five joint states was saved.
4. The arm was moved between each joint state, collecting an image at each state.
5. Calibration was performed, and the results were noted. The above step was repeated two more times.
6. A second set of calibrations is performed by moving the robot end effector by hand to each of the five states.

4.4 Experiments: Eye To Hand Calibration

Following the methods in section 3.2.2, we conducted experiments where the following parameters were investigated.

- Image count
- Position-Shift
- Sliding Window Mean of the transform ${}^{t}\boldsymbol{T}_{c}$

Requirements for experiments:

- `roscore` running
- A publishing camera topic
- Small ChArUco board attached to 57cm long stick
- Two D455 camera mounted in front of and above the robot arm

4.4.1 Pose Count

In this experiment, we look at calibration quality for a different number of sample sizes. This is done by collecting 20 transformations for the front camera and comparing the calibration results for each new sample added. For the top camera, the pose count is 10.

We seek to investigate whether or not the calibration converges to a specific value over time, and how many poses such convergence would require.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_front
```

2. The stick with the calibration target was gripped by the robot, well in view of the camera, and an image was collected.

3. The arm with the calibration target was moved to a new pose and a new image was collected.
4. Above step was repeated 20 times for the front camera and 13 for the top camera, while ensuring that the images covered different views of the calibration target and poses of the end-effector.
5. The target pose for the wrist-mounted camera was calculated for each sample size.

4.4.2 Position Shift

In this experiment, we investigate how a known shift in translation is reflected in the calibration results by moving the front camera 20 cm towards the robot base link (from 155 cm to 135 cm) and comparing the results for the two positions.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_front
```

2. The stick with the calibration target was gripped by the robot, well in view of the camera, and an image was collected.
3. The arm with the wrist camera was moved to a new pose and a new image was collected.
4. Above step was repeated 10 times while ensuring that the images covered different views of the calibration target and poses of the end-effector.
5. The front camera was moved 20cm closer to the robot base link, and the process repeated.
6. The target pose for the front camera was calibrated and compared for the two positions.

4.4.3 Sliding Window Mean

In this experiment, we investigate how the calibration result is affected by introducing additional stability to the $camera \rightarrow target$ transform by using the mean value of the last 30 pose estimations contra only using the latest one.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_front_window_30
```

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_front_window_1
```

2. The stick with the calibration target was gripped by the robot, well in view of the camera, and an image was collected.
3. The arm with the wrist camera was moved to a new pose and a new image was collected while averaging the last 30 poses.
4. Above step was repeated 10 times while ensuring that the images covered different views of the calibration target and poses of the end-effector.
5. Another set was collected, using the same poses but without the averaging.
6. The target pose for the wrist-mounted camera was calculated with and without the smoothing effect and results were compared.

4.5 Dynamic Calibration

4.5.1 Comparison

In this experiment, we look at how the dynamic calibration method compares to the standard eye-to-hand calibration, see section 4.4, with 10 images.

- **Steps:**

1. The extrinsic calibration node was launched like so:

```
roslaunch camera_calibration
    extrinsic.launch
    config=cam_front
```

2. The calibration target was placed tilted visible towards the front camera, in a place also visible to the wrist camera.
3. The estimated pose of the front camera was calculated.
4. The robot arm was moved to a new position while keeping the calibration target in view.
5. The above step was repeated five times, and an average was calculated.

4.6 Experiments: Verification

4.6.1 Pick and Place

In this experiment, we seek to verify the camera calibration by attempting to pick up a coloured cube and place it back down on top of another coloured cube 3.3.1. This is done in two versions, first without an offset for the cube measurements and once with an offset of half of the cube's side length.

- **Requirements for experiments:**
  - `roscore` running
  - A publishing camera topic
  - Colored cubes
  - Wrist mounted D435 camera/ Front or Top D455 Camera
  - Pose estimation from 4.3 for the separate cameras
  - Intrinsic matrix from 4.2 for the separate cameras
- **Steps:**
  1. The object node was launched like so:

     ```
     roslaunch object_finder hsv_cubes.launch
     ```

  2. The coloured cubes were placed randomly on the workspace, well in view of the camera.
  3. In the GUI a cube was selected for pick-up, ensuring that the cube was detected properly.
  4. The above step was repeated for the placement cube.
  5. Above step was repeated 10 times for the wrist camera and 5 times for the front and top cameras, with random positions of the cubes every time.
  6. Above steps were repeated with an offset of half of the cube's side length for the front and top cameras.

4.6.2 ArUco centre pinpoint

This experiment verifies camera calibration by having the robot grasp a sharp object in its tool-centre-point, thus aligned with the z-axis thereof, and moving the point of said object to the estimated centre of a printed ArUco marker, see section 3.3.2. The error provided is then used to adjust the camera estimates, which are then reevaluated.

- **Requirements:**
  - `roscore` running
  - A publishing camera topic
  - An ArUco marker
  - Wrist mounted D435 camera / Front D455 Camera / Top D455 Camera
  - An intrinsic and extrinsic estimate for the camera to be verified
- **Steps:**
  1. The object node was launched like so:

     ```
     roslaunch object_finder hsv_cubes.launch
     ```

  2. The ArUco marker is placed in view of the camera.
  3. ArUco detection mode was entered.
  4. ArUco centre point estimate was examined.
  5. Estimate was deemed to be a safe coordinate, and the arm was moved to said point.
  6. Adjustments were made based on the error.

Chapter 5
Results

Herein we present the results of our experiments. The results are visualised in graphs and tables, with further more in-depth discussion following in chapter 6.

5.1 Intrinsic Calibration

5.1.1 Image Count and Initial Guess

In this section, we display graphs portraying how the calibrated intrinsic and distortion parameters, see section 2.4.2, vary over a set of 24 images. Each figure contains three curves, one using no initial guess for the parameters, one using the factory settings as an initial guess and lastly the factory settings themselves. The factory settings serve as our ground truth, but what we seek to investigate is how the curves change over the image set and how using and not using an initial guess affects the calibration. The results were produced by following the steps laid out in section 4.2.1.

After only five images the calibrated values when using an initial guess align with those not using an initial guess. This indicates that using or not using an initial guess is a non-factor as long as the image set is not very small.

Figures 5.1 and 5.2 show the focal length along the x and y axes respectively. In both cases, the calibrated values quickly, after five images, align with each other and plateau.

Figures 5.3 and 5.4 show the image centre point along the x and y axes respectively. We see a similar pattern to that seen in 5.1 and 5.2, with a plateau forming between the five and ten image count.

The distortion parameters, figures 5.5 through 5.9, also follow this trend of guess contra no guess not mattering past the first five images. We do see an interesting point around images 18-19, where a shift or spike can be observed. The most likely reasons for this spike are noise in the data or a previously unseen view of the calibration target. This far into the image set the previous is more likely than the latter.

This tendency to plateau between five and ten images indicates that there is no need to collect an excessive number of images and that 10 to 20 should suffice.

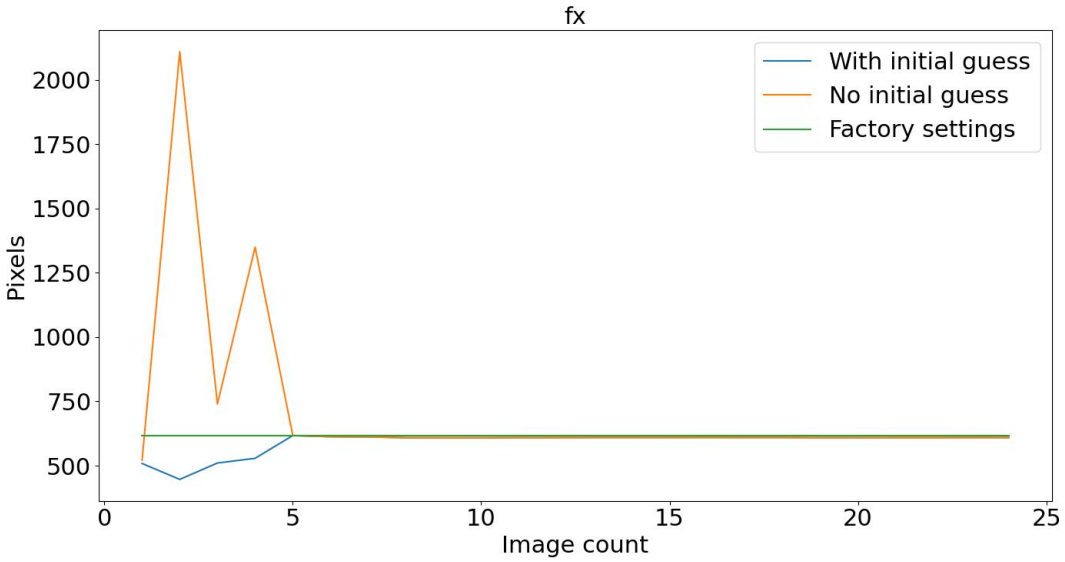


Figure 5.1: Calibration results for the intrinsic parameter fx.

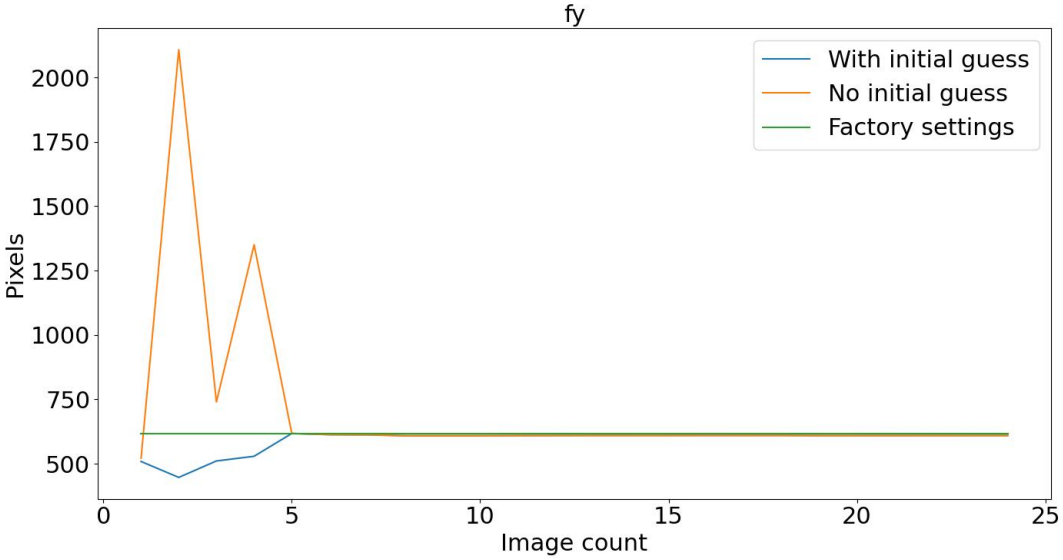


Figure 5.2: Calibration results for the intrinsic parameter fy.

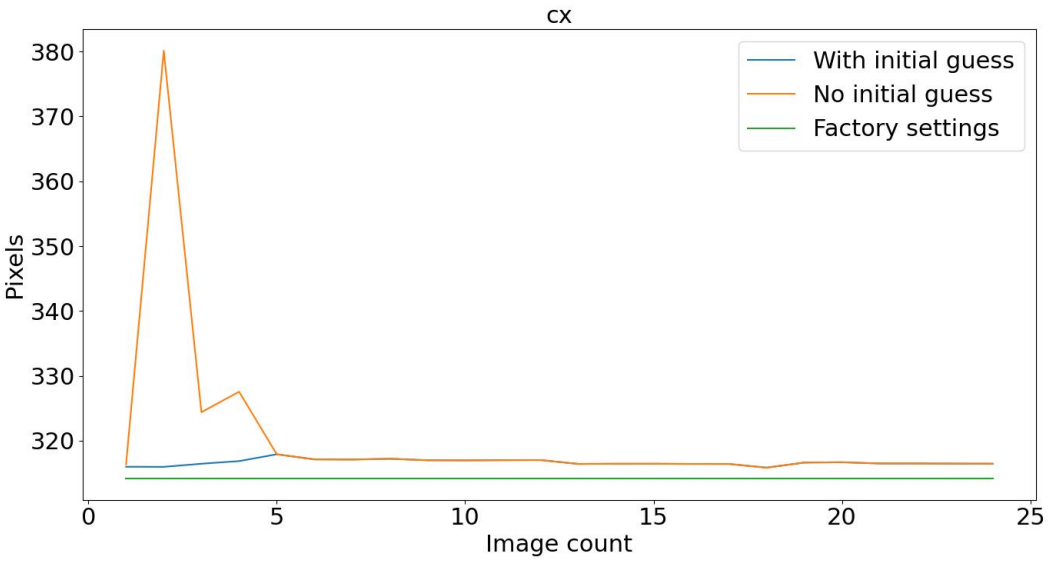


Figure 5.3: Calibration results for the intrinsic parameter cx.

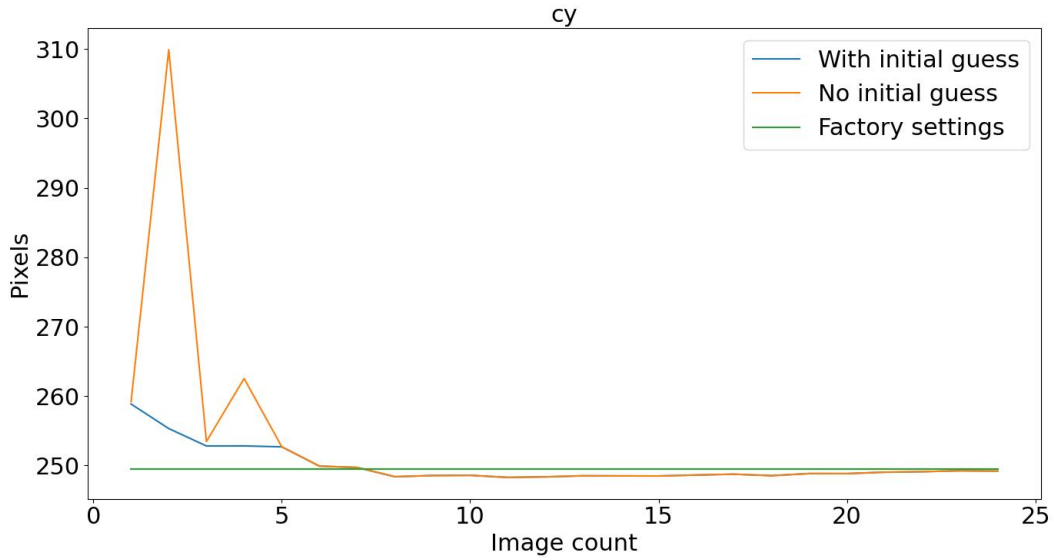


Figure 5.4: Calibration results for the intrinsic parameter cy.

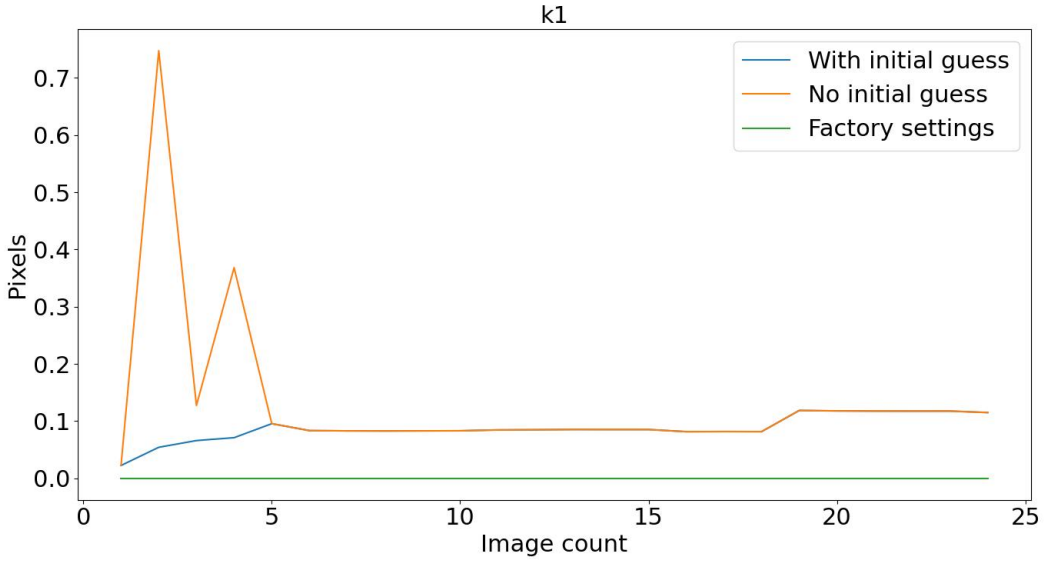


Figure 5.5: Calibration results for the distortion parameter $k1$.

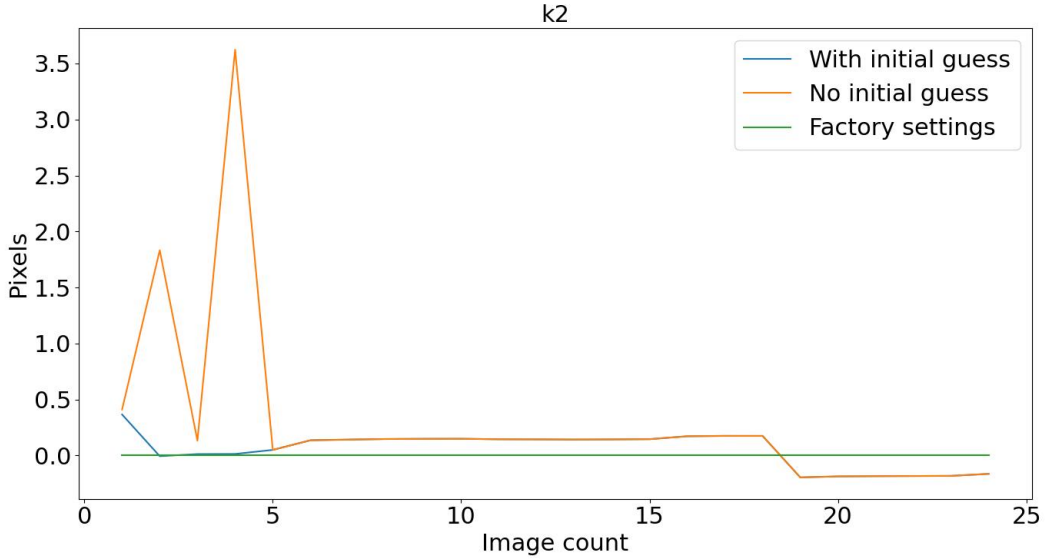


Figure 5.6: Calibration results for the distortion parameter $k2$.

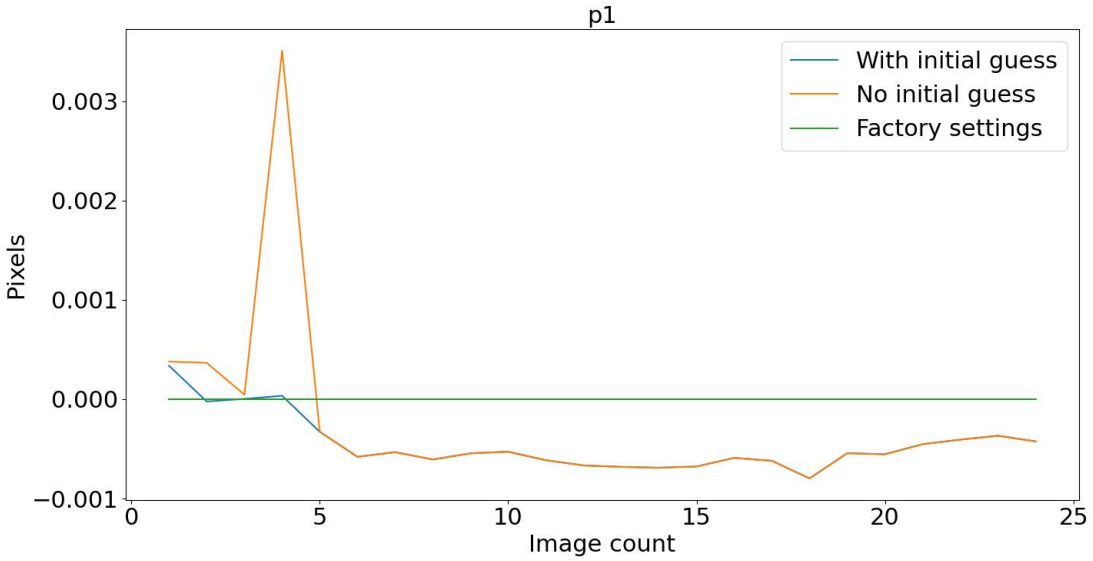


Figure 5.7: Calibration results for the distortion parameter $p1$.

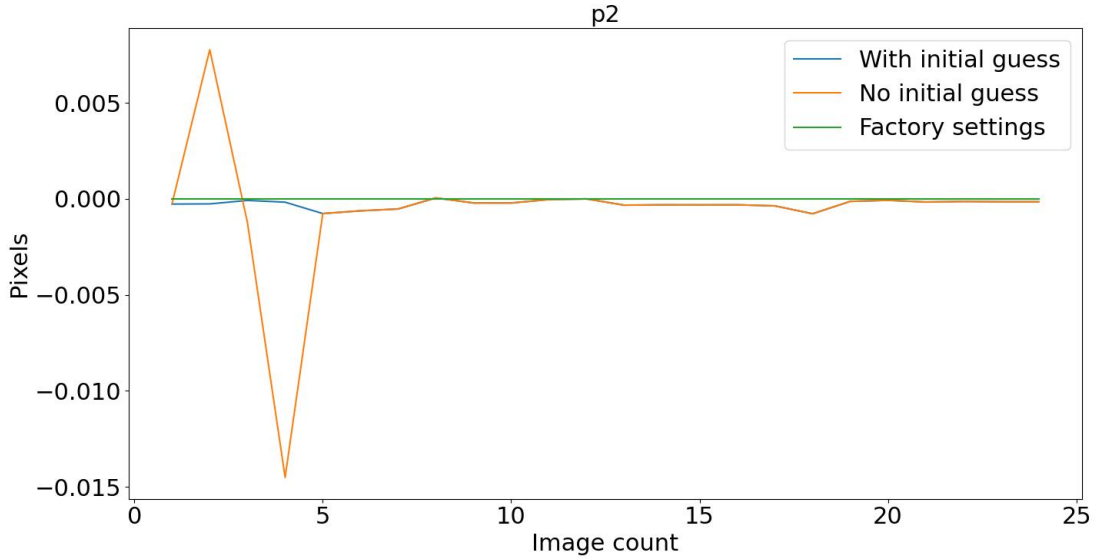


Figure 5.8: Calibration results for the distortion parameter $p2$.

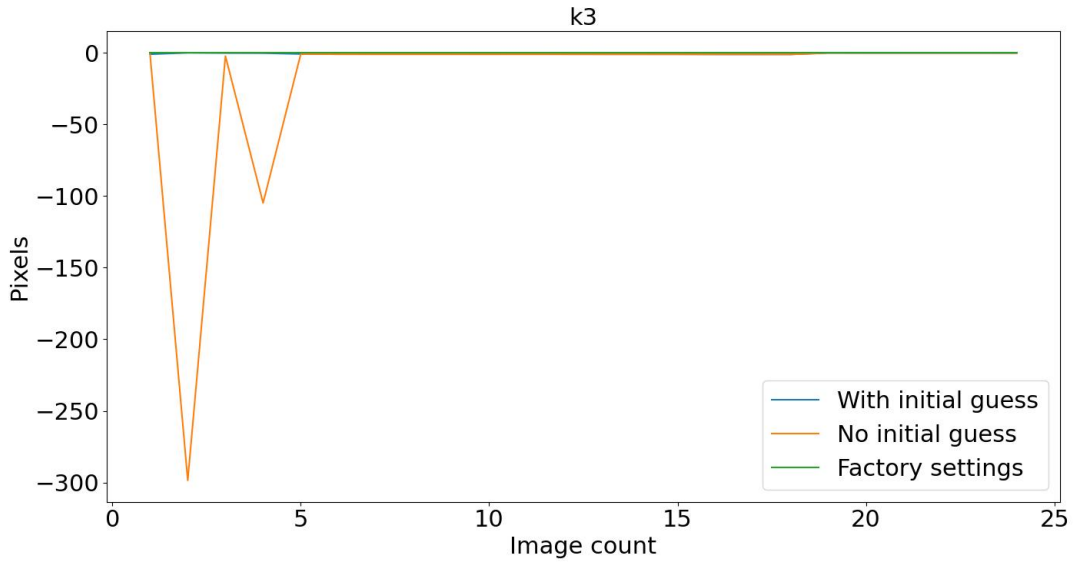


Figure 5.9: Calibration results for the distortion parameter $k3$.

5.1.2 Distance to Target

Figures 5.10-5.18 showcase how the calibrated intrinsic and distortion parameters vary over a set of ten images for three different image sets. The first image set is taken at a fixed distance of 70 centimetres from the calibration target, the second at 40 centimetres and the third is randomly merged from the two previous sets. The details of this acquisition can be seen in section 4.2.2. The factory settings for the parameters are included in each graph to serve as our ground truth.

Figures 5.10-5.13 show the set of images taken from 70 centimetres trending closer to the factory settings than those in the 40-centimetre set. However, in figures 5.10 and 5.11 the third set, featuring a random subset of near and far images, trends closer still. Provided the factory settings are correct, this indicates that using images taken too close to the calibration target is ill-advised.

The distortion parameter graphs, figures 5.14-5.18 are far more mixed, meaning that it is difficult to draw a definite conclusion from the data. It is noteworthy that the combined line trends closer to the "far away" line for the parameters $k1$, $k2$ and $k3$, despite being created from an equal number of near and far away images.

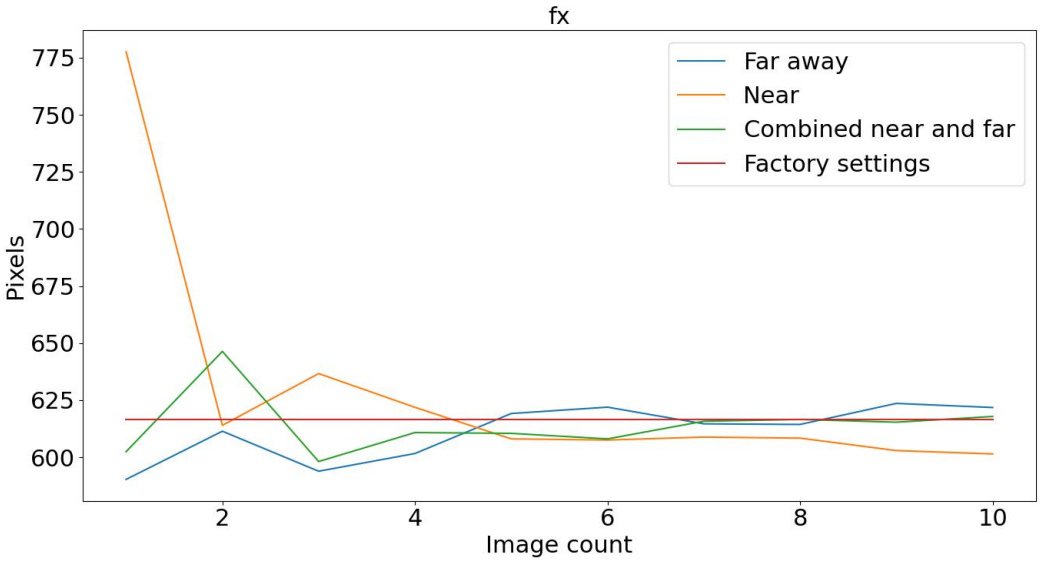


Figure 5.10: Calibration results for the intrinsic parameter fx.

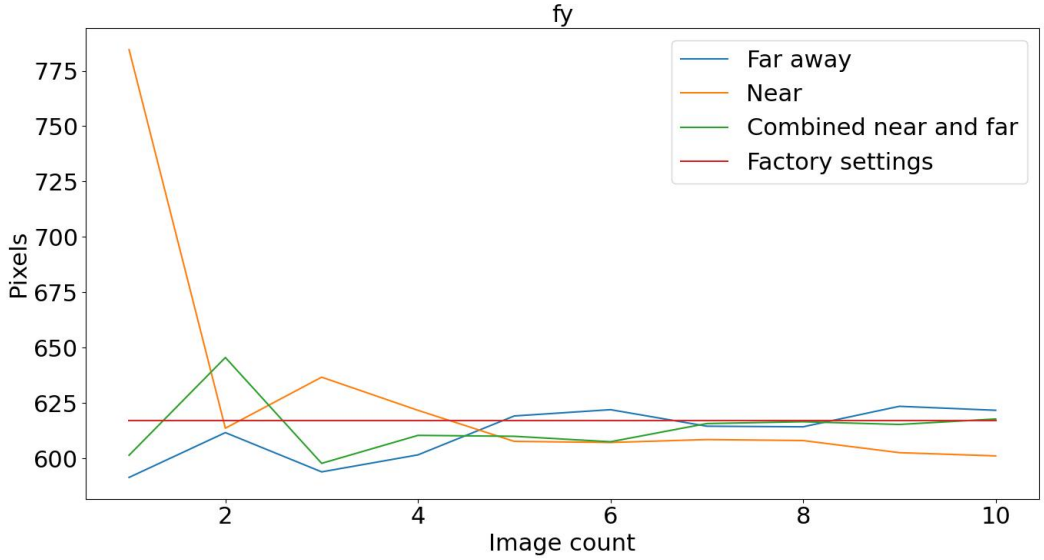


Figure 5.11: Calibration results for the intrinsic parameter fy.

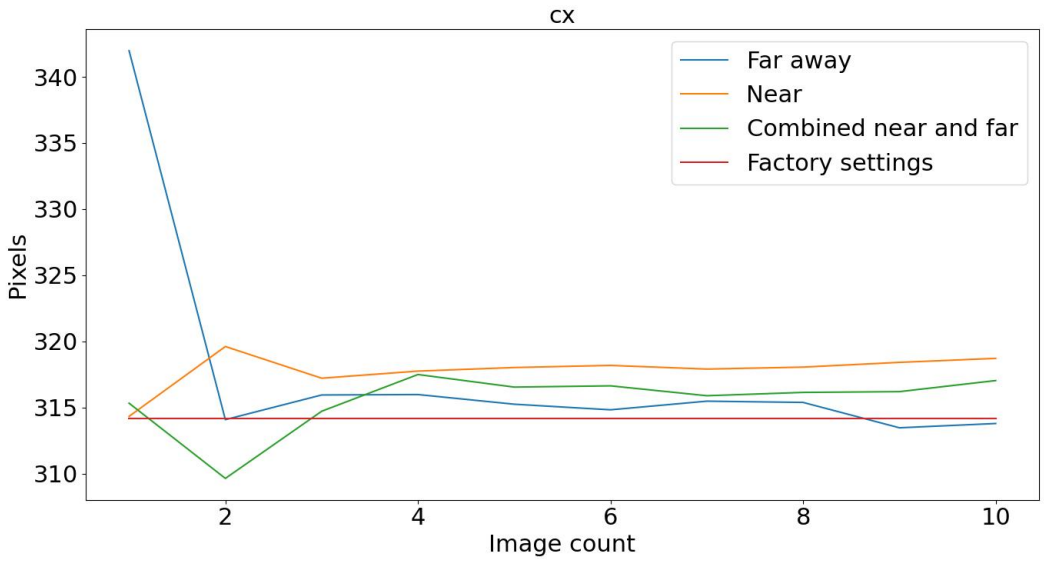


Figure 5.12: Calibration results for the intrinsic parameter cx.

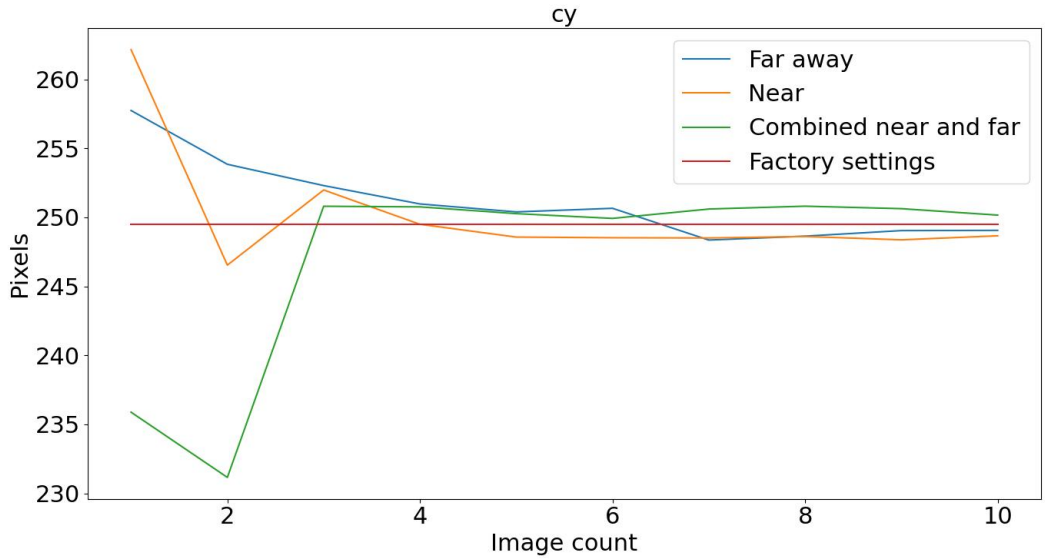


Figure 5.13: Calibration results for the intrinsic parameter cy.

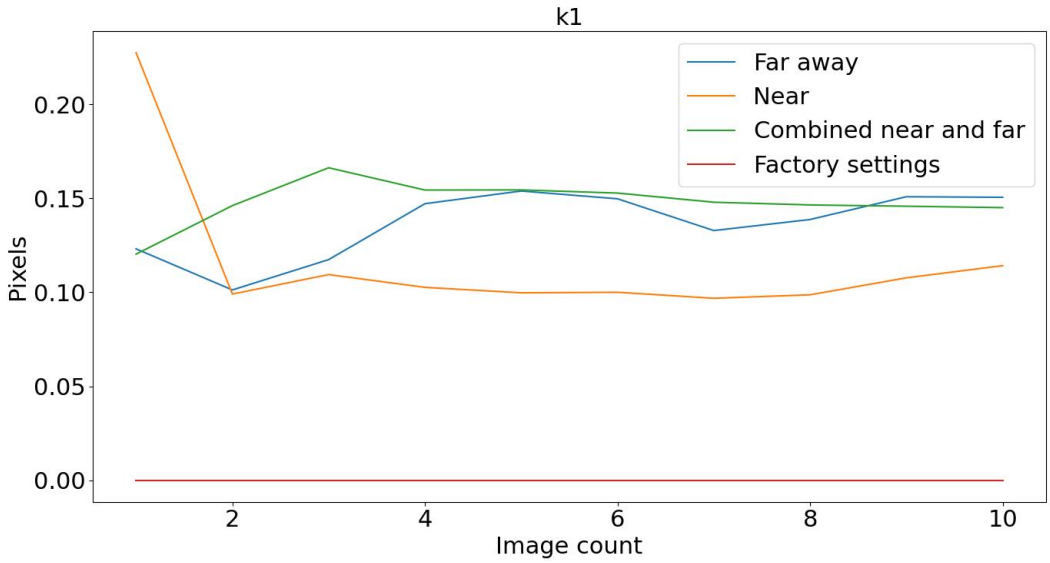


Figure 5.14: Calibration results for the distortion parameter $k1$.

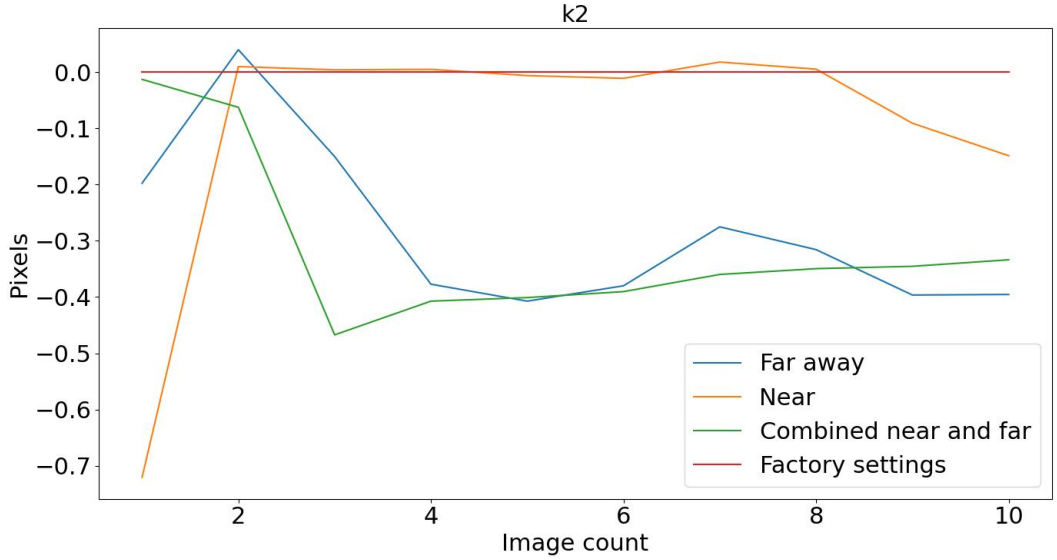


Figure 5.15: Calibration results for the distortion parameter $k2$.

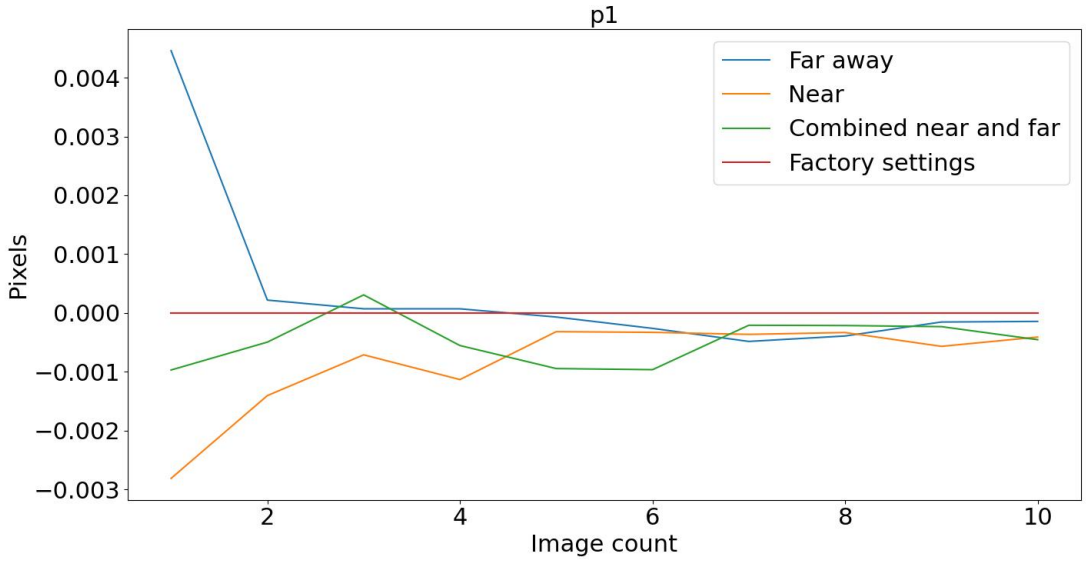


Figure 5.16: Calibration results for the distortion parameter $p1$.

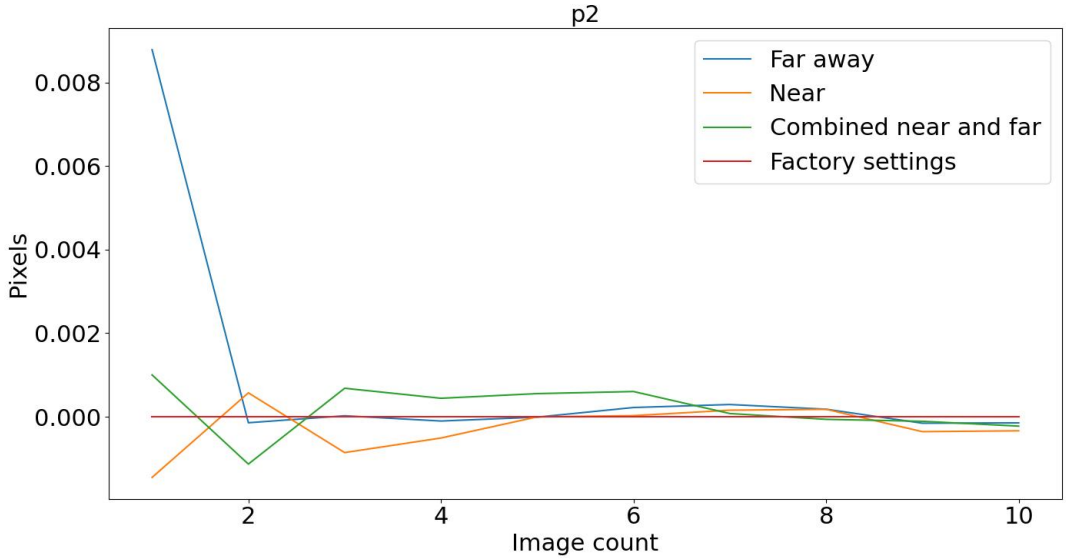


Figure 5.17: Calibration results for the distortion parameter $p2$.

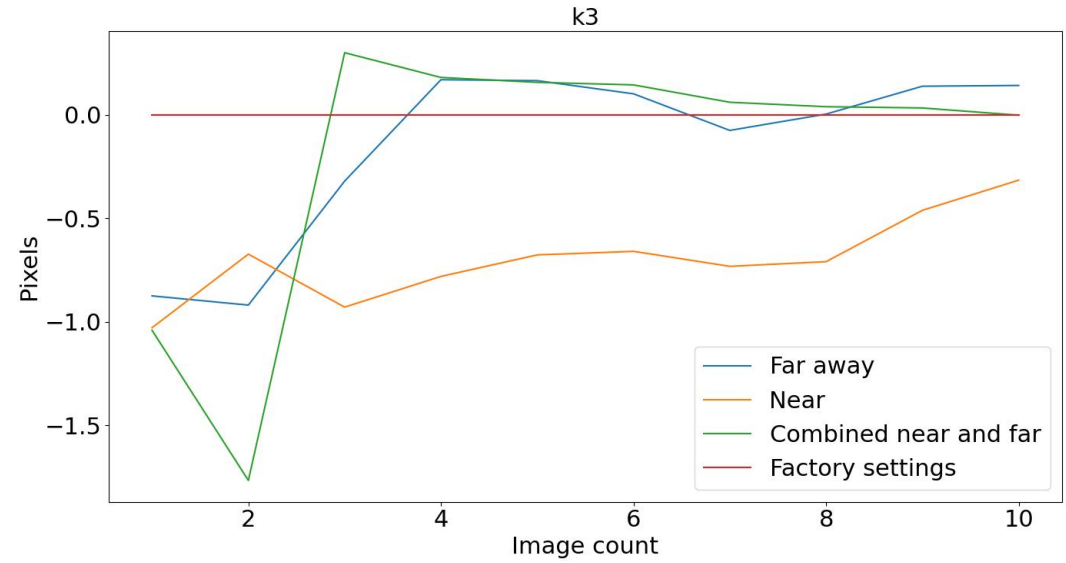


Figure 5.18: Calibration results for the distortion parameter $k3$.

5.2 Extrinsic Calibration

5.2.1 Eye in Hand

Static End Effector

In tables 5.1 and 5.2 we can see how varying contra not varying the rotation of the robot end effector affects the resulting calibration when performing eye-in-hand calibration. The calibrated values are compared to hand-measured values which serve to provide a baseline. In both cases, the rotation values are closely aligned with the baseline. With regard to translation, the difference is more pronounced. Table 5.1 shows that only moving the end effector in a plane causes large shifts from the baseline along both the x and y axes. These shifts are not seen when varying the rotation, see table 5.2, indicating that this is an important factor for accurate results.

Table 5.1: Extrinsic calibration results when not varying the rotation of the end effector, with δ being the absolute distance between calibration results and the hand-measured baseline.

Version	Translation (m)		Rotation (quaternions)	
Hand measured	x	0.035	r_x	0
	y	-0.04	r_y	0
	z	0.07	r_z	0.7071068
			r_w	0.7071068
Along plane	x	-0.18909	r_x	-0.013466
	y	-0.090834	r_y	0.029904
	z	0.065904	r_z	0.69595
			r_w	0.71734
δ	x	0.22409	r_x	0.013466
	y	0.050834	r_y	0.029904
	z	0.0040960	r_z	0.011157
			r_w	0.010233

Table 5.2: Extrinsic calibration results when varying the rotation of the end effector, with δ being the absolute distance between calibration results and the hand-measured baseline.

Version	Translation (m)		Rotation (quaternions)	
Hand measured	x	0.035	r_x	0
	y	-0.04	r_y	0
			r_z	0.7071068
	z	0.07	r_w	0.7071068
Varied rotation	x	0.031839	r_x	0.0046064
	y	-0.036249	r_y	0.0057965
			r_z	0.69465
	z	0.063206	r_w	0.71931
δ	x	0.0031610	r_x	0.0046064
	y	0.0037510	r_y	0.0057965
			r_z	0.012457
	z	0.0067940	r_w	0.012203

Repeat Calibrations

In tables 5.3 and 5.4 we see the results of a set of three calibrations using the same poses, in order to investigate consistency. The poses used in table 5.3 have identical joint states across all three calibration series. We can see that using such identical joint states yields results which only vary at a sub-millimetre level. The poses used in table 5.4 do not have identical joint states, but rather the robot end effector was moved by hand to positions and rotations which sought to mimic those used in table 5.3. The results thereof show a variance at a millimetre level, meaning that it stands to reason that when given similar input data the system will yield deterministic results.

Table 5.3: Results for repeat extrinsic calibrations using identical poses, with σ being the standard deviation of calibration results.

Calibration number	Translation (m)		Rotation (quaternions)	
1	x	0.029693	r_x	0.010709
	y	-0.037034	r_y	0.0065737
			r_z	0.69301
	z	0.062277	r_w	0.72082

Table 5.3: Results for repeat extrinsic calibrations using identical poses, with σ being the standard deviation of calibration results. (Continued)

2	x	0.029649	r_x	0.010698
	y	-0.037073	r_y	0.0066157
	z	0.062266	r_z	0.69293
			r_w	0.72090
3	x	0.029468	r_x	0.010919
	y	-0.036932	r_y	0.0066677
	z	0.062233	r_z	0.69303
			r_w	0.72080
σ	x	0.000097366	r_x	0.00010169
	y	0.000059447	r_y	0.000038447
	z	0.000018696	r_z	0.000043205
			r_w	0.000043205

Table 5.4: Results for repeat extrinsic calibrations using non-identical poses, with σ being the standard deviation of calibration results.

Calibration number	Translation (m)		Rotation (quaternions)	
1	x	0.030732	r_x	0.0088218
	y	-0.034494	r_y	0.0028619
	z	0.064493	r_z	0.69391
			r_w	0.72000
2	x	0.032384	r_x	0.011220
	y	-0.031342	r_y	0.0044303
	z	0.067536	r_z	0.69417
			r_w	0.71971
3	x	0.031646	r_x	0.0093004
	y	-0.035569	r_y	0.0070461
	z	0.065195	r_z	0.69434
			r_w	0.71955
σ	x	0.00067570	r_x	0.0010363
	y	0.0017937	r_y	0.0017259
	z	0.0013009	r_z	0.00017682
			r_w	0.00018624

Sliding Window Mean of the camera to target transform

The results found in table 5.5 reflect the effect of smoothening the $camera \rightarrow target$ transform. This is done by using a sliding window to calculate the mean value thereof, contra relying on the newest singular value. The resulting difference between the two approaches is very small, indicating that this is a non-factor and that the transformation between the camera and ChArUco board can be considered stable.

Table 5.5: Extrinsic calibration results with and without a sliding window mean, with δ being the absolute distance between calibration results.

Window Size	Translation (m)		Rotation (quaternions)	
1	x	0.029706	r_x	0.0073148
	y	-0.035097	r_y	0.0055223
	z	0.065835	r_z	0.69278
			r_w	0.72109
30	x	0.029673	r_x	0.0073256
	y	-0.035111	r_y	0.0056041
	z	0.065808	r_z	0.69276
			r_w	0.72111
δ	x	0.000033	r_x	0.0000108
	y	0.000014	r_y	0.0000818
	z	0.000027	r_z	0.00002
			r_w	0.00002

Pose Count

Figures 5.19-5.25 show how the wrist camera extrinsic calibration results vary over a set of 20 poses for each of the five calibration methods used.

The general tendency is for the first three poses to produce a very scattered result, which quickly converges and plateaus, at a centimetre level, within five to ten poses. However, at a millimetre level, there is a continued shift throughout the series of poses. This means that while it is highly recommended to use more than the required three poses, using a large number of poses seemingly does not provide a singular outcome. Rather it will continue to shift slightly with each new pose added.

Several of the graphs, figures 5.19, 5.20 and 5.23, show a sudden shift between poses 17 and 18, with smaller shifts between poses 18 and 20. Though not certain, this sudden shift so late in the series could be explained by noise in the data. Such noise is usually caused by a pose where the transform between the ChArUco board and the camera was not properly calculated.

Figure 5.21 shows the method dubbed "Andreff" producing very different results than the other four methods for the z translation during the first 15 poses. This is not an uncommon occurrence and is not specific to this method. We do not have a verified explanation for this but with the information gained regarding reproducibility in tables 5.3 and 5.4 we can say that

the outcomes are deterministic. This means that this specific data is seemingly considered noisy with regards to "Andreff" when calculating the z translation.

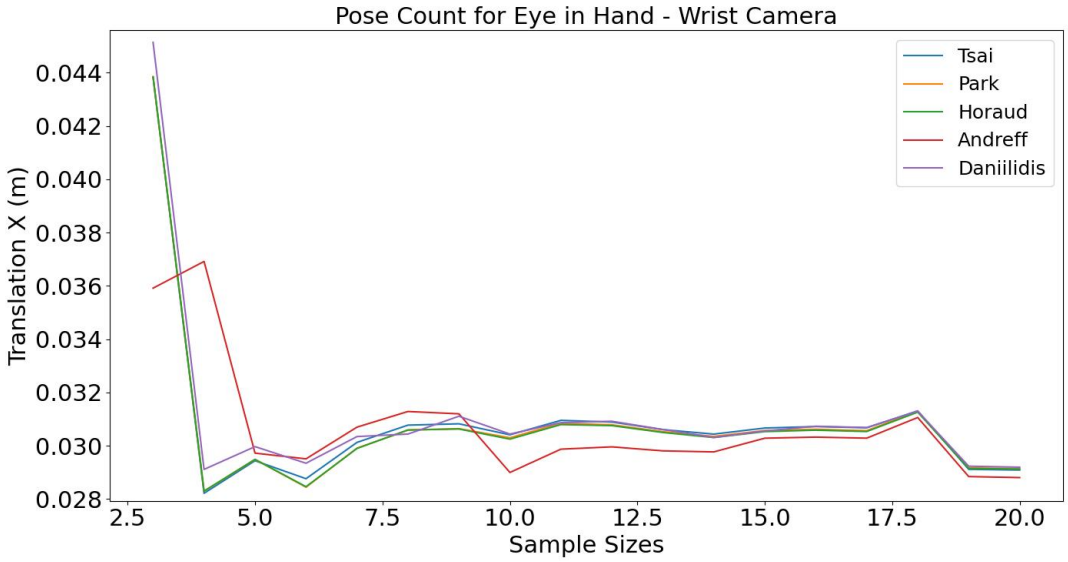


Figure 5.19: The estimated X-translation for *ee* → *camera*.

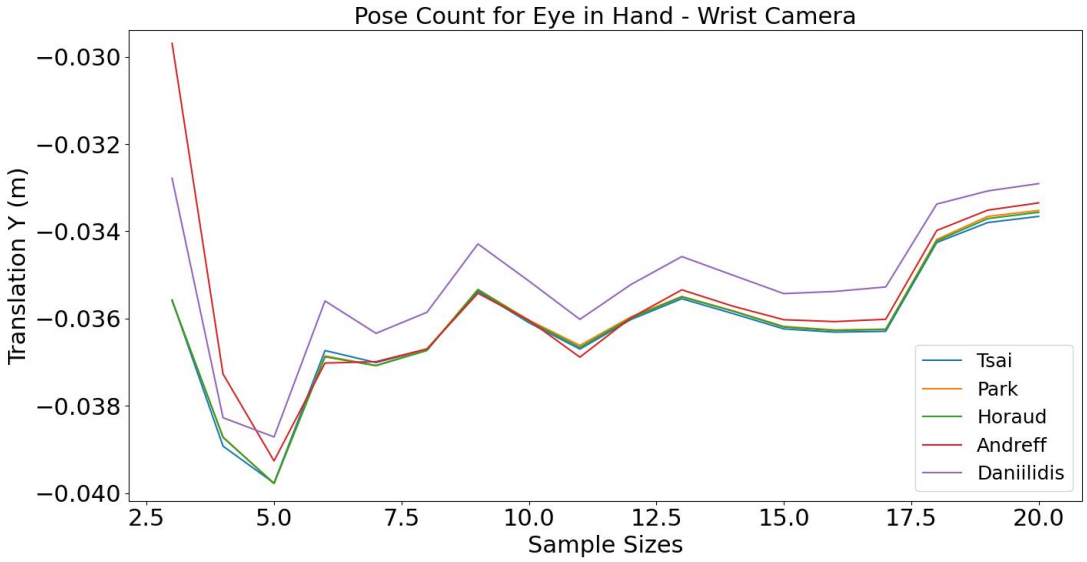


Figure 5.20: The estimated Y-translation for *ee* → *camera*.

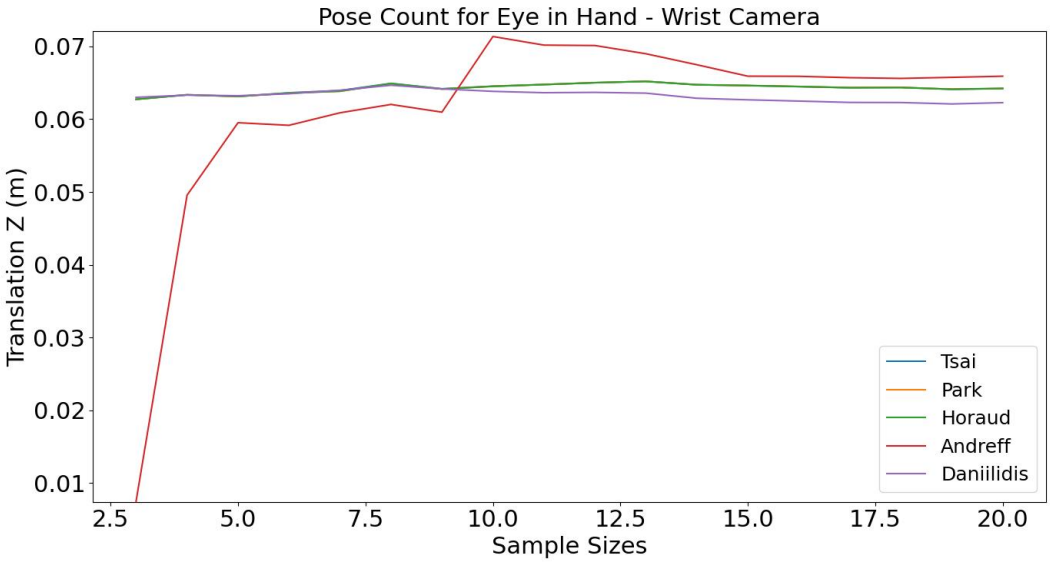


Figure 5.21: The estimated Z-translation for *ee → camera*.

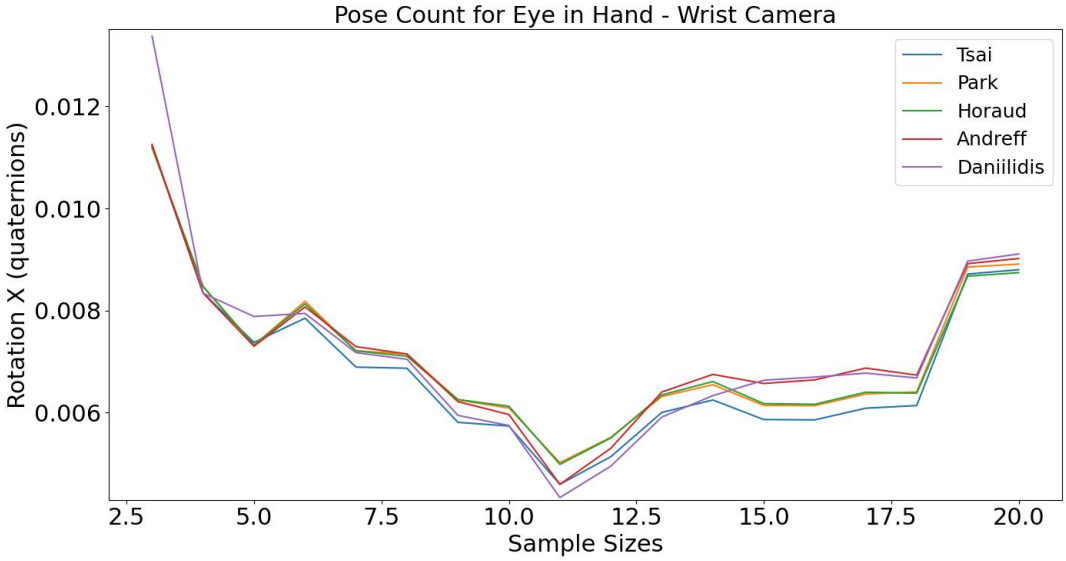


Figure 5.22: The estimated X-rotation for *ee → camera*.

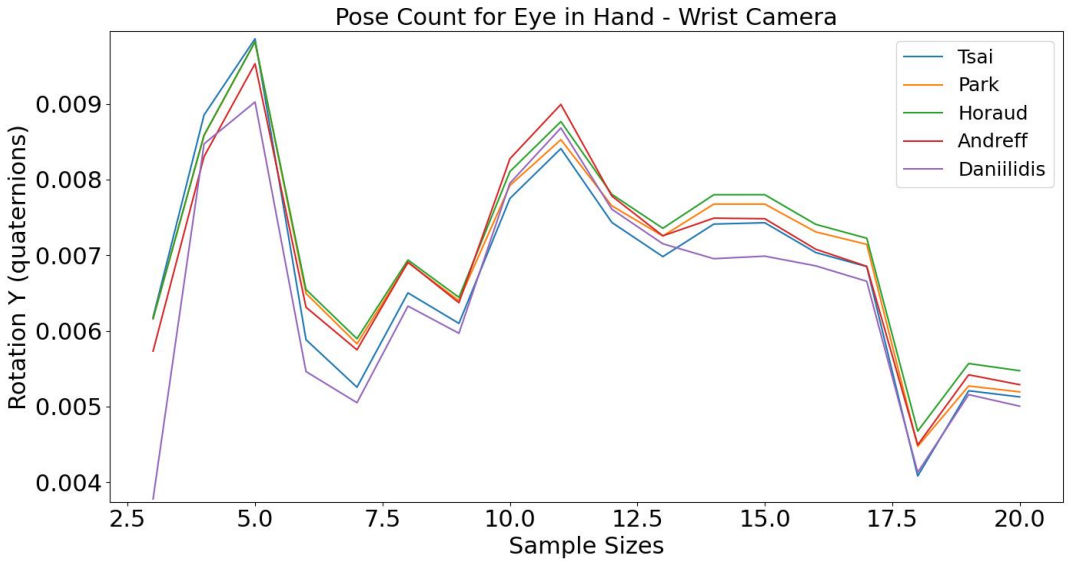


Figure 5.23: The estimated Y-rotation for $ee \rightarrow camera$.

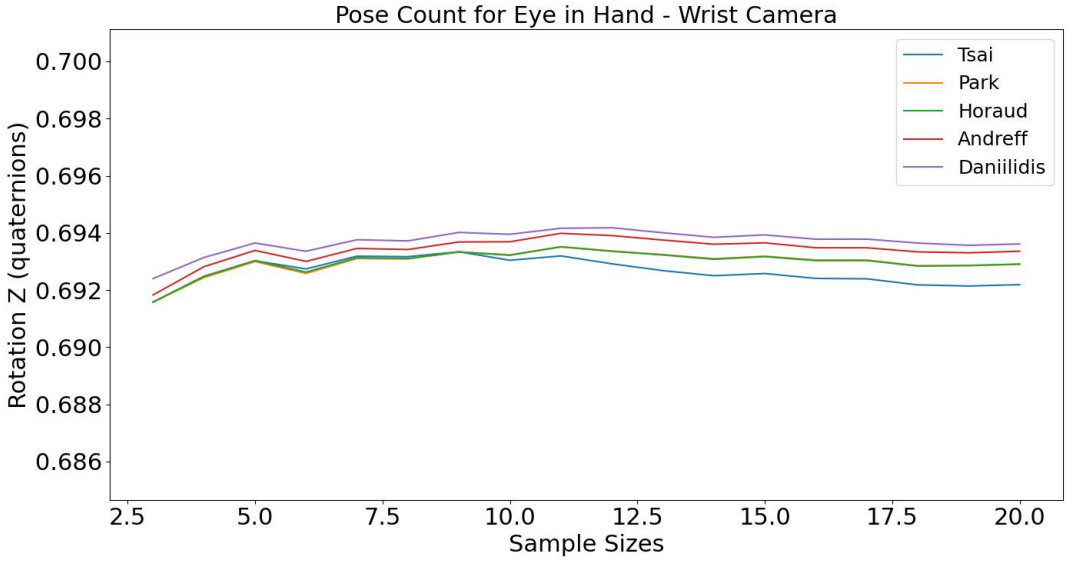


Figure 5.24: The estimated Z-rotation for $ee \rightarrow camera$.

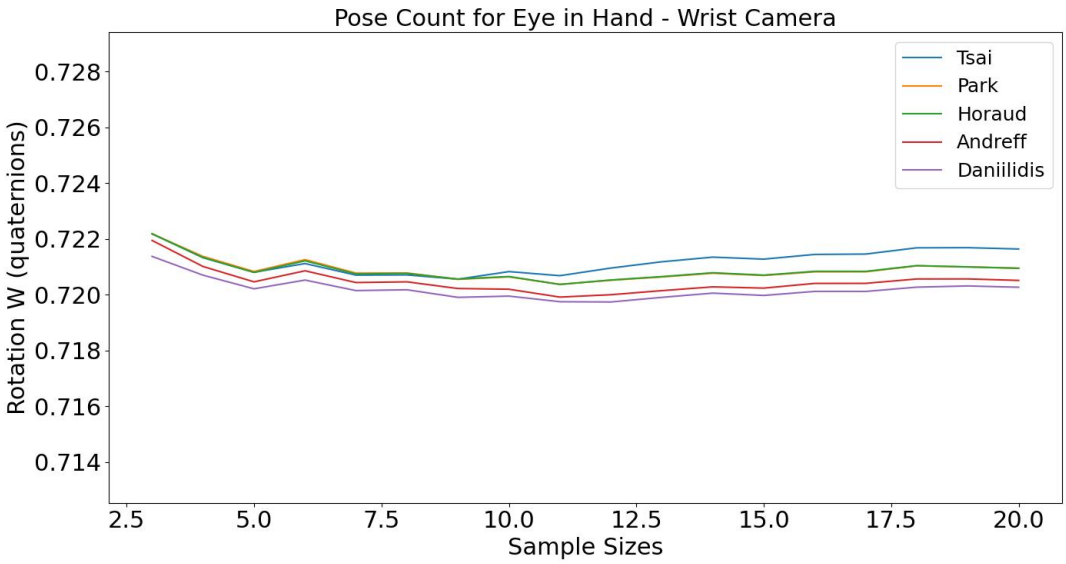


Figure 5.25: The estimated W-rotation for *ee* → *camera*.

5.2.2 Eye to hand

Pose Count - Front Camera

Figures 5.26-5.32 show how the front camera extrinsic calibration results vary over a set of 20 poses for each of the five calibration methods used.

The most striking feature of these graphs is the large initial spread when using only the three required poses, solidifying the conclusion that using more than three poses is highly recommended.

Concerning translation, figures 5.26-5.28, we see a clear tendency towards plateaus after the fourth pose. These plateaus are however broken up at poses 6, 19 and 20. Much like in the case of eye-in-hand calibration, the likely culprit for these spreading events is noise and bad data. This indicates that merely adding more data does not necessitate more stable calibration results, but that the entire process needs to be carefully monitored throughout.

Regarding rotation, the x and y rotation graphs, figures 5.29 and 5.30, follow the same trend seen for the translation. However, the z and w rotation graphs, figures 5.31 and 5.32, seem to be forming a mirrored inclination up and down respectively. More data would be needed to see if this trend would plateau at a later stage.

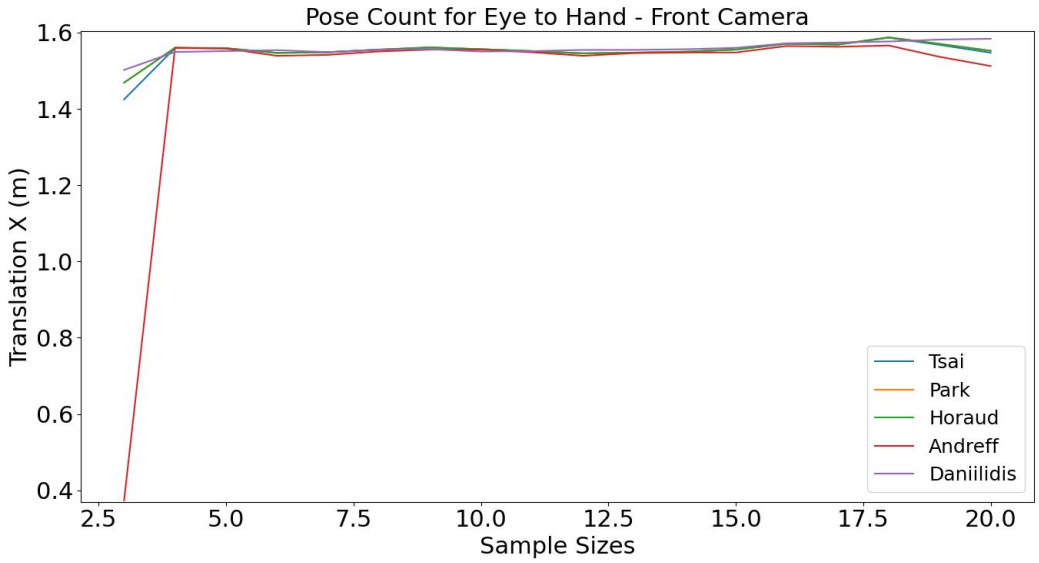


Figure 5.26: The estimated X-translation for *world* → *camera*.

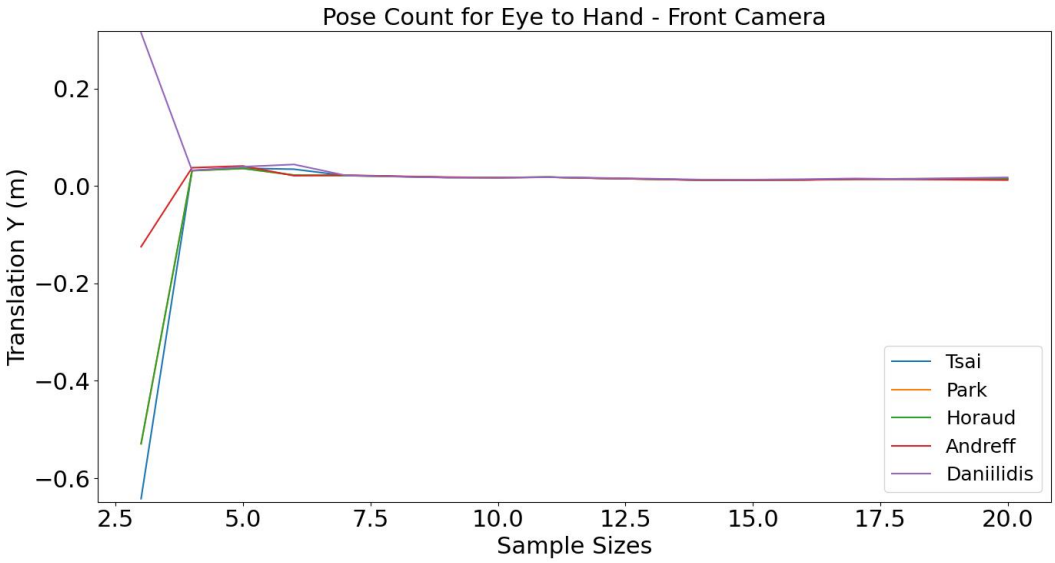


Figure 5.27: The estimated Y-translation for *world → camera*.

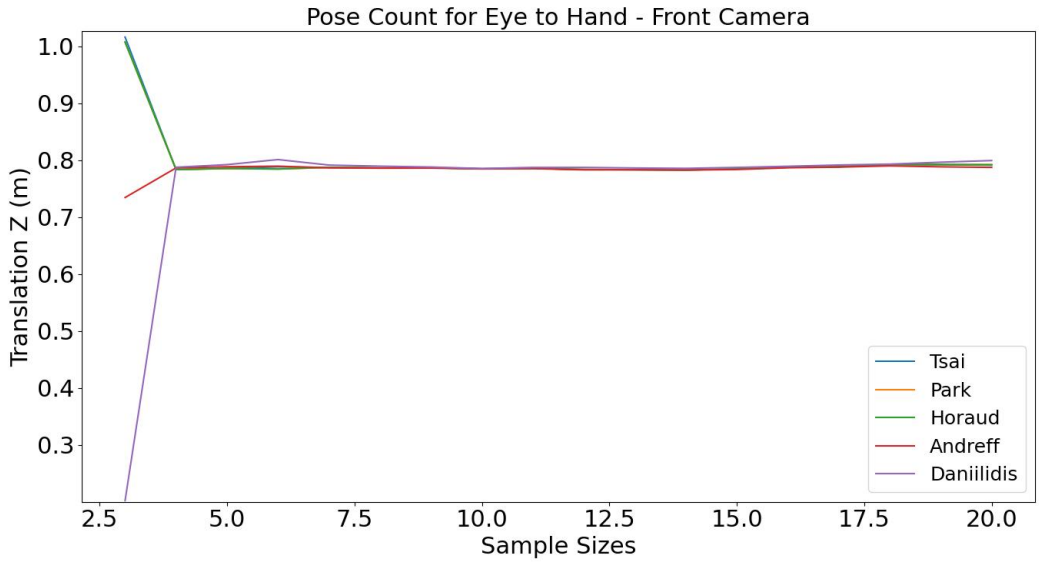


Figure 5.28: The estimated Z-translation for *world → camera*.

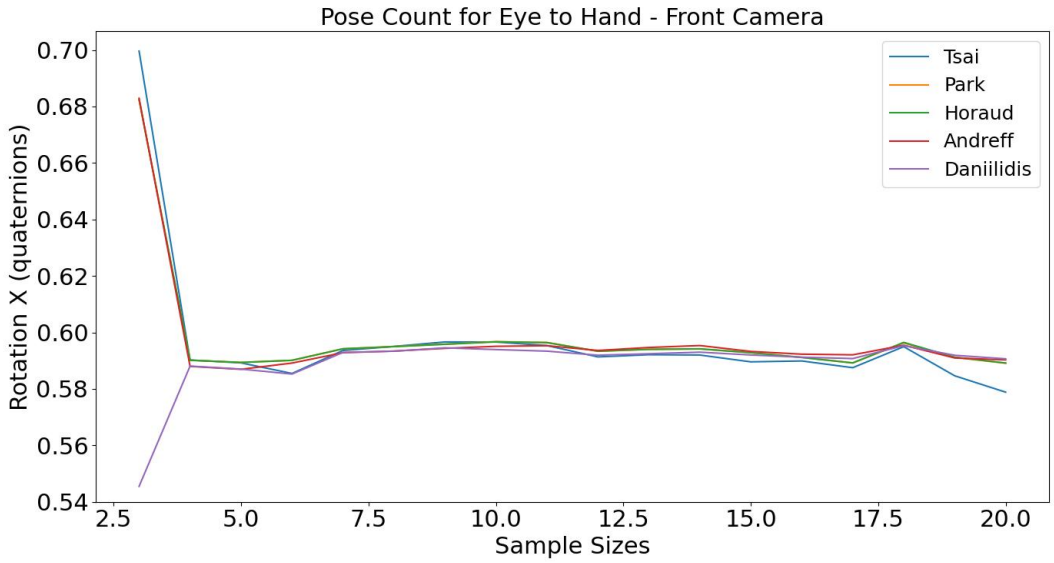


Figure 5.29: The estimated X-rotation for *world → camera*.

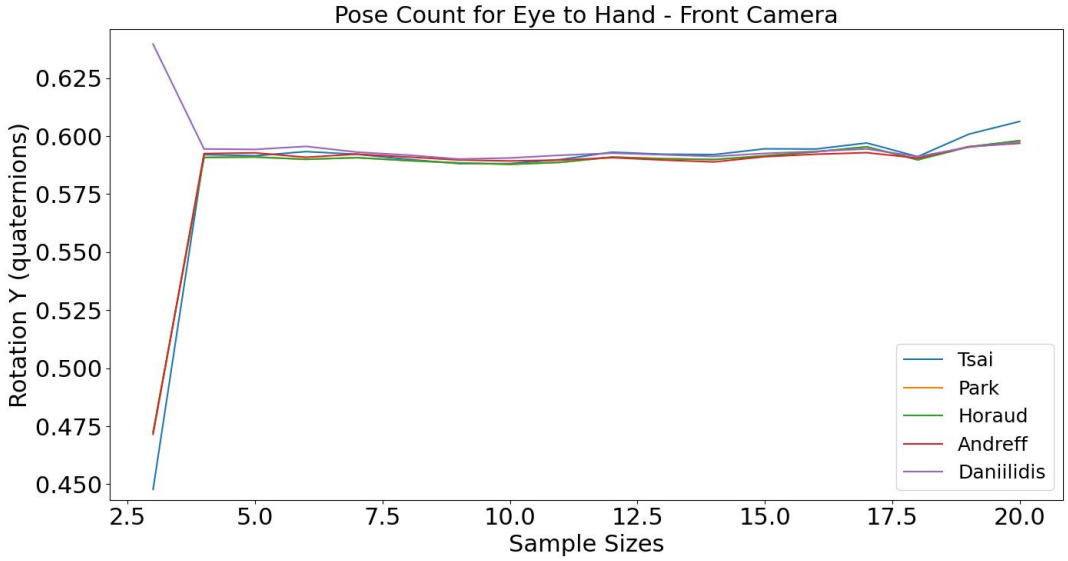


Figure 5.30: The estimated Y-rotation for *world → camera*.

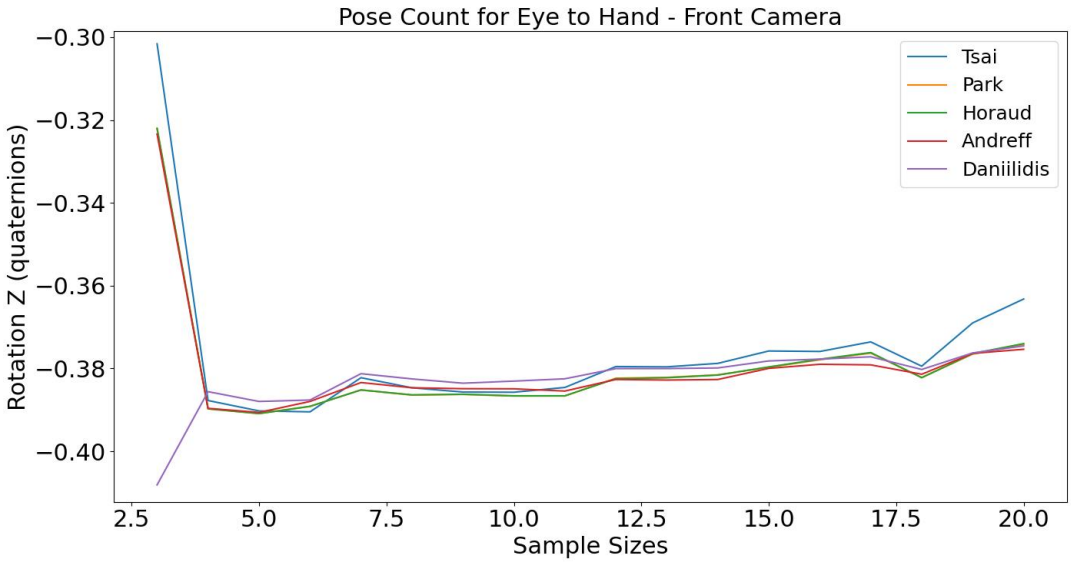


Figure 5.31: The estimated Z-rotation for *world → camera*.

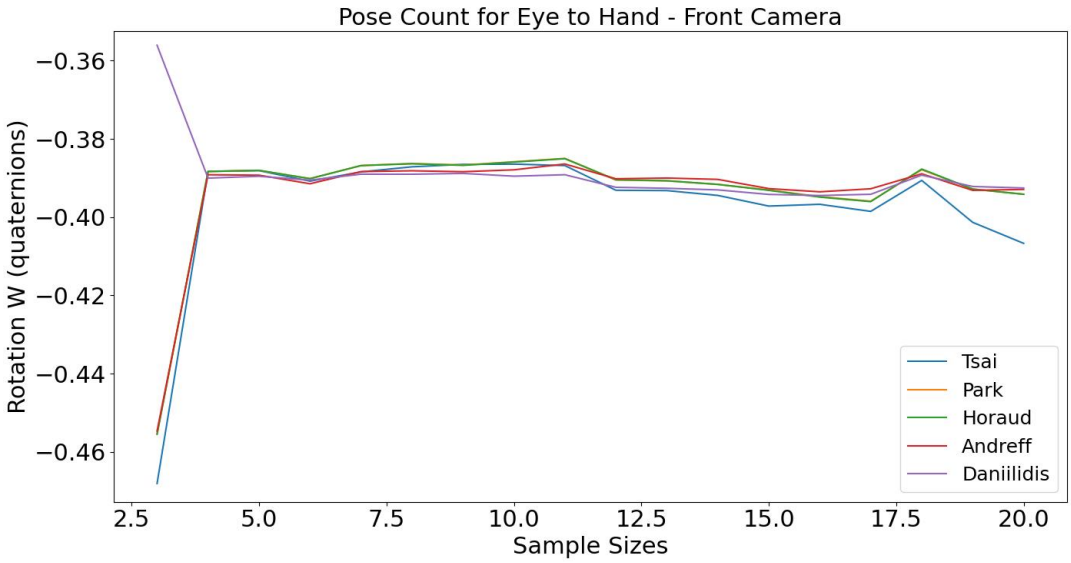


Figure 5.32: The estimated W-rotation for *world → camera*.

Pose Count - Top Camera

Figures 5.33-5.39 show how the top camera extrinsic calibration results vary over a set of 10 poses for each of the five calibration methods used.

Here we see how very different the results obtained can be depending on which implementation is used. Regarding translation, the two simultaneous solutions (dubbed "Andreff" and "Daniilidis") provide results which conflict with the separable solutions (dubbed "Tsai", "Park" and "Horaud"). Meanwhile, "Tsai" is a consistent outlier with regard to rotation.

It is noteworthy that none of the rotation graphs, figures 5.36-5.39 plateau to the same degree as was seen for the wrist or front cameras. This could be due to the series containing fewer poses than those two cameras, indicating that 10 poses could be too few.

However, we do see a similar plateauing effect as was seen for the wrist and front cameras for the translation, even though the methods may not be in agreement as to where those plateaus should be. This further enforces the conclusion that it is important to monitor the calibration process and be mindful of the quality of the data collected, rather than aiming for as many poses as possible.

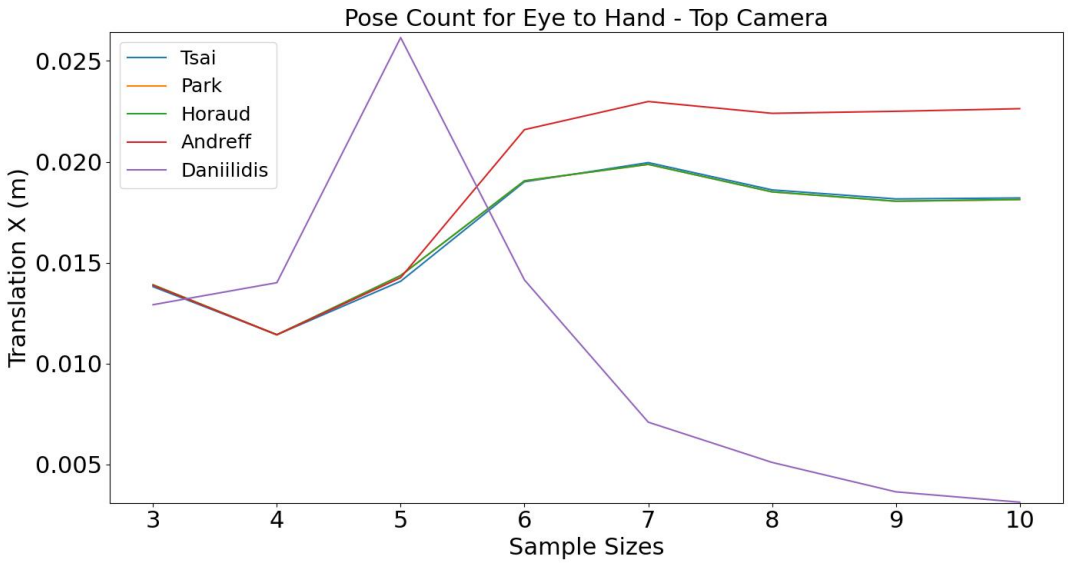


Figure 5.33: The estimated X-translation for *world → camera*.

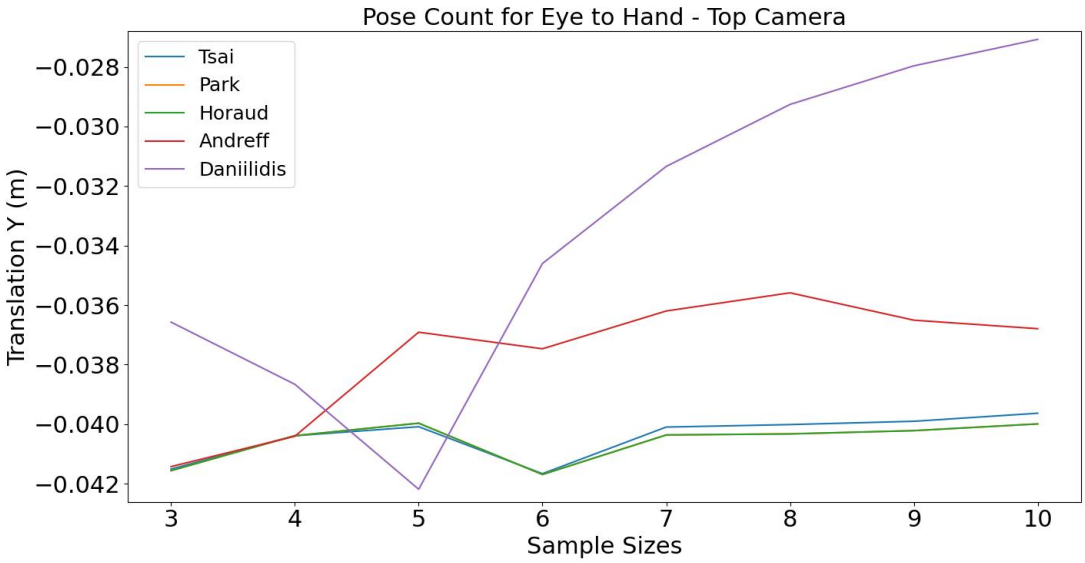


Figure 5.34: The estimated Y-translation for *world → camera*.

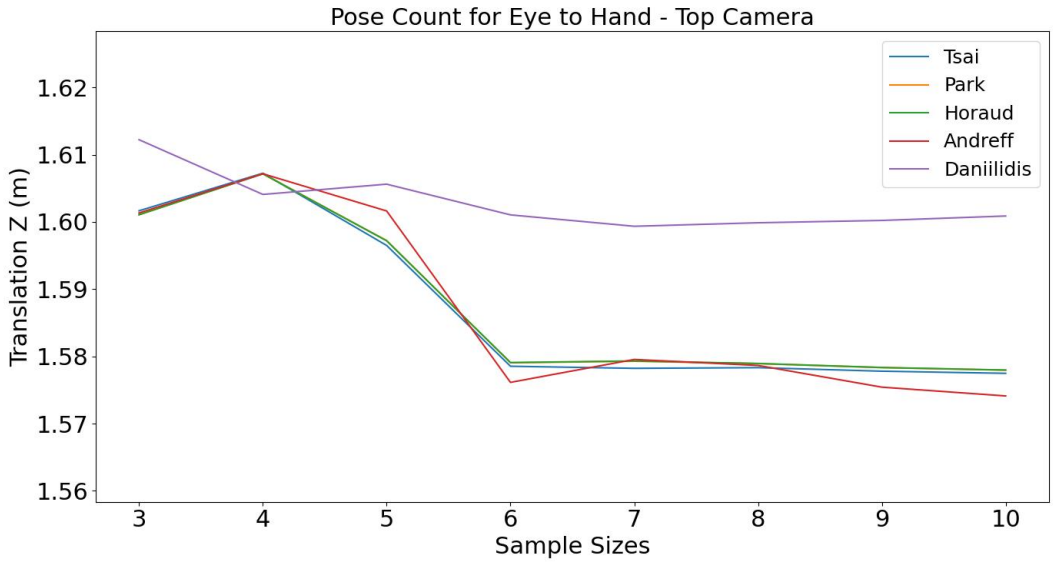


Figure 5.35: The estimated Z-translation for *world → camera*.

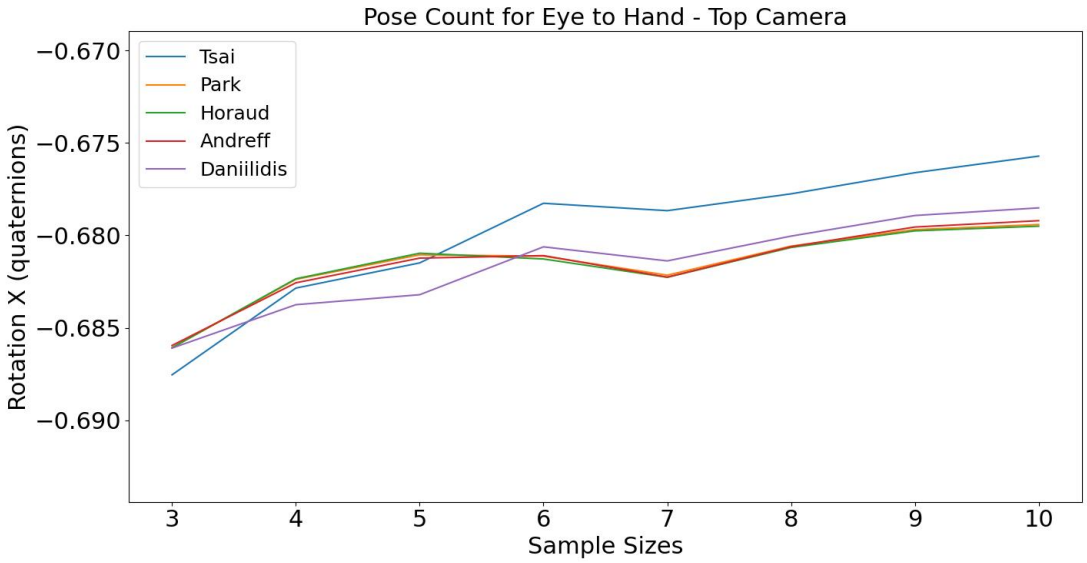


Figure 5.36: The estimated X-rotation for *world → camera*.

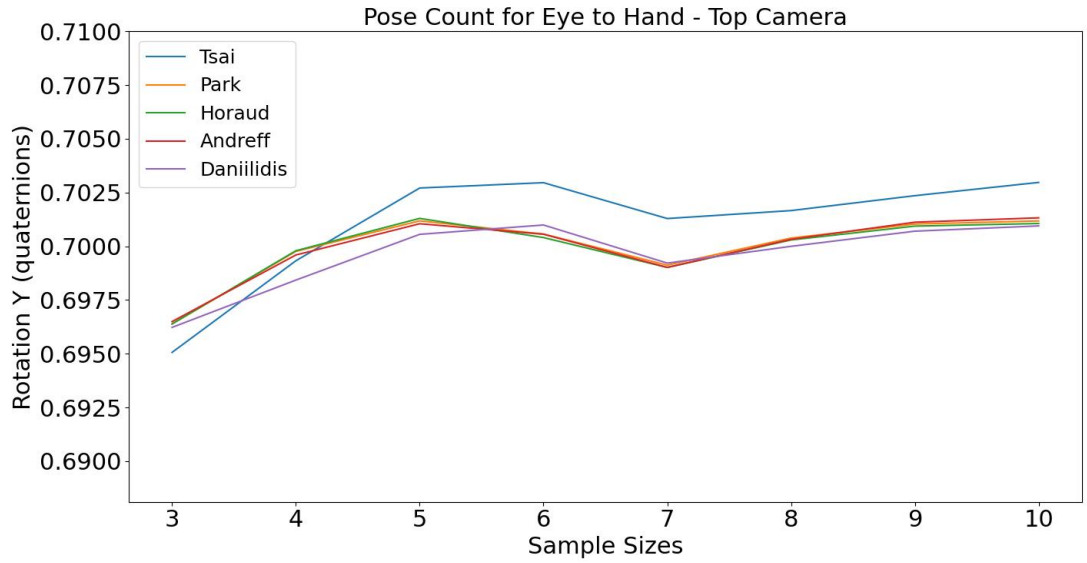


Figure 5.37: The estimated Y-rotation for *world → camera*.

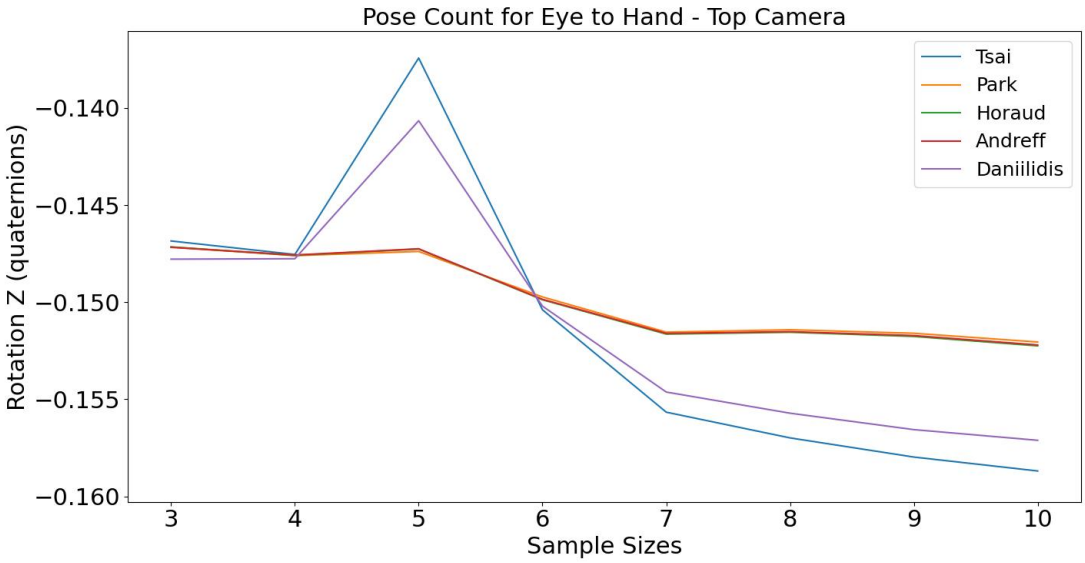


Figure 5.38: The estimated Z-rotation for *world* → *camera*.

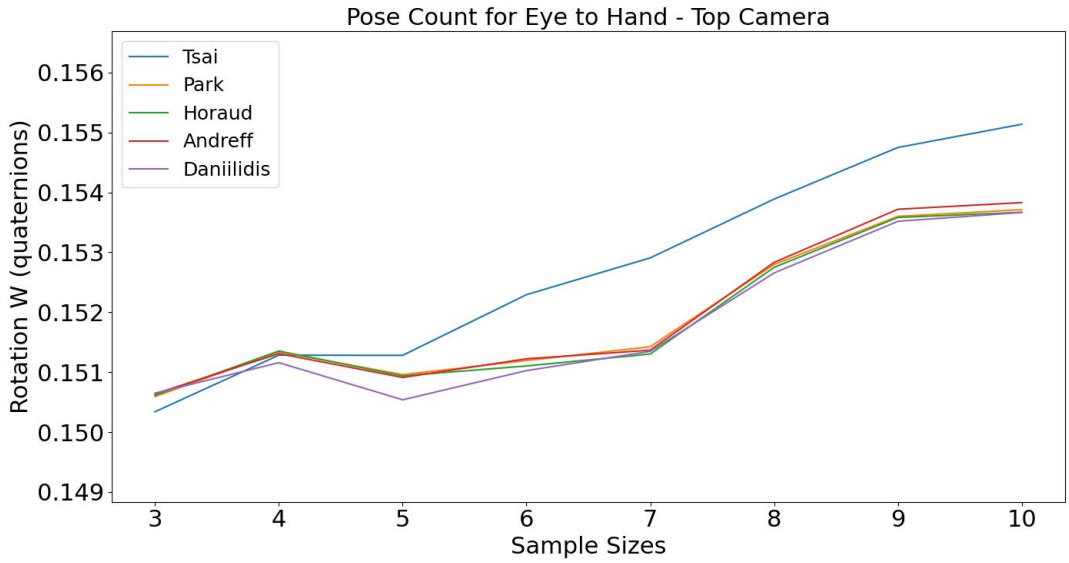


Figure 5.39: The estimated W-rotation for *world* → *camera*.

Position Shift

The results in table 5.6 were obtained when performing an extrinsic calibration before and after a known shift in location along a single space axis, namely the x axis. The expected result would be for the new calibration estimate to be shifted minimally (close to zero) along the y and z axes while being shifted -0.2 metres along the x axis. The result is similar to this expectation, with a shift of 3 millimetres along the y and z axes and a -0.1966 metre shift along the x axis. This means that our calibration system correctly reflects changes in the real-world setup with an error of 3.4 millimetres, provided the shift was exactly 0.2 metres.

Table 5.6: Extrinsic calibration before and after shifting the camera 20 cm closer to the robot base link along the x-axis, with δ being the absolute distance between calibration results.

Position	Translation (m)		Rotation (quaternions)	
Before Shift	x	1.5572	r_x	0.59735
	y	0.0078588	r_y	0.59064
	z	0.79304	r_z	-0.37846
			r_w	-0.38871
After Shift	x	1.3606	r_x	0.59669
	y	0.010794	r_y	0.58960
	z	0.78998	r_z	-0.38004
			r_w	-0.38974
δ	x	0.19660	r_x	0.00066
	y	0.0029352	r_y	0.00104
	z	0.00306	r_z	0.00158
			r_w	0.00103

Sliding Window Mean of the camera to target transform

These results reflect the effect of smoothening the $camera \rightarrow target$ transform. This is done by using a sliding window to calculate the mean value thereof, contra relying on the newest singular value. Much like in the case of eye-in-hand calibration, the resulting difference between the two approaches is very small. This further enforces that this is a non-factor, even when using a smaller ChArUco board, and that the transformation between the camera and ChArUco board can be considered stable.

Table 5.7: Extrinsic calibration results with and without a sliding window, with δ being the absolute distance between calibration results.

Window Size	Translation (m)		Rotation (quaternions)	
1	x	1.5758	r_x	0.59547
	y	0.0098020	r_y	0.59360
	z	0.79920	r_z	-0.37577
			r_w	-0.38968
30	x	1.5747	r_x	0.59468
	y	0.0072503	r_y	0.59415
	z	0.79891	r_z	-0.37364
			r_w	-0.39209
δ	x	0.0011	r_x	0.00079
	y	0.0025517	r_y	0.00055
	z	0.00029	r_z	0.00213
			r_w	0.00147

Dynamic Comparison

Table 5.8 displays the difference between calibrating by solving $AX = XB$ and our proposed dynamic calibration approach. The camera being calibrated is the front camera, and it was not moved between calibrations.

The results show a difference in output between the two approaches, out of which the shifts along the y and z axes are of most concern. Here we see a difference of 1.6 and 2.7 centimetres respectively, which is a far greater degree of error than has been seen throughout the other experiments. This means that our dynamic estimate method would need more work and investigation to be reliable for precision calibration.

Table 5.8: Extrinsic calibration results using the dynamic approach contra solving $AX = XB$, with δ being the absolute distance between calibration results.

Method	Translation (m)		Rotation (quaternions)	
Solving $AX = XB$	x	1.5572	r_x	0.59735
	y	0.0078588	r_y	0.59064
	z	0.79304	r_z	-0.37846
			r_w	-0.38871
Dynamic Estimate	x	1.5530	r_x	0.58971
	y	0.023936	r_y	0.59092
	z	0.76595	r_z	-0.38495
			r_w	-0.39353

Table 5.8: Extrinsic calibration results using the dynamic approach contra solving $AX = XB$, with δ being the absolute distance between calibration results. (Continued)

δ	x	0.0042	r_x	0.00764
	y	0.016077	r_y	0.00028
	z	0.02709	r_z	0.00376
			r_w	0.00482

Sensitivity to Bad Data

Figure 5.40 shows an extrinsic calibration series that highlights the sensitivity of the process by showcasing a final pose that consistently produces noise in the calibration results.

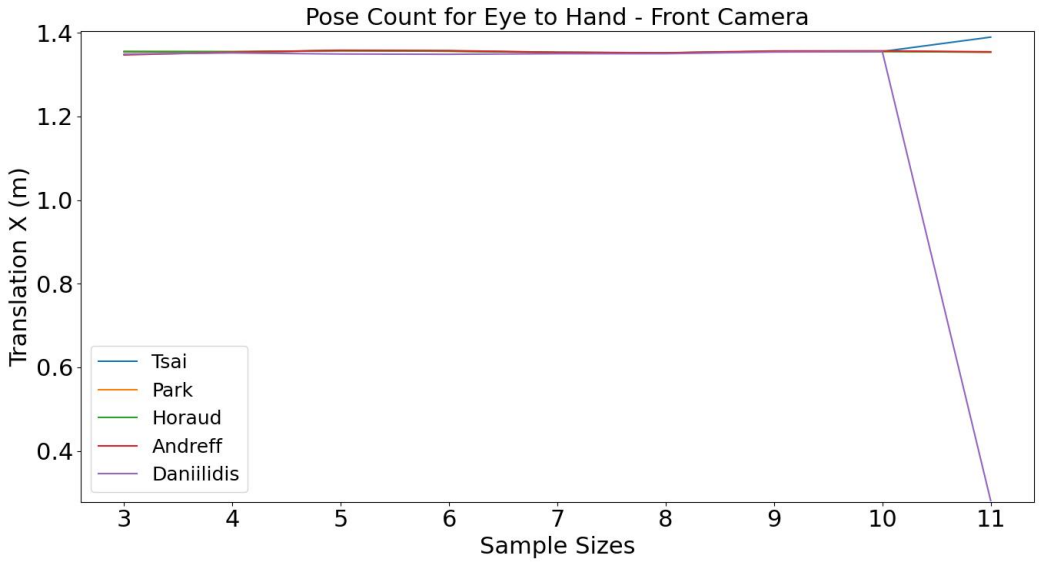


Figure 5.40: The estimated X-translation for *world* → *camera*.

5.3 Verification

5.3.1 Pick & Place Verification

Below are two tables reflecting the results when performing verification by using the pick and place method. Table 5.9 reflects the attempt without taking cube geometry into account, unlike table 5.10 which does include an offset equal to half of the width of the cube.

While the wrist-mounted camera manages a 100% success rate, it is evident that the front and top cameras struggle without taking the cube's geometry into account and introducing offsets. Introducing these offsets improves the success rate to 90% for the front camera and 60% for the top camera.

Table 5.9: Verifying the calibration with a pick and place task without offsets for cube geometry.

Camera position	Successful	Failed	Total
Wrist	10	0	10
Front	0	5	5
Top	0	5	5

Table 5.10: Verifying the calibration with a pick and place task with offsets for cube geometry.

Camera position	Successful	Failed	Total
Wrist	10	0	10
Front	9	1	10
Top	6	4	10

5.3.2 ArUco Centre Pinpoint Verification

Figures 5.41-5.43 show the results when verifying a set of extrinsic camera calibration estimates with the ArUco centre pinpoint method. For the front and top cameras, two images are provided, one using the calibrated values and one using adjusted values based on the verification results.

Using the calibrated camera position for the wrist, the robot pinpoints a centre location ca. 2 millimetres shifted along the positive x axis from the actual centre. Due to this small error, no correction pass was carried out. This is largely due to the fact that outstanding error sources could equally well account for the remaining error.

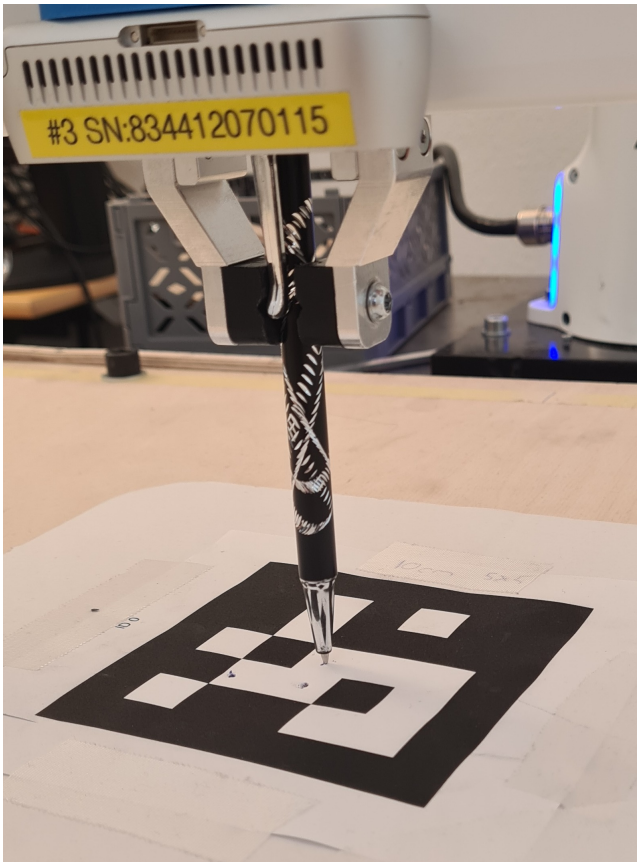
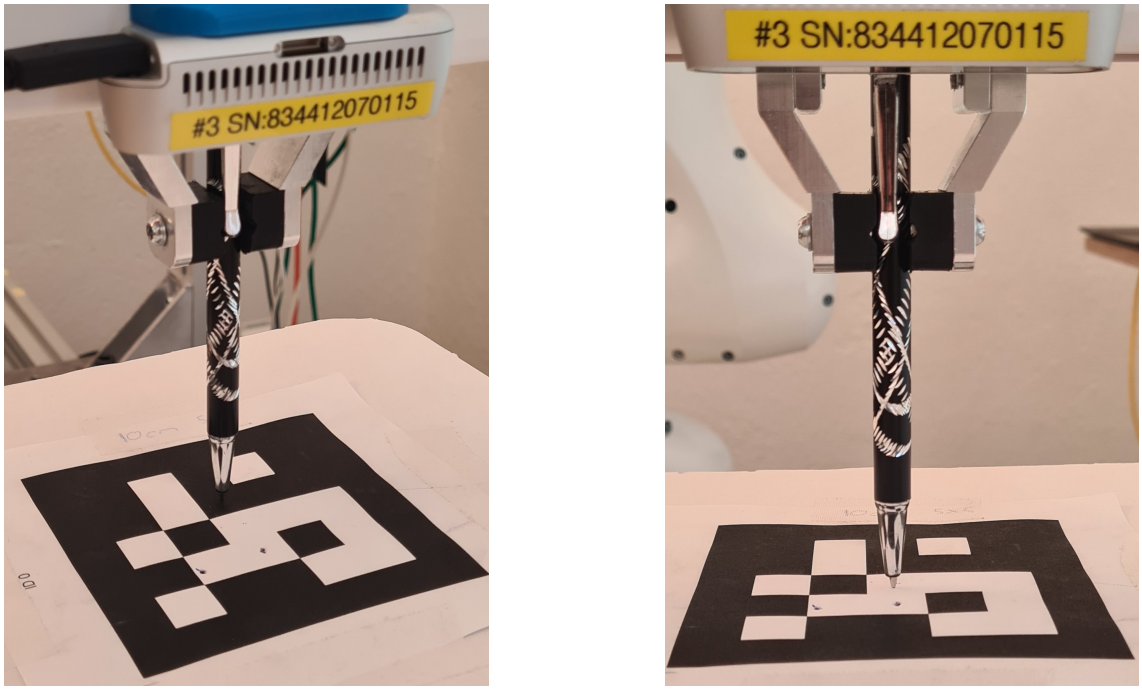


Figure 5.41: ArUco centre pinpoint using the wrist camera. The ArUco is 10 cm across, with each tile being 1.43 cm wide.

When verifying the calibrated front camera position, the robot pinpoints a centre location which is shifted 1.4 centimetres along the negative x axis from the actual centre. Using this information, the estimated location of the camera was shifted 1.4 centimetres away from the robot base, yielding a result where the centre was shifted ca. 3 millimetres along the positive axis. Much like in the case of the wrist camera, this error could equally well be due to outstanding sources.

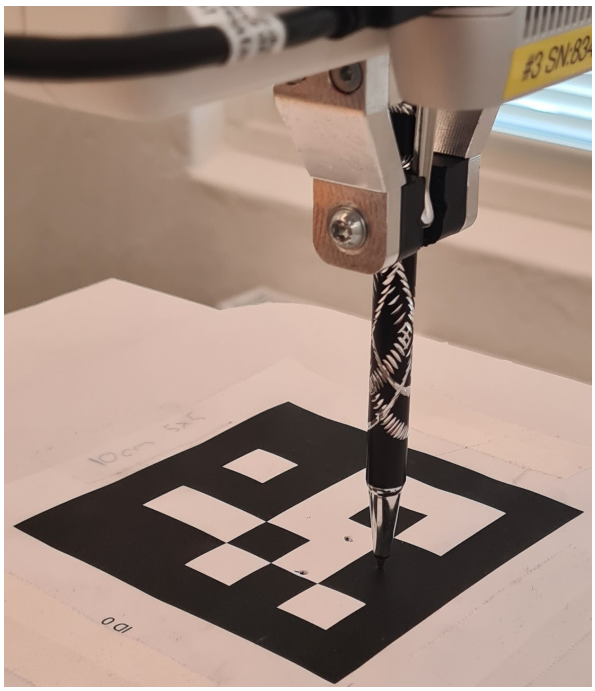


(a) Using calibrated value.

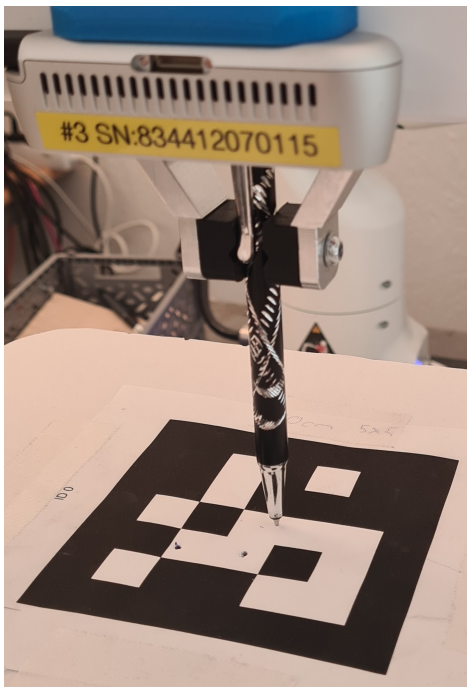
(b) Using adjusted value.

Figure 5.42: ArUco centre pinpoint for front camera.

The top camera had the worst initial performance, locating a point 2.1 centimetres along the positive x axis and 0.7 centimetres along the negative y axis. Using these values to shift the estimation, a new centre was located 2 millimetres along the negative x axis and ca. 3-4 millimetres along the positive y axis. Again this error is small enough that the error may stem from outstanding factors. Additionally, the top camera is located the furthest from the ArUco, even though the ArUco was placed in an elevated position for this experiment. This means that it struggles to correctly calculate the camera to ArUco transform when compared to the other two cameras.



(a) Using calibrated value.



(b) Using adjusted value.

Figure 5.43: ArUco centre pinpoint for the top camera.

Chapter 6
Discussion & Conclusions

6.1 Intrinsic Camera Calibration

Figures 5.1 through 5.4 show that every intrinsic parameter converges after between five and ten images. Thus, in order to obtain a camera matrix there is no need to collect an exorbitant amount of images. This might not be true for the distortion parameters though, see figure 5.5 through 5.9. Here, a distinct shift can be seen between 15 and 20 images, even though the scale of these parameters is small. Due to the abrupt nature of the shift, it could be that the underlying cause is a bad data point. Such a data point could potentially contain fewer located ChArUco corners, thus leading to a worse estimate being provided.

The same figures also show the difference between using and not using an initial guess. A very clear pattern can be seen in these graphs. When not supplying an initial guess, the first five images all produce wildly varying results and then converge to the same values as with an initial guess. The reason for these first few images resulting in such varied outputs is probably due to the first five images in this set being required to cover the entire image plane. So while the result seems to converge to the same value for having and not having an initial guess, supplying one can be preferable if you are constricted to a very small set of images.

Looking at how distance to the calibration target factors in, figures 5.10 through 5.18 show a difference in the results obtained. It is hard to know with certainty that this is only caused by the change in distance since changing the distance will also change the angle at which the calibration target is viewed. Of note is that for the intrinsic parameters, the *far away* image set trended closer to the factory settings than the *near* image set. For the distortion parameters, the results are not so clear-cut and the best-performing image set varies between each parameter.

In summary, it would be recommended to use enough images to cover the entire image plane and ensure variation in the image set. We found this to be doable with 5 images. Due to the observed sensitivity, it is important to gather high-quality images with many visible corners (if using a ChArUco).

6.2 Extrinsic Camera Calibration

6.2.1 Eye In Hand

According to the results in tables 5.3 and 5.4, we can say that the algorithms can consistently provide the same results. Over a set of three calibrations using identical poses, we see stability on a millimetre level with a standard deviation of no more than 9.7×10^{-5}. When using non-identical poses the results are still good, although not to the same degree. Here we see a variation at the millimetre level, which is reflected by the higher standard deviation.

From figure 5.19 through figure 5.25 we can see that the calibration results generally do not stabilise to a precise value regardless of the underlying implementation for solving the $AX = XB$ matrix equation. The algorithms all seem to fluctuate with each new pose added at a millimetre level, meaning that it is not possible to conclude that there is a certain number of poses which would yield better results.

Table 5.1 shows a marked difference in the results when not varying the rotation of the end effector, thus moving it along a plane. This mainly affects the translation, but strongly indicates that ensuring varied poses is of utmost importance for a good calibration result.

In an earlier version of the project, singular ArUco markers were employed instead of ChArUco boards. During this time, the concept of smoothening the $camera \rightarrow target$ transform was introduced due to a notable instability in the estimation on a frame-by-frame basis. However, table 5.5 shows that this is mostly a non-factor when using ChArUco boards. The observed difference in results when taking the mean of the latest 30 transforms, as compared to using only the latest singular transform, is comparable to what is seen in table 5.4. This can likely be attributed to the ChArUco board simply having more located points, leading to greater stability.

6.2.2 Eye To Hand

It was assumed that extrinsic calibration for the eye-to-hand case would behave much like the eye-in-hand counterpart. This is largely true, with a similar pattern of quick convergence followed by a meandering plateau.

Figures 5.26 to 5.32 show a tendency for certain data points to produce a spreading effect, see the 6th and 18th sample sizes in figure 5.27 and 5.28. A similar effect can be seen for the eye-in-hand setup, see figure 5.21 in a vicinity around the 10th sample size, where the methods dubbed "Andreff" and "Daniliidis" diverge. The probable reason for such occurrences is noise in the data. We believe that the utilisation of a smaller ChArUco board combined with the more constrained range of available poses when calibrating the front and top cameras gives rise to such noise.

Where we do see a bigger difference is in figure 5.33 to 5.39. Here we see several algorithms, Daniilidis and Andreff for translation and Tsai and Daniilidis for rotation, providing markedly different results than the rest. It has generally been found that the top camera is more difficult to calibrate, with a possible reason being the difficulty of obtaining good poses with the robot arm.

Figure 5.40 shows a calibration series of the front camera, where the last value is a pose which has been seen to produce bad data. Such a pose is reproducible and found when turning

the robot arm "upside down" by heavily contorting and rotating the robot joints. It is worth noting that much like for the top camera, it is the method dubbed "Daniilidis" which reacts most to this data point. We theorise that it is easier to obtain such poses when calibrating the top camera, due to its location relative to the robot. This combined with the lack of variety in poses, due to the same reason, could explain why the top camera is more difficult to calibrate. This is also the reason only 10 images were used in the pose count experiment for the top camera. Managing to collect 20 images for the top camera in a single dataset, without introducing significant noise, proved challenging. Especially when combined with a thread-related bug caused by OpenCV and matplotlib, which caused the application to crash if left running for extended periods of time.

The results found in table 5.6 show that moving the camera a known distance will yield good results. The intended change in X-translation was 20 centimetres, and the resulting calibration was 3.4 millimetres short of this. Unfortunately, it is hard to say with certainty where the fault lies. It is not guaranteed that the camera was moved exactly 20 centimetres, due to the nature of the rig, and there is the uncertainty of both the pre- and post-shift calibration results to consider.

Much like what was observed in the eye-in-hand case, table 5.7 shows that in spite of using a smaller ChArUco board, applying a sliding window mean to the $camera \rightarrow target$ transform does not provide markedly different results.

6.2.3 Dynamic

The results when comparing the proposed dynamic approach with results from solving the $AX = XB$ matrix equation, see table 5.8, show differences of 1.6 centimetres and 2.7 centimetres for the Y- and Z-translations respectively. Such a large difference essentially means that it is not feasible to use this approach for its intended use case, that of making slight adjustments without having to go through several extrinsic calibration cycles. It can however be used to find roughly the correct position instead, thus forgoing the need to measure distances to the robot manually for initial positioning.

The first potential reason for this offset would be the quality of the calibration results for the wrist-mounted camera, intrinsic and extrinsic alike. This is due to the pivotal role said calibration result plays when establishing the ChArUco board's location in the world frame. However, the good results obtained when verifying the wrist camera speaks against this being the main issue, see tables 5.9, 5.10 and figure 5.41.

Furthermore, both the wrist-mounted camera and the camera to be calibrated must have a good view of the ChArUco board, so as to acquire a good estimate for the transform therebetween. This seems to be a more likely reason for the offset, as the setup does not allow for both cameras to have a direct, unslanted, view of the ChArUco simultaneously.

6.3 Verification

The pick and place verification method proved to be highly situational, as can be seen in table 5.9. When used with the wrist-mounted camera, the robot managed to pick and place the cube correctly 100% of the time. But the same can not be said for the front and top cameras, where the robot failed every time before introducing offsets to account for the cube's

geometry, which improved the results to 90% and 60% respectively. As can be seen in figures 5.41 to 5.43, the issue does not seem to be the calibrated estimates. The front camera estimates the centre point to be less than 1.5 cm off from the true centre, and the top camera estimates it to be off by less than 3 cm.

A recurring issue for the top camera in particular, was it not being able to register the cube with its depth sensor. As can be seen when comparing the infrared speckle patterns of each respective camera, figure 6.1, 6.2 and 6.3, due to the distance and small size of the cube, there are very few depth measuring points which actually hit the cube. This would lead to the arm trying to pick up a cube "inside" the workspace table, forcing an emergency shutdown of the arm. This was a problem for the front camera as well, but not to the same extent as the top camera.

Moreover, the results in table 5.9 indicate a deficiency in our HSV-object detection. The implementation picks the centre of the detected area as the x, y and z coordinates for the pick and/or place action. This means that the front camera, which sees the cubes largely from the side, would pick out coordinates on the very edge of the cube. To remedy this, offsets were added for the front and top cameras with the assumption that coordinates would be picked along the side of the cube. With these offsets in place, performance improved drastically, as can be seen in table 5.10.

With all this in mind, the ArUcO centre pinpoint method then provides a far better understanding of how well an extrinsic calibration estimate reflects the actual real-world setup. However, there are a few outstanding sources of error with the approach in its current incarnation. Firstly, the pen used is rather long, meaning that if it is not held completely straight by the gripper there will be an error in the reading. Using a shorter object is therefore advisable. Secondly, the ArUco used is not of the best quality. It was printed using a home printer on normal paper and as is visible in the images it buckled slightly after some time of usage.

Additionally, it can be difficult to verify the z axis translation. If the estimate is too far above the ArUco, that can be measured. But if the estimate is below the ArUco, the pen will either gently press into the ArUco or outright pierce it. This is difficult to quantify into a measurement but still provides the knowledge that the estimate is wrong.

Lastly, while the ArUco centre pinpoint verification method works when verifying translation, the current implementation does not provide any information regarding the rotation estimate. This could however be amended by exchanging the tool used for pinpointing with something which reflects the estimated rotation by mimicking the axes of the ArUco instead of a simple sharp object like the pencil used in our experiment.

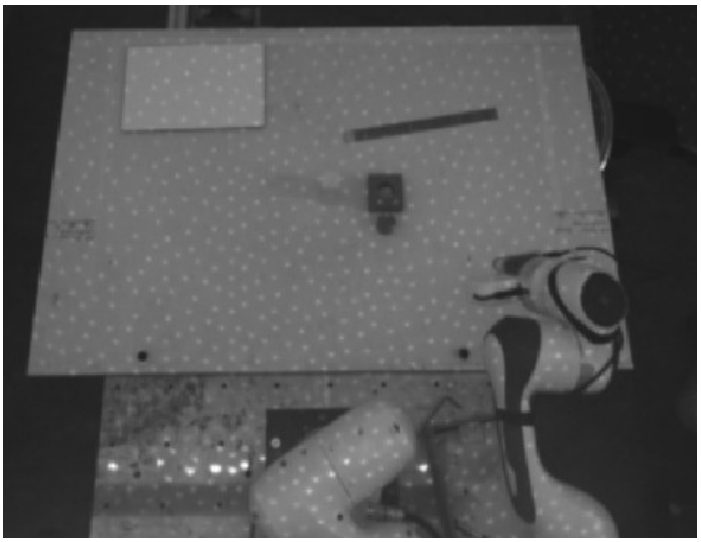


Figure 6.1: The figure shows the infrared speckle pattern with which depth data is collected by the top camera.

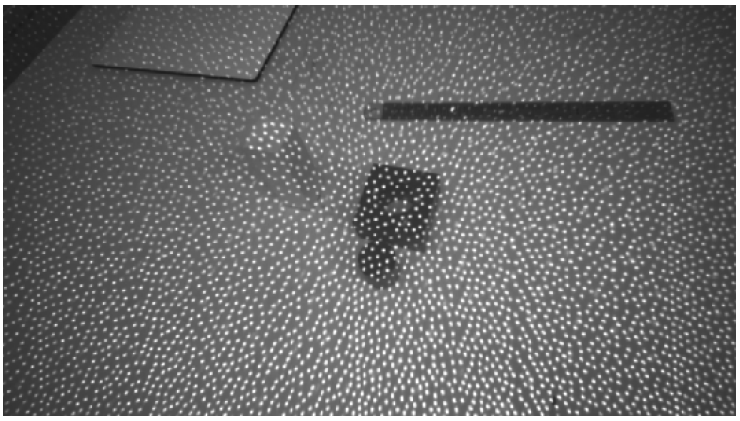


Figure 6.2: The figure shows the infrared speckle pattern with which depth data is collected by the wrist camera.

Figure 6.3: The figure shows the infrared speckle pattern with which depth data is collected by the front camera.

6.4 Comparison to related work

The paper "A Deep Learning-Based Hand-eye Calibration" in related work 1.2 used the TSAI [32] method as a baseline. The robot used was a Universal Robot (UR3) with a 0.1 millimetre repeatability and a Stereolabs ZED camera placed in 24 different positions for training. The baseline achieved a 1.039966529 millimetres mean and 0.4237020321 millimetres standard deviation while their model achieved a 2.07 millimetres mean and 0.081 millimetres standard deviation in position error.

In the other paper mentioned in related work "Continuous hand-eye calibration using 3D points" they achieved around 7 millimetres in position error in their real-world experiment for the best method called "YQuatT" with a UR10 robot arm and Xtion RGB-D camera mounted on a rack above the arm.

Compared to our position errors for the pinpoint test converted to Euclidean distance we were able to get a 2 millimetres position error for the wrist-mounted camera. The front camera achieved 14 millimetres at first and 3 millimetres after adjustment. The final top camera achieved a 22 millimetre position error that could be adjusted down to 3.6 millimetres.

This suggests that the proposed hand-eye calibration system performs competitively with existing methods, particularly following position adjustments. However, it is crucial to note that further testing is necessary to ensure the reliability of these results.

6.5 Answer to questions

The questions we sought to answer with this project were:

- RQ1: What would be a suitable method for camera calibration in the context of sim2real alignment?
- RQ2: How can the results of such estimates be verified?
- RQ3: Which factors affect the quality of the calibration process?

We have found that a combination of the dynamic approach for initial placement and then repeatedly performing the extrinsic calibration by solving the $AX = XB$ matrix equation with adjustments in between can be a suitable method for extrinsic camera calibration in the context of sim2real alignment. This means that given a set of coordinates, you can with the above schema place the camera in a correct pose in selfsame location. It has also been shown that shifting a camera's location will be correctly reflected by the calibration methods, lending further credit to the approach.

Furthermore, the resulting estimates can be verified with the ArUco centre pinpoint approach and further adjusted by acting on the results thereof. This proved to be a very successful approach for the front camera, as can be seen in figure 5.42. The results for the top camera also improved with adjustment, see figure 5.43, but the transform between the camera and ArUco was unstable due to the distance therebetween. This could be remedied with a larger ArUco. The wrist camera performed well enough that no adjustment pass was conducted.

We do not find the pick-and-place implementation using HSV-object detection to be a favourable option for verification, due to its sensitivity to detection quality. Additionally, the feedback provided by pick and place is far less actionable than that of the ArUco centre pinpoint.

It was found that noise in the data can greatly affect the results of the calibration. Therefore, ensuring good data by monitoring the progress during calibration is important. Moreover, collecting a varied data set by ensuring the robot arm and end effector have been sufficiently moved is important. The above holds true for both intrinsic and extrinsic calibration. A difference is observed with regard to data set size. It is inconclusive how important it is to gather a large data set for extrinsic calibration since larger sets do not provide inherently better results. However, one can confidently say that using more than the minimum three poses required for the algorithms is advisable. Intrinsic calibration on the other hand can generally be considered stable after 10 images, provided they are of good quality.

6.6 Limitations of our approach

The schema has only been tested on a single setup, meaning that it is uncertain how well it would work on other cameras and robots. This means that our aim of a platform-agnostic application is unverified.

For the scope of this project, aligning the real-world setup to the simulated environment has been constrained to being able to precisely locate cameras and verifying that a shift in location is correctly reflected. This allows for the correct positioning of cameras in accordance with their simulated counterparts, which were unfortunately not replicable with the hardware and location available to us. Furthermore, true alignment requires more than precise camera location replication, as there can be differences in field of view and camera intrinsics which are not accounted for.

6.7 Future work

In order to properly align the real-world setup with the simulated environment, a way to compare the two must be found. A first step could be an image comparison between the two setups. Such an approach would seek to create identical images in both environments and use the potential offset as an alignment indicator.

To facilitate calibration, a more usable application could be developed. Such an application could allow for selective inclusion of gathered data, which would make it easier to see how specific poses affect the resulting calibration.

Improving the dynamic approach could lead to speedy calibration of the static external cameras, thus avoiding the tedious process of repeated calibrations when performing slight adjustments to the camera's location.

Finally, once the setup has been deemed to be aligned properly, an agent trained in simulation could be deployed and evaluated on the real-world setup.

References

[1] N. Andreff, R. Horaud, and B. Espiau. On-line hand-eye calibration. *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062), 3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*, pages 430 – 436, 1999.

[2] Ozan Bahadir, Jan Paul Siebert, and Gerardo Aragon-Camarasa. A deep learning-based hand-eye calibration approach using a single reference point on a robot manipulator. *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1109–1114, 2022.

[3] Ilian Bonev. How is orientation in space represented with euler angles? `https://www.mecademic.com/academic_articles/space-orientation-euler-angles/`.

[4] The ROS community. Robot operating system. `https://www.ros.org/`.

[5] The Constructsim. Ros basics. `https://app.theconstructsim.com/courses/55`.

[6] K. Daniilidis. Hand-eye calibration using dual quaternions. *International Journal of Robotics Research*, 18(3):286 – 298, 1999.

[7] Franka Emika. Franka emika research. `https://frankaemika.github.io/docs/overview.html`.

[8] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[9] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[10] Ben Eater Grant Sanderson. Visualizing quaternions. `https://eater.net/quaternions`.

[11] Bjarne Grossmann and Volker Krüger. Continuous hand-eye calibration using 3d points. *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 311–318, 2017.

[12] Kenji Hata and Silvio Savarese. Cs231a course notes 1: Camera models. https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf.

[13] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *International Journal of Robotics Research*, 14(3):195, 1995.

[14] KUKA. Kuka - lbr iiwa. https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa.

[15] ROS Robotics Learning. Inverse kinematics. https://www.rosroboticslearning.com/inverse-kinematics.

[16] ROS Robotics Learning. Jacobian. https://www.rosroboticslearning.com/jacobian.

[17] MoveIt. Moveit motion planning framework. https://moveit.ros.org/.

[18] Sunita Nayak. Augmented reality using aruco markers in opencv (c++ / python). https://learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/.

[19] F.C. Park and B.J. Martin. Robot sensor calibration: solving ax=xb on the euclidean group. *IEEE Transactions on Robotics and Automation*, 10(5):717–721, 1994.

[20] Intel RealSense. Intel realsense cameras. https://www.intelrealsense.com/.

[21] Articulated Robotics. Getting ready for ros part 7: Describing a robot with urdf. https://articulatedrobotics.xyz/ready-for-ros-7-urdf/.

[22] Articulated Robotics. Transformations part 3: 2d rotations. https://articulatedrobotics.xyz/transformations-3-rotation_matrices_2d/.

[23] Articulated Robotics. Transformations part 4: Translations. https://articulatedrobotics.xyz/4-translations/.

[24] Reach Robotics. 'degrees of freedom' vs 'functions' of a robotic arm. https://reachrobotics.com/blog/degrees-of-freedom-vs-functions-of-a-robotic-arm/.

[25] Universal Robots. Robotic arm design. https://www.universal-robots.com/in/blog/robotic-arm-design/.

[26] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. pages 127–147, 2020.

[27] stepjam. Rlbench: Robot learning benchmark. https://github.com/stepjam/RLBench.

[28] OpenCV Team. Detection of charuco boards. https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html.

[29] OpenCV Team. Morphological transformations. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

[30] OpenCV team. Open source computer vision library. https://opencv.org/.

[31] OpenCV Team. Opencv docs. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d.

[32] R.Y. Tsai and R.K. Lenz. A new technique for fully autonomous and efficient 3d robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 5(3):345–358, 1989.

[33] Jeremiah van Oosten. Understanding quaternions. https://www.3dgep.com/understanding-quaternions/.

[34] W3Schools. Colors hsl and hsla. https://www.w3schools.com/colors/colors_hsl.asp.

[35] w3schools. What is json? https://www.w3schools.com/whatis/whatis_json.asp.

[36] Xiaowei Xing and Dong Eui Chang. Deep reinforcement learning based robot arm manipulation with efficient training data through simulation. *2019 19th International Conference on Control, Automation and Systems (ICCAS), Control, Automation and Systems (ICCAS), 2019 19th International Conference on*, pages 112 – 116, 2019.

[37] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-08-24

EXAMENSARBETE Camera Calibration for Alignment of a Real World Setup to a Simulated Environment
STUDENTER Ludwig Jakobsson, Oskar Larsson
HANDLEDARE Alexander Dürr (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Kamerakalibrering för justering av en verklig miljö till en simulerad miljö

POPULÄRVETENSKAPLIG SAMMANFATTNING **Ludwig Jakobsson, Oskar Larsson**

Användandet av Reinforcement Learning för att träna robotar är allt mer vanligt för att utföra uppgifter av allehanda slag. Detta arbete har skapat ett kalibreringssystem för att enklare kunna matcha den simulerade träningsmiljön med verkligheten.

Idag hanterar robotar många repetitiva tillverkningsprocesser. Den ökade processorkraften under de senaste åren har kraftigt accelererat möjligheten till användning av bildanalys och förstärkningsinlärning (Reinforcement Learning) för att lära robotarna att utföra sina uppgifter. För att undvika mekaniskt slitage och långa träningsperioder har träningen flyttats till en simulerad miljö som efterliknar den verkliga. Den simulerade miljön möjliggör att inlärningen kortas ner markant. För att säkerställa att träningen sker på rätt uppgifter är det viktigt att de båda miljöerna överensstämmer.

I vårat examensarbete har vi byggt ett kalibreringssystem för kameror som kan användas för att matcha en verklig miljö med en simulerad, men även för att underlätta mer generella uppgifter som att öppna dörrar och plocka upp mjölkpaket. Systemet strävar efter att vara platformsagnostiskt och fungera med olika kameror och robot armar. Systemet kalibreras genom att låta roboten hålla i ett kalibreringsbräde, som även syns i kameran, och relationen mellan roboten, brädet och kameran används för att estimera relationen mellan roboten och kameran.

Systemet verifieras med ett "plocka och släpp" test för att se om allt var rätt kalibrerat, samt ett test där robotens ändverktyg flyttas till en markörs centerpunkt. För att se vad som gav ett bra kalibreringsresultat, utfördes experiment med olika typer av data för att utforska hur det skulle kunna förbättras. Exempel på sådana data typer är: varierat avstånd mellan kamera och kalibreringsbrädet, hur pass varierade robotens ställningar var samt hur resultatet påverkas av att utjämna det avlästa förhållandet mellan kamera och kalibreringsbräda.

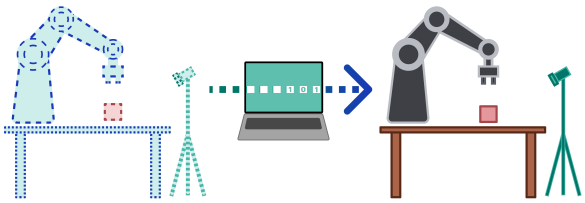


Fig. 1: Justering av en verklig miljö, för att matcha en simulerad.

Under projektets gång blev det tydligt att det vore väldigt omständigt att efterlikna den simulerade miljön med repeterade kalibreringar, så vi utvecklade ett alternativt system för att få snabbare återkoppling. Detta system använder en färdigkalibrerad kamera för att uppskatta en annan kameras plats med hjälp av ett kalibreringsbräde. Metoden behöver dock vidareutvecklas för att se praktisk användning.