Robotic Hand-Eye Calibration using Point Clouds

Markus Bjønnes

December 11, 2019

Abstract

Hand-eye calibration is an important step in implementing vision systems for robotics. The presented report investigates the use of point clouds generated from an accurate 3D scanner to calculate relative camera motion for hand-eye calibration purposes. A brief review of related research is presented in the introduction chapter. Findings suggest that hand-eye calibration methods using 3D point clouds are promising, but that the methods implemented were computationally expensive in comparison to conventional methods. In this project two methods for hand-eye calibration using point clouds are implemented and tested against the co-planar point calibration algorithm implemented in the OpenCV library. The 3D methods utilize color images of a chessboard to find distinct points in the image, a 3D point cloud is then extracted from these points and used for extrinsic camera calibration. This method of extracting 3D data for calibration is fast, since it does not depend on advanced matching algorithms like the ones utilizing a 3D calibration object. Simulations on generated noisy calibration datasets show that both methods implemented produce similar results, allowing the hand-eye calibration algorithm to converge. A laboratory robot cell consisting of a KUKA Agilus KR6 r900 sixx robot manipulator with a Zivid One structured light 3D scanner attached to its end-effector is used for hand-eye calibration experiments. Experimental results on datasets created using the laboratory setup show the 3D point cloud methods performing slightly better than the OpenCV algorithm, especially when the point cloud data being used has a low degree of noise.

Methods for tuning the parameters of the Zivid One 3D scanner to improve the quality of calibration datasets are also explored, resulting in improved performance of the 3D point cloud algorithms. Experiments using different calibration object setups and varying imaging strategies regarding the position of the camera in relation to the calibration board indicated that there exists an optimal distance for the specific 3D scanner used for the experiments.

Contents

1 Introduction **1**

1.1 Problem Description and Objectives 1
1.2 Related Work 1
1.3 Report Structure 3

2 Preliminaries on Robot Kinematics and Computer Vision **4**

2.1 Kinematic Modelling of Rigid Bodies 4
2.1.1 Pose of Rigid Bodies 4
2.1.2 Rotation Matrices 5
2.1.3 Elementary Rotations 6
2.1.4 Rotations as Linear Transformations 6
2.1.5 Euler Angles 7
2.1.6 Skew Symmetric Representation of a Vector 7
2.1.7 Rotations as Matrix Exponentials and Logarithms 8
2.1.8 Homogeneous Transformations 9
2.1.9 Forward Kinematics 9
2.2 Computer Vision 10
2.2.1 Pinhole Camera Model 10
2.2.2 Corner Detection 13
2.2.3 Camera Calibration 14
2.2.4 Point Clouds 14
2.2.5 Mathematical Description of Planes 14
2.2.6 Fitting a Plane to a Set of Points 15
2.2.7 Finding the Transformation Between two Point Clouds 15
2.2.8 Structured Light 3D Scanners 17

3 Hand-Eye Calibration **18**

3.1 Kinematics of the Hand-Eye Calibration Problem 18
3.2 A Solution to the Hand-Eye Calibration Problem by Solving $AX = XB$ on $SE(3)$ 20
3.2.1 Procedure for Hand-Eye Calibration 22

4 System Description **23**
4.1 Robot Manipulator . . . 23
4.2 Zivid One Camera . . . 25
4.3 Hand-Eye Calibration Software . . . 25
4.3.1 3D Point Cloud Calibration Method . . . 27
4.3.2 Calculation of Estimation Error . . . 28

5 Experiments **32**
5.1 Experiments . . . 32
5.1.1 Tuning 3D Camera Capture Parameters . . . 32
5.1.2 Hand-Eye Calibration Using Generated Noisy Data . . . 33
5.1.3 Hand-Eye Calibration on Real Datasets . . . 35
5.2 Results . . . 35
5.2.1 Camera Parameter Tuning . . . 35
5.2.2 Hand-Eye Calibration Using Generated Noisy Data . . . 38
5.2.3 Hand-Eye Calibration on Real Datasets . . . 43

6 Discussion **51**
6.1 Capture parameter tuning . . . 51
6.2 Hand-Eye Calibration on Generated noisy data . . . 52
6.3 Hand-Eye Calibration on Real datasets . . . 52

7 Conclusion and future work **55**
7.1 Conclusion . . . 55
7.2 Future Work . . . 56

Appendix A Calibration Chessboards **59**

Appendix B Code **63**
B.1 homogeneous_transformations.py . . . 63
B.2 utils.py . . . 64
B.3 zivid_hand_eye_calibrator.py . . . 64
B.4 capture.py . . . 65

Appendix C Design Drawings **66**

List of Figures

2.1 Coordinate frames $\{a\}$ and $\{b\}$. 6
2.2 Pinhole camera model [12] 11
2.3 Camera and world coordinate systems 12
2.4 Plane with normal vector, n 15
2.5 Illustration of triangulation principle for a structured light scanner. [15] . . 17

3.1 Robot with a camera mounted close to the end-effector, and an object with attached coordinate frame. 19
3.2 A camera mounted rigidly close to the robot end-effector [9] 20

4.1 KUKA Agilus KR6 r900 sixx, workspace [16] 24
4.2 Robot manipulator with mounted camera 24
4.3 End-effector assembly 25
4.4 Method and logic of the capture module. 30
4.5 Procedure for hand-eye calibration using the ZividHECalibrator class. . . . 31

5.1 Depth maps for images taken with default and tuned parameters. 37
5.2 Logarithmic intensity histograms images taken with default and tuned parameters. 37
5.3 Depth map from image taken of a calibration object with black squares. Tuned parameters from table 5.1 are used. 38
5.4 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.01)$ 38
5.5 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.04)$ 39
5.6 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.09)$ 39
5.7 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.16)$ 39
5.8 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.25)$ 40
5.9 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 1.0)$ 40
5.10 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 4.0)$ 40
5.11 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 9.0)$ 41
5.12 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 16.0)$ 41
5.13 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 25.0)$ 41
5.14 Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 100.0)$ 42

5.15 Scatter plot and trend line for noise standard deviations 0.1, 0.2, 0.3, 0.4, 0.5 43
5.16 Scatter plot and trend line for noise standard deviations 1.0, 2.0, 3.0, 4.0, 5.0, 10.0 43
5.17 Chessboard pose, estimated using OpenCV extrinsic camera calibration. The placement of the chessboard coordinate frame is defined in the camera calibration stage. Chessboard corners are marked using the drawChessboardCorners() function of OpenCV. 44
5.18 Chessboard pose, estimated using 3D point clouds for extrinsic camera calibration. Square center points are marked by blue dots. The chessboard coordinate frame is placed in the centroid of the point cloud comprised of the center points. 44
5.19 Visualization of robot end-effector poses relative to the base frame 45
5.20 Chessboard pose, estimated using 3D point clouds for extrinsic camera calibration. 45
5.21 Dataset1. Hand-eye calibration errors on images captured at a distance in the range 600-700 mm from the calibration object . 9×6 chessboard with square size of 20.0 mm fig. A.1. Camera settings as in table 5.2 46
5.22 Dataset 2. Hand-eye calibration errors on images captured at a distance of 600-800 mm from the calibration object. 9×6 chessboard with square size of 20.0 mm. fig. A.1. Camera settings as in table 5.1 46
5.23 Dataset 3. Hand-eye calibration errors on images captured at a distance of 750-950 mm from the calibration object. 21×14 chessboard with square size of 13.8 mm. Gray value 180 fig. A.2. Camera settings as in table 5.1 . 47
5.24 Dataset 4. Hand-eye calibration errors on images captured at a distance of 800-1000 mm from the calibration object. 20×13 chessboard with square size of 13.9 mm. Gray value 130fig. A.3. Camera settings as in table 5.2 . . 48
5.25 Dataset 5. Hand-eye calibration errors on images captured at a distance of 800-1200 mm from the calibration object. 20×13 chessboard with square size of 13.9 mm. Gray value 130 fig. A.3. Camera settings as in table 5.2 . 48
5.26 Visualization of the hand-eye transformation in (5.4) resulting from hand-eye calibration on dataset 3. The robot end-effector is denoted "EF" and the camera center is denoted "c0" 49
5.27 Rough estimate of the position of the camera center, illustrated on a 3D model of the end-effector assembly. 50

A.1 9×6. Square size 20.0 mm. [18] 60
A.2 21×14. Gray value 180. 61
A.3 20×13. Gray value 130. 62

List of Tables

4.1 KUKA Agilus KR6 r900 sixx, technical specifications 23
4.2 Overview of files used for hand-eye calibration 26

5.1 Manually tuned camera parameters. 36
5.2 Automatically tuned camera parameters. 36
5.3 Final translation errors and rotation errors for 3D and 3D plane fit methods. 42
5.4 Final errors on Dataset 1. 46
5.5 Final errors on Dataset 2. 47
5.6 Final errors on Dataset 3. 47
5.7 Final errors on Dataset 4. 48
5.8 Final errors on Dataset 5. 49

Chapter 1

Introduction

1.1 Problem Description and Objectives

In manufacturing environments with a high degree of automation, one particularly challenging task is the handling of randomly positioned and orientated objects by robotic manipulators. This task is dependent on the utilization of computer vision technologies for accurate estimation of the six degrees of freedom pose of objects in 3D space, in consort with robust trajectory control of the robot manipulator. When the pose of the object has been determined in relation to the coordinate system of the camera it is necessary to calculate the position of the object in relation to the base coordinate frame of the robot in order to manipulate it. The transformation between the robot end-effector frame and the camera frame is called the hand-eye transformation, and is found by means of hand-eye calibration. Hand-eye calibration is the focus of this project.

With the introduction of high accuracy 3D imaging equipment used for point cloud generation, it is necessary to investigate whether this accurate position data can be utilized to increase the accuracy of hand-eye calibration compared with conventional methods. To this end the main objectives of this project are:

1. Design and assemble a bracket for mounting the 3D scanner on a robot arm.
2. Design and implement a software system for hand-eye calibration utilizing point cloud data of a scene captured by a 3D scanner.
3. Evaluate the accuracy of the hand-eye calibration when using different methods for calculation of relative camera poses.

1.2 Related Work

Work on solving the hand-eye calibration problem began in the late 1980s. At that time digital cameras were becoming good enough to be utilized in computer vision tasks for

robotics. The hand-eye calibration problem was first formulated as a system of homogeneous matrix equations on the form $AX = XB$ where A represents robot motion and B represents camera motion. Shiu and Ahmad [1] found a closed form solution to this problem which satisfied the conditions for uniqueness. The rotation determination problem was solved by solving a set of eight linear equations with four unknowns. Tsai and Lenz [2] proposed a different solution to the same problem formulation. They solved the rotational part of the calibration equation by represented the rotation by its unit eigenvector n_x and an angle θ_x. The solution is formulated in a way that it can yield an exact closed form solution when two pose pairs are used, or a least squares solution if more pose pairs are used. Many pose pairs are used to help mitigate the effect of noise in measurements. Their solution is tested on both generated noisy data and on real robot/camera pose pairs. The camera positions are estimated using the extrinsic calibration methods introduced in [3]. Another solution which also generalizes to a linear least squares solution when noise is present is proposed by Park and Martin [4]. All these solutions decouple the rotation and translation parts of the formulated system of equations. First the solution for the rotational part of the hand-eye calibration matrix is found, then this is used to solve a set of vector equations to find the translation part. This is a good idea because it yields a relatively simple numerical optimization problem, but the linearisation on of the rotation problem can become ill conditioned in the presence of noise. Some of these issues were mitigated in the solution proposed by Horaud and Fadi [5]. They proposed a formulation to the hand-eye calibration problem on the form of $MY = M'YB$ where the 3×4 perspective matrices M and M' of the camera in two different positions are used in stead of an explicit formulation of the extrinsic and intrinsic parameters. Two different solutions to this new formulation is presented. One closed form solution using linear least squares for optimization, and a non-linear solution using the Levenberg-Marquardt method to optimize the hand-eye transformation matrix. The second solution mitigates errors associated with decoupling rotation and translation, by optimizing both simultaneously.

The authors of [6] use time-of-flight and structured light 3D scanners to find the extrinsic parameters of the sensor by matching a the pose of a 3D model with captured 3D data. This was then used for hand-eye calibration using the method introduced by Tsai and Lenz. Their findings suggest that using their method of finding the extrinsic parameters of the cameras resulted in slightly more accurate hand-eye calibration results compared with the 2D image based method of extrinsic calibration. The 3D method did however require a more labour intensive process for dataset gathering, and was significantly more computationally expensive than the 2D image method.

1.3 Report Structure

The report is structured as follows. In Chapter 2 the necessary pre-requisites related to robotics and computer vision are presented in order to provide the reader with the required theoretical background. In Chapter 3 the kinematics of the hand-eye calibration problem and a solution based on [4] is presented, as well as the general procedure for hand-eye calibration of a robot utilizing a vision system. In Chapter 4 the setup used for experiments related to hand-eye calibration is presented. In Chapter 5 the experimental results are presented, these are discussed in Chapter 6. Chapter 7 contains the conclusion and suggested future work.

Chapter 2

Preliminaries on Robot Kinematics and Computer Vision

This chapter will provide the reader with the necessary theoretical knowledge regarding methods used for modelling the kinematics of rigid bodies in 3D space, as well as the fundamentals used in computer vision systems. The theoretical concepts presented in this chapter is gathered from several textbooks as well as papers published in widely accepted journals [7], [8], [9], [10], [11].

2.1 Kinematic Modelling of Rigid Bodies

The kinematics of rigid bodies is a central concept used in modelling and control of robot manipulators. It provides a powerful tool for representing the position and orientation, or pose, of rigid bodies in three dimensional space. The term robot manipulator is used as a general term meaning a robotic arm. Robotic arms consist of rigid links connected together by joints. These joints can vary in type, but the most common ones are revoulute with one rotational degree of freedom and prismatic joints with one translational degree of freedom.

2.1.1 Pose of Rigid Bodies

Rigid bodies in 3D space have three positional degrees of freedom and three rotational degrees of freedom, adding up to a total of six degrees of freedom. Having a robust mathematical representation for the pose of rigid bodies is fundamental for the development of methods used in modelling kinematic chains. This representation is based on matrices in the special orthogonal group, $SO(3)$ and the special Euclidean group, $SE(3)$.

2.1.2 Rotation Matrices

Rotation matrices can be used to represent the difference in orientation of a coordinate system $\{a\}$ and a rotated coordinate system $\{b\}$. Coordinate frames in 3D space are represented as 3×3 matrices with each column being a unit vector. This set of column vectors make up an orthogonal basis for $\mathbb{R}^3$. The axes of the reference coordinate system can be represented as the columns of the identity matrix in $\mathbb{R}^{3\times3}$.

$$\{a\} = \begin{bmatrix} x_a & y_a & z_a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The rotated frame $\{b\}$ is obtained by applying the linear transformation R_{ab}, representing the rotation from frame $\{a\}$ to frame $\{b\}$. The elements of the rotation matrix R_{ab} is as shown below.

$$R_{ab} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{2.2}$$

Each column of the matrix representing coordinate frame $\{b\}$ are the the directional unit vectors of the the axes of frame $\{b\}$ given in the coordinates of frame $\{a\}$. $SO(3)$ has the following definition:

$$R \in SO(3) = \{R \in \mathbb{R}^{3\times3}, R^T R = I^{3\times3}, \det(R) = 1\} \tag{2.3}$$

From this it follows that

$$R^T = R^{-1} \tag{2.4}$$

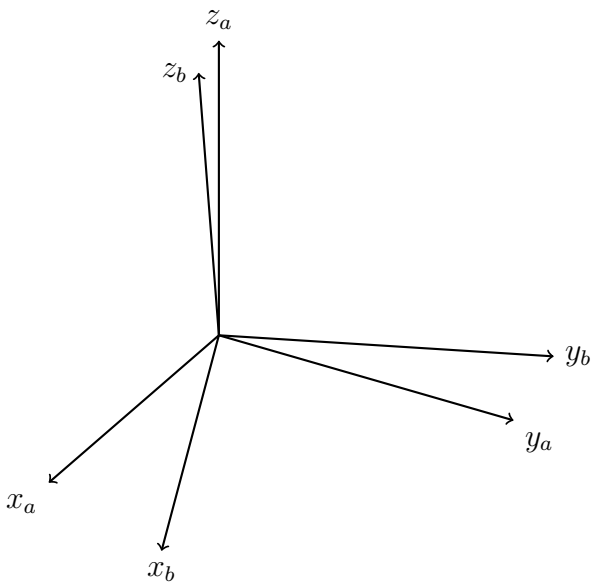


Figure 2.1: Coordinate frames $\{a\}$ and $\{b\}$.

2.1.3 Elementary Rotations

Elementary rotations are defined as rotations by an angle θ about one of the three principal axes of a coordinate frame. Elementary rotations about the z, y, and x axes are presented below.

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.6}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.7}$$

2.1.4 Rotations as Linear Transformations

Given the coordinate frames $a,\ b,\ c$ where the orientation of frame b in the coordinates of frame a is defined by the rotation matrix R_{ab}, and the orientation of frame c in the coordinates of frame b is defined by the rotation matrix R_{bc}, we can obtain the orientation of frame c in the coordinates of frame a as the rotation matrix R_{ac} composed from the intermediate rotations R_{ab} and R_{bc} as $R_{ac} = R_{ab}R_{bc}$. The interpretation of this operation is that frame a is first rotated to frame b by R_{ab}, frame b is then rotated by R_{bc} to frame c.

It is important to note that matrix multiplication operations for matrices in $SO(3)$ are associative: $(R_1R_2)R_3 = R_1(R_2R_3)$, but not commutative: $(R_1R_2 \neq R_2R_1)$. The interpretation of pre- or post multiplication by a rotation matrix is that pre-multiplcation results in a rotation about the current body frame, while a post-multiplication results in a rotation about a fixed reference frame.

2.1.5 Euler Angles

As there are three degrees of rotational freedom in 3D space, the parametrization of the orientation of a rigid body can be represented by three independent rotation angles. This parametrization is referred to as Euler angles. It is possible to construct several different representations of the orientation of a rigid body using Euler angles e.g. ZYZ, XYZ, XZX and ZYX, where X, Y and Z represent elementary rotations about the x, y and z axes respectively. ZYX-Euler angles are derived below.

Given a rigid body with orientation defined by the body frame b, initially aligned with the fixed space frame s. The orientation of frame b in the coordinates of frame a are parametrized by the triplet of angles (α, β, γ) in the following rotation operations.

- The rotation about the z-axis of the fixed frame s by the angle α, resulting in the body frame s'
- The rotation about the y-axis of the body frame s' by the angle β, resulting in the body frame s''
- The rotation about the x-axis of the body frame s'' by the angle γ, resulting in the body frame b

$$
\begin{gathered}
R_{sb} = R_z(\alpha)R_y(\beta)R_x(\gamma) = \\
\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} = \\
\begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix}
\end{gathered}
\tag{2.8}
$$

Where $c_\theta = \cos\theta, s_\theta = \sin\theta$ are used for the convenience of a more compact notation.

2.1.6 Skew Symmetric Representation of a Vector

A skew symmetric matrix $S \in \mathbb{R}^{n\times n}$ satisfies the condition that $S^T = -S$, meaning that the transpose of the matrix is equal to its negative. Given a vector $v \in \mathbb{R}^3$, a skew

symmetric matrix can be constructed from v by (2.9).

$$v^{\times} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \tag{2.9}$$

2.1.7 Rotations as Matrix Exponentials and Logarithms

The matrix exponential coordinates is a parametrization of a rotation matrix by a vector $k\theta \in \mathbb{R}^3$ where k is the unit vector representing the axis of rotation in $\mathbb{R}^3$ and θ is a scalar value representing the angle of rotation about the axis k. Writing θ and k separately gives the angle-axis representation of the rotation matrix. The matrix logarithm is represented by the skew symmetric matrix $k^{\times}\theta$, and is a member of the Lie algebra, denoted $so(3)$, of the special orthogonal group, $SO(3)$. The elements of the lie algebra $so(3)$ can be interpreted as the tangent space of the manifold $SO(3)$ at the identity element. The matrix logarithm is related to the rotation matrix by:

$$k^{\times}\theta \in so(3) \rightarrow R \in SO(3) : e^{k^{\times}\theta} = R \tag{2.10}$$

The matrix exponential, $e^{k^{\times}\theta}$, is calculated by Rodrigues' formula (2.11).

$$R(k,\theta) = I + \sin(\theta)k^{\times} + (1 - \cos(\theta))k^{\times}k^{\times} \tag{2.11}$$

where I is the identity matrix in $\mathbb{R}^{3\times 3}$. Given a rotation represented by $R \in SO(3)$ the matrix exponential coordinate representation can be found by the following relation:

$$SO(3) \rightarrow so(3) : \; R \mapsto \log(R) = k^{\times}\theta \tag{2.12}$$

The angle of rotation θ and the axis of rotation k are calculated from (2.13) and (2.14).

$$\theta = \cos^{-1}\left(\frac{\text{trace}(R) - 1}{2}\right) \tag{2.13}$$

$$k = \frac{1}{2\sin\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{2.14}$$

Where trace(R) is the sum of the elements on the principal diagonal of R and r_{ij} are the elements of R. Note that this holds only when trace$(R) \neq -1$ and trace$(R) \neq 3$, as it would result in $\sin\theta = 0$.

2.1.8 Homogeneous Transformations

Homogeneous transformation matrices in the special Euclidean group $SE(3)$ are the set of all 4×4 matrices on the form

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

where $R \in SO(3)$ and $t \in \mathbb{R}^3$. Multiplication with homogeneous transformation matrices is associative: $(T_1T_2)T_3 = T_1(T_2T_3)$, but not commutative: $T_1T_2 \neq T_2T_1$. Homogeneous transformation matrices have three important uses.

- Represent the orientation and position of a rigid body.
- Change the reference frame in which a vector or frame is represented.
- Displace a vector or frame.

The inverse of a matrix $T \in SE(3)$ is also in $SE(3)$, this also holds for the product of two matrices in $SE(3)$. The inverse of a homogeneous transformation matrix can be calculated from:

$$T^{-1} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix} \tag{2.16}$$

2.1.9 Forward Kinematics

Forward, or direct, kinematics are used to calculate the position and orientation of the robot end-effector based on the joint configuration. For a robot with n variable joints the joint configuration vector q is written as

$$q = [q_1,\ q_2, ...,\ q_n] \tag{2.17}$$

where q_1 is the joint parameter of joint 1, q_2 is the joint parameter for joint 2 an so on. The forward kinematics of a kinematic chain can be expressed as the product of homogeneous transformation matrices, each depending on a single joint variable q_i.

$$T_{0n}(q) = T_{01}(q_1)T_{12}(q_2)...T_{(n-1)n}(q_n) \tag{2.18}$$

Where $T_{(i-1)i}$ is the transformation from the coordinate frame attached to joint number $i-1$ to the coordinate frame attached to joint number i. The resulting matrix T_{0n} is the transformation from the coordinate frame attached to joint 1 to the coordinate frame attached to joint n.

The pose of the robot end-effector frame $\{t\}$ relative to the base frame $\{b\}$ is obtained from

$$T_{bt} = T_{b0}T_{0n}(q)T_{nt} \tag{2.19}$$

Where T_{b0} and T_{nt} are constant transformations. The transformation from the last robot joint to the end-effector frame T_{nt} has to be calibrated when using different end-effectors. There are several different methods used for deriving the forward kinematics of a robotic manipulator, such as the Denavit-Hartenberg (DH) convention [11] and the method using products of exponentials based on joint twists. The method of deriving the forward kinematics of an open chain manipulator based on the DH-convention based on is presented below.

The DH-convention uses four parameters $(a_i, d_i, \alpha_i, \theta_i)$ to define the pose of the frame attached to joint i in reference to the frame attached to joint $i-1$. The parameters are assigned when the manipulator is in a pre-defined zero-position, meaning that all the joint variables are set to zero ($q = [0,\ 0, ...,\ 0]$). a_i and d_i are the respective translations along the x_{i-1} and z_{i-1} axes from the origin of frame $i-1$ to the origin of frame i. α_i is the rotation about the x_{i-1} axis and θ_i is the rotation about the z_{i-1} axis. For revolute joints the variable $q_i = \theta_i$ and for prismatic joints the joint variable is $q_i = d_i$. The joint transformation between joint $i-1$ and i is a function of the joint variable q_i and is calculated as

$$T_{(i-1)i}(q_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i cos\alpha_i & -\cos\theta_i sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

2.2 Computer Vision

Making use of cameras in robotics systems enables the robot to interact autonomously with its environment. Some of the central methods and models in computer vision are presented in this section.

2.2.1 Pinhole Camera Model

The pinhole camera model shown in Figure 2.2 is commonly used for modelling a real camera for use in computer vision applications. It models a simple camera where light is emitted through a pin-hole in the camera center, this simplification means that each point in the scene is projected as a single point in the image plane. Given a camera coordinate frame denoted $\{c\}$ and a world coordinate frame denoted by $\{w\}$, as shown in Figure 2.3. A point p in the scene with homogeneous coordinates given in the the camera frame is

defined as

$$\tilde{r}_{cp}^{c} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{2.21}$$

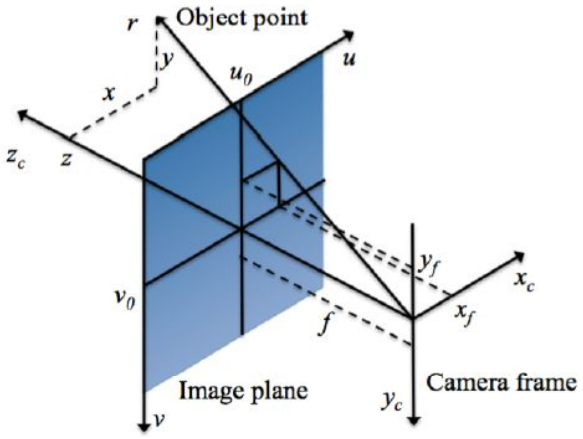


Figure 2.2: Pinhole camera model [12]

and the position of the same point in the world coordinate frame is

$$\tilde{r}_{wp}^{w} = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.22}$$

the transformation of reference frames for this point is given by

$$\tilde{r}_{cp}^{c} = T_{cw}\tilde{r}_{wp}^{w} \tag{2.23}$$

where T_{cw} is the homogeneous transformation matrix representing the transformation from $\{c\}$ to $\{w\}$.

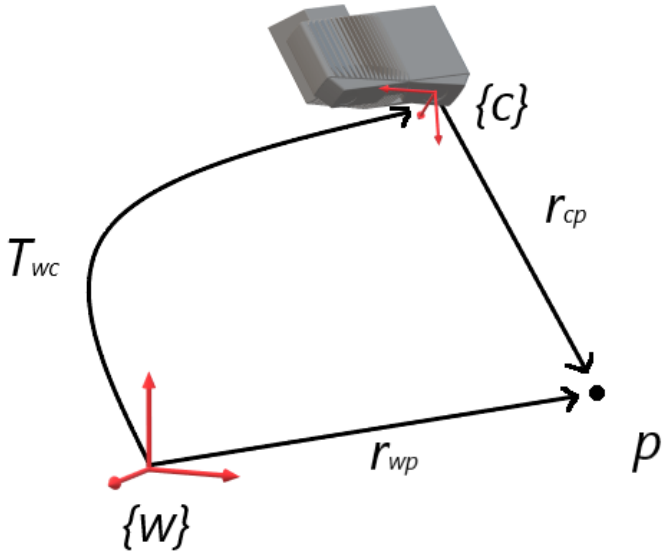


Figure 2.3: Camera and world coordinate systems

The normalized coordinates of the point represented by r^c_{cp} projected onto the image plane is given by

$$\tilde{s} = \begin{bmatrix} s_x \\ s_y \\ 1 \end{bmatrix} = \frac{1}{z_c} r^c_{cp} = \frac{1}{z_c} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} \frac{x_c}{z_c} \\ \frac{y_c}{z_c} \\ 1 \end{bmatrix} \tag{2.24}$$

The conversion between homogeneous and non-homogeneous vector coordinates can be done by

$$r = \Pi \tilde{r} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{2.25}$$

The normalized image coordinates can now be calculated as

$$\tilde{s} = \frac{1}{z_c} \Pi \tilde{r}^c_{cp} = \frac{1}{z_c} \Pi T_{cw} \tilde{r}^w_{wp} \tag{2.26}$$

The corresponding pixel coordinate values are calculated from the following.

$$u = \frac{f}{\rho_w} s_x + u_0 \tag{2.27}$$

$$v = \frac{f}{\rho_h} s_x + v_0 \tag{2.28}$$

Here f is the focal length of the camera, ρ_w and ρ_h is the width and height of each pixel, and u_0 and v_0 is the image plane center coordinates. Conversion from normalized image coordinates to homogeneous pixel coordinates is possible using.

$$\tilde{p} = K\tilde{s} \tag{2.29}$$

Where K is the camera intrinsic parameter matrix.

$$K = \begin{bmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

Equations (2.26) and (2.29) can be combined to the projective camera transformation.

$$z_c\tilde{p} = K\Pi T_{cw}\tilde{r}^w_{wp} \tag{2.31}$$

Which enables the calculation of pixel coordinates corresponding to a point $\tilde{r}^w_{wp}$, given the relative transformation between $\{c\}$ and $\{w\}$. This formulation is the basis for solving of the inverse problem where the pixel values of a point is known, and the position in the coordinates of the fixed world frame $\{w\}$ is desired.

2.2.2 Corner Detection

In computer vision applications, the ability to detect distinct points on an object reliably under varying lighting conditions and from different viewing angles is very important for several applications including calibration, tracking and pose estimation. A corner can be formalized as the point at which two edges meet. Therefore an important step in any corner detection algorithm requires the detection of edges in an image.

Edges are usually found by finding the boundaries where the gradient of pixel intensity and color is abrupt. One way of doing this is by convolving the image with a gradient finding patch for x, and y directions. For instance the Sobel edge detector uses the patches in (2.32) and (2.33) to find the points at which the image gradient is large in the x and y directions respectively.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.32}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.33}$$

The edges found by this method can be used to efficiently find corners in an image using

a variety of different corner detection algorithms like Harris [10], CSS [13] or SUSAN [14].

2.2.3 Camera Calibration

Intrinsic camera calibration is the process of establishing the camera's intrinsic parameters. The model used in Section 2.2.1 is simplified in the sense that the light passes through a pin-hole directly to the sensor. In a real camera, lenses focus the light in a way not accounted for in the pin-hole model, this needs to be corrected for by finding a set of distortion coefficients. The calibration parameters obtained by intrinsic calibration are the ones in (2.30) accounting for focal distance, principal distance and the coordinates of the principal point. The distortion coefficients account for radial and tangential distortion, these are used to remove distortion effects from the image.

Extrinsic camera calibration solves the problem of estimating the pose of the camera in relation to the calibration object. This is a required step in the two step camera calibration algorithm presented in [3]. This algorithm provides efficient camera calibration based on coplanar calibration points. The extrinsic camera calibration is also central in solving the hand-eye calibration problem.

2.2.4 Point Clouds

A point cloud can be defined as a set consisting of n points having positions in 3D space $p_i = [x_i, y_i, z_i]^T$. For generation of point clouds the accurate estimation of the distance of a point from the camera along the optical axis, z_i, is critical as this allows for calculation of the 3D position of the point relative to the camera using (2.31). This distance can be estimated using a variety of methods such as stereo vision, time-of-flight range finders or structured light 3D scanners. Point clouds generated using these methods will often have an associated image color intensity for each point, $(I_{rgb})_i = [r_i, g_i, b_i]^T$, resulting in the notation $[(I_{rgb})_i,\ \ p_i]$ for each point in the image.

2.2.5 Mathematical Description of Planes

A plane $\pi = [a, b, c, d]^T$, with normal vector $n = [a, b, c]^T$ at a distance $\frac{d}{|n|}$ from the origin, is defined as the set of all points $p = [x, y, z]^T$ satisfying

$$ax + by + cz + d = 0 \tag{2.34}$$

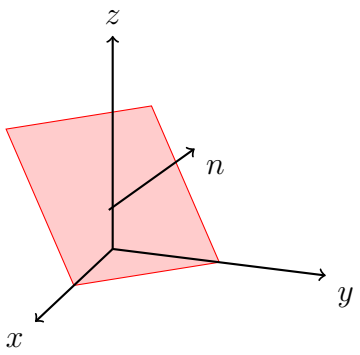


Figure 2.4: Plane with normal vector, n

2.2.6 Fitting a Plane to a Set of Points

If a homogeneous point $\tilde{p}_i = [x_i, y_i, z_i, 1]^T$ is on the plane π, the following holds.

$$\tilde{p}_i^T \pi = 0 \tag{2.35}$$

This property can be used to find the best fit plane corresponding to a set of n homogeneous points. Following the derivation in [12], define a matrix $A = [p_1,\ p_2, ...,\ p_n]^T \in \mathbb{R}^{n\times 4}$. The plane π must satisfy

$$A\pi = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_n^T \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0 \tag{2.36}$$

The singular value decomposition of A is

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \sigma_3 u_3 v_3^T + \sigma_4 u_4 v_4^T \tag{2.37}$$

where $\sigma_1 > \sigma_2 > \sigma_3$ and $\sigma_4 = 0$. The only non-trivial solution for the plane π is along v_4 giving the solution

$$\pi = k v_4 \tag{2.38}$$

for some scale factor k. If the points in A are all on the plane, the solution is exact, otherwise it is the best fit solution minimizing the absolute orthogonal distance from each point to the plane.

2.2.7 Finding the Transformation Between two Point Clouds

Given two point clouds, $A \in \mathbb{R}^{n\times 4}$ and $B \in \mathbb{R}^{n\times 4}$, with n corresponding homogeneous points denoted a_i and b_i respectively the optimal transformation between them can be found by the minimization problem in (2.42).

The correspondence between the point clouds is formulated as

$$B = TA \tag{2.39}$$

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in SE(3) \tag{2.40}$$

Which can be written as

$$b_i = Ra_i + t \tag{2.41}$$

The expression to be minimized is the mean sum of square errors between the points in B and the transformed points TA.

$$\eta = \frac{1}{n}\sum_{i=1}^{n} \|Ta_i - b_i\|^2 = \frac{1}{n}\sum_{i=1}^{n} \|Ra_i + t - b_i\|^2 \tag{2.42}$$

The optimal rotation is found by centring the two point clouds in the origin by subtracting the centroid (2.43) of the point cloud from each point, then finding the best fit rotation by using the solution to orthogonal Procrustes problem (2.47), which maximizes (2.44).

$$c_A = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad c_B = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{2.43}$$

$$\underset{R}{max}\ \text{trace}(RH) \tag{2.44}$$

$$H = (A - c_A)(B - c_B)^T \tag{2.45}$$

The optimal rotation is then found ny (2.47) by using the singular value decomposition of H

$$U\Sigma V^T = \text{svd}(H) \tag{2.46}$$

$$R = VSU^T \tag{2.47}$$

Where S is the Umeyama correction (2.49) to ensure that R is on $SO(3)$. The optimal translation is then found from

$$t = -Rc_A + c_B \tag{2.48}$$

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(VU^T) \end{bmatrix} \tag{2.49}$$

It should be noted that this translation is dependent on the solution for $R \in SO(3)$, which means that the resulting translation is dependent on the optimization of the rotational problem in (2.44).

2.2.8 Structured Light 3D Scanners

Structured light 3D scanners is a category of depth sensors for imaging purposes which utilizes triangulation of a known light pattern to determine depth. The scanner consists of a light projector and a offset camera with known relative position and orientations to one another. The light is projected onto the scene and the deformation in the light pattern is used to triangulate the distance to the object on which the light shines.

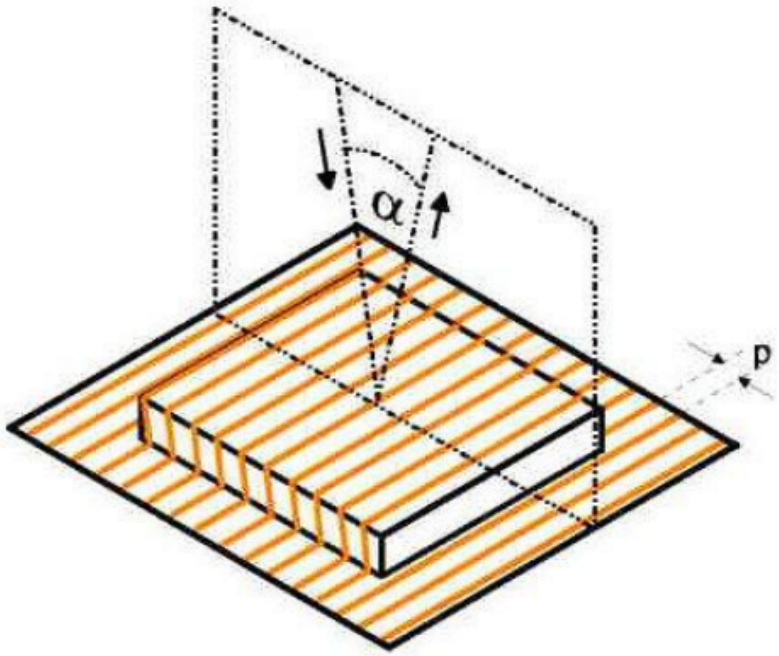


Figure 2.5: Illustration of triangulation principle for a structured light scanner. [15]

Using the principle illustrated in Figure 2.5 the distance z from the camera to the object is calculated from the angle α and the displacement of projected lines p as

$$z = \frac{p}{\tan \alpha} \tag{2.50}$$

Chapter 3

Hand-Eye Calibration

In this chapter the kinematics of the hand eye calibration problem will be presented. A solution to the hand eye calibration problem will be derived following the method in [4]. In order for a robotic manipulator to interact with its environment using vision systems, the pose of the camera relative to the robot end-effector frame must be found. This is to be able to transform the estimated 6D object pose of an object in the camera frame to obtain the 6D pose of this object in the robot base-frame.

3.1 Kinematics of the Hand-Eye Calibration Problem

Given a robot configuration as shown in Figure 3.1, with the following coordinate frames defined: robot base frame $\{b\}$, robot end-effector/tool frame $\{t\}$, a camera frame $\{c\}$ and an object frame $\{o\}$, the objective is to determine the pose of the object frame in the coordinates of the robot base frame. The transformation from the $\{b\}$ frame to the $\{t\}$ frame, T_{bt}, is found by using the forward kinematics of the robot manipulator, and the transformation from the $\{c\}$ frame to the $\{o\}$ frame, T_{co}, is found by a pose estimation algorithm. The transformation from the $\{t\}$ frame to the $\{c\}$ frame, T_{tc}, is calculated by hand-eye calibration. The notation $X = T_{tc}$ is often used for the unknown transformation between $\{t\}$ and $\{c\}$. Having determined all these transformations the pose of the object in the robot base frame, T_{bo} can be calculated by

$$T_{bo} = T_{bt}T_{tc}T_{to} \tag{3.1}$$

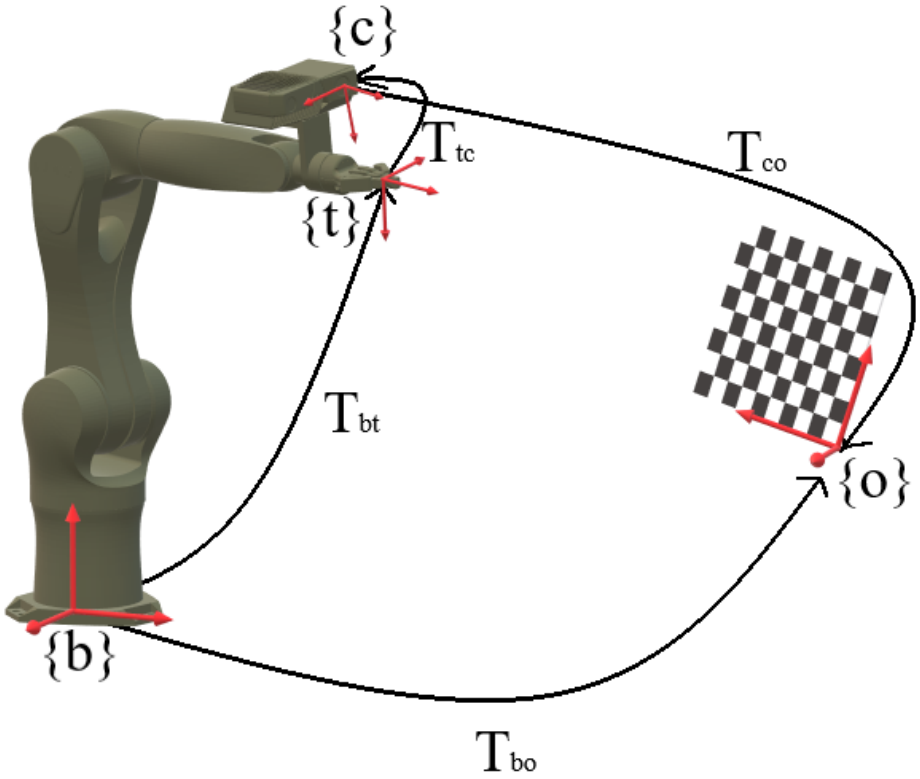


Figure 3.1: Robot with a camera mounted close to the end-effector, and an object with attached coordinate frame.

To find the unknown camera calibration X pairs of robot-camera configurations are used, as shown in figure 3.2. In the first configuration of the pose pair, the pose of the tool frame in $\{b\}$ and the pose of the object frame in $\{c\}$ is denoted A_{1a} and B_{1a} respectively. In the second configuration of the pose pair the poses are denoted A_{1b} and B_{1b}. The convention used is that the capital letter is used to distinguish between tool and object pose, the number is used to tell the pose pairs apart, and the lower case letters are used to differentiate between the first and second configuration in a specific pose pair.

Since the pose of the object in the robot base frame, T_{bo}, is the same for all pose configurations we can insert the pose pairs in to (3.1) to get the following equation: (3.2).

$$A_{1a}XB_{1a} = A_{1b}XB_{1b} \tag{3.2}$$

Pre-multiplying both sides of eq. 3.2 with A_{1b}^{-1} and post-multiplying both sides with B_{1b}^{-1} yields eq. 3.3, 3.4.

$$A_{1b}^{-1}A_{1a}X = XB_{1b}B_{1a}^{-1} \tag{3.3}$$

$$A_1 X = X B_1 \tag{3.4}$$

Where A_1, B_1 are the relative transformations between the two poses in pose pair 1. These transformations are shown as A and B in figure 3.2. In order to find a unique solution for X at least two pose pairs must be used. To reduce the impact of noisy measurements it is usual to use many (n) pose pairs, yielding a set of n equations (3.5).

$$\begin{aligned} A_1 X &= X B_1 \\ A_2 X &= X B_2 \\ A_3 X &= X B_3 \\ &\dots \\ A_{n-1} X &= X B_{n-1} \\ A_n X &= X B_n \end{aligned} \tag{3.5}$$

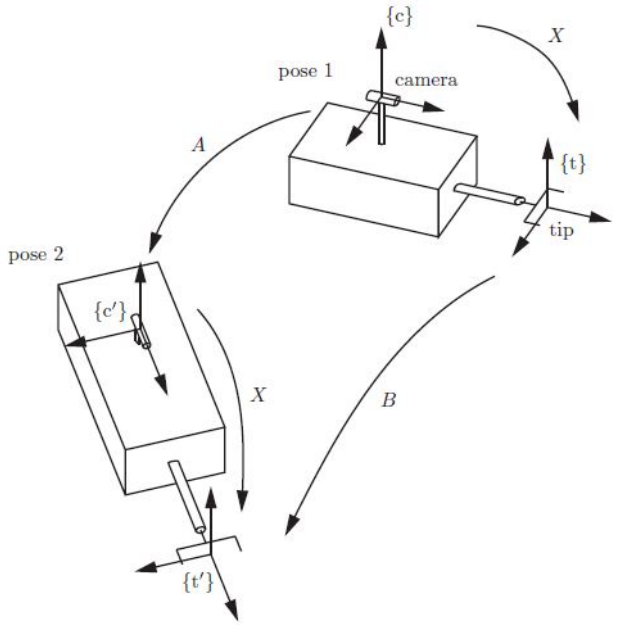


Figure 3.2: A camera mounted rigidly close to the robot end-effector [9]

3.2 A Solution to the Hand-Eye Calibration Problem by Solving $AX = XB$ on $SE(3)$

The derivation of the least squares solution to the set of equations presented in (3.5), follows the method developed by Park and Martin[4]. This solution splits the problem of solving for the optimal orientation and the optimal translation between the tool-frame

and the camera-frame into to separate problems. First the optimal rotation problem on $SO(3)$ is solved, this is then used to find the optimal translation. The combined result is the optimal transformation on $SE(3)$.

The matrix form of equation 3.4 is (3.6), which can be written as the pair of equations (3.7) and (3.8).

$$\begin{bmatrix} R_a & t_a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_x & t_x \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_x & t_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_b & t_b \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

$$R_a R_x = R_x R_b \tag{3.7}$$

$$R_a t_x + t_a = R_x t_b + t_x \tag{3.8}$$

The best fit solution for the unknown rotation R_x between frame $\{t\}$ and frame $\{c\}$ minimizes η_1.

$$\eta_1 = \sum_{i=1}^{n} \|R_x log(R_{bi}) - log(R_{ai})\|^2 = \sum_{i=1}^{n} \|R_x(\theta_b k_b)_i - (\theta_a k_a)_i\|^2 \tag{3.9}$$

This minimization problem is equivalent to solving for the R_x which maximizes

$$\text{trace}(R_x K_b K_a^T) \tag{3.10}$$

Where $K_a = [(\theta_a k_a)_1, ..., (\theta_a k_a)_n] \in \mathbb{R}^{3\times n}$ and $K_b = [(\theta_b k_b)_1, ..., (\theta_b k_b)_n] \in \mathbb{R}^{3\times n}$. The optimal R_x is then found by the singular value decomposition of the matrix product $K_b K_a^T$ and forming R_x as in (3.12), where S the Umeyama correction (2.49) to ensure that R_x is a member of $SO(3)$.

$$U\Sigma V^T = \text{svd}(K_b K_a^T) \tag{3.11}$$

$$R_x = VSU^T \tag{3.12}$$

The optimal translation is found as the vector t_x which minimizes η_2.

$$\eta_2 = \sum_{i=1}^{n} \left\|(R_{ai} - I^{3\times 3})t_x - R_x t_{bi} + t_{ai}\right\|^2 \tag{3.13}$$

The solution to this minimization problem is formulated as the least square solution to the matrix equation

$$Ct_x = d \tag{3.14}$$

$$C = \begin{bmatrix} R_{a1} - I^{3\times3} \\ R_{a2} - I^{3\times3} \\ ... \\ R_{an} - I^{3\times3} \end{bmatrix}, \quad d = \begin{bmatrix} R_x t_{b1} + t_{a1} \\ R_x t_{b2} + t_{a2} \\ ... \\ R_x t_{bn} + t_{an} \end{bmatrix} \tag{3.15}$$

The optimal translation is then found as

$$t_x = (C^T C)^{-1} C d \tag{3.16}$$

Where $(C^T C)^{-1} C$ is the pseudo-inverse of C.

The resulting hand eye calibration matrix representing the tool-to-camera transformation is

$$T_{tc} = \begin{bmatrix} R_x & t_x \\ 0 & 1 \end{bmatrix} \tag{3.17}$$

3.2.1 Procedure for Hand-Eye Calibration

The basic procedure for hand-eye calibration consists of the following steps.

1. Create a calibration dataset consisting of robot end-effector poses and images taken of the calibration object in different robot configurations.

2. Conduct extrinsic camera calibration for each robot configuration in order to find the pose of the calibration object in the camera frame for each image.

3. Create pose pairs and calculate relative camera and robot motion for each pose pair to obtain a set of equations as in (3.5).

4. Calculate the best fit transformation between the robot end-effector frame and the camera frame by solving this set of equations.

Chapter 4

System Description

For the purposes of verifying the functionality of the implemented hand eye calibration methods, a robot manipulator with an in-hand mounted 3D camera is used. This chapter describes the hardware and software components comprising the robot cell, and the hand eye calibration tools used.

4.1 Robot Manipulator

The robot manipulator used in this project is the KUKA Agilus KR6 r900 sixx. The technical specifications for the manipulator found in [16] are presented in Table 4.1 and the operational workspace of the manipulator as well as the dimensions of each link are shown in Figure 4.1.

The end-effector assembly is pictured in Figure 4.3. The camera mounting-bracket is fixed onto the robot end-effector flange and is designed with the required clearances as to not limit the movement of the robot end-effector. Drawings for the mounting-bracket are found in appendix C. The end-effector tool is mounted onto the camera mount. This is a Robotiq 2D-85 two finger gripper with a maximum grip width of 85 mm. The use of this gripper is outside the scope of this project, however the grip-center of the tool is the reference for the tool-to-camera transformation (3.17) attained from the hand eye calibration.

Maximum reach	901.5 mm
Maximum payload	6 kg
Pose repeatability	$\pm$0.03 mm
Number of axes	6
Footprint	320 mm $\times$ 320 mm
Weight	approx. 52 kg

Table 4.1: KUKA Agilus KR6 r900 sixx, technical specifications

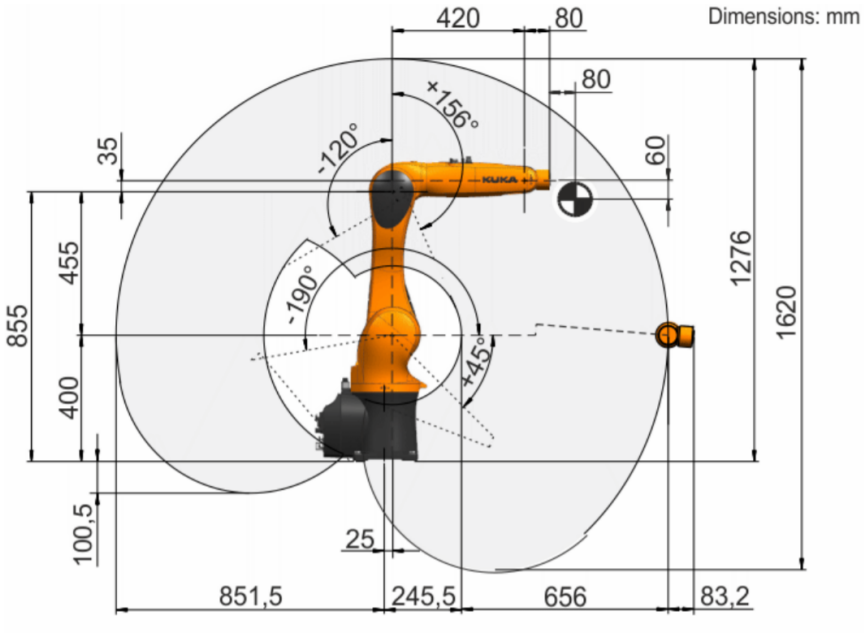


Figure 4.1: KUKA Agilus KR6 r900 sixx, workspace [16]

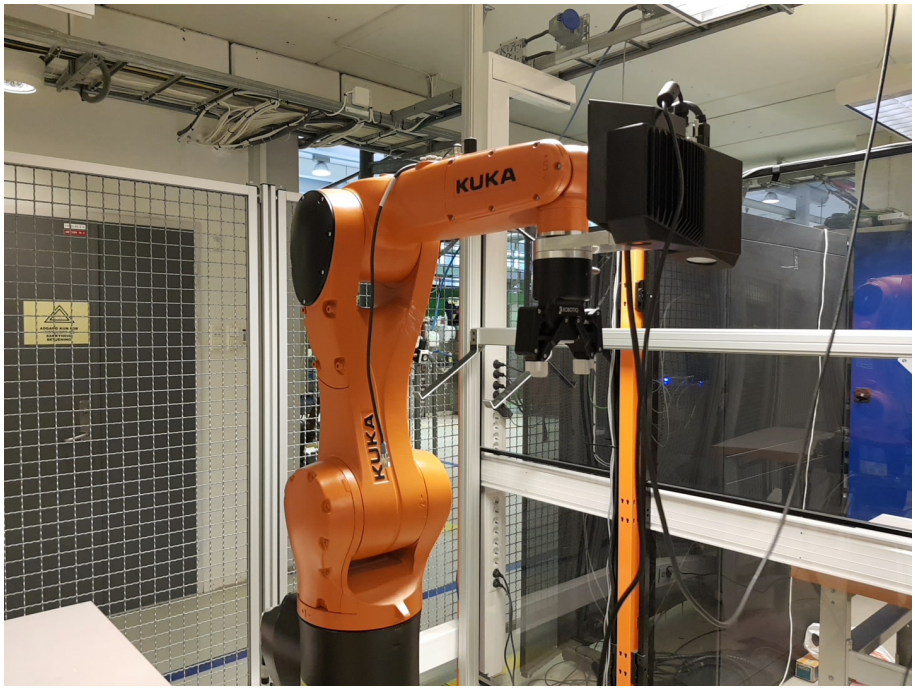


Figure 4.2: Robot manipulator with mounted camera

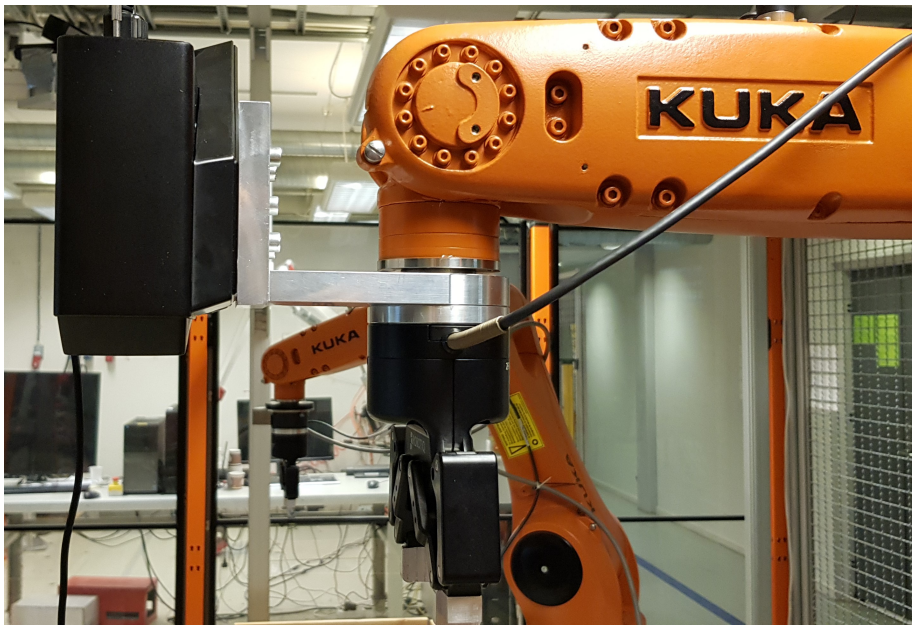


Figure 4.3: End-effector assembly

4.2 Zivid One Camera

The Zivid One structured light 3D scanner used in this project facilitates the fast capture of accurate 3D point cloud data from a scene, as well as a high resolution color image. The specified accuracy of the camera is 0.1 mm and the maximum acquisition frequency is 10 Hz. The camera has a wide array of settings which can be tuned in order to capture high precision 3D data in varying lighting conditions. In addition the settings can be adjusted for difficult objects, such as very dark objects, low texture objects or shiny objects. This makes it suited to object detection and pose estimation tasks related to robotic bin picking.

4.3 Hand-Eye Calibration Software

The software developed for hand-eye calibration of the laboratory robot setup consists of two main modules. A short description of all source files is provided in Table 4.2. The module used for dataset capture interfaces with both the Zivid One camera and the robot controller in order to capture both the pose of the robot end-effector and the .zdf file from the camera in an efficient manner. The software communicates with the robot controller over a UDP connection and receives continuous updates of the robot pose formatted as an xml-file. Communication with the camera is handled trough the python wrapper for the Zivid C++ API which provides access to functions for easy adjustment of image capture settings and capturing of .zdf files.

The robot end-effector pose is represented in the xml-file as a vector $[x, y, z, A, B, C]$ where x, y and z represent the position given in cartesian coordinates and A, B and C is the ZYX Euler angle parametrization of the orientation (2.8). The software stores the pose as a homogeneous transformation matrix in a plain-text file with the file-extension .t4. The n pairs of images and poses are numbered from 0 to $(n-1)$ in order to keep track of the matching robot poses and images. The procedure for creating a calibration dataset using this module is shown in Figure 4.4.

Filename	Language	External Dependencies	Description
homogeneous_transformations.py	python	numpy	Contains the class HTransf which provides functionality for manipulating matrices in $SE(3)$ using theory from Section 2.1
capture.py	python	Zivid-API, numpy, OpenCV	Used to create datasets for HE-calibration
zividHEcalibrator.py	python	numpy, Zivid-API, OpenCV, matplotlib	Contains ZividHECalibrator class used for hand-eye calibration of Zivid cameras.
utils.py	python	numpy, Zivid-API, OpenCV	Contains utilities used for calibration and experiments.
create_chessboard.py	python	OpenCV, numpy	Used to create custom calibration patterns
intrinsics.cpp	C++	Zivid-API	Used to retrieve the intrinsic camera parameters of the Zivid camera

Table 4.2: Overview of files used for hand-eye calibration

The main module for doing hand-eye calibration facilitates the following core functionality:

- loading of datasets
- intrinsic and extrinsic camera calibration
- hand-eye calibration using the algorithm developed by Park and Martin
- visualization of camera poses, robot poses and the hand-eye calibration represented as the pose of the camera relative to the end-effector frame

- calculation of error in the hand-eye calibration

The order of required operations when using the module for calibration on an existing dataset is shown in Figure 4.5.

4.3.1 3D Point Cloud Calibration Method

To make use of the 3D point cloud data which the Zivid One camera generates, a method for extrinsic camera calibration utilizing point cloud data has been implemented. First the corners of the calibration chessboard are found in the rgb image using the OpenCV python method cv2.findChessboardCorners(...). This function call returns the sub-pixel coordinates of the internal corners on the chessboard. From these it would be possible to extract the 3D data directly, but since chessboard corners are high color contrast areas, the 3D data at these points is generally noisier. Instead the center point of each chessboard square is considered. The sub-pixel coordinates of these points are found as the point of intersection of each square's diagonals. The 3D point cloud is arranged in a $1920 \times 1200 \times 3$ matrix where points can be extracted using integer indices, e.g $[202,\ 560]$. Since the center point pixel-coordinates of each square generally will be a tuple of floats, e.g $(202.56,\ 560.23)$, the 3D point corresponding to these pixel coordinates cannot be extracted directly. The four 3D points corresponding to the integer indices around the center point, $(202,\ 560)$, $(203,\ 560)$, $(202,\ 561)$, $(203,\ 261)$, are linearly interpolated to find the 3D coordinates of the center point in the camera coordinate system. This process is carried out for every center point on the calibration board, and for every image taken, to obtain a 3D point cloud on the calibration object for each camera pose.

For extrinsic camera calibration, the pose of the chessboard in the camera frame is found by fitting a plane to the 3D center-points, where the normal vector of the plane defines the z-axis of the chessboard coordinate frame. The x-axis is found by fitting a line to the 3D points along the long axis of the board, this vector is then projected into the plane orthogonal with the z-axis. The y-axis is found as the cross product $y = z \times x$. Each axis is then normalized to obtain a set of unit coordinate axes forming a basis of $\mathbb{R}^3$. This coordinate system is affixed to the centroid (2.43) of the point cloud. When using this method of finding the extrinsic calibration of the camera for hand-eye calibration, the calibration method is referenced as the "3D plane fit method".

Since the formulation of the hand-eye calibration problem does not require the explicit pose of the camera in each configuration, only the relative motion of the camera between two configurations, the transformation between the point clouds obtained in each configuration can be used instead. When using this method for hand-eye calibration it is referenced as the "3D correspondence method" or simply as the "3D method". These two methods are collectivly called the "3D point cloud methods".

4.3.2 Calculation of Estimation Error

To evaluate the error in the different hand-eye calibration methods, the mean deviation in pose from the average estimated calibration object pose is considered. The implemented methods are slightly different for OpenCV/3D plane fit method and for the 3D point cloud method, but both are based on the same principle.

For error calculation on hand-eye calibration using OpenCV and the 3D plane fit method the pose of the calibration object is calculated for each robot configuration. This is done using the robot pose, the hand eye calibration and the extrinsic camera calibration (3.1). From this the mean rotation vector and the mean translation vector is calculated and used as an estimate for the ground truth pose of the calibration object. The error for each object pose estimation is then calculated as the transformation between the ground truth pose and the estimated object pose.

$$(T_{error})_i = (T_{bo})_{GT}^{-1}(T_{bo})_i \tag{4.1}$$

Where $(T_{error})_i$ is the estimation error for robot configuration i, $(T_{bo})_{GT}$ is the estimated ground truth pose, and $(T_{bo})_i$ is the pose estimation of the calibration object using configuration i. The hand eye calibration error is represented as translation error and rotation error. Translation error is calculated as the mean norm of the translation error vectors for all the estimates (4.2), rotation error is calculated as the mean of the absolute value of the rotation angle error for all the estimates (4.3). The angle of rotation, θ_{err}, is calculated using (2.13).

$$t_{err} = \frac{1}{n}\sum_{i=0}^{n} ||(t_{err})_i|| \tag{4.2}$$

$$\theta_{err} = \frac{1}{n}\sum_{i=0}^{n} |(\theta_{err})_i| \tag{4.3}$$

The hand eye calibration error using the 3D correspondence method is calculated in a similar way to the 2D planar method. The position of each calibration point on the calibration object is estimated using (4.4) for each configuration i.

$$(P_o^b)_i = (T_{bt})_i(T_{tc})(P_o^c)_i \tag{4.4}$$

where $P_o^b \in \mathbb{R}^{4\times n}$ is the homogeneous representation of the object point cloud in the robot base coordinate system and $P_o^c \in \mathbb{R}^{4\times n}$ is the position of each calibration point in the camera coordinate system. An estimated ground truth position for each calibration point is calculated as the mean position from all the estimates $[(P_o^b)_0, (P_o^b)_1, (P_o^b)_2, ..., (P_o^b)_n]$. The error transformation is then calculated for each object pose estimation using point correspondences between the ground truth point cloud $(P_o^b)_{GT}$ and the calibration point

cloud P_o^b for each configuration i. The translation and rotation error is then calculated using Equations (4.2) and (4.3).

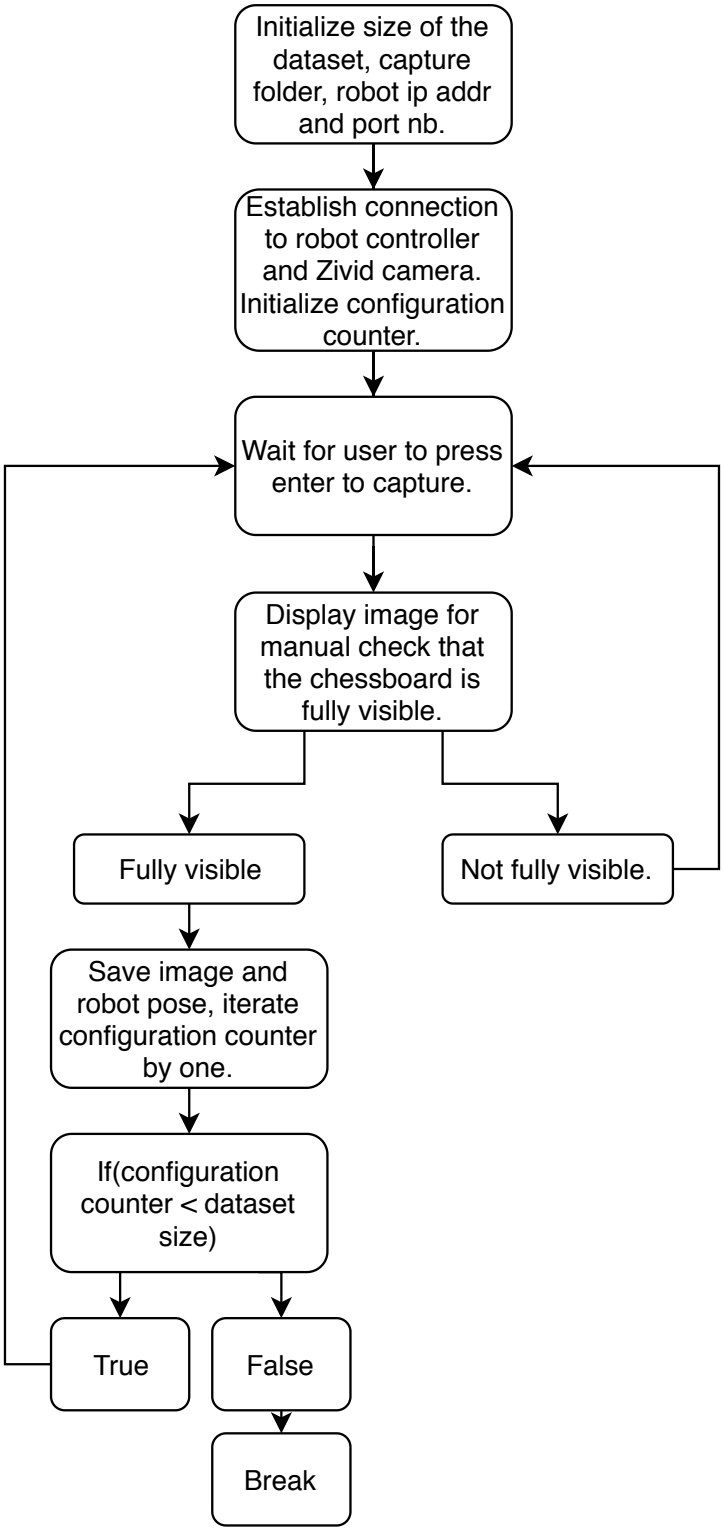


Figure 4.4: Method and logic of the capture module.

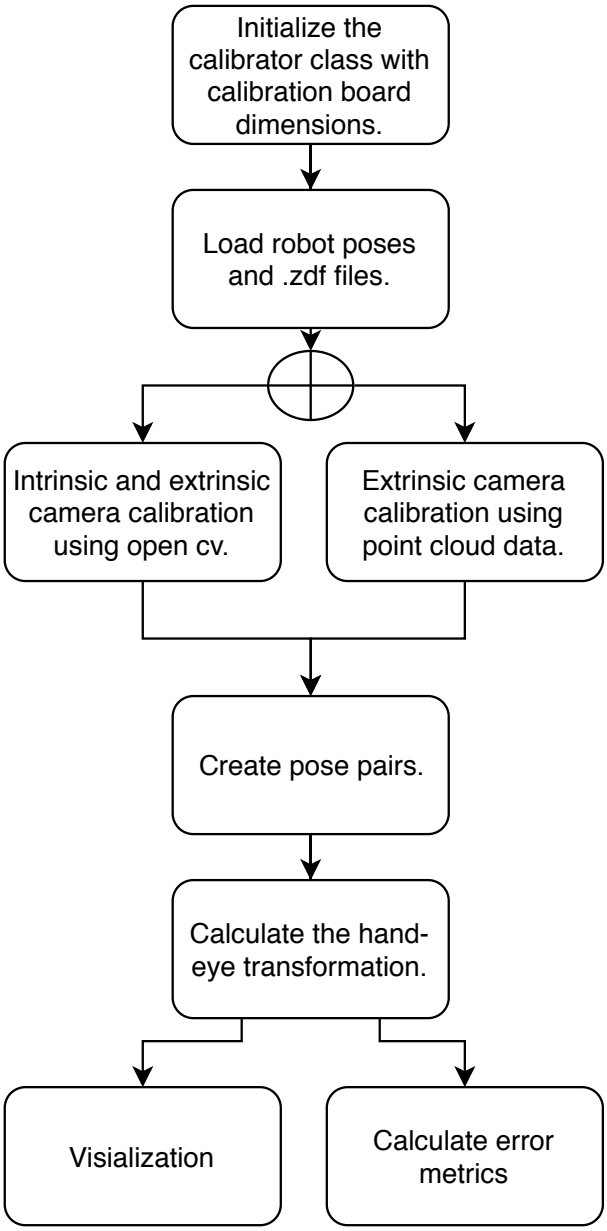


Figure 4.5: Procedure for hand-eye calibration using the ZividHECalibrator class.

Chapter 5

Experiments

In this chapter, experiments conducted to evaluate the robustness of the hand eye calibration methods and an experiment in which the optimal capture parameters are found are presented. The description, goal, methods, and metrics of each experiment is provided in Section 5.1 and the results are presented in Section 5.2.

5.1 Experiments

5.1.1 Tuning 3D Camera Capture Parameters

In order for the Zivid camera to be able to capture accurate 3D data from an image, the capture parameters must be tuned.

Goal

The goal of this experiment is to find a set of optimal capture parameters for the Zivid One camera. These are to be used to increase the accuracy and precision of the 3D data used for hand eye calibration.

Method

The following parameters are tunable trough the Zivid Studio desktop application.

Setting	Options
Exposure time	$6500\mu s - 100000\mu s$
Iris	$0 - 72$
Reflection filter	On/Off
Contrast filter	On/Off
High Dynamic Range (HDR)	On/Off
Brightness	$0.00 - 1.80$
Gain	$1.0 - 16.0$

The method used for manually finding the best possible parameters follows the guide provided by Zivid on their website [17] for getting good quality data on calibration chessboards. This method uses the logarithmic pixel intensity histogram as well as the depth image to optimize the parameters above by trial and error. It recommends using HDR with frames optimized for camera positions close to, at medium distance, and further away from the chessboard. The assisted capture functionality in Zivid Studio is also used to automatically tune the camera parameters.

Evaluation Metric

The evaluation metrics used for manual parameter tuning are the pixel intensity histogram and the subjective observation of the evenness of the chessboard depth data. In addition the total number of invalid values on the chessboard, returned as "not a number" (NaN) should be minimized. The tuned parameters should place the logarithmic intensity of the pixels on the calibration board in the rightmost section of the histogram. The number of NaN points are found by visual inspection of the color coded depth map of the point cloud. NaN points show up as black pixels in the depth map.

5.1.2 Hand-Eye Calibration Using Generated Noisy Data

Using generated noisy pose data for hand eye calibration is useful to determine the noise-sensitivity of the calibration methods. By generating a synthetic dataset and adding Gaussian noise the robustness of the methods can be determined by evaluating the calibration error.

Goal

The findings of this experiment will be used as a basis for discussing the robustness of 3D point cloud methods for hand eye calibration on noisy datasets.

Method

Noisy point cloud data is generated by selecting a ground truth calibration object pose in robot base frame $(T_{bo}^{b})_{GT}$ and hand eye transformation matrix, $(T_{tc})_{GT}$. These were chosen to be

$$(T_{bo}^{b})_{GT} = \begin{bmatrix} 0 & -1 & 0 & 200 \\ 1 & 0 & 0 & 70 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, (T_{tc})_{GT} = \begin{bmatrix} 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 100 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Random robot poses $(T_{bt})_i$ are then generated. The values for the elements of the rotation vectors $k\theta$ are drawn from the normal distribution $N(\mu, \sigma^2) = N(0, \pi^2)$ and the elements for the translation vector of each pose are generated from the normal distribution $N(\mu, \sigma^2) = N(0, (300\sqrt{3})^2)$. It should be noted that the generated poses need not be feasible in the physical system, however $\sigma = 300\sqrt{3}mm$ ensures that the generated positions are somewhat close to being within the operational space of the KUKA agilus robot. The corresponding pose of the chessboard in the camera frame $((T_{co})_i)$ is then calculated as

$$(T_{co})_i = ((T_{bt})_i (T_{tc})_{GT})^{-1} (T_{bo})_{GT} \tag{5.2}$$

We have now obtained a noise free dataset of robot poses and camera poses. To evaluate the sensitivity of the calibration methods utilizing point clouds for extrinsic camera calibration, a point cloud centred in the robot base, consisting of $n \times m$ points spaced in correspondence to the center points on the calibration chessboard is generated. To each point in this point cloud a Gaussian noise vector with mean value μ and standard deviation σ is added, resulting in a set of noisy homogeneous chessboard points for each robot configuration, $P_i \in \mathbb{R}^{4\times(n\times m)}$. This point cloud is then transformed to the position of the calibration object using $(T_{bo})_{GT}$ resulting in the object point cloud $(P_o)_i$.

$$(P_o)_i = (T_{bo})_{GT} P_i \tag{5.3}$$

Evaluation Metric

The set of generated noisy calibration points and generated robot poses are used to do hand eye calibration. Error estimates are calculated using (4.2) and (4.3) in Section 4.3.2. This is done using an increasing number of configurations in order to determine when the error converges. The final error on translation and rotation for each method is used as the performance metric.

5.1.3 Hand-Eye Calibration on Real Datasets

Goal

The goal of this experiment is to evaluate the robustness of the hand eye calibration based on OpenCV extrinsic calibration and based on the 3D point cloud methods. The outcome of this experiment will hopefully provide some insight into the strengths and weaknesses of each method, which can be used to decide which method to use under various conditions. In addition to this, finding the hand-eye transformation for the setup is necessary in order to make further use of the robot vision system.

Method

Datasets were created using the capture software described in Section 4.3. Emphasis was put varying the pose of the end-effector orientations about all three axes of rotation as is recommended in [2]. Both the OpenCV method and the 3D point cloud methods were used for extrinsic calibration of the camera for each dataset.

Evaluation metrics

The error values for both methods are plotted against the number of configurations used in the calibration. The final rotation and translation error obtained using all configurations in the dataset, in addition to the rate of conversion, is used as the evaluation criterion for each calibration method.

5.2 Results

5.2.1 Camera Parameter Tuning

The camera settings are tuned using the method above. To illustrate the difference between the default settings and the tuned settings, example depth maps are presented in fig. 5.1.

Parameter tuning resulted in the settings presented in table 5.1 being used when capturing calibration datasets. Three HDR frames were used, with iris settings as listed in the talbe. All other settings were kept the same for each HDR frame. These frames were tuned separately to yield best possible results on near, middle and far distances to the calibration board. This should result in good 3D point cloud data over a range of distances [17].

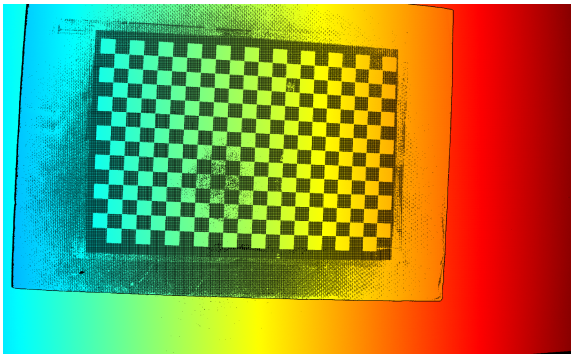
Setting	**Value**
Exposure time	$8333\mu s$
Iris	$[16, 26, 50]$
Reflection filter	On
Contrast filter	On
High Dynamic Range (HDR)	On
Brightness	1.0
Gain	1.0

Table 5.1: Manually tuned camera parameters.

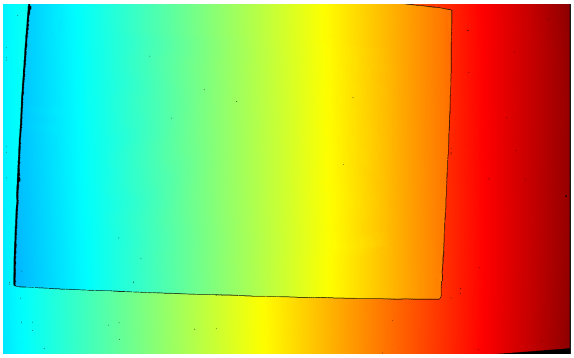
The assisted capture mode in Zivid Studio was also used to attain camera settings. This resulted in the settings in table 5.2.

Setting	**Value**
Exposure time	$8333\mu s$
Iris	$[22, 31, 68]$
Reflection filter	On
Contrast filter	On
High Dynamic Range (HDR)	On
Brightness	1.0
Gain	1.0

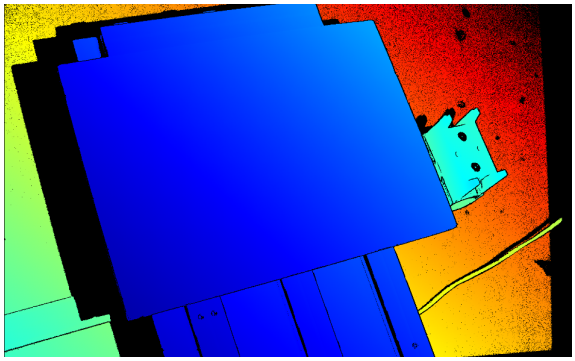
Table 5.2: Automatically tuned camera parameters.



(a) Default settings

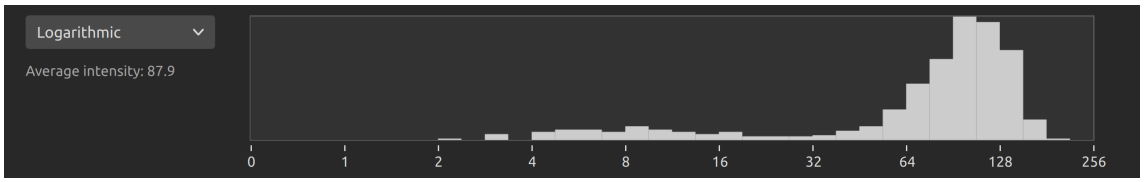


(b) Manually tuned settings



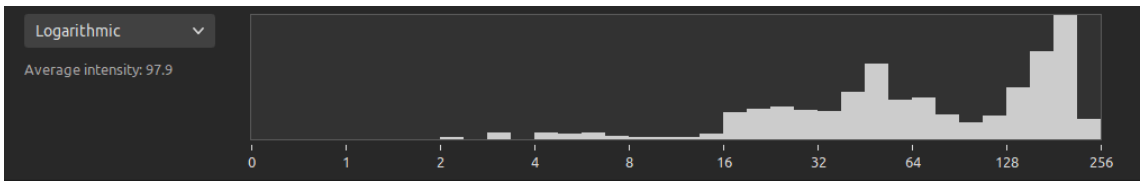
(c) Automatically tuned settings

Figure 5.1: Depth maps for images taken with default and tuned parameters.



(a) Default settings

(b) Manually tuned settings



(c) Automatically tuned settings

Figure 5.2: Logarithmic intensity histograms images taken with default and tuned parameters.

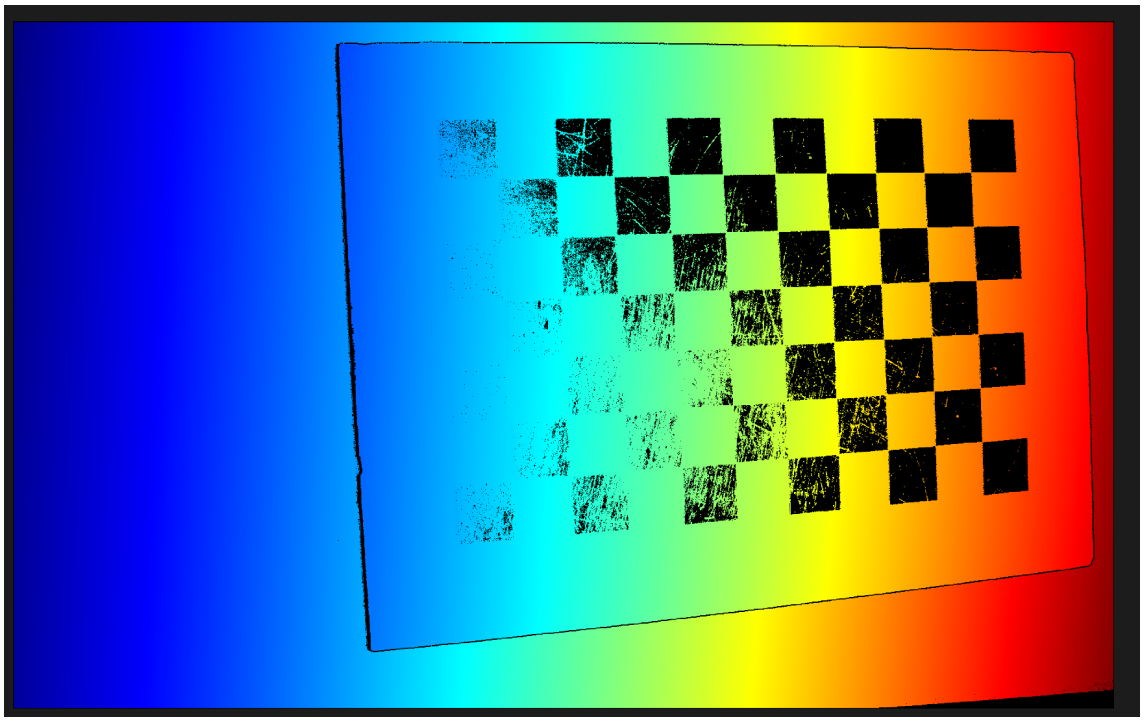
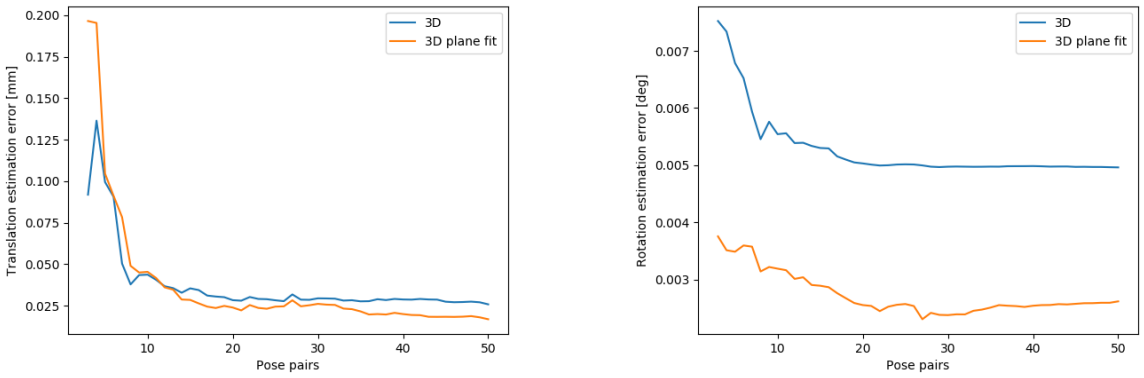


Figure 5.3: Depth map from image taken of a calibration object with black squares. Tuned parameters from table 5.1 are used.

5.2.2 Hand-Eye Calibration Using Generated Noisy Data

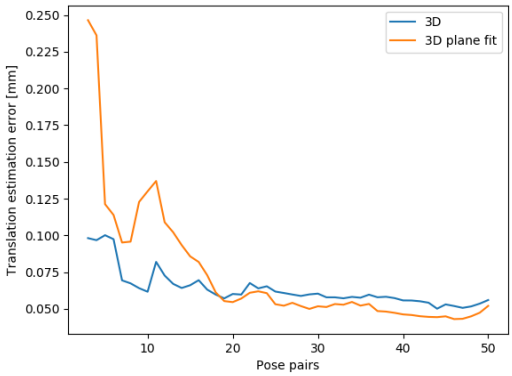
Simulations were done by generating a calibration set consisting of 50 robot configurations. Gaussian noise with increasing standard deviation was added to each point cloud. Hand eye calibration and error estimation was then carried out with an increasing number of pose pairs for each noise level, this is shown in Figures 5.4 to 5.14. The final errors for each simulation are shown in Table 5.3, and plotted in Figures 5.15 and 5.16.



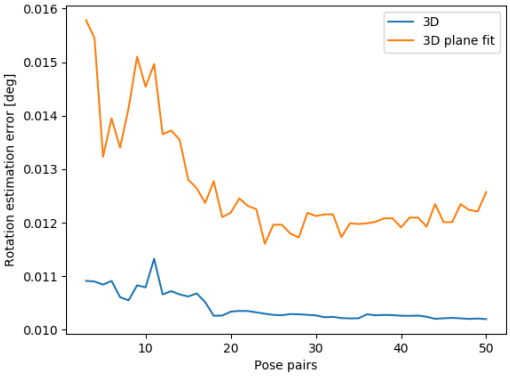
(a) Translation error

(b) Rotation error

Figure 5.4: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.01)$

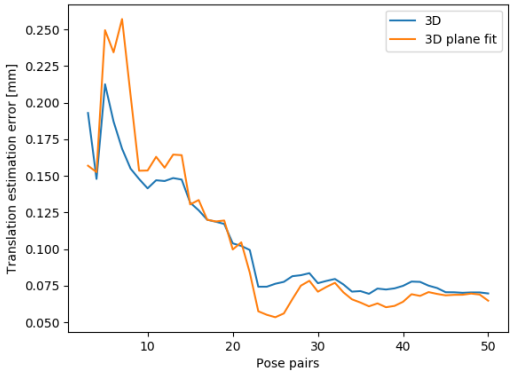


(a) Translation error

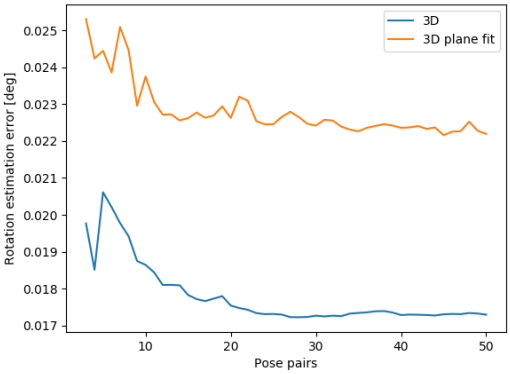


(b) Rotation error

Figure 5.5: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.04)$

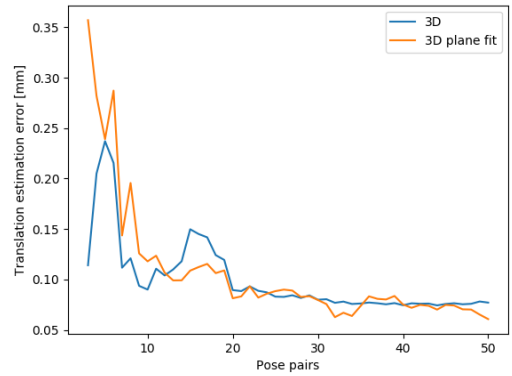


(a) Translation error

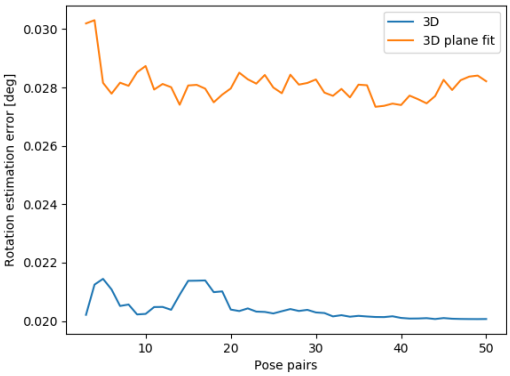


(b) Rotation error

Figure 5.6: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.09)$

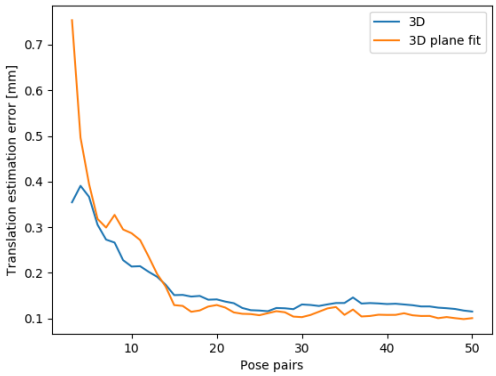


(a) Translation error

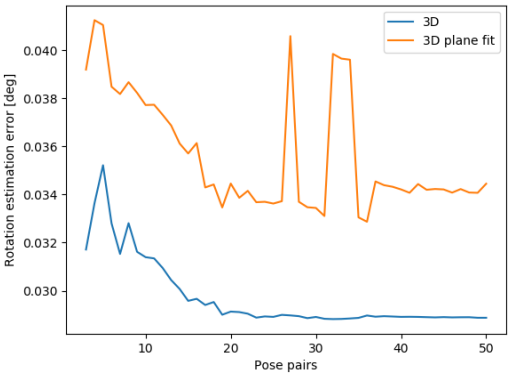


(b) Rotation error

Figure 5.7: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.16)$

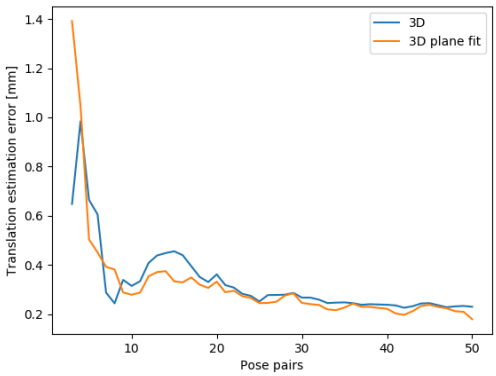


(a) Translation error

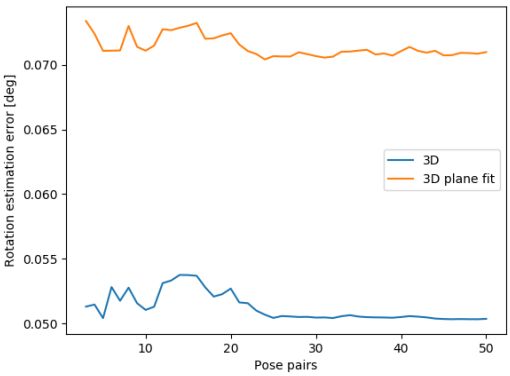


(b) Rotation error

Figure 5.8: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 0.25)$

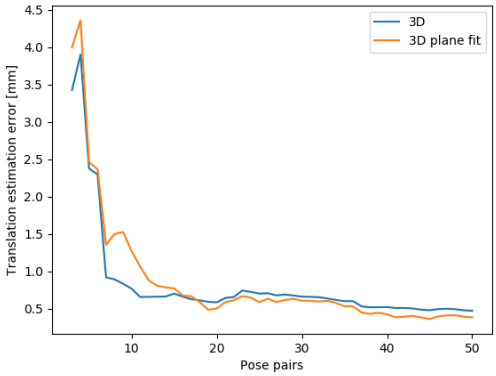


(a) Translation error

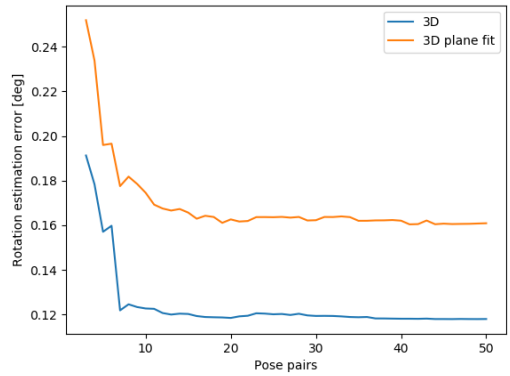


(b) Rotation error

Figure 5.9: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 1.0)$

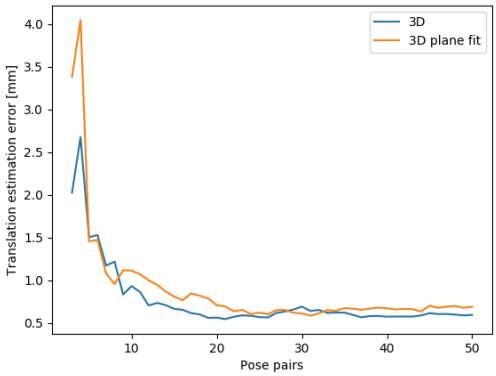


(a) Translation error

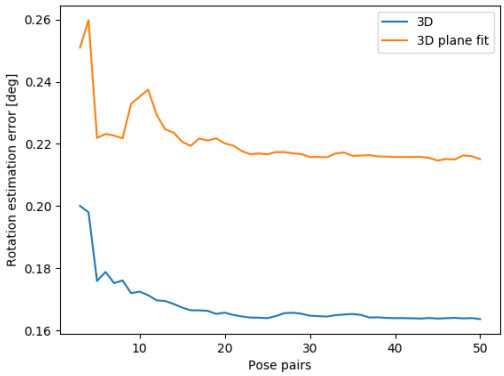


(b) Rotation error

Figure 5.10: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 4.0)$

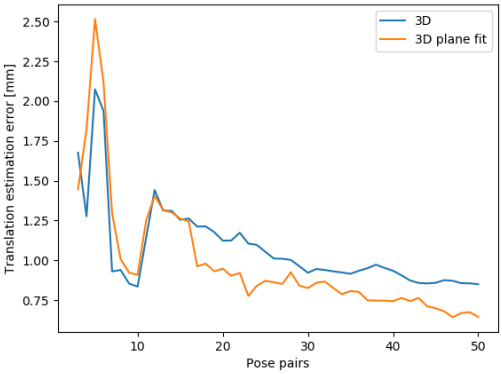


(a) Translation error

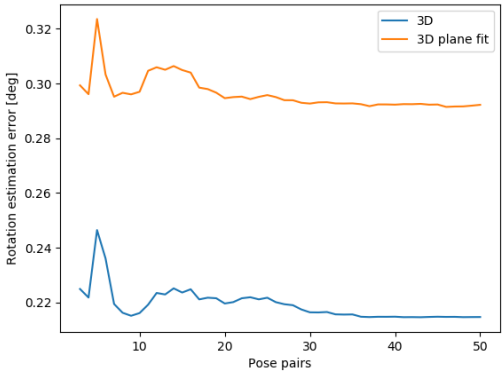


(b) Rotation error

Figure 5.11: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 9.0)$

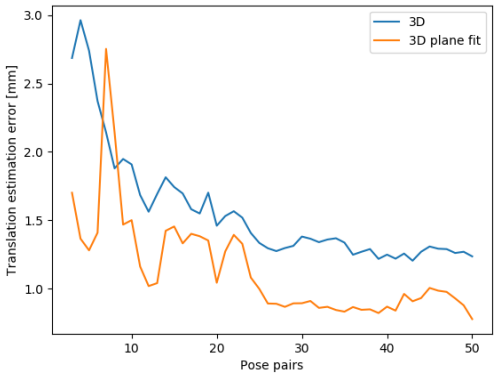


(a) Translation error

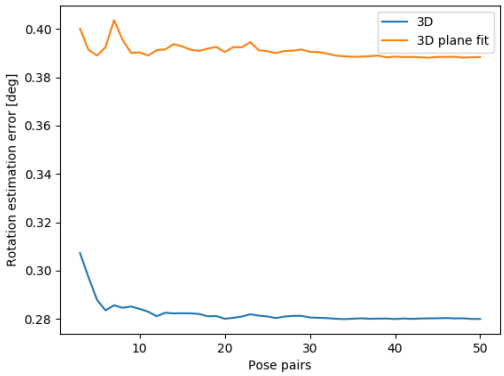


(b) Rotation error

Figure 5.12: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 16.0)$

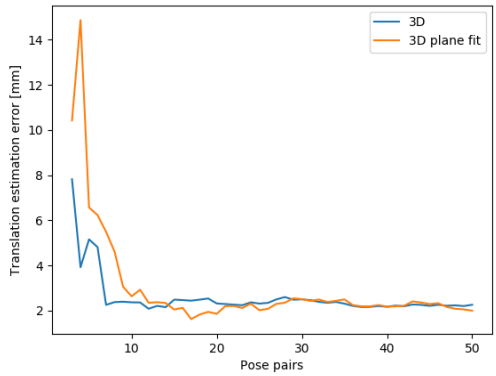


(a) Translation error

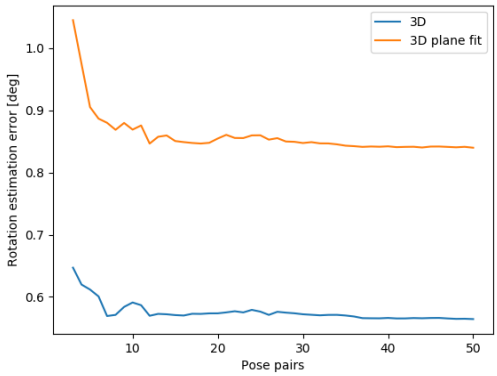


(b) Rotation error

Figure 5.13: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 25.0)$



(a) Translation error



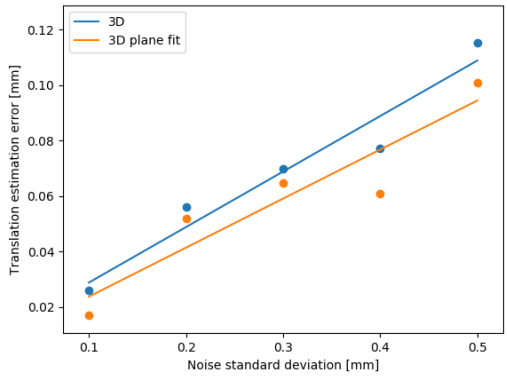
(b) Rotation error

Figure 5.14: Calibration on data with added noise from $N(\mu, \sigma^2) = N(0, 100.0)$

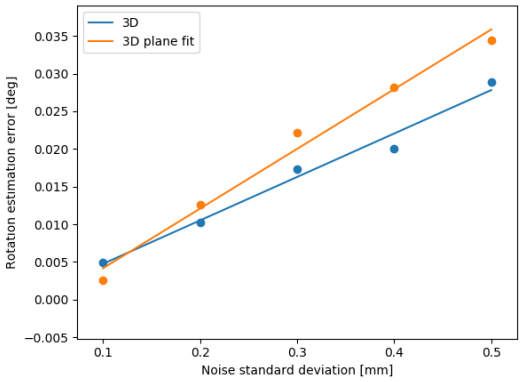
It should be noted that while some of the graphs appear to show more variation in error as it converges, this is mostly due to the different scaling on the y-axis, which i dependent on the initial error.

Noise distribution $N(\mu, \sigma^2)$	Translation error 3D [mm]	Translation error 3D plane fit [mm]	Rotation error 3D [deg]	Rotation error 3D plane fit [deg]
$N(0, 0.01)$	0.02581	0.01683	0.00496	0.002617
$N(0, 0.04)$	0.05593	0.05195	0.01019	0.01257
$N(0, 0.09)$	0.06975	0.06480	0.01729	0.02219
$N(0, 0.16)$	0.07710	0.06077	0.02007	0.02821
$N(0, 0.25)$	0.11530	0.10083	0.02887	0.03444
$N(0, 1.0)$	0.22951	0.17882	0.05036	0.070979
$N(0, 4.0)$	0.47090	0.38327	0.11798	0.16087
$N(0, 9.0)$	0.59217	0.68763	0.16361	0.21507
$N(0, 16.0)$	0.85008	0.64449	0.21464	0.29222
$N(0, 25.0)$	1.23648	0.77810	0.27990	0.38845
$N(0, 100.0)$	2.25496	1.99512	0.56434	0.83999

Table 5.3: Final translation errors and rotation errors for 3D and 3D plane fit methods.

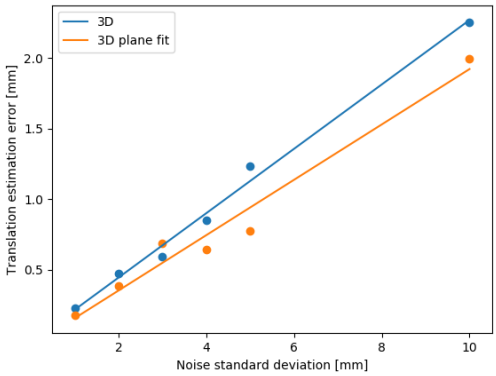


(a) Noise/translation error relationship

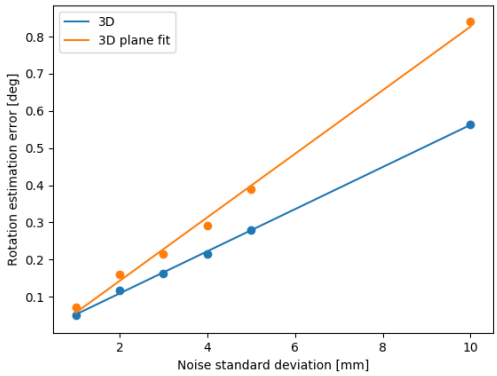


(b) Noise/rotation error relationship

Figure 5.15: Scatter plot and trend line for noise standard deviations 0.1, 0.2, 0.3, 0.4, 0.5



(a) Noise/translation error relationship



(b) Noise/rotation error relationship

Figure 5.16: Scatter plot and trend line for noise standard deviations 1.0, 2.0, 3.0, 4.0, 5.0, 10.0

5.2.3 Hand-Eye Calibration on Real Datasets

Presented here are the results of hand-eye calibration on datasets captured using the laboratory setup. Including examples showing the estimated pose of the calibration boards in the camera coordinate frame using both the OpenCV- and the 3D plane fit methods (Figures 5.17 and 5.18). The tools used for visualization of camera- and robot poses are also demonstrated in Figures 5.18 and 5.19, these have been useful for validation of extrinsic camera calibration. The error plots from calibration on the datasets are shown in Figures 5.21 to 5.25. Additionally the hand-eye transformation resulting in the lowest translational error is presented in (5.4).

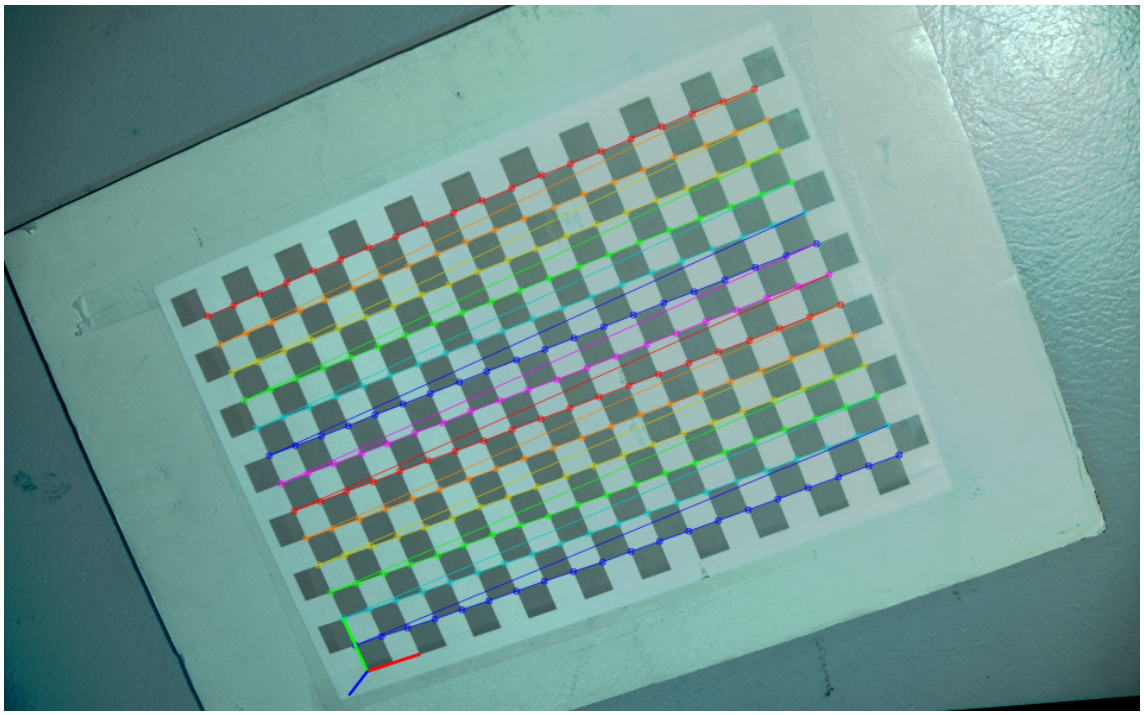


Figure 5.17: Chessboard pose, estimated using OpenCV extrinsic camera calibration. The placement of the chessboard coordinate frame is defined in the camera calibration stage. Chessboard corners are marked using the drawChessboardCorners() function of OpenCV.

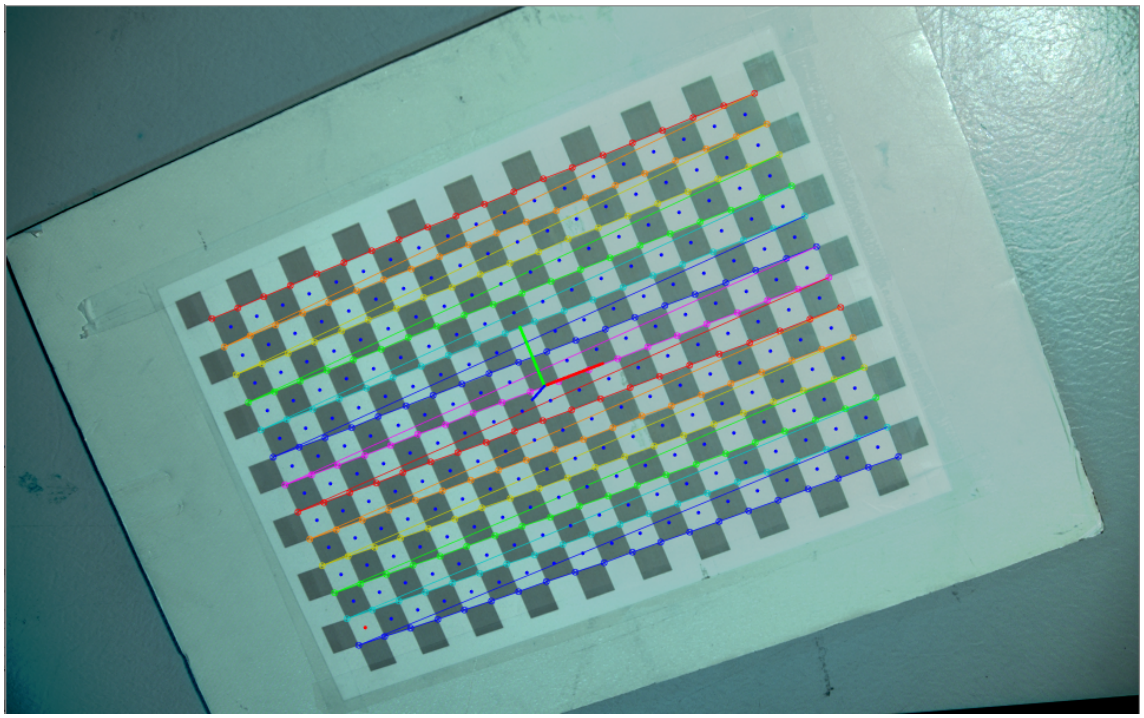


Figure 5.18: Chessboard pose, estimated using 3D point clouds for extrinsic camera calibration. Square center points are marked by blue dots. The chessboard coordinate frame is placed in the centroid of the point cloud comprised of the center points.

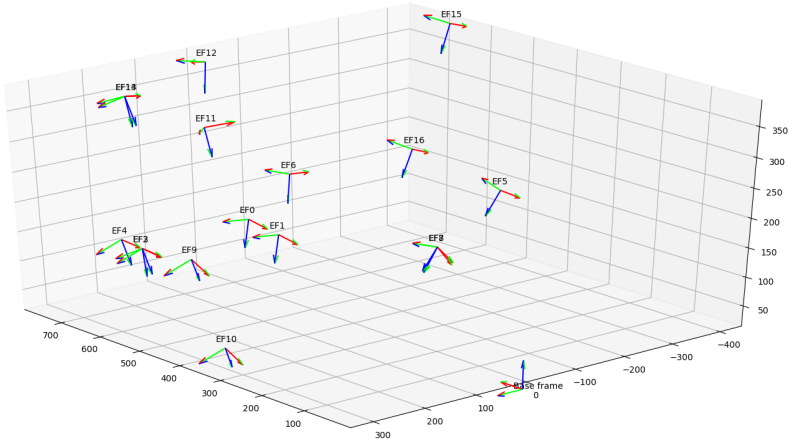


Figure 5.19: Visualization of robot end-effector poses relative to the base frame

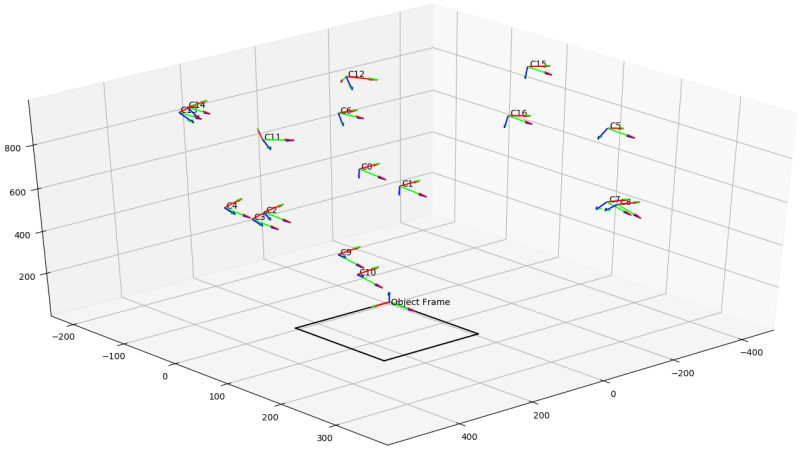
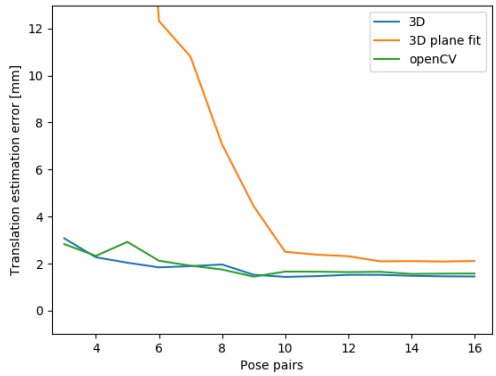
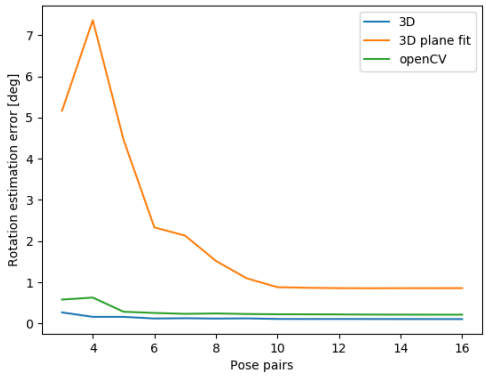


Figure 5.20: Chessboard pose, estimated using 3D point clouds for extrinsic camera calibration.



(a) Translation error

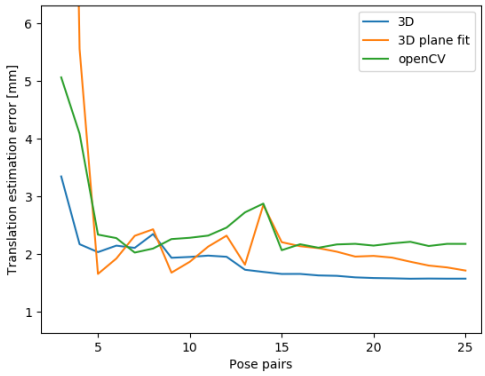


(b) Rotation error

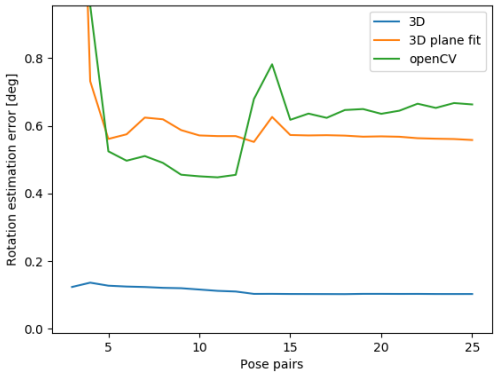
Figure 5.21: Dataset1. Hand-eye calibration errors on images captured at a distance in the range 600-700 mm from the calibration object . 9×6 chessboard with square size of 20.0 mm fig. A.1. Camera settings as in table 5.2

	3D	3D plane fit	OpenCV
Translation error [mm]	1.455	2.112	1.582
Rotation error [deg]	0.109	0.860	0.216

Table 5.4: Final errors on Dataset 1.



(a) Translation error

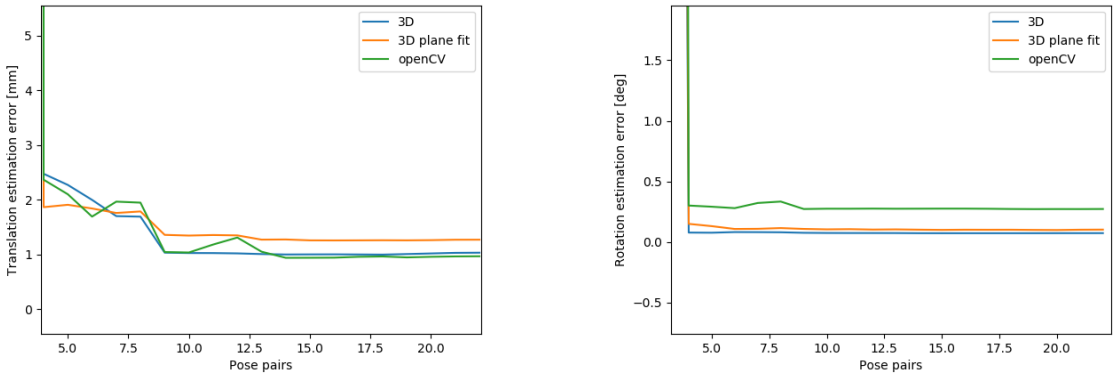


(b) Rotation error

Figure 5.22: Dataset 2. Hand-eye calibration errors on images captured at a distance of 600-800 mm from the calibration object. 9×6 chessboard with square size of 20.0 mm. fig. A.1. Camera settings as in table 5.1

	3D	3D plane fit	OpenCV
Translation error [mm]	1.575	1.715	2.178
Rotation error [deg]	0.103	0.558	0.663

Table 5.5: Final errors on Dataset 2.



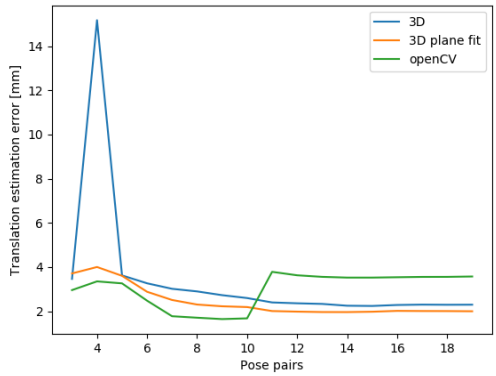
(a) Translation error

(b) Rotation error

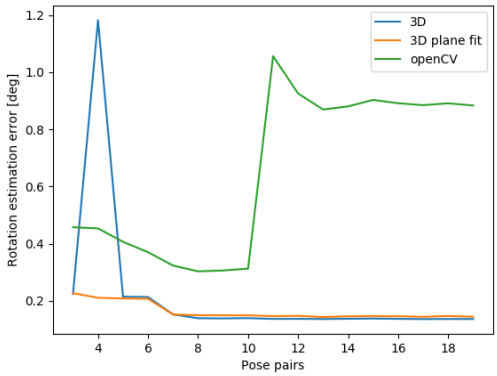
Figure 5.23: Dataset 3. Hand-eye calibration errors on images captured at a distance of 750-950 mm from the calibration object. 21×14 chessboard with square size of 13.8 mm. Gray value 180 fig. A.2. Camera settings as in table 5.1

	3D	3D plane fit	OpenCV
Translation error [mm]	1.035	1.272	0.970
Rotation error [deg]	0.073	0.102	0.272

Table 5.6: Final errors on Dataset 3.



(a) Translation error

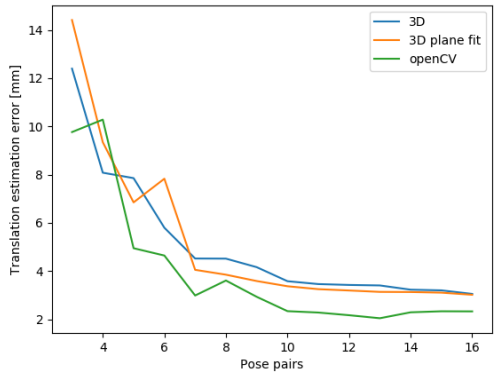


(b) Rotation error

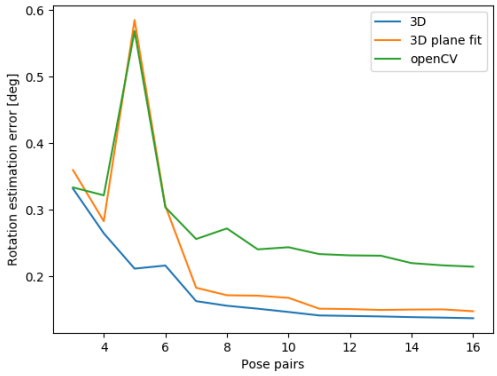
Figure 5.24: Dataset 4. Hand-eye calibration errors on images captured at a distance of 800-1000 mm from the calibration object. 20×13 chessboard with square size of 13.9 mm. Gray value 130fig. A.3. Camera settings as in table 5.2

	3D	3D plane fit	OpenCV
Translation error [mm]	2.295	1.998	3.573
Rotation error [deg]	0.136	0.144	0.884

Table 5.7: Final errors on Dataset 4.



(a) Translation error



(b) Rotation error

Figure 5.25: Dataset 5. Hand-eye calibration errors on images captured at a distance of 800-1200 mm from the calibration object. 20×13 chessboard with square size of 13.9 mm. Gray value 130 fig. A.3. Camera settings as in table 5.2

	3D	3D plane fit	OpenCV
Translation error [mm]	3.052	3.016	2.327
Rotation error [deg]	0.137	0.148	0.215

Table 5.8: Final errors on Dataset 5.

$$T_{tc} = \begin{bmatrix} -0.045 & 0.999 & 0.000 & -145.462 \\ -0.986 & -0.045 & -0.159 & 72.845 \\ -0.159 & -0.007 & 0.987 & -180.656 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

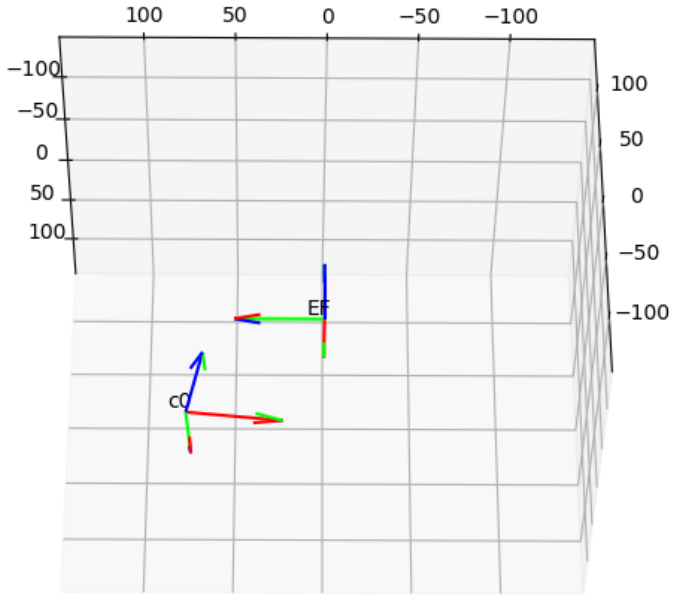


Figure 5.26: Visualization of the hand-eye transformation in (5.4) resulting from hand-eye calibration on dataset 3. The robot end-effector is denoted “EF” and the camera center is denoted “c0”

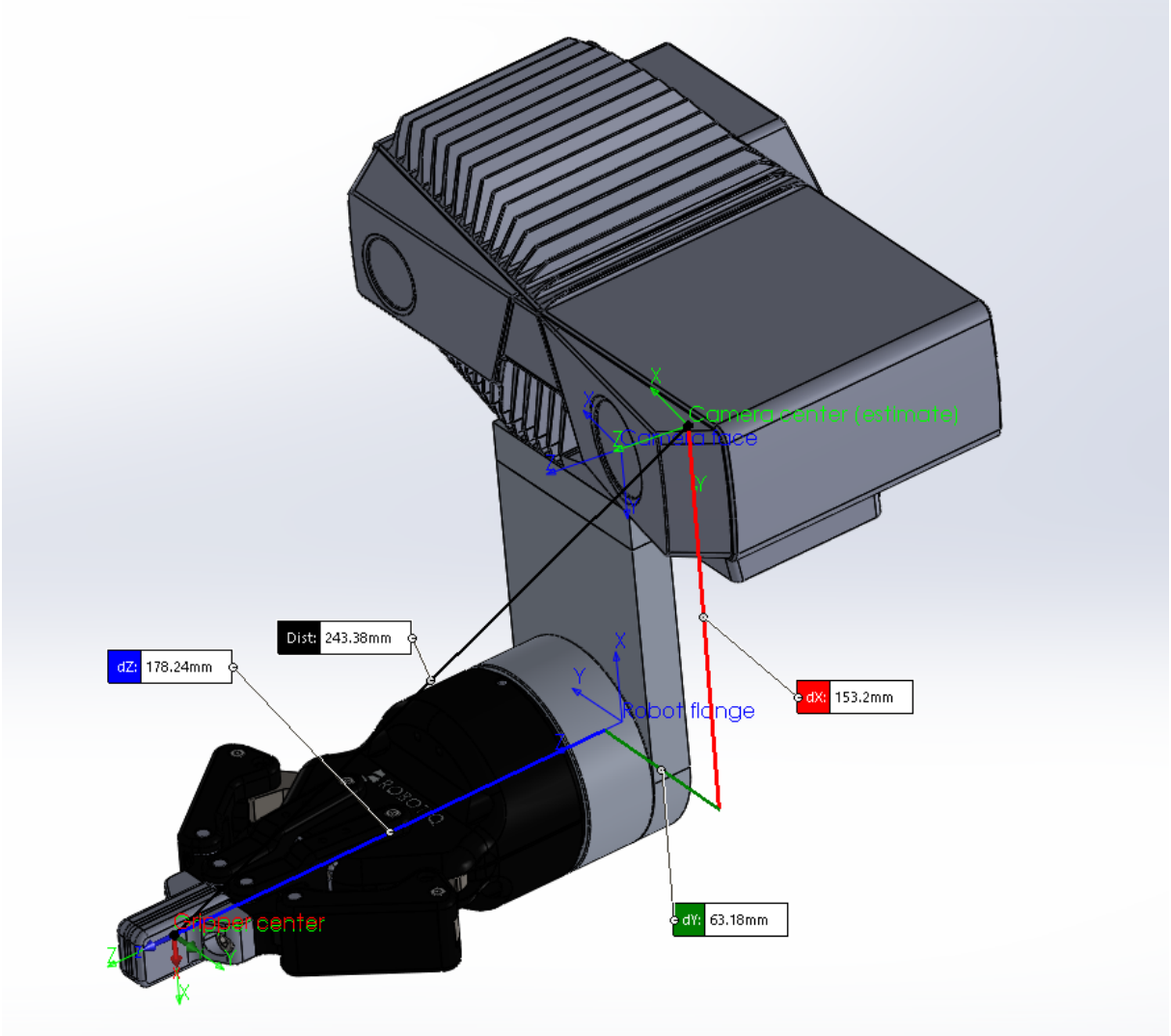


Figure 5.27: Rough estimate of the position of the camera center, illustrated on a 3D model of the end-effector assembly.

Chapter 6

Discussion

This chapter contains analysis and discussion of the results obtained from the experiments presented in section 5.2.

6.1 Capture parameter tuning

The process of manually tuning the capture parameters on the Zivid One camera was fairly straight forward using the steps and method described in [17]. The images taken with the default settings were overexposed which resulted in noisy or no point cloud data in several areas of the calibration board. Tuning mainly consisted of adjusting the iris value to reduce or increase the exposure of the image depending on what was observed in the histograms and the depth maps. It was found that adjusting the other parameters such as gain, brightness, exposure time and filter values had little to no positive effect on the resulting image. These were therefore left on their default values. Exposure time could also be used to adjust the level of exposure in the image, but tuning of the iris values was found to give more control over the exposure and the number of values in the depth map for a larger range of distances to the calibration object.

Using the assisted capture mode in Zivid Studio resulted in fairly even depth mappings on the calibration board as can be seen in Figure 5.1c. However in the rgb images the calibration board tended to be slightly overexposed compared with the manually tuned settings, this is because of the higher iris values used which allows for more light to hit the imaging sensor. It is noted that the setup of the calibration object when automatic tuning was used, while similar to the one used for manual tuning, was different in that the calibration board was offset from the floor. This likely led to the automatic tuning software trying to optimize the camera parameters to capture good 3D data over a larger range of distances. Which might lead to more noisy data compared with images captured with the manually tuned parameters. Automatic tuning with the same calibration board setup as was used for the manual tuning resulted in similar settings to the ones in Table 5.2, but

with a lower max iris value of 54.

It was also observed that using calibration boards with black squares resulted in poor depth data, as can be seen in Figure 5.3. This is because of the high contrast between white and black pixels in the image. The solution to this problem is to use calibration boards which have white and gray squares. This solves the problem by removing high contrast pixel values close together, while still providing enough contrast to allow for reliable corner detection on the chessboard.

6.2 Hand-Eye Calibration on Generated noisy data

The results of this experiment show that both the 3D point cloud correspondence method and the 3D plane fit method yield relative camera poses which are good enough for the hand-eye algorithm to converge on a noisy dataset. Both methods also result in similar convergence rates. There does however appear to be a slight difference in which error the two methods minimizes. By looking at the plots of the final errors from each noise level in Figures 5.15 and 5.16 it is clear that the 3D plane fit method results in a lower translation error at all noise levels. The error trend lines for the two methods also diverge slightly as the noise level increases, indicating that the position estimate of the 3D plane fit method is also more robust when it comes to increases in noise levels compared with the 3D correspondence method. The opposite is observed in the plots of rotation error vs. noise level. Here the 3D correspondence method is consistently more accurate than the 3D planar method. The trend lines also diverge indicating that the orientation estimate of the 3D planar method is more sensitive to increases in noise level than the 3D correspondence method.

It is difficult to pin-point the exact reason for the observed results. Both methods calculate the relative motion of the camera, for use in hand-eye calibration, by considering the translation of the centroid of each image point cloud. The average translation of the centroid between the ground truth point cloud and the centroid for each image point cloud is used as the metric for calculating the error in the calibration for both methods. It would be reasonable to expect that the method which can achieve the highest orientation accuracy would also achieve the highest position accuracy, since the resulting rotation matrix is used to optimize the translation vector in the hand-eye calibration algorithm used. This is however not consistent with the simulation results.

6.3 Hand-Eye Calibration on Real datasets

The results from hand-eye calibration on real datasets show that there are some differences in the calibration accuracy achieved by the different methods used (3D correspondence, 3D plane fit and OpenCV) to calculate relative camera motion in a pose pair . On

dataset 1 (Figure 5.21) the camera parameters obtained from the assisted capture mode were used. The calibration object was a 9×6 chessboard with 20.0 mm gray and white squares. Here the 3D point correspondence method and OpenCV performed significantly better than the 3D plane fit method, both on translation- and rotation accuracy. On dataset 2 (Figure 5.22) the same chessboard as in dataset 1 was used, however the camera parameters were obtained by manual tuning. The results show the two 3D methods performing better than OpenCV on translation accuracy while only the 3D correspondence method performed well on rotation accuracy. While the only difference between these two datasets should be the camera parameters, this is unlikely to be the case. Since datasets are captured by manually by moving the robot between poses, the quality of the images taken of the calibration object and how well the resulting pose pairs facilitate optimization of the hand-eye calibration algorithm has an effect on the final results. However, it is expected that the point cloud data captured in dataset 2 is less noisy than the point cloud data in dataset 1 because of the manually tuned camera parameters. This is the probable reason why the two 3D methods outperform OpenCV on this dataset.

Dataset 3 (fig. 5.23) is captured using the same manually tuned parameters as in dataset 2. The main difference is that the calibration board used is a 21×14 chessboard with 13.8 mm gray and white squares. The gray level of the squares are comparable to the chessboard used in dataset 1 and 2. On this dataset the 3D correspondence method and the OpenCV method performed best on translation accuracy while the two 3D methods outperformed OpenCV on rotation accuracy. By comparing the results on dataset 2 and 3 it appears as using calibration boards with more squares result in higher overall accuracy across all methods tested. This is most likely due to the fact that any particularly noisy points have less of an effect on the overall accuracy of the camera pose estimation.

Datasets 4 (fig. 5.24) and 5 (fig. 5.25) were captured using the automatically tuned camera settings. The chessboard used here was of similar dimensions as the one in dataset 3, but with darker gray squares. The images in these datasets were captured further away from the calibration object which likely resulted in both noisier point cloud data as well as a reduced number of pixels on the calibration object. The two 3D methods outperformed OpenCV on dataset 4, but OpenCV had a lower translational error on dataset 5. The sudden spike observed in rotation error of OpenCV on dataset 4 is likely to be caused by a large error in an extrinsic camera calibration on a single image. But looking at the at the placement of the coordinate frames for each board no suspects were found.

The results of hand-eye calibration on datasets 1, 2 and 3 show better performance than calibration done on datasets 4 and 5. This is most likely due to the differences in calibration setup. Firstly a lighter gray was used on the calibration chessboard in datasets 1, 2 and 3 compared to the one in datasets 4 and 5. Second, datasets 1 and 2 were captured with the calibration board laying directly on a flat surface, the images were taken closer to and over a smaller range of distances than in datasets 4 and 5. Thirdly

the camera settings used in datasets 4 and 5 could have resulted in noisier image- and point cloud data. These factors are seen as likely to affect the quality of the datasets, which has an impact on the calibration accuracy.

The results of hand-eye calibration on datasets 1, 2 and 3 are inconsistent with the results from the results from hand-eye calibration on generated noisy datasets. The results of hand-eye calibration on datasets 1, 2 and 3 show that the accuracy of the 3D correspondence method is better than the 3D plane fit method both on translation and on rotation. The results of hand-eye calibration on datasets 4 and 5 are more in line with what was expected based on the simulations. On these datasets the 3D plane fit method was better than the 3D correspondence method on translation accuracy, while the opposite was true for rotation accuracy. This was also observed in the simulations. One possible reason for this is that the 3D plane fit method is more noise-tolerant than the 3D correspondence method, while the base-line accuracy of the 3D correspondence method is lower than the 3D plane fit method in cases where the point cloud data has low noise levels. Since the point cloud data in datasets 4 and 5 is likely to be noisier than in 1, 2 and 3, due to the setup of the experiment, the same tendencies are observed for datasets 4 and 5 as for the simulations.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this project report theory and methodology for hand-eye calibration using point clouds for extrinsic camera calibration is presented. Experiments investigating the optimal parameters for capturing datasets, and the performance of the various calibration methods were conducted. Experimental results indicate that the accuracy of the hand-eye calibration is correlated with the overall quality of the calibration datasets.

Two methods were implemented for hand-eye calibration using 3D point clouds, referred to as the "3D correspondence method" and the "3D plane fit method". Simulations on noisy point cloud data indicate both methods as suitable. Analysis show that the 3D correspondence method resulted in better performance on orientation estimate, while the 3D plane fit method performed better on position estimates. The results show that both methods perform well on point clouds with high levels of noise.

Datasets for hand-eye calibration were created using a robot manipulator with a Zivid One structured light 3D scanner attached to the end-effector. Hand-eye calibration resulted in the 3D correspondence method performing slightly better than the 3D plane fit method overall. The extrinsic camera calibration method implemented in OpenCV generally performed on a comparable level to the 3D methods. OpenCV achieved the lowest translation error on some datasets, but the rotational errors were generally higher than for the 3D methods. The 3D correspondence method was the most consistent overall resulting in both low translation and rotation error, while the two other methods generally seem to optimize for one of them.

Using 3D point clouds extracted from simple planar calibration objects allow for fast extrinsic camera calibration, since robust corner detection algorithms can be used for point matching. The overall accuracy is dependent on the accuracy of the 3D scanner and the precision is dependent on the level and distribution of noise in the point clouds. Using these methods are a viable alternative to the conventional methods utilizing coplanar

image points for extrinsic camera calibration when calibrating 3D scanners for use in vision tasks for robotic applications.

7.2 Future Work

The results of the work presented in this report can be used to calibrate vision systems used in robotics. The natural continuation of this is to implement such a system to do various tasks such as assembly operations or random bin picking. Verifying the accuracy of the hand-eye calibration methods for practical applications would be an important step in doing so. Additionally a system for automatic dataset creation using optimal robot poses would be very beneficial for rapid and more consistent hand-eye calibration. This is because manual acquisition of datasets is slow and might not capture the optimal poses for hand-eye calibration.

Bibliography

[1] Y. C. Shiu and S. Ahmad, "Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax= xb", *ieee Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 16–29, 1989.

[2] 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*, vol. 5, no. 3, pp. 345–358, 1989.

[3] R. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses", *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[4] F. C. Park and B. J. Martin, "Robot sensor calibration: Solving ax= xb on the euclidean group", *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 717–721, 1994.

[5] R. Horaud and F. Dornaika, "Hand-eye calibration", *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[6] S. Kahn, D. Haumann, and V. Willert, "Hand-eye calibration with a depth camera: 2d or 3d?", in *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, IEEE, vol. 3, 2014, pp. 481–489.

[7] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[8] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Springer, 2017, vol. 118.

[9] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[10] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.", in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.

[11] R. S. Hartenberg and J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices", *Journal of applied mechanics*, vol. 77, no. 2, pp. 215–221, 1955.

[12] O. Egeland, *Robot vision*, Jan. 2019.

[13] F. Mokhtarian and R. Suomela, "Robust image corner detection through curvature scale space", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1376–1381, 1998.

[14] S. M. Smith and J. M. Brady, "Susan—a new approach to low level image processing", *International journal of computer vision*, vol. 23, no. 1, pp. 45–78, 1997.

[15] A. Georgopoulos, C. Ioannidis, and A. Valanis, "Assessing the performance of a structured light scanner", *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. Part 5, pp. 251–255, 2010.

[16] *Kr6 r900 sixx datasheet*, 0000-205-456, Rev. 24.1, KUKA Deutschland GmbH, May 2019.

[17] S. Sivcev. (2019). How to get good quality data on zivid calibration checkerboard, [Online]. Available: `https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/96895020/5.+How+to+get+good+quality+data+on+Zivid+calibration+checkerboard` (visited on 11/07/2019).

[18] Zivid. (2019). Software and documentation, [Online]. Available: `http://www.zivid.com/downloads` (visited on 12/02/2019).

Appendix A

Calibration Chessboards

Generated chessboards used for hand-eye calibration. Gray value in Figures A.2 and A.3 are the integer value for 8-bit gray scale encoding of the intensity of the gray squares. This means that the white squares have a value of 255, and black squares would have a value of 0.

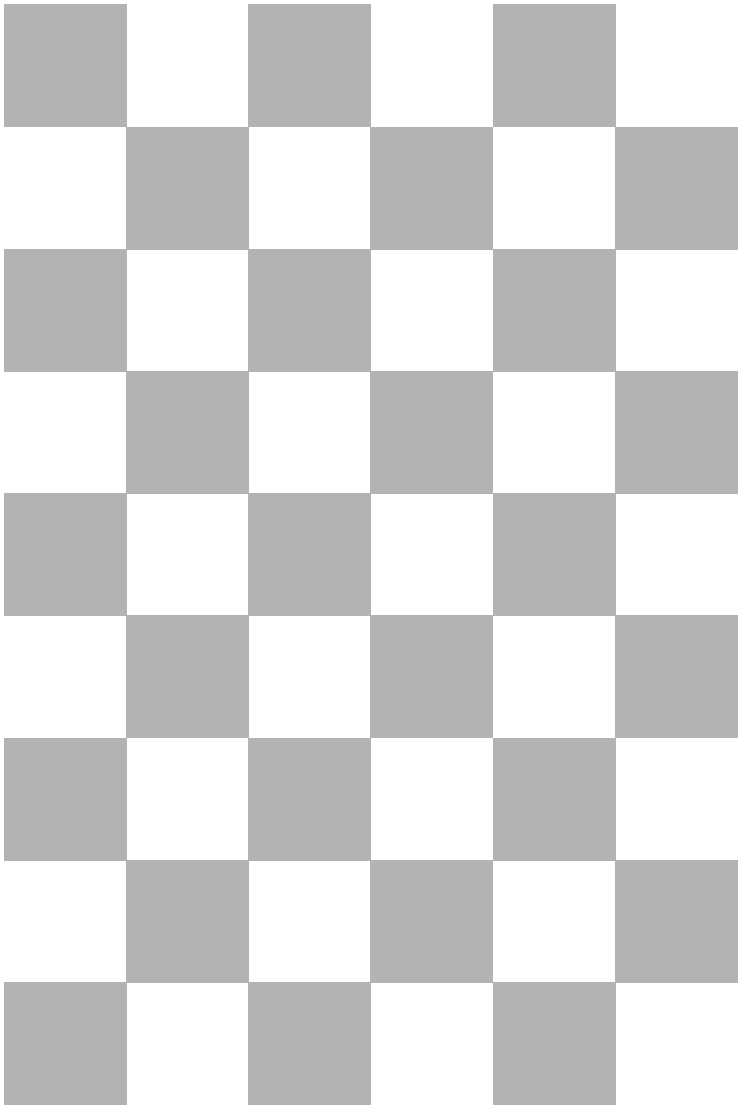


Figure A.1: 9×6. Square size 20.0 mm. [18]

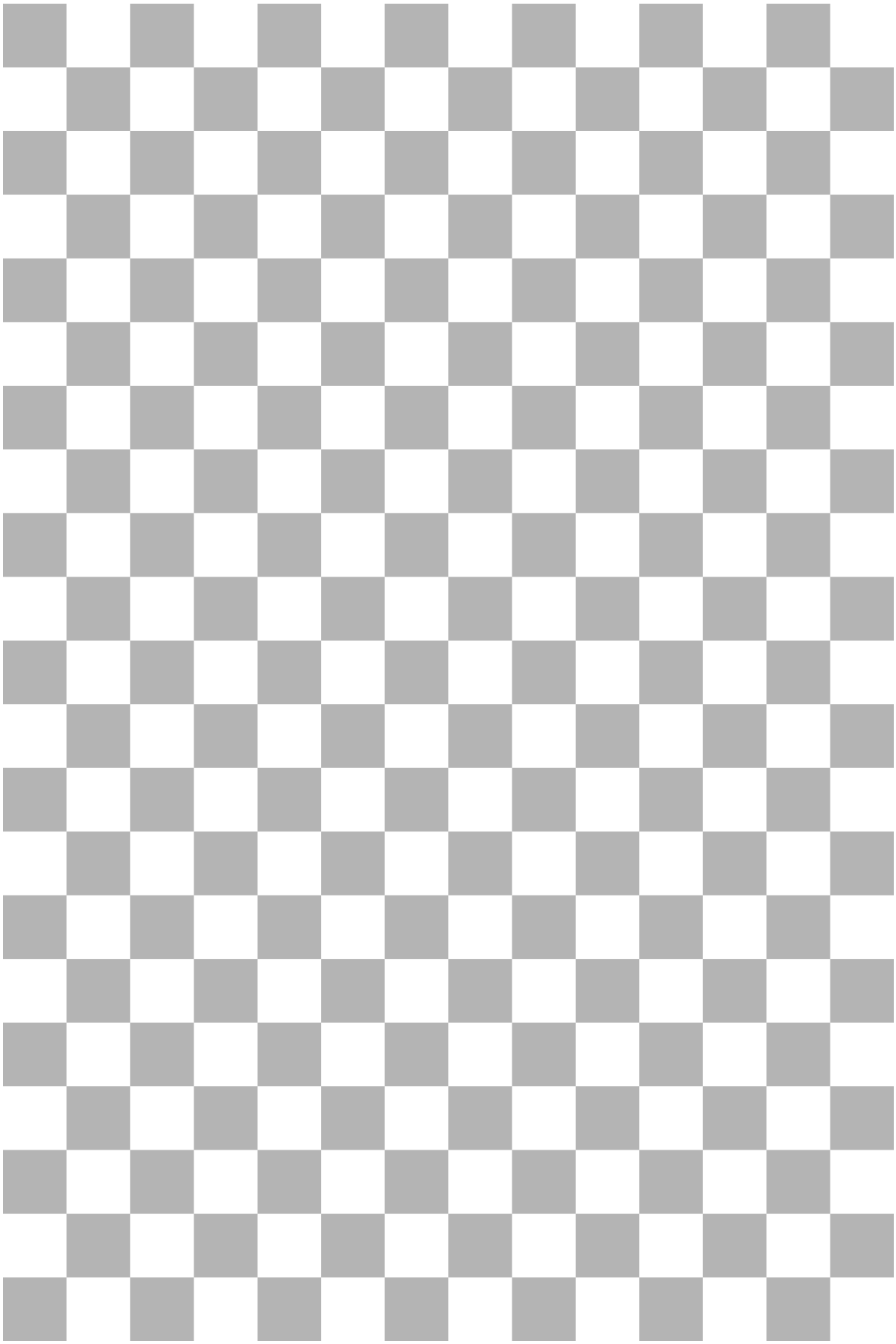


Figure A.2: 21×14. Gray value 180.

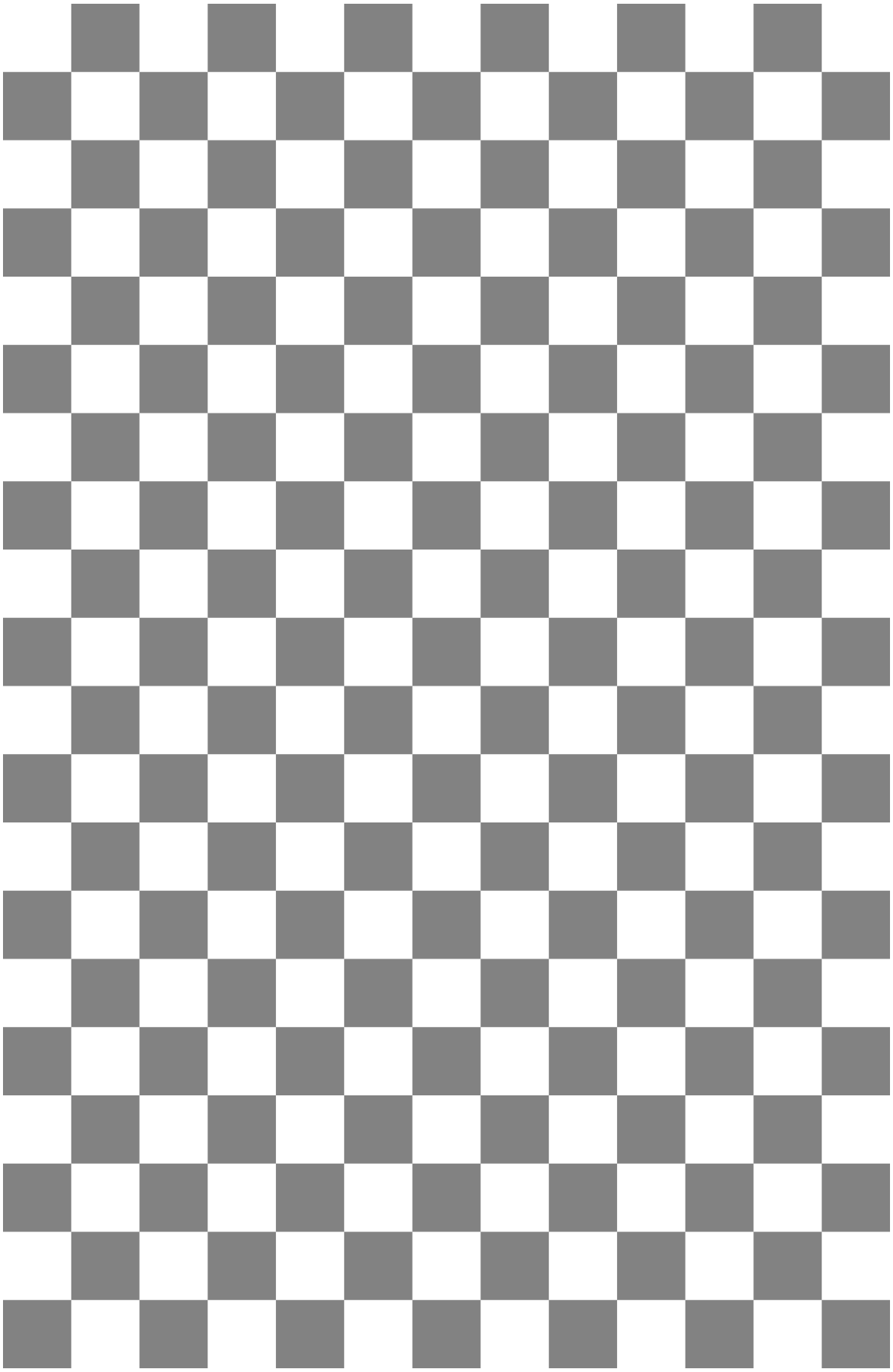


Figure A.3: 20×13. Gray value 130.

Appendix B

Code

Contains an overview of the available method calls and exposed member variables implemented in the files and classes used for hand-eye calibration. Each file is separated into different sections.

B.1 homogeneous_transformations.py

Members of HTransf class

```
HTransf.angle
HTransf.axis
HTransf.t
HTransf.skew_axis
HTransf.matrix
```

Methods of HTransf:

```
HTransf(a: float, k: np.ndarray, t: np.ndarray)
HTransf.from_matrix(m: np.ndarray)
HTransf.from_vecs(r: np.ndarray, t: np.ndarray)
HTransf.from_kuka_pose(x: float, y: float, z: float,
                       a: float, b: float, c: float)
HTransf.to_kuka_pose()
HTransf.to_mm()
HTransf.tr_rot()
HTransf.tr()
HTransf.angle_axis()
HTransf.rvec()
HTransf.inv()
HTransf.confirm_SO3()
```

B.2 utils.py

Available functions in file:

```
load_zdf(file: str)
load_t4s(path: str)
calibration_pts(nx: int, ny: int,
                square_size: float, pnt_clds: np.ndarray)
line_line_intersect(p1: float, p2: float, p3: float, p4: float)
center_pts(corners: list, nx: int, ny: int, num_imgs: int)
pnt_cld_transf(pnt_cld1: np.ndarray, pnt_cld2: np.ndarray)
park_martin(A: list, B: list)
calib_err_2D(hec, rob_poses: list, board_poses: list)
calib_err_3D(hec, rob_poses: list, board_pts: list)
interpolate_2D(f: list, xy: list, x: float, y: float)
generate_poses(n: int, noise_list: list)
plane_fit(pnt_cld: np.ndarray, nx: int, ny: int)
```

B.3 zivid_hand_eye_calibrator.py

Members of ZividHECalibrator class:

```
ZividHECalibrator.app
ZividHECalibrator.nx
ZividHECalibrator.ny
ZividHECalibrator.square_size
ZividHECalibrator.images
ZividHECalibrator.num_imgs
ZividHECalibrator.point_clouds
ZividHECalibrator.XYZs
ZividHECalibrator.chessboard_poses
ZividHECalibrator.camera_poses
ZividHECalibrator.robot_poses
ZividHECalibrator.rgbs
ZividHECalibrator.camera_matrix
ZividHECalibrator.dist_coeffs
ZividHECalibrator.image_files
ZividHECalibrator.pose_files
ZividHECalibrator.corners
ZividHECalibrator.center_points
```

```
ZividHECalibrator.object_planes
ZividHECalibrator.board_points
ZividHECalibrator.rvecs
ZividHECalibrator.tvecs
ZividHECalibrator.HE_calib
ZividHECalibrator.method
```

Methods of ZividHECalibrator class:

```
ZividHECalibrator(nx: int, ny: int, sqrSize: float)
ZividHECalibrator.load_robot_poses(path: str)
ZividHECalibrator.load_zdfs(path: str)
ZividHECalibrator.load_zdfs_parallel(folder_path: str)
ZividHECalibrator.calculate_chessboard_poses_2D()
ZividHECalibrator.calculate_chessboard_poses_3D()
ZividHECalibrator.draw_coord_sys(imgNb: int)
ZividHECalibrator.draw_coord_sys(self, imgNb: int)
ZividHECalibrator.viz_cam_pose(
        focus: str = 'objectCentric', axLen: int = 50)
ZividHECalibrator.viz_rob_pose(self, axLen: int = 50)
ZividHECalibrator.viz_HE_transf(
        X_HE: list, showTranslation: bool = True, inCamera=False)
ZividHECalibrator.set_intrinsics(pathToK: str, pathToDist: str)
ZividHECalibrator.calculate_relative_poses(
        pose_pairs: int = -1, use_board_pts: bool = True)
ZividHECalibrator.HE_calibration(A: list, B: list)
ZividHECalibrator.calib_error()
ZividHECalibrator.err_iter(plot=False)
```

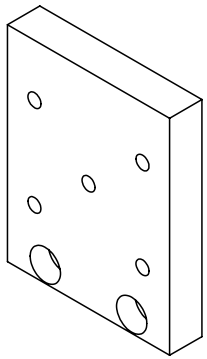
B.4 capture.py

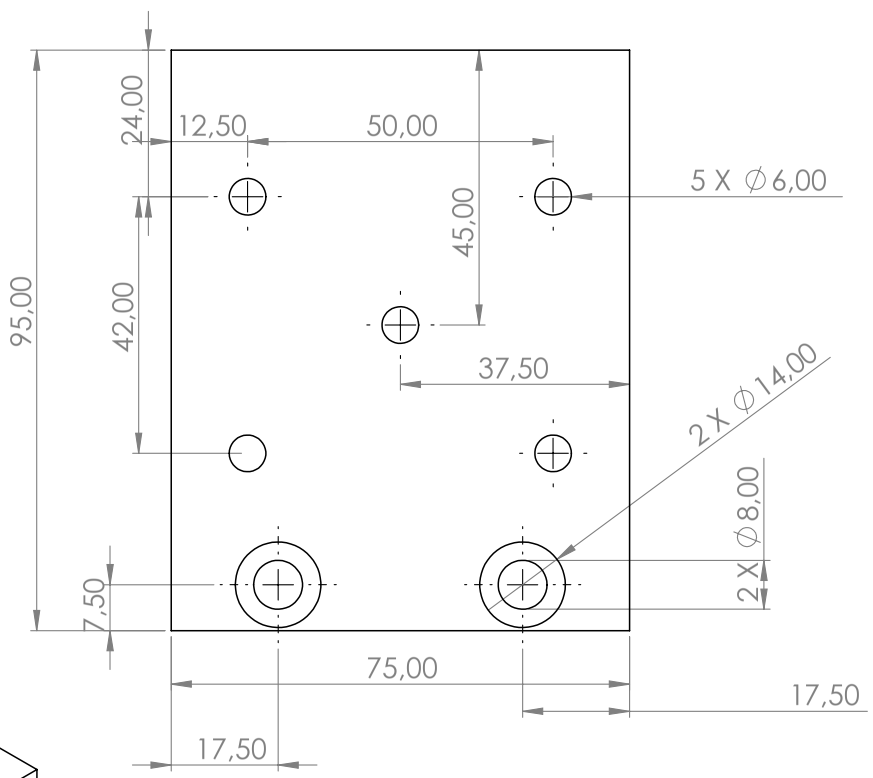
Available functions in file:

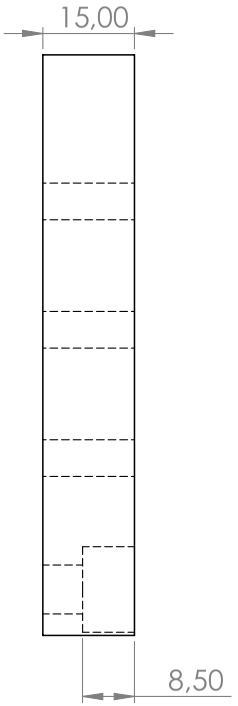
```
connect_to_camera()
capture_frame(camera)
save_robot_state(filename, xml)
pose_monitor(client)
```

Appendix C

Design Drawings







UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE		
DRAWN	Markus Bjønnes		Tlf.: 40295635		
CHK'D					
APPV'D					
MFG					
Q.A					

TITLE: Brakett for Zivid one (C)

MATERIAL: Aluminium

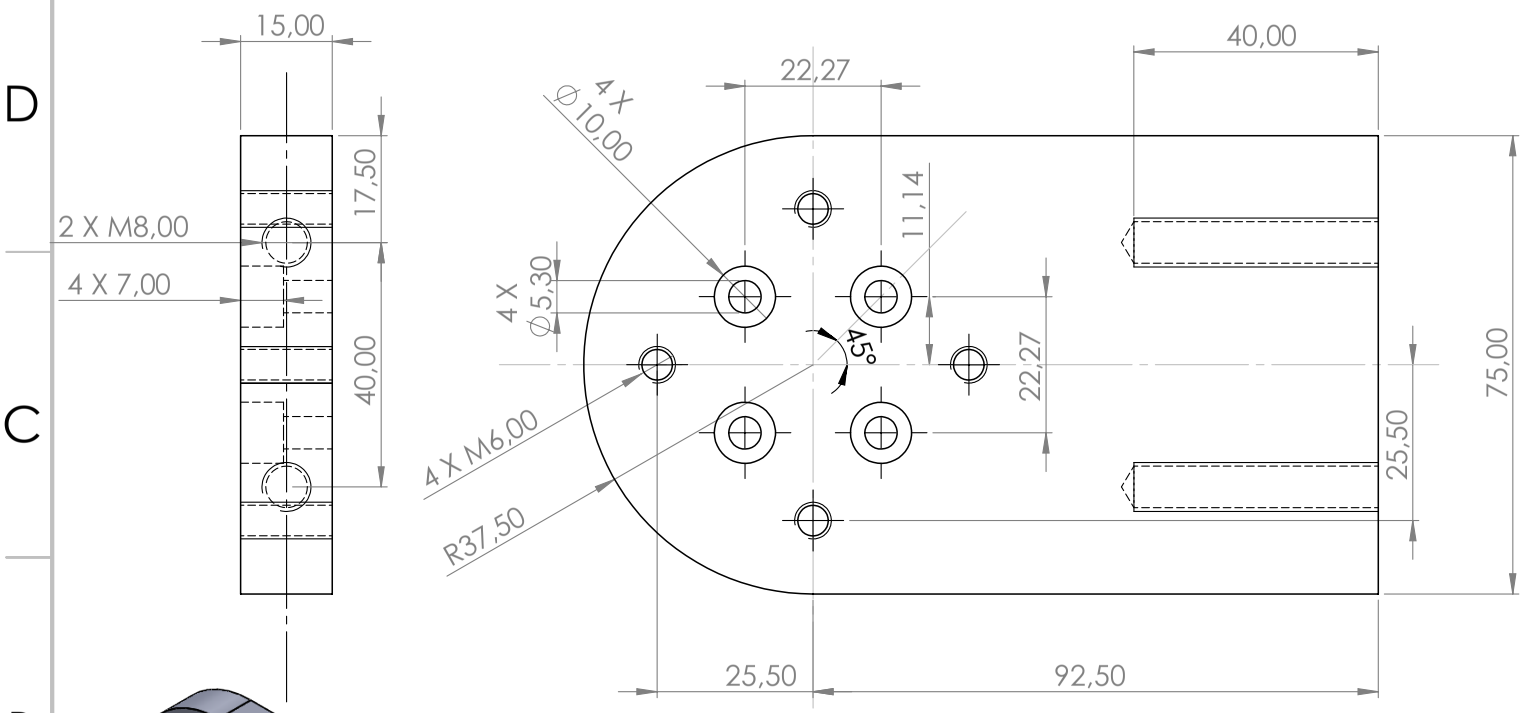
DWG NO. camera_bracket_C

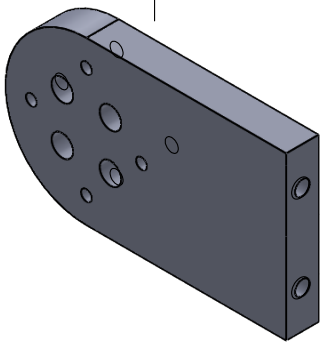
A4

WEIGHT:

SCALE: 1:1

SHEET 1 OF 1





UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN	Markus Bjønnes		Tlf.: 40295635	Brakett for Zivid one (R)			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.		A4
				Aluminium	camera_bracket_R		
				WEIGHT:	SCALE: 1:1	SHEET 1 OF 1	