Topias Liikanen

Automatic Calibration Procedure for a Service Robot

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Automation Engineering

Thesis

11 May 2017

Metropolia

Author	Topias Liikanen
Title	Automatic Calibration Procedure for a Service Robot
Number of Pages	23 pages + 3 appendices
Date	11 May 2017
Degree	Bachelor of Engineering
Degree Programme	Automation Engineering
Instructors	Nadia Hammoudeh García, Researcher, Fraunhofer IPA Jukka-Pekka Pirinen, Senior Lecturer, Metropolia UAS

Modern service robots operate in constantly changing environments. For a robot to be able to perform vision-based manipulation tasks in these environments, the robot's vision system and manipulators must be calibrated.

The aim of this thesis project was to develop an automatic calibration procedure for the Care-O-bot 4, a service robot developed by the Fraunhofer Institute for Manufacturing Engineering and Automation (IPA). Different calibration methods were researched and the best suiting methods were selected for implementation.

As a result, an automatic calibration procedure was specified and a software package for performing the procedure was developed. The software package was tested to provide most of the automatic calibration procedure's expected functionalities: *automatic calculation of calibration poses*, *calibration data acquisition*, and *camera calibration*. One important feature for the calibration procedure to be ready for use, *calculation of kinematic calibration parameters*, was not implemented due to the limited time spent on the project.

When finished, the calibration procedure can be utilized to calibrate all Care-O-bot 4 service robots produced by Fraunhofer IPA.

Keywords	ROS, Robot, Calibration, Machine vision, Kinematics

Tekijä Otsikko Sivumäärä Päivämäärä	Topias Liikanen Automaattinen kalibrointiproseduuri palvelurobotille 23 sivua + 3 liitettä 11. toukokuuta 2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Automaatiotekniikka
Ohjaajat	Nadia Hammoudeh García, Tutkija, Fraunhofer IPA Jukka-Pekka Pirinen, Lehtori, Metropolia AMK

Modernit palvelurobotit operoivat jatkuvasti muuttuvissa ja monimutkaisissa ympäristöissä. Jotta robotin on mahdollista suorittaa konenäkö-perusteista manipuolointia näissä olosuhteissa, robotin konenäköjärjestelmä ja manipulaattorit on kalibroitava.

Tämän lopputyöprojektin tavoitteena oli kehittää automaattinen kalibrointiproseduuri Care-O-bot 4 -robotille, joka on Fraunhofer-Instituutissa kehitetty palvelurobotti. Työssä valittiin tämän projektin kannalta parhaiten soveltuvat kalibrointialgoritmit ja niitä sovellettiin automaattisen kalibroinnin kehittämiseksi.

Tuloksena määriteltiin automaattinen kalibrointiproseduuri ja kehitettiin ohjelmistopaketti proseduurin suorittamiseksi. Ohjelmistopaketti tarjoaa suurimman osan automaatiselle kalibroinnille asetetuista vaatimuksista: *kalibrointisijaintien laskenta*, *kalibrointidatan keräys* ja *kameran kalibrointi*. Yhtä tärkeää toiminnallisuutta, *kinemaattisten kalibrointiparametrien laskentaa*, ei ehditty kehittää insinöörityön aikana johtuen aikataulurajoitteista.

Kun automaattinen kalibrointiproseduuri saadaan valmiiksi, sitä voidaan käyttää kaikkien Fraunhofer-Instituutissa valmistettujen Care-O-bot 4 -palvelurobottien kalibrointiin.

Avainsanat	ROS, Robotiikka, Kalibrointi, Konenäkö, Kinematiikka

Contents

Appendices

Appendix 1. Care-O-bot 4 performing calibration in a simulation environment

Appendix 2. Simulated output from the Care-O-bot 4's left camera while performing calibration in a simulation environment

Appendix 3. Automatically generated calibration poses visualized in a simulation

List of Abbreviations

API	Application programming interface
DOF	Degrees of freedom
Frame	Fixed coordinate system
Joint	In this thesis joint refers exclusively to a revolute joint
Link	Rigid body connecting joints
Pose	Position and orientation
ROS	Robot Operating System, a meta-operating system for robot development
ROS stack	Collection of ROS packages that collectively provide functionality
ROS package	Set of ROS software libraries
tf	ROS package that lets the user keep track of multiple coordinate frames over time
YAML	Human-readable data serialization language. Commonly used for configuration files.
XML	Extensible Markup Language. Used to describe data.

1 Introduction

Modern service robots consist of multiple sensors and manipulators, typically including machine vision systems and one or two robotic arms.

For the robot to be able to perform vision-based manipulation tasks, such as object grasping in unstructured environment, the robot has to use its sensors and actuators, and the precise and safe use of these components in constantly changing environments requires their inter-relationships and the sensors' so called intrinsic parameters to be well known. To obtain this information the robot must be calibrated. That is, the sensors' and the actuators' relative locations to each other must be known and possible optical errors in the camera must be corrected. The former is called extrinsic calibration and the latter intrinsic camera calibration. [9]

The aim of this study was to develop an automatic calibration procedure for a robot, namely, the Care-O-bot 4 service robot developed by Fraunhofer IPA. The project was carried out between March 2015 and July 2015 at Fraunhofer IPA in Stuttgart, an institute for manufacturing engineering and automation.

One of the main requirements for the task is that the whole calibration procedure should be highly automated, requiring as little effort and expertise by the end user as possible. As the COB 4 is highly modular robot, the calibration should also be flexible; i.e. the calibration procedure shall be easily configurable for the different setups of COB 4.

In this study, the most sufficient extrinsic and intrinsic calibration methods were selected and implemented. A software package for automatic calibration procedure was developed as a result.

2 Care-O-bot Project

Care-O-bot is the product vision of a mobile robot assistant to actively support humans in their daily life developed by the Fraunhofer IPA institute [3]. The first Care-O-bot prototype was built in 1998, and it was already able to navigate safely and reliably by itself. The fourth and the latest generation, Care-O-bot 4, was completed in January 2015. Care-O-bot 3 was being used by numerous research institution and universities around the world and the Care-O-bot 4 is expected to follow its success [11].

2.1 Care-O-bot 4

Care-O-bot 4 is a modular robot research platform, which potential applications include providing household assistance in daily life, providing room service in hotels or shelf picking in warehouses [3].

Figure 1. Front side of the Care-O-bot 4. [2]

The robot's main parts, seen in figure 1, are: head, torso, base, and arms. Each of these are modularly connected to each other.

The Care-O-bot 4 setup used in this project comprise two 7-DOF arms, a 3-DOF head, a 3-DOF torso and an omni-directional base. It includes the following sensors and actuators:

Actuators:

- Arms
  - The two robotic arms of Care-O-bot 4 are specifically designed for Care-O-bot 4 by Schunk company. Each arm consists of seven Schunk Powerball joints, connected to each other by lightweight carbon fiber links.
- Neck
  - The spherical joint in the neck allows the Care-O-bot 4 to turn its head in three dimensions.
- Hip
  - The hip joint between the base and the torso is also spherical, extending the working space and allowing the Care-O-bot 4 to reach the floor level with its arms.
- Base
  - The mobile base platform comes with two possible configurations: an omnidirectional drive system allowing movements in all directions or a car-like differential drive system.
- Grippers
  - The grippers are custom design and have two joints each allowing the manipulation of most everyday objects.

Sensors:

- Laser scanners
  - Laser scanners are used for simultaneous localization and mapping (SLAM). The three laser scanners are located on each three sides of the base, leaving no blind spots.

- Cameras and depth sensors
  - Several Asus depth sensors, each containing a point cloud sensor and an RGB camera. These sensors are used to obtain 3D information of the environment and for object detection in manipulation tasks. Three sensors are located at the torso, one at the head, and one at each gripper.

3 Motive for Calibration

The need for calibrating a robot derives from the fact that the physical properties of a robot are not known in desirable precision, which can be result of, for example, inaccuracies in assembling the robot, deformations caused by transporting the robot or switching the robot's sensors. Although these inaccuracies are not likely to be more than few millimeters or degrees, a small mounting angle error between the elbow and the torso can cause a significant misplacement at the end of the arm making it impossible to manipulate small objects without calibration.

If the robot is to be perform vision-based manipulation tasks, the mounting position of the camera or cameras must be calibrated as well.

To perform vision based manipulation tasks, the object to be manipulated must be observed by the sensors and be localized in the space with respect to the robot. If the cameras are attached to unknown places, no purposeful action for the robot can be calculated from the obtained data.

3.1 Grasping Process

A detected object will be localized with respect to the camera coordinate system. This position must then be converted to the robot coordinate system, namely, torso link as both the cameras and the arms are attached to the torso of the Care-O-bot 4. Part of this transformation is the unknown mounting position of the camera to the torso. In order to finally grab the object, the object's position must be known relative to the first link of the arm, which requires the mounting position of the arm also to be known. The rest can be calculated by utilizing forward kinematics. Thereby, small errors can add up to large inaccuracies in the outcome of a grasping process.

4 ROS (Robot Operating System)

ROS is a collection of open-source software frameworks for robot software development. It is a meta-operating system in a sense that it provides standard operating system services such as hardware abstraction, low-lever device control, implementation of commonly used functionalities, message passing between processes, and package management. [8,13]

O'Kane [8] lists several specific issues in robot development that ROS can help to overcome:

- Distributed computation
  - Many modern robot systems rely heavily on distributed processing where the robot's software is divided into small stand-alone parts and in some cases, such as COB4, runs across several different computers. Inherently there has to be a way for the processes to communicate between each other, which ROS does provide.

- Software reuse
  - The progression in the robotics research continues to produce increasing amount of generally applicable algorithms, such as motion planning, mapping, navigation and many more. ROS's standard packages provide stable implementations of many important robotics algorithms without the need to reimplement each algorithm for new systems.

- Rapid testing
  - Developing software for a physical machine usually requires time-consuming testing. Physical robots may not always be available and when they are, the process may be slow and error-prone. ROS provides capabilities to overcome these challenges in forms of simulation tools, abilities to separate low-level control of the hardware and high-level decision making, and abilities to record and play back sensor data.

4.1 Basic ROS Concepts

4.1.1 Nodes

A node is a process that performs computation; i.e. a running instance of a ROS program. A robot control system will usually comprise many nodes that communicate with each other using topics, services and the parameter server. [14]

The use of nodes provides several benefits in comparison to monolithic systems, such as additional fault tolerance and reduced code complexity [14].

4.1.2 Messages

A message is a strictly typed data structure that is used in communication between nodes. Standard primitive types, such as integer, floating point and boolean, are supported, as are arrays of primitive types. Messages also support compositions of other messages and arrays of other messages. [10]

4.1.3 Topics

Topics are buses over which nodes can exchange messages in a streaming manner. The concept is that a node willing to share data will publish messages to the relevant topic or topics and a node that is interested in a certain kind of data will subscribe to the appropriate topic or topics. As such, nodes do not have to be aware whom they are communicating with, but only the name of the relevant topic must be known. Multiple nodes can publish and subscribe to the same topic. Topics are intended for streaming communication such as sensor or control data. Each topic is strongly typed by the ROS message type, which is described in a *.msg* file. The description is composed of two parts: a type and a name, separated by a space. [8,18] See listing 1 as an example.

```
Header header
string chain_id
float32[] translation
float32[] rotation
```

Listing 1. Message description.

4.1.4 Services

While the publish/subscribe model is a flexible communication paradigm, its one-way transport is not ideal for request/reply type, or synchronous, transactions. This type of transaction in ROS are called a service, which is defined by a pair of strictly typed messages: one for the request and one for the response. See listing 2 as an example. The idea is that a providing node offers a service, and a client node sends a request to server node and waits for a reply. This is analogous to web services, but for the programmer it usually appears as if it were a remote procedure call. In contrast with topics, services are bi-directional and communication is of one-to-one type. Service descriptions are stored in *.srv* files. [8,10,17]

```
#request fields
---
#response fields
bool master
bool every
bool[] visibleImages
string[] cameraTopicID
```

Listing 2. Service message description.

4.1.5 Parameter Server

In addition to topics and services, ROS provides another method of data transfer across nodes called the parameter server. The centralized parameter server keeps track of a collection of values identified by a short string name, and is ideal for storing configuration parameters that will be globally viewable to and modifiable by nodes. [8,16] Parameters can be defined in a YAML file and be called at the program start-up, uploading the parameters to the parameter server. See listing 3 as an example of parameter definition in YAML file.

```
checkerboards:
  cb_9x6:
    corners_x: 9
    corners_y: 6
    spacing_x: 0.03
    spacing_y: 0.03
```

Listing 3. Parameter definition in YAML file.

4.1.6 Bags

A bag is a file format in ROS for storing message data published on one or more topics. Bags are created by a tool called rosbag, which subscribe to a topic or topics, and records the serialized message data in a file. The bag files can then be replayed on those same topics or be remapped to replay on new topics. [8,12] In this project bags are used to store joint position data and image data for later processing by the calibration algorithm.

4.2 Coordinate Frames and Transformations in ROS

In ROS, every component (e.g. joint, link, sensor) that is involved in the operation of the robot has its own coordinate frame, usually forming a tree structure, with the base link as the root frame.

The most popular way to handle transformations in ROS is by a package called tf. tf is a package that lets the user keep track of multiple coordinate frames over time and transform data within an entire system without requiring knowledge of all the coordinate frames in the system. [5]

5 Basic Concepts

After the motive for the robot calibration have been explained, the fundamental concepts and theory behind the calibration algorithms will be explained in this section.

First, coordinate transformations needed in kinematic calculations will be explained, followed by the concept of camera calibration. Finally, the robot operating system ROS, with which Care-O-bot 4 is operated, will be presented. The automatic calibration will be implemented as a component for ROS.

5.1 Coordinate Transformations

An important concept in robotics is the transformation from one coordinate system to another. It is important especially in autonomous robotics. If a robot wants to localize an object, observed by a sensor, with respect to the robot coordinate system, the relationship between the coordinate frame of the sensor and the coordinate frame of the robot (base link) must be known. The same applies in manipulation tasks; if a sensor determines a position of an object relative to the sensor frame, the robot should be able to compute the transformation all the way from the sensor to the robotic arm's gripper in order to accurately grasp the object. In a simple system this could require the following transformations:

camera -> base link -> arm's mounting frame -> arm link 1 -> … -> arm link n -> gripper

This results in a so-called *kinematic chain* between the camera, the object, and the end-effector. By forming a kinematic chain the robot can compare the gripper's position with respect to the object in question and move the gripper accordingly to grasp the object.

Robot tasks are often defined in Cartesian workspace using Cartesian coordinates. [19] Coordinate transformations allow for moving between different coordinate frames and it can be decomposed into a rotation and a translation [7]. There are many ways to represent rotation, but the most used one in robotics is homogeneous transformation [7].

Homogeneous transformation combines rotation and displacement into a single transformation matrix. Homogeneous transformation expressing the orientation and position of frame $o_j x_j y_j z_j$ with respect to $o_i x_i y_i z_i$ can be denoted as

$$H_j^i = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n & s & a & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

where $n = (n_x, n_y, n_z)^T$, $s = (s_x, s_y, s_z)^T$ and $a = (a_x, a_y, a_z)^T$ are vectors representing the directions of x_j, y_j and z_j, respectively, in the frame $o_i x_i y_i z_i$.

Each link in a robotic arm is assigned with its own coordinate frame ($o_0 x_0 y_0 z_0$ … $o_n x_n y_n z_n$) with link *0* being attached to the base link and link n being the last link, thus forming a kinematic chain. See figure 2 as an example.

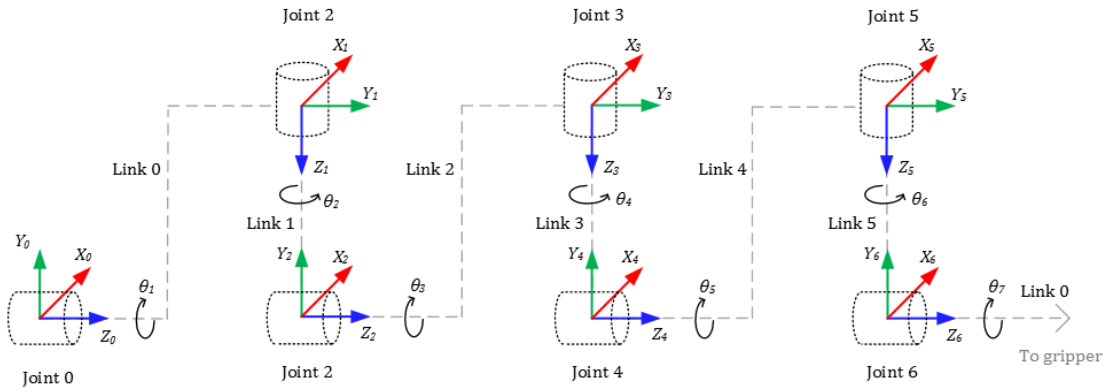


Figure 2. Coordinate frames of the Care-O-bot 4's left arm.

Now utilizing homogeneous transformation, it is convenient to carry out forward and inverse kinematic analysis. Forward kinematics refers to finding the pose of the end effector with respect to the base frame, provided the joint variables are known. Inverse kinematics is the opposite: finding the joint parameters that provide a desired pose of the end effector. [19]

5.2 Camera Calibration

One essential step in the robot calibration is camera calibration, or camera resectioning. All cameras come with some level of distortion in the output image, which will eventually affect negatively on the extrinsic robot calibration's accuracy as the cameras will be utilized during the extrinsic calibration. Therefore, a successful extrinsic calibration requires that the cameras' intrinsic parameters are well known, which means that the cameras' must be calibrated before performing extrinsic robot calibration.

A camera is usually modeled by the pinhole camera model (see figure 3): the relationship between a 3D point M and its image projection m is given by

$$s\widetilde{m} = A[R \quad t]\widetilde{M} \tag{2}$$

or

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

where

- (X, Y, Z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- A is a camera matrix, or a matrix of intrinsic parameters
- (u_0, v_0) is a principal point that is usually at the image center
- α, β are the scale factors in image u and v axes. [20]

The transformation includes the rotation-translation matrix $[R \quad t]$, which transforms coordinates of a point (X, Y, Z) to the coordinate system of the camera [20]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{4}$$

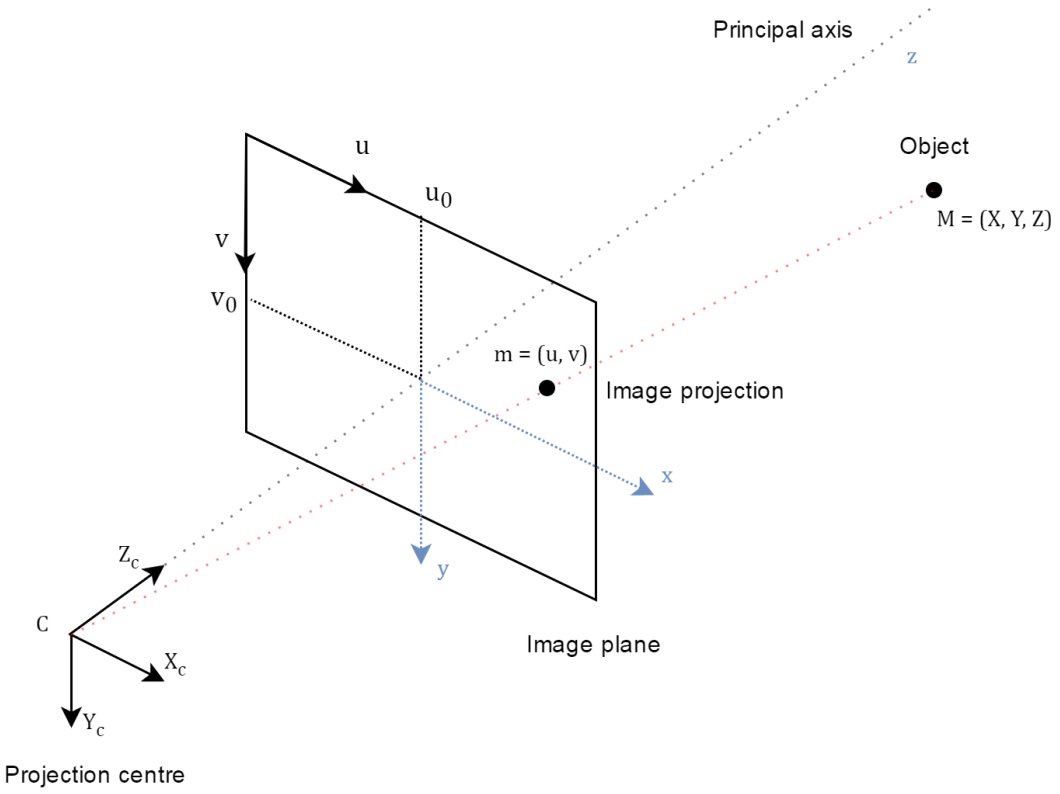


Figure 3. Pinhole camera model.

The task of camera calibration is to determine the parameters of the transformation between an object in 3D space and the 2D image observed by the camera. The model described above is used by the camera calibration algorithm used in this project. The algorithm is implemented in ROS as a package called camera_calibration [15] and it is based on the work of Zhang and Bouguet. [20]

6 Choosing the Calibration Algorithms

It is by no means sensible to develop robot calibration algorithms from the ground up for several reasons, but most importantly, robot calibration is a well-researched field and multiple solutions already exist in ROS. Therefore,

6.1.1 Intrinsic Camera Calibration Algorithm

Camera calibration is a well-researched topic in machine vision among other fields, and therefore many calibration methods exist for different purposes. In case the requirements were: already implemented in ROS, and automatable requiring not much effort by the end-user.

The selected method is based on the work of Zhang and Bouguet and is implemented in ROS as a popular ROS package called camera_calibration [15]. The method requires a planar calibration object to be observed by the camera at a few different poses. The procedure is easily automatable: the calibration pattern can be attached to the robot hand and moved automatically to each calibration position. The method does not require effort by the end-user other than providing a calibration pattern and attaching it to the arm.

6.1.2 Kinematic Calibration Algorithm

A method developed by Pradeep, Konolige and Berger [9] was chosen as the kinematic calibration algorithm for the listed reasons:

- It has already been successfully implemented for the Care-O-bot 3 [1,6], and some parts of the code can potentially be reused. The Care-O-bot 3 implementation exists as a ROS package
- It allows for simultaneous calibration of both actors and sensors, i.e. it calculates the extrinsic calibration results for all the involved cameras and arms in a single run.

It is a general-purpose framework for extrinsically calibrating all the sensors and actuators of a robot. [9]

7 Implementing Automatic Calibration for Care-O-bot 4

In this section, the development of the calibration procedure for the Care-O-bot 4 along with its functionality will be briefly explained.

As stated in the previous chapter, an automatic calibration procedure for the Care-O-bot 3 exists as a ROS stack called cob_calibration. The Care-O-bot 4's calibration procedure was decided to be built on top of the cob_calibration stack, leveraging the existing code and likely reducing the amount of new code.

There are however many differences between the Care-O-bot 4 and the Care-O-bot 3, which must be taken into consideration when re-implementing the existing code: Care-O-bot 4's kinematics are different to the ones of the Care-O-bot 3; for instance, Care-O-bot 4 can have two arms compared to the one arm of the Care-O-bot 3. Care-O-bot 4 can be equipped with several point cloud cameras whereas Care-O-bot 3 had a stereo-pair camera as its sensor for manipulation tasks. The Care-O-bot 3's calibration procedure was programmed only to work with the non-modular construct of the robot. The programming interfaces of the robot's basic functionalities, such as actuator controls, are also different in the new model.

After gathering all the base information and achieving a general understanding main concepts, the programming part was done. Some parts of the Care-O-bot 3's calibration code could be reused, but most of it had to be rewritten to account for the requirements of this project and the vastly different robot. The source code was programmed in Python programming language. Some configuration files were written in YAML and XML languages.

The calibration procedure can be divided into the five major steps listed below:

1. The user sets necessary parameters and attaches the checker boards to the end effectors
2. Calculate calibration poses. That is, the robot will compute all the calibration poses for each arm in advance before moving the checkerboard.
3. Move the arm to every calibration pose and by stopping the checkerboard at each pose and capturing the joint values and an image of the checkerboard at given pose. Repeat for each arm. Save the output into a bag file.

4. Calculate intrinsic camera calibration parameters from the data gathered in step 3. Upload the parameters to the camera driver.
5. Calculate kinematic calibration parameters will be calculated using bundle adjustment method. The bag file generated in the step 3 will be fed in as an input. The kinematic parameters will be written to the robot's configuration description file and the camera correction parameters will be uploaded in to the cameras' firmware.

The parameters mentioned in the step 1 will be addressed in the following subsection. A more detailed explanation of the whole procedure will be given in the subsection 7.2.

7.1 Parameter Configuration

The variety of camera models must be considered when designing the automatic calibration algorithm: the algorithm must be easy to use with as little configuration and effort by the user as possible. A configuration file was defined where the user fills necessary parameters for each camera. Configuration of the left torso camera is shown in listing 4.

```
name: torso_cam3d_left_link #identification of the camera
topic: /torso_cam3d_left/rgb/image_raw #rostopic where the raw camera
data is published
cam_info_topic: /torso_cam3d_left/set_camera_info #rostopic that pub-
lishes the projection matrix to the camera driver
file_prefix: left_kinect #calibration file pre-fix
frame_id: /torso_cam3d_left_link #cameras tf frame id
```

Listing 4. Configuration of a camera.

Similiarly to the cameras, the arm configuration must be also configurable to account for the different configurations. Care-O-bot 4 uses a software called MoveIt! for manipulation tasks. MoveIt! has a ROS API, through which all the necessary information considering the robot's arms can be retrieved. By utilizing this fact, the configuration of the arms can be made simple for the end-user: the user only has to give a name, called planning group, that a given arm is configured as in MoveIt!.

In addition to an arm's name, more configuration parameters will be needed for the calibration. Namely, coordinates of an area inside which the calibration poses will be generated, discretization of the area, end-effector's orientation at calibration poses, and a camera's frame link that the arm is to be calibrated against. A sample configuration is shown in listing 5.

```
arms:
  arm_left:
    name: arm_left  #Name of the planning group
    limits_corner_1: [0.0, 0.30, 0.2] #x1, y1, z1 These are the limits for the end effector link
    limits_corner_2: [-0.50, -0.30, 0.8] #x2, y2, z2
    discritization: [3,3,3] #Discretization of x, y, z
    hand_orientation: [0.5, 0.5, 0.5, 0.5] #Quaternion values. Use converter to convert from Euler
    arm_planning_reference_frame:  torso_cam3d_left_link     #Relative to which frame the limits and the orientation is defined

  arm_right:
    name: arm_right
    limits_corner_1: [0.0, 0.30, 0.2]
    limits_corner_2: [-0.50, -0.30, 0.8]
    discritization: [3,3,3]
    hand_orientation: [0.5, 0.5, 0.5, 0.5]
    arm_planning_reference_frame: torso_cam3d_right_link
```

Listing 5. Configuration of the left and the right arms.

7.2 Functionality

The description of the procedure presented at the beginning of this section was to offer a general view of the calibration procedure. Now the steps will be fractioned and explained to give a more in-depth look at the procedure and the program's functionality:

1. Let the user prepare the robot by attaching checkerboards to each arm and set the following parameters:

    - Defining which arms will be involved in the calibration by telling their planning group name

    - Limits for the area in Cartesian coordinates inside which the end-effector position generator will try to generate the calibration poses

    - Discretization of X, Y, Z axes, which defines into how many segments the position generator will divide each axis. The larger the discretization, the more poses will be tried to generate among the axis.

    - Desired hand orientation in quaternion values with respect to the camera during the calibration. This will define the orientation of the checkerboard at the calibration poses. This parameter should optimally be defined so that the checkerboard orientation will be normal to the camera's optical axis. The position generator will add some randomness to the orientation during the calibration.

    - Reference frame of the limits_corner and hand_orientation parameters. The reference frame should be one of the cameras' frame so that the checkerboard will be moved in front of the camera during the calibration

2. Calculate calibration poses for every arm using inverse kinematics. This will be done by the generate_positions script.

    - The script will calculate the poses using inverse kinematics and applying collision avoidance algorithms to prevent generating any poses that will result in a collision

3. Move arm (with a checkerboard attached to it) to every calibration pose generated in the step 1. At every pose, capture an image of the checkerboard with every camera that the checkerboard is fully visible for. If the checkerboard can be seen by at least one camera, save the current joint values and the image data (in a raw format) into a bag file at the given pose by calling services capture_image and capture_kinematics.

    - Repeat this step for each arm

4. Calibrate cameras' optics by calculating intrinsic calibration parameters:

    - The script will iterate through the images captured into the bag file

        - If image contains enough new and relevant information for intrinsic calibration, it will be added to a sample database

    - If the sample database is sufficient enough for each camera (enough images and enough variation among the images), the script will calculate the calibration parameters. New calibration parameters will be uploaded to the camera driver.

    - If there is not enough image data for intrinsic calibration of every camera, the user will be given a decision to select between two options on how to proceed:

        a) Terminate the calibration procedure and let the user set parameters that will result in more calibration poses and thus more image data. This means that the user must increase the limits that restrict the area of the automatically generated poses and/or increase the discretization parameter that tells how many poses the generate_positions script will try to generate among the axes X, Y, Z.
        b) Proceed to extrinsic calibration with potentially insufficiently calibrated cameras.

The full source code for the project can be found on GitHub [4]. As stated before, the functional description above describes the calibration procedure's current state. The last necessary step for a successful calibration, which will be described in the next subchapter, was not implemented due to the limited time spent on the thesis project. The procedure's functionality until the step 4 was fully tested in Gazebo and RViz simulation softwares and was observed to perform as expected. Screenshots from testing the robot in simulation environment are attached as appendices 1, 2 and 3. The program was not tested on a real Care-O-bot 4 as no robot was available for testing at that time.

7.3 Future Development

The following step was not, but shall be implemented in the future to finalize the calibration procedure:

5. Calculate the kinematic calibration parameters by utilizing the calibration algorithm developed by Pradeep, Konolige and Berger [9]. The algorithm will take the bag file described in the step 3 as an input. The resulting output will be the extrinsic calibration parameters for both the cameras and the arms. The parameters will be written to the robot's URDF description file which, among other things, describes the robot's kinematic properties.

As the development could not be finished on time, no measurable test results regarding the calibration performance could be produced. The implementation of the last final step of the robot calibration will be continued by another developer at Fraunhofer IPA.

8 Summary

The goal of this thesis project was to develop an automatic calibration procedure for the Care-O-bot 4 robot. As a result, a ROS package called cob_calibration was developed comprising the functionalities described in the chapter 7.2.

The task turned out to be bigger than what was expected, and thus the project could not be finished entirely during the limited time that I worked on it. However, most of the calibration procedure's desired functionalities were developed and verified to perform correctly, forming a strong basis for the finalization of the project. The automatic calibration procedure, when finished, can be used to calibrate all Care-O-bot 4 robots produced by Fraunhofer IPA in the future, despite the differing configurations it may come in.

The future work on the project will consist of implementing the kinematic calibration parameter calculation algorithm, which will calculate the extrinsic calibration parameters.

References

1 Abbenseth JS. (2013). Automatische Kalibrierung eines mobilen Serviceroboters. Hochschule Furtwangen University, Furtwangen, Germany.

2 Bez R. (2015). Care-O-bot 4 press photos. [Internet]. Available from: http://www.care-o-bot.de/en/care-o-bot-4/download/images.html [Accessed 11th May 2017].

3 Care-O-bot 4. [Internet]. Available from: http://www.care-o-bot-4.de/ [Accessed 27th April 2017].

4 cob_calibration source code on GitHub. [Internet]. Available from: https://github.com/ipa-nhg-tl/cob_calibration [Accessed 27th April 2017].

5 Foote T. (2013). tf: The Transform Library. In: IEEE Conference on Technologies for Practical Robot Applications (TePRA).

6 Haug SA. (2012). Automatische Kalibrierung eines komplexen Serviceroboters. University of Stuttgart, Stuttgart, Germany.

7 Kucuk S, Bingul Z. (2006). Robot Kinematics: Forward and Inverse Kinematics, Industrial Robotics: Theory, Modelling and Control, Sam Cubero (Ed.). InTech. [Internet]. Available from: http://www.intechopen.com/books/industrial_robotics_theory_modelling_and_control/robot_kinematics__forward_and_inverse_kinematics [Accessed 27th April 2017].

8 O'Kane JM. (2013). A Gentle Introduction to ROS. [Internet]. Available from: https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf [Accessed 27th April 2017].

9 Pradeep V, Konolige K, Berger E. (2010). Calibrating a multi-arm multi-sensor robot: A Bundle Adjustment Approach. In: International Symposium on Experimental Robotics (ISER), New Delhi, India, 2010.

10 Quigley M, Gerkey B, Conley K, et al. (2009). ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software.

11 Robots as multifunctional gentlemen. (2015). [Internet]. Available from: http://www.ipa.fraunhofer.de/en/care-o-bot4.html [Accessed 27th April 2017].

12 ROS Bags. [Internet]. Available from: http://wiki.ros.org/Bags [Accessed 27th April 2017].

13 ROS Introduction. [Internet]. Available from: http://wiki.ros.org/ROS/Introduction [Accessed 27th April 2017].

14 ROS Nodes. [Internet]. Available from: http://wiki.ros.org/Nodes [Accessed 27th April 2017].

15 ROS package camera_calibration. [Internet]. Available from: http://wiki.ros.org/camera_calibration [Accessed 27th April 2017].

16 ROS Parameter Server. [Internet]. Available from: http://wiki.ros.org/Parameter%20Server [Accessed 27th April 2017].

17 ROS Services. [Internet]. Available from: http://wiki.ros.org/Services [Accessed 27th April 2017].

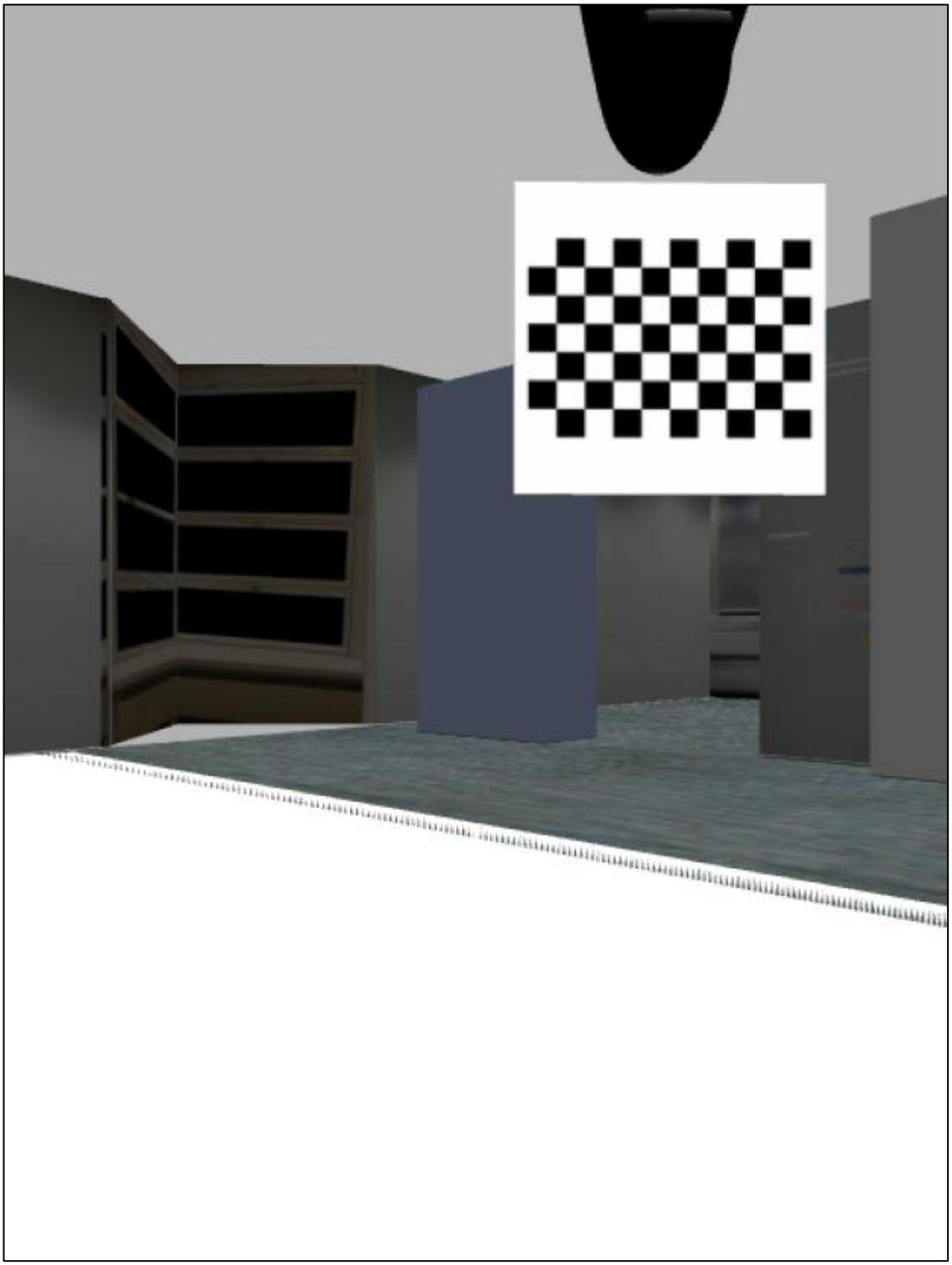
18 ROS Topics. [Internet]. Available from: http://wiki.ros.org/Topics [Accessed 27th April 2017].

19 Spong MW, Hutchinson S, Vidyasagar M. (2004). Robot Dynamics and Control (2nd ed.). [Internet]. Available from: http://smpp.northwestern.edu/savedLiterature/Spong_Textbook.pdf [Accessed 27th April 2017].

20 The OpenCV Reference Manual. (2017). Release 2.4.13.2. [Internet]. Available from: http://docs.opencv.org/2.4.13.2/opencv2refman.pdf [Accessed 27th April 2017]

Care-O-bot 4 performing calibration in a simulation environment.

Simulated output from the Care-O-bot 4's left camera while performing calibration in a simulation environment.



Automatically generated calibration poses visualized in a simulation.