Calibration of spatial relationships between multiple robots and sensors

by

Zachariah Goh

An essay submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science.

Baltimore, Maryland

July, 2016

Approved

[illegible]

Gregory S. Chirikjian

Abstract

Classic hand-eye calibration methods have been limited to single robots and sensors. Recently a new calibration formulation for multiple robots has been proposed that solves for the extrinsic calibration parameters for each robot simultaneously instead of sequentially. The existing solutions for this new problem required data to have correspondence, but Ma, Goh and Chirikjian (MGC) proposed a probabilistic method to solve this problem which eliminated the need for correspondence. In this thesis, the literature of the various robot-sensor calibration problems and solutions are surveyed, and the MGC method is reviewed in detail. Lastly comparison with other methods using numerical simulations were carried out to draw some conclusions.

Approved

Primary Reader: Dr. Gregory Chirikjian [illegible]

Secondary Reader: Dr. Simon Leonard

Acknowledgments

Thanks to my Lord and Savior Jesus Christ, and also to family and friends who have supported me during my time in graduate school. Special thanks to Qianli Ma who shared his expertise in the calibration field with me. I also would like to express my gratitude to Dr. Gregory Chirikjian and Dr. Nassir Navab for advice and guidance provided throughout the entire process, and to Dr. Simon Leonard for spending his time to give helpful and detailed feedback.

Dedication

For my Dad and Mom.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The proliferation of inexpensive and reliable sensors in the recent decade caused robotics to take off not only in corporations and government but also in academia and the backyard of hobbyists. This can be seen from the burgeoning market of commercial-off-the-shelf (COTS) components catering to aerial robots, mobile vehicles and humanoid robots.

1.1 Background and Motivation

Before robots can use data from sensors to perform intelligent tasks, the sensors often have to be calibrated with the robot, which involves establishing the pose (i.e position and orientation) between the robot and sensor. Once that is done, the robot can use data from the sensor to tell where objects are and hence the robot knows where to move to reach it. When the sensor is a camera, the calibration solution can also be used to move the robot, and hence the camera, such that it can capture images for 3D reconstruction of a scene.

In the literature on sensors, the term "calibration" can be used to denote two different but related processes. A sensor relies on some physical process to sense or measure the physical world. But in order to give meaningful and accurate measurements, we have to obtain intrinsic parameters that enable the conversion of raw sensor values to real-world units and this is called "calibration". For instance, a calibrated camera will give us the dimensions of objects from images in units like meters instead of in pixels. The second way people talk about "calibration" is obtaining extrinsic parameters like the relative pose of a sensor with another object, which could be another sensor or a robotic manipulator. Estimating the pose of the sensor to the external world is also related to what the computer vision community calls "registration" which is to obtain the spatial relationship of objects in an image to those in a reference image. The second meaning is what I will be using in this thesis.

Almost 30 years ago, the problem of estimating the spatial relationship of the sensor to the tool flange, known as the hand-eye calibration problem, was first formulated for a robot manipulator with a sensor attached to the tool flange. It was modeled as a mathematical problem that people over the years solved using many different techniques depending on the factors under consideration. Then came subsequent attempts to expand this formulation to include other types of robotic systems with sensors. One such system, which was proposed two years ago, involves multiple robots, each with its own attached sensor. This new problem formulation seeks to solve the hand-eye calibration problem for each robot simultaneously by using the spatial relationship between any two pairs of robots.

As this problem is fairly new, not much comparison and analysis have been done for the different methods of solving it. Each method has its strengths and weaknesses and so this research proposes to collect data to compare and evaluate properties like robustness, accuracy and ease of use.

1.2 Plan

In this thesis, one such calibration technique will be investigated, which is called the Ma, Goh and Chirikjian (MGC) method after the authors in [23], and compare it with the other solutions in the literature. The strengths and limitations of the method will be evaluated and its performance on simulated data under various conditions will be collected.

Firstly, the preliminary mathematical content required in Chapter 2 will be listed, followed by a literature survey of robot and sensor calibration problems and their solutions in Chapter 3. Then in Chapter 4, the algorithm of the MGC method will be reviewed and in Chapter 5, it will be compared to other methods. Lastly, results and insights will be highlighted and new directions for research will be suggested.

Chapter 2

Mathematical Background

This chapter, especially Section 2.1 and 2.2, lays down the convention of mathematical notations used in the literature review chapter but is not meant to provide a primer on the wide range of topics covered there. Readers who are interested should refer to each paper for more information. Section 2.3 and 2.4 covers the mathematics required for the MGC method.

2.1 Notation and Basic Concepts

1. The transpose of a matrix is denoted with the superscript $\top$ symbol instead of the letter T.

2. The matrix of zeros is denoted by $\mathbb{O}_{m\times n}$ with the subscript indicating the size of the matrix. And $\mathbb{O}_n$ means that it is a square matrix with dimension n.

3. The identity matrix of size n is denoted by $\mathbb{I}_n$.

4. The Kronecker or tensor product of two matrices A, B:

$$A_{m\times n} \otimes B_{p\times q} = \begin{bmatrix} A_{11}B & \dots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \dots & A_{mn}B \end{bmatrix}_{mp\times nq}$$

5. The vec() operator turns a $m \times n$ matrix A to a $mn \times 1$ vector:

$$\text{vec}(A) = \begin{bmatrix} A_{11} \\ \vdots \\ A_{1n} \\ A_{21} \\ \vdots \\ A_{mn} \end{bmatrix}$$

6. A rotation matrix R in 3D space is an element of the Lie group $SO(3)$ where

$$SO(3) := \left\{ R \in \mathbb{R}^{3\times 3} \mid R^\top R = RR^\top = \mathbb{I}, \det(R) = 1 \right\}.$$

7. A homogeneous transformation H in 3D space is a 4×4 matrix of the form

$$H = \begin{bmatrix} R & t \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix} \in SE(3) \tag{2.1}$$

where

$$SE(3) := \left\{ H(R, t) \in \mathbb{R}^{4\times 4} \mid R \in SO(3), t \in \mathbb{R}^{3\times 1} \right\}$$

Such a matrix is used to represent rigid body motions in 3D space which are composed of a rotation (represented by the matrix R) and a translation (represented by the vector t).

8. The mapping $\{\mathbb{R}^3 \rightarrow \mathfrak{so}(3)\}$ of a vector

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

to a skew-symmetric matrix is defined using the "hat" $\wedge$ operator

$$[\omega]^\wedge = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \mathfrak{so}(3)$$

where $\mathfrak{so}(3)$ is an example of a Lie algebra, whose matrix exponential gives its counterpart in the Lie group $SO(3)$:

$$\exp{[\omega]^\wedge} = R \in SO(3)$$

and the matrix logarithm does the inverse:

$$\log R = [\omega]^\wedge .$$

Similarly, the mapping $\{\mathbb{R}^6 \rightarrow \mathfrak{se}(3)\}$ of the vector

$$h = \begin{bmatrix} h_1 \\ \vdots \\ h_6 \end{bmatrix} \in \mathbb{R}^6$$

to a matrix $[h]^\wedge \in \mathfrak{se}(3)$ is called a twist and is defined as

$$[h]^\wedge = \begin{bmatrix} 0 & -h_3 & h_2 & h_4 \\ h_3 & 0 & -h_1 & h_5 \\ -h_2 & h_1 & 0 & h_6 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathfrak{se}(3).$$

Similarly, the exponential of $[h]^\wedge$ maps to an element in $SE(3)$.

9. The "vee" $\vee$ operator performs the reverse mapping $\{\mathfrak{so}(3) \to \mathbb{R}^3\}$, so operation on a skew-symmetric matrix gives

$$\begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}^{\vee} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}.$$

A summary of the various mappings between Lie algebras and Lie groups is shown in Table 2.1.

Vector	map	Lie algebra	map	Lie group
$\omega \in \mathbb{R}^3$	$\xrightarrow{\wedge}$	$[\omega]^{\wedge} \in \mathfrak{so}(3)$	$\xrightarrow{\exp}$	$\exp([\omega]^{\wedge}) \in SO(3)$
$\log^{\vee} R \in \mathbb{R}^3$	$\xleftarrow{\vee}$	$\log R \in \mathfrak{so}(3)$	$\xleftarrow{\log}$	$R \in SO(3)$
$h \in \mathbb{R}^6$	$\xrightarrow{\wedge}$	$[h]^{\wedge} \in \mathfrak{se}(3)$	$\xrightarrow{\exp}$	$\exp([h]^{\wedge}) \in SE(3)$
$\log^{\vee} H \in \mathbb{R}^6$	$\xleftarrow{\vee}$	$\log H \in \mathfrak{se}(3)$	$\xleftarrow{\log}$	$H \in SE(3)$

Table 2.1: Summary of the relationships between Lie algebras and Lie groups

10. The Dirac Delta function defined on $SE(3)$ is

$$\delta(H) = \begin{cases} +\infty, & H = \mathbb{I}_4 \\ 0, & H \neq \mathbb{I}_4 \end{cases}$$

with the constraint

$$\int_{SE(3)} \delta(H)\, dH = 1$$

where integraton over $SE(3)$ is defined below.

A Dirac Delta function shifted to $X \in SE(3)$ is denoted by

$$\delta_X(H) = \delta(X^{-1}H). \tag{2.2}$$

11. Integration on $SE(3)$ is expressed differently depending on the parametrizations. For instance, if ZXZ Euler angles are used for a rotation R, then

$$R = R_Z(\alpha)R_X(\beta)R_Z(\gamma)$$

where $R_i(\phi)$ represents a counterclockwise rotation by ϕ about axis i. And if translations are expressed in Cartesian coordinates in the xyz frame, then

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}.$$

Hence using these parametrizations of R, t for a homogeneous transformation H like in (2.1), the integral of a function in $SE(3)$ is

$$\begin{aligned} \int_{SE(3)} f(H)\,dH &= \int_{\mathbb{R}^3}\int_{SO(3)} f(H(R,t))\,dR\,dt \\ &= \int_{\mathbb{R}^3}\int_{SO(3)} f(H(R,t))\,\sin\beta\,d\alpha d\beta d\gamma\,dt_x dt_y dt_z \end{aligned}$$

[2] where $t_x, t_y, t_z \in \mathbb{R}$ and $(\alpha, \beta, \gamma) \in [0, 2\pi] \times [0, \pi] \times [0, 2\pi]$.

2.2 Quaternions and Dual Quaternions

1. Quaternions $\mathbf{q}$ are a generalization of complex numbers with the form

$$\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3,$$

where

$$i^2 = j^2 = k^2 = ijk = -1.$$

It can also be written in vector form as

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \begin{matrix} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \vec{q} \end{matrix}$$

where $\vec{q} \in \mathbb{R}^3$ is the vector or imaginary part of $\mathbf{q}$ and q_0 is the real part. If $\mathbf{q}$ is a unit quaternion, it has the property that

$$\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1.$$

Unit quaternions are one way to parametrize rotations in $SO(3)$ using 4 parameters, other methods include Euler angles and angle-axis parametrizations.

2. Multiplication of two quaternions

$$\mathbf{p} = \begin{bmatrix} p_0 \\ \vec{p} \end{bmatrix} \quad \text{and} \quad \mathbf{q} = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix}$$

is denoted by the symbol $\star$ and defined as

$$\mathbf{p} \star \mathbf{q} = \begin{bmatrix} p_0 q_0 - \vec{p}^{\top} \vec{q} \\ p_0 \vec{q} + q_0 \vec{p} + \vec{p} \times \vec{q} \end{bmatrix} \tag{2.3}$$

3. Quaternions can also be formulated as 4×4 matrices which helps to recast quaternion multiplication in (2.3) as matrix multiplication, which could be helpful to "switch" the order of multiplication so that terms could be factored out. In the transformation of a quaternion to a matrix, the uppercase letter is used

and with overhead $+$ and $-$ symbols as defined here

$$\mathbf{p} \star \mathbf{q} = \overset{+}{\mathbf{P}}\mathbf{q} = \overset{-}{\mathbf{Q}}\mathbf{p}$$

where

$$\begin{aligned}\overset{+}{\mathbf{P}} &:= \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix}, & \overset{-}{\mathbf{Q}} &:= \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \\ &= \begin{bmatrix} p_0 & -\vec{p}^\top \\ \vec{p} & p_0\mathbb{I} + [\vec{p}]^\wedge \end{bmatrix} & &= \begin{bmatrix} q_0 & -\vec{q}^\top \\ \vec{q} & q_0\mathbb{I} - [\vec{q}]^\wedge \end{bmatrix}\end{aligned} \tag{2.4}$$

4. The conjugate $\bar{\mathbf{q}}$ of a quaternion is

$$\bar{\mathbf{q}} = \begin{bmatrix} q_0 \\ -\vec{q} \end{bmatrix}$$

which represents an inverse rotation of $\mathbf{q}$.

5. Relationship to axis-angle: If a quaternion $\mathbf{q}$ represents a rotation by $\theta \in \mathbb{R}$ about an axis $\vec{k} \in \mathbb{R}^3$, then it can be written as

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \vec{k}\sin\frac{\theta}{2} \end{bmatrix}.$$

6. Dual quaternions $\check{\mathbf{q}}$ are the algebraic counterpart of screws and are denoted by

$$\check{\mathbf{q}} = \begin{bmatrix} \check{q}_0 \\ \check{\vec{q}} \end{bmatrix}$$

where $\check{q}_0$ is a dual number and $\check{\vec{q}}$ is a dual vector. Since screws represents rigid body motion, dual quaternions also contain information about rotation and translation, in contrast to quaternions which parameterizes rotations only. A dual quaternion is made of two ordinary quaternions and can be written as

$$\breve{\mathbf{q}} = \mathbf{q} + \varepsilon \mathbf{q}', \qquad \varepsilon^2 = 0 \tag{2.5}$$

where the quaternion $\mathbf{q}$ is the real or rotational part and the quaternion $\mathbf{q}'$ is the dual or displacement part. The conjugate of a dual quaternion is analagously denoted as $\bar{\breve{\mathbf{q}}}$. Multiplication of two dual quaternions is denoted by the $\odot$ symbol. The properties of dual quaternions are presented in more detail in [9].

2.3 Probability

1. A Gaussian probability density function (p.d.f) $\rho(\cdot)$ on $SE(3)$ is

$$\rho(H; M, \Sigma) = \frac{1}{(2\pi)^3\sqrt{|\det(\Sigma)|}} \exp\left[-\frac{1}{2}F(M^{-1}H)\right]$$

where

- $H \in SE(3)$ is a homogeneous transformation,
- $M \in SE(3)$ is the mean of the p.d.f.,
- $\Sigma \in \mathbb{R}^{6\times 6}$ is the covariance matrix of the p.d.f. which is positive definite,
- $F(H) = \left[(\log H)^\vee\right]^\top \Sigma^{-1} \left[(\log H)^\vee\right]$.

2. Given a probability density function $f(H)$, the mean M satisfies

$$\int_{SE(3)} \log\left(M^{-1}H\right) f(H)\, dH = \mathbb{O}_4, \tag{2.6}$$

and in turn, the covariance Σ is defined by

$$\Sigma := \int_{SE(3)} \left(\log(M^{-1}H)\right)^{\vee} \left[(\log(M^{-1}H))^{\vee}\right]^{\top} f(H)\, dH \tag{2.7}$$

3. The convolution (denoted by $*$) of two probability density functions f_1, f_2 on $SE(3)$ is defined as

$$(f_1 * f_2)(H) = \int_{SE(3)} f_1(K) f_2(K^{-1}H)\, dK$$

where $K \in SE(3)$ is a dummy integration variable. Convolution of a probability density function f with a Dirac Delta function δ is analagous to the one-dimensional case:

$$(f * \delta)(H) = \int_{SE(3)} f(K)\delta(K^{-1}H)\, dK = f(H) \tag{2.8}$$

4. The property of bilinearity for the convolution of two probability density functions $f = a_1 f_1 + a_2 f_2$ and $g = b_1 g_1 + b_2 g_2$ means that

$$(a_1 f_1 + a_2 f_2) * g = a_1(f_1 * g) + a_2(f_2 * g)$$

and

$$f * (b_1 g_1 + b_2 g_2) = a_1(f * g_1) + b_1(f * g_2)$$

where $a_1, a_2, b_1, b_2 \in \mathbb{R}$.

5. A probability density function $f(H)$ on $SE(3)$ is said to be highly "focused" or "concentrated" when the norm of its covariance Σ satisfies

$$\|\Sigma\|_2 \ll 1, \tag{2.9}$$

where the induced 2-norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined as

$$\|A\|_2 := \max_{\|x\|\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

and the usual Euclidean norm for vectors $x, Ax \in \mathbb{R}^n$ is meant. This condition involves mainly the rotation component of Σ and hence the spread in the orientations of $f \in SE(3)$ has to be small.

6. Given two highly focused probability density functions f_1, f_2 on $SE(3)$, the mean of their convolution $(f_1 * f_2)(H)$ is

$$M_{1*2} = M_1 M_2 \tag{2.10}$$

and the covariance is

$$\Sigma_{1*2} = \mathrm{Ad}\left(M_2^{-1}\right) \Sigma_1 \, \mathrm{Ad}^\top\!\left(M_2^{-1}\right) + \Sigma_2 \tag{2.11}$$

where the Adjoint operator on $H \in SE(3)$ is defined as

$$\mathrm{Ad}(H) := \begin{bmatrix} R & \mathbb{O}_3 \\ [t]^\wedge R & R \end{bmatrix}. \tag{2.12}$$

Furthermore, the following approximation holds for probability density functions that are highly focused:

$$\sum_{i=1}^{n} f_1^{(i)} f_2^{(i)} \approx \left(\sum_{i=1}^{n} f_1^{(i)} \right) \left(\sum_{i=1}^{n} f_2^{(i)} \right). \tag{2.13}$$

2.4 Distance metrics

We now briefly mention distance metrics because it allows for a comparison of two objects, which in this context are translation vectors, rotation matrices and homogeneous matrices. In layman terms, a metric is simply a way of expressing how "near" or "far" apart two objects are. For points in 3D space, the most intuitive and common measure of distance between two points is the Euclidean norm or straight line distance. But this metric would not be easy for comparison because it depends on the units of length used. Hence given two translation vectors $t_A, t_B \in \mathbb{R}^3$, if we want to compare the distance of t_A relative to t_B, we divide by the Euclidean norm of t_B to give the relative translation which is a unitless value:

$$d(t_A, t_B) = \frac{\|t_A - t_B\|_2}{\|t_B\|_2}. \tag{2.14}$$

For rotations in 3D space, an intuitive metric is the smallest angle $\theta_{A,B}$ between the two orientations $R_A, R_B \in SO(3)$ which can be computed as such:

$$d(R_A, R_B) = |\theta_{A,B}| = \|\log^\vee(R_A^\top R_B)\|_2. \tag{2.15}$$

Chapter 3

Literature Review

This is a literature review of hand-eye calibration in robotics throughout its approximately 25 year history and the related problems of hand-eye/robot-world and multi-robot/sensor calibration. First I give an overview of the various calibration problems in Section 3.1, then I highlight the common methods used to solve the various problems in Section 3.2. This review does not include current techniques that apply only when the sensor to be calibrated is a camera. Such methods usually involve computer vision concepts like structure-from-motion or image projection using a pinhole camera model. I have chosen not to include them because when modeling the problem, they usually make assumptions that cause the methods to be narrowly focused only on specific robotic systems.

3.1 Calibration problems

3.1.1 Hand-eye calibration

The basic hand-eye calibration problem seeks to identify the position and orientation between the frame of the a robot's flange (hand) and the frame of a sensor (eye) mounted on the flange, hence the term "eye-in-hand" is also used to describe this system. The data that we collect are relative transformations that can be computed using absolute transformations from the robot and sensor, as seen in Figure 3.1. The robot moves to different positions and uses its sensor to locate the same target or marker (that remains stationary when the robot is moving). The relative transformation of the flange between poses, A, can then be obtained from the forward kinematics of the robot in each pose, and the relative transformation of the sensor between poses, B, can be computed from the poses of the marker as located by the sensor at each robot position. This problem is commonly formulated as a matrix equation of homogeneous transformation matrices:

$$AX = XB \tag{3.1}$$

where X is the transformation of the sensor frame relative to the robot frame which we seek to solve for, given the relative transformations A and B [30]. A related robot system is when the sensor is stationary while the marker is attached to the robot flange. The robot can only move to positions where the sensor can locate the marker, but otherwise the same matrix equation (3.1) can still be formulated although the relationships that A and B represent may be different.

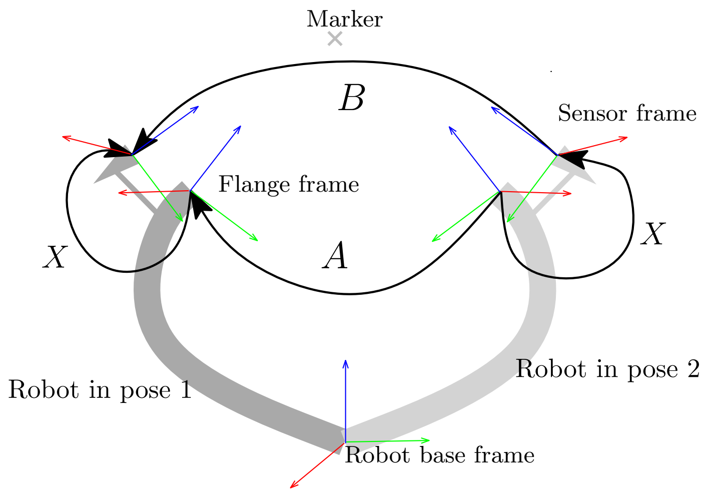

Figure 3.1: Hand-eye calibration formulated as $AX = XB$.

Since A, X, B are homogeneous transformations, the matrix equation $AX = XB$ can be written as

$$\overbrace{\begin{bmatrix} R_A & t_A \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} R_X & t_X \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}}^{X} = \overbrace{\begin{bmatrix} R_X & t_X \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}}^{X} \overbrace{\begin{bmatrix} R_B & t_B \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}}^{B}$$
$$\begin{bmatrix} R_A R_X & R_A t_X + t_A \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} R_X R_B & R_X t_B + t_X \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}$$

and hence the rotation can be obtained by solving [30]

$$R_A R_X = R_X R_B \tag{3.2}$$

and the translation by solving

$$R_A t_X + t_A = R_X t_B + t_X \tag{3.3}$$

Solving (3.1) in this way is known as a sequential method because it involves solving for R_X and t_X sequentially. One disadvantage of sequential type solutions was that any error obtained in the calculation of the rotation R_X would be propagated to the computation of the translation t_X. Simultaneous type methods solved for the rotation and translation at the same time and could eliminate such sources of error propagation. However for such methods, the results could be sensitive to the units of translation, whether it is millimeters or inches. This is because they give equal weight to the rotation and translation components of X during the estimation. An overview of solutions to (3.1) is given in [22, 29].

It has been shown [30] that in the absence of noise, only two sets of A and B are necessary to obtain a unique X but the two sets must be nondegenerate. This means that there must be at least three poses of the robot to obtain two sets of A and B. Additional considerations when solving such a system are accounting for noisy data and when the A_i and B_i data may not be in sync.

3.1.2 Hand-eye/Robot-world calibration

If the robot-world relationship needs to be obtained together with the hand-eye relationship X, then the matrix equation to be solved is

$$AX = YB \tag{3.4}$$

where Y is the pose of the robot base frame to the world frame (see Figure 3.2). In this system, the robot still moves to different poses so that the sensor locates the same marker at different positions. Here, A and B are absolute, not relative, transformations; A is the transformation of the flange from the robot base and is obtained using forward kinematics, while B is the pose of the marker as seen by the sensor. As in hand-eye calibration, the sensor and marker for this system can be swapped and we can still formulate the same matrix equation (3.4).

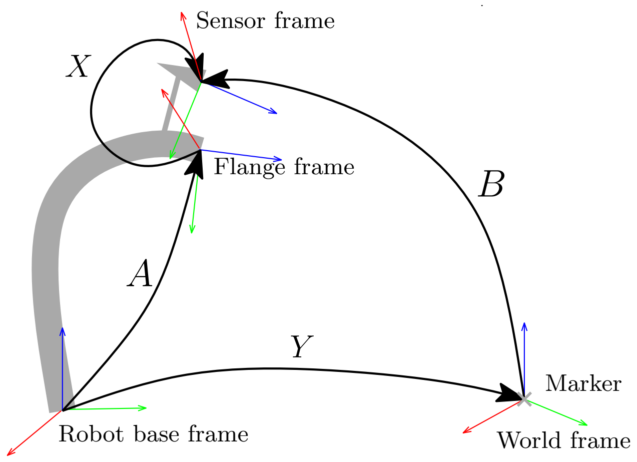

Figure 3.2: Hand-eye/robot-world calibration formulated as $AX = YB$

Similar to the $AX = XB$ case, the matrix equation $AX = YB$ can be written as

$$\begin{bmatrix} R_A & t_A \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}\begin{bmatrix} R_X & t_X \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} R_Y & t_Y \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}\begin{bmatrix} R_B & t_B \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}$$
$$\begin{bmatrix} R_A R_X & R_A t_X + t_A \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} R_Y R_B & R_Y t_B + t_Y \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix}$$

and hence the rotation can be obtained by solving

$$R_A R_X = R_Y R_B \tag{3.5}$$

and the translation can then be obtained by solving

$$R_A t_X + t_A = R_Y t_B + t_Y \tag{3.6}$$

It has been shown [41] that in the absence of noise, only three nondegenerate sets of A and B are necessary to obtain a unique X and Y. This means that there must be at least three poses of the robot to obtain three sets of A and B, since for the $AX = YB$ problem, A and B are not relative transformations like in the $AX = XB$ problem. An overview of solutions to (3.4) can be found in [29].

3.1.3 Multi-robot/sensor calibration

When multiple robots are involved and the pose between each robot needs to be calibrated, then the matrix equation to be solved can be formulated as [34]

$$AXB = YCZ \tag{3.7}$$

where the transformations represent different relationships depending on the robotic system. One such robotic system is shown in Figure 3.3 which shows three mobile robots each mounted with a camera that looks at another robot's marker to form a chain. Here A, B, C are the marker-camera transformations between different robots, while X, Y, Z are the marker-camera transformations within the same robot. More details of this and other relevant robotic systems are given in Section 4.1.

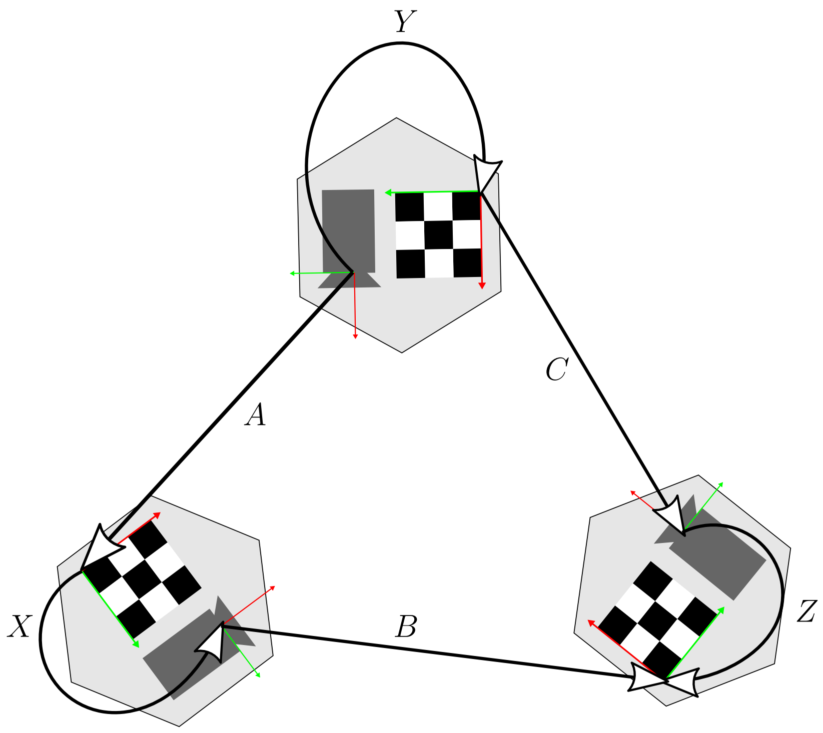

Figure 3.3: Top-down view of a team of mobile robots with the unknown calibrations formulated as $AXB = YCZ$

To solve (3.7) sequentially, write it as

$$\begin{bmatrix} R_A & t_A \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_X & t_X \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_B & t_B \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} R_Y & t_Y \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_C & t_C \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_Z & t_Z \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix}$$
$$\begin{bmatrix} R_A R_X R_B & R_A R_X t_B + R_A t_X + t_A \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} R_Y R_C R_Z & R_Y R_C t_Z + R_Y t_C + t_Y \\ \mathbb{O}_{1\times3} & 1 \end{bmatrix}$$

and hence the rotation can be obtained by solving

$$R_A R_X R_B = R_Y R_C R_Z \tag{3.8}$$

and the translation can then be obtained by solving

$$R_A R_X t_B + R_A t_X + t_A = R_Y R_C t_Z + R_Y t_C + t_Y \tag{3.9}$$

3.2 Current methods

The techniques in the literature for solving calibration problems (3.1), (3.4) and (3.7) are categorized as shown in Table 3.1. The methods are first classified within three main categories depending on the type of algorithm: closed-form, iterative or batch. They are then further subdivided depending on the type of mathematical tool used. The categories will be described in more detail in the following sub-sections.

3.2.1 Transform to Linear System

These methods typically set up a homogeneous linear equation of the form

$$\Gamma\xi = \gamma$$

where $\Gamma \in \mathbb{R}^{p\times q}, \xi \in \mathbb{R}^{q}, \gamma \in \mathbb{R}^{p}$. This can then be solved using the normal equations to give

$$\xi = \left(\Gamma^{\top}\Gamma\right)^{-1}\Gamma^{\top}\gamma$$

as the least squares solution. In the presence of noisy data, Γ is formed by stacking the quantities from n measurements of A_i, B_i. The resulting linear system is overdetermined and hence the "best fit" solution is taken to be the least squares solution.

Method	$AX = XB$	$AX = YB$	$AXB = YCZ$
Closed-form: Linear System			
3.2.1.1 Angle-axis	[30], [32], [33], [40]		
3.2.1.2 Quaternion	[8], [21], [41], [39]		[36]
3.2.1.3 Dual Quaternion	[9]	[18]	
3.2.1.4 Lie Group	[26]		
3.2.1.5 Kronecker Product	[5], [6], [20], [4]	[18], [28], [11]	[37]
Closed-form: Nonlinear System			
3.2.2.1 Constrained Optimization	[16]	[10]	
Iterative			
3.2.3.1 Nonlinear least squares	[42], [16], [12]	[10], [13]	[37]
3.2.3.2 Convex Optimization	[38]		
3.2.3.3 Global Optimization	[14],[27]	[14]	
3.2.3.4 Stochastic Optimization		[13]	
3.2.3.5 Jacobian Optimization	[25]		
3.2.3.6 Quaternion		[15]	
3.2.3.7 Lie Group	[3]		
3.2.3.8 Linear Approximation			[34, 36]
3.2.3.9 Statistical Model	[31], [1]	[31]	
Batch			
3.2.4.1 Probabilistic	[2], [24]	[19]	[23]

Table 3.1: Categories of methods for solving the three different types of calibration problems. The section numbers and citations link to the corresponding subsections in the electronic document.

3.2.1.1 Angle-axis

1. The earliest solutions to (3.1) were of the sequential type, with Shiu and Ahmad [30] being the first authors who provided a complete solution. They represented rotations using an angle-axis parametrization, which allowed them to recast the rotation component equation (3.2) as a system of linear equations

$$\Gamma_{9n\times 4n}\xi_{4n\times 1} = \gamma_{9n\times 1} \tag{3.10}$$

where Γ and γ were comprised of the axes of rotation of R_{A_i} and R_{B_i}, while ξ was comprised of cosines and sines of the angles of rotation. After obtaining the rotation R_X from ξ, they then reformulated (3.3) as another linear system

$$\begin{bmatrix} R_{A_1} - \mathbb{I}_3 \\ \vdots \\ R_{A_n} - \mathbb{I}_3 \end{bmatrix} t_X = \begin{bmatrix} R_X t_{B_1} - t_{A_1} \\ \vdots \\ R_X t_{B_n} - t_{A_n} \end{bmatrix} \tag{3.11}$$

whose least squares solution gave the translation t_X. The authors also proved that at least 2 relative rotations (i.e. $A_i, B_i, i = 1, 2$) are needed in order for a unique solution in the noiseless case, and the rotation axes of A_1, B_1 cannot be parallel or antiparallel to that of A_2, B_2 respectively.

2. Soon after [30] was published, Tsai and Lenz [32] came up with a sequential method that also used an angle-axis formulation. They converted rotation matrices, R_H, into angle-axis representations, $\text{Rot}(\vec{n}_H, \theta_H)$, where $\vec{n}_H$ and θ_H is the axis and angle of rotation of R_H respectively. (Here and in other places in this chapter, H represents the transformation A or B unless otherwise stated.) They then defined $\vec{k}_H := 2\,\vec{n}_H \sin\frac{\theta_H}{2}$ to form the system of linear equations

$$\begin{bmatrix} \left[\vec{k}_{A_1} + \vec{k}_{B_1}\right]^\wedge \\ \vdots \\ \left[\vec{k}_{A_n} + \vec{k}_{B_n}\right]^\wedge \end{bmatrix} \underbrace{\begin{bmatrix} \dfrac{1}{\sqrt{4 - \|\vec{k}_X\|^2}} \vec{k}_X \end{bmatrix}}_{\xi_{3\times 1}} = \begin{bmatrix} \vec{k}_{B_1} - \vec{k}_{A_1} \\ \vdots \\ \vec{k}_{B_n} - \vec{k}_{A_n} \end{bmatrix} \tag{3.12}$$

which could be solved using a least squares solution to recover R_X while the translation t_X was obtained by solving (3.11). They claimed their method was easier to formulate and faster to computer than Shiu and Ahmad's solution because Shiu and Ahmad's solution solved (3.10) for ξ whose dimension increased with the number of measurements n. But in Tsai and Lenz's method formulation in (3.12), the number of unknowns in ξ remains constant at 3.

3. Zhuang and Roth [40] improved on Shiu and Ahmad's method of obtaining the rotation R_X for the hand-eye calibration problem by using quaternion algebra to obtain a relationship between the axes and angle of rotation of R_A and R_B. This was then used to form a linear systems of equations

$$\begin{bmatrix} [k_{A_1} + k_{B_1}]^\wedge \\ \vdots \\ [k_{A_n} + k_{B_n}]^\wedge \end{bmatrix} \begin{bmatrix} \tan\left(\frac{\omega}{2}\right) k_X \end{bmatrix} = \begin{bmatrix} k_{B_1} - k_{A_1} \\ \vdots \\ k_{B_n} - k_{A_n} \end{bmatrix}$$

where $k_A, k_B, k_X \in \mathbb{R}^3$ were the axes of rotation for R_A, R_B, R_X respectively and ω is the angle of rotation for R_X. From the least squares solution $z \in \mathbb{R}^3$ of this system (assuming $\|z\| \neq 0$), they recovered R_X with

$$\begin{aligned} k_X &= \frac{z}{\|z\|}, \\ \omega &= 2\arctan\left(\frac{\max\{z_1, z_2, z_3\}}{\max\{k_{X,1}, k_{X,2}, k_{X,3}\}}\right) \end{aligned}$$

where the vectors $z = \begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^\top$ and $k_X = \begin{bmatrix} k_{X,1} & k_{X,2} & k_{X,3} \end{bmatrix}^\top$ were written in their components. As opposed to Shiu and Ahmad's, their method allowed the rotation axes of A_1, B_1 and A_2, B_2 to be antiparallel.

4. Wang [33] provided sequential solutions to the rotation and translation of (3.1) in his Class B calibration procedure. He used the angle-axis representation

to list the possible relationships between the rotational data $\{R_{A_i}\}$ and $\{R_{B_i}\}$, and provided closed-form solutions for each case. The translation was obtained by using the normal equations to solve (3.11). He also computed covariances of the rotation and translation estimates as a measure of accuracy, and did a sensitivity analysis of the resulting X. He compared his method with that of Shiu [30] and Tsai [32] and found that the Tsai method was the most accurate and efficient, followed by his method and then Shiu's.

3.2.1.2 Quaternion

1. Chou and Kamel [8] used quaternions to transform (3.2) into a homogeneous system of linear equations

$$\underbrace{\begin{bmatrix} 0 & -(k_{A_1} - k_{B_1})^\top \\ k_{A_1} - k_{B_1} & [k_{A_1}]^\wedge + [k_{B_1}]^\wedge \\ \vdots & \vdots \\ 0 & -(k_{A_n} - k_{B_n})^\top \\ k_{A_n} - k_{B_n} & [k_{A_n}]^\wedge + [k_{B_n}]^\wedge \end{bmatrix}}_{\Gamma_{4n\times 4}} \mathbf{q}_X = \mathbb{O}_{4n\times 1}$$

where k_A, k_B were the rotation axes that could be computed from the vector part $\vec{q}_A, \vec{q}_B$ of quaternions for A and B respectively. The SVD of Γ was then used to solve for the rotation R_X. To obtain the translation t_X, another set of linear equations

$$\begin{bmatrix} [k_{A_1}]^\wedge \\ \vdots \\ [k_{A_n}]^\wedge \end{bmatrix} t_X = \begin{bmatrix} \frac{1}{2\sin\left(\frac{1}{2}\theta_{A_1}\right)} \left(R_X t_{B_1} - t_{A_1}\right) \\ \vdots \\ \frac{1}{2\sin\left(\frac{1}{2}\theta_{A_n}\right)} \left(R_X t_{B_n} - t_{A_n}\right) \end{bmatrix}$$

was solved using SVD where (k_{A_i}, θ_{A_i}) was the angle-axis representation of R_{A_i}.

2. Then Lu and Chou [21] presented another method that they called the "eight space method", which solved for the rotation and translation of the hand-eye calibration problem simultaneously. Using (2.4), they reformulated the problem using quaternion algebra into an overdetermined linear system

$$\underbrace{\begin{bmatrix} \overset{-}{\mathbf{Q}}_{B_1}\left(\overset{+}{\mathbf{T}}_{A_1} - \overset{-}{\mathbf{T}}_{B_1}\right) & \overset{+}{\mathbf{Q}}_{A_1} - \overset{-}{\mathbf{Q}}_{B_1} \\ \overset{+}{\mathbf{Q}}_{A_1} - \overset{-}{\mathbf{Q}}_{B_1} & \mathbb{O}_4 \\ \vdots & \vdots \\ \overset{-}{\mathbf{Q}}_{B_n}\left(\overset{+}{\mathbf{T}}_{A_n} - \overset{-}{\mathbf{T}}_{B_n}\right) & \overset{+}{\mathbf{Q}}_{A_n} - \overset{-}{\mathbf{Q}}_{B_n} \\ \overset{+}{\mathbf{Q}}_{A_n} - \overset{-}{\mathbf{Q}}_{B_n} & \mathbb{O}_4 \end{bmatrix}}_{\Gamma_{8n\times 8}} \underbrace{\begin{bmatrix} \mathbf{q}_X \\ \overset{-}{\mathbf{Q}}_X \mathbf{t}_X \end{bmatrix}}_{\xi_{8\times 1}} = \mathbb{O}_{8n\times 1}$$

where $\overset{\pm}{\mathbf{Q}}_H$ is formed from the rotation of $H \in \{A, B, X\}$ that was parametrized as a quaternion, and $\overset{\pm}{\mathbf{T}}_H$ comes from the translation of H that is represented only in the vector part of a quaternion $\mathbf{t}_H$ with the scalar part being 0. For instance, $\mathbf{t}_X = \begin{bmatrix} 0 & \vec{t}_X^\top \end{bmatrix}^\top$ where the vector $\vec{t}_X$ is the same as t_X which was a notation used earlier but now written with an arrowhead notation to emphasize that it is a 3×1 vector and avoid confusion with scalar quantities in this context. They then formed the normal equations $\Gamma^\top \Gamma \xi = \mathbb{O}$ which gave a least squares solution by finding the eigenvector $v_{\min} = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix}^\top$ corresponding to the minimum eigenvalue of $\Gamma^\top\Gamma$. $v_{\min}$ then had to be normalized using its first four components to give ξ

$$\xi = \frac{1}{\sqrt{v_1^2 + \cdots + v_4^2}} v_{\min}$$

and hence retrieve X. They also provided another closed-form least squares solution that can handle noisy data. They achieved this by dividing $\Gamma^\top\Gamma$ into block matrices via the Schur decomposition and using SVD subsequently.

3. For the hand-eye/robot-world problem, Zhuang et al [41] used quaternions to formulate a sequential type solution and converted the rotation part (3.5) into a linear homogeneous system

$$\underbrace{\begin{bmatrix} a_{0_1}\mathbb{I}_3 + [\vec{a}_1]^\wedge + \frac{1}{a_{0_1}}\vec{a}_1\vec{a}_1^\top & -b_{0_1}\mathbb{I}_3 + \left[\vec{b}_1\right]^\wedge - \frac{1}{a_{0_1}}\vec{a}_1\vec{b}_1^\top \\ \vdots & \vdots \\ a_{0_n}\mathbb{I}_3 + [\vec{a}_n]^\wedge + \frac{1}{a_{0_n}}\vec{a}_n\vec{a}_n^\top & -b_{0_n}\mathbb{I}_3 + \left[\vec{b}_n\right]^\wedge - \frac{1}{a_{0_n}}\vec{a}_n\vec{b}_n^\top \end{bmatrix}}_{\Gamma_{3n\times 6}} \begin{bmatrix} \frac{1}{y_0}\vec{x} \\ \frac{1}{y_0}\vec{y} \end{bmatrix} = \begin{bmatrix} \vec{b}_1 - \frac{b_{0_1}}{a_{0_1}}\vec{a}_1 \\ \vdots \\ \vec{b}_n - \frac{b_{0_n}}{a_{0_n}}\vec{a}_n \end{bmatrix}$$

where $R_{A_i}, R_{B_i}, R_X, R_Y$ were represented as quaternions $\begin{bmatrix} a_{0_i} \\ \vec{a}_i \end{bmatrix}$, $\begin{bmatrix} b_{0_i} \\ \vec{b}_i \end{bmatrix}$, $\begin{bmatrix} x_0 \\ \vec{x} \end{bmatrix}$, $\begin{bmatrix} y_0 \\ \vec{y} \end{bmatrix}$ respectively over $i = 1, ..., n$ measurements. But this conversion was only valid if the rotation angles were not π when R_A, R_Y were converted into the angle-axis representation. They then used SVD and the unit quaternion constraint to solve for $x_0, \vec{x}$ and $y_0, \vec{y}$. To get the translation, they formulated

(3.6) as a linear system

$$\begin{bmatrix} R_{A_1} & -\mathbb{I}_3 \\ \vdots & \vdots \\ R_{A_n} & -\mathbb{I}_3 \end{bmatrix} \begin{bmatrix} t_X \\ t_Y \end{bmatrix} = \begin{bmatrix} R_Y t_{B_1} - t_{A_1} \\ \vdots \\ R_Y t_{B_n} - t_{A_n} \end{bmatrix} \tag{3.13}$$

which they also solved using SVD. The authors also proved that a minimum of 3 measurements are required for a unique solution of X and Y, and the axes of rotation must not be parallel or antiparallel and the relative rotation angles between poses must not be 0 or π.

4. Zhao and Liu [39] obtained the rotation and translation simultaneously for the hand-eye calibration problem (3.1) by forming the linear system

$$\underbrace{\begin{bmatrix} \vec{k}_{A_1} - \vec{k}_{B_1} & [\vec{k}_{A_1} + \vec{k}_{B_1}]^\wedge & \mathbb{O}_{3\times 1} & \mathbb{O}_{3\times 1} \\ \vec{c}_{A_1} - \vec{c}_{B_1} & [\vec{c}_{A_1} + \vec{c}_{B_1}]^\wedge & \mathbb{O}_{3\times 1} & -U_{A_1} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{k}_{A_n} - \vec{k}_{B_n} & [\vec{k}_{A_n} + \vec{k}_{B_n}]^\wedge & \mathbb{O}_{3\times 1} & \mathbb{O}_{3\times 1} \\ \vec{c}_{A_n} - \vec{c}_{B_n} & [\vec{c}_{A_n} + \vec{c}_{B_n}]^\wedge & \mathbb{O}_{3\times n} & -U_{A_n} \end{bmatrix}}_{\Gamma_{6n\times 8}} \underbrace{\begin{bmatrix} \mathbf{q}_X \\ \mathbf{t}_X \star \mathbf{q}_X \end{bmatrix}}_{\xi_{8\times 1}} = \mathbb{O}_{6n\times 1}$$

where $\vec{c}_{A_i}, \vec{c}_{B_i}$ were points on the rotation axes $\vec{k}_{A_i}, \vec{k}_{B_i}$, and U_{A_i} was composed of the components of $\vec{k}_{A_i}$. They then used SVD on Γ to find the two smallest singular values and their corresponding right singular vectors. The value of ξ was then given by a linear combination of these two vectors.

5. Wu et al [36] provided a closed-form solution to the multi-robot/sensor calibration problem (3.7) that solved only for the rotational components of X, Y, Z

simultaneously. They formed the linear system

$$\underbrace{\begin{bmatrix} \overset{-}{\mathbf{Q}}_{B_1}\overset{+}{\mathbf{Q}}_{A_1} & -\mathbf{W}_{C_1} \\ \vdots & \vdots \\ \overset{-}{\mathbf{Q}}_{B_n}\overset{+}{\mathbf{Q}}_{A_n} & -\mathbf{W}_{C_n} \end{bmatrix}}_{\Gamma_{4n\times 20}} \underbrace{\begin{bmatrix} \mathbf{q}_X \\ \mathbf{q}_Y \otimes \mathbf{q}_Z \end{bmatrix}}_{\xi_{20\times 1}} = \mathbb{O}_{4n\times 1}$$

where $\mathbf{W}_{C_i} \in \mathbb{R}^{4\times 16}$ was obtained using $\overset{-}{\mathbf{Q}}_Z\overset{+}{\mathbf{Q}}_Y\mathbf{q}_{C_i} = \mathbf{W}_{C_i}(\mathbf{q}_Y \otimes \mathbf{q}_Z)$. They then solved for ξ using the eigenvector corresponding to the minimum eigenvalue of $\Gamma^\top\Gamma$. But because of the ambiguous transformation between rotation matrices and quaternions, there were two possibilities for Γ. Computing both of them was only feasible when the number of measurements n was small, and hence the algorithm's efficiency decreased exponentially with n and its use was limited to obtaining an initial estimate of X, Y, Z that could be input into their iterative method mentioned in Section 3.2.3.8.

3.2.1.3 Dual Quaternion

1. Daniilidis [9] solved (3.1) by introducing dual quaternions $\check{\mathbf{q}}_{A_i}, \check{\mathbf{q}}_{B_i}, \check{\mathbf{q}}_X$ to represent A_i, B_i, X respectively. As indicated in (2.5), the dual quaternions can be written as a sum of real and dual parts $\check{\mathbf{q}}_{A_i} = \mathbf{a} + \varepsilon\mathbf{a}'$, $\check{\mathbf{q}}_{B_i} = \mathbf{b} + \varepsilon\mathbf{b}'$, $\check{\mathbf{q}}_X =$

$\mathbf{q}_X + \varepsilon\mathbf{q}'_X$. The linear system

$$\underbrace{\begin{bmatrix} \vec{a}_1 - \vec{b}_1 & \left[\vec{a}_1 + \vec{b}_1\right]^\wedge & \mathbb{O}_{3\times1} & \mathbb{O}_{3\times3} \\ \vec{a}'_1 - \vec{b}'_1 & \left[\vec{a}'_1 + \vec{b}'_1\right]^\wedge & \vec{a}_1 - \vec{b}_1 & \left[\vec{a}_1 + \vec{b}_1\right]^\wedge \\ \vdots & \vdots & \vdots & \vdots \\ \vec{a}_n - \vec{b}_n & \left[\vec{a}_n + \vec{b}_n\right]^\wedge & \mathbb{O}_{3\times1} & \mathbb{O}_{3\times3} \\ \vec{a}'_n - \vec{b}'_n & \left[\vec{a}'_n + \vec{b}'_n\right]^\wedge & \vec{a}_n - \vec{b}_n & \left[\vec{a}_n + \vec{b}_n\right]^\wedge \end{bmatrix}}_{\Gamma_{6n\times8}} \begin{bmatrix} \mathbf{q}_X \\ \mathbf{q}'_X \end{bmatrix} = \mathbb{O}_{6n\times1}. \tag{3.14}$$

was solved using SVD to get the rotation and translation of X simultaneously. R_X can be obtained from $\mathbf{q}_X$ while t_X is contained in $\mathbf{q}'_X := \frac{1}{2}\mathbf{t}_X \star \mathbf{q}_X$. In this formulation, the scalar parts of the quaternions were not used.

2. Li et al [18] applied dual quaternions to the hand-eye/robot-world calibration problem (3.4) to solve for the rotation and translation simultaneously. They used the matrix representations in (2.4) for the real and dual parts of $\check{\mathbf{q}}_{A_i}, \check{\mathbf{q}}_{B_i}$ and this is denoted by the non-primed and primed uppercase letters. Their linear system was formulated as

$$\underbrace{\begin{bmatrix} \overset{+}{\mathbf{A}}_1 & \mathbb{O}_{4\times4} & \overset{-}{\mathbf{B}}_1 & \mathbb{O}_{4\times4} \\ \overset{+}{\mathbf{A}}'_1 & \overset{+}{\mathbf{A}}_1 & \overset{-}{\mathbf{B}}'_1 & \overset{-}{\mathbf{B}}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \overset{+}{\mathbf{A}}_n & \mathbb{O}_{4\times4} & \overset{-}{\mathbf{B}}_n & \mathbb{O}_{4\times4} \\ \overset{+}{\mathbf{A}}'_n & \overset{+}{\mathbf{A}}_n & \overset{-}{\mathbf{B}}'_n & \overset{-}{\mathbf{B}}_n \end{bmatrix}}_{\Gamma_{8n\times16}} \begin{bmatrix} \mathbf{q}_X \\ \mathbf{q}'_X \\ \mathbf{q}_Y \\ \mathbf{q}'_Y \end{bmatrix} = \mathbb{O}_{8n\times1}$$

which was solved using SVD. The solution for $\mathbf{q}_X, \mathbf{q}_Y$ gave the rotations R_X, R_Y while the translations t_X, t_Y were recovered using $\mathbf{q}'_X = \frac{1}{2}\mathbf{t}_X \star \mathbf{q}_X$ and $\mathbf{q}'_Y =$

$\frac{1}{2}\mathbf{t}_Y \star \mathbf{q}_Y$.

3.2.1.4 Lie Group

1. Park and Martin [26] used Lie groups to form linear systems from (3.2) and (3.3) which were then solved using the standard least squares method for the rotation and translation sequentially. First they obtained the rotation using

$$R_X = \left(M^\top M\right)^{-\frac{1}{2}} M^\top$$

where M is defined using the rotation axes of A and B

$$M = \sum_{i=1}^{n} k_{B_i} k_{A_i}^\top$$

Then the translation t_X was computed using the rotation by solving the linear equation (3.11) using the standard least squares method.

3.2.1.5 Kronecker Product

1. Andreff et al [5, 6] used the Kronecker product to formulate (3.1) as a linear system that solved for the rotation and translation simultaneously:

$$\begin{bmatrix} \mathbb{I}_9 - R_{A_1} \otimes R_{B_1} & \mathbb{O}_{9\times 3} \\ \mathbb{I}_3 \otimes t_{B_1}^\top & \mathbb{I}_3 - R_{A_1} \\ \vdots & \vdots \\ \mathbb{I}_9 - R_{A_n} \otimes R_{B_n} & \mathbb{O}_{9\times 3} \\ \mathbb{I}_3 \otimes t_{B_n}^\top & \mathbb{I}_3 - R_{A_n} \end{bmatrix} \begin{bmatrix} \mathrm{vec}(R_X) \\ t_X \end{bmatrix} = \begin{bmatrix} \mathbb{O}_{9\times 1} \\ t_{A_1} \\ \vdots \\ \mathbb{O}_{9\times 1} \\ t_{A_n} \end{bmatrix} \tag{3.15}$$

They also provided algebraic analysis on what can be recovered depending on the type of motions of the camera. They also found that in the presence of noise, the solution obtained for R_X may not be orthogonal. Hence the problem had to be recast as a sequential one where the rotation was solved first using:

$$\begin{bmatrix} \mathbb{I}_9 - R_{A_1} \otimes R_{B_1} \\ \vdots \\ \mathbb{I}_9 - R_{A_n} \otimes R_{B_n} \end{bmatrix} \operatorname{vec}(R_X) = \mathbb{O}_{9n\times 1}$$

and then orthogonalizing the resulting R_X before using it to obtain the translation by solving (3.11).

2. Liang and Mao [20] also used the Kronecker product to obtain a linear system that solved the rotation first:

$$\underbrace{\begin{bmatrix} R_{A_1} \otimes \mathbb{I}_3 - \mathbb{I}_3 \otimes R_{B_1}^\top \\ \vdots \\ R_{A_n} \otimes \mathbb{I}_3 - \mathbb{I}_3 \otimes R_{B_n}^\top \end{bmatrix}}_{\Gamma} \operatorname{vec}(R_X) = \mathbb{O}_{9n\times 1} \tag{3.16}$$

by computing the SVD of Γ, which say was $U_\Gamma \Sigma_\Gamma V_\Gamma^\top$. Then the solution to $\operatorname{vec}(R_X)$ is given by the columns of V_Γ. But R_X had to be orthogonalized by computing its SVD $\left(U_X \Sigma_X V_X^\top\right)$ to obtain the nearest orthogonal matrix $U_X V_X^\top$. The translation t_X was found using (3.11) and then applying QR factorization.

3. Ackerman et al [4] solved the hand-eye calibration problem (3.1) for the case where a priori correspondence between the measurements A_i, B_i were not given. They first recovered the correspondence using four invariants of $SE(3)$ under conjugation. Then they formed the linear system (3.15) to solve for the rota-

tion and translation simultaneously. The obtained rotation R_X might not be orthogonal so there was a need to project it onto $SO(3)$.

4. Li et al [18] also provided a second solution to the hand-eye/robot-world calibration problem (3.4) but solved the rotation and translation simultaneously (in contrast to Shah's method — see item 5). Their linear system to be solved was

$$\begin{bmatrix} R_{A_1} \otimes \mathbb{I}_3 & -\mathbb{I}_3 \otimes R_{B_1}^\top & \mathbb{O}_{9\times3} & \mathbb{O}_{9\times3} \\ \mathbb{O}_{3\times9} & \mathbb{I}_3 \otimes t_{B_1}^\top & -R_{A_1} & \mathbb{I}_3 \\ \vdots & \vdots & \vdots & \vdots \\ R_{A_n} \otimes \mathbb{I}_3 & -\mathbb{I}_3 \otimes R_{B_n}^\top & \mathbb{O}_{9\times3} & \mathbb{O}_{9\times3} \\ \mathbb{O}_{3\times9} & \mathbb{I}_3 \otimes t_{B_n}^\top & -R_{A_n} & \mathbb{I}_3 \end{bmatrix} \begin{bmatrix} \mathrm{vec}(R_X) \\ \mathrm{vec}(R_Y) \\ t_X \\ t_Y \end{bmatrix} = \begin{bmatrix} \mathbb{O}_{9\times1} \\ t_{A_1} \\ \vdots \\ \mathbb{O}_{9\times1} \\ t_{A_n} \end{bmatrix}$$

and in the presence of noise, they used Rodrigues' rotation formula to make R_X, R_Y orthogonal.

5. Shah [28] solved (3.4) sequentially using the Kronecker product by transforming (3.5) into

$$\underbrace{\begin{bmatrix} R_{B_1} \otimes R_{A_1} & -\mathbb{I}_9 \\ \vdots & \vdots \\ R_{B_n} \otimes R_{A_n} & -\mathbb{I}_9 \end{bmatrix}}_{\Gamma_{9n\times18}} \begin{bmatrix} \mathrm{vec}(R_X) \\ \mathrm{vec}(R_Y) \end{bmatrix} = \mathbb{O}_{9n\times1}$$

and computing $\Gamma^\top\Gamma$ to give

$$\begin{bmatrix} n\mathbb{I}_9 & -\sum_{i=1}^{n} R_{B_i}^\top \otimes R_{A_i}^\top \\ \underbrace{-\sum_{i=1}^{n} R_{B_i} \otimes R_{A_i}}_{K} & n\mathbb{I}_9 \end{bmatrix}.$$

vec(R_X) and vec(R_Y) were then proportional to the right and left singular vector of K respectively that correspond to the singular value n. The proportionality constants are determined by enforcing R_X, R_Y to have a determinant of $+1$. Because of noise, the computed matrices had to be re-orthogonalized so that they remained in $SO(3)$. To get the translations t_X, t_Y, she used the same method as Zhuang et al and solved the linear system (3.13).

6. Ernst et al [11] used the Kronecker product to solve (3.4) although they did not explicitly mention that term. They proposed a simultaneous solution that formed the linear system:

$$\underbrace{\begin{bmatrix} R_{B_1}^\top \otimes R_{A_1} & \mathbb{O}_{9\times 3} & \multirow{2}{*}{$-\mathbb{I}_{12}$} \\ t_{B_1}^\top \otimes R_{A_1} & R_{A_1} & \\ \vdots & & \vdots \\ R_{B_n}^\top \otimes R_{A_n} & \mathbb{O}_{9\times 3} & -\mathbb{I}_{12} \\ t_{B_n}^\top \otimes R_{A_n} & R_{A_n} & \end{bmatrix}}_{\Gamma_{12n\times 24}} \underbrace{\begin{bmatrix} \text{vec}(R_X) \\ t_X \\ \text{vec}(R_Y) \\ t_Y \end{bmatrix}}_{\xi_{24\times 1}} = \underbrace{\begin{bmatrix} \mathbb{O}_{9\times 1} \\ -t_{A_1} \\ \vdots \\ \mathbb{O}_{9\times 1} \\ -t_{A_n} \end{bmatrix}}_{\gamma_{12n\times 1}}$$

and solved it using QR factorization. Since the computed R_X, R_Y were not orthogonal, they were decomposed using SVD. For instance if R_X was decomposed into $U\Sigma V^\top$, then the "closest" orthogonal matrix would be

$$R_X = UV^\top.$$

7. Yan et al [37] solved the calibration problem (3.7) for a hybrid robot which consists of a parallel manipulator mounted on the tool flange of a serial manipulator.

See Figure 4.3 for an illustration, bearing in mind that the transformations may represent different relationships when compared to [34]. This robot system will be described further in Section 4.1.3 and hence we only state their methods here. Their first method (called the "Degradation-Kronecker" method) split the problem into two $AX = YB$ subproblems by making B or C constant and only varying the other two matrices to collect measurements. That is, by fixing B, (3.7) can be reduced to

$$A\widetilde{X} = YC \tag{3.17}$$

where

$$\widetilde{X} = XBZ^{-1} \tag{3.18}$$

Similarly by fixing C, (3.7) can be reduced to

$$AX = \widetilde{Y}B^{-1} \tag{3.19}$$

where

$$\widetilde{Y} = YCZ \tag{3.20}$$

(3.17) and (3.19) were then solved using the Kronecker product method in [18] to obtain X, Y, $\widetilde{X}$ and $\widetilde{Y}$. They then used SVD to enforce the orthogonality of R_X and R_Y and finally obtained Z by solving (3.18) and (3.20) and choosing the Z with smaller errors. This method only applied if fixing B was feasible.

This method gave a closed-form solution which was fast but was not as accurate in the presence of noise. See A.3 for the MATLAB code.

3.2.2 Transform to Nonlinear System

3.2.2.1 Constrained Optimization

1. Horaud and Dornaika [16] presented one hand-eye calibration method to (3.1) using quaternions, and it was a sequential method that found the rotation by solving a contrained minimization problem

$$\min_{\mathbf{q}} \left\{ \mathbf{q}^\top \left[\sum_{i=1}^{n} \left(\overset{+}{\mathbf{V}}_{A_i} - \overset{-}{\mathbf{V}}_{B_i} \right)^\top \left(\overset{+}{\mathbf{V}}_{A_i} - \overset{-}{\mathbf{V}}_{B_i} \right) \right] \mathbf{q} \right\}, \quad \text{such that} \quad \mathbf{q}^\top \mathbf{q} = 1$$

using Lagrange multipliers. If the eigenvectors of R_{A_i}, R_{B_i} corresponding to the eigenvalue 1 are $\vec{v}_{A_i}, \vec{v}_{B_i}$ respectively, then $\overset{+}{\mathbf{V}}_{A_i}, \overset{-}{\mathbf{V}}_{B_i}$ are the matrices corresponding to the quaternions $\mathbf{q}_{A_i} = \begin{bmatrix} 0 & \vec{v}_{A_i}^\top \end{bmatrix}^\top$ and $\mathbf{q}_{B_i} = \begin{bmatrix} 0 & \vec{v}_{B_i}^\top \end{bmatrix}^\top$ respectively. They then obtained the translation t_X by minimizing

$$\min_{R,t} \sum_{i=1}^{n} \| R_X t_{B_i} - (R_{A_i} - \mathbb{I}) t_X - t_{A_i} \|^2,$$

which is just solving a linear least squares problem.

2. Dornaika and Horaud [10] presented a similar Lagrange multiplier method to hand-eye/robot-world calibration (3.4) that formulated the rotation portion (3.5) as a constrained optimization problem that could be solved using Lagrange

multipliers in closed-form. The objective function is a positive semi-definite quadratic form

$$\min_{\mathbf{q}_X,\mathbf{q}_Y} (\mathbf{q}_X \star \mathbf{q}_Y)^\top S(\mathbf{q}_X \star \mathbf{q}_Y), \quad \text{such that} \quad \mathbf{q}_X^\top \mathbf{q}_X = 1,\ \mathbf{q}_Y^\top \mathbf{q}_Y = 1.$$

where S is a 8×8 positive semi-definite symmetric matrix

$$S = \begin{bmatrix} n\mathbb{I}_4 & \sum_{i=1}^{n} -\overset{+}{\mathbf{Q}}{}_{A_i}^\top \overset{-}{\mathbf{Q}}_{B_i} \\ \sum_{i=1}^{n} -\overset{-}{\mathbf{Q}}{}_{B_i}^\top \overset{+}{\mathbf{Q}}_{A_i} & n\mathbb{I}_4 \end{bmatrix}.$$

To get the translation t_X, t_Y, they solved (3.13) using linear least squares.

3.2.3 Iterative Methods

An iterative method usually solves the problem by approximating the solution using optimization techiniques.

3.2.3.1 Nonlinear least squares

1. Zhuang and Shiu [42] presented an iterative algorithm that solved for the rotation and translation simultaneously for hand-eye calibration. They first defined

$$Z_i := A_i X - X B_i, \qquad i = 1, ..., n$$

where n is the number of measurements, and then they solve the problem

$$\arg\min_X \sum_{i=1}^{n} \text{vec}(Z_i)^\top \text{vec}(Z_i).$$

This nonlinear least squares problem could then be solved using the Gauss-Newton or Levenberg-Marquardt methods. Their algorithm could also handle the case when the orientation of the sensor is not known. It was also more accurate in most cases except when the position data is much noisier than the orientation data.

2. Horaud and Dornaika [16] presented another technique to solve (3.1), in addition to the one in Section 3.2.2. This method solved for the rotation R_X and translation t_X simultaneously by forming a nonlinear sum of squares optimization problem

$$\operatorname*{arg\,min}_{\mathbf{q},t} \left\{ \sum_{i=1}^{n} \left\| \mathbf{v}_{A_i} - \mathbf{q} \star \mathbf{v}_{B_i} \star \bar{\mathbf{q}} \right\|^2 + \sum_{i=1}^{n} \left\| \Im \left(\mathbf{q} \star \mathbf{t}_{B_i} \star \bar{\mathbf{q}} \right) - \left(R_{A_i} - \mathbb{I} \right) t - t_{A_i} \right\|^2 + \lambda \left(1 - \mathbf{q}^\top \mathbf{q} \right)^2 \right\}$$

where λ is a Lagrange multiplier to enforce unit quaternions and $\mathbf{v}_{A_i}, \mathbf{v}_{B_i}$ are the quaternions (with real part zero) corresponding to the eigenvectors of R_{A_i}, R_{B_i} that are associated with the unit eigenvalue. The term $\Im(\cdot)$ refers to the imaginary or vector part of the quaternion. This problem could then be solved using nonlinear least squares methods like Levenberg-Marquardt or be simplified further to be amenable to constrained step methods like trust-region.

3. Dornaika and Horaud [10] proposed a second solution to the hand-eye/robot-

world calibration problem (3.4) using a nonlinear least-squares constrained minimization approach. They minimzed the error function

$$\begin{aligned}\min_{x} \quad & \left\{ \sum_{i=1}^{n} \|R_{A_i}R_X - R_{B_i}R_Y\|^2 \right.\\ & + \sum_{i=1}^{n} \|R_{A_i}t_X + t_{A_i} - R_Y t_{B_i} - t_Y\|^2 \\ & + \lambda_1 \left\|R_X^\top R_X - \mathbb{I}\right\|^2 \\ & \left. + \lambda_2 \left\|R_Y^\top R_Y - \mathbb{I}\right\|^2 \right\}\end{aligned}$$

over x where $x \in \mathbb{R}^{24}$ consisted of the elements in the rotation matrices and translation vectors R_X, t_X, R_Y, t_Y. The solution to this problem was obtained using the Levenberg-Marquardt algorithm.

4. Fassi and Legani [12] gave a geometric interpretation for (3.1) using screw theory. They provided a closed-form algorithm to determine the unique solution using two sets of measurements that were not degenerate. For a set of n measurements, they formed the minimization problem

$$\min_{\xi} \sum_{i=1}^{n} \|A_i X(\xi) - X(\xi) B_i\|$$

where ξ consists of the rotation axis, rotation angle, point on the rotation axis and amount of translation along the axis. Two measurements were chosen to compute an initial value of X using the closed-form algorithm that that was then passed into an iterative optimization method to obtain the solution.

5. The first solution in Ha et al [13] used geometric optimization to solve the hand-eye/robot-world calibration problem (3.4). They formed the optimization problem

$$\min_{R_X, R_Y} \frac{1}{2}\sum_{i=1}^{18} \lambda_i \left[\operatorname{tr}(P_i R_X) + \operatorname{tr}(Q_i R_Y)\right]^2 + \operatorname{tr}(P_0 R_X) + \operatorname{tr}(Q_0 R_Y) + c \tag{3.21}$$

where $P_i, Q_i \in \mathbb{R}^{3\times 3}, i = 0, \ldots, 18$ and $\lambda_i, c \in \mathbb{R}$ were computed from the eigenvalue analysis of the objective function. They also showed how to compute the initial estimate and the step size for gradient descent and Newton's method to find the optimal R_X, R_Y.

6. The second solution proposed by Yan et al [37] to solve (3.7) (called the "purely nonlinear" method) solved for X, Y, Z simultaneously using nonlinear minimization of the error in rotation and translation. That is, they formed:

$$\arg\min_{X,Y,Z} \sum_{j=1}^{12} f_j^2$$

where f_j is an element of the vector $f \in \mathbb{R}^1 2$

$$f = \begin{bmatrix} \operatorname{vec}\left(R_A R_X R_B - R_Y R_C R_Z\right) \\ R_A R_X t_B + R_A t_X + t_A - (R_Y R_C t_Z + R_Y t_C + t_Y) \end{bmatrix}$$

which was obtained by manipulating (3.8) and (3.9). This nonlinear least squares problem was then solved using the Levenberg-Marquardt algorithm, and used random X, Y, Z as initial estimates. The method was sensitive to the initial estimates and hence it might need multiple executions to find the optimal solution. In my implementation (see A.4) with random initial estimates, it typically needed about 5 executions to get back the original X, Y, Z in the no noise case.

3.2.3.2 Convex Optimization

1. Zhao [38] used convex optimization to solve (3.1) without the need for an initial value. He formulated the problem by representing rotations as orthonormal matrices and quaternions and applying the L_∞ norm. For orthonormal matrices by representing the one matrix and two vectors in the equation (3.15) as $Cx = d$, he transformed it into the equivalent problem

$$\min_{\delta,x} \delta \quad \text{such that} \quad \|C_i x - d_i\|_2 \leqslant \delta \qquad \text{for } i = 1, ..., n$$

which could be solved using second-order cone programming (SOCP).

For dual quaternions, Zhao represented the one matrix and one vector in (3.14) as $C_i x = 0$ and hence converted it into the following SOCP optimization problem

$$\min_{\delta,x} \delta \quad \text{such that} \quad \begin{array}{l} \|C_i x\|_2 \leqslant \delta \qquad \text{for } i = 1, ..., n \quad \text{with} \\ Dx \geqslant f \end{array}$$

where $Dx \geqslant f$ represents the constraint to avoid the trivial solution $x = 0$.

3.2.3.3 Global Optimization

1. Heller et al [14] proposed three parametrizations to (3.1) and (3.4) that used polynomial optimization over semi-algebraic sets with linear matrix inequality (LMI) relaxations. The first two used the orthonormal and quaternion parameterization for rotations and (3.1) became

   $$\min_{\xi} \sum_{i=1}^{n} \|A_i X(\xi) - X(\xi) B_i\|^2, \quad \text{such that} \quad g(\xi) \geqslant 0$$

   where the variables ξ to be minimized over and the constraints $g(\xi)$ differed depending on the orthonormal or quaternion case. The norm used here is the Frobenius norm. The third parameterization used dual quaternions to form

   $$\min_{\check{\mathbf{q}}} \sum_{i=1}^{n} \|\check{\mathbf{a}}_i \star \check{\mathbf{q}}_X - \check{\mathbf{q}}_X \star \check{\mathbf{b}}_i\|^2, \quad \text{such that} \quad g(\check{\mathbf{q}}) \geqslant 0.$$

   All these polynomial objective functions were then relaxed using LMIs and solved via semidefinite programming (SDP). Similar formulations and solutions were stated for the hand-eye/robot-world problem (3.4).

2. Ruland et al [27] formulated (3.1) as a nonconvex global optimization problem that separates the estimation of the rotation and translation. They represented rotations using angle-axis and then applied the branch-and-bound algorithm to solve the optimization problem.

3.2.3.4 Stochastic Optimization

1. For the hand-eye/robot-world calibration problem (3.4), Ha et al [13] also proposed a two-phase stochastic optimization algorithm for the objective function (3.21). This involved uniform random sampling on $SO(3)$, applying local search on those samples using their first algorithm (refer to Section 3.2.3.1 item 5) and then checking optimal Bayesian stopping rules.

3.2.3.5 Jacobian Optimization

1. Mao et al [25] proposed solving (3.1) for the rotation and translation simultaneously using Jacobians of the objective function. They formed the problem

$$\min_X \sum_{i=1}^{n} \underbrace{\left[\|F_i\|^2 + \|G_i\|^2\right]}_{H} \tag{3.22}$$

where F_i came from (3.16) and G_i came from (3.3):

$$F_i = \left(R_{A_i} \otimes \mathbb{I}_3 - \mathbb{I}_3 \otimes R_{B_i}^\top\right) \operatorname{vec}(R_X)$$

$$G_i = \left(R_{A_i} - \mathbb{I}_3\right) t_X - R_X t_{B_i} + t_{A_i}$$

They required initial estimates which could be computed by their earlier method [20] that was a closed-form solution using the Kronecker product (see item 2 in section 3.2.1). The Jacobian J for (3.22) was computed to be

$$J_i = \begin{bmatrix} \frac{dF_i}{d\vec{r}^{\top}} & \frac{dF_i}{d\vec{t}^{\top}} \\ \frac{dG_i}{d\vec{r}^{\top}} & \frac{dG_i}{d\vec{t}^{\top}} \end{bmatrix}$$

where $\vec{r}$ was the vector of Euler angles for R_X and $\vec{t} := t_X$. This was used to compute the update step $\Delta\chi$ for the iterative algorithm in this way:

$$J\Delta\chi = -H.$$

3.2.3.6 Quaternion

1. Hirsh et al [15] proposed an iterative algorithm to the hand-eye/robot-world problem (3.4) by averaging quaternions and vectors. From some initial estimate of Y, they used three measurements each of R_A, R_B to compute three estimates of R_X using (3.5). These estimates were then converted to quaternions and averaged to get the "best" estimate of R_X. This average was then used to compute 3 estimates of R_Y, which was again averaged after conversion to quaternions. This cycle then continued until the estimates of R_X, R_Y converged to a specified tolerance. With these values, the translation was then computed using the same idea: update the estimates t_X, t_Y using (3.4) and carry out the averaging on only the translation vector part of the homogeneous matrix.

3.2.3.7 Lie Group

1. Ackerman et al [3] solved (3.1) by formulating the optimization problem

$$\min_{X} \ \|AX - XB\|_W^2$$

where the weighted Frobenius norm was used. To solve this problem, they applied gradient descent on the Euclidean group $SE(3)$ using the update step

$$g_{s+1} \approx g_s e^{\Delta t\, v_g}$$

over a small time step Δt where $g \in SE(3)$ and $v_g = g^{-1}\dot{g}$ is the rigid body velocity. They also provided four conditions that could filter out those A_i, B_i measurements which were too noisy.

3.2.3.8 Linear Approximation

1. For the multi-robot/sensor calibration problem (3.7), Wang et al [34] solved for X, Y, Z simultaneously but the rotational and translational components were handled sequentially. Hence this would be a sequential method and their algorithm required at least 3 sets of data. They solved approximately for the rotational components using a linear iterative method by applying Taylor's expansion of the exponential map to form a linear system. This system can then be solved for the change in rotation using the normal equations. The translational components were obtained by a linear least squares method by rewriting

(3.9) and stacking n measurements to obtain

$$\underbrace{\begin{bmatrix} R_{A_1} & -\mathbb{I} & -R_Y R_{C_1} \\ \vdots & \vdots & \vdots \\ R_{A_n} & -\mathbb{I} & -R_Y R_{C_n} \end{bmatrix}}_{J_{3n\times 9}} \begin{bmatrix} t_X \\ t_Y \\ t_Z \end{bmatrix} = \underbrace{\begin{bmatrix} R_Y t_{C_1} - t_{A_1} - R_{A_1} R_X t_{B_1} \\ \vdots \\ R_Y t_{C_n} - t_{A_n} - R_{A_n} R_X t_{B_n} \end{bmatrix}}_{p_{3n\times 1}}$$

which could be solved using standard linear least squares. In a subsequent journal paper, Wu et al [36] proposed a closed-form solution (reviewed in Section 3.2.1.2 item 5) to obtain an initial estimate of X, Y, Z that could be input into their iterative method. Refer to Listing A.2 for the MATLAB code.

3.2.3.9 Statistical Model

1. Strobl and Hirzinger [31] proposed a new metric for the error in rotation and translation that was used to form an optimization problem for hand-eye (3.1) or hand-eye/robot-world (3.4) calibration. Their formulation used Gaussian distributions for the rotation and translation error, and could automatically compute the optimal weights for the rotation and translation components to improve accuracy. Numerical optimization algorithms could then be applied to obtain the rotation and translation simultaneously.

2. Ackerman et al [1] provided two information theoretic approaches to the hand-eye calibration problem (3.1) by viewing the A_i, B_i, X as probability density functions on $SE(3)$. They formed constraints on X so that it was parametrized

by two parameters

$$X(\phi, s) = H\left((R(k_A, k_B)R(k_B, \phi), t(s)\right)$$

where $(\phi, s) \in [0, 2\pi) \times \mathbb{R}$. Their two methods solved the respective minimization problems

$$\min_{\phi,s} \left\| \mathrm{Ad}\left(X(\phi, s)^{-1}\right) \Sigma_A - \Sigma_B \,\mathrm{Ad}^\top(X(\phi, s)) \right\|_F^2,$$

$$\min_{\phi,s} \; \mathrm{tr}\left\{\Sigma_A^{-1} \mathrm{Ad}(X(\phi, s))\, \Sigma_B \,\mathrm{Ad}^\top(X(\phi, s))\right\}$$

using a closed form expression by noticing that the cost function was quadratic in s. The solution for s was then substituted back to solve for ϕ which was just a one dimensional search.

3.2.4 Batch Methods

3.2.4.1 Probabilistic

This section introduces a method that have been applied to all three calibration problems in Section 3.1. The strength of this method is that the measurements $\{A_i, B_i\}$ or $\{A_i, B_i, C_i\}$ (depending on the problem) did not need a priori correspondence, unlike other methods. The loss of correspondence could be because the measurements were not taken synchronously so there was a temporal shift between the data. Another reason could be that the data within each pair or triplet was shifted by a different value from the other pairs or triplets, which produced "scrambled data". The main theory for such probabilistic batch methods can be found in [7, 35].

1. Ackerman and Chirikjian [2] first applied the properties of probability distributions on $SE(3)$ (covered in Section 2.3) to solve the hand-eye calibration problem (3.1). Since the data $H \in \{A_i, B_i\}$ were discrete, they defined the discrete version of the mean $\bar{H}$ and covariance Σ_H for a p.d.f. f_H, where the continuous version was defined in (2.6) and (2.7) respectively. Hence

$$\sum_{i=1}^{n} \log\left(\bar{H}^{-1} H_i\right) := \mathbb{O}_4 \tag{3.23}$$

defined the mean, and the corresponding covariance was

$$\Sigma_H := \sum_{i=1}^{n} \log^{\vee}\left(\bar{H}^{-1} H_i\right)\left[\log^{\vee}\left(\bar{H}^{-1} H_i\right)^{\top}\right].$$

Hence (3.1) produced

$$\bar{A}X = X\bar{B} \tag{3.24}$$

$$\mathrm{Ad}\left(X^{-1}\right)\Sigma_A\,\mathrm{Ad}^{\top}\left(X^{-1}\right) = \Sigma_B \tag{3.25}$$

where $\bar{A}, \bar{B}$ were the mean of $\{A_i\}, \{B_i\}$. This produced a set of 4 possibilities for R_X and the correct one could be determined by solving

$$\arg\min_{X} \|k_{\bar{A}} - R_X k_{\bar{B}}\|$$

where k_H was the screw axis of H. After getting R_X, the translation t_X was then obtained using (3.25).

2. Li et al [19] used a similar probabilistic method to perform batch optimization for the hand-eye/robot-world problem (3.4). The main advantage of their method was the ability to recover X and Y despite a constant shift in time between the corresponding $\{A_i\}, \{B_i\}$ data. They formed the following covariance relationships out of the first two blocks of (3.25):

$$\Sigma_B^{(1)} = R_X^\top \Sigma_A^{(1)} R_X \tag{3.26}$$

$$\Sigma_B^{(2)} = R_X^\top \Sigma_A^{(1)} R_X \left[R_X^\top t_X\right]^\wedge + R_X^\top \Sigma_A^{(2)} R_X. \tag{3.27}$$

From (3.24), (3.26) and (3.27), they obtained 8 candidates pairs for $(X_k, Y_k),\ k = 1, \ldots, 8$. To pick an optimal pair, the obtained the time shift (to recover the correspondence) by solving an optimization problem using the correlation function between $\{A_i\}$ and $\{B_j\}$. With the correspondence between the measurements $\{A_i, B_i\}$ restored, the optimal pair could now be determined by solving

$$(X, Y) = \underset{X_k, Y_k}{\arg\min}\, \frac{1}{n} \sum_{i=1}^{n} \left[\|\theta_{A_i} - \theta_{\widetilde{B}_i}\| + \|d_{A_i} - d_{\widetilde{B}_i}\|\right] \tag{3.28}$$

where $\widetilde{B}_i = X_k^{-1} Y_k B_i$, and θ, d are screw parameters. Then an optimal X can be obtained from (3.28) using Euclidean group invariants for the $AX = XB$ case and subsequently obtain the optimal Y.

3. Ma et al [24] adopted a similar method to Ackerman in [2] for (3.1) but used a different definition of the mean and covariance on $SE(3)$ which improved the

accuracy of the recovered X provided the distributions of A_i, B_i satisfied some conditions. They used the first-order and second-order approximations of (3.23) to derive two new means for their batch methods which they call "Batch1" and "Batch2". In closed-form, the mean of A_i for Batch1 is

$$\bar{A}_{\text{Batch1}} = \begin{bmatrix} R_{\bar{A}} & \frac{1}{n}\sum_{i=1}^{n} t_{A_i} \\ \mathbb{O}^{\top} & 1 \end{bmatrix}$$

with a similar form for B_i. The mean for Batch2 could not be written in closed-form as it was an iterative solution to an optimization problem. This method works on scrambled data and hence did not need to recover the correspondence between pairs of A_i and B_i.

4. Ma et al [23] applied a similar batch method to the multi-robot/sensor calibration problem formulated using (3.7). This method is reviewed in detail in Chapter 4.

Chapter 4

Review of the MGC method

This chapter contains a detailed review of the MGC method [23] for solving the multi-robot/sensor calibration problem that has been formulated as the matrix equation

$$A_i X B_i = Y C_i Z, \qquad i = 1, \ldots, n. \tag{4.1}$$

where we are given $\{A_i, B_i, C_i\}$ to solve for the unknowns X, Y, Z.

This method can solve (4.1) under two types of formulations, which Ma et al called "frameworks". The first framework solves for X, Y, Z simultaneously under certain conditions and is a sequential type method, i.e. it solves for R_X, R_Y, R_Z first and then uses that to obtain t_X, t_Y, t_Z. The second framework solves for X and Z simultaneously and uses them to solve for Y next, and is also a sequential type method.

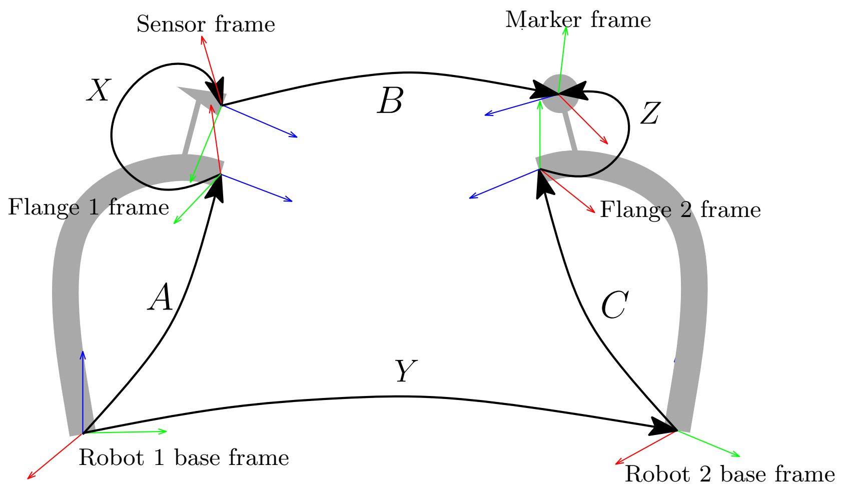

Figure 4.1: Dual-arm robot-sensor calibration formulated as $AXB = YCZ$.

4.1 Applications to Robotic Systems

The MGC method can be applied to calibrate multiple robot systems that can be represented by (4.1). Here are three such systems:

4.1.1 Mobile robots

Figure 3.3 shows three mobile robots with cameras and targets mounted on them. For each robot, we are interested to find out the relative position and orientation of the target and camera, which can be represented by homogeneous matrices X, Y, Z. When each robot points its camera at the target of another robot in a chain as shown in Figure 3.3, the position and orientation of a target relative to the camera that is looking at it can be represented by the matrices A, B, C. Hence we can formulate the calibration problem for this robotic system using (4.1).

4.1.2 Dual serial manipulator

Another applicable system has two robot manipulators fixed to the ground as shown in Figure 4.1. One robot has a sensor mounted on it and the other robot has a marker mounted on it. For one robot, we want to calculate the flange to sensor transformation, X, while for the other robot, we want to compute the flange to marker transformation, Z. The remaining unknown is the base-to-base transformation, Y. The data that can be collected are the base-to-flange transformation for each robot, A, C, which can be computed using the forward kinematics. The relative transformation, B, between the camera and target is collected as well, but the robots have to move in such a way that at every robot pose, the target is visible in the camera's field-of-view.

A variation of this system is to have two manipulators, each with a sensor mounted on it, looking at a common target (see Figure 4.2 for the diagram where the target is a checkerboard pattern). Here the data $\{B_i\}$ is obtained indirectly as $B = B_1 B_2^{-1}$, using the transformations in each sensor frame B_1, B_2. A naive approach to solve (3.7) is to do two hand-eye calibrations for each robot separately and then calibrate the base-to-base transformation Y. But this three step method has error propagation so Y will be more inaccurate compared to X or Z.

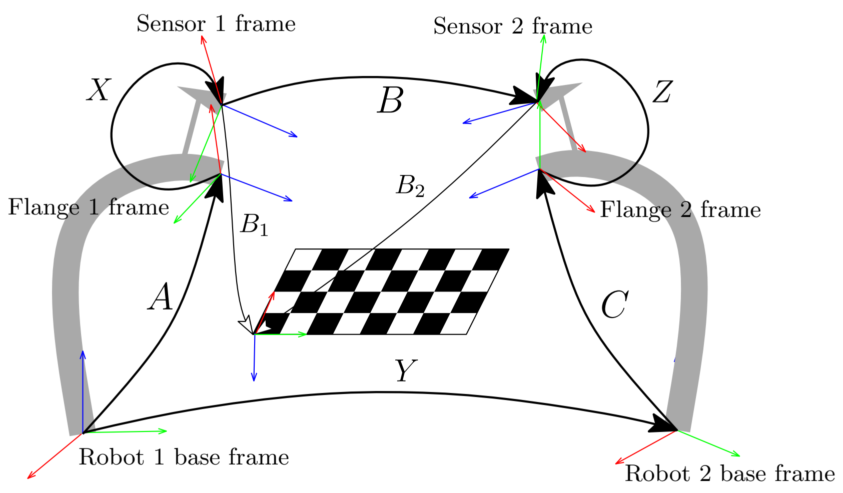

Figure 4.2: System of two manipulators with cameras looking at a common target

4.1.3 Hybrid (serial-parallel) robot

Yan et al [37] introduced a hybrid robot system that can be calibrated by modeling the system using (4.1). As seen in Figure 4.3, this hybrid robot consists of a parallel manipulator mounted on the flange of a serial manipulator robot, and there is a stationary camera looking at a marker on the tool of the parallel robot. The unknowns in this system are the transformation between the flange of the serial robot and base of the parallel robot, X, the transformation between the base of the serial robot and the camera frame, Y, and the transformation between the base of the marker and the flange of the parallel robot, Z. We can compute the serial robot's and the parallel robot's base-flange transformation $(A, B$ respectively) using their forward kinematics. The camera captures images of the marker and provides the camera-tool transformation, C. When calibrated, the camera can track the position of the marker which can in turn guide the movements of the robot.

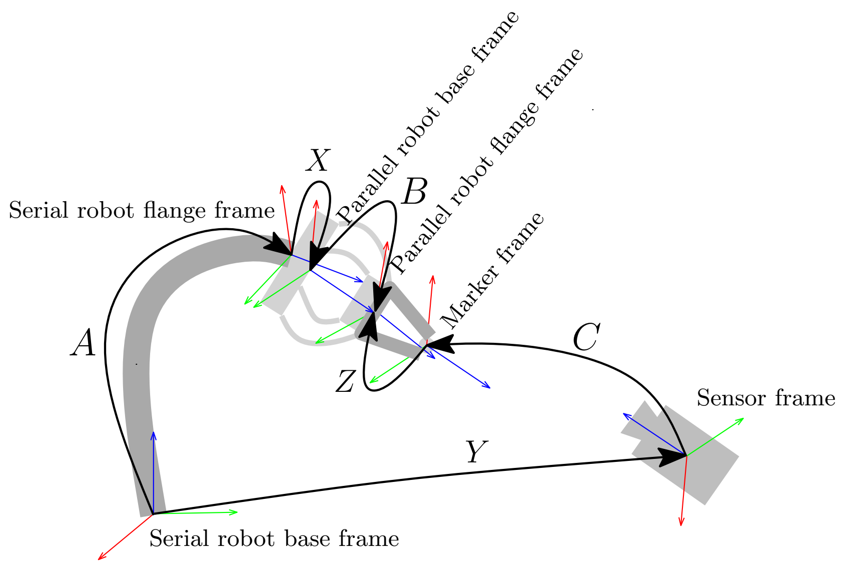
Parallel robot base frame
Parallel robot flange frame
Marker frame
Serial robot flange frame
X
B
A
C
Z
Sensor frame
Y
Serial robot base frame

Figure 4.3: Serial-parallel arm hybrid robot calibration formulated as $AXB = YCZ$

4.2 Mathematical Framework

In order for the MGC method to work, the initial step is to collect data in such a way where the user can fix any one of A, B, or C while varying the other two transformations. This is certainly possible in the mobile robot system mentioned in Section 4.1.1 where any two mobile robots are stationary and the third robot is roaming around. For systems in Figures 4.1, 4.2 and 4.3, fixing A or C means not moving one manipulator while moving the other. But fixing B and varying A, C means that the sensor frame to marker frame transformation (for Figures 4.1 and 4.3) or transformation between two sensor frames (for Figure 4.2) has to be fixed while moving the manipulators around. In such systems, this is not feasible as it would not be easy to move the two manipulators such that the relative transformation computed by the sensor is kept constant.

Two variations of the same basic method were proposed to handle systems where A, B or C can be fixed, and systems where only A or C (but not B) can be fixed. The first framework (which Ma et al calls "Prob2") is meant for systems like the mobile robot system that is described in Section 4.2.1. The second framework (which Ma et al calls "Prob1") works for systems like the dual manipulators and hybrid robot and is described in Section 4.2.2.

4.2.1 Prob2: Fix A or B or C

Each homogeneous transformation in (4.1) can be represented as a Dirac delta function in $SE(3)$. Hence each of the n equations can be transformed into convolutions of Dirac delta functions using the shifted Dirac delta function property (2.2) and the convolution property (2.8), and so

$$(\delta_{A_i} * \delta_X * \delta_{B_i})(K) = (\delta_Y * \delta_{C_i} * \delta_Z)(K)$$

where the variable $K \in SE(3)$. Since there are n sets of $\{A_i, B_i, C_i\}$, we can "sum" over the index i to get

$$\sum_{i=1}^{n} (\delta_{A_i} * \delta_X * \delta_{B_i})(K) = \sum_{i=1}^{n} (\delta_Y * \delta_{C_i} * \delta_Z)(K). \tag{4.2}$$

If we now define probability density functions f_H where $H \in \{A, B, C\}$ as

$$f_H(K) := \frac{1}{n}\sum_{i=1}^{n} \delta_{H_i}(K) = \frac{1}{n}\sum_{i=1}^{n} \delta\left(H_i^{-1}K\right)$$

which are all highly focused, then we can use (2.13) to get the approximation

$$(f_A * \delta_X * f_B)(K) \approx (\delta_Y * f_C * \delta_Z)(K). \tag{4.3}$$

Let $\bar{H}$ be the mean of f_H and using (2.10), we can transform (4.3) into

$$\bar{A}X\bar{B} \approx Y\bar{C}Z. \tag{4.4}$$

Similarly, let Σ_H be the covariances of f_H. Using (2.11), we obtain the following relationship for the covariances of f_H:

$$\mathrm{Ad}\big(\bar{B}^{-1}\big)\,\mathrm{Ad}\big(X^{-1}\big)\,\Sigma_A\,\mathrm{Ad}^{\top}\!\big(X^{-1}\big)\,\mathrm{Ad}^{\top}\!\big(\bar{B}^{-1}\big) + \Sigma_B \approx \mathrm{Ad}\big(Z^{-1}\big)\,\Sigma_C\,\mathrm{Ad}^{\top}\!\big(Z^{-1}\big) \tag{4.5}$$

where the adjoint of the inverse of a homogeneous transformation can be derived from (2.12) and is

$$\mathrm{Ad}\big(H^{-1}\big) = \mathrm{Ad}^{-1}(H) = \begin{bmatrix} R^{\top} & \mathbb{O}_3 \\ -[R^{\top}t]^{\wedge}R^{\top} & R^{\top} \end{bmatrix}.$$

Since the definition of adjoint operator in (2.12) is in block matrix form, this motivates us to write the covariances Σ_H in a similar way:

$$\Sigma_H = \begin{bmatrix} \Sigma_H^{(1)} & \Sigma_H^{(2)} \\ \Sigma_H^{(3)} & \Sigma_H^{(4)} \end{bmatrix} \in \mathbb{R}^{6\times 6}$$

where $\Sigma_H^{(i)} \in \mathbb{R}^{3\times 3}, i = 1, 2, 3, 4$.

Then we can write (4.5) in block matrix form and for the top left element, we get this matrix approximation

$$R_B^\top R_X^\top \Sigma_A^{(1)} R_X R_B + \Sigma_B^{(1)} \approx R_Z^\top \Sigma_C^{(1)} R_Z. \tag{4.6}$$

The bottom right block matrix gives

$$\begin{aligned} R_B^\top R_X^\top \Sigma_A^{(1)} \left([t_B^\wedge] R_B^\top R_X^\top + R_B^\top [t_X^\wedge] R_X^\top\right) + R_B^\top R_X^\top \Sigma_A^{(2)} R_X R_B + \Sigma_B^{(2)} \\ \approx R_Z^\top \Sigma_C^{(1)} R_Z [t_Z^\wedge]^\top + R_Z^\top \Sigma_C^{(2)} R_Z. \end{aligned} \tag{4.7}$$

Note that (4.6) allows us to solve for the rotations R_X and R_Z, while (4.7) involves the translations t_X and t_Z. Hence we can first solve (4.6) to get rotations and then use (4.7) to get the translations. Notice that Y has disappeared in (4.5) so there is no way to recover it from (4.6) and (4.7).

However we can permute the order of the homogeneous matrices in (3.7). For instance, premultiplying (3.7) by A^{-1} and postmultiplying by Z^{-1} on both sides of the equation gives

$$XBZ^{-1} = A^{-1}YC \tag{4.8}$$

which is a different "representation" of (3.7). We can do another permutation by premultiplying (4.8) by X^{-1} and postmultiplying by C^{-1} to get another representation. For each of the 6 permutations, the corresponding approximations for rotations, analgous to (4.6), are listed as follows:

1. $AXB = YCZ$:

$$R_B^\top R_X^\top \Sigma_A^{(1)} R_X R_B + \Sigma_B^{(1)} \approx R_Z^\top \Sigma_C^{(1)} R_Z \tag{4.9}$$

2. $A^{-1}YC = XBZ^{-1}$:

$$R_C^\top R_Y^\top \Sigma_{A^{-1}}^{(1)} R_Y R_C + \Sigma_C^{(1)} \approx R_{Z^{-1}}^\top \Sigma_B^{(1)} R_{Z^{-1}} \tag{4.10}$$

3. $BZ^{-1}C^{-1} = X^{-1}A^{-1}Y$:

$$R_{C^{-1}}^\top R_{Z^{-1}}^\top \Sigma_B^{(1)} R_{Z^{-1}} R_{C^{-1}} + \Sigma_{C^{-1}}^{(1)} \approx R_Y^\top \Sigma_{A^{-1}}^{(1)} R_Y \tag{4.11}$$

4. $B^{-1}X^{-1}A^{-1} = Z^{-1}C^{-1}Y^{-1}$:

$$R_{A^{-1}}^\top R_{X^{-1}}^\top \Sigma_{B^{-1}}^{(1)} R_{X^{-1}} R_{A^{-1}} + \Sigma_{A^{-1}}^{(1)} \approx R_{Y^{-1}}^\top \Sigma_{C^{-1}}^{(1)} R_{Y^{-1}} \tag{4.12}$$

5. $CZB^{-1} = Y^{-1}AX$:

$$R_{B^{-1}}^\top R_Z^\top \Sigma_C^{(1)} R_Z R_{B^{-1}} + \Sigma_{B^{-1}}^{(1)} \approx R_X^\top \Sigma_A^{(1)} R_X \tag{4.13}$$

6. $C^{-1}Y^{-1}A = ZB^{-1}X^{-1}$:

$$R_A^\top R_{Y^{-1}}^\top \Sigma_{C^{-1}}^{(1)} R_{Y^{-1}} R_A + \Sigma_A^{(1)} \approx R_{X^{-1}}^\top \Sigma_{B^{-1}}^{(1)} R_{X^{-1}} \tag{4.14}$$

Fixing A, B or C means that

$$\Sigma_A = \mathbb{O}, \quad \Sigma_B = \mathbb{O}, \quad \Sigma_C = \mathbb{O}$$

respectively and this is denoted as the zero-covariance constraint. This constraint enables us to simplify the approximations (4.9), (4.10), (4.12), (4.13), (4.11) and (4.14) into the form shown in Table 4.1. By fixing A, it turns out that the covariance equations for representation 1 and 2 have the same simplifed form. Furthermore, the approximations become equations because when A is fixed, the summation in (4.2) passes through to B and instead of (4.3), we get

$$(\delta_A * \delta_X * f_B)(K) = (\delta_Y * f_C * \delta_Z)(K).$$

A similar argument holds when B or C is fixed.

No.	Representation	Fixing	Simplified Form	Eq. number
1	$AXB = YCZ$	A	$\Sigma_B^{(1)} = R_Z^\top \Sigma_C^{(1)} R_Z$	(4.15)
2	$A^{-1}YC = XBZ^{-1}$			
3	$BZ^{-1}C^{-1} = X^{-1}A^{-1}Y$	B	$\Sigma_{C^{-1}}^{(1)} = R_Y^\top \Sigma_{A^{-1}}^{(1)} R_Y$	(4.16)
4	$B^{-1}X^{-1}A^{-1} = Z^{-1}C^{-1}Y^{-1}$			
5	$CZB^{-1} = Y^{-1}AX$	C	$\Sigma_{B^{-1}}^{(1)} = R_X^\top \Sigma_A^{(1)} R_X$	(4.17)
6	$C^{-1}Y^{-1}A = ZB^{-1}X^{-1}$			

Table 4.1: The simplified equations for covariances of rotations after fixing A, B or C in turn

Hence by fixing A and varying B and C, we can solve (4.15) for R_Z. To do this, we note that there is a similarity transformation between $\Sigma_B^{(1)}$ and $\Sigma_C^{(1)}$ since for a rotation matrix $R^\top = R^{-1}$. Hence they share the same three eigenvalues that are then used to form the diagonal elements of the 3×3 diagonal matrix Λ_1. Then we compute their eigendecomposition as

$$\Sigma_B^{(1)} = Q_B \Lambda_1 Q_B^\top \tag{4.18}$$

$$\Sigma_C^{(1)} = Q_C \Lambda_1 Q_C^\top \tag{4.19}$$

where the columns of Q_B, Q_C are the eigenvectors of $\Sigma_B^{(1)}, \Sigma_C^{(1)}$ respectively. Note that Q_B, Q_C are orthogonal because the covariance matrices Σ_H are constructed to be symmetric and hence the top left block is also symmetric. Substitute (4.18) and (4.19) into (4.15):

$$\begin{aligned} Q_B \Lambda_1 Q_B^\top &= R_Z^\top \left(Q_C \Lambda_1 Q_C^\top \right) R_Z \\ \Lambda_1 &= \underbrace{Q_B^\top R_Z^\top Q_C}_{S_1} \Lambda_1 Q_C^\top R_Z Q_B \\ &= S_1 \Lambda_1 S_1^\top \end{aligned}$$

where

$$S_1 := Q_B^\top R_Z^\top Q_C. \tag{4.20}$$

If Q_B and Q_C are further constrained to be rotation matrices, then according to [2], the possible solutions of S_1 for an equation with such structure are

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{4.21}$$

Substituting each of these 4 solutions into (4.20) produces 4 solutions for R_Z

$$R_Z = Q_C S_1 Q_B^\top$$

where we used the fact that $S_1 = S_1^\top$.

We can also simplify (4.7) using the zero-covariance constraint $\Sigma_A = \mathbb{O}$ to get

$$\Sigma_B^{(2)} = R_Z^\top \Sigma_C^{(1)} R_Z [t_Z^\wedge]^\top + R_Z^\top \Sigma_C^{(2)} R_Z. \tag{4.22}$$

Then using each possible value of R_Z, a value of t_Z can be computed by rearranging the terms of 4.22:

$$t_Z = \left[\left(\Sigma_B^{(2)} - R_Z^\top \Sigma_C^{(2)} R_Z \right)^\top R_Z^\top \left(\Sigma_C^{(1)} \right)^{-1} R_Z \right]^\vee \tag{4.23}$$

where we used the facts that $\Sigma_C^{(1)}$ is symmetric and $(H^{-1})^\top = (H^\top)^{-1}$ for a square matrix H.

By fixing B and varying A, C, the same steps as the above are used for solving (4.16). Hence we compute the eigendecomposition of $\Sigma_{A^{-1}}^{(1)}$ and $\Sigma_{C^{-1}}^{(1)}$,

$$\begin{aligned}
Q_{C^{-1}} \Lambda_2 Q_{C^{-1}}^\top &= R_Y^\top \left(Q_{A^{-1}} \Lambda_2 Q_{A^{-1}}^\top \right) R_Y \\
\Lambda_2 &= \underbrace{Q_{C^{-1}}^\top R_Y^\top Q_{A^{-1}}}_{S_2} \Lambda_2 Q_{A^{-1}}^\top R_Y Q_{C^{-1}} \\
&= S_2 \Lambda_2 S_2^\top
\end{aligned}$$

where

$$S_2 := Q_{C^{-1}}^\top R_Y^\top Q_{A^{-1}}, \tag{4.24}$$

and Λ_2 is the diagonal matrix made of the common eigenvalues of $\Sigma_{A^{-1}}^{(1)}$ and $\Sigma_{C^{-1}}^{(1)}$. Since (4.20) has the same structure as (4.24), the solutions of S_2 are also one of the 4 matrices in (4.21). Hence the 4 possible solutions of R_Y can be computed using

(4.24):

$$R_Y \approx Q_{A^{-1}} S_2 Q_{C^{-1}}^\top.$$

For each solution of R_Y, we can compute a possible t_Y using the zero-covariance constraint on the counterpart equation to (4.22). This produces a corresponding equation for t_Y similar to (4.23).

Lastly, we can also fix C and vary A, B to solve (4.17). We carry out the same process with $\Sigma_{B^{-1}}^{(1)}$ and $\Sigma_A^{(1)}$ to get

$$\begin{aligned}\Lambda_3 &= \underbrace{Q_{B^{-1}}^\top R_X^\top Q_A}_{S_3} \Lambda_3 Q_A^\top R_X Q_{B^{-1}} \\ &= S_3 \Lambda_3 S_3^\top\end{aligned}$$

where

$$S_3 := Q_{B^{-1}}^\top R_X^\top Q_A, \tag{4.25}$$

and Λ_3 is the diagonal matrix made of the common eigenvalues of $\Sigma_A^{(1)}$ and $\Sigma_{B^{-1}}^{(1)}$. The 4 possible solutions of S_3 are also in (4.21) and hence the 4 possible solutions of R_X can be computed using (4.25):

$$R_X = Q_A S_3 Q_{B^{-1}}^\top.$$

Then the corresponding 4 solutions for t_X can be computed using the analagous equation to (4.23).

With the 4 possible solutions of R_X, we obtained 4 possible solutions of t_X and similarly for R_Y, t_Y, R_Z, t_Z. Since R_X, R_Y, R_Z were solved independently, there are a total of $4 \times 4 \times 4 = 64$ possible combinations of the homogeneous transformations $\{X_i, Y_j, Z_k\}$. In order to identify the correct set that solves (4.1), we form an optimization problem by minimizing the errors of the rotation and translation components. By fixing A, we can define two expressions which are essentially the left and right hand side of (4.1):

$$\mathcal{A}_i := AX_i\bar{B}, \quad i = 1, \ldots, 4 \tag{4.26}$$

$$\mathcal{A}_{jk} := Y_j\bar{C}Z_k, \quad j = 1, \ldots 4,\ k = 1, \ldots, 4 \tag{4.27}$$

Similarly, fixing B allows us to define:

$$\mathcal{B}_i := \bar{A}X_iB, \quad i = 1, \ldots, 4 \tag{4.28}$$

$$\mathcal{B}_{jk} := Y_j\bar{C}Z_k, \quad j = 1, \ldots 4,\ k = 1, \ldots, 4 \tag{4.29}$$

and fixing C gives us:

$$\mathcal{C}_i := \bar{A}X_i\bar{B}, \quad i = 1, \ldots, 4 \tag{4.30}$$

$$\mathcal{C}_{jk} := Y_jCZ_k, \quad j = 1, \ldots 4,\ k = 1, \ldots, 4. \tag{4.31}$$

Note that the transformations in script font are also homogeneous matrices, and hence they have rotation and translation components too. The optimization problem seeks to minimize the errors in rotation and translation between the left and right hand sides using the metrics stated in (2.15) and (2.14). It also has a weighting factor w that can be tweaked depending on the required amount of translational error compared to the rotational error. Since this minimization problem

$$\begin{aligned}\arg\min_{i,j,k}\Big[&\left\|\log^\vee\left(R_{\mathcal{A}_i}^\top R_{\mathcal{A}_{jk}}\right)\right\|_2+\left\|\log^\vee\left(R_{\mathcal{B}_i}^\top R_{\mathcal{B}_{jk}}\right)\right\|_2+\left\|\log^\vee\left(R_{\mathcal{C}_i}^\top R_{\mathcal{C}_{jk}}\right)\right\|_2+\\ &w\|t_{\mathcal{A}_i}-t_{\mathcal{A}_{jk}}\|_2+w\|t_{\mathcal{B}_i}-t_{\mathcal{B}_{jk}}\|_2+w\|t_{\mathcal{C}_i}-t_{\mathcal{C}_{jk}}\|_2\Big]\end{aligned} \tag{4.32}$$

is discrete, the solution just involves iterating over the i, j, k indices and finding the set of indices that minimizes the objective function. The returned i, j, k values will then correspond to a set of $\{X_i, Y_j, Z_k\}$ that is the solution to (4.1) returned by the algorithm.

To aid implementation, the steps of Prob2 as described in Section 4.2.1 are explicitly listed here. Before this algorithm can be used, the data has to be in the form in Table 4.2. Hence for data set $\mathcal{A}$, we fix A and in the mobile robot system, this means two mobile robots are stationary. Then by moving the third mobile robot, we can collect n sets of B and C measurements. This is indicated in the table by showing the fixed variable as repeated without subscripts, and the varied variables with subscripts 1 to n. The same process is done when fixing B or C and varying the other two. Hence in total, there should be $3n$ sets of measurement "triples" $\{A, B, C\}$ in all of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. Also note that the sizes of $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ need not be equal, although they are shown to be the same in Table 4.2.

Label of set	Pose being fixed	Measurement data	Mean
$\mathcal{A}$	A	$\overbrace{A,\ldots,A}^{n}$	A
		$B_1,\ldots,B_n$	$\bar{B}$
		$C_1,\ldots,C_n$	$\bar{C}$
$\mathcal{B}$	B	$A_1,\ldots,A_n$	$\bar{A}$
		$B,\ldots,B$	B
		$C_1,\ldots,C_n$	$\bar{C}$
$\mathcal{C}$	C	$A_1,\ldots,A_n$	$\bar{A}$
		$B_1,\ldots,B_n$	$\bar{B}$
		$C,\ldots,C$	C

Table 4.2: The data sets obtained after fixing A, B or C

In the algorithm described below, any operation performed on the transformation H applies to A,B and C.

Step 1 After collecting data sets $\mathcal{A},\mathcal{B},\mathcal{C}$, compute the mean $\bar{H}$ of each transformation $\{A_i\},\{B_i\},\{C_i\}$ that was varied in each set according to the definition:

$$\sum_{i=1}^{n} \log\left(\bar{H}^{-1}H_i\right) = \mathbb{O}_4$$

where the subscript H represents A, B or C. Practically, $\bar{H}$ can be calculated using an iterative formula [35]

$$\bar{H}_{k+1} = \bar{H}_k \exp\left[\frac{1}{n}\sum_{i=1}^{n}\log\left(\bar{H}_k^{-1}H_i\right)\right]$$

where a possible initial estimate is

$$\bar{H}_0 = \exp\left[\frac{1}{n}\sum_{i=1}^{n}\log H_i\right].$$

For the transformations that were fixed, the mean is just that homogeneous matrix. Hence for data set $\mathcal{A}$ there should be only three transformations at the end of this step: A, $\bar{B}$ and $\bar{C}$, as shown in the last column of Table 4.2. The same applies for data sets $\mathcal{B}$ and $\mathcal{C}$.

Step 2 After getting the means $\bar{H}$, the covariance of each set is computed using the definition:

$$\Sigma_H = \sum_{i=1}^{n} \log^\vee\left(\bar{H}^{-1}H_i\right)\left(\log^\vee\left(\bar{H}^{-1}H_i\right)\right)^\top .$$

Step 3 For each Σ_H, extract the top-left and top-right 3×3 block matrices which we call $\Sigma_H^{(1)}$ and $\Sigma_H^{(2)}$ respectively.

Step 4 Invert all the transformations in $\{A_i\}, \{B_i\}, \{C_i\}$. Then compute the mean and covariance as in **Step 1** and **Step 2** to get $\overline{H^{-1}}$ and $\Sigma_{H^{-1}}$ respectively.

Step 5 As in **Step 3**, extract the top-left and top-right blocks of every $\Sigma_{H^{-1}}$, which will be $\Sigma_{H^{-1}}^{(1)}$ and $\Sigma_{H^{-1}}^{(2)}$ respectively.

Step 6 Compute the eigendecomposition of every $\Sigma_H^{(1)}$ and $\Sigma_{H^{-1}}^{(1)}$ to get the three corresponding eigenvectors. Use these three eigenvectors to form the columns of a new matrix Q_H and $Q_{H^{-1}}$.

Step 7 Then the 4 possible solutions of R_X, R_Y, R_Z can be computed as shown in Table 4.3 where S is one of the 4 matrices in (4.21).

Step 8 Using the computed rotations, the 4 possible solutions of t_X can be found using (4.23). The respective formulas for t_Y and t_X are

$$t_Y = \left[\left(\Sigma^{(2)}_{C^{-1}} - R_Y^\top \Sigma^{(2)}_{A^{-1}} R_Y \right)^\top R_Y^\top \left(\Sigma^{(1)}_{A^{-1}} \right)^{-1} R_Y \right]^\vee , \tag{4.33}$$

$$t_X = \left[\left(\Sigma^{(2)}_{B^{-1}} - R_X^\top \Sigma^{(2)}_{A} R_X \right)^\top R_X^\top \left(\Sigma^{(1)}_{A} \right)^{-1} R_X \right]^\vee . \tag{4.34}$$

Label of set	Rotation	Translation	Homoegeneous matrix
$\mathcal{A}$	$R_Z = Q_C S Q_B^\top$	(4.23)	Z_k, $k = 1, \ldots 4$
$\mathcal{B}$	$R_Y = Q_{A^{-1}} S Q_{C^{-1}}^\top$	(4.33)	Y_j, $j = 1, \ldots 4$
$\mathcal{C}$	$R_X = Q_A S Q_{B^{-1}}^\top$	(4.34)	X_i, $i = 1, \ldots 4$

Table 4.3: The formulas for getting $R_X, R_Y, R_Z, t_X, t_Y, t_Z$

Step 9 Form the possible homogeneous matrices H_i using the computed values of R_H and t_H and we should have 64 sets of $\{X_i, Y_j, Z_k\}$.

Step 10 For each set of $\{X_i, Y_j, Z_k\}$, compute the transformations with the formulas (4.26),(4.28),(4.30),(4.27),(4.29),(4.31).

Step 11 Then extract out the rotational and translational components of these transformations and determine the values of i, j, k that give the minimum of the objective function in (4.32). The corresponding X_i, Y_j and Z_k is then the solution to the calibration problem (4.1).

4.2.2 Prob1: Fix A or C only

In the case where B cannot be fixed while varying A or C, we can only obtain data sets $\mathcal{A}$ and $\mathcal{C}$. Consequently, the method is almost similar except for fixing B and obtaining R_Y using (4.16), we obtain R_Y using the mean equation (4.4). First we need to compute R_X, t_X and R_Z, t_Z as described for Prob2 in Section 4.2.1 by fixing A and C respectively. By fixing A (i.e. $\bar{A} = A$), there are 4 values each for X and Z, which when substituted into (4.4) gives 16 possibilities for Y:

$$Y = AX\bar{B}Z^{-1}\bar{C}^{-1}.$$

Fixing C so that $\bar{C} = C$ also gives us another 16 possibilities for Y:

$$Y = \bar{A}X\bar{B}Z^{-1}C^{-1}$$

Hence there are a total of $4 \times 4 \times (16 + 16) = 512$ sets of $\{X_i, Y_j, Z_k\}$ from which we must find the optimal one. We then form a discrete minimization problem similar to (4.32) (which in this case does not involve data set $\mathcal{B}$):

$$\begin{aligned}\arg\min_{i,j,k}\Big[&\left\|\log^{\vee}\left(R_{\mathcal{A}_i}^{\top}R_{\mathcal{A}_{jk}}\right)\right\|_2 + \left\|\log^{\vee}\left(R_{\mathcal{C}_i}^{\top}R_{\mathcal{C}_{jk}}\right)\right\|_2 + \\ &w\|t_{\mathcal{A}_i} - t_{\mathcal{A}_{jk}}\|_2 + w\|t_{\mathcal{C}_i} - t_{\mathcal{C}_{jk}}\|_2\Big]\end{aligned} \tag{4.35}$$

and can be solved as well to give the optimal $\{X_i, Y_j, Z_k\}$. Here the weight w has the same function as in (4.32) but its ideal value may be different.

Chapter 5

Simulations

Presently there are three other methods that solve the $AXB = YCZ$ type problem as stated in the literature review (Chapter 3). We will call the method in [36], "Wu" and the two methods in [37], "DK" and "PN". The theory in Chapter 4 indicated that the strengths of the MGC method compared to these existing solutions are

1. the ability to handle loss of correspondence between the $\{A_i\}$, $\{B_i\}$, $\{C_i\}$ data, and
2. the ability to not require initial estimates in order to compute the solution,

while its weaknesses are that

1. it does not handle noisy data very well, and
2. sometimes it returns a non-optimal X, Y, Z depending on the objective function.

This chapter describes simulations that were carried out to verify and evaluate the strengths and weaknesses of this method that have predicted by the theory. Data were collected during numerical simulations where some condition was varied while keeping the other conditions constant.

Section 5.1 lists the procedure that was used to carry out the simulation, including the parameters that were varied or kept constant. Section 5.2 shows the results by plotting them on line graphs. Lastly, interpretaion and discussion of the results are in Section 5.3.

5.1 Procedure

The simulations in this chapter were carried out using MATLAB 2016a. The following parameters were kept constant across simulations.

1. Number of measurement data, n, every time A, B or C was fixed. This is the same n shown in Table 4.2.

2. Number of trials, N, is the number of times the algorithms executes, each time with a different set of measurement data $\{A_i, B_i, C_i\}$ that was generated randomly using MATLAB's `randn` function. The purpose of running multiple executions was to be able to take the average of the results.

3. The value of w in the objective functions (4.32) and (4.35).

For each simulation, the following parameters were varied one at a time, i.e. if one parameter was varied the others were kept constant:

1. Scrambling rate r,
2. Standard deviation of the noise applied to the data, σ_{noise},
3. Standard deviation of the data, σ_{data},

Using the above conditions and $n = 100$, $N = 10$ and $w = 1.5$, the simulations were executed using the following steps:

Step 1 **XYZ Generation:** Transformations for X, Y, Z were selected by generating a normally distributed random vector $\zeta \in se(3)$ using MATLAB's `randn` function

$$\zeta \sim \mathcal{N}\left(\mathbb{O}_{6\times 1}, \mathbb{I}_6\right).$$

Then the corresponding $SE(3)$ transformation was obtained using the matrix exponential function `expm` in MATLAB:

$$H = \exp\left([\zeta]^{\wedge}\right)$$

where $H \in \{X, Y, Z\}$.

Step 2 **ABC Generation:** The initial transformation for each of A, B, C was generated using the kinematics of the Puma 560 serial manipulator. The 6 joint angles of the manipulator that were used to generate the initial transformations are shown in Table 5.1.

Transformation	J1	J2	J3	J4	J5	J6
A_{init}	$\frac{\pi}{6}$	$\frac{\pi}{3}$	$\frac{\pi}{4}$	$\frac{\pi}{4}$	$-\frac{\pi}{4}$	0
B_{init}	$\frac{\pi}{3}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$-\frac{\pi}{4}$	$\frac{\pi}{4}$	0
C_{init}	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{3}$	$\frac{\pi}{6}$	$-\frac{\pi}{4}$	0

Table 5.1: The Puma's joint angles (rad) that were used to generate the initial transformations for A, B, C

Then the initial transformations was perturbed by the standard deviation of the data for the current trial, σ_{data}, to obtain the data sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ for $i = 1, \ldots 100$ in Table 4.2. For instance, to generate the data set $\mathcal{A}$ A was fixed, and hence A_{init} was repeated so that $\{A\}$ had the same number of elements as $\{B_i\}$ and $\{C_i\}$. Then `randn` was used to generate a vector $\delta_i \in se(3)$

$$\delta_i \sim \mathcal{N}\left(\mathbb{O}_{6\times 1}, \sigma_{\text{data}}\mathbb{I}_6\right)$$

and that was used to perturb $\{B_i\}$ as follows:

$$B_i = \exp\left([\delta_i]^\wedge\right) B_{\text{init}}.$$

The corresponding C_i was then computed using (4.1):

$$C_i = Y^{-1} A_{\text{init}} X B_i Z^{-1}.$$

After doing similar operations, data sets $\mathcal{B}$ and $\mathcal{C}$ were obtained. The norm of the covariances, $\|\Sigma_A\|$, $\|\Sigma_B\|$, $\|\Sigma_C\|$, were also computed for the generated data to verify that they satisfied the highly focused assumption (2.9) for the values of σ_{data} that was chosen.

Step 3 **Scramble Data:** Next the correspondences between $\{A_i\}$, $\{B_i\}$ and $\{C_i\}$ were scrambled. Depending on the desired scrambling rate r, $r\%$ of the n data triples (A_i, B_i, C_i) in $\mathcal{A}$ were randomly selected and the order switched. Since all the A_i data in $\mathcal{A}$ are identical, this essentially switched the B_i and C_i pairs. For data sets $\mathcal{B}$ and $\mathcal{C}$, another $r\%$ was selected randomly and the process was similar.

Step 4 **Add Noise:** Following [36], noise was applied to the rotations, using the angle-axis parameterization where θ (in degrees) was the angle about a random unit vector k that each rotational component in the scrambled data sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ would be perturbed. Hence

$$R_H^{\text{noise}} = R_H \text{Rot}(k, \theta)$$

where H represents A, B, C. The translational component was shifted ϵ mm in the direction of a random unit vector p. Thus

$$t_H^{\text{noise}} = t_H + \epsilon p.$$

The values of θ and ϵ were taken from a continuous uniform distribution:

$$\theta \sim \mathcal{U}(-\theta_{\max}, +\theta_{\max})$$
$$\epsilon \sim \mathcal{U}(-\epsilon_{\max}, +\epsilon_{\max})$$

The applied noise was varied by changing $\theta_{\max}$ and $\epsilon_{\max}$. The `randn` function was used to obtain the random unit vectors k and p by generating a random 3×1 vector and dividing it by its norm.

Step 5 **Input into Method:** Now the data sets $\mathcal{A}, \mathcal{B}, \mathcal{C}$ have some level of noise applied. Then the data sets were passed to each method according to Table 5.2. Each method then returned a set of homogeneous matrices $X_{\text{solved}}, Y_{\text{solved}}, Z_{\text{solved}}$ which solved (3.7).

Method	Data sets
Prob2	$\mathcal{A}, \mathcal{B}, \mathcal{C}$
Prob1	$\mathcal{A}, \mathcal{C}$
Wu	$\mathcal{A}, \mathcal{B}, \mathcal{C}$
DK	$\mathcal{A}, \mathcal{C}$
PN	$\mathcal{A}, \mathcal{B}, \mathcal{C}$

Table 5.2: The data sets that were passed into each method

Step 6 **Compute Errors:** By comparing the solved values with the true values in **Step 1**, the rotational and translational errors of X, Y, Z were computed separately using the metrics in (2.15) and (2.14) as such:

$$\begin{aligned} \text{Error}(R_H) &= \|\log^\vee(R_{\text{solved}}^T R_{\text{true}})\| \\ \text{Error}(t_H) &= \frac{\|t_{\text{solved}} - t_{\text{true}}\|}{\|t_{\text{true}}\|}. \end{aligned}$$

The values of $\text{Error}(R_H)$ and $\text{Error}(t_H)$ for $H \in \{X, Y, Z\}$ were then plotted as line plots in Section 5.2.

The above procedure is shown in Appendix A Listing A.1. Note that the MATLAB code assumes that Peter Corke's Robotics Toolbox v9.9 has been installed. It also requires the following toolboxes from MATLAB:

1. Statistics and Machine Learning,
2. Optimization.

5.2 Results

This sections states the results from numerical simulations in MATLAB. The probabilistic algorithms, Prob1 and Prob2, and the non-probabilistic "traditional" algorithms, Wu, DK and PN, were executed under the various conditions listed in Table 5.3. The noise level values for θ_{max} and ϵ_{max} were chosen to mirror the accuracies of most modern industrial manipulators.

Simulation	Vary	Constant
I	Scrambling rate $r \in \{0, 20, 40, 60, 80, 100\}$	$\begin{bmatrix}\theta_{\text{max}} \\ \epsilon_{\text{max}}\end{bmatrix} = \begin{bmatrix}0 \\ 0\end{bmatrix}, \sigma_{\text{data}} = 0.02$
II	Noise Level $\begin{bmatrix}\theta_{\text{max}} \\ \epsilon_{\text{max}}\end{bmatrix} \in \left\{\begin{bmatrix}0 \\ 0\end{bmatrix}, \begin{bmatrix}0.05 \\ 0.1\end{bmatrix}, \begin{bmatrix}0.1 \\ 0.5\end{bmatrix} \begin{bmatrix}0.5 \\ 1\end{bmatrix} \begin{bmatrix}1 \\ 2\end{bmatrix}\right\}$	$r = 0,\ \sigma_{\text{data}} = 0.02$
III	Std Dev of Data $\sigma_{\text{data}} \in \{0.02, 0.04, 0.06, 0.08, 0.10\}$	$r = 0, \begin{bmatrix}\theta_{\text{max}} \\ \epsilon_{\text{max}}\end{bmatrix} = \begin{bmatrix}0 \\ 0\end{bmatrix}$

Table 5.3: The conditions for each simulation

Figure 5.1 shows the results of Simulation I by varying the scrambling rate applied to the $\{A_i\}, \{B_i\}, \{C_i\}$ data without applying noise and keeping $\sigma_{\text{data}} = 0.02$. Because the magnitude of the errors of the probabilistic and traditional methods had a large difference, we used the logarithmic scale for the y-axis to show the differences on the same plot.

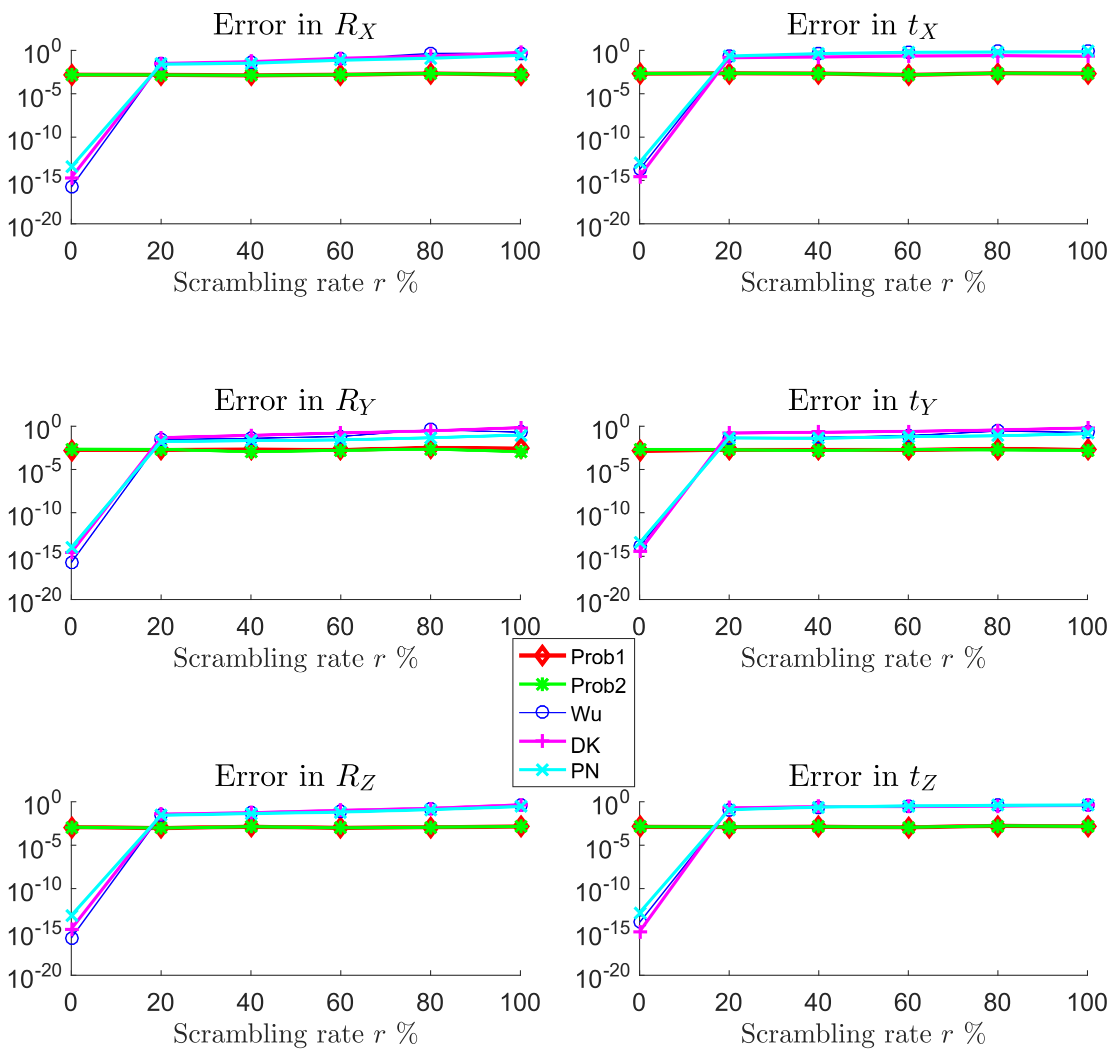

Figure 5.1: Varying scrambling rate while keeping $\sigma_{\text{data}} = 0.02$ and $(\theta = 0, \epsilon = 0)$

Figure 5.2 shows the results of Simulation II by varying the noise applied to the $\{A_i\}, \{B_i\}, \{C_i\}$ data while scrambling 0% of them and keeping $\sigma_{\text{data}} = 0.02$. The horizontal axis indicates the maximum perturbation in orientation $\theta_{\max}$ (in deg) and position $\epsilon_{\max}$ (in mm) of the applied noise.

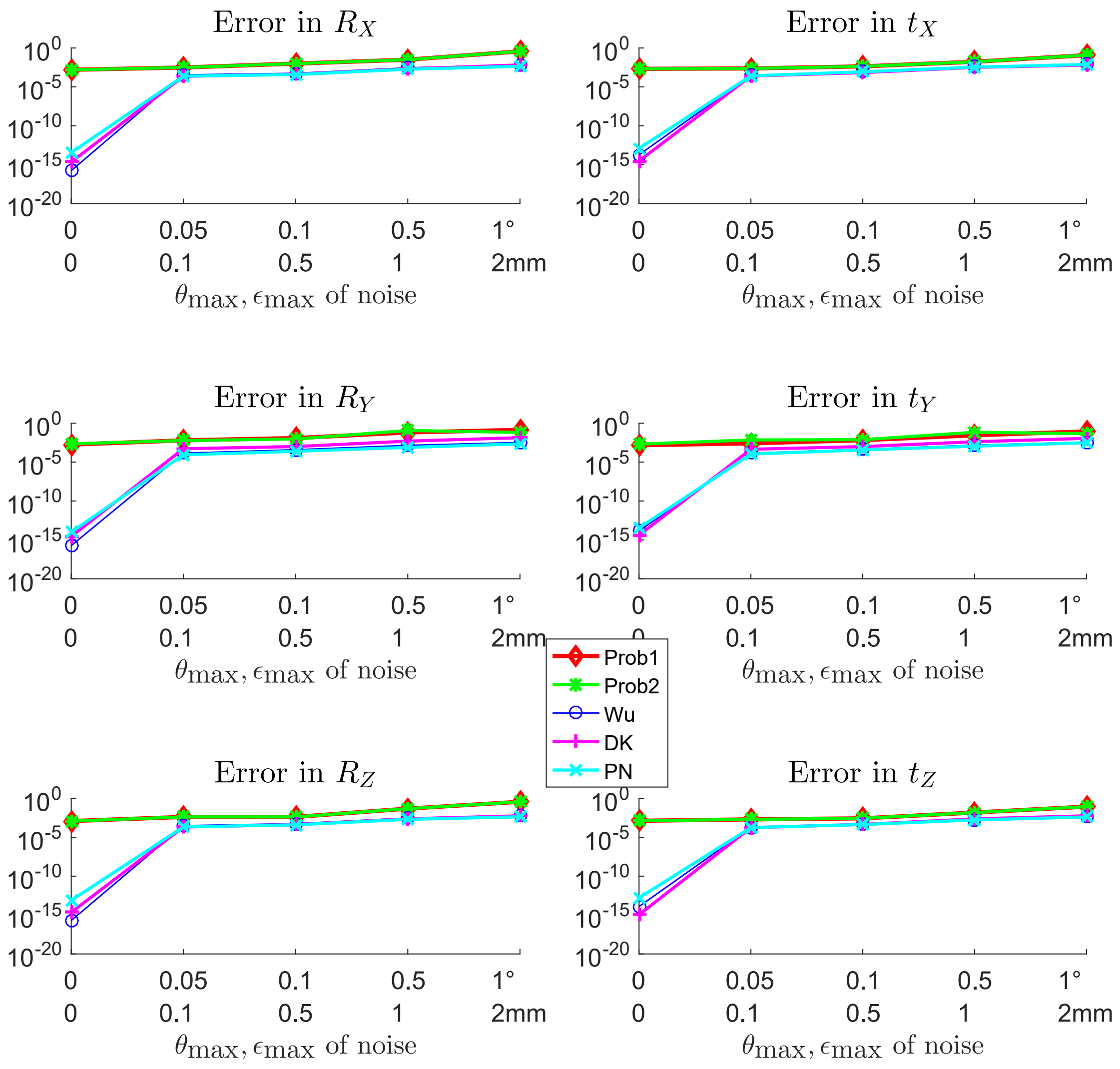

Figure 5.2: Varying noise level while keeping $\sigma_{\text{data}} = 0.02$ and $r = 0\%$

Figure 5.3 shows the results of Simulation III by varying the standard deviation of the $\{A_i\}, \{B_i\}, \{C_i\}$ data with 0% of them scrambled and without applying noise.

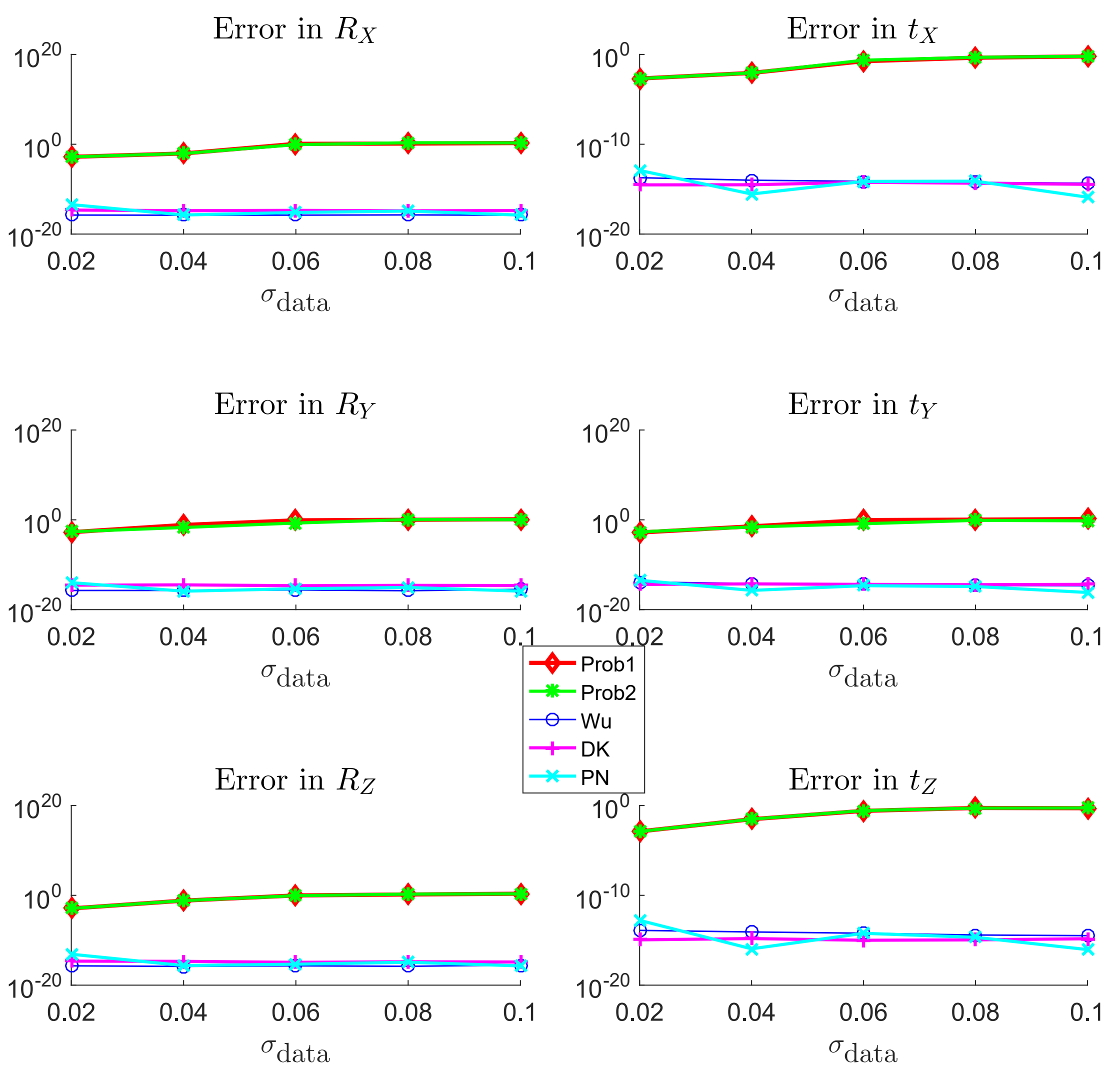

Figure 5.3: Varying data standard deviation while keeping $(\theta = 0, \epsilon = 0)$ and $r = 0\%$

5.3 Discussion

From the graphs in Section 5.2 the following observations can be made:

1. From Figure 5.1, when there is perfect correspondence and no noise in the data, the traditional methods have a lower error than the probabilistic ones. However

when the scrambling rate increases, the errors from Prob1 and Prob2 remains fairly constant while Wu and DK increases exponentially (since a straight line in a log plot indicates an exponential relationship). It was expected that the probabilistic methods would outperform the non-probabilistic ones when the data had a loss of correspondence and these simulations proved the theory.

Within each class of methods – probabilistic and non-probabilistic – the differences in errors are not significant. This is expected because any method either handles scrambled data well or not, and hence each class reacts to scrambled data in a similar way.

2. Since Prob1 and Prob2 are sequential methods, any error in computing the rotational component will be propagated to the translational component. The error propagation property also holds for the Wu method. Although DK and PN are simultaneous solvers, the absence of error propagation in these solvers compared to the others is not noticeable from the plots since other factors have a larger influence on the final errors of X, Y, and Z. Also it is not meaningful to compare the magnitutes of the rotational vs translational errors, and hence it is hard to quantify the amount of error propagation in sequential methods.

3. Prob1 uses X and Z to compute Y, and hence error from the former gets propagated to the latter. The same is true for DK. Hence Prob2 is expected to have lower error than Prob1 although this cannot be clearly seen from the plots.

4. The highly focused assumption needed for the approximations in Prob1 and Prob2 to be valid meant that the standard deviation of the data had to be small. Table 5.4 shows the average values of the covariance norms for each σ_{data} and it shows that they satisfy (2.9). As seen from Figure 5.3 the errors of Prob1 and Prob2 increased as σ_{data} increased which is expected. Notice also that the plots for the non-probabilistic methods are clearly lower than the probabilistic ones over the entire range of σ_{data}, and this is also expected when the data are not scrambled (i.e. perfect correspondence). In fact, the non-probabilistic methods would have lower error as σ_{data} increased and this is also seen in their plots which have a slight downward slope.

   This requirement of a small spread of the data is interesting and counter-intuitive because having a small spread makes the data susceptible to be degenerate and any noise in the data will have a huge impact on the result. However, the requirement for highly focused data makes it easier and faster to collect data on a real system because less time and effort will be needed to move the robots or sensors around.

σ_{data}	$\|\Sigma_A\|$	$\|\Sigma_B\|$	$\|\Sigma_C\|$
0.02	0.0023	0.0021	0.0043
0.04	0.0043	0.0040	0.0043
0.06	0.0069	0.0059	0.0130
0.08	0.0098	0.0078	0.0184
0.10	0.0172	0.0102	0.0293

Table 5.4: The average values of $\|\Sigma\|$ based on data generated from σ_{data} compared with experimental data

5. Because of the need for highly focused data, even a small amount of noise added to the data magnifies the error and so it does not perform well. This is evident from Figure 5.2 where the errors of Prob1 and Prob2 increased as the noise increased, while Wu, DK and PN had errors that were also increasing (at about the same rate) but were lower than the probabilistic ones across rotation and translation.

6. The choice of objective functions in (4.32) and (4.35) turned out to be critical for Prob1 and Prob2 to return the optimal solution. The role of the objective functions is to select the optimal $\{X, Y, Z\}$ from a set of 512 and 64 candidates for Prob1 and Prob2 respectively. If the functions returned non-optimal solutions for some of the trials, the average error would be increased by these outliers.

7. Wu and PN are iterative optimization methods and require reasonably good initial guesses in order to reach a global minimum. In contrast, Prob1 and Prob2 do not need initial guesses, and likewise DK. The price to pay for this is that the data sets have to be constructed by fixing either A or C. In [36], Wu et al used a closed form solution to obtain an initial estimate but Yan et al [37] did not provide an alternative than using random transformations as the estimates. To mitigate the effect of non-global optimum solutions when a bad initial estimate was generated, in Section 5.1 **Step 5** only PN was executed 10 times with the same set of data but with different random initial estimates. The

errors from the 10 executions were computed in **Step 6** and the minimum error was used as the final result for that set of data.

To determine which method is the best when solving (3.7), the following recommendations can be made depending on the application and the most important factor for the user:

1. When correspondence between data will be lost: if the data lacks full orrespondence, the probabilistic methods should be used;
2. If rotational accuracy is more important than translational accuracy for that application, then sequential methods like Wu can be used. Otherwise, simultaneous methods like PN should be applied instead;
3. If the variation in the A, B and C data fulfils the highly focused assumption, then the probabilistic methods should be applied;
4. If initial estimates of X, Y, Z are easily obtainable and are accurate enough, then the iterative solutions like PN and Wu should be used. Otherwise methods which do not require initial estimates should be used, namely Prob1, Prob2 and DK;
5. If the B transformation can be fixed in the robotic system, Prob2 should be used instead of Prob1 because it produces lower errors in general.

6. If the noise level in the data is expected to be high, the non-probabilistic methods should be used.

Hence the choice of the most appropriate method comes down to prioritizing each factor so as to take into account the strengths and weaknesses of each method.

Chapter 6

Conclusion

My thesis surveyed the literature regarding three main types of calibration problems relating to sensors and robots:

1. hand-eye calibration formulated as $AX = XB$,

2. hand-eye/robot-world calibration formulated as $AX = YB$, and

3. multi-robot and sensor calibration formulated as $AXB = YCZ$.

All the methods were classified based on the approach taken to solve the relevant matrix equation. When the problems were new, most of the proposed solutions were in closed-form. But with the recent rise of faster computing in smaller packages, more iterative solutions have come out.

The motivation for solving the $AXB = YCZ$ calibration problem instead of solving two $AX = XB$ problems separtely is to avoid error in the computation of

X or Z propagating to Y. In particular, I have reviewed in detail the MGC method for solving $AXB = YCZ$. The MGC method contains two related probabilistic approaches that are applicable to different robot systems. For systems that allow either A, B or C measurements to be fixed, the Prob2 method can be used, while Prob1 can only be used for systems where only A or C can be fixed. I also compared its performance with other solutions by running simulations that varied the amount of scrambling, noise and spread in the data. The advantage of the MGC method is most clearly seen when the correspondence between the data triplets are lost. And by fixing one transformation A, B or C, the MGC method does not require initial estimates for obtaining X, Y, Z. However the other methods like Wu, DK and PN perform better when the data has perfect correspondence but is noisy.

6.1 Future Work

1. Obtaining the robot-sensor transformation simultaneously for a fleet of robots would be quicker and avoid error propagation issues compared to doing it separately for each robot. However the $AXB = YCZ$ formulation extends to only a maximum of three robots. As swarm robotics becomes more common, it remains to be seen if there is a similar generalization for calibrating N robots simultaneously. It may be that as N becomes very large, the calibration problem becomes intractable.

2. Ma et al [24] proposed two new definitions of the mean of distributions on $SE(3)$ for the hand-eye calibration problem that would replace (2.6). They found that was smaller error in X with the new definitions. Hence it would be interesting to see if there are similar results when applied to the $AXB = YCZ$ problem.

3. The results show that the MGC method does not handle noise very well. If the method could be adapted to handle a reasonable amount of noise, then this probabilistic approach could be used on real-world data.

4. In the last step of the MGC method, optimization problems 4.32 and (4.35) had to be solved to obtain the solutions to X, Y and Z. However the solutions were not always globally optimal and hence further investigation can be conducted for better cost functions.

5. There have been theoretical results on the minimum number of data required for solving (3.1) and (3.4) uniquely without noise. It would be interesting if a corresponding result could be obtained for (3.7).

In conclusion, the calibration problem for multi-robot systems is just starting and new approaches will be expected in the future. This is especially as researchers are working more and more with multiple robots that cooperate to perform tasks. The Holy Grail method that can calibrate all robots simultaneously, be robust to noise and non-corresponding data might well be within reach.

Appendix A

Code implementation for $AXB = YCZ$ methods

This appendix contains the MATLAB code for generating the plots in Chapter 5 and the functions for the $AXB = YCZ$ algorithms in Wu et al [36] and Yan et al [37].

```
%% Main program for ZG thesis simulations (NOT for RSS paper)
clear all
close all
% clc
warning('off', 'MATLAB:logm:nonPosRealEig') %supress warnings about nonpositive
    eigenvalues
warning('off', 'MATLAB:hg:DiceyTransformMatrix') % det of RX,RY,RZ not 1 within eps
    so warning appears

%%
fprintf('Execution started at %s\n', datetime('now','TimeZone','local','Format','d-
    MMM-y HH:mm:ss Z'))
tic;

rng default % for reproducibility when calling mvg in ABC_Generate (remove after
    testing)

%% Set parameters for the experiment
% what to vary:
% 1:  scrambling rate from 0 to 100%, std dev of data=0.02, no noise
% 2:  std dev of data from 0.02 to 0.1, scrambling rate=1%, no noise
% 3:  std dev of noise from 0 to 0.01, scrambling rate=1%, std dev of data=0.02
% 4:  std dev of noise from 0 to 0.01, scrambling rate=10%, std dev of data=0.02
varywhat = 2;
removeOutliers = false;

```

```
% keep these 2 constant
numTrials = 10; % number of simulations
numData = 100; % number of data/measurements per family of A,B,C (depending on
                % whether A, B or C is kept constant and the other 2 varied

% what to plot
tfplots = false;  % should display graphs of X,Y,Z 3D plots?
lineplot = true;

fprintf('Experiment conditions:\n-------------------------------------\n');
fprintf('  Num of trials: %d\n', numTrials);
fprintf('  Num of measurements per family of A,B,C: %d\n', numData);
if removeOutliers
  fprintf('  Removing outliers\n');
else
  fprintf('  Not removing outliers\n');
end

% decide what methods to execute - useful for debugging each method without long
    execution times
% order of binary switches are : Prob1, Prob2, Wang, DK, PN
methodNames = {'Prob1', 'Prob2', 'Wu', 'DK', 'PN'};
methods2run = [1 1 1 1 1];

% scrambling rate
scramRate = 0:20:100;

% weighting factor in objective function of Prob1 and Prob2 (not useful for
% evaluating other methods)
weight = 0.2:0.2:2.0;

meanD = [0; 0; 0; 0; 0 ;0]; % mean for generating data A, B, C
Cov = eye(6,6); % cov for generating A, B, C.  Will be multiplied by sigD later.
sigD = 0.02:0.02:0.1; % std dev for generating A, B, C

% mean and cov for noise (affects both rot and trans)
% meanN = [0;0;0;0;0;0];  %Gaussian Noise Mean
% sigN = 0.000:0.002:0.01;  %Gaussian Noise standard deviation Range
% rep rot and trans noise as cell array of 1x2 matrices which rep noise values
%     [ rotation noise (deg), translation noise (mm) ]
sigN = { [0,0], [0.05, 0.1], [0.1, 0.5], [0.5, 1], [1, 2 ] };


%% generate random X, Y and Z
ran = 1;
[XActual,YActual,ZActual] = generateXYZ(ran);


%% vary conditions and generate A, B, C data
switch varywhat
  case 1 % vary scrambling rate, fix data std dev = 0.02 and noise std dev = 0
    sigD = sigD(1);  % take first element of range as the constant value
    sigN = sigN(1);
    % variable only used for printing to terminal
    variable = ['scrambling rate and sigma for data=', num2str(sigD), ' & noise= ',
    num2str(sigN{1})];

  case 2 % vary data std dev, fix scrambling rate = 1% and noise std dev = 0
    sigN = sigN(1);
    scramRate = scramRate(1);
    variable = ['data std dev with scram-rate=', num2str(scramRate), ' & sigma for
    noise=', num2str(sigN{1})];

  case 3 % vary noise std dev, fix scrambling rate = 1% and data std dev = 0.02
    sigD = sigD(1);
    scramRate = scramRate(1);
    variable = ['noise std dev with scram-rate=', num2str(scramRate), ' & sigma for
    data=', num2str(sigD) ];

  case 4 % vary noise std dev, fix scrambling rate = 10% and data std dev = 0.02
    sigD = sigD(1);
    scramRate = 10;
    variable = ['noise std dev with scram-rate=', num2str(scramRate), ' & sigma for
    data=', num2str(sigD) ];

  case 5 % vary weighting factor (reuse variable sigN, i.e. as if varying noise)
    sigD = sigD(1);
    scramRate = scramRate(1);
    sigN = weight;
    variable = ['weight with scram-rate=', num2str(scramRate), ' & sigma for data=', num2str(sigD) ];
end

fprintf('  Varying only %s\n', variable);

% Allocated sizes of arrays to store error data
Err1 = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials); %3rd dim=6 because we store err for each of 6 DOF
Err2 = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials);
ErrWang = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials);
ErrDK = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials);
ErrPN = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials);

% allocated arrays for storing norms of covariances of B1,C1,A3,C3,A2,B2 data over all trials
normCov = zeros(length(scramRate), length(sigD), length(sigN), 6, numTrials);

counter = 0;
for sr = 1:length(scramRate) % vary the scrambling rate
  for sD = 1:length(sigD) % vary the std dev of data generated
    for sN = 1:length(sigN) % vary the std dev of noise generated

      % zg+ Plot the actual X, Y, Z in red
      if tfplots
        counter = counter + 1;
        ftr = figure(counter);
        ftr.Name = ['Prob1 when std dev of A, B, C data is ' num2str(sigD(sD)) ' and noise is ' num2str(sigN(sN)) ];
        subplot(1,3,1);
        trplot(XActual(:,:), 'color', 'r', 'length', 0.08, 'thick', 1.5, 'text_opts', {'FontSize',10, 'Interpreter','latex'});
        hold on
        subplot(1,3,2);
        trplot(YActual(:,:), 'color', 'r', 'length', 0.08, 'thick', 1.5, 'text_opts', {'FontSize',10, 'Interpreter','latex'});
        hold on
        subplot(1,3,3);
        trplot(ZActual(:,:), 'color', 'r', 'length', 0.08, 'thick', 1.5, 'text_opts', {'FontSize',10, 'Interpreter','latex'});
      end
      %zg-

      for sim = 1:numTrials
        fprintf('Elapsed time at sr=%d, sD=%d, sN=%d, sim=%d: %0.3fsec\n', sr,sD,sN, sim,toc );

        %% Generate constant A1, free B1 and C1
        opt = 1;
        [A1, B1, C1] =  ABC_Generate(numData, opt, meanD, sigD(sD)*Cov, XActual, YActual, ZActual);
%          [ A1, B1, C1 ] = scrambleCorrspondence( scramRate(sr), A1, B1, C1 );
        [ A1, B1, C1 ] = jumbleCorrspondence( scramRate(sr), A1, B1, C1 );

        % compute norm of covariance of generated data, except for the constant matrices
        [m1, m2, m3] = size(B1); % dimensions should be the same for A, B, C
        [ ~, SigA1 ] = distibutionPropsMex_mex(reshape(A1, m1, m2*m3)); % SigA1 should be zero matrix
        [ ~, SigB1 ] = distibutionPropsMex_mex(reshape(B1, m1, m2*m3));
        [ ~, SigC1 ] = distibutionPropsMex_mex(reshape(C1, m1, m2*m3));
        normCov(sr,sD,sN,1,sim) = norm(SigB1);
        normCov(sr,sD,sN,2,sim) = norm(SigC1);
%          fprintf('norm of covariance of A1: %.4f, B1: %.4f, C1: %.4f\n', norm(SigA1), norm(SigB1), norm(SigC1) )

      %% Generate constant B3, free A3 and C3
        opt = 2;
        [A3, B3, C3] =  ABC_Generate(numData, opt, meanD, sigD(sD)*Cov, XActual, YActual, ZActual);
      % compute norm of covariance of generated data
        [ ~, SigA3 ] = distibutionPropsMex_mex(reshape(A3, m1, m2*m3));
        [ ~, SigC3 ] = distibutionPropsMex_mex(reshape(C3, m1, m2*m3));
        normCov(sr,sD,sN,3,sim) = norm(SigA3);
        normCov(sr,sD,sN,4,sim) = norm(SigC3);

        %          [ B3 , A3 , C3 ] = scrambleCorrspondence( scramRate(sr), B3 , A3 , C3 ) ;
        [ B3 , A3 , C3 ] = jumbleCorrspondence( scramRate(sr), B3 , A3 , C3 );

      %% Generate constant C2, free A2 and B2
        opt = 3;
        [A2 , B2 , C2] =  ABC_Generate(numData , opt , meanD , sigD(sD)*Cov , XActual , YActual , ZActual);
        % compute norm of covariance of generated data
        [ ~, SigA2 ] = distibutionPropsMex_mex(reshape(A2, m1 , m2*m3));
        [ ~, SigB2 ] = distibutionPropsMex_mex(reshape(B2, m1 , m2*m3));
        normCov(sr,sD,sN,5,sim) = norm(SigA2);
        normCov(sr,sD,sN,6,sim) = norm(SigB2);

%          [ C2 , A2 , B2 ] = scrambleCorrspondence( scramRate(sr), C2 , A2 , B2 );
        [ C2 , A2 , B2 ] = jumbleCorrspondence( scramRate(sr), C2 , A2 , B2 );

        %% ZG+ add noise to A, B, C
        if varywhat == 5 % only for Prob1 and Prob2 methods
          wt = weight(sN);
          A1n = A1; B1n = B1; C1n = C1;
          A2n = A2; B2n = B2; C2n = C2;
          A3n = A3; B3n = B3; C3n = C3;
        else
          wt = 1.5; % used only by Prob1 and Prob2 methods
          A1n = addNoise(A1 , sigN{sN}(1) , sigN{sN}(2) );
          B1n = addNoise(B1 , sigN{sN}(1) , sigN{sN}(2) );
          C1n = addNoise(C1 , sigN{sN}(1) , sigN{sN}(2) );

          A2n = addNoise(A2 , sigN{sN}(1) , sigN{sN}(2) );
          B2n = addNoise(B2 , sigN{sN}(1) , sigN{sN}(2) );
          C2n = addNoise(C2 , sigN{sN}(1) , sigN{sN}(2) );

          A3n = addNoise(A3 , sigN{sN}(1) , sigN{sN}(2) );
          B3n = addNoise(B3 , sigN{sN}(1) , sigN{sN}(2) );
          C3n = addNoise(C3 , sigN{sN}(1) , sigN{sN}(2) );
        end

        %% run each method depending on demand
        if methods2run(1) == 1 % i.e. run Prob1
          [X_final_1 , Y_final_1 , Z_final_1] = axbyczProb1(A1n(:,:,1), B1n, C1n, A2n, B2n, C2n(:,:,1), wt);
          Err1(sr,sD,sN,:,sim) = rottran_error(X_final_1,Y_final_1,Z_final_1,XActual, YActual,ZActual);
        end

        if methods2run(2) == 1 % i.e. run Prob2
          [X_final_2 , Y_final_2 , Z_final_2] = axbyczProb2(A1n(:,:,1), B1n, C1n, A2n, B2n, C2n(:,:,1), A3n, B3n(:,:,1), C3n, wt);
          Err2(sr,sD,sN,:,sim) = rottran_error(X_final_2,Y_final_2,Z_final_2,XActual, YActual,ZActual);
        end

        % stack all relevant data for non-probablisitic methods only (needed because
        % input arguments for the methods were coded differently from the probabilistic methods)
        A_perm = cat(3, A1n, A2n, A3n);
        B_perm = cat(3, B1n, B2n, B3n);
        C_perm = cat(3, C1n, C2n, C3n);

        if methods2run(3) == 1 % i.e. run Wu
%            [X_wang , Y_wang , Z_wang ] = Wang2014_AXBYCZ( A_perm, B_perm, C_perm, XActual, YActual, ZActual);
          [X_wang , Y_wang , Z_wang ] = Wu2016_AXBYCZ( A_perm, B_perm, C_perm );
          ErrWang(sr,sD,sN,:,sim) = rottran_error(X_wang,Y_wang,Z_wang, XActual, YActual,ZActual);
        end

        if methods2run(4) == 1 % i.e. run DK
          [X_DK ,   Y_DK ,   Z_DK ] = Yan_AXBYCZ_DK( A1n(:,:,1), B1n, C1n, A2n, B2n, C2n(:,:,1) );
          ErrDK(sr,sD,sN,:,sim) = rottran_error(X_DK,Y_DK,Z_DK, XActual,YActual, ZActual);
        end

        if methods2run(5) == 1 % i.e. run PN - uses perturbation of actual value not
```

```
  random initial estimates
%          [X_PN,   Y_PN,   Z_PN ] = Yan_AXBYCZ_PN( A_perm, B_perm, C_perm, XActual, YActual, ZActual);
%          ErrPN(sr,sD,sN,:,sim) = rottran_error(X_PN,Y_PN,Z_PN, XActual,YActual,ZActual);

         Err_random_initial = zeros(10,6); % store errors over 10 sub-trials
         for kk=1:10 % run 10 times with different initial guesses each time and get one with min err
           [X_PN,   Y_PN,   Z_PN ] = Yan_AXBYCZ_PN( A_perm, B_perm, C_perm, XActual, YActual, ZActual);
           Err_random_initial(kk,:) = rottran_error(X_PN,Y_PN,Z_PN, XActual,YActual,ZActual);
         end
         ErrPN(sr,sD,sN,:,sim) = min(Err_random_initial,[],1);
       end

       %% ------ plot X and Y and Z as 3D transformations------
       if tfplots
           subplot(1,3,1);
           if methods2run(1) == 1
             hold on
             trplot(X_final_1,'color','k', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(2) == 1
             hold on
             trplot(X_final_2,'color','g', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(3) == 1
             hold on
             trplot(X_wang,'color','b', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(4) == 1
             hold on
             trplot(X_DK,'color','m', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(5) == 1
             hold on
             trplot(X_PN,'color','c', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
             text( X_PN(1,4),X_PN(2,4),X_PN(3,4), num2str(sim) ,'Color','c', 'FontSize',14, 'FontWeight', 'bold');
           end
           axis auto
           title('Estimated X')

           hold on
           subplot(1,3,2);
           if methods2run(1) == 1
             hold on
             trplot(Y_final_1(:,:),'color','k', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(2) == 1
             hold on
             trplot(Y_final_2(:,:),'color','g', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(3) == 1
             hold on
             trplot(Y_wang(:,:),'color','b', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(4) == 1
             hold on
             trplot(Y_DK(:,:),'color','m', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
           end
           if methods2run(5) == 1
             hold on
             trplot(Y_PN(:,:),'color','c', 'text_opts', {'FontSize',10, 'Interpreter','latex'});
             text( Y_PN(1,4),Y_PN(2,4),Y_PN(3,4), num2str(sim) ,'Color','c', '
    FontSize',14, 'FontWeight', 'bold');
            end
            axis auto
            title('Estimated Y')

            hold on
            subplot(1,3,3);
            if methods2run(1) == 1
              hold on
              trplot(Z_final_1(:,:),'color','k', 'text_opts', {'FontSize',10, '
    Interpreter','latex'});
            end
            if methods2run(2) == 1
              hold on
              trplot(Z_final_2(:,:),'color','g', 'text_opts', {'FontSize',10, '
    Interpreter','latex'});
            end
            if methods2run(3) == 1
              hold on
              trplot(Z_wang(:,:),'color','b', 'text_opts', {'FontSize',10, '
    Interpreter','latex'});
            end
            if methods2run(4) == 1
              hold on
              trplot(Z_DK(:,:),'color','m', 'text_opts', {'FontSize',10, 'Interpreter
    ','latex'});
            end
            if methods2run(5) == 1
              hold on
              trplot(Z_PN(:,:),'color','c', 'text_opts', {'FontSize',10, 'Interpreter
    ','latex'});
              text( Z_PN(1,4),Z_PN(2,4),Z_PN(3,4), num2str(sim) ,'Color','c', '
    FontSize',14, 'FontWeight', 'bold');
            end
            axis auto
            title('Estimated Z')
        end
      end %for varying number of trials
    end %for varying std dev of noise
  end %for varying std dev of data
end % for varying scrambling rate
toc
fprintf('---------------------------------------------------------\n')

% compute norm of covariances to see if << 1
norm_Avg = mean(normCov, 5);
norm_Avg = reshape(norm_Avg, [length(scramRate)*length(sigD)*length(sigN) 6]);
normA = (norm_Avg(:,3) + norm_Avg(:,5)) ./ 2
normB = (norm_Avg(:,1) + norm_Avg(:,6)) ./ 2
normC = (norm_Avg(:,2) + norm_Avg(:,4)) ./ 2


%% compute averages of the errors with and without outliers
% after reshape, the row ordering is varying over dSig first then nSig
if methods2run(1) == 1 % i.e. run Prob1
  if removeOutliers
    Err1_Avg = meanWithoutOutliers(Err1);
  else % includes outliers
    Err1_Avg = mean(Err1, 5);
  end
  Err1_Avg = reshape(Err1_Avg, [length(scramRate)*length(sigD)*length(sigN) 6])
end

if methods2run(2) == 1 % i.e. run Prob2
  if removeOutliers
    Err2_Avg = meanWithoutOutliers(Err2);
  else
    Err2_Avg = mean(Err2, 5);
  end
  Err2_Avg = reshape(Err2_Avg, [length(scramRate)*length(sigD)*length(sigN) 6])
end

if methods2run(3) == 1 % Wang/Wu method
  if removeOutliers
    ErrWu_Avg = meanWithoutOutliers(ErrWang);
  else
    ErrWu_Avg = mean(ErrWang, 5);
  end
  ErrWu_Avg = reshape(ErrWu_Avg, [length(scramRate)*length(sigD)*length(sigN) 6])
end

if methods2run(4) == 1
  if removeOutliers
    ErrDK_Avg = meanWithoutOutliers(ErrDK);
  else
    ErrDK_Avg = mean(ErrDK, 5);
  end
  ErrDK_Avg = reshape(ErrDK_Avg, [length(scramRate)*length(sigD)*length(sigN) 6])
end

if methods2run(5) == 1 % PN method not stable, might sometimes give the wrong result
%   ErrPN
  if removeOutliers
%     figure, boxplot( permute(ErrPN,  [ 5 4 1 2 3]) );
    ErrPN_Avg = meanWithoutOutliers(ErrPN);
  else
    ErrPN_Avg = mean(ErrPN, 5);
  end
  ErrPN_Avg = reshape(ErrPN_Avg, [length(scramRate)*length(sigD)*length(sigN) 6])
end
%zg-
% end

%% Plot graphs
if lineplot
  f1 = figure('Name', 'Errors for X,Y,Z rotation and translation separately');
%   set(f1,'units','normalized','outerposition',[0 0 1 0.5])
  set(f1,'units','normalized','outerposition',[0 0 0.5 1]) % make fig fill screen
    width but only 0.8 of screen height

  switch varywhat
    case 1 % vary scrambling rate, fix data std dev = 0.02 and noise std dev = 0
      x_axis = scramRate;
      x_label = '$\textrm{Scrambling rate}\ r\ \%$';
    case 2 % vary data std dev, fix scrambling rate = 0 and noise std dev = 0
      x_axis = sigD;
      x_label = '$\sigma_{\textrm{data}}$';
    case 3 % vary noise std dev, fix scrambling rate = 0 and data std dev = 0.02
      x_axis = 1:length(sigN);
      x_label = '$\theta_{\textrm{max}},\epsilon_{\textrm{max}}\textrm{ of noise}$';

    case 4 % vary noise std dev, fix scrambling rate = 10 and data std dev = 0.02
      x_axis = 1:length(sigN);
      x_label = '$\theta_{\textrm{max}},\epsilon_{\textrm{max}}\textrm{ of noise}$';
    case 5 % vary noise std dev, fix scrambling rate = 0 and data std dev = 0.02
      x_axis = weight;
      x_label = '$\textrm{weight}$';
  end

  labels = {'$R_X$', '$R_Y$', '$R_Z$', '$t_X$', '$t_Y$', '$t_Z$'};
  % use file exchange function "panel" instead of subplot in order to remove white
    space
  p = panel('no-manage-font');
  p.pack(3,2);
  p.margin = [20 33 8 15];  %left bottom right top (space for ticks and labels)

  for ii = 1:6
    [i,j] = ind2sub([3,2], ii);% swap indices because linear indexing moves down the
      row first then col
    p(i,j).select();
    if methods2run(1) == 1
      plot(x_axis, Err1_Avg(:,ii), 'r-d', 'LineWidth', 2 ); %Prob1
%       semilogy(x_axis, Err1_Avg(:,ii), 'r-d', 'LineWidth', 1.1 );
      hold on
    end
    if methods2run(2) == 1
      plot(x_axis, Err2_Avg(:,ii), 'g-*', 'LineWidth', 1.5  ); %Prob2
      hold on
    end
    if methods2run(3) == 1
      plot(x_axis, ErrWu_Avg(:,ii), 'b-o', 'LineWidth', 0.7  ); %Wang
%       plot(x_axis, ErrWang_Avg(:,ii), 'b-o', 'LineWidth', 1.5  ); %Wang
      hold on
    end
    if methods2run(4) == 1
      plot(x_axis, ErrDK_Avg(:,ii), 'm-+', 'LineWidth', 1.5  ); %DK
      hold on
    end
    if methods2run(5) == 1
      plot(x_axis, ErrPN_Avg(:,ii), 'c-x', 'LineWidth', 1.5  ); %PN
      hold on
    end

    xlabel(x_label, 'Interpreter','latex', 'FontSize', 12);
    ax = gca;
    ax.XTick = x_axis;
    ax.YScale = 'log';
%     ylim = ax.YLim;
%     ax.YLim = [ylim(1), ylim(2)];
%     if log10(ylim(2)/ylim(1)) > 5
%       numYTicks = 1/4*(log10(ylim(2)/ylim(1))+1);
%     else
%       numYTicks = log10(ylim(2))-log10(ylim(1))+1;
%     end
%     ax.YTick = logspace(log10(ylim(1)),log10(ylim(2)), numYTicks );%add max of 5
    ticks
    ax.FontSize = 12;

    if varywhat == 3 || varywhat == 4
      noise_param = { sprintf('%.2g\\newline%.2g', sigN{1}(1), sigN{1}(2))  };
      noise_param = [ noise_param, sprintf('%.2g\\newline%.2g', sigN{2}(1), sigN
    {2}(2) ) ];
      noise_param = [ noise_param, sprintf('%.2g\\newline%.2g', sigN{3}(1), sigN
    {3}(2) ) ];
      noise_param = [ noise_param, sprintf('%.2g\\newline%.2g', sigN{4}(1), sigN
    {4}(2) ) ];
      noise_param = [ noise_param, sprintf('%.2g%c\\newline%.2gmm', sigN{5}(1), char
    (176), sigN{5}(2)) ];

      ax.XTickLabel = noise_param;
      ax.TickLabelInterpreter = 'tex';
    end

%     y_label = sprintf ('$\\textrm{error in }%s$', labels{ii} );
%     ylabel('Error','FontSize',14,'Interpreter','latex');

    title(sprintf ('Error in %s', labels{ii} ), 'FontSize',15,'Interpreter','latex','
    FontWeight','bold')

    if ii == 6 % only show legend on last plot (i.e. for tZ)
      lgd = {};
      for jj = 1:numel(methodNames)
        if methods2run(jj) == 1
          lgd = [lgd{:}  methodNames(jj) ];
        end
      end
      legend(lgd, 'FontSize',10, 'Location','best');
    end
  end
end
```

Listing A.1: Main program for the simulations

```
function [ X, Y, Z ] = Wu2016_AXBYCZ( A, B, C )
% Implements the algorithm in Wu et al (2016) - same as Wang2014 except
% that it uses a closed form solution for obtaining initial estimate.
%   Solves for X, Y, Z in the matrix equation AXB=YCZ given A, B, C

% Input: A, B, C are 4 x 4 x n homogeneous matrices
%        Xact, Yact, Zact are used to get a good initial estimate
% Output: X, Y, Z are 4 x 4 homogeneous matrices

num = size(A, 3); % number of measurements

%% Get rotation and translation components of A, B, C
RA = A(1:3, 1:3, :); % 3x3xnum
RB = B(1:3, 1:3, :);
RC = C(1:3, 1:3, :);
TA = A(1:3, 4, :);  % 3x1xnum
TB = B(1:3, 4, :);
```

```
TC = C(1:3, 4, :);


%% ============ Solve for RX, RY, RZ first ==============
% Closed form solution to get an initial estimate of rotations from a subset of data
[RX_init, RY_init, RZ_init] = Wu2016_closedForm( RA, RB, RC );

% figure
% trplot(RY_init, 'color', 'b');
%     hold on
%     trplot(Yact(1:3,1:3), 'color', 'r');

% fprintf('Err in initial guess: RX = %.5f, RY = %.5f, RZ = %.5f\n',...
%   roterror( RX_init, Xact(1:3,1:3) ), ...
%   roterror( RY_init, Yact(1:3,1:3) ),...
%   roterror( RZ_init, Zact(1:3,1:3) ) );

% Iterate until norm of delR = [delRX; delRY; delRZ] falls below a predefined
  threshold
delR = 10000 * ones(9,1);  % use a large value initially
NumIterations = 0;

while norm(delR) > 10e-10

  q = zeros(num*9,1); % q_tilde in paper
  F = zeros(num*9,9); % F_tilde in paper

  for i = 1:num
    tmp1 = RX_init * RB(:,:,i);
    tmp2 = RY_init * RC(:,:,i) * RZ_init;
    qq = -RA(:,:,i) * tmp1 + tmp2;
    q( (i-1)*9+1:i*9 ) = [qq(:,1); qq(:,2); qq(:,3)]; % 9x1

    F11 = -RA(:,:,i) * so3_vec( tmp1(:,1) ); % 3x3
    F21 = -RA(:,:,i) * so3_vec( tmp1(:,2) );
    F31 = -RA(:,:,i) * so3_vec( tmp1(:,3) );
    F12 = so3_vec( tmp2(:,1) );
    F22 = so3_vec( tmp2(:,2) );
    F32 = so3_vec( tmp2(:,3) );
    F13 = RY_init * RC(:,:,i) * so3_vec( RZ_init(:,1) );
    F23 = RY_init * RC(:,:,i) * so3_vec( RZ_init(:,2) );
    F33 = RY_init * RC(:,:,i) * so3_vec( RZ_init(:,3) );
    F( (i-1)*9+1:i*9, : ) = [ F11 F12 F13;
                              F21 F22 F23;
                              F31 F32 F33 ];
  end

  delR = (F'*F) \ F' * q;  % = inv(F'F)F'q
  RX_init = expm( so3_vec(delR(1:3)) ) * RX_init;
  RY_init = expm( so3_vec(delR(4:6)) ) * RY_init;
  RZ_init = expm( so3_vec(delR(7:9)) ) * RZ_init;

  NumIterations = NumIterations+1;
end

% fprintf('Number of iterations to converge to <10e-10: %d\n', NumIterations);

%% ============ Solve for TX, TY, TZ next ==============
J = zeros(3*num, 9); % J_tilde
p = zeros(3*num, 1); % p_tilde

for i = 1:num
  J( ((i-1)*3+1):(i*3), : ) = [ RA(:,:,i)  -eye(3)  -RY_init * RC(:,:,i) ];
  p( ((i-1)*3+1):(i*3) ) = -TA(:,:,i) - RA(:,:,i)*RX_init*TB(:,:,i) + RY_init*TC(:,:,
    i);
end

translation = (J'*J) \ J' * p;
tX = translation(1:3);
tY = translation(4:6);
tZ = translation(7:9);

%% Form the homogeneous matrices for X, Y, Z
X = zeros(4); Y = zeros(4); Z = zeros(4);
X(4,4) = 1;   Y(4,4) = 1;   Z(4,4) = 1;
X(1:3,1:3) = RX_init;
Y(1:3,1:3) = RY_init;
```

```
Z(1:3,1:3) = RZ_init;
X(1:3,4) = tX;
Y(1:3,4) = tY;
Z(1:3,4) = tZ;
end

%% Returns estimate of RX,RY,RZ using closed form solution in Wu et al
% Only used for getting initial estimate of data
function [RX, RY, RZ] = Wu2016_closedForm( RA, RB, RC )
% RA,RB,RC are 3x3xn rotation matrices.

% use only the first 10 sets or 10% of data whichever is lower
n = min( floor(0.1 * size(RA,3)), 10 );

% randomly sample n sets - don't take first n sets because RA is fixed in those sets
[RA_sample, idx] = datasample(RA, n, 3, 'Replace', false);
qA = Quaternion( RA_sample );  % datasample() requires Statistics toolbox
qB = Quaternion( RB(:,:,idx) );  % use the corresponding data that was sampled from
    RA
qC = Quaternion( RC(:,:,idx) );

W_ABC_plus = zeros( 4*n, 20 );
W_ABC_minus = zeros( 4*n, 20 );
for i=1:n % stack all the measurements
  [ LQ_A, ~ ] = quaternion_matrix(qA(i).double());
  [ ~, RQ_B ] = quaternion_matrix(qB(i).double());

  c = qC(i).double();
  W_C = [ c(1) -c(2) -c(3) -c(4) -c(2) -c(1) c(4) -c(3) -c(3) -c(4) -c(1) c(2) -c(4)
    c(3) -c(2) -c(1); % 4x16
          c(2) c(1) -c(4) c(3) c(1) -c(2) -c(3) -c(4) c(4) -c(3) c(2) c(1) -c(3) -c
    (4) -c(1) c(2);
          c(3) c(4) c(1) -c(2) -c(4) c(3) -c(2) -c(1) c(1) -c(2) -c(3) -c(4) c(2) c
    (1) -c(4) c(3);
          c(4) -c(3) c(2) c(1) c(3) c(4) c(1) -c(2) -c(2) -c(1) c(4) -c(3) c(1) -c(2)
     -c(3) -c(4) ];
  W_ABC_plus( ((i-1)*4+1):(4*i), : ) = [ RQ_B * LQ_A , W_C ]; % 4x20
  W_ABC_minus( ((i-1)*4+1):(4*i), : ) = [ RQ_B * LQ_A , -W_C ]; % 4x20
end

%return the smallest eigenvalue and the corresponding eigenvector from a 20x20
    symmetric matrix
% [ Vplus, dplus ] = eigs( W_ABC_plus'*W_ABC_plus, 1,'sm'); %using sparse version
% [ Vminus, dminus ] = eigs( W_ABC_minus'*W_ABC_minus, 1,'sm');

% might be better to use non-sparse version of eig and sort the e-values manually
[ Vplus, dplus ] = eig( W_ABC_plus'*W_ABC_plus );
[ dplus, I ] = sort( diag(dplus) );
Vplus = Vplus(:,I);
[ Vminus, dminus ] = eig( W_ABC_minus'*W_ABC_minus );
[ dminus, I ] = sort( diag(dminus) );
Vminus = Vminus(:,I);

if dplus(1) < dminus(1) % choose the one whose eigenvalue is smaller
  V_XYZ = Vplus; % 20x1
else
  V_XYZ = Vminus;
end

qX = Quaternion( V_XYZ(1:4)/norm(V_XYZ(1:4)) );
RX = qX.R;

V_XY = reshape( V_XYZ(5:20)/norm(V_XYZ(5:20)), 4,4 ); % 4x4
y0 = sqrt( V_XY(:,1)'*V_XY(:,1) );
y1 = sqrt( V_XY(:,2)'*V_XY(:,2) );
y2 = sqrt( V_XY(:,3)'*V_XY(:,3) );
y3 = sqrt( V_XY(:,4)'*V_XY(:,4) );
qY = Quaternion( [y0 y1 y2 y3] );
RY = qY.R;

z0 = sqrt( V_XY(1,:)*V_XY(1,:)' );
z1 = sqrt( V_XY(2,:)*V_XY(2,:)' );
z2 = sqrt( V_XY(3,:)*V_XY(3,:)' );
z3 = sqrt( V_XY(4,:)*V_XY(4,:)' );
qZ = Quaternion( [z0 z1 z2 z3] );
RZ = qZ.R;

```

```
end

%% forms a 4x4 matrix matrix representation of a quaternion
function [ LQ, RQ ] = quaternion_matrix(q)
% q(0) is the scalar part, [q(1) q(2) q(3)] are the vector part
  LQ = [  q(1) -q(2) -q(3) -q(4);
          q(2)  q(1) -q(4)  q(3);
          q(3)  q(4)  q(1) -q(2);
          q(4) -q(3)  q(2)  q(1)  ];

  RQ = [  q(1) -q(2) -q(3) -q(4);
          q(2)  q(1)  q(4) -q(3);
          q(3) -q(4)  q(1)  q(2);
          q(4)  q(3) -q(2)  q(1)  ];
end
```

Listing A.2: MATLAB implementation of Wu et al (2016)

```
function [ X, Y, Z ] = Yan_AXBYCZ_DK( A1, B1, C1, A2, B2, C2 )
% Implements the D-K algorithm in Yan et al (2015)
%   Solves for X, Y, Z in the matrix equation AXB=YCZ given A, B, C

% Input: A1 and C2 are 4x4 homogeneous matrices since they are fixed
%        B1,C1, A2,B2 are 4 x 4 x n homogeneous matrices
% Output: X, Y, Z are 4 x 4 homogeneous matrices

num = size(B1, 3); % number of measurements
% fprintf('Num of data in Yan_AXBYCZ_DK: %d\n', 2*num);

%% Solve AX=YB type of equations using Li(2010) method of Kronecker product
[Z, Xt] = Li_AXYB_kron( C1, B1 ); % fixing A
% [Z, Xt] = li(C1,B1); % use shah's implementation
Binv = zeros(size(B2)); % 4x4xnum
for i=1:num
  Binv(:,:,i) = inv(B2(:,:,i)); % calc the inv of B b4 passing to solver
end
[X, Zt] = Li_AXYB_kron( A2, Binv ); % fixing C
% [X, Zt] = li(A2,Binv); % use shah's implementation

%% Enforce orthogonality on rotational part of Xt and Z, X and Zt
%  by finding the nearest orthogonal matrix using SVD
[U,~,V] = svd(Xt(1:3,1:3));
Xt(1:3,1:3) = U*V';
[U,~,V] = svd(X(1:3,1:3));
X(1:3,1:3) = U*V';
[U,~,V] = svd(Zt(1:3,1:3));
Zt(1:3,1:3) = U*V';
[U,~,V] = svd(Z(1:3,1:3));
Z(1:3,1:3) = U*V';

%% calc the two possibilities of Y and choose the one with the smallest
% error
Y1 = A1 * X / Xt;
Y2 = Zt / Z / C2;

Err1 = norm( A1*X*B1(:,:,1) - Y1*C1(:,:,i)*Z, 'fro' );
Err2 = norm( A2(:,:,1)*X*B2(:,:,1) - Y2*C2*Z, 'fro' );
if ( Err1 < Err2 )
  Y = Y1;
else
  Y = Y2;
end
end

%% implements the Kronecker product method in Li et al (2010) paper
function [X, Y] = Li_AXYB_kron( A, B )

num = size(A, 3); % number of measurements

RA = A(1:3, 1:3, :); % 3x3xnum
RB = B(1:3, 1:3, :);
TA = A(1:3, 4, :);  % 3x1xnum
TB = B(1:3, 4, :);

```

```
K = zeros(12*num, 24);
t = zeros(12*num, 1);
for i = 1:num
  K( (i-1)*12+1:(i-1)*12+9, 1:9 ) = kron( RA(:,:,i), eye(3) );
  K( (i-1)*12+1:(i-1)*12+9, 10:18 ) = -kron( eye(3), RB(:,:,i)' );
  K( (i-1)*12+10:i*12, 10:18 ) = kron(eye(3), TB(:,:,i)' );
  K( (i-1)*12+10:i*12, 19:21 ) = -RA(:,:,i);
  K( (i-1)*12+10:i*12, 22:24 ) = eye(3);
  t( (i-1)*12+10:i*12 ) = TA(:,:,i);
end

% solve Kv = t using least squares
v = pinv(K) * t; % 24x1

X = zeros(4,4); X(4,4) = 1;
Y = zeros(4,4); Y(4,4) = 1;

% reform the X, Y homogeneous matrices from the vectorized versions
X(1:3,1:3) = reshape(v(1:9), 3, 3)'; %need transpose because reshape goes down col
    first then row
Y(1:3,1:3) = reshape(v(10:18), 3, 3)';
X(1:3, 4) = reshape(v(19:21), 3, 1);
Y(1:3, 4) = reshape(v(22:24), 3, 1);
end
```

Listing A.3: MATLAB implementation of the DK method in Yan et al (2015)

```
function [ X, Y, Z ] = Yan_AXBYCZ_PN( A, B, C, Xact, Yact, Zact)
% Implements the D-K algorithm in Yan et al (2015)
%   Solves for X, Y, Z in the matrix equation AXB=YCZ given A, B, C

% Input: A1 and C2 are 4x4 homogeneous matrices since they are fixed
%        B1,C1, A2,B2 are 4 x 4 x n homogeneous matrices
% Output: X, Y, Z are 4 x 4 homogeneous matrices

num = size(A, 3); % number of measurements
% fprintf('Num of data in Yan_AXBYCZ_PN: %d\n', num);

%% Get rotation and translation components of A, B, C
RA = A(1:3, 1:3, :); % 3x3xnum
RB = B(1:3, 1:3, :);
RC = C(1:3, 1:3, :);
TA = A(1:3, 4, :);  % 3x1xnum
TB = B(1:3, 4, :);
TC = C(1:3, 4, :);

%% Form error function - nested because it needs to access some variables above
function F = myfun(xyz)
  % xyz is a 21x1 vector of [qx qy qz tx ty tz]
  F = zeros(num*12,1);

  qx = Quaternion(xyz(1:4)).unit(); % normalize quaternion
  RX = qx.R;
  qy = Quaternion(xyz(5:8)).unit();
  RY = qy.R;
  qz = Quaternion(xyz(9:12)).unit();
  RZ = qz.R;
  TX = xyz(13:15);
  TY = xyz(16:18);
  TZ = xyz(19:21);

  for i = 1:num
    Rerr = RA(:,:,i)*RX*RB(:,:,i)-RY*RC(:,:,i)*RZ; % 3x3
    Terr = RA(:,:,i)*RX*TB(:,:,i)+RA(:,:,i)*TX+TA(:,:,i)...
            - RY*RC(:,:,i)*TZ-RY*TC(:,:,i)-TY; % 3x1
    F( (i-1)*12+1 : (i-1)*12+9 ) = reshape(Rerr, 9, 1);
    F( (i-1)*12+10 : (i-1)*12+12 ) = Terr;
  end

end

%% get initial estimate of quaternions and translation
opt  = 2;  % determines how the initial estimate was determined

```

```matlab
if opt == 1
% a) Use perturbation of actual X, Y, Z
e = pi/5;
RX_init = expm( so3_vec(e*ones(3,1)) ) * Xact(1:3,1:3); % 3x3
RY_init = expm( so3_vec(e*ones(3,1)) ) * Yact(1:3,1:3);
RZ_init = expm( so3_vec(e*ones(3,1)) ) * Zact(1:3, 1:3);
TX_init = Xact(1:3, 4) + e*ones(3,1);  % 3x1xnum
TY_init = Yact(1:3, 4) + e*ones(3,1);
TZ_init = Zact(1:3, 4) + e*ones(3,1);

elseif opt == 2
% b) Use randomly generated X, Y, Z
M = zeros(6,1); %mean
Sig = eye(6)*2; %covariance
XActual = expm(se3_vec(mvg(M, Sig, 1)));
YActual = expm(se3_vec(mvg(M, Sig, 1)));
ZActual = expm(se3_vec(mvg(M, Sig, 1)));
RX_init = XActual(1:3,1:3); % 3x3
RY_init = YActual(1:3,1:3);
RZ_init = ZActual(1:3,1:3);
TX_init = XActual(1:3,4);  % 3x1xnum
TY_init = YActual(1:3,4);
TZ_init = ZActual(1:3,4);
%-------------------------
end
qx_init = Quaternion(RX_init).double();
qy_init = Quaternion(RY_init).double();
qz_init = Quaternion(RZ_init).double();
x0 = [qx_init'; qy_init'; qz_init'; TX_init; TY_init; TZ_init];

%% call lsqnonlin() using Levenberg-Marquardt algorithm
% default termination tolerance is 1e-6
options = optimoptions(@lsqnonlin,'Algorithm','levenberg-marquardt');
[res,resnorm,~,~,output] = lsqnonlin(@myfun,x0,[],[],options);

%% reform X, Y, Z from "res"
X = zeros(4); Y = zeros(4); Z = zeros(4);
X(4,4) = 1;   Y(4,4) = 1;   Z(4,4) = 1;

X(1:3,1:3) = Quaternion(res(1:4)).unit().R; %normalize b4 converting to rot matrix
Y(1:3,1:3) = Quaternion(res(5:8)).unit().R;
Z(1:3,1:3) = Quaternion(res(9:12)).unit().R;
X(1:3,4) = res(13:15); % translation components
Y(1:3,4) = res(16:18);
Z(1:3,4) = res(19:21);
end
```

Listing A.4: MATLAB implementation of the PN method in Yan et al (2015)

Bibliography

[1] Martin Kendal Ackerman, Andrew Cheng, and Gregory Chirikjian. “An information-theoretic approach to the correspondence-free $AX = XB$ sensor calibration problem”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4893–4899.

[2] Martin Kendal Ackerman and Gregory Chirikjian. “A Probabilistic Solution to the $AX = XB$ Problem: Sensor Calibration without Correspondence”. In: *Geometric Science of Information*. Springer, 2013, pp. 693–701. ISBN: 3642400191.

[3] Martin Kendal Ackerman et al. “Online ultrasound sensor calibration using gradient descent on the Euclidean Group”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4900–4905.

[4] Martin Kendal Ackerman et al. “Sensor calibration with unknown correspondence: Solving $AX = XB$ using Euclidean-group invariants”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1308–1313. ISBN: 2153-0858.

[5] Nicolas Andreff, Radu Horaud, and Bernard Espiau. "On-line hand-eye calibration". In: *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on.* IEEE, 1999, pp. 430–436. ISBN: 0769500625.

[6] Nicolas Andreff, Radu Horaud, and Bernard Espiau. "Robot hand-eye calibration using structure-from-motion". In: *The International Journal of Robotics Research* 20.3 (2001), pp. 228–248. ISSN: 0278-3649.

[7] Gregory S. Chirikjian. *Stochastic Models, Information Theory, and Lie Groups.* Applied and numerical harmonic analysis. Boston: Birkhäuser, 2009. ISBN: 9780817648022 (v. 1 acid-free paper) 081764802X (v. 1 acid-free paper) 9780817649432 (v. 2 acid-free paper) 9780817649449 (v. 2 ebook).

[8] Jack CK Chou and M Kamel. "Finding the position and orientation of a sensor on a robot manipulator using quaternions". In: *The International Journal of Robotics Research* 10.3 (1991), pp. 240–254. ISSN: 0278-3649.

[9] Konstantinos Daniilidis. "Hand-eye calibration using dual quaternions". In: *The International Journal of Robotics Research* 18.3 (1999), pp. 286–298. ISSN: 0278-3649.

[10] Fadi Dornaika and Radu Horaud. "Simultaneous robot-world and hand-eye calibration". In: *Robotics and Automation, IEEE Transactions on* 14.4 (1998), pp. 617–622. ISSN: 1042-296X.

[11] F. Ernst et al. “Non-orthogonal tool/flange and robot/world calibration”. In: *International Journal of Medical Robotics and Computer Assisted Surgery* 8.4 (2012), pp. 407–20. ISSN: 1478-596X (Electronic) 1478-5951 (Linking). DOI: `10.1002/rcs.1427`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/22508570`.

[12] Irene Fassi and Giovanni Legnani. “Hand to sensor calibration: A geometrical interpretation of the matrix equation $AX = XB$”. In: *Journal of Robotic Systems* 22.9 (2005), pp. 497–506. ISSN: 0741-2223 1097-4563. DOI: `10.1002/rob.20082`.

[13] Junhyoung Ha, Donghoon Kang, and Frank C Park. “A Stochastic Global Optimization Algorithm for the Two-Frame Sensor Calibration Problem”. In: *IEEE Transactions on Industrial Electronics* 63.4 (2016), pp. 2434–2446. ISSN: 0278-0046. DOI: `10.1109/TIE.2015.2505690`.

[14] Johann Heller, Didier Henrion, and Tomas Pajdla. “Hand-eye and robot-world calibration by global polynomial optimization”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 3157–3164.

[15] Robert Lee Hirsh, Guilherme Nelson DeSouza, and Aviriash C Kak. “An iterative approach to the hand-eye and base-world calibration problem”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on.* Vol. 3. IEEE, 2001, pp. 2171–2176. ISBN: 0780365763.

[16] Radu Horaud and Fadi Dornaika. “Hand-eye calibration”. In: *The international journal of robotics research* 14.3 (1995), pp. 195–210. ISSN: 0278-3649.

[17] Sin-Jung Kim et al. “Robot head-eye calibration using the minimum variance method”. In: *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1446–1451. ISBN: 1424493196.

[18] Aiguo Li, Lin Wang, and Defeng Wu. “Simultaneous robot-world and hand-eye calibration using dual-quaternions and Kronecker product”. In: *International Journal of the Physical Sciences* 5.10 (2010), pp. 1530–1536.

[19] Haiyuan Li et al. “Simultaneous Hand-eye and Robot-world Calibration by Solving the $AX = YB$ Problem without Correspondence”. In: *Robotics and Automation Letters, IEEE* 1.1 (2016), pp. 145–152. ISSN: 2377-3766. DOI: 10.1109/LRA.2015.2506663.

[20] Rong-hua Liang and Jian-fei Mao. “Hand-eye calibration with a new linear decomposition algorithm”. In: *Journal of Zhejiang University Science A* 9.10 (2008), pp. 1363–1368. ISSN: 1673-565X.

[21] Ying-Cherng Lu and Jack CK Chou. “Eight-space quaternion approach for robotic hand-eye calibration”. In: *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*. Vol. 4. IEEE, 1995, pp. 3316–3321. ISBN: 0780325591.

[22] Qianli Ma and Gregory Chirikjian. “Sensor Calibration, Modeling and Simulation”. Under review for publication.

[23] Qianli Ma, Zachariah Goh, and Gregory Chirikjian. “Probabilistic Approaches to the $AXB = YCZ$ Calibration Problem in Multi-Robot Systems”. In: *Proceedings of Robotics: Science and Systems*. 2016. DOI: `10.15607/RSS.2016.XII.003`.

[24] Qianli Ma, Haiyuan Li, and Gregory Chirikjian. “New Probabilistic Approaches to the $AX = XB$ Hand-eye Calibration without Correspondence”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 4365–4371. DOI: `10.1109/ICRA.2016.7487635`.

[25] Jianfei Mao, Xianping Huang, and Li Jiang. “A flexible solution to $AX = XB$ for robot hand-eye calibration”. In: *Proceedings of the 10th WSEAS international conference on Robotics, control and manufacturing technology*. World Scientific, Engineering Academy, and Society (WSEAS), 2010, pp. 118–122. ISBN: 9604741756.

[26] Frank C Park and Bryan J Martin. “Robot sensor calibration: solving $AX = XB$ on the Euclidean group”. In: *IEEE Transactions on Robotics and Automation (Institute of Electrical and Electronics Engineers);(United States)* 10.5 (1994).

[27] Thomas Ruland, Tomas Pajdla, and Lars Kruger. “Globally optimal hand-eye calibration”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1035–1042. ISBN: 1467312266.

[28] Mili Shah. “Solving the robot-world/hand-eye calibration problem using the kronecker product”. In: *Journal of Mechanisms and Robotics* 5.3 (2013), p. 031007. ISSN: 1942-4302.

[29] Mili Shah, Roger D Eastman, and Tsai Hong. “An overview of robot-sensor calibration methods for evaluation of perception systems”. In: *Proceedings of the Workshop on Performance Metrics for Intelligent Systems.* ACM, 2012, pp. 15–20. ISBN: 1450311261.

[30] Yiu Cheung Shiu and Shaheen Ahmad. “Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form $AX = XB$”. In: *Robotics and Automation, IEEE Transactions on* 5.1 (1989), pp. 16–29. ISSN: 1042-296X.

[31] Klaus H Strobl and Gerd Hirzinger. “Optimal hand-eye calibration”. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 4647–4653. ISBN: 1424402581.

[32] Roger Y Tsai and Reimar K Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration”. In: *Robotics and Automation, IEEE Transactions on* 5.3 (1989), pp. 345–358. ISSN: 1042-296X.

[33] Ching-Cheng Wang. “Extrinsic calibration of a vision sensor mounted on a robot”. In: *Robotics and Automation, IEEE Transactions on* 8.2 (1992), pp. 161–175. ISSN: 1042-296X.

[34] Jiaole Wang et al. “Towards simultaneous coordinate calibrations for cooperative multiple robots”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 410–415.

[35] Yunfeng Wang and Gregory S Chirikjian. “Nonparametric second-order theory of error propagation on motion groups”. In: *The International journal of robotics research* 27.11-12 (2008), pp. 1258–1273. ISSN: 0278-3649. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2843106/pdf/nihms-175123.pdf.

[36] Liao Wu et al. “Simultaneous Hand-Eye, Tool-Flange, and Robot-Robot Calibration for Comanipulation by Solving the $AXB = YCZ$ Problem”. In: *IEEE Transactions on Robotics* 32.2 (2016), pp. 413–428. ISSN: 1552-3098. DOI: 10.1109/TRO.2016.2530079.

[37] S. J. Yan, S. K. Ong, and A. Y. C. Nee. “Registration of a hybrid robot using the Degradation-Kronecker method and a purely nonlinear method”. In: *Robotica* FirstView (2015), pp. 1–12. ISSN: 1469-8668. DOI: doi:10.1017/S0263574715000338. URL: http://dx.doi.org/10.1017/S0263574715000338%20http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=9725702&fileId=S0263574715000338.

[38] Zijian Zhao. “Hand-eye calibration using convex optimization”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2947–2952. ISBN: 1612843867.

[39] Zijian Zhao and Yuncai Liu. "A hand–eye calibration algorithm based on screw motions". In: *Robotica* 27.02 (2009), pp. 217–223. ISSN: 1469-8668. DOI: doi:10.1017/S0263574708004608. URL: http://dx.doi.org/10.1017/S0263574708004608%20http://journals.cambridge.org/action/displayFulltext?type=1&fid=3754220&jid=ROB&volumeId=27&issueId=02&aid=3754212.

[40] Hanqi Zhuang and Zvi S Roth. "Comments on" Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX= XB"[with reply]". In: *Robotics and Automation, IEEE Transactions on* 7.6 (1991), pp. 877–878. ISSN: 1042-296X.

[41] Hanqi Zhuang, Zvi S Roth, and Raghavan Sudhakar. "Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form $AX = YB$". In: *Robotics and Automation, IEEE Transactions on* 10.4 (1994), pp. 549–554. ISSN: 1042-296X.

[42] Hanqi Zhuang and Yiu Cheung Shiu. "A noise-tolerant algorithm for robotic hand-eye calibration with or without sensor orientation measurement". In: *Systems, Man and Cybernetics, IEEE Transactions on* 23.4 (1993), pp. 1168–1175. ISSN: 0018-9472.

Vita

Zachariah Goh is currently awaiting the conferral of his M.S.E. in Robotics from the Johns Hopkins University. He received a B.Comp in Computer Engineering from the National University of Singapore in 2006. He then started working as an applications engineer at the Singapore branch of Adept Technology Inc., a manufacturer of robots for industrial automation from 2006 to 2014. During that time, he travelled throughout Asia-Pacific to solve the automation needs of various industries, including electronics assembly and food packaging. He also acted as a consultant for many customers by providing hardware and software advice for efficient, cost-effective and scalable robotic solutions in manufacturing.