Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Automatic self-calibration from self-observation and self-touch on a dual arm industrial manipulator

František Puciow

Supervisor: Mgr. Matěj Hoffmann, Ph.D.
Supervisor–specialist: Mgr. Karla Štěpánová, Ph.D.
Field of study: Robotics
Subfield: Robotic perception
June 2018

Acknowledgements

My gratitude goes to everyone who aided me with the creation of this thesis. Most of all to Karla Štěpánová, who co-supervised this thesis, introduced me to the field of optimization, answered my many questions about calibration and assisted me when I was unsure about the interpretation of my results. More of my thanks go to Matěj Hoffmann, who supervised this thesis, created the assignment of this thesis based on my previous experiences and helped me many times with editing of this thesis.

My thanks go also to Tomáš Pajdla, who advised me on the shape, appearance and operating principle of the self-touch end effectors, Martin Hoffmann, who created the CAD model of the end effectors and helped me with installing them on the robot, Zdeněk Straka, who captured footage of the robot during the experiment and Tomáš Báča, who managed the 3D print of the end effectors.

I also wish to thank to Libor Wágner and Vladimír Smutný for letting me experiment on the CloPeMa robot, answering my questions about its hardware and software, and also Tomáš Hejda for creating the LaTeX template used in this thesis. Finally I thank the CTU in Prague for being such a good *alma mater* and my family for their continuous support and encouragement.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 8. June 2018

Abstract

Accurate calibration is key for the performance of every robot. Traditional calibration procedures involve some form of external measuring apparatus and become impractical if the robot itself or the site where it is deployed change frequently—current trend in automation with shift from mass to small batch production. At the same time, advances in sensor technology make affordable but increasingly accurate devices such as cameras, RGB-D, and force/tactile sensors available, making it possible to perform automated self-contained calibration relying on redundant information in these sensory streams. In this work, we employ two industrial manipulators mounted on a common base, with Force/Torque sensors at the tool and additionally equip them with: (i) two external cameras mounted on the robot base, (ii) special end-effectors with fiducial markers. Driving the robot to contact ("self-touch") configurations relying on force feedback, we collect images from the cameras as well as encoder readings. Using non-linear least squares optimization, this dataset is then used to quantitatively compare the performance of kinematic calibration by employing different combinations of intersecting kinematic chains—either through self-observation (using the cameras) or self-touch, where the physical contact constraint is exploited.

Keywords: dual-arm robot, self-calibration, kinematic chains, machine perception, optimization

Supervisor: Mgr. Matěj Hoffmann, Ph.D.

Abstrakt

Přesná kalibrace je klíčový faktor určující chování robotu. Tradiční kalibrační procedury zahrnují různé formy přídavných měřicích zařízení a stávají se nepraktickými, pokud se robot nebo prostředí ve kterém je nasazen často mění. Současným trendem v automatizaci je posun od velkosériové k malosériové výrobě. Zároveň se díky pokroku v měřicí technologii stávají dostupnými pokročilé senzory, jako například kamery, RGB-D a silové senzory. Díky tomu je možné provádět automatizovanou sebekalibraci na základě redundantní informace poskytované těmito senzory. V této práci jsme využili dva průmyslové manipulátory upevněné na společné základně se silovými senzory na koncovém článku a vybavili je: (i) dvěma externími fotoaparáty upevněnými na společné základně manipulátorů, (ii) speciálními koncovými články s referenčními značkami. Po uvedení robotu do kontaktních ("sebedotykových") pozic pomocí silové zpětné vazby jsme sbírali data z obou fotoaparátů a kloubových enkodérů robotu. Použitím metod nelineární optimalizace nejmenších čtverců jsme kvantitativně porovnali výkony kinematické kalibrace při využití různých kombinací protínajících se kinematických řetězců — sebepozorování (pomocí fotoaparátů) nebo sebedotyk, kde je využito omezení dané fyzickým kontaktem.

Klíčová slova: dvouruký robot, sebekalibrace, kinematické řetězce, strojové vnímání, optimalizace

Překlad názvu: Automatická kalibrace pomocí sebedotyku a sebepozorování u dvojrukého průmyslového manipulátoru

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Robots have widespread applications in industry. They are used not only for manufacturing, but also for testing, sorting, packaging and many other tasks. One of the main requirements for an industrial robot is precision. Robot control is based on its kinematics, which requires accurate robot model to be known, including joint offsets. The model provides a relation between the joint configuration of the robot (displacements/rotations, rates, accelerations of individual joints) and kinematic configuration of the robot (positions, velocities, accelerations of individual links). The model may also include dynamic properties, i.e. relation between joint forces or moments, forces acting upon the robot links and change in momentum of the robot links. The control of the robot is based on the model. In this work, we focus on the kinematic parameters of the robot model.

Many sources of imprecision may cause a difference between the actual dimensions of the robot and the specifications. Influences such as imprecision of the manufacturing process, imprecision of the assembly or thermal expansion of the robot links cause the actual dimensions to differ. Also excessive stress during transport, deployment into production or using may cause deformations and change the robot dimensions.

Calibration refers to a practice where the robot dimensions are obtained using a set of measurements carried out on the particular robot. By having the kinematics work with calibrated dimensions, the precision of the robot is increased. Commonly used methods of calibration depend on high-precision-manufactured flat tables or very accurate sensors that are installed directly on the robot or in the robot workspace. These methods are known to provide desired results, but the time required to prepare the calibration may be rather long, and the rent or purchase of the required sensors may be rather costly.

Modern robots equipped with multitude of sensors intended for perception of their surroundings, for example "collaborative robots" that are intended to work alongside humans, may be calibrated using only their built-in sensors, saving time and financial resources. Furthermore, it becomes feasible to repeat the calibration in periodic intervals to amount for robot dimensions changing over time. This procedure is commonly known as self-calibration. In this thesis we evaluate the use of self-touch and self-observation on a dual arm manipulator using force sensors and photo cameras for robot self-calibration.

1.2 Calibration methods comparison

Kinematic calibration is a task of finding transformations between coordinate systems of objects. A typical kinematic calibration task is finding the transformations between neighboring links of a manipulator; other tasks include extrinsic calibration of a sensor—a task of finding the pose of a rigidly mounted sensor in the robot base reference frame. A similar concept is hand-eye calibration—a task of finding a transformation between a sensor mounted on a robot gripper and the gripper frame of reference.

Hollerbach et al. [1] consider kinematic calibration as a parameter estimation problem with parametric function that represents robot model. They recognize two distinct approaches to calibration: open-loop and closed-loop. Open-loop calibration is characterized by measurements of robot end effector in various poses, typically by some external metrology system. In contrary, closed-loop calibration is characterized by introduction of one or more physical constraints on end-effector position or orientation. Hollerbach et al. show that the same formulation of error equation can be used for both open-loop and closed-loop calibration problems. They propose Gauss-Newton algorithm for solving the calibration task, with a note that this method requires a good initial estimate of the parameters.

An interesting method of hand-eye calibration is shown by Kukelová, Heller and Pajdla [2], where they obtain an exact solution of the problem using

Gröbner basis method over a minimal dataset and then use the result as an initial estimate for optimization algorithm over a larger dataset. Thanks to this the method is universally applicable (global), without the requirement of initial parameters to be known.

Recently, with robot platforms becoming more complex and more sensorized, it becomes necessary to calibrate several kinematic chains, but at the same time, several sensors may be employed to estimate the pose of different parts of the system. We may recognize pair-wise procedural (sequential) calibration, where a series of calibrations is carried out with single kinematic chain or sensor being calibrated at a time, and joint calibration, where all sensors or kinematic chains are calibrated using unified error function and optimization. Birbach et al. [3] evaluate the advantages of joint calibration on multiple sensors (a pair of cameras, a Microsoft Kinect RGBD sensor and an inertial measurement unit) on a humanoid robot. With this method, no external measuring apparatus is needed; instead, internal sensors on the robot are used for calibration. Birbach et al. formulate an error function as a weighted sum of squares over the errors of individual sensors and use Levenberg-Marquardt algorithm to optimize the intrinsic parameters of individual sensors and their position with relation to the robot head. They claim that joint calibration is more efficient than pair-wise procedural calibration, because in the case of pair-wise procedural calibration, inconsistencies in the obtained calibration results may occur, while joint calibration ensures consistent result.

Bennet et. al. [4] propose an algorithm for unified calibration of both the manipulators and sensors. They used Levenberg-Marquardt algorithm to solve a simulated optimization problem. Hersch et al. [5] show an interesting method that allows autonomous learning of robot body schema with very little prior information. The algorithm only knows the number of joints of every kinematic chain and learns the full structure by self-observation and proprioception (motor encoders) and in simulation also using tactile sensors.

1.3 Previous work

More closely related to this work, Roncone et al. [6] show the self-calibration of a humanoid robot iCub based on tactile perception using artificial skin. The robot uses a finger on one hand to touch a tactile skin on the other arm. They consider the point of touch as a base frame and interpret the kinematic chains from this point to torso and from torso to the touching finger as one long kinematic chain. Their calibration optimizes error function, which is the difference of the finger tip position from forward kinematics and the true touching position measured by tactile skin. Since the tactile skin provides a measurement of the finger position, Roncone et al. consider their method as *open-loop calibration*. Similarly, Li et al. [7] employed a dual KUKA arm setup with a sensorized "finger" and a tactile array on the other manipulator.

The CloPeMa robot setup used in this work (see Ch. 2) has been previously calibrated using two different methods: redundant parallel calibration and measuring machine (RedCaM) by Beneš et. al. [8][9] and Leica laser tracker. Petrík and Smutný [10] review the precision of these methods using linear probe sensor. Based on a dataset of 43 different poses with touching end effectors, they calculate the mean error as 0.67 (range 2.92) mm on CAD model, 0.54 (range 2.55) mm on Leica based calibration and 2.45 (range 9.92) mm on RedCaM based calibration.

Štěpánová and Hoffmann [11] provide a synthetic experiment with self-calibration of a simulated robot based on a real humanoid robot, iCub. The robot has a given point on each of its palms touching, thus closing a kinematic loop, while also observing the points with both of its eyes, providing measurements. The calibration problem is approached as an open-loop calibration, considering the touching constraint as a measurement of relative position of the two points that always reports zero displacement. Štěpánová and Hoffmann evaluate the quality of parameters estimation using joint calibration of multiple kinematic chains while utilizing either only the contact information, only the cameras, or both—contact constraint and cameras. Then they compare their results to a pair-wise sequential calibration.

1.4 Contribution

In contrast to [6], where only "self-touch" is employed, and Birbach et al. [3] and many others that employ "self-observation", this thesis aims to assess the pros and cons of these approaches, thereby complementing the synthetic experiments of [11], where self-touch and self-observation on a simulated humanoid robot is used for self-calibration. For this purpose we modified a two-arm robotic platform previously used for cloth folding and small object sorting experiment. We developed and manufactured custom "self-touch end effectors" with fiducial markers, configured the robotic platform to operate with the end effectors and integrated two photo cameras into the system. We also created a high-level Python library for self-touch using force sensors to enable safe contact behavior and developed a graphical application for planning and execution of complex motions.

We developed a simple data collecting application and collected several datasets on the robot. We created a parametric model of the robot using Denavit-Hartenberg parameters in Matlab and an objective function for the calibration Optimization Toolbox. In the process, we formulated a DH-only description of the robot, without intermediate transformation methods used previously to describe the mounting of the manipulators on the base.

We also collected two datasets containing 82 and 100 poses on the robot, where every pose comprises a touching configuration. We tested the stability of our calibration using perturbed initial state, compared calibrations of all DH parameters and only joint offsets, and evaluated the difference between two-step procedural and one-step joint calibrations. We also compared the errors of our results to the errors of manufacture parameters and to the laser-based calibration reviewed by Petrík and Smutný [10].

Chapter 2

Setup overview

This project makes use of a platform created in the CloPeMa project (Clothes Perception and Manipulation) [12] and further developed during the RadioRoSo project (Radioactive Waste Robotic Sorter) [13]. CloPeMa was a 3 year research project aiming at advancing the state of the art in autonomous perception and manipulation of textiles [14]. The robot learned to manipulate, perceive and fold a variety of textiles. The operating-software is based on ROS and written in C++, Python and Java. The project concluded in 2015.

The robot (Fig. 2.1; [15]) developed in the CloPeMa project consists of two industrial manipulators installed on top of a robotic turntable, a control unit, a photo camera and two computers connected on a local network. The robot has been setup with a variety of components, such as Xtion RGBD (RGB + depth) cameras, photo cameras, force sensors or photometric close-range sensors. All the different parts of the robot system were integrated in and operated with Robot Operating System (ROS).

In this work, we used a special end effector covered with a regular pattern of visual markers to enable safe self-touch behavior and visual recognition of the end effector position.

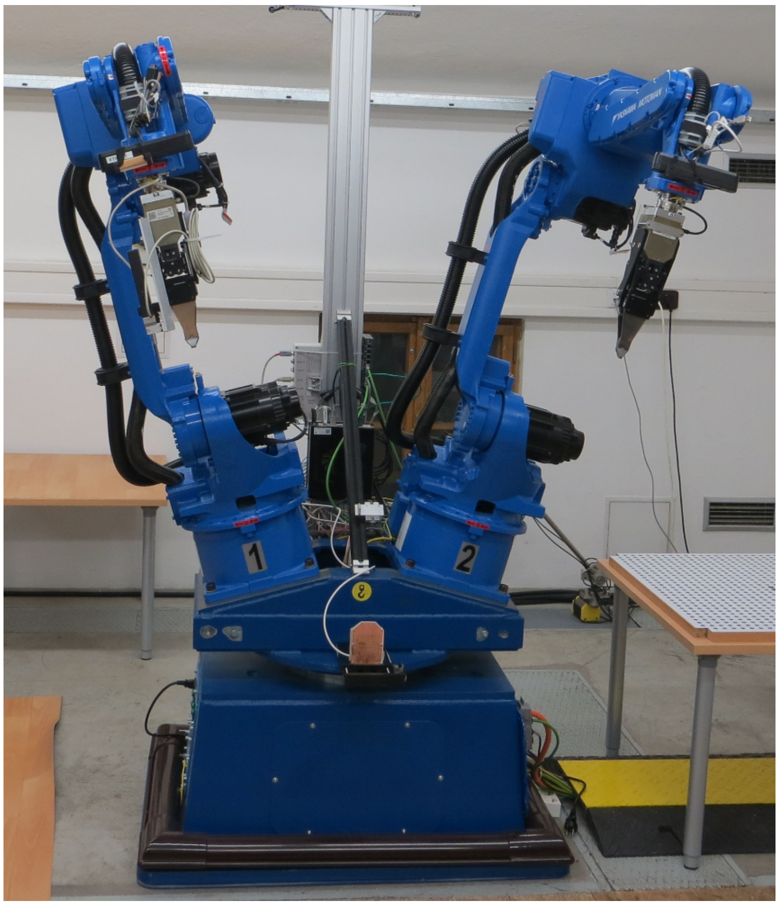


Figure 2.1: The turntable and manipulators assembly.

2.1 Hardware

2.1.1 Dual arm manipulator

The base of the unit is a Yaskawa R750 robotic turntable, which allows rotation of the mounting adapter carrying two manipulators around the vertical axis. On top of the turntable, there are two Yaskawa-Motoman MA1400 serial manipulators equipped with legacy two-fingered grippers developed for cloth folding and retooled for small cylindrical object grasping. Our self-touching end effector is attached over the gripper, to better define the contact surfaces and protect the fingers from excessive strain.

Both manipulators are equipped with ATI Industrial Automation Mini45 6-axis force/torque sensors, which are located between the last link of the

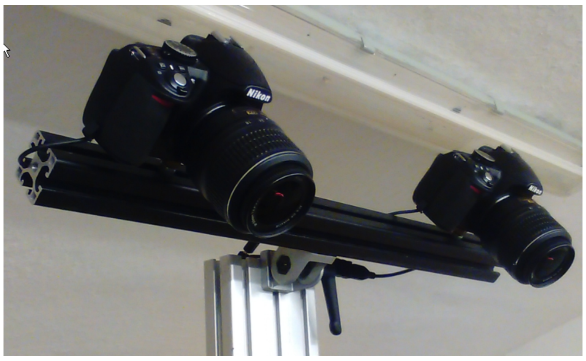


Figure 2.2: Setup of stereo cameras.

manipulator and the gripper. The sensors are connected to local network via ATI NetBox. During the experiment, we only used the force data to recognize contact at the end effector. The robot is powered and controlled via Motoman DX100 control unit, which can communicate with ordinary PC over LAN.

2.1.2 Cameras

Two Nikon D3100 cameras with Nikkor 18-55 AF-S DX VR lens were used in the experiment. They were attached in a side-by-side configuration on top of a vertical beam connected to the mounting adapter. The D3100 is a DSLR photo camera with APS-C CMOS imaging sensor with 14.2 MPx resolution (maximum image size 4608x3072 Px). The camera supports PTP (Picture Transfer Protocol), which is used to download the pictures taken by the cameras onto the PC. It is also equipped with 1/4-20 UNC thread, over which both cameras are attached to the robot.

2.1.3 Custom end effector

We developed a simple end effector to facilitate safe self-touch actions and visual self-observation. Two identical end effectors were attached to the tip of each manipulator 2.4. The requirements were:

- convex shape, so that the effectors cannot grapple with each other
- a shape with well defined contact surfaces, so that the contact point can be derived without tactile sensing and under positional uncertainty

- sufficient amount of flat faces so that visual markers can be attached onto it

We decided for a body of the shape of truncated icosahedron (like a soccer ball) with spherical tiles in place of the pentagonal faces and with markers glued onto the hexagonal faces. Each of the hexagonal faces has a square grove with side of length 20 mm. The square is concentric with the face and defines the position for the marker to be glued to. Every square has a unique number that defines its position with relation to the center of the icosahedron. The number is written next to the square grove. Figure 2.3 shows a colored CAD model of the end effector.

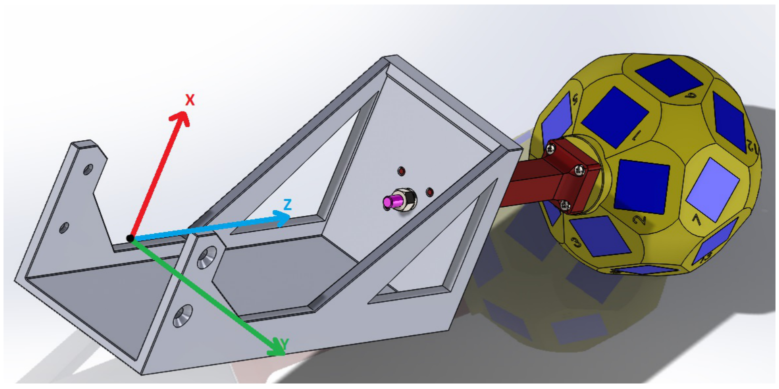


Figure 2.3: Detail of the end effector, squares for marker attachment are colored blue

Having 20 evenly spaced markers should ensure that at least 3 of them are always seen by each camera and recognized by the program (unless the view of the icosahedron is occluded by another part of the robot). The number 3 was chosen because 3 known points is a minimum number required to unambiguously discern the position using one camera.

The icosahedron is attached over the gripper with means of a bracket fastened to the gripper motor using four screws. A thin beam connects the icosahedron to the bracket. The custom end effectors were made using additive manufacturing (i.e. 3D print). We used two different materials to compare their properties and the difference in quality of the print. The blue icosahedron was made out of PLA, while the black one was made out of ABS plastic.

Models based on our requirements and after consultations with Prof. Tomáš Pajdla were supplied by Bc. Martin Hoffmann, who also carried out the assembly of the end effectors and their attachment to the robot. The 3D print was managed by Ing. Tomáš Báča.

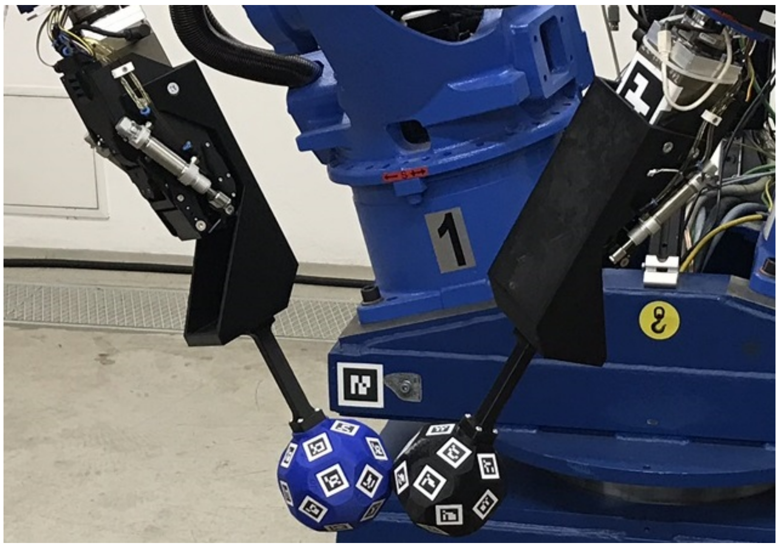


Figure 2.4: Custom end-effectors in contact.

Touching configurations

The center of the curvature of the tiles lies in the center of the icosahedron, such that when the two end effectors come into contact with the spherical tiles touching, the point of contact lies exactly halfway between the two effector centers, up to a precision given by the manufacturing of the effectors. For simplicity, in this thesis we only use the cases with two spherical tiles touching.

Touching of a spherical tile and a flat face is also possible and provides additional information regarding the flat face's orientation. In this case, the point of contact is the perpendicular projection of the center of the spherical tile to the plane of the flat face. These configurations were not used in this thesis, but might be used in future work.

The information obtained from the collision of two flat faces is not easy to evaluate. The point of contact is located on the edge of one of the faces, unless the faces are perfectly aligned, which cannot be easily assured. Because the face doesn't have a well defined outline, there is a complicated dependence between the pose of the end effectors and the point of contact. These configurations were not used and shall not be ever used with the current end effector.

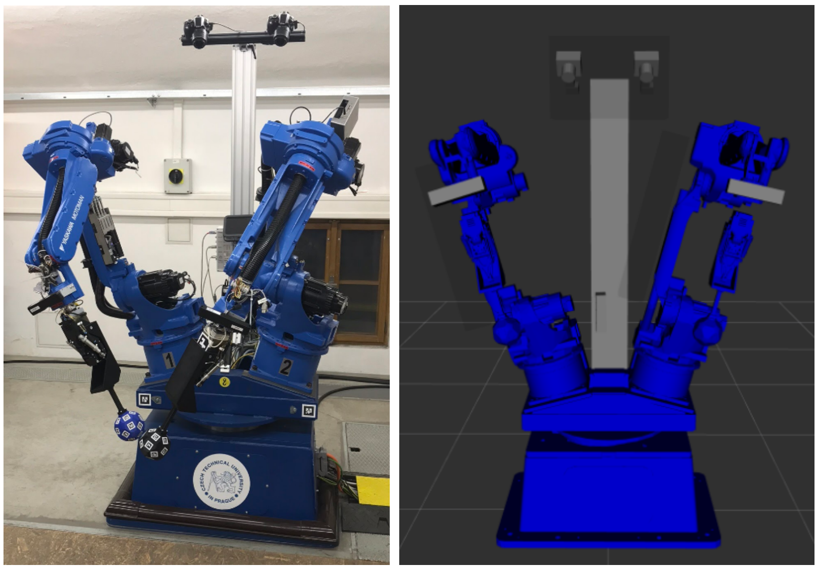


Figure 2.5: Setup of the whole dual arm robot for the self-touch experiment: real robot (left) and RViz visualisation of the robot (right).

Mechanical properties

Mechanical properties of the custom end effector are of concern. The end effector shall be sufficiently rigid, so that its deformation under the force applied during contact is significantly lower than the precision of the most precise localization technique used, which in this case is camera tracking. To this end, we reinforced the beam with a threaded rod.

At the same time, the end effector should not be completely rigid so that in case of a program failure during contact, it will absorb the damage and protect other parts of the manipulator from damage. It is desirable for the end effector to break and not to deform, because a slight deformation may be hard to discern and may cause inconsistency of robot parameters during the collection of datasets. Damping properties of the end effector are marginally important. Due to its shape and the fact that robot gearboxes cause a fair amount of vibration, the end effector is prone to resonance along the longitudinal axis.

The rigidity was observed to be acceptable. Several touches were performed and no visible deformation was noticed. The force sensors reported peak force of 5 Newtons during this test. Resonance was observed during phases of rapid motion of the robot, but never during the touching approach. The robot was programmed to stop for a few seconds before the touching approach to let the resonance fade away, and then to move very slowly, so that the resonance does not build up again. As a result, little to no resonance was measured by the force sensors during the touching approach. See figure 4.6 to see a typical course of forces measured during the touching approach.

2.2 Software

The robot is controlled from either one of two personal computers connected to the local network. The computers are running Ubuntu 16.04 with ROS Kinetic 1.12.7 and a set of ROS modules commonly named CloPeMa Workspace (found on [12]).

2.2.1 ROS

Robot Operating System (ROS)[16], is a widely used framework for robotic software. ROS is meant to provide collaborative environment for development of robotic projects. It is distributed via a system of packages.

Nodes are executables running under ROS. Nodes can publish to a Topic, sending messages of predefined structure to other nodes which subscribe to the same topic. Nodes may also request a service (act as a client) from other nodes which provide the given service (act as a server). The client sends a request message and waits until it receives a response message from the server.

Actionlib

Actionlib [17] is an alternative to the services interface, which in addition to the goal (request) and result (response), allows the client to monitor the state of the server via status message, the state of its request being processed through feedback messages, and preempt the request by sending a cancel message. We used an Actionlib server for interfacing with the cameras.

MoveIt!

MoveIt! [18] is an universal robotic software for motion planning, kinematics, robot control and manipulation. MoveIt! works under ROS. It was integrated within the robot software setup, works with the robot model and uses CloPeMa workspace software to control the robot. The robot relies on motion planning and kinematics carried out by MoveIt!

RViz

RViz [19] is a 3D visualization environment used with ROS. It allowed us to inspect the robot's collision model as we were developing it. It may also show all the coordinate systems defined on the robot, which was desirable for viewing the expected positions of visual markers on the end effector. RViz also provides a drag-and-drop interface for posing the robot and motion planning. This interface proved not flexible enough for our needs, so we eventually had to develop our own application for that purpose.

CloPeMa Workspace

CloPeMa workspace includes ROS packages and settings developed for this particular robot and is required for for the robot to work. The most important features are the "clopema_controller" package, which manages the communication with the DX100 control unit, the "netft_rdt_driver" package, which contains the "force_driver" nodes that provide the force data, and "clopema_kinematics", which facilitates forward and inverse kinematics used by MoveIt!. CloPeMa workspace adds support for all the sensors and actuators used on the robot, e.g. gripper motors, gripper rubbing actuators, force sensors, Xtion sensors, etc.

The full contents of the workspace and guide to its installation can be found here: [20]

2.2.2 OpenCV

OpenCV [21] stands for Open Source Computer Vision Library. It is a widely used, open source multi-platform library based on C/C++ but also usable with Python, Java and Matlab. OpenCV provides many ready-to-use functions for image processing, recognition and tracking of objects, 3d vision and photo camera calibration and many other uses.

We used OpenCV for visual markers detection and camera calibration.

2.2.3 GPhoto2

gPhoto2 [22] is an open-source set of applications for working with photo cameras under UNIX systems. Its creators aim to provide functionality which is typical for (proprietary) software bundled with the camera, but without their compatibility restrictions to Windows or MacOS systems.

gPhoto2 consists of two main parts. The gPhoto2 command-line application and the "libgphoto2" camera access and control library. We used gPhoto2 and libgphoto2 python binding to configure the cameras. Furthermore, our "PhotoActionServer" under ROS is also based on libgphoto2, thus we also used libgphoto2 during the experiment.

Chapter 3

Materials and Methods

3.1 Robot setup description

3.1.1 Robot dimensions

Figure 3.1 shows the nominal dimensions of the MA1400 manipulator, from which the parameters of the manipulator were obtained using Denavit-Hartenberg (DH in what follows) convention.

Table 3.1 states the DH parameters of the MA1400 manipulator, table 3.2 states the DH parameters of the r750 turntable, table 3.3 then states the transformations from the r750 flange to the mounting point of both manipulators. The base reference frame is defined as the Z axis lying on the rotation axis of the turntable, X axis in the direction towards the mounting of manipulator 2 and Y axis making a right-hand coordinate system with X and Z.

3.1.2 Robot control

The robot was controlled using MoveIt! under ROS (found in the subsection 2.2.1). We developed several control programs for the robot, all except

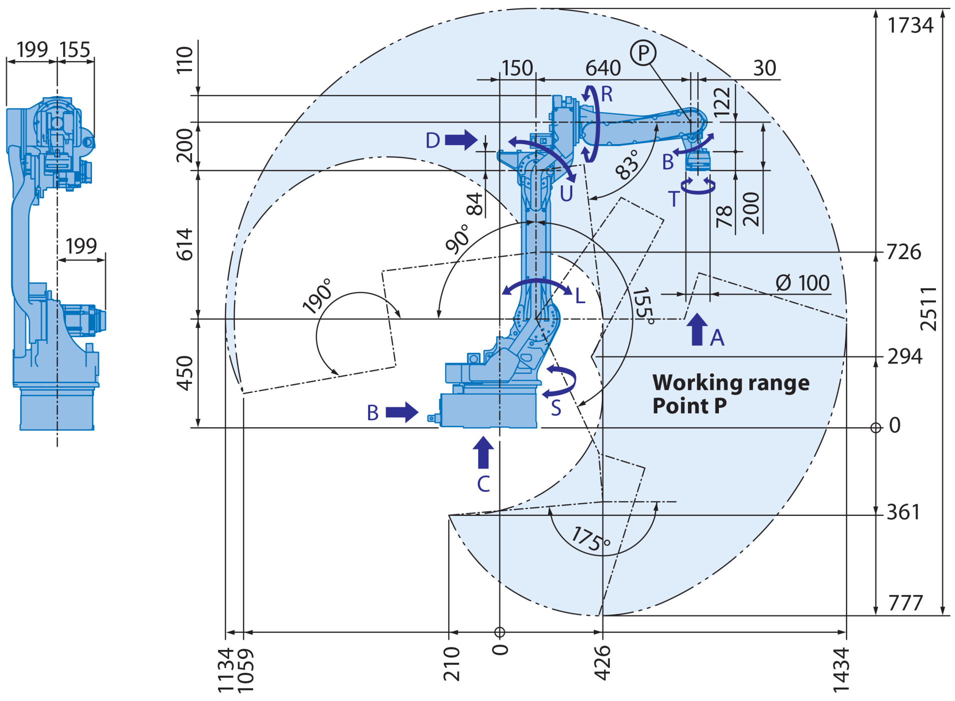


Figure 3.1: Dimensions and workspace of the MA1400 manipulator—front and top view, taken from [23]

a [m]	d [m]	α [rad]	θ offset [rad]
0.15	0.45	$-\pi/2$	0
0.614	0	π	$-\pi/2$
0.2	0	$-\pi/2$	0
0	-0.64	$\pi/2$	0
0.03	0	$\pi/2$	$-\pi/2$
0	0.2	0	0

Table 3.1: Nominal DH parameters of the MA1400, as stated by the manufacturer in Figure 3.1

a [m]	d [m]	α [rad]	θ offset [rad]
0	0.51	0	$-\pi/2$

Table 3.2: DH parameters of the r750 turntable link.

	Manipulator 1	Manipulator 2
displacement [m] (X, Y, Z)	0, -0.25, 0.16	0, 0.25, 0.16
rotation [rad] (X, Y, Z)	$\pi/12$, 0, 0	$-\pi/12$, 0, 0

Table 3.3: The mounting point of both manipulators in the r750 reference frame. Rotation is given in Euler angles ordered with rotation around Z being closest to the base frame.

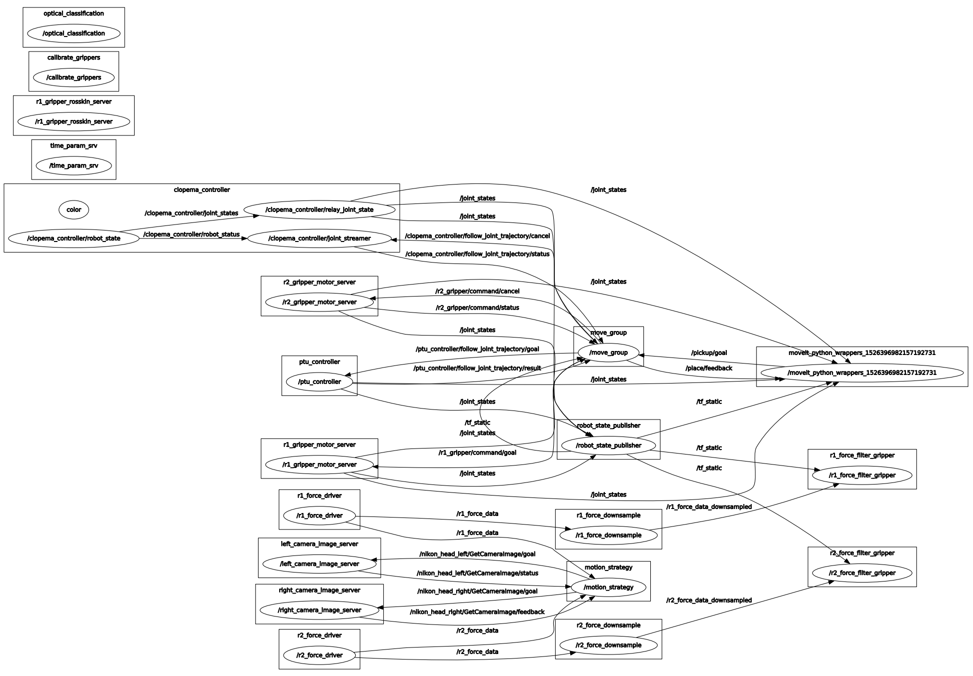


Figure 3.2: ROS nodes running during the self-touch.

“calibrate_force_sensor” being applications with a GUI.

The programs have a specialized robot library that customizes the workflow of underlying generic library to suit the task of force-sensor based touching. The underlying library is “RobotCommander”, a CloPeMa workspace Python library using “moveit_commander” to facilitate robot controls.

All program calls for forward kinematics, inverse kinematics, path planning, current robot state, collisions in given pose and trajectory execution are sent to MoveIt!’s “move_group”, which fulfills them using one of its modules or delegates them to “clopema_commander”, which interfaces with the robot control unit.

Move_group uses OMPL (Open Motion Planning Library) for planning trajectory, employing sampling-based methods. Forward and inverse kinematics is done by “clopema_kinematics” module, a CloPeMa workspace module that interfaces with move_group. Inverse kinematics is based on an analytical solution of a “similar” manipulator, which serves as a starting point for numerical optimization of the actual manipulator.

All the code used is organized in a Git repository [24]. Chapter 4 elaborates the safe collision behavior using force feedback.

3.1.3 Dimensions of the end effector

The icosahedron, as described in Section 2.1.3, lies in the axis of the last joint. The displacement of its center from the manipulator tip is 522.25 mm. The reference frame of the end effector lies in the center of the end effector and is rotated by $\pi/20$ radians around the Z axis with relation to the manipulator tip reference frame.

Positions of the markers

There are 20 markers on each icosahedron. They are placed in an equidistant pattern concentric with its faces. The rotations of the markers are of no particular pattern and are not relevant in this task. Table 3.4 shows the list of all markers with their positions in the icosahedron reference frame. Image 3.3 shows the centers of the markers on the icosahedron.

Face Number	X [mm]	Y [mm]	Z [mm]
1	32	0	-41.89
2	9.89	30.435	-41.89
3	-25.89	18.81	-41.89
4	-25.89	-18.81	-41.89
5	9.89	-30.435	-41.89
6	51.78	0	-9.89
7	16	49.245	-9.89
8	-41.89	30.435	-9.89
9	-41.89	-30.435	-9.89
10	16	-49.245	-9.89
11	41.89	-30.435	9.89
12	41.89	30.435	9.89
13	-16	49.245	9.89
14	-51.78	0	9.89
15	-16	-49.245	9.89
16	25.89	-18.81	41.89
17	25.89	18.81	41.89
18	-9.89	30.435	41.89
19	-32	0	41.89
20	-9.89	-30.435	41.89

Table 3.4: The centers of the markers in the end effector reference frame

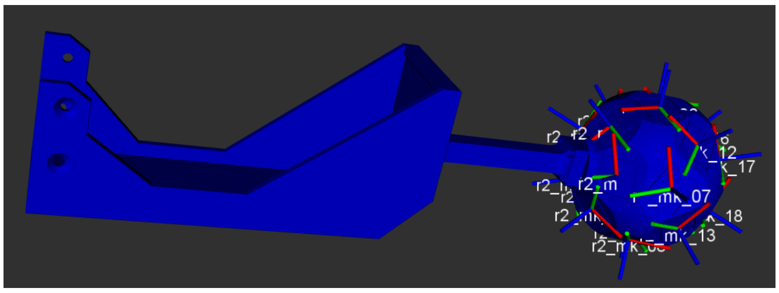


Figure 3.3: A selection of reference frames at the markers' centers.

Modification of the robot model

It is vital to assure the collision model of the robot used with MoveIt! is an accurate representation of the robot. The planner used by MoveIt! checks the model for collisions during planning. If a certain feature of the robot reached out of the area which the model claims as occupied, the planner could create a trajectory that would lead the robot into collision with an obstacle or even with a part of itself, which could cause damage to the manipulators.

We modified the robot model[1] to add all the parts of the end effector. We wrote a *xacro* file "selfcalib_sphere.urdf.xacro" that defines the end effector as three distinct bodies: the Bracket, the Beam and the Icosahedron. It uses the .stl model of the end effector as visual representation, but the collision model is formed by three geometrical primitives: two boxes and a sphere. Figure 3.4 shows the collision model.

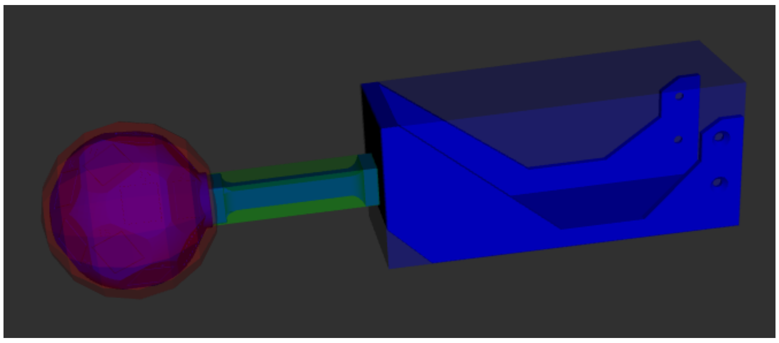


Figure 3.4: Collision model of the end effector over visual model, Bracket is blue, Beam is green and Icosahedron is red.

[1] /clopema_testbed/clopema_description/robots/clopema_spheres.urdf

The dimensions of the collision primitives:

- Bracket : 106 × 91 × 246 mm
- Beam : 28 × 28 × 120 mm
- Icosahedron : radius 60 mm

We also created coordinate frames in the center of each marker and each spherical patch with Z axis perpendicular to the plane or to the sphere. This simplifies the task of defining contact (self-touch) configurations, because we no longer need to consider the geometry of the icosahedron. We only need to ensure the Z axes — the normals of both spherical patches are anti-parallel (pointed towards each other) during the touching approach.

We modified the top-level xarco file, "clopema_CVUT_spheres.urdf.xacro", to load the end effector and place it on both grippers of the robot model. The file is compiled into "clopema_CVUT_spheres.urdf" by invoking the included bash script "clopema_CVUT_spheres.urdf.xacro".

3.1.4 Camera pose

Two Nikon D3100 photo cameras (see Section 2.1.2) were attached onto the robot. The cameras are located on a vertical beam that is attached to the r750 turntable, rotating the cameras together with the base of both MA1400 manipulators. Table 3.5 states the poses of both cameras.

	Camera 1	Camera2
displacement [m] (X, Y, Z)	-0.308, -0.2, 1.785	-0.308, 0.2, 1.785
rotation [rad] (X, Y, Z)	-2.4419, 0.0615, -1.5191	-2.4419, -0.0615, -1.6225

Table 3.5: The reference frames of both cameras in the turntable reference frame. Rotation is given in Euler angles ordered with rotation around Z being closest to the base frame.

3.1.5 Conversion of transformations to DH parameters

For sake of simplicity, we decided to represent the mounting of the manipulators described in Table 3.3 as another link in the DH parameters description of both manipulators. We wrote a simple Matlab function that takes a transformation matrix and calculates a set of two virtual links that reach to the point where the input transformation would put the new reference frame,

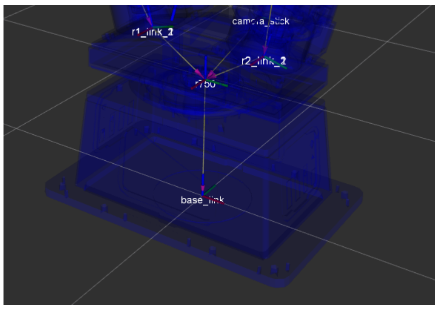


Figure 3.5: Detail of the turntable as seen in RViz. The turntable reference frame is denoted as "r750" and is located straight above the base frame.

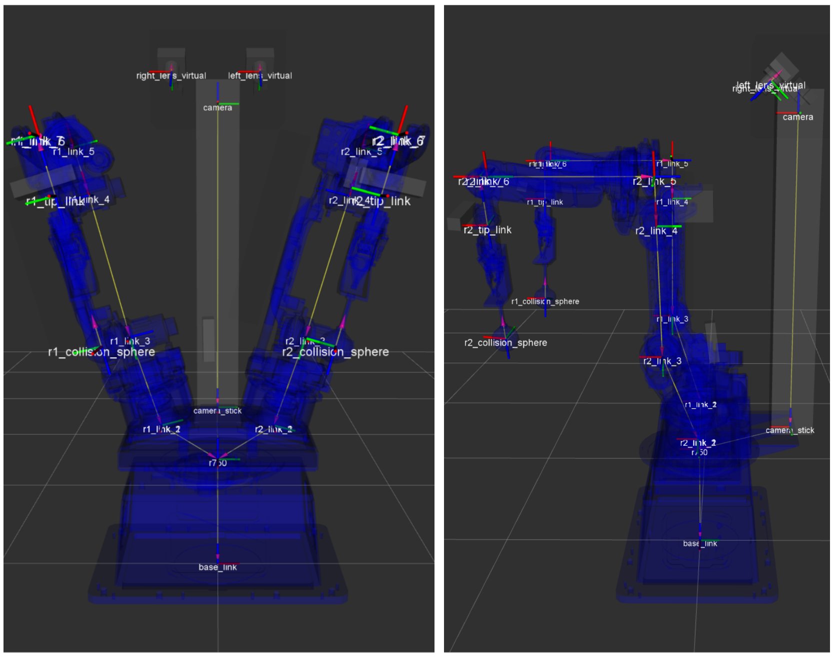


Figure 3.6: Reference frames and their location on the robot.

also preserving the orientation. Figure 3.5 shows the reference frame of the turntable and mounting of both manipulators. Table 3.6 shows the calculated DH parameters.

Camera 1				Camera 2			
a [m]	**d [m]**	α **[rad]**	θ **[rad]**	**a [m]**	**d [m]**	α **[rad]**	θ **[rad]**
0.1602	1.3942	$\pi/12$	0	0.1602	1.3942	$\pi/12$	π
0	-0.5118	0	0	0	-0.5118	0	π

Table 3.6: The DH parameters linking the turntable with the base of both cameras.

Since the parameters represent fixed joints and the previous joint, the turntable, has zero α and a parameters, the θ offset and d of the turntable can be directly added to the first virtual joint. If the previous joint did not have zero α and a parameters, the transformation would have to be redefined to account for these α and a, so that they would be removed from the turntable parameters and then the aforementioned approach could be applied.

As the second virtual joint has got zero α and a parameters and also represents a fixed joint, it can be added to the first joint of the manipulator, acting as its base link was elongated and rotated along the Z axis. We can also add one more set of DH parameters to represent the transformation from the manipulator tip to the end effector. In this case its formulation is trivial since the end effector is only displaced and rotated along the tip's Z axis.

Table 3.7 shows the complete DH parameter description of both arms kinematic chains. Note that after merging with the transformation matrix, the turntable joint is different for each manipulator. This needs to be so because the rotation axes of the first joints of the manipulators are not identical. Both joints, however, share the same joint coordinate (i.e. θ).

Manipulator 1 (right arm)				Manipulator 2 (left arm)			
a [m]	**d** [m]	α [rad]	θ [rad]	**a** [m]	**d** [m]	α [rad]	θ [rad]
0	-0.2630	$\pi/12$	$-\pi/2$	0	-0.2630	$\pi/12$	$\pi/2$
0.15	1.4159	$-\pi/2$	0	0.15	1.4159	$-\pi/2$	π
0.614	0	π	$-\pi/2$	0.614	0	π	$-\pi/2$
0.2	0	$-\pi/2$	0	0.2	0	$-\pi/2$	0
0	-0.64	$\pi/2$	0	0	-0.64	$\pi/2$	0
0.03	0	$\pi/2$	$-\pi/2$	0.03	0	$\pi/2$	$-\pi/2$
0	0.2	0	0	0	0.2	0	0
0	0.52225	0	pi/20	0	0.52225	0	pi/20

Table 3.7: The complete DH parameter description of both arm kinematic chains

This operation can be viewed as finding the proper DH parameters describing the robot as whole and, in fact, geometric approach comes to the same result.

Using the very same method, we converted the camera poses (see Table 3.5) into DH parameters (see Table 3.8). The obtained DH parameters serve as initial values for the calibration.

Camera 1				Camera 2			
a [m]	**d** [m]	α [rad]	θ [rad]	**a** [m]	**d** [m]	α [rad]	θ [rad]
0.1602	1.9042	-2.4396	-3.0171	0.1602	1.9042	2.4396	-0.1245
0	-0.5118	0	0.0953	0	-0.5118	0	3.0463

Table 3.8: The DH parameters of camera chains

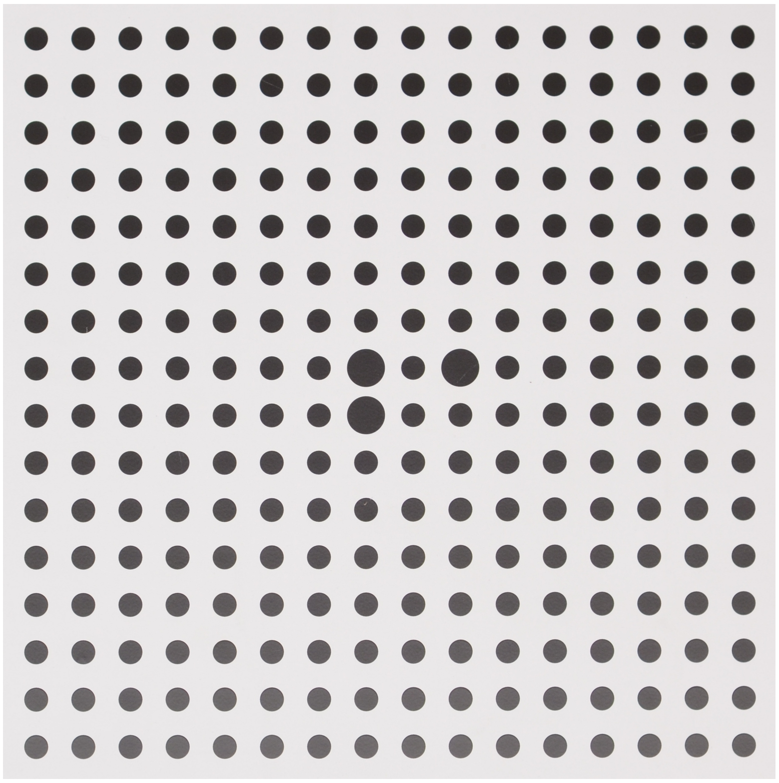


Figure 3.7: The calibration dot pattern used with legacy calibration method. This is a zoomed detail for clarity, the whole pattern comprises of 30 times 30 of the smaller dots, the three larger dots are around the middle of the pattern.

3.2 Standard camera calibration

For camera calibration, we used OpenCV. It provides functions for camera intrinsic and extrinsic calibration, projection of points, removal of camera distortion and 3D object reconstruction, as stated by the manual, which can be found in Subsection 2.2.2

We used a calibration library developed during the CloPeMa experiment, which relies on OpenCV for camera calibration but employs a closed-source binary for dot pattern recognition and acquisition of the object and image points. A specific dot pattern, Fig. 3.7, was used, which was not compatible with “findCirclesGrid” provided by OpenCV [25].

3.2.1 ArUco markers recognition

ArUco markers were used for pose estimation of the end effector. To detect ArUco markers in the image from camera, we use OpenCV ArUco module. Aruco markers, as stated by [26], are a kind of square-pixel binary fiducial markers. They are used to unambiguously detect a marked object in the image and discern its position and orientation with relation to the camera. The library allows markers of various sizes and pixel counts to be used.

Individual markers are defined by a data structure called Dictionary. Every marker known to the dictionary has a unique ID number assigned. OpenCV ArUco module provides a set of predefined dictionaries and also allows more dictionaries to be created automatically with m distinct markers on a $n \times n$ pixel grid or manually by specifying every pixel of every marker to be recognized. We used a predefined dictionary “DICT_ARUCO_ORIGINAL” that contains 1024 distinct markers and provides a simple way to inspect the ID by decoding the markers' pixels. Table 3.9 shows how the ID is encoded into the pixels, Figure 3.8 gives an example.

$\neg$ 0x200	0x200	0x100	0x100	0x200 $\oplus$ 0x100
$\neg$ 0x080	0x080	0x040	0x040	0x080 $\oplus$ 0x040
$\neg$ 0x020	0x020	0x010	0x010	0x020 $\oplus$ 0x010
$\neg$ 0x008	0x008	0x004	0x004	0x008 $\oplus$ 0x004
$\neg$ 0x002	0x002	0x001	0x001	0x002 $\oplus$ 0x001

Table 3.9: The encoding of an ArUco marker

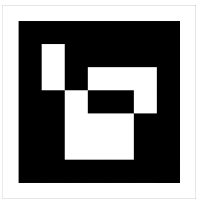


Figure 3.8: Example of an ArUco marker, the value can be decoded as 1×0x1 + 1×0x2 + 1×0x4 + 1×0x8 + 0×0x10 + 1×0x20 + 1×0x40 + 0×0x80 + 0×0x100 + 0×0x200 = 0x6F = 111

We used distinct marker IDs for the right and for the left icosahedron. The right one has IDs 101 to 120, the left one has IDs 201 to 220. The last two digits of the ID match the number of the icosahedron face.

3.2.2 Camera Model

The camera model used (Fig. 3.9, Eq. 3.1) is based on the pinhole camera extended with radial and tangential distortion coefficients. We use this model because it is supported by OpenCV [25].

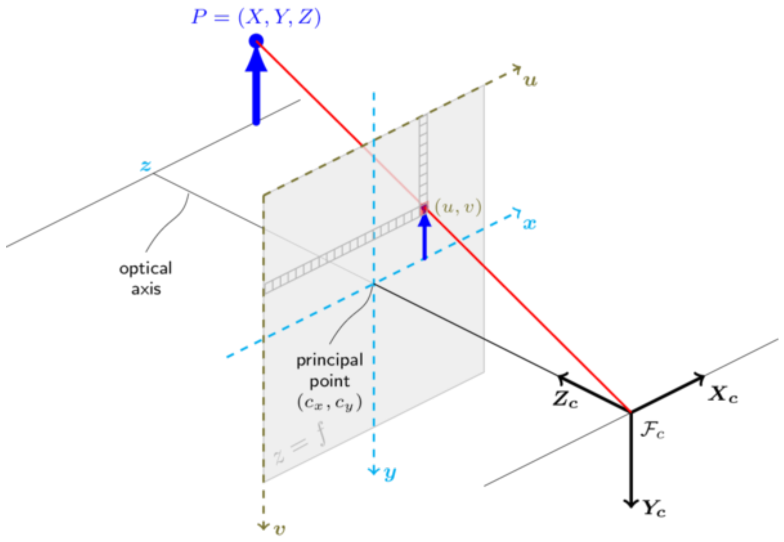


Figure 3.9: Pinhole camera model, taken from [25], Copyright 2011-2014, OpenCV Dev Team

$$\begin{aligned}
\begin{bmatrix} x \\ y \\ z \end{bmatrix} &= R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \\
x' &= x/z \\
y' &= y/z \\
r &= \sqrt{x'^2 + y'^2} \\
x'' &= x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\
y'' &= y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \\
K &= \begin{bmatrix} f_x & 0 & g_x \\ 0 & f_y & g_y \\ 0 & 0 & 1 \end{bmatrix} \\
\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= K \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix}
\end{aligned} \tag{3.1}$$

In this expression, X, Y and Z denote coordinates of a point in base reference frame, x, y and z denote coordinates of a point in the camera reference frame (point P in X_c, Y_c, Z_c frame in Fig. 3.9), R and t are rotation and translation defining a transformation from the calibration pattern frame to the camera frame. K is a camera matrix and points x' and y' are relative coordinates in the image plane, u and v are the image points.

A pure pinhole camera model would obtain u and v by multiplication of the K matrix with a vector of $[x', y', 1]$. This extended model computes a radius of the two points from the principal point (where the optical axis crosses the image plane) and amounts for radial distortion with coefficients k_1, k_2, k_3 and tangential distortion with coefficients p_1 and p_2, producing distorted coordinates x'' and y''. These are then transformed with K, resulting in u and v. Note that r only appears in powers of r^2, so it is not necessary to ever calculate the square root. Table 3.10 shows what parameters are considered intrinsic or extrinsic. Using camera calibration, we only calibrate intrinsic parameters.

Intrinsic	Extrinsic
f_x, f_y, g_x, g_y, k_1, k_2, k_3, p_1, p_2,	R, t

Table 3.10: Camera parameters classification.

3.2.3 Camera setup

Before the cameras could be calibrated, they had to be panned, tilted, zoomed and focused onto the area where the end effector touching would occur. Also the F-number had to be decided and set before calibration. The reason for this is that altering the zoom, focus or F-number of the camera would inevitably result in change of the camera intrinsic parameters.

We chose a point lying 1 meter in front of the robot base and 1 meter above ground as the center of the touching area. To pan and tilt the cameras, we placed an object into the center and altered the mounting of both cameras to make the object appear also in the center of both camera images.

We decided to set the zoom in such a way that the cameras would take an area about 0.6 meter on each side of the center. The reason for this is that there is a roughly 1 m wide intersection of both manipulators' workspace and if the touch would occur on its boundary, parts of the icosahedron with 0.1 m diameter could appear up to 0.6 m from the center.

The focusing was done using the autofocus with lowest F-number supported by the camera. This created very low depth of field and increased the precision of the autofocus. After the focusing was done, the lens was set to manual focusing, so that the autofocus would not alter the intrinsic parameters of the camera.

Finally, the F number was set to F/20 to increase the depth of field. We compensated for less exposure by setting slower shutter speed, 0.5 s. Because the robot comes to full stop before the picture is taken and resumes the motion after the picture is taken, the shutter speed can be arbitrarily high without causing motion blur.

Intrinsic calibration

Intrinsic calibration serves for estimation of the intrinsic parameters of the camera. Intrinsic calibration generally consists of two phases: (1) creation of a dataset and (2) optimization task with camera model over the dataset.

Creation of the dataset often involves capturing an object with known structure, often a planar chessboard or dot grid pattern manufactured for the purpose of calibration (other methods also exist, with multiple cameras observing a single point, e.g. a laser pointer dot). Here, for clarity, we assume a generic calibration pattern is used.

In every view, object points with known, unambiguous position on the calibration pattern need to be recognized. Then, correspondences between the object points and their two-dimensional projections, image points (pixels in the image) are found. Finally, the correspondences are inserted into the dataset.

The dataset should contain correspondences across multiple views, evenly covering the camera's field of view and containing the calibration pattern in diverse poses with relation to the camera.

The optimization stands for estimation of the parameters of a camera model, as well as the pose of the object in every view. We used "calibrateCamera" function from OpenCV [25]. The function performs three main steps:

1. Initial estimation of the camera matrix

2. Initial estimation of the pose of the calibration pattern in every view
3. Global Levenberg-Marquardt optimization on the re-projection errors, that is, the total sum of squared distances between the observed feature points and the projected object points, computed using the current estimates for camera parameters and the poses.

Parameter	Camera 1	Camera 2
f_x	5757	5744
f_y	5765	5747
g_x	1573	1614
g_y	2279	2272
k_1	-0.05426	-0.03299
k_2	0.09294	-0.07279
k_3	-0.1977	0.2432
p_1	-0.001700	0.00002765
p_2	0.001657	0.002601

Table 3.11: Intrinsic parameters of the cameras.

For the robot calibration, we assumed the intrinsic parameters of both cameras are given and extrinsic parameters (poses of the cameras with relation to the base coordinate frame) are among the parameters to be calibrated.

3.2.4 Camera integration

CloPeMa workspace includes support for one particular type of camera, namely Nikon D5100. To enable use of any other camera, we had to fall back to previous solution, a less known camera package called “iai_photo” [27], which is itself a modified camera library from the “bosch_drivers” ROS stack [28]. We extended it with the ability to select from multiple cameras connected to the system using udev symlink. We also implemented an ActionLib action server based on the extended iai_photo.

The cameras are connected to the PC over USB, using 10 m cable with repeater. We added udev rule on the PC for each of the cameras. Table 3.12 shows the configuration used. We also configured a launch file that launches iai_photo based ActionLib servers for both cameras while passing the proper parameters for each camera. The stability of the camera servers was tested and while not perfect, it was found sufficient to be used to collect the dataset.

The aforementioned stability issues seem to be caused by either one of the cameras stopping to respond to PTP messages generated by libgphoto2.

Setting	Camera 1	Camera 2
Symlink name	nikon_head_right	nikon_head_left
Serial number	000006062188	000006061413
Vendor ID	04b0	
Product ID	0427	
Subsystem	usb	
Group	plugdev	
Mode	0660	

Table 3.12: Udev rules for creating symlinks of the cameras.

The camera did not resume its function until restarted either by switch on the device or by power outage. This was most likely a problem with the communication protocol.

3.3 Multi-chain calibration

We consider the open-chain calibration as an optimization of an objective function $f(\phi, D, \zeta)$ over the vector

$$\phi = \{[a_1, ..., a_N], [d_1, ..., d_N], [\alpha_1, ..., \alpha_N], [o_1, ..., o_N]\}, \tag{3.2}$$

where $i \in 1..N$ is an index identifying one particular link of the kinematic chain, N is the number of all links across all kinematic chains of the robot model, a_i, d_i and α_i are DH parameters of the link i, and o_i is the offset of θ_i, the last DH parameter. The D denotes the dataset

$$D = \{[\xi_1, ..., \xi_M], [\vec{z}_1, ..., \vec{z}_M], [\theta_{1,1}, ..., \theta_{1,N}, \theta_{2,1}, ..., \theta_{M,N}]\}, \tag{3.3}$$

where $j \in 1..M$ is an index identifying one particular dataset point, M is the number of dataset points, ξ_j is information about how the measurement was taken (where, with which sensor), $\vec{z}_j$ is one particular measurement vector and $\theta_{j,i}$ is the DH parameter of the link i in the configuration in which the measurement $\vec{z}_j$ was taken. The constant ζ defines all other necessary parameters such as camera calibration, fixed transformations, fixed DH parameters or other properties of the robot. The optimization task is the problem of solving Eq. 3.4.

$$\phi^* = \arg\min_{\phi} f(\phi, D, \zeta) \tag{3.4}$$

where

$$f(\phi, D, \zeta) = \|\vec{g}(\phi, D, \zeta)\|^2 = \sum_{i=1}^{length\,\vec{g}} g_i(\phi, D, \zeta)^2$$

is the quadratic error.

3.3.1 Individual chain description

We consider the robot as four individual kinematic chains described by the DH parameters derived in the previous section. All the kinematic chains start in the base frame. The transformation from base to the first joint, i.e. the turntable joint, is identity. The two longer chains end with the end effectors—the end is in the center of each icosahedron. The two shorter chains end with the cameras. All the chains have common rotation of the first joint, but all other joints and parameters are separated. Note that since the mounting of the two manipulators on the turntable is not identical, the first 4-tuple of DH parameters also isn't the same, and we recognize two distinct turntable links in the arm chains and two more turntable links in the camera chains.

The manipulator has got 6 driven joints, denoted "S", "L", "U","R", "B" and "T" in the order from base to tip. Joint S connects the turntable with link S of the robot, joint L connects link S with link L and so on.

We added one last link to each kinematic chain, which represents the transformation to the end effector, denoted "icosahedron", or to the camera entrance pupil, denoted "camera". This last link is not connected by an actual joint and we consider its θ to be always zero, while it may still have non-zero offset.

Our calibration can treat any single component of any single DH parameter of any kinematic chain as fixed ($\in \zeta$) or as a parameter to be optimized ($\in \phi$). In addition to that, we consider one more parameter for optimization and that is the distance of icosahedron centers when they touch. Since the two icosahedrons have identical shape, up to the precision of manufacture, this parameter equals one diameter and so we call it the diameter and denote it q (q may $\in \zeta$ or $\in \phi$). Table 3.13 shows the names of individual links of the four kinematic chains.

Chain name	Links
Right arm	turntable1, S1, L1, U1, R1, B1, T1, icosahedron1
Left arm	turntable2, S2, L2, U2, R2, B2, T2, icosahedron2
Right camera	turntable3, camera1
Left camera	turntable4, camera2

Table 3.13: Links in kinematic chains.

3.3.2 Forward kinematics

We use standard approach for DH parameters forward kinematics [29, pp. 76-83]. The transformation T_i^{i-1} from link i to link $i-1$ is described by one set of DH parameters. It can be obtained as a product of four sub-transformations representing individual parameters a_i, d_i, α_i, θ_i. Eq. 3.5 show the sub-transformations and the resulting transformation.

$$\begin{aligned}
T_i^{i\alpha} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T_{i\alpha}^{ia} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
T_{ia}^{id} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T_{id}^{i-1} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
T_i^{i-1} &= T_{id}^{i-1} \cdot T_{ia}^{id} \cdot T_{i\alpha}^{ia} \cdot T_i^{i\alpha} \\
&= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.5}$$

The transformation from the icosahedron center to the base frame is then obtained as:

$$T_{icosahedron1}^{base} = T_{turntable1}^{base} T_{S1}^{turntable1} T_{L1}^{S1} T_{U1}^{L1} T_{R1}^{U1} T_{B1}^{R1} T_{T1}^{B1} T_{icosahedron1}^{T1}$$

for the right arm and

$$T_{icosahedron2}^{base} = T_{turntable2}^{base} T_{S2}^{turntable2} T_{L2}^{S2} T_{U2}^{L2} T_{R2}^{U2} T_{B2}^{R2} T_{T2}^{B2} T_{icosahedron2}^{T2}$$

for the left arm.

The transformations from the cameras to the base frame are

$$T^{base}_{camera1} = T^{base}_{turntable3} T^{turntable3}_{camera1}$$

for the right camera and

$$T^{base}_{camera2} = T^{base}_{turntable4} T^{turntable4}_{camera2}$$

for the left camera.

Using $T^{base}_{icosahedron1}$ or $T^{base}_{icosahedron2}$, we transform a marker M_i from the right or left icosahedron to the base frame:

$$T^{base}_{Mi} = T^{base}_{icosahedron1} \cdot T^{icosahedron1}_{Mi}$$

Positions of the markers in the icosahedron frame are stated in Table 3.4.

From these we can derive the $T^{icosahedron1}_{Mi}$. Then we transform the marker to the given camera frame using $T^{camera1}_{base}$ (inverse transformation of $T^{base}_{camera1}$) or $T^{camera2}_{base}$:

$$T^{camera1}_{Mi} = T^{camera1}_{base} \cdot T^{base}_{Mi}$$

Projection to camera frame

To obtain projected points we apply the calibrated camera model described in Eq. 3.6. First, we have to obtain the points in the camera frame:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = T^{camera1}_{Mi} \cdot \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix}, \text{ respectively } \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = T^{camera2}_{Mi} \cdot \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix}$$

where $[x_m, y_m, z_m]^T$ is a point in the frame of marker M_i and $[x_c, y_c, z_c]^T$ is the transformed point in the frame of the given camera.

Afterwards, we apply on this point the calibrated camera model to obtain the projected point $[u, v]$ in the 2D plane of the given camera:

$$\begin{aligned}
x'_c &= x_c/z_c \\
y'_c &= y_c/z_c \\
r &= \sqrt{x'^2 + y'^2} \\
x''_c &= x'_c(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x'_cy'_c + p_2(r^2 + 2x'^2_c) \\
y''_c &= y'_c(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y'^2_c) + 2p_2x'_cy'_c \\
\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= K \begin{bmatrix} x''_c \\ y''_c \\ 1 \end{bmatrix}
\end{aligned} \tag{3.6}$$

where K, $k_1..k_3$ and $p_1..p_2$ are camera parameters obtained by camera calibration described in 3.2.3.

3.3.3 Optimization problem formulation

We formulated the objective function as the error of computed marker projections across all markers captured by the cameras in all robot poses in the dataset and the error of distances of touching end effectors:

$$\vec{g}(\phi, D, \zeta) = \left[\vec{P}(\phi, D, \zeta),\ 3000 \cdot \vec{C}(\phi, D, \zeta)]\right], \tag{3.7}$$

where $\vec{P}(\phi, D, \zeta)$ is a vector of length $2M$ containing errors of computed marker projections in u and v (where M is a number of points in dataset D), and $\vec{C}(\phi, D, \zeta)$ is a K-element vector of the error of distances of touching end effectors. The ϕ is the parameter vector defined in Eq. 3.2, D denotes the dataset defined in Eq. 3.3 and ζ contains all other necessary parameters.

The marker error is defined as

$$\vec{P}(\phi, D, \zeta) = \left[\vec{p}(\phi, D_1, \zeta) - \vec{z}(D_1), ...,\ \vec{p}(\phi, D_M, \zeta) - \vec{z}(D_M)\right]$$

where $\vec{p}(\phi, D_i, \zeta)$ projects the marker given by the dataset point D_i, where $i \in A$, $A = \{1, ..., M\}$ (M is a number of points in dataset D) into the camera plane given by D_i using forward kinematics and camera model, as stated in the previous section. The dataset point includes information about which marker was detected, on which hand and using which camera. The $\vec{z}(D_i)$ obtains the actual position in the camera from the dataset point.

The distance error is defined as

$$\vec{C}(\phi, D, \zeta) = [c(\phi, D_1, \zeta) - q(\zeta), ..., \ c(\phi, D_K, \zeta) - q(\zeta)]$$

where the function $c(\phi, D_i, \zeta)$ computes the distance of the icosahedron centers in the configuration given by the dataset point D_i, where $i \in B$, $B \subset A$ is a subset of the dataset points where the pose of the robot is unique. We assume all robot poses in B feature touching end effectors. The function $q(\zeta)$ obtains the distance of the centers of touching icosahedrons from ζ.

In the Eq.3.7, we scaled up the distance error by the factor of 3000. The reason for this is that the focal lengths of both cameras are approximately 6000 px. An object of length 1 m posed 1 m in front of the camera will measure 6000 px in the image. The average distance of the markers on the end effectors from the cameras is about 2 m during the experiment, so a 1 m displacement causes a roughly 3000 px displacement in the image. By scaling the distance error, we converted the distance unit from meters to an equivalent of pixels.

3.3.4 Non-linear least squares optimization

Our calibration pipeline was based on Matlab and Optimization Toolbox. The calibration was done offline, based on a dataset collected during experiments on the robot, therefore it didn't need to be installed on the same computer used for robot control. Considering the computational intensity of the task, we chose a more powerful PC with Intel i7-6700K processor.

Optimization toolbox offers multitude of ready-to-use optimization functions. We chose "lsqnonlin"function, which offers two algorithms to choose from - Trust-Region-Reflective and Levenberg-Marquardt. We chose Levenberg-Marquardt algorithm over Trust-Region-Reflective, because Birbach et al. [3] and Bennet et. al. [4] use it in their works, furthermore, Hollerbach et al. [1] propose Gauss-Newton algorithm, which is related to Levenberg-Marquardt algorithm, for solving the calibration task.

General Levenberg-Marquardt algorithm requires the first derivative of the objective function, the Jacobian, to be known. The "lsqnonlin" implementation allows the estimation of Jacobian using finite difference. This increases the computational intensity of the optimization, because multiple evaluations of the objective function are required in each iteration. The reward for this drawback is that “lsqnonlin” is universally applicable on functions that don’t have Jacobian in analytical form or require complex calculus for it to be obtained.

3.3.5 Line descent optimization algorithms

Levenberg-Marquardt algorithm is a numerical optimization method used for solving non-linear least-squares problems. According to [30], it belongs to the family of line descent algorithms. This family of algorithms finds a free local minimum of the objective function $f(\vec{x})$ by iterative stepping from initial value $\vec{x}_0$ in the direction $\vec{v}$ by the step size γ. Other algorithms in this family are Gradient descent, Newton’s method, Gauss-Newton algorithm, etc. The algorithms in general work by repeating the iteration steps:

- find the direction $\vec{v}_k$ using $f(\vec{x})$, $f'(\vec{x})$ or other supplied function
- find the step size γ_k, e.g. using line search in the direction $\vec{v}_k$
- update the value $\vec{x}_{k+1} = \vec{x}_k + \gamma_k \cdot \vec{v}_k$
- check the accept condition, e.g. $f(\vec{x}_{k+1}) < f(\vec{x}_k)$,
  - if not satisfied, revert to $\vec{x}_k$ and modify $\vec{v}_k$ or γ_k for the next step
- check the stopping condition(s), e.g. number of iterations, size of $f(\vec{x}_{k+1})$, size of the step
  - if satisfied, end computation

Gradient descent

Gradient descent [30] uses the step direction opposite to the direction of the gradient, i.e. Jacobian of the objective function in the point x_k, and step size is either constant, proportional to the gradient or determined by line search for minimum value in the direction of the gradient. With step size proportional to the gradient with a coefficient γ, the update step is given as

$$\vec{x}_{k+1} = \vec{x}_k - \gamma f'(\vec{x}_k)^T.$$

Gradient descent is proven to be reliable, never diverges ($f(\vec{x}_{k+1})$ is always $< f(\vec{x}_k)$), but it suffers from slow convergence on certain objective functions as it nears the local minimum. In such case, a characteristic zig-zag pattern in $\vec{x}$ iterations can be observed.

Newton's optimization method

Newton's optimization method [30] is a way to numerically solve a set of nonlinear equations based on first-order Taylor polynomial approximation, denoted with subscript $_{T1}$. For the set of equations $\vec{g}(\vec{x}) \in \mathbb{R}^m \to \mathbb{R}^n$ the solution of $\vec{g}(\vec{x}) = \vec{0}$ is found by iteratively solving the problem

$$\vec{g}_{T1}(\vec{x}_{k+1})\Big|_{\vec{x}=\vec{x}_k} = \vec{g}(\vec{x}_k) + \vec{g}\,'(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k) = \vec{0},$$

where the solution can be formulated as

$$\vec{x}_{k+1} = \vec{x}_k - \vec{g}\,'(\vec{x}_k)^{-1}\vec{g}(\vec{x}_k).$$

The algorithm iterates until $\vec{x}_{k-1} - \vec{x}_k <$ required precision. This method can be used to minimize any nonlinear function $f(\vec{x})$ that has second derivation. In such case we search for a stationary point of $f(\vec{x})$, i.e. a point where $f'(\vec{x}) = \vec{0}$. If we substitute $\vec{g}(\vec{x}_k)$ with $f'(\vec{x})$ in the previous equation, we obtain

$$f'_{T1}(\vec{x}_{k+1})\Big|_{\vec{x}=\vec{x}_k} = f'(\vec{x}_k) + f''(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k) = \vec{0},$$

which gives the iteration step

$$\vec{x}_{k+1} = \vec{x}_k - f''(\vec{x}_k)^{-1}f'(\vec{x}_k).$$

Gauss-Newton algorithm

Gauss-Newton algorithm [30] is an employment of the Newton's optimization method on least-squares nonlinear functions. The algorithm is based on the fact that when we substitute the $\vec{g}(\vec{x})$ in the objective function $f(\vec{x}) = \|\vec{g}(\vec{x})\|^2$ with first-order Taylor approximation, we obtain

$$\|\vec{g}(\vec{x}_0) + \vec{g}\,'(\vec{x}_0)(\vec{x} - \vec{x}_0)\|^2 = \|\vec{g}\,'(\vec{x}_0)\vec{x} - \vec{g}\,'(\vec{x}_0)\vec{x}_0 + \vec{g}(\vec{x}_0)\|^2 = \|A\vec{x} - \vec{b}\|^2$$

where $A = \vec{g}\,'(\vec{x}_0)$ and $\vec{b} = \vec{g}\,'(\vec{x}_0)\vec{x}_0 - \vec{g}(\vec{x}_0)$.

The optimal solution $\vec{x}^*$:

$$\vec{x}^* = \arg\min_{\vec{x}} \|A\vec{x} - \vec{b}\|^2$$

can be found using pseudo-inversion: $\vec{x} = A^+\vec{b} = (A^TA)^{-1}A^T\vec{b}$. We obtain

$$\vec{x} = \vec{g}\,'(\vec{x}_0)^+(\vec{g}\,'(\vec{x}_0)\vec{x}_0 - \vec{g}(\vec{x}_0)) = \vec{x}_0 - \vec{g}\,'(\vec{x}_0)^+\vec{g}(\vec{x}_0).$$

Thus the iteration step is

$$\vec{x}_{k+1} = \vec{x}_k - \vec{g}\,'(\vec{x}_k)^+\vec{g}(\vec{x}_k).$$

Both Newton's method and Gauss-Newton algorithm are known to converge rapidly when the $\vec{x}$ is close to the minimum. The advantage of Gauss-Newton algorithm is that it doesn't require the second derivation of $f(\vec{x})$, i.e. the Hessian, which is needed in standard Newton's method. The main disadvantage is that the algorithm may not converge if the initial value $\vec{x}_0$ is too far from the minimum.

Levenberg-Marquardt algorithm

Levenberg-Marquardt algorithm eliminates the disadvantages of Gradient descent and Gauss-Newton algorithm by merging both approaches [30].

$$\vec{x}_{k+1} = \vec{x}_k - (\vec{g}\,'(\vec{x}_k)^T\vec{g}\,'(\vec{x}_k) - \lambda_k I)^{-1}\vec{g}\,'(\vec{x}_k)^T\vec{g}(\vec{x}_k)$$

With λ_k very high, the term $(\vec{g}\,'(\vec{x}_k)^T\vec{g}\,'(\vec{x}_k) - \lambda_k I)^{-1}$ approaches $\lambda_k^{-1}I$ and the behavior of the algorithm becomes akin to Gradient descent.

$$\vec{x}_{k+1} = \vec{x}_k - \frac{1}{\lambda_k}\vec{g}\,'(\vec{x}_k)^T\vec{g}((\vec{x}_k))$$

With λ_k very low, the term $(\vec{g}\,'(\vec{x}_k)^T\vec{g}\,'(\vec{x}_k) - \lambda_k I)^-1$ approaches $(\vec{g}\,'(\vec{x}_k)^T\vec{g}\,'(\vec{x}_k))$ and the behavior of the algorithm becomes akin to Gauss-Newton algorithm.

$$\vec{x}_{k+1} = \vec{x}_k - \vec{g}\,'(\vec{x}_k)^+\vec{g}(\vec{x}_k).$$

Thus, the variable λ allows a choice between Gradient descent's stability and Gauss-Newton algorithm's fast convergence on a continuous scale. Choice of appropriate λ is carried out during each iteration of the algorithm. The algorithm starts with $\vec{x}_0$ and λ_0 supplied by the user or default values are used. Then, the algorithm iterates over these steps:

- update the value $\vec{x}_{k+1} = \vec{x}_k - (\vec{g}\,'(\vec{x}_k)^T\vec{g}\,'(\vec{x}_k) - \lambda_k I)^{-1}\vec{g}\,'(\vec{x}_k)^T\vec{g}(\vec{x}_k)$

- check the accept condition $f(\vec{x}_{k+1}) < f(\vec{x}_k)$
  - if satisfied, set $\lambda_{k+1} = \frac{\lambda_k}{10}$
  - else revert $\vec{x}_{k+1} = \vec{x}_k$ and set $\lambda_{k+1} = 10\lambda_k$
- check the stopping conditions, e.g. number of iterations, size of the step, first order optimality
  - if satisfied, end computation

By employing this form of regulation, the parameter λ is held just above the boundary below which the algorithm becomes unstable. Thus the algorithm maintains just enough Gradient descent's stability while utilizing maximum of Gauss-Newton algorithm's convergence rate.

3.3.6 Evaluation

Before the optimization, we randomly divided the dataset to training and testing data with a ratio of approximately 70 to 30. Training and testing datasets contained distinct set of poses, this means that multiple observations of markers in a given pose were all added to the same batch. The optimization was based on the training data, the testing data was used to evaluate the result. We used root-mean-square (RMS) error to compare the results from multiple runs of the optimization. In addition to that, we used quiver plot to show the individual errors of the objective function after optimization. We also created a histogram of the errors to examine their distribution.

We calculated the RMS error of the marker positions (error over $\vec{P}(\phi, D, \zeta)$, using formulation of optimization problem from Section 3.3.3) as

$$\sqrt{\sum_{i=1}^{M} \frac{1}{M}(p_1(\phi, D_i, \zeta) - z_1(D_i))^2 + (p_2(\phi, D_i, \zeta) - z_2(D_i))^2} = \sqrt{\frac{1}{M}\|\vec{P}(\phi, D, \zeta)\|^2},$$

where M is a number of points in dataset D, ϕ is parameter vector and ζ defines all other necessary parameters, $\vec{p}(\phi, D_i, \zeta) = \left[p_1(\phi, D_i, \zeta),\ p_2(\phi, D_i, \zeta)\right]$ projects the marker given by the dataset point D_i, where $i \in A$, $A = \{1, ..., M\}$ into the camera plane given by D_i using forward kinematics and camera model. The $\vec{z}(D_i) = \left[z_1(D_i),\ z_2(D_i)\right]$ obtains the actual position in the camera from the dataset point, as was already described in Section 3.3.3.

In a similar way, we calculated the RMS error of the distances (error over $\vec{C}(\phi, D, \zeta)$) as

$$\sqrt{\sum_{i=1}^{K} \frac{1}{K}(c(\phi, D_i, \zeta) - q(\zeta))^2} = \sqrt{\frac{1}{K}\|\vec{C}(\phi, D, \zeta)\|^2},$$

where K is a number of poses, the function $c(\phi, D_i, \zeta)$ computes the distance of the icosahedron centers in the configuration given by the dataset point D_i, where $i \in B$, $B \subset A$. The function $q(\zeta)$ obtains the distance of the centers of touching icosahedrons from ζ, as was already described in Section 3.3.3.

We compare individual results of the optimization using a quadratic sum of both values,

$$\sqrt{\frac{1}{M}\|\vec{P}(\phi, D, \zeta)\|^2 + \frac{1}{K}\|3000\,\vec{C}(\phi, D, \zeta)\|^2}$$

This sum assumes the same weight of the marker positions and distances errors regardless of the ratio of marker positions and robot poses (for explanation of the constant coefficient 3000 see the end of Section 3.3.3).

Chapter 4

Safe self-touch behavior

When using MoveIt! manipulation software, any collisions are avoided by default, based on the robot collision model and a model of its surroundings. This avoidance behavior restricts any two objects on the scene from getting close enough to touch each other.

To enable collision of particular objects, one needs to specify a list of pairs of objects that may collide with each other and pass the list to the planner.

At this point, operation of the robot without other means of collision detection is potentially risky. Although there may seem to be a straight path without collisions between the start and the desired position, the planner may generate trajectory in which a collision occurs. The reason for this are joint angle constraints, which may cause the planner to change the robot configuration on the way.

A workaround for this problem exists. It is possible to plan a major part of the trajectory with full collision avoidance, until the gripper gets to close vicinity of the object, and then interpolate the final approach with collision of the gripper enabled. The interpolation function ensures that its output is only straight path or nothing.

However, to perform safe touching with the object, a force sensor on the base of the gripper must be used during the final approach.

4.1 Force sensor calibration

Precise self-touch behavior without excessive forces requires sufficiently sensitive collision detection via measurement of the forces acting on the gripper. The robot is equipped with 6-D force/torque sensors on the base of each gripper. The total force acting on the sensor equals to a sum of the weight of the gripper, inertial force of the gripper, and any external force originating from a collision. In addition to that, the sensor has a significant offset, which needs to be measured before the actual force can be obtained.

We employed a simple method to obtain the offset of the sensor. The reported force was measured across multiple orientations of the gripper w.r.t. the gravity vector. In every orientation, 1000 samples of the sensor output were taken and arithmetic average was logged. Because the inertial force in a static position is zero, in each position, only the weight of the gripper and the offset are measured. By rotating the gripper, the direction of the weight vector is changed while its magnitude stays the same and offsets in individual dimensions remain constant. Therefore, the forces measured in 3D in individual orientations shall all lie on a sphere with the center position equal to the offset and radius equal to the weight of the gripper.

We found that in addition to significant offset, the sensor also suffers from minor multiplicative error. All attempts to fit a sphere to the measured data were met with systematic errors which resulted in the center tending to the direction where the density of measured samples was highest. Consequently, fitting of an ellipsoid was found necessary.

The position of the center of the ellipsoid equals the offset of the force sensor. The semi-axes are equal to the weight of the gripper with small multiplicative error. We claim the arithmetic average of the three semi-axes as the "true" weight and the ratio of individual semi-axes to the average as the multiplicative error.

Our calibration script can be found in the project's GitLab repository [24].

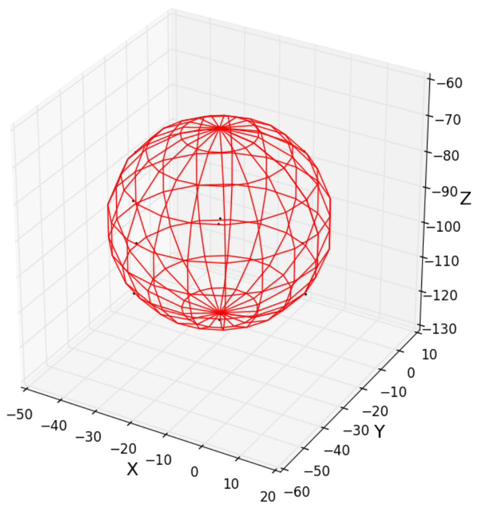


Figure 4.1: Example output of the calibration script, black dots are logged forces, red wire frame represents a fitted ellipsoid.

4.1.1 Drift of the offset

After repeated runs of the calibration script, it was discovered that the calculated offset varies by up to one newton between two executions of the script. This variance is too big to be explained by sensor noise or vibrations. Failure of the fitting algorithm was also rejected.

The calculated offset was eventually found dependent on the orientation of the gripper before the calibration. The variance was confirmed to be caused by offset drift.

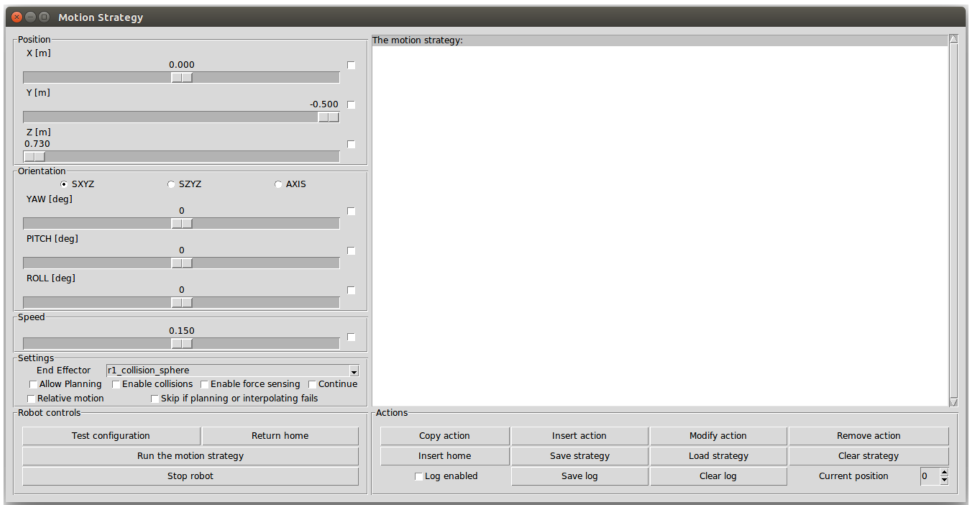
This disabled our intended user case - the sensor would be calibrated once and offset would be saved into a file, which would be loaded by the collision detection program at every startup. Because of the drift, the offset needs to be measured a certain time after every change of the gripper orientation and then repeatedly in prolonging time intervals.

Since the full calibration requires a certain amount of samples in many orientations to be taken and is relatively time-consuming, we decided for a simpler calibration method, where 1000 samples are taken in the current orientation before every touching motion. The method needs to be implemented within the collision detection program and by design cannot tell apart the offset and weight of the gripper. The calibration is only valid until the gripper changes its orientation, but due to the aforementioned issues, so does the full calibration. We decided to use this simplified calibration in our collision detection program.

4.2 Motion Strategy GUI

To carry out experiments, we had to develop a user interface, which would allow the user to specify a complex motion "plan" with collision detections and resolutions. Because the "plan" lies above the motion plan generated by MoveIt!, we call it motion strategy.

4.2.1 Description of the interface



X [m] 0.000
Y [m] -0.500
Z [m] 0.730
[✓ SXYZ] SZYZ AXIS
YAW [deg] 0
PITCH [deg] 0
ROLL [deg] 0
Speed 0.150
End Effector r1_collision_sphere
Allow Planning
Enable collisions
Enable force sensing
Continue
Relative motion
Skip if planning or interpolating fails
Log enabled
Current position 0

Figure 4.2: Screenshot of the GUI.

Motion strategy editor panel

The motion strategy GUI consists of four main areas. The large area on the right is the motion strategy editor panel. The current motion strategy is shown here.

Action configuration

On the left of the main window is the action configuration. The configuration is further divided to the Position section, which allows the user to specify the position of the gripper, the Orientation section, which allows the user to specify the orientation of the gripper, the Speed section, which allows the user to specify the maximum rotation speed of the robot motors during the action, and the Settings section with the following options:

- What is the point on the end effectors to be posed into the specified pose (choice from a list).
- Whether to use planning or interpolation.
- Whether to enable collisions, which relates to the collision of the two
- icosahedrons only.
- Whether to use collision detection via force sensing.
- Whether the next action will terminate the motion strategy in case of collision or whether the strategy should continue with the next action.
- Whether the position is given w.r.t. the previous position.
- Whether to skip the action when planning or interpolating cannot find an acceptable path.

Motion strategy editor controls

Below the editor window, motion strategy editor controls are located. The first row works in conjunction with the currently selected action in the editor window. The second row works with the whole strategy, with exception of "Insert home", which works with the currently selected action. The last line controls dataset creation. Logging positions for the dataset is disabled at default. The log can be saved by the "Save log" button or cleared with the "Clear log" button. The "Current position" states how many positions have been logged and at the same time allows the user to rewrite any number of last-logged positions by reducing the number in the spin box.

Robot controls

The last area is the robot controls area, located in the left bottom corner. This allows to test the current action configuration, run the motion strategy, return the robot to home position and stop the robot.

4.2.2 Program Structure

The application was written in Python 2.7. Several Python modules were used for various purposes. The following table lists the most important modules and their use:

- ActionLib: Interaction with the camera server
- CV2 (OpenCV Python binding): Finding the markers in the image
- NumPy: Math, especially matrix calculations
- RobotCommander: Interaction with the robot, path planning, kinematics
- Rospy: Interaction with ROS
- Tkinter: Creation of the GUI

Motion Strategy application

The “motion_strategy.py” file contains the program that creates the GUI and defines the objects that represent actions in the motion strategy. The motion strategy itself is implemented as a linked list of actions. All modifications to the motion strategy are recursive calls that are passed by each action to the following actions until the desired action is found. All actions are able to add themselves to the GUI component that presents the motion strategy to the user and also remove themselves from the component.

Robot library

The “robot_lib.py” file contains the robot top-level library. All calls to robot controls are asynchronous, which means they initiate the trajectory execution and do not wait for the robot to reach the end of the trajectory. During initialization, the robot library launches a standalone thread that in periodic time intervals checks whether the end of the trajectory has been reached and if so, starts another action. As the action is run, it passes a reference to the next action to the robot library. When the end of the trajectory is reached, the robot library calls the next action using this reference, upon which a reference to the next action of the called action is passed to the library. This way the program iterates over the entire motion strategy, until a “None” reference is returned. The same mechanism is used to schedule an action to be run if a collision is detected.

The collision detection logic is carried out whenever the ROS subscriber receives a message containing current force measurements from the force sensor. If the last started action allowed collision detection, the raw value is corrected for offset and then compared to a threshold. If the threshold is exceeded, the robot is immediately stopped and the current position is logged, marker positions are requested from the camera library and are logged too. The offset is calculated by the method described in Subsection 4.1.1.

Camera library

The The “camera_lib.py” file contains the camera top-level library, which allows images to be captured by both of the cameras and ArUco markers to be detected in images. The library also counts the centers of the markers from corner points and converts the array of marker coordinates into a string for logging purposes.

Calculation of marker centers from corner points

OpenCV ArUco module provides the coordinates of all four corners of the marker. In the dataset, we only log the position of the center of each marker. The reason for this is that the rotation of the marker along the face normal is chosen as what fits best. We don’t have the information required to reproject the corners of the marker, only the center, which is invariant to the rotation along the face normal.

We assume that the center of any quadrilateral, i.e. the intersection of both diagonals, under pinhole projection retains its position at the intersection of both diagonals. For four corners $\vec{p}_1 = \begin{bmatrix} x_1 & y_1 \end{bmatrix}$, $\vec{p}_2 = \begin{bmatrix} x_1 & y_1 \end{bmatrix}$, $\vec{p}_3 = \begin{bmatrix} x_1 & y_1 \end{bmatrix}$ and $\vec{p}_4 = \begin{bmatrix} x_1 & y_1 \end{bmatrix}$ in clockwise order, the center is found as the intersection of a line given by $\vec{p}_1$ and $\vec{p}_3$ and a line given by $\vec{p}_2$ and $\vec{p}_4$. The intersection (x, y) can be found as

$$x = \frac{(x_3y_1 - x_1y_3)(x_4 - x_2) - (x_4y_2 - x_2y_4)(x_3 - x_1)}{(x_3 - x_1)(y_4 - y_2) - (x_4 - x_2)(y_3 - y_1)},$$

$$y = \frac{(x_3y_1 - x_1y_3)(y_4 - y_2) - (x_4y_2 - x_2y_4)(y_3 - y_1)}{(x_3 - x_1)(y_4 - y_2) - (x_4 - x_2)(y_3 - y_1)}.$$

This assumption about the intersection of diagonals does not stand after the camera model described in chapter 3.2.2 is applied, however, the error may be rather low. We tested the error arising from center being found on distorted image using the corner points from real images captured by one of the cameras. We undistorted the points by OpenCV function "undistortPoints", then found the centers of these undistorted markers and reapplied the camera model. Finally we found the centers using the distorted corner points and compared both results. The error was found to be in order of 10^{-2} px, which is negligible considering that with our settings, OpenCV finds the corner points with granularity of whole pixels.

4.3 Test of the collision detection

For reliable collision detection, it is essential to set an appropriate detection threshold. Three various thresholds were tested on 7 different collision paths with a weighted paper box lying freely on the table. Force and force difference were logged during the experiment. Figure 4.3 shows absolute value of force during the experiment. Zero value is reported when collision detection is disabled. Figure 4.4 shows measured peak forces. Figure 4.5 shows a typical course of force during collision.

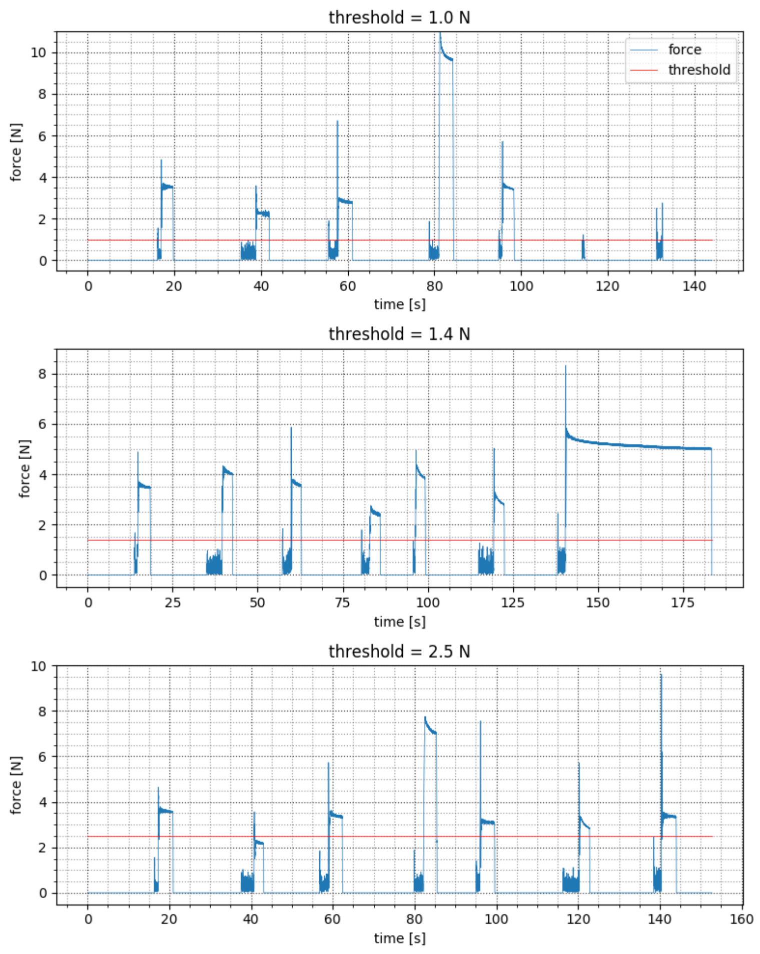


Figure 4.3: Force during the experiment.

4.3.1 Evaluation of the collision detection behavior

The course of the force exhibits a characteristic peak and settling behavior. The height of the peak correlates with the chosen detection threshold.

There seems to be a wide range of acceptable detection thresholds that aren't prone to false positive detections and also don't cause excessive force load to the robot. The difference was found too noisy to be used for reliable collision detection alongside the force.

Because of issues with slow response time, a very low approach speed had to be used during the dataset collection to minimize the error caused by late response to the collision. We recorded about 200 to 300 millisecond delay from collision occurrence to robot stopping. To minimize the error,

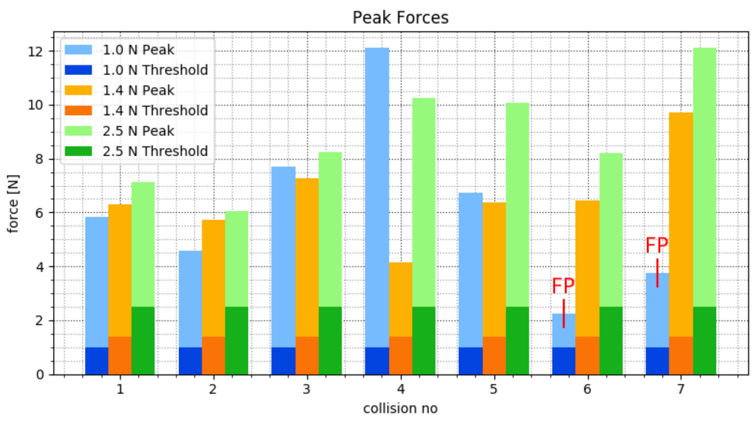


Figure 4.4: Peak forces on collisions.

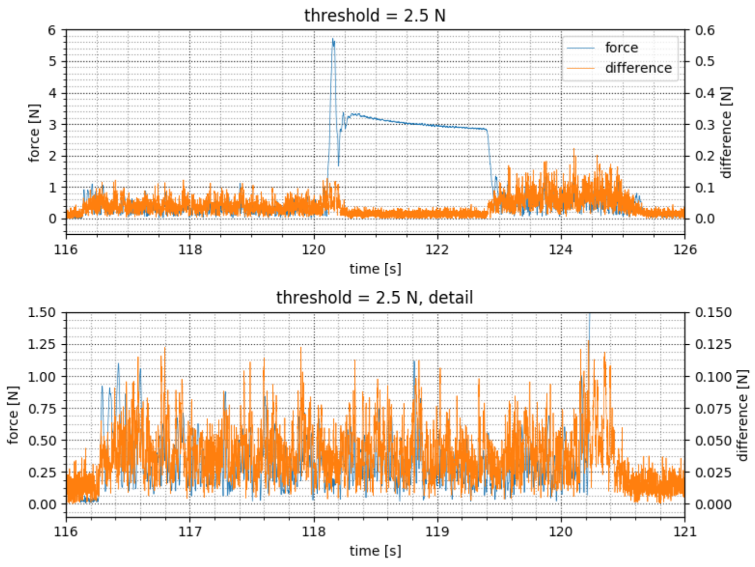


Figure 4.5: Typical course of force during collision with a paper box.

we reduced the maximal angular rate of the joints during collision approach to 0.01 radians, resulting in a displacement of 2 to 3 mm per axis and per meter of rotation radius. We found that during collision approaches in the direction of X axis and in the region where collisions occur during dataset collection (0.6 to to 1.3 m in front of the robot, 0.5 m to either side and heights 0.7 to 1 m as written in 5.2), only the first joint of the manipulator (joint S, as denoted in Subsection 3.3.1) reaches the speed limit and other joint speeds are substantially lower. Thus the maximum error is bound to $\sqrt{1.3^2 + (0.5 + 0.25)^2} \cdot 0.01 \doteq 1.5 \cdot 0.01 \cdot 0.3 = 4.5 \cdot 10^{-3}$ m, which is 4.5 mm.

Due to the reduced speed of the joints, less noise caused by vibrations was perceived by the sensor and we could reduce the threshold to 0.8 N

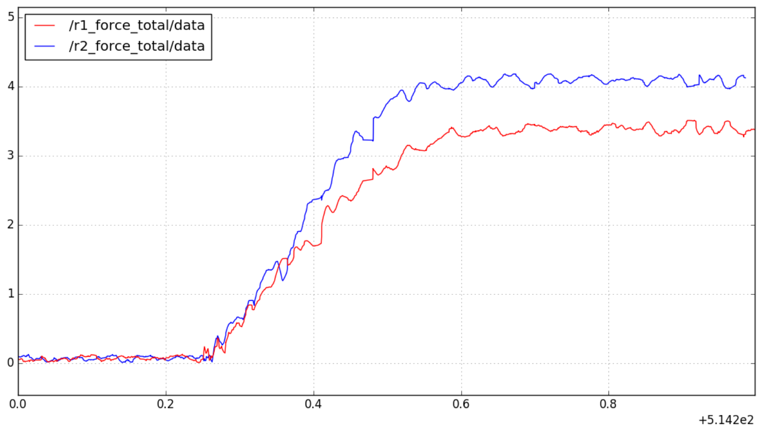


Figure 4.6: Typical course of force during touching of the icosahedrons, right icosahedron approaches static left icosahedron, red = right, blue = left, touch occurs at 0.25 s and robot comes to full stop at 0.55 s

without causing false positive detections. Figure 4.6 shows the course of forces measured by both force sensors during approach and touching of the icosahedrons. Compare to 4.5 to see the improvement in noise levels.

Chapter 5

Datasets

5.1 Dataset Structure

We work with two different dataset structures. Original dataset form (Subsection 5.1.1) is the one produced by our dataset collection program, found in the Section 5.3. Reformed dataset (Subsection 5.1.2) is loaded by the calibration script. We provide a simple Matlab script to convert from Original dataset form to Reformed dataset. It can be found in our GitLab repository [24].

5.1.1 Original Dataset Form

The dataset consists of the assumed pose of both icosahedron centers (from forward kinematics), the joint configuration of the robot, the magnitude of force measured by both force sensors during the contact, the names of saved camera images and coordinates of the projections of every marker into each of the cameras.

The projections are sorted from lowest to highest marker ID in the right camera image, and again from lowest to highest marker ID in the left camera image. If a marker was not found in the image, its coordinates are denoted as (-10000, -10000). The structure of one line in the dataset is as follows.

- Position of the right icosahedron: X, Y, Z [m]
- Orientation of the right icosahedron: W, X, Y, Z [quaternion]
- Position of the left icosahedron: X, Y, Z [m]
- Orientation of the left icosahedron: W, X, Y, Z [quaternion]
- Joint configuration of the robot: turntable, S1, L1, U1, R1, B1, T1, S2, L2, U2, R2, B2, T2 [rad]
- Peak force (magnitude) during collision on right arm [N]
- Peak force (magnitude) during collision on left arm [N]
- Name of the image file from right camera
- Name of the image file from left camera
- 40 $\times$ positions of the marker centers (IDs 101 to 120, 201 to 220) in the right camera: u, v [px]
- 40 $\times$ positions of the marker centers (IDs 101 to 120, 201 to 220) in the left camera: u, v [px]

5.1.2 Reformed Dataset

For the optimization, we decided to reform the dataset so that one line would relate to one data point. The reformed dataset contains only the markers that were actually detected. It consists of a number uniquely defining the robot pose, a face number of the detected marker, the number of the arm (1 for right or 2 for left), the number of the camera (1 for right or 2 for left), the coordinates of the projected point and the current robot joint configuration. The structure of one line in the dataset is as follows.

- Pose ID: monotonic integer
- Face ID: 1 to 20
- Arm number: 1 or 2
- Camera number: 1 or 2
- Position of the marker center in the camera: u, v [px]
- Joint configuration of the robot: turntable, S1, L1, U1, R1, B1, T1, S2, L2, U2, R2, B2, T2 [rad]

5.2 Individual datasets

We collected 2 different datasets that have very similar structure. Both of the datasets are collected based on a grid of positions, both use randomized touch angles and contain approximately the same number of poses.

5.2.1 Dataset 1

Dataset 1 contains poses with randomly chosen spherical tiles touching under randomized azimuth and with randomized rotation around the common normal. Both of the angles are whole numbers of degrees between -30 and 30, sampled from discrete uniform distribution. The collisions were done in a horizontal 5×5 grid in heights of 0.8 m and 1 m. The grid is 0.8 m long and 0.8 m wide, with the center lying 0.95 m in front of the robot. In each position, touching was done within two different orientations, with the point on the grid lying approximately in the point of touch. Some of the poses in the higher grid could not be reached and so they were skipped. Fig. 5.1 shows the positions of centers of both icosahedrons as they traversed the grid.

The dataset contains 82 poses and 1649 marker projections in total, which makes circa 20 markers per pose. Out of the 1649 marker projections, 892 come from right camera and 757 from the left camera. The reason why the left camera took lesser amount of markers in total seems to be the asymmetric shape of the manipulators. See figure 3.1, additional cables are guided through a gooseneck tubing on the left side of the L link, further increasing the asymmetry.

In Fig. 5.2, projections of markers from both arms to right and left camera are visualized. In Fig. 5.3 we show orientations of individual icosahedron in all measured poses. In the figure, two different orientations of the icosahedrons can be seen in each position on the grid. In Fig. 5.4 we visualize the distributions of joint angles across measured poses. As can be seen, the stimulation of joints R1 and R2 may be insufficient. Subtle over-learning of the robot model might occur, that would not be apparent until the calibration would be tested on a new dataset where joints R1 and R2 were sufficiently stimulated.

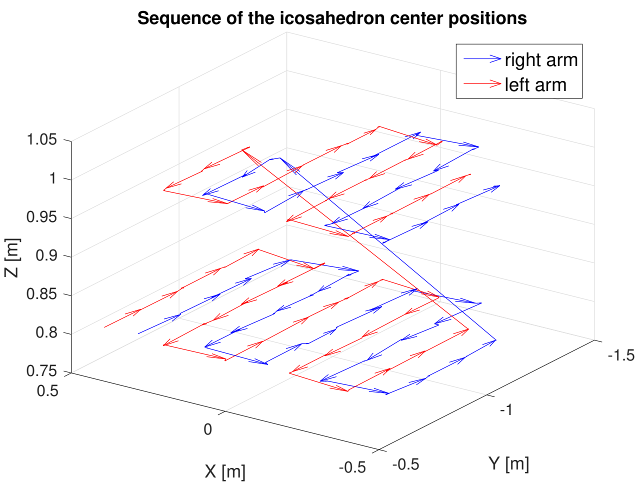


Figure 5.1: Locations of icosahedron centers in dataset 1. The end of each arrow denotes a position of icosahedron center in individual poses (red - left arm, blue - right arm). As can be seen, data were acquired in heights of 0.8 and 1 m.

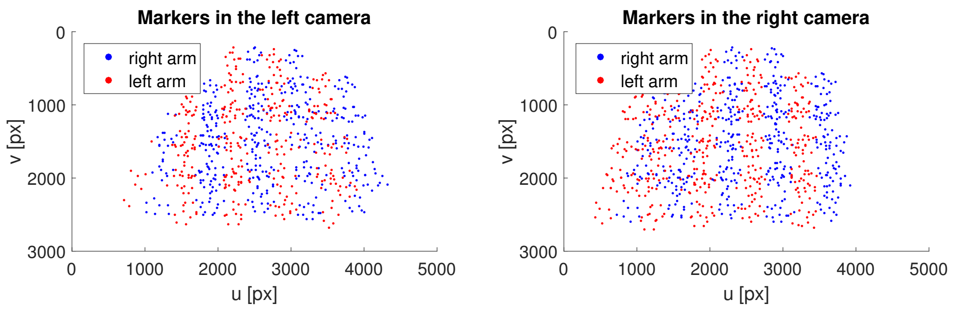


Figure 5.2: Projection of markers on icosahedron from right/left arm to right/left camera.

5.2.2 Dataset 2

Just like in dataset 1, this dataset contains poses with randomly chosen spherical tiles touching. All aspects of the dataset are the same except the heights of the grids, which are 0.7 m and 0.9 m (see Fig. 5.5). In each position, touching was done in two different orientations. Moving the grid lower helped with reaching all the poses. Fig. 5.5 shows the positions of centers of both icosahedrons as they traversed the grid.

The dataset contains 100 poses and 1774 marker projections in total, which makes circa 18 markers per pose. Out of the 1774 marker projections, 961

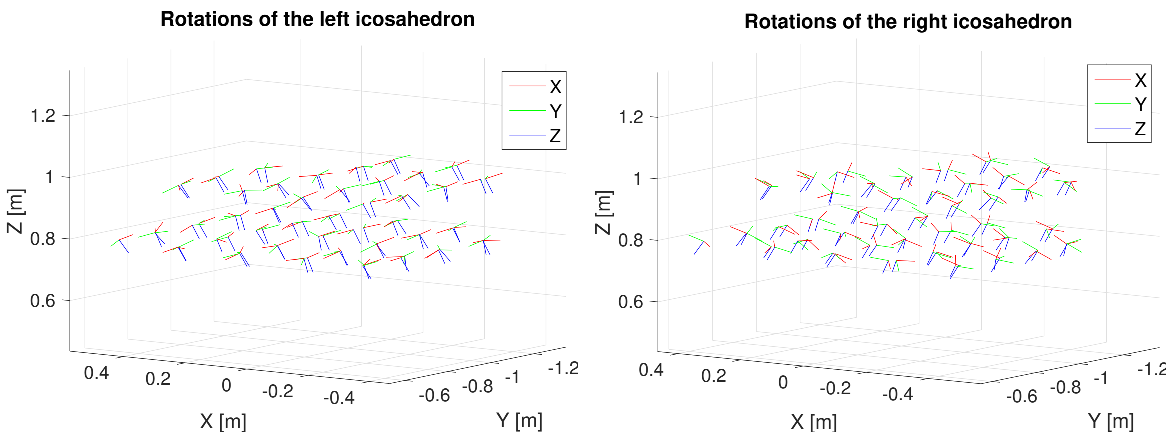


Figure 5.3: Orientations of individual icosahedron in all measured poses of dataset 1.

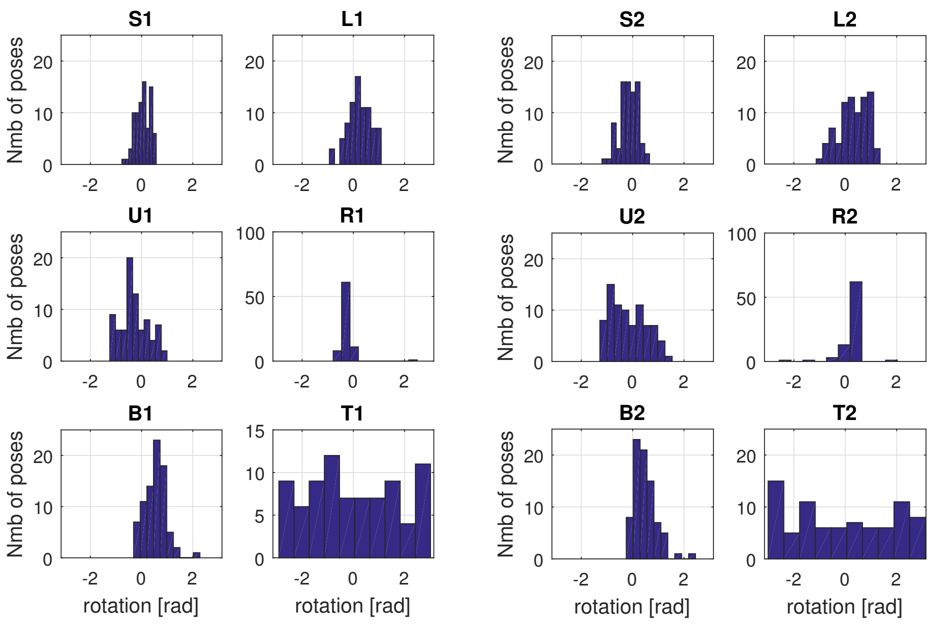


Figure 5.4: Distributions of joint angles across measured poses of dataset 1 (joints with number 1 correspond to right arm, joints with number 2 correspond to left arm).

come from right camera and 813 from the left camera.

In Figures 5.2, 5.3 and 5.4, we show the projection of markers on icosahedron from right/left arm to right/left camera, orientations of individual icosahedrons in all measured poses of dataset 2 and distributions of joint angles across measured poses of dataset 2, respectively.

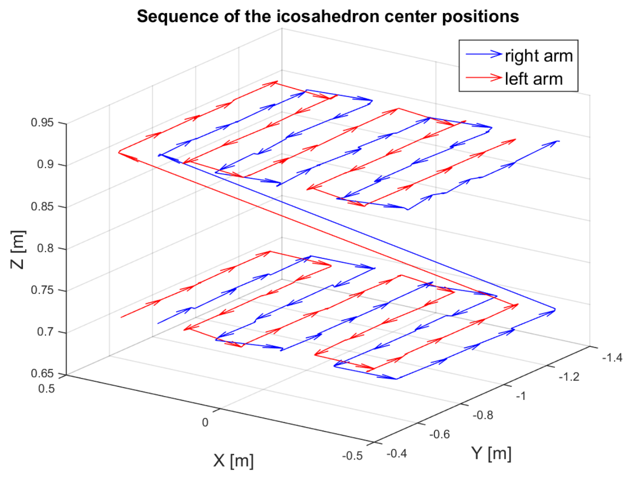


Figure 5.5: Locations of icosahedron centers in dataset 2. The end of each arrow denotes a position of icosahedron center in individual poses (red - left arm, blue - right arm). As can be seen, data were acquired in heights of 0.7 and 0.9 m.

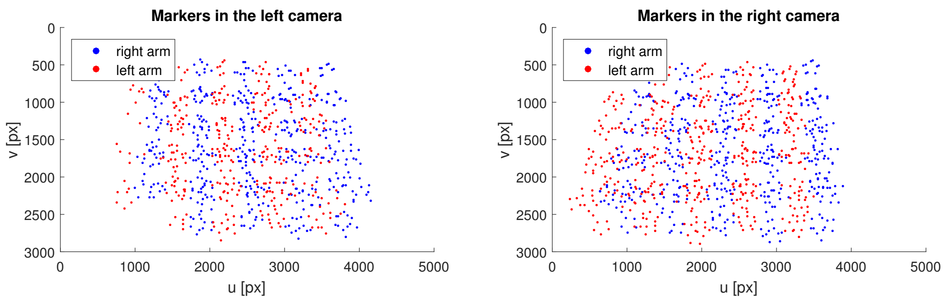


Figure 5.6: Projection of markers on icosahedron from right/left arm to right/left camera.

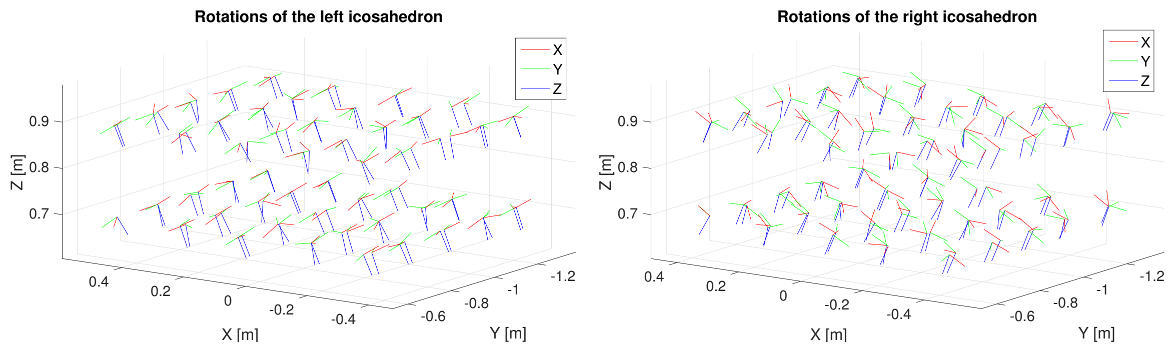


Figure 5.7: Orientations of individual icosahedron in all measured poses of dataset 2.

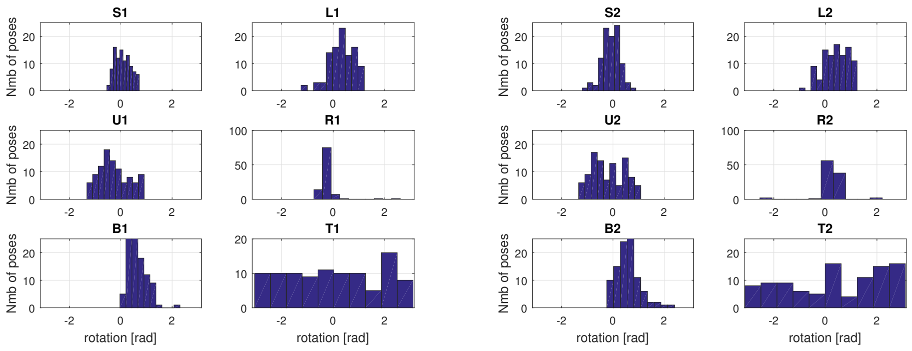
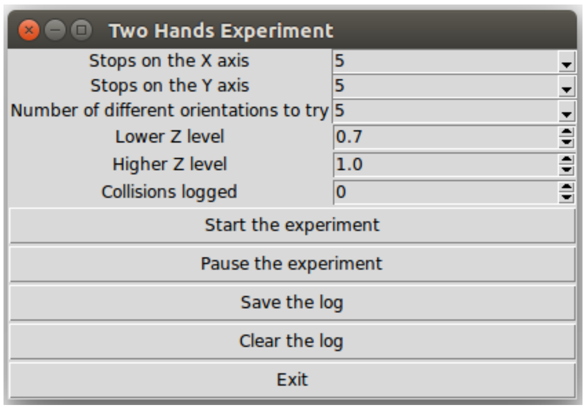


Figure 5.8: Distributions of joint angles across measured poses of dataset 2 (joints with number 1 correspond to right arm, joints with number 2 correspond to left arm).

5.3 Two Hands Experiment applet

We created a simple GUI applet to collect the dataset. The robot will collect the data over a grid defined by the given number of stops on X axis and on Y axis. Two runs through this grid in two given heights (Z axis coordinates) will be done. Finally, the robot will attempt to do several collisions in one node of the grid. The number of collisions can be specified too. The execution of the dataset collection can be paused and resumed any time, but the last started action will come to end before the robot will stop.

The applet makes use of the robot library described in Subsection 4.2.2 and the camera library described in Subsection 4.2.2. Unlike the Motion Strategy application, Two Hands Experiment applet does not use the structure of objects representing actions, instead, a simple state machine is used. The robot library calls a function of the applet itself, whenever a current action is completed. This function makes the state machine progress to the next state and issue a new action to the robot library. Fig. 5.9 shows a screenshot of the applet.


Two Hands Experiment

Stops on the X axis	5
Stops on the Y axis	5
Number of different orientations to try	5
Lower Z level	0.7
Higher Z level	1.0
Collisions logged	0

Figure 5.9: Screenshot of the Two Hands Experiment applet

Chapter 6

Experiments and Results

We carried out multiple sets of calibrations with different sets of parameters to be calibrated to evaluate how the calibration copes with various sensors being used. Each set represents one combination of cameras, arms and touch information. In each set, we carried out 11 calibrations to evaluate the stability of the calibration. Table 6.1 lists the combinations of cameras, arms and touch information we used and their abbreviations.

Configuration	Abbreviation
Left arm, left camera	LaLc
Left arm, both cameras	LaRcLc
Right arm, right camera	RaRc
Right arm, both cameras	RaRcLc
Both arms, right camera	RaLaRc
Both arms, left camera	RaLaLc
Both arms, left camera, touch	RaLaLcTd
Both arms, both cameras, touch	RaLaRcLcTd

Table 6.1: Configurations of the robot

Table 6.2 shows what parameters are being calibrated within kinematic chains with given abbreviations. If the abbreviation is, for example, RaLaRcLcTd, then the parameters are under RaLa, Rc and Lc in the table. Icosahedron radius is never calibrated. Camera parameters are always the same set, regardless of the choice of all DH or only offsets.

Certain parameters had to be excluded from the calibration due to them being badly conditioned or unobservable. For example, all the *turntable* DH parameters, d_S and θ_S are unobservable because the manipulators and

Abbrev.	only offset	DH parameters
Ra	no	$a_{L1}, \alpha_{L1}, o_{L1}, a_{U1}, d_{U1}, \alpha_{U1}, o_{U1}, a_{R1}, d_{R1}, \alpha_{R1}, o_{R1}, a_{B1}, d_{B1}, \alpha_{B1}, o_{B1}, a_{T1}, \alpha_{T1}, d_{ico1}, o_{ico1}$
La	no	$a_{L2}, \alpha_{L2}, o_{L2}, a_{U2}, d_{U2}, \alpha_{U2}, o_{U2}, a_{R2}, d_{R2}, \alpha_{R2}, o_{R2}, a_{B2}, d_{B2}, \alpha_{B2}, o_{B2}, a_{T2}, \alpha_{T2}, d_{ico2}, o_{ico2}$
RaLa	no	$a_{S1}, \alpha_{S1}, a_{L1}, \alpha_{L1}, o_{L1}, a_{U1}, d_{U1}, \alpha_{U1}, o_{U1}, a_{R1}, d_{R1}, \alpha_{R1}, o_{R1}, a_{B1}, d_{B1}, \alpha_{B1}, o_{B1}, a_{T1}, \alpha_{T1}, d_{ico1}, o_{ico1}, a_{turn2}, \alpha_{turn2}, a_{S2}, d_{S2}, \alpha_{S2}, o_{S2}, a_{L2}, \alpha_{L2}, o_{L2}, a_{U2}, d_{U2}, \alpha_{U2}, o_{U2}, a_{R2}, d_{R2}, \alpha_{R2}, o_{R2}, a_{B2}, d_{B2}, \alpha_{B2}, o_{B2}, a_{T2}, \alpha_{T2}, d_{ico2}, o_{ico2}$
Ra	yes	$o_{S1}, o_{L1}, o_{U1}, o_{R1}, o_{B1}, o_{T1}$
La	yes	$o_{S2}, o_{L2}, o_{U2}, o_{R2}, o_{B2}, o_{T2}$
RaLa	yes	$o_{S1}, o_{L1}, o_{U1}, o_{R1}, o_{B1}, o_{T1}, o_{L2}, o_{U2}, o_{R2}, o_{B2}, o_{T2}$
Rc	yes/no	$a_{turn3}, d_{turn3}, \alpha_{turn3}, o_{turn3}, d_{cam1}, o_{cam1}$
Lc	yes/no	$a_{turn4}, d_{turn4}, \alpha_{turn4}, o_{turn4}, d_{cam2}, o_{cam2}$

Table 6.2: DH parameters optimized within given kinematic chain, RaLa stands for the case when both Ra and La are used.

the cameras are all attached to the top of the turntable. We only optimize some of them when we need to lift the constraints on mutual position of the manipulators. The badly conditioned parameters are typically near-parallel rotation axes of the manipulator. By having one of the parameters fixed, we improve the conditionality of the solution, but we lose a degree of freedom of the model and the reduced set of optimization parameters may not be sufficient to explain all phenomena observed on the dataset. The L and U axes of the manipulators are stated as parallel by the manufacturer, so we fix the d_L parameter of each manipulator. The Z axis of the icosahedron is also near-parallel to the Z axis of the T joint and the offset of the T joint is badly observable because the θ of the icosahedron is a rotation almost in the same plane, so we fix the d_T and o_T parameters of each manipulator. Some parameters are also unnecessary, but we keep them for formal reasons. These are a_{cam} and α_{cam}. They are unnecessary because the position of each camera has got 6 DoF and is fully determined by 6 DH parameters.

6.1 Calibration without perturbation

This calibration starts from nominal DH parameters and calibrates the parameters of each involved kinematic chain. We carried out two sets of calibration, all DH parameters and only offsets. Note that the kinematic chain formulation in Table 6.2 applies here, so we did not truly calibrate every single DH parameter, just the greatest subset that doesn't lead to ill-conditioned optimization task.

As can be seen from Figures 6.1 and Table 6.4, attempts to calibrate one arm using only the offsets result in increased touch distance error. Even if both cameras are used, the touch distance error does not decrease for RaRcLc chain. This may be caused by the position of the arm in one-arm calibration being unobservable or badly conditioned, giving the optimization algorithm freedom to choose one of many solutions in a case where the solution should be unique. If two arms are used with one camera, the touch distance error decreases, as the calibration is able to optimize the position of the arms with relation to each other, even if the touch distance error is not used for calibration.

From Figure 6.1 and Table 6.4, it can be concluded that RaRc and LaLc are not symmetrical. This is no surprise as we noticed the asymmetry of the dataset back in the Section 5.2. The left camera of the robot consistently sees lower number of markers then the right camera. In Section 5.2, we suggested that the left camera is partially occluded by the asymmetric left robot arm, but we didn't deal with this problem into much detail.

The excessive touch distance error does not seem to appear when all DH parameters are calibrated, albeit the trend of decreasing touch distance errors stands here too. The increased degree of freedom seems to allow the RaLaRc chain to optimize the marker error for the cost of increasing touch distance errors, but the touch information in RaLaRcTd helps to achieve better optimization of touch distance errors while keeping the lowered marker error. As expected, the training error is only very slightly lower than the testing error in most of the cases, suggesting little to no over-fitting occurs during optimization. The few cases where the testing error is lower than the training error are most likely due to noise in the dataset and are statistically insignificant, considering how little the difference is and how high the standard deviations of the errors are.

In Table 6.3 we show average errors before calibration on training and testing data for markers positions and distances. The data is very similar for all kinematic chains since no perturbation was done and so we present the mean and the standard deviation for all errors over 11 repetitions across all the chains. Values for individual kinematic chains are shown in the appendix, see Chapter A), Table A.1. Most of the chains with all DH parameters optimized produce lower marker error but higher touch distance error than the nominal DH parameters with only camera chains and end effector links calibrated by our method (See Table 6.9).

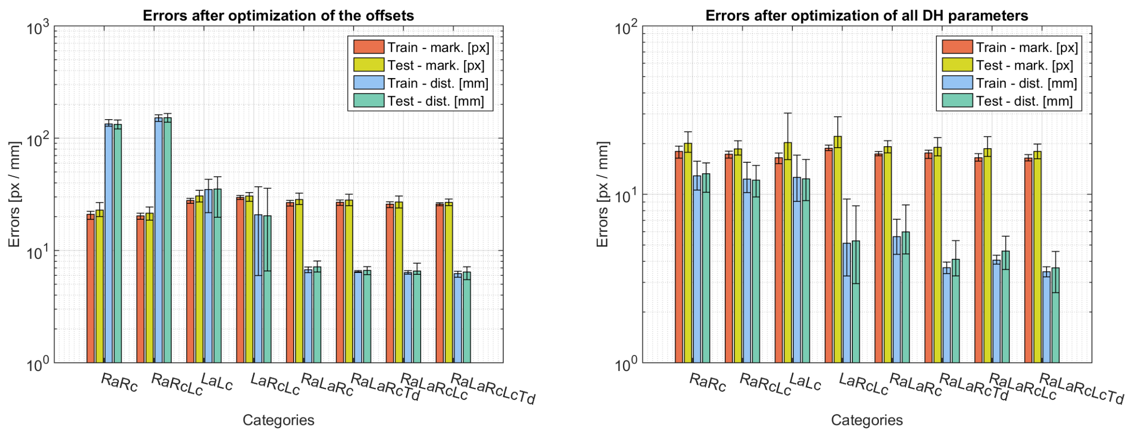


Figure 6.1: Errors after calibration of the offsets (left column) and all DH parameters (right column) of individual kinematic chains. We show mean and standard deviation (over 11 repetitions) of absolute errors of marker distances in camera frame (in pixels) and icosahedrons distances (in mm) over train and test dataset.

Absolute error		Only offsets		
		mean (std)	min	max
Markers	Train	1392 (5)	1342	1466
[px]	Test	1391 (11)	1311	1473
Distances	Train	5.73 (0.14)	5.369	6.039
[mm]	Test	5.69 (0.29)	4.928	6.451

Table 6.3: Errors before calibration, mean value, standard deviation, minimum and maximum over all kinematic chains over all repetitions.

In Fig 6.2 we compare the touch distance errors for individual chains before and after calibration. If there was a perfect calibration over a dataset with measurement errors of purely Gaussian characteristic, the histogram of errors should form a Gaussian curve with its peak in zero. A look into Fig 6.2 reveals that this approximately stands for chains RaLaRc, RaLaRcTd, RaLaRcLc and RaLaRcLcTd. For RaLaRcLcTd the peak is narrowest, which means the lowest error in touch distance. This is an expected outcome, since RaLaRcLcTd comprises the most informed and least constrained calibration. Results over multiple chains including errors of markers can be found in appendix A.

We take the icosahedron diameter of 116 mm as a ground truth because it was found that the optimization tries to compensate an error in Z coordinate of the icosahedron position by altering the diameter of the icosahedron. This is because all the touching tiles lie in one plane with relation to the icosahedron and the error of their distance from the center may be explained either by displacement of the icosahedron or by difference in its diameter.

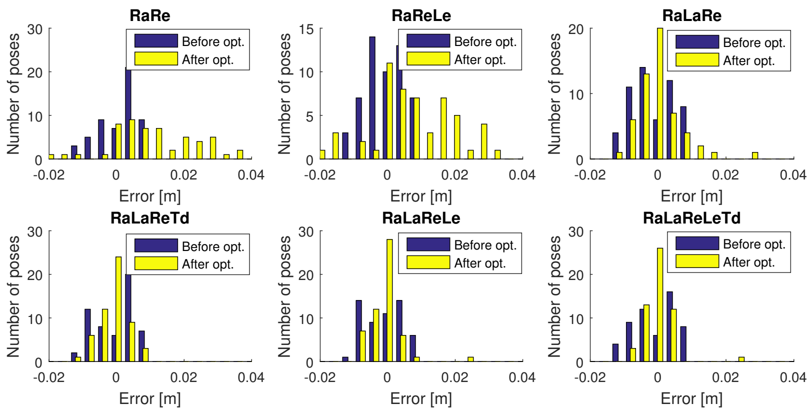


Figure 6.2: Comparison of error on distance between icosahedrons on both arms before and after calibration when individual chains are calibrated.

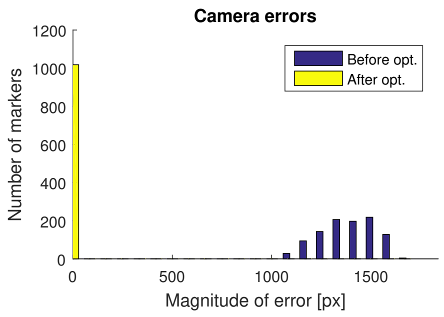


Figure 6.4: Comparison of error of marker positions in camera frame before and after calibration for RaLaRcLc kinematic chain with touch (RaLaRcLcTd).

Figure 6.4 shows the marker projection errors before and after calibration for the best kinematic chain - RaLaReLcTd. The error distributions for other chains can be found in the appendix, see Fig. A.1- A.3. As can be seen from the graphs, calibration always provides a substantial improvement since the positions of both cameras were determined during the assembly using only basic tools - ruler and protractor. For this reason, calibration improves the marker projection errors by up to two orders of magnitude.

Comparison of marker positions errors after calibration for various kinematic chains can be found in Fig. 6.5. The results are all very similar, regardless of whether the touch distance was used. There is just a bit larger variance in the optimization results in chains containing one camera, compared to chains containing both cameras.

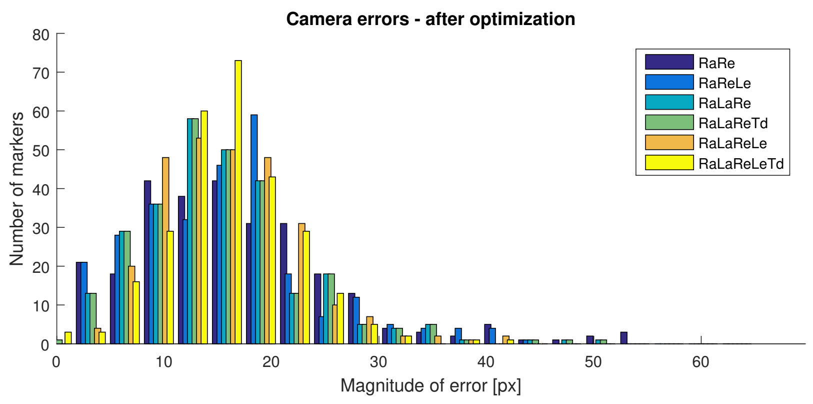


Figure 6.5: Error of marker positions in camera frame (restricted to same number of evaluated marker positions) after calibration for individual chains.

6.2 Calibration with perturbation

The parameters were perturbed before the calibration to assess the stability of the calibration algorithm. The perturbations were drawn from an uniform distribution with symmetric ranges. We chose three distinct levels of perturbation, slight, moderate and intense. Table 6.5 shows the ranges of distributions used in each perturbation level.

In Table 6.6 we show absolute errors of markers positions and distances between icosahedrons after calibration of DH parameters for multi-chain calibration (RaLaRcLcTd chain - both arms, both eyes + touch). Results for all evaluated chains are shown in appendix (see Table A.2 - A.4).

The errors in the calibrations of all DH parameters are lower than in the calibrations of offsets only, and that stands even when the parameters are perturbed. We can see some kinematic chains with moderate perturbation reach the same minimum value as the kinematic chains with slight perturbation, but the variance is much higher. For intense perturbation, the errors stay large, because none of the chains with intense perturbation converged.

Evaluation of the calibration result over various perturbation of the parameters allows us to evaluate the ability of the calibration algorithm to find the global optimum. Such behavior is not guaranteed with numerical optimization methods. Our results show that under moderate perturbation as defined in Table 6.5, the algorithm sometimes converges to the global optimum. We haven't recorded a single case of global optimum under intense perturbation, as defined in Table 6.5.

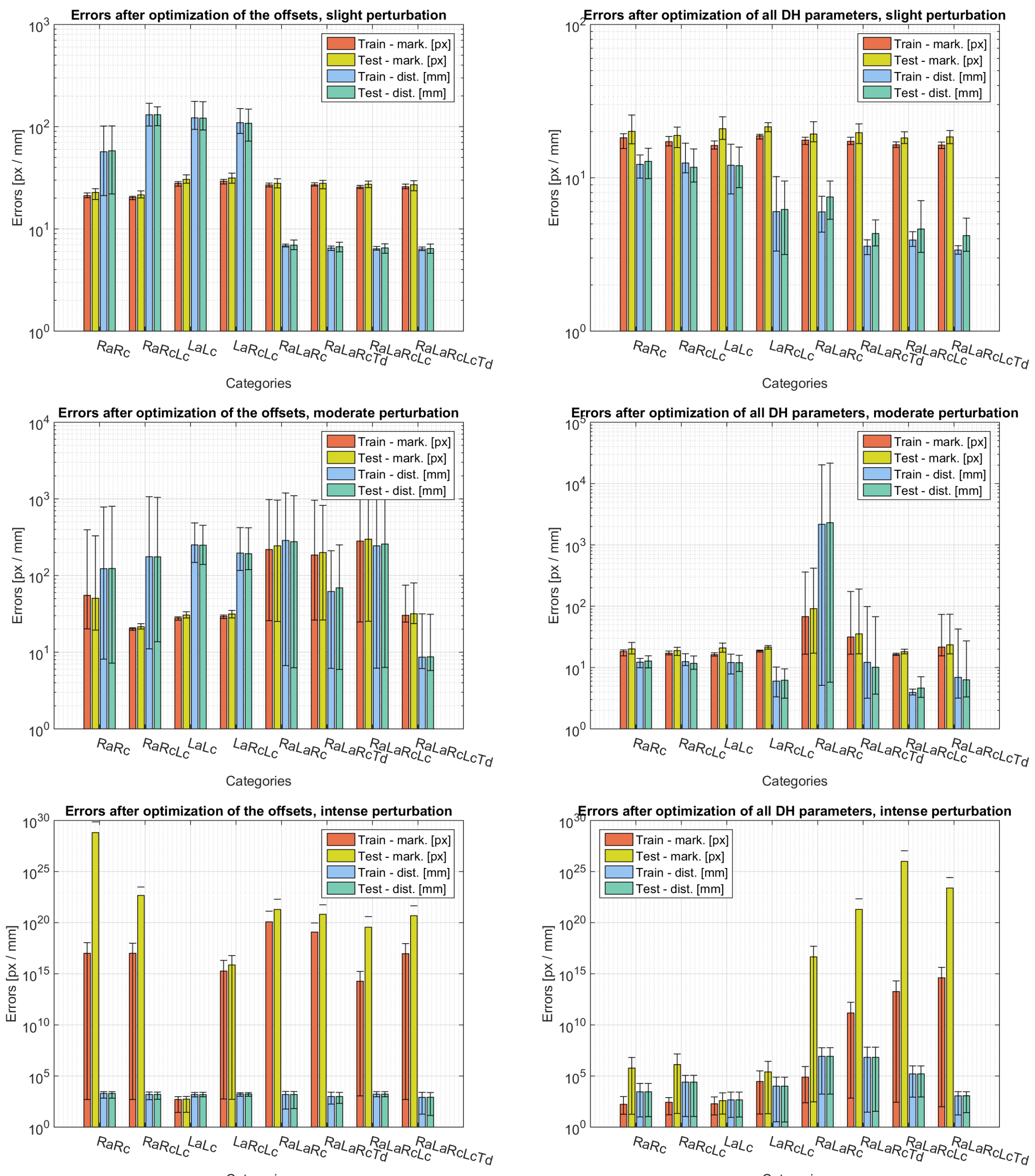


Figure 6.6: Errors after calibration of the offsets (left) and all DH parameters (right) of individual kinematic chains. Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. We show mean and standard deviation (over 11 repetitions) of absolute errors of marker distances in camera frame (in pixels) and icosahedrons distances (in mm) over train and test dataset.

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaRc							
Markers	Train	20.97 (0.92)	18.92	22.31	17.99 (0.81)	16.39	19.32
[px]	Test	22.9 (1.9)	20.04	26.72	20.1 (1.7)	17.77	23.5
Distances	Train	133.9 (5.3)	127.8	146.3	12.9 (1.6)	10.6	15.73
[mm]	Test	132.7 (7.1)	120.7	145.2	13.2 (1.5)	10.28	15.36
RaRcLc							
Markers	Train	20.3 (0.79)	18.95	21.48	17.29 (0.5)	16.41	18.04
[px]	Test	21.5 (1.8)	18.63	24.4	18.6 (1.2)	17.09	20.79
Distances	Train	151.7 (7.2)	141	161.5	12.3 (1.8)	10.24	15.49
[mm]	Test	152.1 (9.2)	138.7	165.9	12.1 (1.9)	9.646	14.85
LaLc							
Markers	Train	27.9 (1)	26.29	29.13	16.48 (0.69)	15.2	17.58
[px]	Test	30.6 (2.5)	27.01	34.32	20.3 (3.9)	16.07	30.36
Distances	Train	34.9 (6.1)	21.71	43.01	12.6 (3.1)	9.073	17.09
[mm]	Test	35.3 (7.2)	19.77	45.33	12.4 (2.7)	9.172	16.1
LaRcLc							
Markers	Train	29.66 (0.76)	28.45	30.92	18.77 (0.5)	18.17	19.6
[px]	Test	30.5 (1.7)	27.37	32.79	22.1 (2.5)	18.93	28.87
Distances	Train	20.8 (9)	5.976	36.94	5.1 (1.9)	3.275	9.374
[mm]	Test	20.4 (9)	6.558	35.81	5.3 (1.8)	2.948	8.547
RaLaRc							
Markers	Train	26.69 (0.79)	24.88	27.92	17.37 (0.36)	16.95	17.96
[px]	Test	28.4 (1.8)	25.67	32.4	19.2 (1)	17.63	20.81
Distances	Train	6.74 (0.22)	6.348	7.131	5.61 (0.84)	4.401	7.112
[mm]	Test	7.14 (0.47)	6.427	8.062	6 (1.3)	4.427	8.647
RaLaRcTd							
Markers	Train	26.86 (0.86)	25.2	28.12	17.61 (0.5)	16.33	18.28
[px]	Test	28.1 (1.9)	25.03	31.69	19 (1.4)	16.92	21.71
Distances	Train	6.459 (0.091)	6.366	6.622	3.67 (0.21)	3.383	3.959
[mm]	Test	6.64 (0.35)	6.09	7.171	4.11 (0.66)	3.277	5.31
RaLaRcLc							
Markers	Train	25.85 (0.82)	24.1	27.1	16.49 (0.61)	15.74	17.45
[px]	Test	27 (1.9)	23.9	30.55	18.7 (1.8)	16.77	22
Distances	Train	6.35 (0.12)	6.153	6.61	4.07 (0.17)	3.852	4.348
[mm]	Test	6.55 (0.42)	6.128	7.709	4.6 (0.73)	3.577	5.638
RaLaRcLcTd							
Markers	Train	25.88 (0.62)	25	26.6	16.41 (0.45)	15.78	17.21
[px]	Test	26.8 (1.4)	25.11	28.69	18 (1.2)	16.26	19.88
Distances	Train	6.22 (0.26)	5.773	6.528	3.47 (0.15)	3.234	3.714
[mm]	Test	6.43 (0.62)	5.494	7.156	3.66 (0.56)	2.607	4.579

Table 6.4: Absolute errors of markers positions and distances between icosahedrons evaluated on training and testing data after calibration of DH parameters using individual kinematic chains. We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results are averaged over 11 runs (standard deviation is shown in parenthesis).

Level	Arm a, d, α	Arm θ offset	Camera a, d, α, θ
slight	$\pm$ 0.01 m/rad	$\pm$ 0.1 m/rad	$\pm$ 0.05 m/rad
moderate	$\pm$ 0.03 m/rad	$\pm$ 0.3 m/rad	$\pm$ 0.03 m/rad
intense	$\pm$ 0.1 m/rad	$\pm$ 1 m/rad	$\pm$ 0.1 m/rad

Table 6.5: Ranges of distributions used in each perturbation level

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaLaRcLcTd		slight perturbation			slight perturbation		
Markers	Train	25.84 (0.72)	24.69	27.44	16.32 (0.55)	15.52	17.1
[px]	Test	27 (1.6)	23.62	29.6	18.5 (1.2)	16.69	20.32
Distances	Train	6.35 (0.15)	6.122	6.651	3.38 (0.11)	3.17	3.609
[mm]	Test	6.42 (0.44)	5.765	7.1	4.2 (0.62)	3.316	5.454
RaLaRcLcTd		moderate perturbation			moderate perturbation		
Markers	Train	30 (15)	24.69	74.97	22 (17)	15.52	73.73
[px]	Test	32 (16)	23.62	80.08	23 (17)	16.69	73.93
Distances	Train	8.6 (7.6)	6.122	31.66	7 (12)	3.17	42.49
[mm]	Test	8.7 (7.5)	5.765	31.3	6.3 (6.9)	3.316	27.17
RaLaRcLcTd		intense perturbation			intense perturbation		
Markers	Train	9e+16 (2.7e+17)	495.7	8.89e+17	4e+14 (1.3e+15)	98.2	4.256e+15
[px]	Test	5e+20 (1.3e+21)	580.1	4.457e+21	2.5e+23 (7.8e+23)	101.3	2.601e+24
Distances	Train	820 (810)	18.47	2490	1100 (1100)	15.42	2950
[mm]	Test	840 (800)	13.77	2416	1200 (1100)	25.74	2890

Table 6.6: Absolute errors on training and testing data after multi-chain calibration of perturbed DH parameters (RaLaRcLcTd chain). We show results for a case where only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. Results are averaged over 11 runs (standard deviation is shown in parenthesis).

6.3 Sequential calibration

In Table 6.7 we show absolute errors on testing and training datasets with parameters calibrated using sequential calibration where first arm is calibrated (starting from nominal DH parameters) using both cameras and then the multi-chain calibration on both arms and cameras is performed. Table 6.8 presents our results for sequential calibration where 3 different levels of parameters perturbation were applied to DH parameters to be calibrated (for perturbation values see Table 6.5). In B, Table B.1, shows values for the case where the right arm is calibrated first.

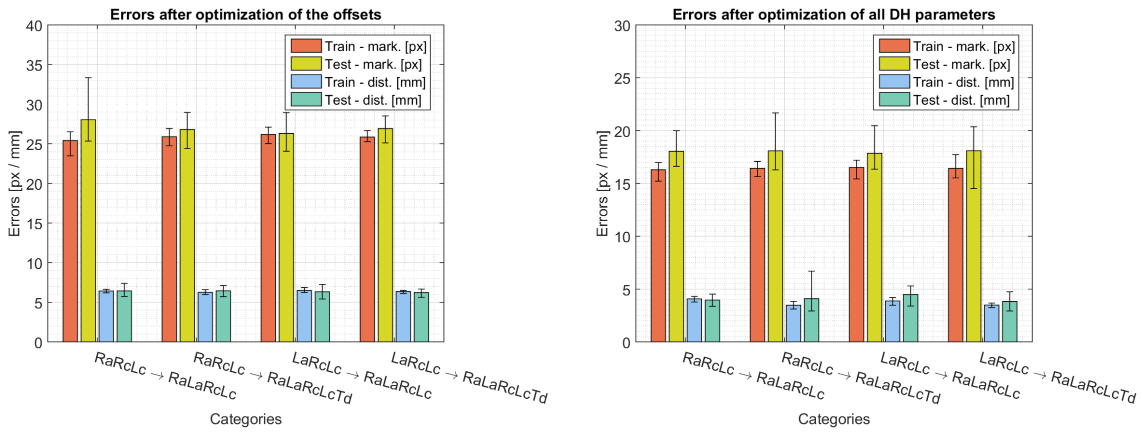


Figure 6.7: Errors after calibration of the offsets of individual kinematic chains (sequential calibration). We show mean and standard deviation (over 11 repetitions) of absolute errors of marker distances in camera frame (in pixels) and icosahedrons distances (in mm) over training and testing dataset.

As can be seen from Figure 6.7 and Table 6.7, best sequential calibration was reached with the optimization of chain LaRcLc followed by the optimization of RaLaRcLcTd with all DH parameters, which led to error of markers positions 18.1(1.6) and mean error of distances between icosahedrons 3.83(0.58) (for the best calibration we achieved error of markers positions 14.5 and mean error of distances between icosahedrons 2.92).

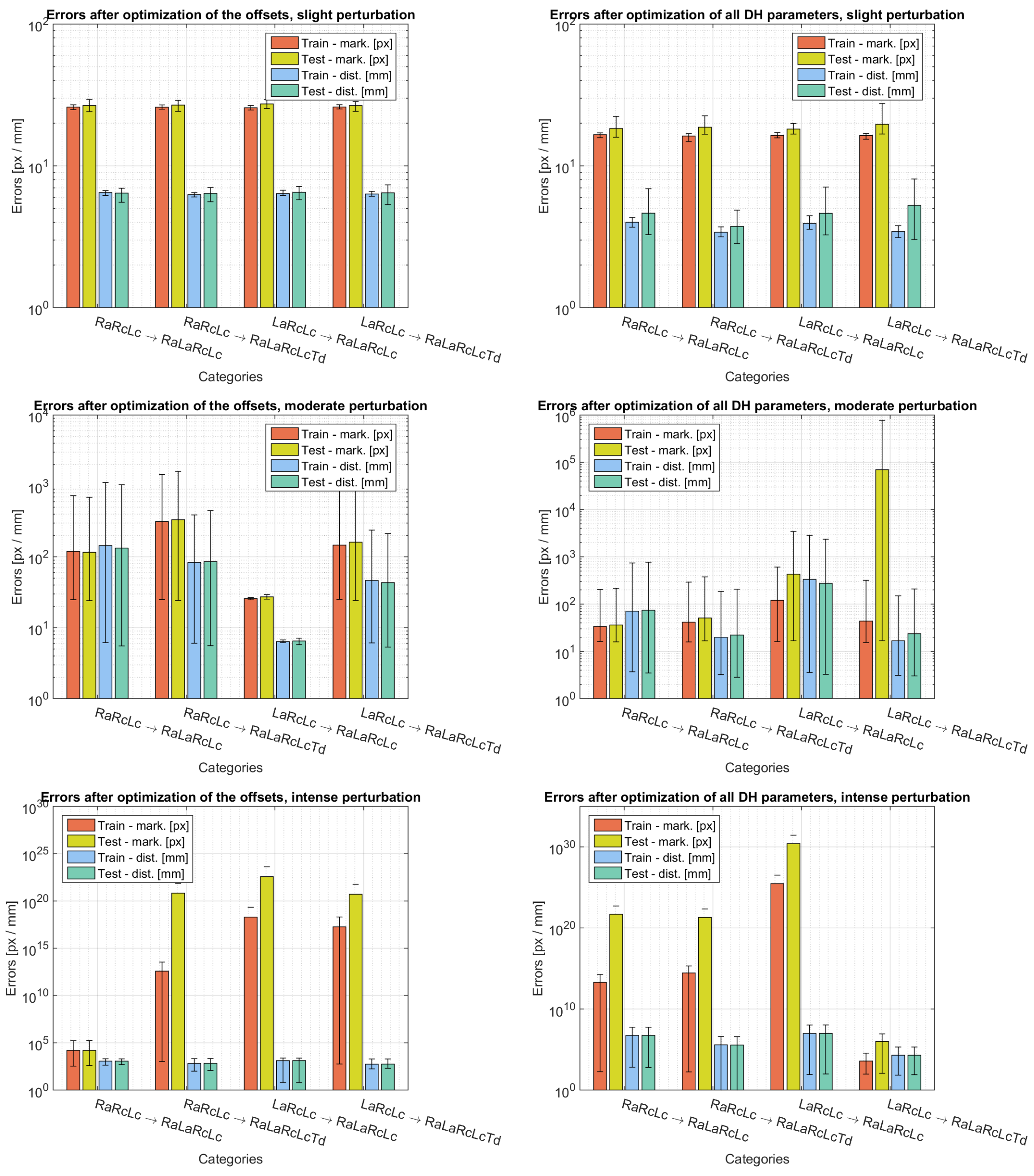


Figure 6.8: Errors after calibration of the offsets (left) and all DH parameters (right) of individual kinematic chains using sequential calibration. Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. We show mean and standard deviation (over 11 repetitions) of absolute errors of marker distances in camera frame (in pixels) and icosahedrons distances (in mm) over train and test dataset.

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaRcLc → RaLaRcLc							
Markers	Train	25.41 (0.98)	23.47	26.51	16.29 (0.47)	15.22	16.96
[px]	Test	28 (2.4)	25.34	33.33	18.04 (1)	16.61	19.98
Distances	Train	6.43 (0.14)	6.196	6.654	4.07 (0.2)	3.776	4.31
[mm]	Test	6.44 (0.48)	5.734	7.397	3.96 (0.36)	3.368	4.523
RaRcLc → RaLaRcLcTd							
Markers	Train	25.89 (0.68)	24.73	26.93	16.43 (0.4)	15.63	17.09
[px]	Test	26.8 (1.5)	24.38	28.95	18.1 (1.5)	16.27	21.67
Distances	Train	6.25 (0.17)	5.976	6.584	3.47 (0.21)	3.111	3.842
[mm]	Test	6.45 (0.49)	5.716	7.136	4.1 (1.1)	2.923	6.699
LaRcLc → RaLaRcLc							
Markers	Train	26.16 (0.68)	25.02	27.1	16.5 (0.53)	15.43	17.21
[px]	Test	26.3 (1.7)	24.06	28.93	17.9 (1.2)	16.35	20.46
Distances	Train	6.5 (0.21)	6.237	6.846	3.87 (0.24)	3.473	4.206
[mm]	Test	6.33 (0.59)	5.405	7.262	4.49 (0.58)	3.4	5.29
LaRcLc → RaLaRcLcTd							
Markers	Train	25.86 (0.48)	25.27	26.65	16.42 (0.57)	15.52	17.72
[px]	Test	26.9 (1.1)	25.1	28.51	18.1 (1.6)	14.5	20.36
Distances	Train	6.35 (0.12)	6.098	6.509	3.49 (0.14)	3.252	3.681
[mm]	Test	6.23 (0.33)	5.632	6.675	3.83 (0.58)	2.923	4.744

Table 6.7: Errors on training and testing data after sequential calibration of perturbed DH parameters where first one hand is calibrated using both eyes (LARcLc and RARcLc for left and right arm respectively) and then the gained calibration is used to calibrate the second hand and end-effector parameters using multi-chain calibration (RaLaRcLc and RaLaRcLc for calibration with and without touch respectively). We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results are averaged over 11 runs (standard deviation is shown in parenthesis).

Chain abbreviation		Only offsets			all DH parameters		
		mean (std)	min	max	mean (std)	min	max
LaRcLc → RaLaRcLc		slight perturbation			slight perturbation		
Markers	Train	25.67 (0.59)	24.78	26.65	16.4 (0.38)	15.75	17.16
[px]	Test	27.3 (1.3)	25.32	29.32	18.2 (1.1)	16.74	19.93
Distances	Train	6.38 (0.15)	6.189	6.728	3.93 (0.27)	3.57	4.451
[mm]	Test	6.51 (0.37)	5.772	7.148	4.6 (1.2)	3.263	7.094
LaRcLc → RaLaRcLc		moderate perturbation			moderate perturbation		
Markers	Train	25.67 (0.59)	24.78	26.65	120 (230)	16.11	607.7
[px]	Test	27.3 (1.3)	25.32	29.32	400 (1100)	16.74	3443
Distances	Train	6.38 (0.15)	6.189	6.728	330 (870)	3.57	2858
[mm]	Test	6.51 (0.37)	5.772	7.148	270 (720)	3.263	2361
LaRcLc → RaLaRcLc		intense perturbation			intense perturbation		
Markers	Train	2e+18 (6.4e+18)	25.09	2.14e+19	3e+25 (1e+26)	58.61	3.326e+26
[px]	Test	4e+22 (1.2e+23)	28.8	4.137e+23	2.6e+30 (8.5e+30)	62.01	2.806e+31
Distances	Train	1300 (890)	6.268	2428	1e+07 (3.2e+07)	80.96	1.068e+08
[mm]	Test	1330 (900)	6.21	2401	1e+07 (3.2e+07)	96.12	1.072e+08
LaRcLc → RaLaRcLcTd		slight perturbation			slight perturbation		
Markers	Train	26.01 (0.68)	25.17	26.97	16.37 (0.43)	15.41	16.92
[px]	Test	26.6 (1.6)	24.24	28.44	19.6 (3.3)	16.76	27.53
Distances	Train	6.33 (0.15)	6.107	6.618	3.44 (0.24)	3.116	3.788
[mm]	Test	6.46 (0.54)	5.336	7.348	5.3 (1.5)	3.025	8.083
LaRcLc → RaLaRcLcTd		moderate perturbation			moderate perturbation		
Markers	Train	150 (310)	25.17	1029	44 (91)	15.41	317.3
[px]	Test	160 (330)	24.24	1055	7e+04 (2.3e+05)	16.76	7.669e+05
Distances	Train	46 (89)	6.107	238.3	17 (44)	3.116	149.6
[mm]	Test	43 (82)	5.336	212.4	24 (61)	3.025	208.6
LaRcLc → RaLaRcLcTd		intense perturbation			intense perturbation		
Markers	Train	1.8e+17 (6.1e+17)	578	2.025e+18	4000 (1e+04)	91.83	3.54e+04
[px]	Test	5e+20 (1.7e+21)	642	5.598e+21	1e+06 (2.6e+06)	114.2	8.575e+06
Distances	Train	540 (560)	172.1	1960	2e+04 (6.2e+04)	69.05	2.055e+05
[mm]	Test	560 (560)	200.7	1948	2e+04 (6.2e+04)	79.01	2.062e+05

Table 6.8: Errors on training and testing data after sequential calibration of perturbed DH parameters where first left hand is calibrated using both eyes (LARcLc) and then this calibration is used to calibrate the second hand and end-effector parameters using multi-chain calibration (RaLaRcLcTd). We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. Results are averaged over 11 runs (standard deviation is shown in parenthesis).

6.4 Comparison to nominal DH and previous calibrations

We measured both marker projection and touch distance errors of the nominal DH description of the robot and the previous calibration described in [10]. Since our measurement of the attachment of the cameras is the main cause of marker projection error, we also list the result of calibration of the DH chains of the cameras only, while keeping the DH parameters of the arms unchanged. We list the results of our multi-chain and sequential calibration. We selected the calibration which showed best performance on the test dataset (see Table 6.4) and evaluated it on the distinct train and test dataset which was used same for all compared methods. Therefore an error which is listed in Table 6.9 is higher than the minimum value for errors listed in Table 6.4. Table 6.9 shows the errors for all compared calibrations.

6.5 Summary

The best results overall were achieved for the RaLaRcLcTd chain with 18 (std 1.2) px error of marker position and 3.66 (std 0.56) mm error of distance between the touching icosahedrons. Stability of our calibration using perturbed initial state was evaluated (see Table 6.6 and Figure 6.6). Furthermore we compared joint calibration to sequential calibration when first one hand is calibrated using cameras and this calibration is used as a starting point for a multi-chain calibration – see Table 6.8. We found out that even our best sequential calibration, LaRcLc followed by RaLaRcLcTd, did not bring any improvement in performance compared to our best joint calibration, RaLaRcLcTd. As a last step we compared our achieved results to previous calibrations and nominal calibration (see Table 6.9) where we took the nominal DH parameters of the robot and additionally calibrated cameras and end-effectors using our calibration. As can be seen, we achieved better results then the nominal calibration for markers positions, but worse results on distance, 18.3 px compared to 23.5 px and 3.80 mm compared to 1.72 mm, respectively.

Our best joint calibration, RaLaRcLcTd, produces results comparable to our best sequential calibration, LaRcLc followed by RaLaRcLcTd. This is hardly surprising considering that both are essentially the same calibration—only the latter is primed with the results of LaRcLc calibration. The joint calibration comes out a little bit better, but that is well within the margin of

Configuration	Markers	Distances
Nominal DH – Train	1395.3	5.56
Nominal DH – Test	1401.4	6.08
Nominal DH + cam. calib. – Train	25.8	5.56
Nominal DH + cam. calib. – Test	29.3	6.08
Nominal DH + cam. calib. + end eff. – Train	20.3	1.61
Nominal DH + cam. calib. + end eff. – Test	23.5	1.72
Previous calib. – Train	1400.8	6.69
Previous calib. – Test	1407.0	6.78
Previous calib. + cam. calib. - Train	26.1	6.69
Previous calib. + cam. calib. - Test	29.8	6.78
Previous calib. + cam. calib. + end eff. - Train	20.7	2.27
Previous calib. + cam. calib. + end eff. - Test	24.1	2.26
Best Multichain calib. no touch (RaLaRcLc) - Train	16.4	3.95
Best Multichain calib. no touch (RaLaRcLc) - Test	20.2	5.12
Best Multichain calib. touch (RaLaRcLcTd) - Train	16.3	3.43
Best Multichain calib. touch (RaLaRcLcTd) - Test	18.3	3.80
Best Seq. calib. no touch (LaRcLc $\rightarrow$ RaLaRcLc) - Train	19.4	7.83
Best Seq. calib. no touch (LaRcLc $\rightarrow$ RaLaRcLc) - Test	20.7	8.34
Best Seq. calib. touch (LaRcLc $\rightarrow$ RaLaRcLcTd) - Train	19.7	5.07
Best Seq. calib. touch (LaRcLc $\rightarrow$ RaLaRcLcTd) - Test	18.2	6.34

Table 6.9: Comparison of absolute error of markers position (Markers, in pixels) and absolute error of distances between icosahedrons (Distances, *mm*) on a subset of poses from a dataset for individual settings. Nominal parameters denote parameters provided by a manufacturer Table 3.1, previous calibration is calibration done in [10] with camera DH parameters determined by a measurement during our attachment of the cameras. We performed an additional camera calibrations on top of these calibrations using a hand eye calibration. End effector calibration was provided from our multi-chain calibration. For multi-chain calibration and sequential calibration, we selected the best calibration found over 11 runs, based on performance on the testing dataset, and evaluated it on the distinct train and test dataset which was retained for all compared methods.

error. For RaLaRcLcTd, the marker position error is 18 (1.2) px and the error on distances is 3.66 (0.56) mm on test dataset. For sequential calibration, we achieved the best results for LaRcLc followed by RaLaRcLcTd, with error of marker positions 18.1(1.6) px and error on distances 3.83 (0.58) mm (for full results see Table 6.4 and Table 6.7 for multi-chain and sequential calibration results respectively. Thus, we didn't manage to improve upon the results of joint calibration with sequential calibration.

Chapter 7

Conclusion and Discussion

7.1 Conclusion

In order to perform self-calibration experiments, we modified a two-arm robotic platform by adding: (i) custom self-touch end effectors with fiducial markers (see Fig. 2.3), and (ii) two photo cameras (see Fig. 2.2). The robotic platform was configured to operate with the new end effectors in a safe self-collision ("self-touch") mode relying on force/torque sensors. To this end, a high-level Python library for self-touch using force sensors was developed, which enables safe contact behavior together with a graphical application for planning and execution of complex motion plans (see Section 4.2).

We carried out calibration of the robot using a joint dataset that we collected on the robot. We evaluated the calibration of DH parameters using individual kinematic chains starting from nominal DH parameters. Best results were achieved with the least constrained and most informed model available—that is, with all the kinematic chains intersecting and all parameters subject to calibration. Stability of our calibration using perturbed initial state was evaluated and found acceptable. Furthermore we compared joint calibration to sequential calibration during which one arm of the robot is first calibrated using cameras only; this calibration is then used as a starting point for a multi-chain calibration. We found out that even our best sequential calibration did not bring any improvement in performance compared to our best joint calibration (when all parameters are subject to optimization at once). As a last step, we compared the results achieved to previous calibrations and nominal (factory) calibration of the platform. Here we took the nominal DH parameters of the robot and additionally calibrated the newly added cameras and custom end-effectors using our calibration. Our calibration achieved better results than the nominal calibration on the markers positions, but worse results on the touch distance.

7.2 Discussion and Future work

We believe we didn't do everything we could to improve on the result of sequential calibration. The results could possibly be further improved if each of the arms was first calibrated separately and then both arms calibrated again jointly. Also, more repetitions of the optimization procedure might be beneficial for obtaining a better result.

In the calibration of one arm and one or two cameras, the touch distance error seems to grow because the task is not sufficiently constrained. We would like to find a set of DH parameters that constrain the robot model so that the touch distance error lowers.

In our robot model, we used traditional DH notation and we ran into some issues with parallel neighboring rotational axes. We had to remove one of the DH parameters from the optimization set, which constrained our model and possibly contributed to the decreased performance of our calibration. Hollerbach et al. [1] suggest to use the Hayati notation, which adds another rotation β around the y axis to the DH parameters. With rotational joints, one may choose whether to describe each revolute joint with a, d, α and θ or a, α, β and θ parameters. Prismatic joints require all five parameters but with a constraint, so that every link in Hayati notation has 4 degrees of freedom, just like a link in DH notation.

The similarity of results with and without touch distances in Figures 6.1 and 6.7 suggests that the self-touch errors were not properly weighted in the objective function. The weight coefficient of the touch distance error seems to be too low, because the touch distance did not take enough influence on the result.

It can be seen that the error counted on independent testing data (Table 6.9) is very different from the error listed in the Table 6.4, which suggests that the training dataset may not sufficiently representative and the calibration may be over-fitted on the marker positions and doesn't sufficiently generalize for the touch distances of the icosahedrons. A bigger, more representative dataset would be required and the objective function may put more weight on the touch distances.

We also had a signigicantly lower amount of datapoints for the touch distances than we had for the marker projections, because every touching configuration provided about 20 marker projections but only one touch distance. This should have been taken into account, for example by weighting each of the datapoints with the reciprocal of their total count. The distance error, unlike marker error, after optimization strongly depends on the set of kinematic chains used for calibration, even among the chains that do use touch distances in the objective function, as seen in Fig. 6.5. This may be another clue that the distance error didn't have enough weight in the objective function.

In future work, a better dataset should be collected that better excites all the joints, possibly employing autonomous search for self-touching poses. As seen in the Histograms 5.4 and 5.8, the $R1$ and $R2$ joints were not sufficiently excited in the current dataset. The results of the calibrations shall then be checked against the nominal DH parameters. The largest differences would appear on the poorly excited joints.

The distribution of errors in 3D space may be beneficial. We might be able to tell in which poses the errors are largest. The overabundance of cameras might hinder the calibration performance instead of aiding it.

As we have shown in Section 4.3.1, the response of our control program may not fast enough to stop the robot in the contact configuration with sufficient precision. Further slowing down the self-touching approach may not bring the desired precision. We may need to develop a faster program in a low-level programming language.

Bibliography

[1] J. Hollerbach, W. Khalil, and M. Gautier, "Model identification," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 113–138, Springer, 2nd ed., 2016.

[2] Z. Kukelova, J. Heller, and T. Pajdla, "Hand-eye calibration without hand orientation measurement using minimal solution," in *Asian Conference on Computer Vision*, pp. 576–589, Springer, 2012.

[3] O. Birbach, U. Frese, and B. Bäuml, "Rapid calibration of a multi-sensorial humanoid's upper body: An automatic and self-contained approach," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 420–436, 2015.

[4] D. J. Bennett, D. Geiger, and J. M. Hollerbach, "Autonomous robot calibration for hand-eye coordination," *The International journal of robotics research*, vol. 10, no. 5, pp. 550–559, 1991.

[5] M. Hersch, E. Sauser, and A. Billard, "Online learning of the body schema," *International Journal of Humanoid Robotics*, vol. 5, no. 02, pp. 161–181, 2008.

[6] A. Roncone, M. Hoffmann, U. Pattacini, and G. Metta, "Automatic kinematic chain calibration using artificial skin: self-touch in the iCub humanoid robot," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2305–2312, 2014.

[7] Q. Li, R. Haschke, and H. Ritter, "Towards body schema learning using training data acquired by continuous self-touch," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pp. 1109–1114, IEEE, 2015.

[8] P. Beneš, M. Valášek, Z. Šika, V. Bauma, and V. Hamrle, “Experiments with redundant parallel calibration and measuring machine redcam,” 2007.

[9] J. Volech, L. Mráz, Z. Šika, and M. Valášek, “Concepts of robot accuracy enhancement by integrated redundant measurements.,” *Bulletin of Applied Mechanics*, vol. 9, no. 33, 2013.

[10] V. Petrík and V. Smutnỳ, “Comparison of calibrations for the CloPeMa robot,” 2014.

[11] K. Stepanova and M. Hoffmann, “Robot self-calibration using multiple kinematic chains,” in *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*, 2018. [submitted].

[12] CloPeMa, “CloPeMa Home.” `http://clopemaweb.felk.cvut.cz/clothes-perception-and-manipulation-clopema-home/`. Accessed: 2018-05-18.

[13] RadioRoSo, “RadioRoSo Home.” `http://echord.eu/radioroso/`. Accessed: 2018-05-18.

[14] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis, “Folding clothes autonomously: A complete pipeline,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1461–1478, 2016.

[15] CloPeMa, “CloPeMa: The Robot.” `http://clopemaweb.felk.cvut.cz/the-robot/`. Accessed: 2018-05-18.

[16] Open Source Robotics Foundation, “About ROS.” `http://www.ros.org/about-ros/`. Accessed: 2018-05-18.

[17] Open Source Robotics Foundation, “Actionlib package summary.” `http://wiki.ros.org/actionlib`. Accessed: 2018-05-18.

[18] I. A. Sucan and S. Chitta, “Moveit!.” `https://moveit.ros.org/`. Accessed: 2018-05-18.

[19] Open Source Robotics Foundation, “Rviz package summary.” `http://wiki.ros.org/rviz`. Accessed: 2018-05-18.

[20] CloPeMa, “Clopema workspace installation.” `http://clopema.felk.cvut.cz/redmine/projects/clopema/wiki/ROS_Installation`. Accessed: 2018-05-18.

[21] OpenCV team, “About OpenCV.” `https://opencv.org/about.html`. Accessed: 2018-05-18.

[22] H. Figuière, “gphoto home.” `http://www.gphoto.org/`. Accessed: 2018-05-18.

[23] YASKAWA Europe GmbH, "Motoman: Product view - MA1400-4." `http://www.motoman.cz/cs/produkty/roboty/product-view/?tx_catalogrobot_pi1%5Buid%5D=2499&cHash=e9e2392124a15a5cf9d2ace0a65545e9`. Accessed: 2018-05-31.

[24] F. Puciow, "Our gitlab repository (FEE CTU account required)." `https://gitlab.fel.cvut.cz/body-schema/code-selfcalib-motoman`. Accessed: 2018-05-18.

[25] OpenCV team, "Camera calibration and 3D reconstruction." `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findcirclesgrid`. Accessed: 2018-05-28.

[26] OpenCV team, "Detection of ArUco markers." `https://docs.opencv.org/3.3.0/d5/dae/tutorial_aruco_detection.html`. Accessed: 2018-05-18.

[27] F. Balint-Benczedi, "iai_photo." `https://github.com/code-iai/iai_photo`. Accessed: 2018-05-18.

[28] P. Roan, B. Pitzer, J. Vasquez, *et al.*, "bosch_drivers." `http://wiki.ros.org/bosch_drivers`. Accessed: 2018-05-18.

[29] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control.* Wiley New York, 2006.

[30] T. Werner, "Optimalizace - study material of the A4B33OPT course (in czech)." `https://cw.fel.cvut.cz/old/_media/courses/a4b33opt/opt.pdf`. Accessed: 2018-05-18.

Appendix A

Multichain calibration - additional results

A.1 Calibration without perturbation

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaRc							
Markers	Train	1379 (7.4)	1365	1393	1379 (7.4)	1365	1393
[px]	Test	1380 (17)	1347	1411	1380 (17)	1347	1411
Distances	Train	5.78 (0.11)	5.599	5.922	5.78 (0.11)	5.599	5.922
[mm]	Test	5.59 (0.26)	5.229	5.998	5.59 (0.26)	5.229	5.998
RaRcLc							
Markers	Train	1407 (2.4)	1405	1413	1407 (1.6)	1405	1410
[px]	Test	1408 (5.2)	1396	1414	1408 (3.7)	1402	1413
Distances	Train	5.66 (0.13)	5.488	5.939	5.75 (0.16)	5.499	5.975
[mm]	Test	5.86 (0.3)	5.186	6.23	5.64 (0.37)	5.089	6.208
LaLc							
Markers	Train	1457 (5.4)	1449	1466	1457 (5.4)	1449	1466
[px]	Test	1456 (12)	1436	1473	1456 (12)	1436	1473
Distances	Train	5.77 (0.1)	5.584	5.893	5.77 (0.1)	5.584	5.893
[mm]	Test	5.72 (0.24)	5.434	6.142	5.72 (0.24)	5.434	6.142
LaRcLc							
Markers	Train	1383 (1.7)	1380	1385	1385 (2.9)	1382	1390
[px]	Test	1390 (4.1)	1386	1397	1387 (6.4)	1376	1394
Distances	Train	5.67 (0.17)	5.401	5.935	5.756 (0.094)	5.546	5.835
[mm]	Test	5.77 (0.4)	5.13	6.352	5.58 (0.22)	5.391	6.056
RaLaRc							
Markers	Train	1357 (9.1)	1343	1375	1359 (3.7)	1354	1366
[px]	Test	1351 (21)	1310	1382	1346 (9.3)	1326	1357
Distances	Train	5.68 (0.12)	5.406	5.821	5.72 (0.13)	5.57	5.928
[mm]	Test	5.82 (0.25)	5.488	6.381	5.71 (0.3)	5.222	6.051
RaLaRcTd							
Markers	Train	1357 (9.4)	1342	1373	1358 (7.2)	1345	1370
[px]	Test	1349 (22)	1312	1383	1349 (16)	1321	1378
Distances	Train	5.7 (0.13)	5.435	5.891	5.79 (0.13)	5.54	6.026
[mm]	Test	5.75 (0.28)	5.315	6.324	5.56 (0.32)	4.961	6.114
RaLaRcLc							
Markers	Train	1398 (2.4)	1393	1401	1397 (2.3)	1394	1401
[px]	Test	1395 (5.5)	1389	1407	1397 (5)	1389	1405
Distances	Train	5.68 (0.11)	5.483	5.808	5.8 (0.11)	5.589	5.989
[mm]	Test	5.81 (0.23)	5.52	6.229	5.54 (0.25)	5.06	6.011
RaLaRcLcTd							
Markers	Train	1398 (1.6)	1395	1401	1398 (2.6)	1394	1402
[px]	Test	1395 (3.5)	1389	1401	1395 (6.1)	1386	1404
Distances	Train	5.69 (0.23)	5.369	6.039	5.77 (0.12)	5.473	5.892
[mm]	Test	5.77 (0.51)	4.923	6.451	5.59 (0.27)	5.313	6.249

Table A.1: Errors before calibration when only offsets are calibrated (left table) and when all DH parameters are calibrated (right table)

A.2 Visualisations of touch distances and markers errors in camera frame - no perturbation

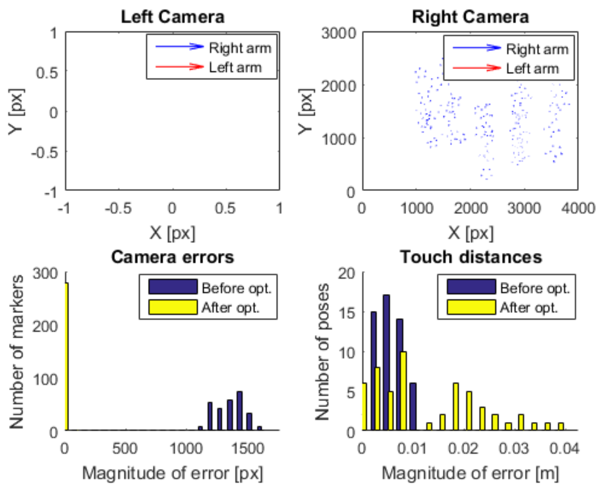


Figure A.1: Visualisation of calibration results for RaRe chain (right arm and right eye). We visualise calibration results of camera markers in right/left camera (each arrow points from the estimated position by a model to the observed position of the marker in camera), distribution of absolute errors of markers positions and distribution of absolute error of distance position.

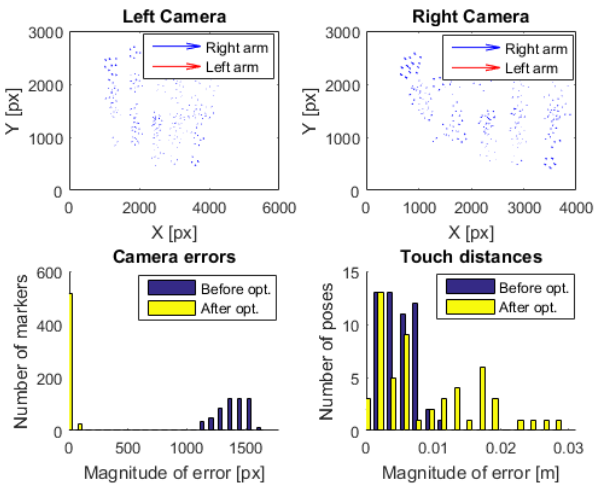
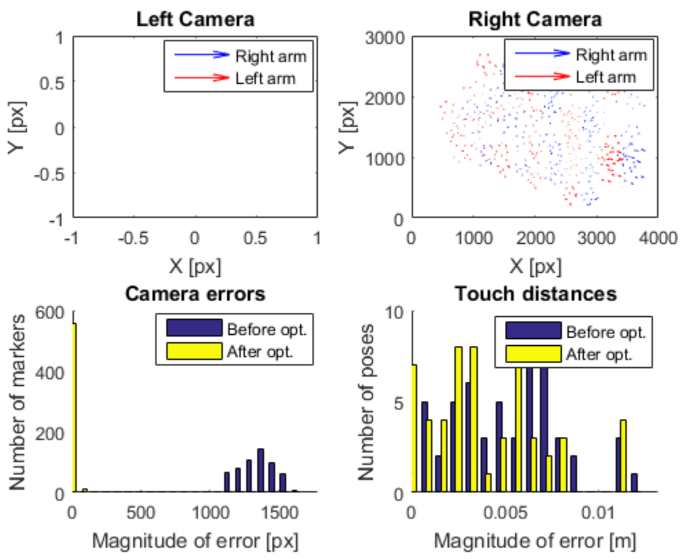
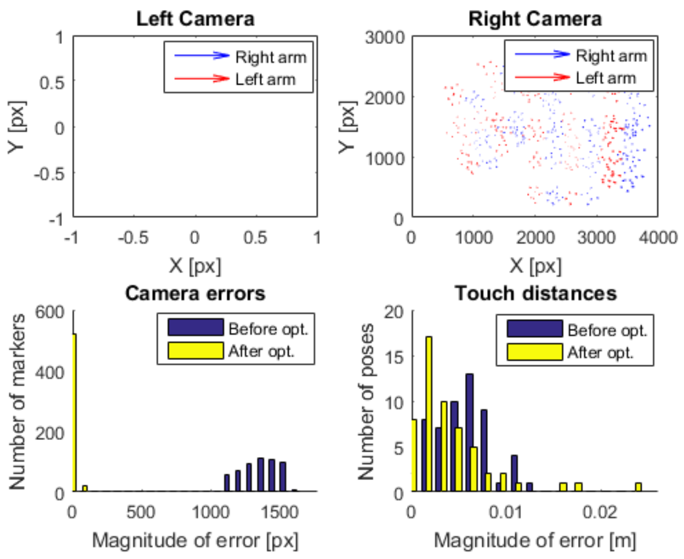


Figure A.2: Visualisation of calibration results for RaReLe chain (right arm and both eyes). We visualise calibration results of camera markers in right/left camera (each arrow points from the estimated position by a model to the observed position of the marker in camera), distribution of absolute errors of markers positions and distribution of absolute error of distance position.

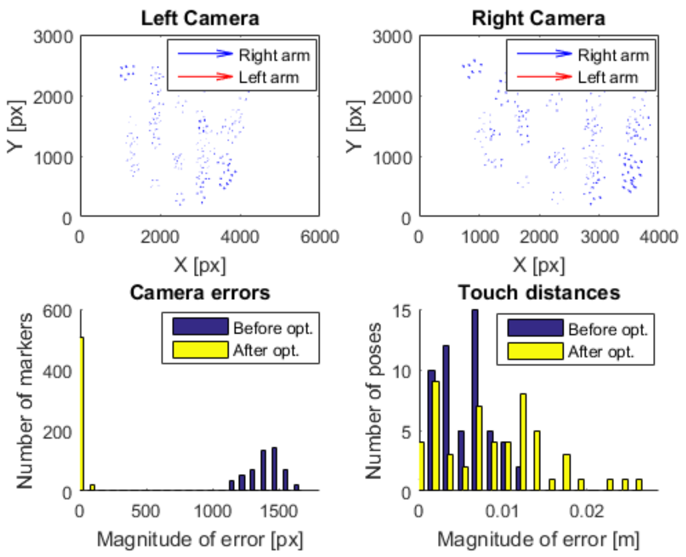


(a) : RaLaRe chain without touch

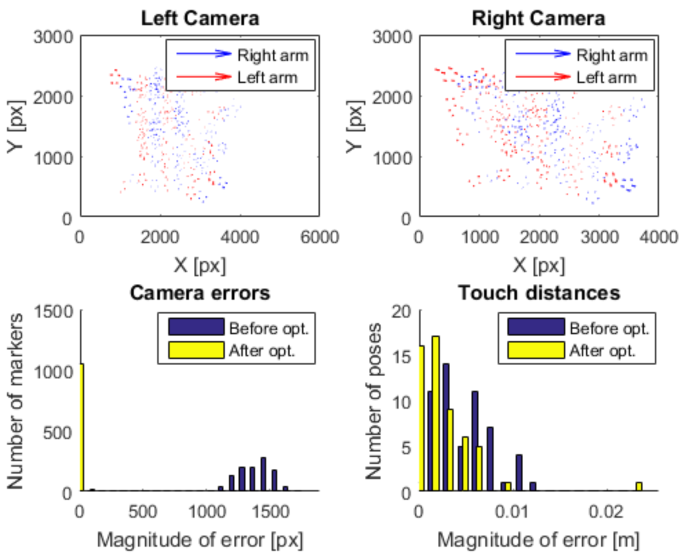


(b) : RaLaRe chain with touch

Figure A.3: Visualisation of calibration results for RaLaRe chain (both arms and right eyes). We visualise calibration results of camera markers in right/left camera (each arrow points from the estimated position by a model to the observed position of the marker in camera), distribution of absolute errors of markers positions and distribution of absolute error of distance position. In subfigure A.3a are visualised results without touch and in the subfigure A.3b are visualised results for a case where we used also touch for calibration.



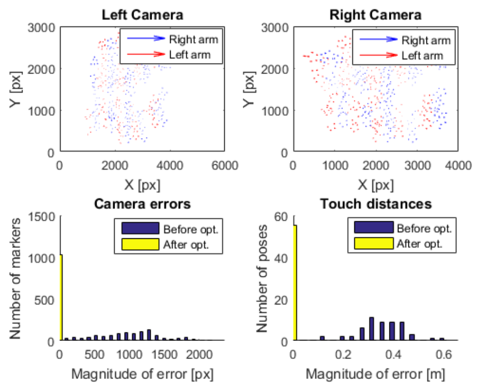
(a) : RaLaReLe chain without touch



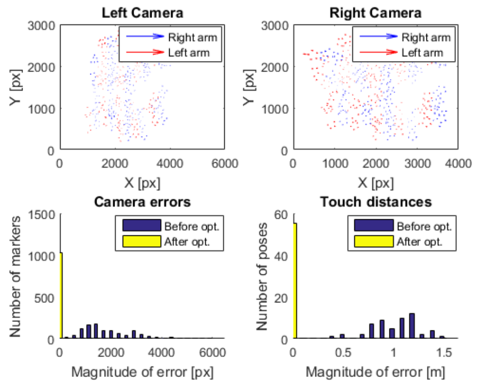
(b) : RaLaReLe chain with touch

Figure A.4: Visualisation of calibration results for RaLaReLe chain (both arms and both eyes). We visualise calibration results of camera markers in right/left camera (each arrow points from the estimated position by a model to the observed position of the marker in camera), distribution of absolute errors of markers positions and distribution of absolute error of distance position. In subfigure **??** are visualised results without touch and in the subfigure A.4b are visualised results for a case where we used also touch for calibration.

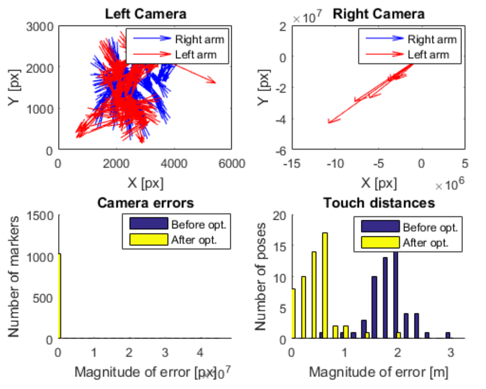
A.3 Calibration with perturbation



(a) : RaLaRe chain without touch - - mild perturbation



(b) : RaLaRe chain with touch



(c) : RaLaRe chain with touch - stormy perturbation

Figure A.5: Visualisation of calibration results for RaLaReLe chain (both arms and both eyes) with 3 levels of initial perturbation of DH parameters.

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaRc		slight perturbation			slight perturbation		
Markers	Train	21.14 (0.68)	20.15	22.47	18.2 (1.1)	15.5	19.37
[px]	Test	22.7 (1.7)	19.4	24.72	20.1 (2.5)	16.65	25.67
Distances	Train	57 (25)	21.19	101.5	12.2 (1.5)	9.945	14.1
[mm]	Test	58 (25)	21.95	101.8	12.8 (1.5)	9.872	15.57
RaRc		moderate perturbation			moderate perturbation		
Markers	Train	60 (110)	20.15	395.3	18.2 (1.1)	15.5	19.37
[px]	Test	51 (93)	19.4	329.5	20.1 (2.5)	16.65	25.67
Distances	Train	120 (240)	8.112	783.3	12.2 (1.5)	9.945	14.1
[mm]	Test	120 (240)	7.205	801.9	12.8 (1.5)	9.872	15.57
RaRc		intense perturbation					
Markers	Train	1e+17 (3.4e+17)	494.9	1.116e+18	170 (280)	18.15	971.8
[px]	Test	7e+28 (2.2e+29)	511.6	7.172e+29	6e+05 (2e+06)	18.05	6.524e+06
Distances	Train	1980 (690)	683.3	2925	2800 (6200)	9.945	1.87e+04
[mm]	Test	1980 (690)	668.9	2929	2800 (6200)	10.52	1.891e+04
RaRcLc		slight perturbation					
Markers	Train	20.31 (0.47)	19.31	20.85	17.2 (0.76)	16.12	18.6
[px]	Test	21.5 (1.1)	20.06	23.53	18.9 (1.8)	15.74	21.41
Distances	Train	131 (21)	101.6	169.3			
[mm]	Test	131 (20)	102.2	156.3			
RaRcLc		slight perturbation					
Markers	Train [px]	Test Distances	Train [mm]	Test	11.7 (1.7)	9.369	15.42
RaRcLc		moderate perturbation					
Markers	Train	20.31 (0.47)	19.31	20.85	17.2 (0.76)	16.12	18.6
[px]	Test	21.5 (1.1)	20.06	23.53	18.9 (1.8)	15.74	21.41
Distances	Train	180 (310)	11.06	1067	12.5 (2.1)	10.76	16.81
[mm]	Test	180 (300)	13.69	1049	11.7 (1.7)	9.369	15.42
RaRcLc		intense perturbation					
Markers	Train	1e+17 (2.9e+17)	496.4	9.704e+17	270 (260)	16.19	773.4
[px]	Test	5e+22 (1e+23)	449.4	3.061e+23	1.3e+06 (4.3e+06)	21.41	1.441e+07
Distances	Train	1570 (780)	481.6	2688	2.5e+04 (4.4e+04)	10.79	1.147e+05
[mm]	Test	1570 (780)	523.1	2656	2.5e+04 (4.4e+04)	11.05	1.158e+05
LaLc		slight perturbation					
Markers	Train	27.9 (0.8)	26.18	28.97	16.24 (0.59)	15.38	17.4
[px]	Test	30.5 (1.8)	27.97	33.78	20.9 (2.4)	17.78	25
Distances	Train	122 (29)	94.05	176.9	12.1 (3)	7.843	16.54
[mm]	Test	121 (31)	92.8	175.6	12 (2.4)	8.63	15.87
LaLc		moderate perturbation			moderate perturbation		
Markers	Train	27.9 (0.8)	26.18	28.97	16.24 (0.59)	15.38	17.4
[px]	Test	30.5 (1.8)	27.97	33.78	20.9 (2.4)	17.78	25
Distances	Train	251 (92)	148.2	486.3	12.1 (3)	7.843	16.54
[mm]	Test	249 (84)	139.4	452			

Table A.2: Absolute errors on training and testing data after calibration of perturbated DH parameters using multiple individual chains. We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. Results are averaged over 10 runs (standard deviation is shown in parenthesis).

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
LaLc		intense perturbation			intense perturbation		
Markers	Train	500 (350)	27.62	955.7	190 (330)	15.59	893.5
[px]	Test	550 (370)	27.97	995	390 (760)	18.63	2246
Distances	Train	1510 (470)	901.7	2441	460 (910)	8.874	2664
[mm]	Test	1510 (480)	920.3	2533	460 (910)	9.7	2661
LaRcLc		slight perturbation			slight perturbation		
Markers	Train	29.3 (1)	27.51	30.53	18.79 (0.4)	17.89	19.22
[px]	Test	31.5 (2.3)	28	35.1	21.5 (0.97)	19.98	22.87
Distances	Train	109 (24)	85.86	150.4	6 (2.1)	3.324	10.19
[mm]	Test	108 (25)	72.15	148.4	6.2 (2)	3.161	9.529
LaRcLc		moderate perturbation			moderate perturbation		
Markers	Train	29.3 (1)	27.51	30.53	18.79 (0.4)	17.89	19.22
[px]	Test	31.5 (2.3)	28	35.1	21.5 (0.97)	19.98	22.87
Distances	Train	196 (90)	116.7	422.8	6 (2.1)	3.324	10.19
[mm]	Test	193 (87)	119.3	420.2	6.2 (2)	3.161	9.529
LaRcLc		intense perturbation			intense perturbation		
Markers	Train	1.9e+15 (6.2e+15)	560.9	2.04e+16	2.9e+04 (9.5e+04)	18.88	3.149e+05
[px]	Test	7e+15 (1.9e+16)	523.9	6.024e+16	2.5e+05 (8.1e+05)	20.37	2.691e+06
Distances	Train	1700 (490)	1046	2331	1e+04 (2.2e+04)	3.324	7.6e+04
[mm]	Test	1720 (490)	1050	2386	1e+04 (2.2e+04)	3.161	7.625e+04
RaLaRc		slight perturbation			slight perturbation		
Markers	Train	26.95 (0.73)	25.72	28.05	17.6 (0.65)	16.5	18.39
[px]	Test	27.9 (1.7)	25.17	31	19.3 (1.8)	17.17	23.18
Distances	Train	6.83 (0.13)	6.662	7.093	5.99 (0.88)	4.416	7.573
[mm]	Test	6.93 (0.49)	6.266	7.781	7.5 (1.4)	5.361	9.53
RaLaRc		moderate perturbation			moderate perturbation		
Markers	Train	220 (310)	25.72	980.2	70 (120)	16.5	361.3
[px]	Test	240 (330)	25.17	965.3	90 (160)	17.17	417.5
Distances	Train	290 (440)	6.692	1195	2200 (6100)	5.115	2.023e+04
[mm]	Test	280 (410)	6.266	1099	2300 (6500)	5.764	2.154e+04
RaLaRc		intense perturbation			intense perturbation		
Markers	Train	1.2e+20 (4e+20)	687.6	1.324e+21	8e+04 (2.5e+05)	239.1	8.383e+05
[px]	Test	2e+21 (5.8e+21)	714.6	1.947e+22	5e+16 (1.5e+17)	296.5	4.966e+17
Distances	Train	1550 (900)	58.63	3065	8e+06 (1.8e+07)	1676	5.866e+07
[mm]	Test	1570 (920)	65.61	2998	8e+06 (1.8e+07)	1677	5.875e+07
RaLaRcTd		slight perturbation			slight perturbation		
Markers	Train	27.01 (0.61)	26.26	28.25	17.29 (0.65)	16.49	18.38
[px]	Test	27.9 (1.5)	24.65	29.82	19.7 (1.8)	16.73	22.47
Distances	Train	6.49 (0.19)	6.154	6.79	3.58 (0.27)	3.154	3.937
[mm]	Test	6.69 (0.39)	5.961	7.401	4.33 (0.6)	3.592	5.311

Table A.3: Absolute errors on training and testing data after calibration of perturbated DH parameters using multiple individual chains. We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. Results are averaged over 10 runs (standard deviation is shown in parenthesis).

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaLaRcTd		moderate perturbation			moderate perturbation		
Markers	Train	190 (290)	26.26	960.9	31 (47)	16.49	174.1
[px]	Test	200 (280)	26.32	823.7	35 (51)	16.73	190.5
Distances	Train	62 (90)	6.154	210.6	12 (29)	3.154	98.51
[mm]	Test	70 (100)	5.961	251.1	10 (19)	3.683	67.45
RaLaRcTd		intense perturbation			intense perturbation		
Markers	Train	1.2e+19 (3e+19)	437.8	9.429e+19	1.5e+11 (4.9e+11)	692.1	1.621e+12
[px]	Test	7e+20 (1.7e+21)	456.2	5.635e+21	2e+21 (6.6e+21)	700.9	2.173e+22
Distances	Train	980 (900)	178.3	2608	7e+06 (2e+07)	29.04	6.594e+07
[mm]	Test	980 (860)	206.5	2455	7e+06 (2e+07)	34.93	6.601e+07
RaLaRcLc		slight perturbation			slight perturbation		
Markers	Train	25.67 (0.59)	24.78	26.65 16.4 (0.38)	15.75	17.16	
[px]	Test	27.3 (1.3)	25.32	29.32	18.2 (1.1)	16.74	19.93
Distances	Train	6.38 (0.15)	6.189	6.728	3.93 (0.27)	3.57	4.451
[mm]	Test	6.51 (0.37)	5.772	7.148	4.6 (1.2)	3.263	7.094
RaLaRcLc		moderate perturbation			moderate perturbation		
Markers	Train	280 (500)	24.78	1661	16.4 (0.38)	15.75	17.16
[px]	Test	300 (500)	25.32	1599	18.2 (1.1)	16.74	19.93
Distances	Train	240 (620)	6.189	2060	3.93 (0.27)	3.57	4.451
[mm]	Test	260 (640)	6.318	2110	4.6 (1.2)	3.263	7.094
RaLaRcLc		intense perturbation			intense perturbation		
Markers	Train	1.9e+14 (5.1e+14)	1114	1.698e+15	1.8e+13 (6.1e+13)	273.4	2.019e+14
[px]	Test	4e+19 (1.2e+20)	1095	3.928e+20	1e+26 (3.3e+26)	365.9	1.096e+27
Distances	Train	1660 (660)	915.5	2921	1.6e+05 (3.6e+05)	850.3	9.733e+05
[mm]	Test	1670 (660)	932	2923	1.6e+05 (3.6e+05)	882.8	9.686e+05
RaLaRcLcTd		slight perturbation			slight perturbation		
Markers	Train	25.84 (0.72)	24.69	27.44	16.32 (0.55)	15.52	17.1
[px]	Test	27 (1.6)	23.62	29.6	18.5 (1.2)	16.69	20.32
Distances	Train	6.35 (0.15)	6.122	6.651	3.38 (0.11)	3.17	3.609
[mm]	Test	6.42 (0.44)	5.765	7.1	4.2 (0.62)	3.316	5.454
RaLaRcLcTd		moderate perturbation			moderate perturbation		
Markers	Train	30 (15)	24.69	74.97	22 (17)	15.52	73.73
[px]	Test	32 (16)	23.62	80.08	23 (17)	16.69	73.93
Distances	Train	8.6 (7.6)	6.122	31.66	7 (12)	3.17	42.49
[mm]	Test	8.7 (7.5)	5.765	31.3	6.3 (6.9)	3.316	27.17
RaLaRcLcTd		intense perturbation			intense perturbation		
Markers	Train	9e+16 (2.7e+17)	495.7	8.89e+17	4e+14 (1.3e+15)	98.2	4.256e+15
[px]	Test	5e+20 (1.3e+21)	580.1	4.457e+21	2.5e+23 (7.8e+23)	101.3	2.601e+24
Distances	Train	820 (810)	18.47	2490	1100 (1100)	15.42	2950
[mm]	Test	840 (800)	13.77	2416	1200 (1100)	25.74	2890

Table A.4: Absolute errors on training and testing data after calibration of perturbated DH parameters using multiple individual chains. We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation of nominal DH parameters (see Table 6.5) are compared. Results are averaged over 10 runs (standard deviation is shown in parenthesis).

Appendix B

Sequential calibration - additional results

Chain abbreviation		Only offsets			all DH		
		mean (std)	min	max	mean (std)	min	max
RaRcLc $\rightarrow$ RaLaRcLc		slight perturbation			slight perturbation		
Markers	Train	25.96 (0.74)	24.9	26.91	16.58 (0.41)	15.81	17.11
[px]	Test	26.7 (1.8)	24.15	29.38	18.3 (1.9)	15.92	22.28
Distances	Train	6.48 (0.18)	6.191	6.696	4.01 (0.25)	3.696	4.327
[mm]	Test	6.43 (0.48)	5.531	6.957	4.6 (1)	3.274	6.905
RaRcLc $\rightarrow$ RaLaRcLc		moderate perturbation			moderate perturbation		
Markers	Train	120 (220)	24.9	728.5	34 (56)	16.12	203.5
[px]	Test	120 (210)	24.15	690.8	36 (60)	15.92	215.6
Distances	Train	140 (340)	6.191	1117	70 (220)	3.696	739.1
[mm]	Test	130 (320)	5.531	1040	70 (230)	3.493	768.3
RaRcLc $\rightarrow$ RaLaRcLc		intense perturbation			intense perturbation		
Markers	Train	1.6e+04 (5e+04)	338.3	1.65e+05	1.9e+13 (5.5e+13)	186.7	1.836e+14
[px]	Test	1.6e+04 (5e+04)	387.9	1.668e+05	5e+21 (1.5e+22)	199.7	5.053e+22
Distances	Train	1130 (630)	424.8	2055	5e+06 (1.7e+07)	671.7	5.636e+07
[mm]	Test	1130 (600)	486.8	2012	5e+06 (1.7e+07)	613.7	5.642e+07
RaRcLc $\rightarrow$ RaLaRcLcTd		slight perturbation			slight perturbation		
Markers	Train	25.92 (0.58)	25.11	26.87	16.23 (0.55)	14.85	16.89
[px]	Test	26.8 (1.5)	24.22	28.96	18.7 (1.5)	16.72	22.55
Distances	Train	6.27 (0.12)	6.04	6.472	3.4 (0.18)	3.157	3.716
[mm]	Test	6.4 (0.44)	5.592	7.037	3.74 (0.61)	2.832	4.871
RaRcLc $\rightarrow$ RaLaRcLcTd		moderate perturbation			moderate perturbation		
Markers	Train	320 (520)	25.11	1450	41 (83)	15.87	291.9
[px]	Test	330 (560)	24.22	1602	50 (110)	16.72	376.2
Distances	Train	80 (140)	6.04	389.3	20 (55)	3.222	185.6
[mm]	Test	90 (150)	5.592	448.7	22 (61)	2.832	207.1
RaRcLc $\rightarrow$ RaLaRcLcTd		intense perturbation			intense perturbation		
Markers	Train	4e+12 (1e+13)	1032	3.437e+13	2.8e+14 (6.6e+14)	176.5	2.025e+15
[px]	Test	7e+20 (2.2e+21)	974.8	7.266e+21	2e+21 (6.6e+21)	797.4	2.203e+22
Distances	Train	650 (690)	99.56	2116	4e+05 (1.2e+06)	0.1956	4.096e+06
[mm]	Test	680 (690)	116.1	2152	4e+05 (1.2e+06)	0.5104	3.898e+06

Table B.1: Errors after sequential calibration where first right hand is calibrated using both eyes (RARcLc) and then this calibration is used to calibrate the second hand and end-effector parameters using multichain calibration (RaLaRcLcTd). We show results for a case when only offsets are calibrated (left) and when all DH parameters are calibrated (right). Results for 3 different values of initial perturbation (see Table 6.5) are compared. Results are averaged over 10 runs (standard deviation is shown in parenthesis).

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Puciow František**	Personal ID number:	**420055**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Automatic self-calibration from self-observation and self-touch on a dual arm industrial manipulator

Master’s thesis title in Czech:

Automatická kalibrace pomocí 'sebedotyku' a sebepozorování u dvojrukého průmyslového manipulátoru

Guidelines:

1. Implement safe self-touch behavior in the dual arm Yaskawa Motoman setup (inverse kinematics plus force feedback from six axes F/T sensors)
2. Installation of two external cameras forming a stereo rig, with viewpoint covering the robot workspace.
3. Data collection in self-touch configurations (joint angles, camera images, contact point detection).
4. Study state-of-the-art in kinematic calibration, focusing specifically on multiple kinematic chain problems.
5. Formalization of the calibration problem for different combinations of intersecting kinematic chains, its solving using non-linear least squares methods, and assessing the benefits of individual problem formulations.

Bibliography / sources:

[1] Birbach, O.; Frese, U. & Bauml, B. (2015), 'Rapid calibration of a multi-sensorial humanoid´s upper body: An automatic and self-contained approach', The International Journal of Robotics Research 34(4-5), 420--436.
[2] Hollerbach, J.; Khalil, W. & Gautier, M. (2016), Model identification, in B. Siciliano & O. Khatib, ed., 'Springer Handbook of Robotics', Springer, , pp. 113--138.
[3] Joubair, A., & Bonev, I. A. (2015). Kinematic calibration of a six-axis serial robot using distance and sphere constraints. The International Journal of Advanced Manufacturing Technology, 77(1-4), 515-523.
[4] Roncone, A.; Hoffmann, M.; Pattacini, U. & Metta, G. (2014), Automatic kinematic chain calibration using artificial skin: self-touch in the iCub humanoid robot, in 'Robotics and Automation (ICRA), 2014 IEEE International Conference on', pp. 2305-2312.

Name and workplace of master’s thesis supervisor:

Mgr. Matěj Hoffmann, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Mgr. Karla Štěpánová, Ph.D., Robotic Perception, CIIRC

Date of master’s thesis assignment: **16.01.2018** Deadline for master's thesis submission: **08.06.2018**

Assignment valid until: **30.09.2019**

Mgr. Matěj Hoffmann, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department's signature	Dean’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC