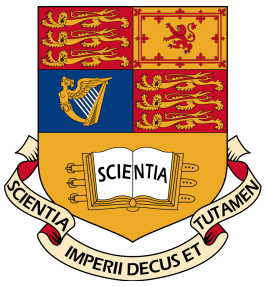


Polytechnic University of Turin
Department of Control and Computer Engineering

Imperial College London
Mechatronics in Medicine

Kinematic Calibration of a Seven Revolute Joints Serial Manipulator

Author:
Jennifer Chimento

Supervisor:
Prof. Marcello Chiaberge
Co-Supervisor:
Dr. Riccardo Secoli

A Thesis submitted for the Master Degree in

Mechatronic Engineering

October 2019

Abstract

The kinematic calibration is a process that computes a realistic mathematical representation of a robot by calculating the appropriate estimates of the model parameters. This method aims to increase the accuracy of the robot manipulator. Therefore, the calibration process allows the robot to perform more accurate positioning tasks without physically modifying the manipulator. In this thesis, a method is developed to compensate inaccuracies due to imprecise manufacturing of robot's parts, by using the tracking system Atracsys fusionTrack 500. In particular, the robot to be calibrated is the KUKA iiwa LBR 7R 800, a serial-link arm manipulator with seven revolute joints, mostly used in both manufacturing and biomedical sectors. This thesis was developed under the supervision of both Polytechnic University of Turin and Imperial College London. Moreover, all the experiments were carried out in the Mechatronics in Medicine Laboratory at Imperial College London.

La calibrazione cinematica è un processo che permette di ottenere un adeguato modello matematico di un robot, attraverso il calcolo dei suoi parametri. Il suo scopo è quello di migliorare l'accuratezza del robot. Quindi, permette di aumentare la precisione di un manipolatore durante l'esecuzione di attivitá di posizionamento, senza modificare strutturalmente e fisicamente il robot. In questa tesi è stato elaborato un metodo per compensare gli errori di fabbricazione del robot utilizzando il sistema di tracking Atracsys fusionTrack 500. In particolare, il robot da calibrare è il KUKA iiwa LBR 7R 800, un manipolatore con sette accoppiamenti rotoidali utilizzato maggiormente nel settore industriale e biomedico. Questa tesi è stata elaborata sotto la supervisione del Politecnico di Torino e dell'Imperial College London. Inoltre, tutti gli esperimenti sono stati condotti all'interno del Laboratorio Mechatronics in Medicine dell'Imperial College London.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

The kinematic calibration of a robot is a method that evaluates appropriate estimates of parameters, which represent the mathematical model of the manipulator, in order to increase the robot's accuracy. Therefore the calibration allows the robot to perform accurate positioning tasks without physically modifying the manipulator.

Accuracy is an important robot's characteristic and it is useful in some robotic applications. For example, a robot could be used to insert precisely in the brain a biopsy needle to pick up some tissue [11]: in this case, accuracy plays a fundamental role since the surgeon dexterity is not enough to complete this type of operation.

When robots were first introduced in the manufacturing field, they were capable to perform pick-and-place operations that did not require accuracy. In particular, this type of action was taught manually by the engineers: an operator would specify the exact motion that the robot would perform and that motion would be performed repeatedly. This was possible because the number of tasks that the robots had to perform was small [5]. As the years went by, manipulators were used for other applications that involved large numbers of taught actions. Thus new electronic components were added to the robots to avoid the time consuming manual teaching phase [5]: the mathematical model of the manipulator, known as kinematic model, was used to insert new positions that a robot had to reach. The engineers immediately noticed that there were substantial differences between where the robot was supposed to go and the desired position: they came to a realization that even small errors in the kinematic model could produce great mistakes in the robot's positioning.

Therefore simple actions taught by manual training can be accomplished without calibration because they rely on repeatability, i.e the capability of the robot to return to a previously achieved pose [5]. When adding new tasks, never manually taught by the user, the robot depends on accuracy, the robot's ability to achieve a position that is not previously reached [5]. When a user moves manually the robot to a specific point, the robot's joint angles are measured by the robot's controller and then stored, hence the manipulator can return to that location. For a new position, the joint angles must be computed and this calculation depends on the kinematic model of the robot, i.e parameters of the robot's geometry. The errors in these values cause inaccuracy when moving to a new location [12]. That is where calibration comes in: it computes the proper values of the kinematic model parameters to compensate the error between the theoretical parameters values, thus the ideal robot geometry, and the actual physical dimension of the manipulator.

The aim of this thesis is to offer a strong calibration method to cope with geometric errors of a serial-link manipulator with seven revolute joints. For this thesis purpose, the KUKA iiwa LBR 7R 800 (KUKA) is employed: it is a robot used for both manufacturing and medical purposes. The tracking system Atracsys fusionTrack 500 (Atracsys) is em-

ployed to record the manipulator positions.
In the following chapters the calibration process is presented and robot modeling and parameter identification are fully described. Later, the used robot is presented and the functionalities of the Atracsys are illustrated in detail. Besides, the main principles of Robotic Operating System (ROS) framework are highlighted to explain the complexity of the system during the recording process needed for the calibration. Then the algorithm used for the calibration procedure is explained step by step and finally, the results are discussed.

This project was developed under the supervision of both Polytechnic University of Turin and Imperial College London. In particular, all the experiments were conducted in the Mechatronics in Medicine Laboratory at the Mechanical Engineering department of Imperial College London.

1.2 Objectives

There are three main objectives in this thesis:

- Development of an algorithm that moves automatically the robot inside a specific workspace and stores the robot's positions recorded by the Atracsys in Excel files.
- Development of an algorithm that minimizes the error between the recorded real poses and the outputs of the theoretical model.
- Comparison between the robot's positions obtained from the new calibrated parameters and the recorded ones.

1.3 Contributions

The following contributions are presented in this thesis are original ideas and presented herein for the first time:

- Design of three platforms needed to position reflective spheres necessary for the measurement phase.
- Contribution of a ROS package development for the Atracsys fusionTrack 500.
- Development of an algorithm that allows the robot to follow dinamically the camera.

1.4 Thesis Overview

- In Chapter 2 the theory behind the robotic manipulator movement and kinematic calibration are discussed. In particular, different methods for performing the calibration are presented.
- In Chapter 3 the system used to perform the calibration process is described. Moreover, an introduction to Robot Operating System (ROS) framework is illustrated.
- In Chapter 4 an open-loop calibration method is presented. In particular, the algorithms developed for this purpose are described step by step.
- In Chapter 5 the experiments performed to verify the validity of the developed algorithms are depicted. A discussion of the results is provided.
- Finally, in Chapter 6 conclusions are presented.

Chapter 2

Theory and Literature Review

This chapter consists in describing the existing theory necessary to understand the kinematic calibration of the KUKA iiwa LBR. Moreover an overview of different methods of calibration is described.

2.1 Kinematic Chains

Kinematics is the science of motion that allows to represent positions, velocities and accelerations of specified points of a multi-body structure, without taking into account the forces or torques that cause the motion. To study the kinematics of a manipulator, the concept of kinematic chain must be investigated.

A kinematic chain consists of a set of ideal rigid links connected by ideal rigid joints: it represents the mechanical or kinematic structure of a robot manipulator. Furthermore, a kinematic chain is considered as a geometric entity, therefore frictions, masses, and elasticities of the kinematic chain are not considered [1]. One end of the kinematic chain is constrained to a rigid surface, called base, while the other one is connected to a specific surface on which a gripper or a tool could be mounted, referred to as end-effector (EF). The tool-center-point (TCP) is a point, generally located in the middle of the end-effector, that a manipulator moves to a specific position or along a specified path: it is the point to refer to when the robot is moving the end-effector, hence it represents the end-effector itself. The representation of the kinematic chain is shown in Figure 2.1.

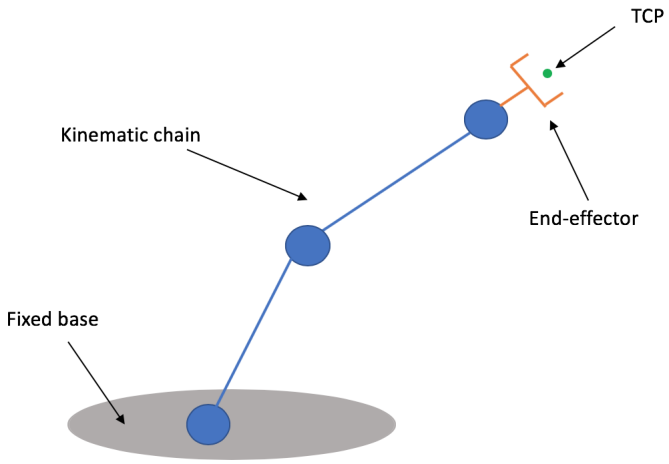


Figure 2.1: The kinematic chain of a manipulator

Since the end-effector is usually a rigid body that moves in three-dimensional space, the TCP has a position as well as an orientation with respect to a fixed reference frame located on the robot base, called the world frame or base frame. The combination of position and orientation of the TCP with respect to the base frame is referred to as a pose. When the end-effector is defined in this manner, the TCP is described in the task space: the task space, or workspace, is the subset of the cartesian space that can be reached by the TCP [1]. Hence, the end-effector pose could be represented in the task space without knowledge of the manipulator geometry.

Joints are components of the kinematic chain that allow the relative motion between two attached links. The two most commons joints are revolute and prismatic joints, each with a single degree of freedom (DOF). The revolute joint allows a rotation about a single axis, while the prismatic joint provides a translation along a specific axis. There are also other types of joints, as shown in Figure 2.2, but they are not discussed in this thesis. The robot used for the kinematic calibration, as said in Subsection 1.1, contains seven revolute joints and so it has seven DOFs. Since the end-effector is positioned and oriented in a three-dimensional space, just six DOFs are required to define its pose (three for positioning and three for orienting the end-effector). If more DOFs than task variables are available, the manipulator is referred to as a redundant robot [2]. Therefore, the KUKA iiwa LBR 7R 800 is a redundant manipulator.

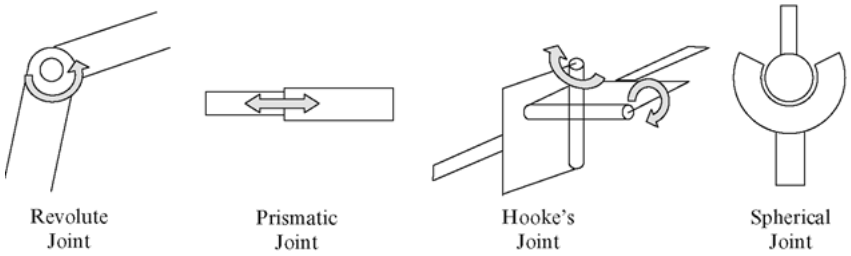


Figure 2.2: Different types of joints used in a robot

Forward kinematics describes the pose of the end-effector with respect to the world frame, depending on the robot's geometry and either the offsets or angles of the robot's joints (offset for prismatic joint and angle for revolute joint). Therefore the end-effector pose can be defined by specifying the manipulator's geometry and joint displacements necessary to achieve the pose [5]. When the pose is defined in this manner, the TCP is described in the joint space: the joint space is a mathematical structure whose elements are joint values, i.e the joint displacements. Hence a joint motion in the joint space produces a motion of the end-effector in the task space. The representation of the joint and task spaces is shown in Figure 2.3.

Inverse kinematics is the opposite process. Given the TCP pose with respect to the world frame, the necessary joint values to reach that pose are computed. The inverse kinematics of the KUKA is mathematically complex and it will not be discussed in this thesis.

2.1.1 Types of Kinematic Chains

There are two types of kinematic chains: serial or open kinematic chains and closed kinematic chains.

An open kinematic chain is a chain in which every joint connects only two links. Therefore there is only one sequence of links connecting the two ends of the chain [2]. Open kinematic chains usually resemble a human arm. An example of a manipulator with an open kinematic chain is represented in Figure 2.4. Its structure contains a shoulder, an

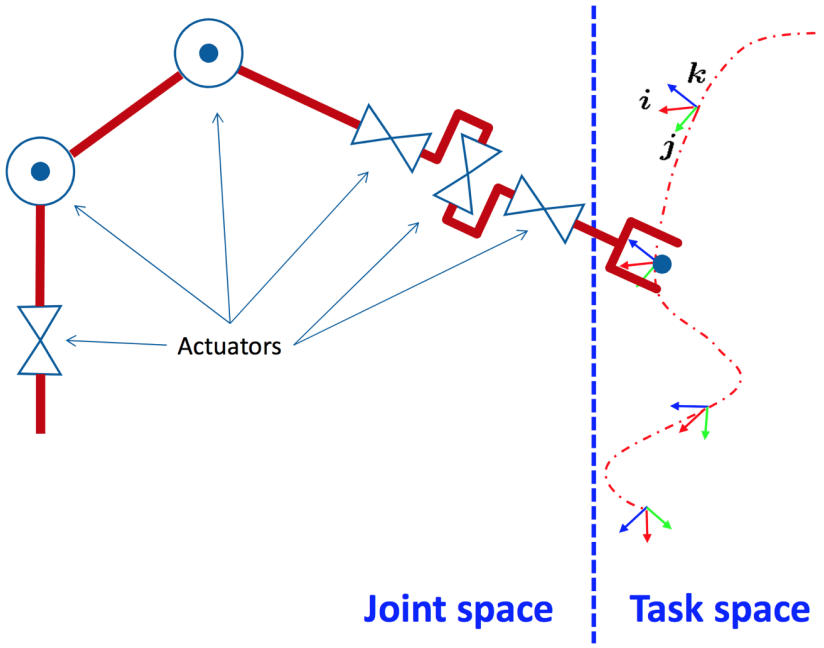


Figure 2.3: Task space vs. Joint space [1]

arm, a forearm, and a wrist. This type of robot provides dexterity but is not accurate inside the task space [1].

A close kinematic chain is a chain where there are more than one link between two joints [1]. The kinematic chain has a cycle-like structure. An example can be provided in Figure 2.5. In this case, the robot's base is mounted above the workspace and three jointed arms extend from the base. The ends of these arms are connected to a small platform. It has high accuracy, but also a small workspace, so the portion of space reachable by the end-effector is limited. Besides, it has a complex kinematic chain.

Figure 2.4: The KUKA KR-15 robot

Figure 2.5: The Delta Wittenstein robot

2.1.2 Direct Kinematics

As described in Section 2.1, the goal of the direct kinematics is to compute the pose of the TCP as a function of the joint variables and robot's geometry with respect to the world frame. To reach this aim, the kinematic chain of a manipulator must be calculated.

Consider an open-chain manipulator composed of n links connected to n+1 joints. To study the robot's open kinematic chain, it is reasonable to consider first the description of the kinematic relationship between two adjacent links and then obtain the overall kinematic chain [2]. To reach this purpose a reference frame for each link must be defined, from link 0 to link n. In order to pass from a link reference frame to another, a transformation of coordinates is done. Then, the coordinate transformation, describing the position and orientation of the frame n with respect to the frame 0, is given by:

$$\mathbf{T}_n^0 = \mathbf{A}_1^0(q_1)\mathbf{A}_2^1(q_2)\dots\mathbf{A}_n^{n-1}(q_n) \tag{2.1}$$

where $\mathbf{A}_i^{i-1}(q_i)$ (for $i = 1, 2, \dots, n$) represent homogeneous trasformation matrices to pass from one reference frame to the other, dependent on q_i, which are the joint variables. Figure 2.6 represents the process explained in Equation (2.1).

The computation of the direct kinematics function is obtained by the product between

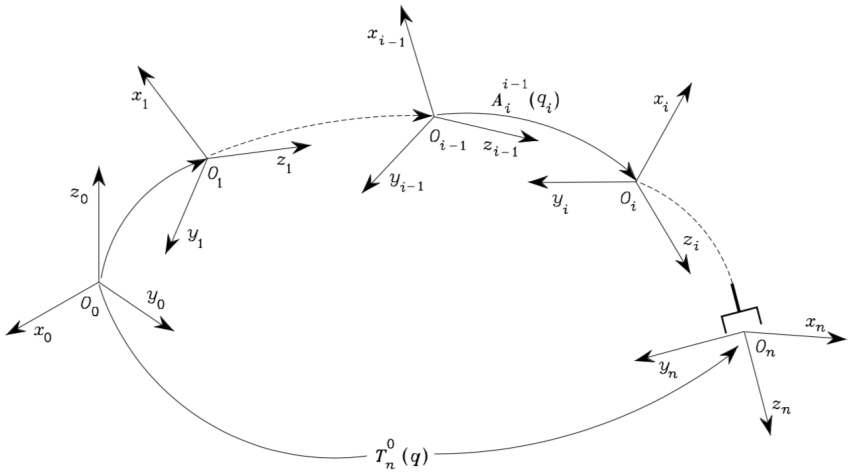


Figure 2.6: The coordinate trasformation from frame 0 to frame n [2]

homogeneous transformation matrices describing the position and orientation of each reference frame. As a consequence, if the frame 0 represents the world frame and the frame n represents the end-effector reference frame, the TCP frame can be calculated with respect to the base frame. The method to attach a reference frame on a link is given by the Denavit–Hartenberg convention, described in Subsection 2.1.3.

2.1.3 Denavit–Hartenberg Convention

The most commonly used notation for selecting reference frames is the Denavit- Hartenberg (DH) convention. This method allows to attach a reference frame to each link of the robot and it uses certain parameters to describe the position and orientation of each reference frame. In particular, it defines the relative position and orientation of two consecutive links. Normally the parameters used to describe the position and orientation of a frame are six. This convention reduces the number of parameters to four. They will be discussed in Subsection 2.1.4.
The first step is to delineate some rules to allocate the reference frames on the robot's links.

The modified version of the DH convention, proposed by Craig [3] is used for this thesis purpose and explained in this Subsection. In Subsection 2.1.5 the difference between the original method and the modified one will be described.
The rules for assigning a rigid reference frame for a given link i of a manipulator is reported below:

- $\hat{Z}_i$ axis of the reference frame i coincides with the joint axis between link i-1 and link i.
- The origin of the reference frame i is located at the intersection of the axis $\hat{Z}_i$ with the common normal to axis $\hat{Z}_i$ and the joint axis between link i and link i+1 (axis i+1).
- $\hat{X}_i$ axis goes along the common normal to $\hat{Z}_i$ and i+1 axes.
- $\hat{Y}_i$ axes is placed following the right-hand rule, once $\hat{X}_i$ and $\hat{Z}_i$ have been located.

Frame 0 is the frame attached to the robot base and does not move. Its Z-axis, $\hat{Z}_0$, lays along axis 1 such that when the joint variable 1 is zero, frame 0 coincides with frame 1. The end-effector frame is attached in a way that the end-effector X-axis ($\hat{X}_n$) aligns with $\hat{X}_{n-1}$ when the last joint angle is zero.

The representation of the frame arrangement is represented in Figure 2.7.

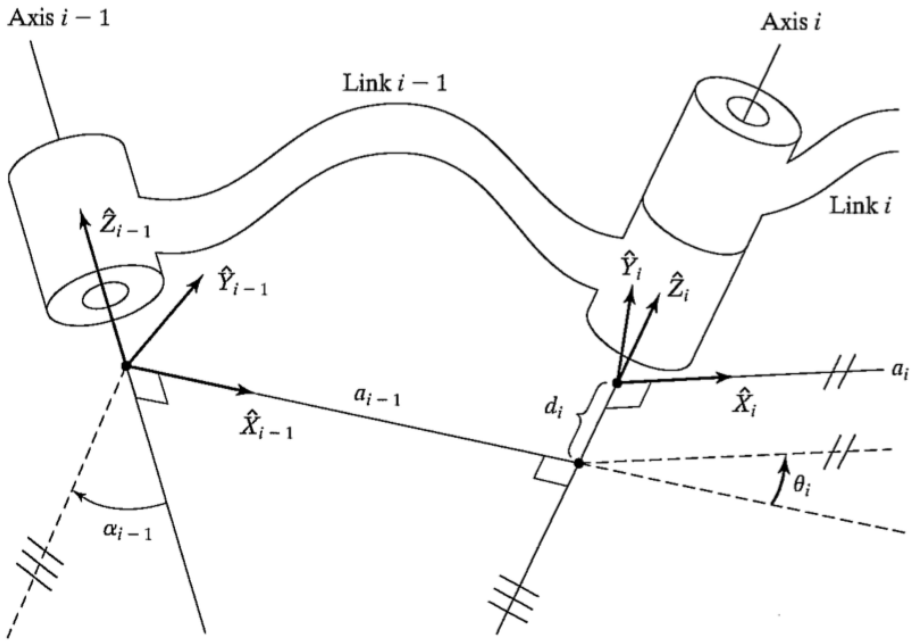


Figure 2.7: Modified DH convention [3]

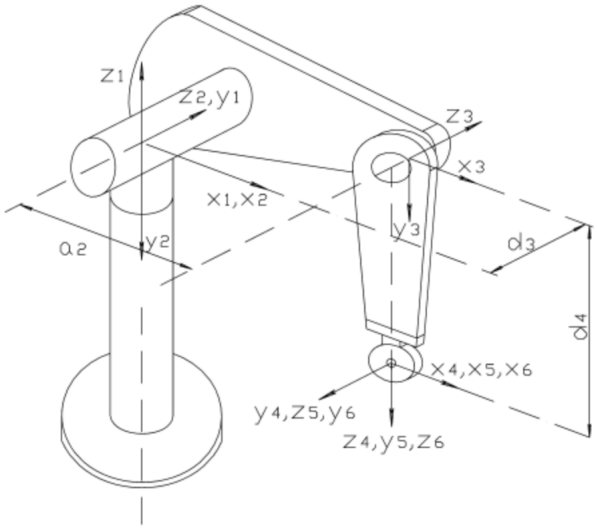


Figure 2.8: The Puma 560 robot with assigned modified DH parameters

2.1.4 Denavit–Hartenberg Parameters

Once the link reference frames have been established, the position and orientation of each frame are defined through a set of parameters known as Denavit–Hartenberg parameters. There are 28 parameters for a serial-robot of seven revolute joints like the KUKA iiwa LBR.

The definition of the parameters for the frame i, discussed in Subsection 2.1.3, is given below [3].

- a_i: the link lenght is the distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured along $\hat{X}_i$.
- α_i: the twist angle is the angle between $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured about $\hat{X}_i$.
- d_i: the offset lenght is the distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured along $\hat{Z}_i$.
- θ_i: the joint angle is the angle between $\hat{X}_{i-1}$ to $\hat{X}_i$ measured about $\hat{Z}_i$.

Two of the four parameters (α_i and a_i) depend only on the geometric connection of link i with the consecutive one (link i+1). One of the other two remaining parameters varies depending on the type of joint (joint i), that connects the previous link (link i-1) with the considered one. In particular:

- If the joint i is a revolute joint, θ_i is the joint variable and so its value changes depending on the relative motion between link i-1 and link i. d_i remains constant and dependent on the geometry of the two links.
- If the joint i is a prismatic joint, the joint variable is d_i. Therefore θ_i depends only on the geometric connection between links i-1 and i.

The representation of these variables for the link i can be seen in Figure 2.7. These parameters applied for a robotic manipulator Puma 560 are shown in Figure 2.8.

At this point, it is possible to express the coordinate transformation between link i-1 and link i following the next steps:

- Choose a frame aligned with frame i-1.

- Translate the chosen frame by a_{i-1} along $\hat{X}_{i-1}$, so:

$$\mathbf{A}_{i'}^{i-1} = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

- Rotate the translated frame by α_{i-1} about $\hat{X}_{i-1}$, hence:

$$\mathbf{A}_{i''}^{i'} = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

- Translate the reference frame by d_i along $\hat{Z}_i$. Then the matrix becomes:

$$\mathbf{A}_{i'''}^{i''} = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

- Rotate the reference frame by θ_i along $\hat{Z}_i$. The homogeneous transformation matrix for passing from the frame i-1 to the frame i thereby becomes:

$$\begin{aligned} \mathbf{A}_i^{i-1} = \mathbf{Transl}(a_{i-1}, \hat{X}_{i-1})\mathbf{Rot}(\alpha_{i-1}, \hat{X}_{i-1})\mathbf{Transl}(d_i, \hat{Z}_i)\mathbf{Rot}(\theta_i, \hat{Z}_i) = \\ = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.5}$$

Now to compute the forward kinematics, as shown in Equation (2.1), the transformation matrices of the (2.5) Equation type must be multiplied to each other. Hence the end-effector position and orientation with respect to the base frame can be derived. The final homogeneous transformation matrix contains a rotation matrix (3x3) and a translation vector (1x3), in the form shown below:

$$\mathbf{T}_{EF}^0 = \begin{bmatrix} \mathbf{R}_{EF}^0 & \mathbf{t}_{EF}^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.6}$$

The position of the TCP with respect to the world frame is described by the translation vector $\mathbf{t}_{EF}^0$, while the three angles, i.e the end-effector orientation, are derived from the rotation matrix $\mathbf{R}_{EF}^0$. In fact $\mathbf{R}_{EF}^0$ can be seen as the result matrix of the product of three elementary rotations around certain axes. Finally, the TCP pose is presented in this mathematical form:

$$\mathbf{x}_e = \begin{bmatrix} x \\ y \\ z \\ u \\ r \\ w \end{bmatrix} \tag{2.7}$$

where x, y and z represent the position of the end-effector with respect to the base frame, while u, r and w the orientation.

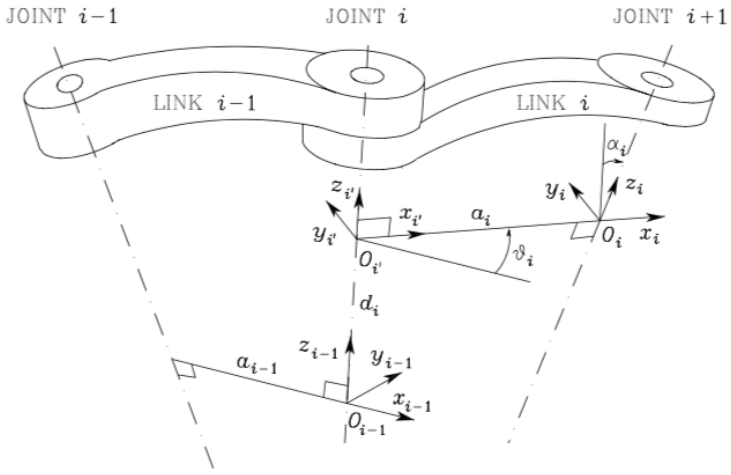


Figure 2.9: Classical DH convention [2]

2.1.5 Difference between DH classical and modified conventions

There are two different forms of the Denavit–Hartenberg method used to calculate the kinematics of a serial-link manipulator [13]:

- The classical convention proposed by Denavit–Hartenberg in [14], used in textbooks such as by Siciliano [2], initially developed for single-close loop kinematic chains and then extended to open-loop chains, such as robotic manipulators [15].
- The modified convention introduced by Craig [3] in his textbook and used in this thesis for calculating the direct kinematics of the KUKA. According to Lipkin [15], it is the most suitable notation for the kinematic analysis of serial manipulators.

The two approaches provide different methods to attach the reference frames to the robot's links. Moreover, both of them use four parameters, two translations (a and d) and two rotations (θ and α), but they calculate them differently.

Given an open-chain manipulator composed of n links connected to n+1 joints, the classical DH convention places the reference frame i on the distal end of link i, i.e where link i and link i+1 meet (Figure 2.9). On the other hand, the modified method locates the i reference frame on the proximal end of link i (where link i-1 meets link i), as shown in Figure 2.7.

The advantage of the modified convention is that the parameters are computed along the frame i axes, therefore this method is transparent and easy to apply. In the classical method, some parameters, in particular θ_i and d_i, are calculated along the z_{i-1} axis.
The disadvantage of the variant method is that it uses multiple indices to switch from a frame to the next one. In fact, the modified convention uses the parameters θ_i, d_i, α_{i-1} and a_{i-1}. The classical one relies on θ_i, d_i, α_i and a_i.

2.2 Calibration

According to Mooring [5], accuracy is the capability of the robot to move the TCP to a defined pose that has not been taught before. To reach that pose, the robot has to move its joints in a manner that the desired location is obtained. This can be calculated using the mathematical model illustrated in Subsections 2.1.2, 2.1.3 and 2.1.4. This model represents an ideal robot: links and joints are manufactured exactly to specifications and the robot is unaffected by any errors in movement [12]. However, the model, used to represent the

manipulator motion, diverges from the actual robot geometry: hence the parameters of the model, that define the end-effector pose, are not accurate [5].

The kinematic calibration copes with all robot inaccuracies that may be arisen after construction, final installation, maintenance or replacements. As a result, the calibration reduces substantially the inaccuracies without making changes to the physical robot but its kinematic model.

2.2.1 Geometric and Nongeometric Errors

A manipulator is subject to many sources of errors that bring to inaccuracies. There are two types of error: geometric and nongeometric errors.

Geometric errors are the result of imprecise manufacturing of robot's part [16]. They depend on the geometric relationship between the links of the manipulator [5]. Furthermore, this type of errors is not connected to the load carried by the robot or to the motion of the end-effector, but, as mentioned before, it includes errors in the joint angle offsets or link lengths due to the impossibility of the perfect robot's part fabrication and matching.

Nongeometric errors are errors that result from external factors. They could be caused by mechanical deflections, due to link and joint flexibility under gravity loading, and thermal errors. These last errors could have a substantial effect on the robot accuracy. The temperature increase could bring to an expansion or contraction of the robot's links, causing a change in the overall structure of the manipulator.

The kinematic calibration method proposed in this thesis copes only with geometric error, while the nongeometric ones are not considered.

2.2.2 Levels of Calibration

According to Mooring [5], three levels of calibration must be distinguished depending on the robot's elements and the target errors of the calibration:

- Level 1 of calibration is defined as joint level calibration. It is also known as mastering. The aim is to correctly relate the signals from the joint displacements transducers to the actual joint displacements [5]. This process is usually performed during the robot building process and repeated only if necessary, such as maintenance or robot's parts substitution. In case the robot contains incremental position transducers or non-absolute transducers [2], this type of calibration has to be executed every time the robot is powered on, to guarantee that the position encoders readings are consistent with the attained manipulator posture [2].
  The relationship between the transducers signals and joint displacements is represented in the following form:
  $$\theta_i = h_i(\eta_i, \boldsymbol{\gamma}_i) \tag{2.8}$$
  where θ_i is the displacement of the joint i, η_i is the signal of the transducer i and $\boldsymbol{\gamma}_i$ is the vector of the parameters in the function $h_i()$.
  In this thesis, level 1 calibration method is not implemented, as the existing relationship between the joint displacements tranduscers and real joint displacements is accurate.

- Level 2 of calibration defines the real robot kinematic model necessary for reaching accuracy. Initially, the ideal mathematical model is taken into consideration, therefore links and joints are considered rigid. In order to model the robot, the modified DH convention is used, as presented in Subsection 2.1.3 and 2.1.4. The modified method describes the robot kinematics using a finite number of parameters. The result of this mathematical model is:
  $$\mathbf{x}_e = \mathbf{k}(\boldsymbol{\eta}, \boldsymbol{\gamma}, \mathbf{K}) \tag{2.9}$$

where $\mathbf{x}_e$ is the end-effector pose vector, represented in the (2.7) Equation form, and $\mathbf{K}$ is the vector of the model parameters (a, α, d and θ). Then the parameters are updated with suitable values in order to describe the real geometry of the robot. The method proposed in this thesis concentrates on this level of calibration.

- Level 3 of calibration is defined as nongeometric or nonkinematic calibration. This type of calibration takes into account nongeometric errors due to effects such as joint compliance, friction, and clearance, as well as link compliance [5]. As a consequence, there are no generalized formulations to comprehend these effects. For this thesis purpose, due to the complexity of this level, this type of calibration is not performed: the proposed kinematic method deals with only geometric errors, as mentioned in Subsection 2.2.1.

2.2.3 The Calibration Process

All levels of calibration explained in Subsection 2.2.2 consist of four steps: modeling, measurement, identification, and implementation. Modeling refers to the suitable mathematical model that represents the robot provided by the modified DH convention: it is described in Subsection 2.1.3 and 2.1.4 and is no longer discussed in this Subsection; in the measurement phase, the end-effector pose of the robot is accurately measured in order to compare it with the model outputs; during the identification, the data recorded from the previous step are used to compute good estimates of the kinematic model parameters; finally, through the implementation step, the nominal model is modified and the corrections are implemented in the position control software of the robot.

2.2.4 Measurement

The goal of this step is to measure either the end-effector pose or some subset of the pose, for a set of joint displacements [5]. Later, the measurement data are necessary to reduce the error between the nominal and real geometries of the robot. An high number of poses must be recorded in order to find more accurate values of the robot's parameters.

The measurement process can be split into two types: open and closed-loop methods. Open-loop methods measure accurately the end-effector pose through external measurement instrumentation: the robot is moved into a position, the joint robot values are read and the TCP pose is fully or partially recorded.
In closed-loop methods, the robot cannot move freely: the end-effector is constrained as if an additional joint were included between the end-effector and the ground [12]. In this case, the robot is moved to a set of given poses that satisfies some constraint. Moreover, joint displacements are recorded at each pose.

Open-Loop Methods

There are different measurement systems for estimating the robot's end-effector pose, depending on the number of the pose components they measure:

- **One component**: the distance between a precise point on the table and a fixed point on the tool is measured through a bar ball, a laser displacement meter or a wire potentiometer. Wire potentiometers (Figure 2.10) are commonly used because they can be automated and quickly used by inexpert users. Generally, the system consists of a potentiometer with a flexible cable that is connected to the robot's tool [4]. The wire is placed so that it does not twist as the arm moves in space. This method is very precise, but it requires a large number of data measurements because only one pose component is obtained.

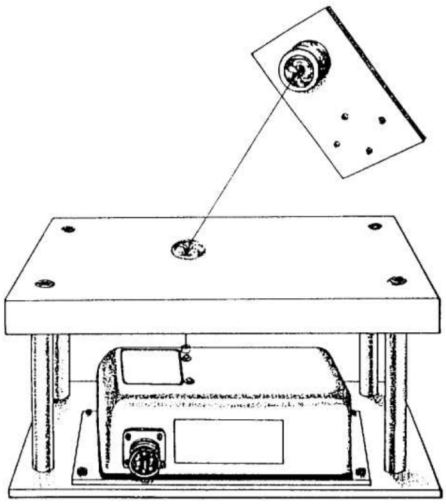


Figure 2.10: Wire potentiometer fixtures [4]

- **Two components**: to measure two orientations of a pose, a theodolite can be used (Figure 2.11). The theodolite is telescope manufactured so that its line of sight is precisely known [5]. It is constituted by three parts: the base, the alidade, and the telescope. The base is usually installed on a tripod: it is adjusted until it is nearly horizontal, as indicated by some bubble levels attached to it. The vertical axis is the one perpendicular to the base. The alidade rotates about the vertical axis and is built so that the rotation is precisely known. The first angle, that the theodolite computes, is the angle between the line of sight and the plane defined by the base. The second one is the angle between an arbitrary chosen horizontal line and the plane formed by the vertical axis and the line of sight [5].
  To use the theodolite, the device is located in a fixed position and the vertical axis is established. Then the operator sees through the telescope until the target is aligned to the telescope viewfinder. Finally, the two angles can be read.
  The theodolite does not provide distance reading, which is very important for the calibration process. Therefore, it is not usually used for the measurement procedure.

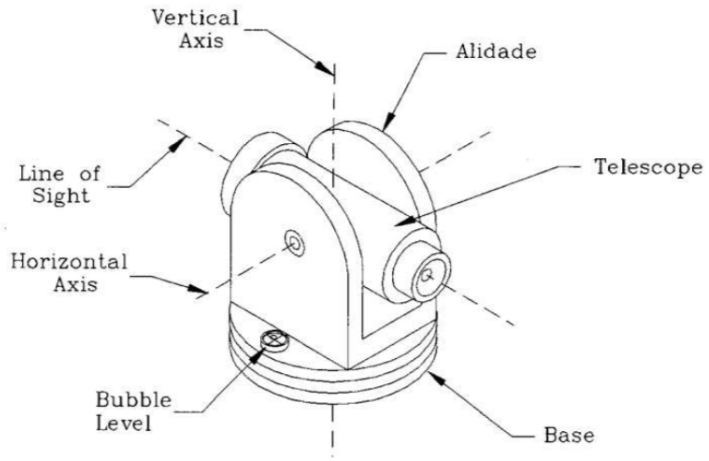


Figure 2.11: A theodolite [5]

- **Three components**: these types of measurement systems can track a fixed point of the robot's tool. An example of these types of equipment is laser interferometers. A laser produces a beam of light that passes through a mirror to direct the beam to a reflective surface mounted on the end-effector. The returning beam is directed to an interferometer to compute the distance between the interferometer and the laser. Photodetectors, on which part of the returning light is pointed to, guide the mirror rotation to follow the robot movements and different poses. This process is usually an automated procedure.
  The resolution of this system is very high, but its accuracy can be influenced by air pressure, humidity, and temperature because they can alter the wavelength of the light. Since these effects are quite small, they are ignored in many cases [5].

- **Six components**: these systems can compute the full pose of the robot's end-effector from the 3D position evaluation of multiple points. This can be done either through stereo camera systems or motion capture equipment. Using the Atracsys, a rigid body can be fixed on the KUKA end-effector so that the position and orientation of the TCP can be evaluated.

The systems presented in this Subsection are the most empolyed and can be used individually or even combined, depending on the desired application.

Closed-loop Methods

Closed-loop methods constraint the manipulator's end-effector movement so that its pose is computed by measuring the joint sensor readings. These approaches limit the cost of the calibration process because it does not require the expensive measuring equipment used in the open-loop method.

The simplest closed-loop method is the manual joint mastering with indicators [12]. In this approach, the joint angles of the robot are constrained. It is assumed that a robot position is known: in this pose, some parts of a link are aligned with other parts of the successive link at a given joint angle. The operator of the robot moves physically the robot until these indicators are aligned: then the joint angle is recorded. The measure alignment can depend on the robot link geometry or can be measured through some indicator markings applied on the link [5]. The result of this approach is very inaccurate because it relies on the user, who judges the position of the robot only by looking at it. Moreover, it is done by using a single pose. It is a calibration that is done when an high accuracy is not required.

Another method is the manual joint mastering with precise measurement [12]. This approach makes use of a built jig with dial gauges to increase the robot's accuracy. The jig must be manufactured and attached to the manipulator's arm. Dial gauges on the jig allow measuring small differences in angles so that an operator can adjust the robot pose until it matches with the desired one.
This process reaches a better accuracy with respect to the method explained before, because it does not rely on the user sight, but it depends on the manufacturing precision of the jig itself. Also this method makes use of one single pose.

2.2.5 Identification

Identification allows finding accurate estimates of the DH parameters from a series of measurements on the manipulator's end-effector pose, obtained from the measurement process [2]. Therefore, the identification step aims to reduce the error between the output of the robot nominal model and the measured pose.

As mentioned in Subsection 2.2.2, the end-effector pose is dependant on many factors, in particular on the theoretical DH parameters:

$$\mathbf{x}_e = \mathbf{k}(\mathbf{a}, \boldsymbol{\alpha}, \mathbf{d}, \boldsymbol{\theta}) \tag{2.10}$$

where $\mathbf{a} = [a_1\, a_2\, \ldots\, a_n]^T$, $\boldsymbol{\alpha} = [\alpha_1\, \alpha_2\, \ldots\, \alpha_n]^T$, $\mathbf{d} = [d_1\, d_2\, \ldots\, d_n]^T$, and $\boldsymbol{\theta} = [\theta_1\, \theta_2\, \ldots\, \theta_n]^T$ denote the vectors of nominal DH paramenters of a n-links robot.

Let $\mathbf{x}_m$ be the measured pose, obtained from the measurement step, and $\mathbf{x}_n$ the nominal pose, computed via Equation (2.5) and Equation (2.1) with the nominal values of the parameters $\mathbf{a}$, $\boldsymbol{\alpha}$, $\mathbf{d}$, and $\boldsymbol{\theta}$. The two poses are represented in the form given by Equation (2.7). The deviation $\boldsymbol{\Delta}\mathbf{x} = \mathbf{x}_m - \mathbf{x}_n$ represents the robot's accuracy at the given position. Assuming that the deviation is small, at first approximation, the error $\boldsymbol{\Delta}\mathbf{x}$ is derived by using the following equation [2]:

$$\boldsymbol{\Delta}\mathbf{x} = \frac{\partial \mathbf{x}_n}{\partial \mathbf{a}}\boldsymbol{\Delta}\mathbf{a} + \frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\alpha}}\boldsymbol{\Delta}\boldsymbol{\alpha} + \frac{\partial \mathbf{x}_n}{\partial \mathbf{d}}\boldsymbol{\Delta}\mathbf{d} + \frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\theta}}\boldsymbol{\Delta}\boldsymbol{\theta} \tag{2.11}$$

where $\boldsymbol{\Delta}\mathbf{a}$, $\boldsymbol{\Delta}\boldsymbol{\alpha}$, $\boldsymbol{\Delta}\mathbf{d}$, and $\boldsymbol{\Delta}\boldsymbol{\theta}$ are the devations between the real parameters of the robot and the nominal parameters of the theoretical model. Moreover, $\frac{\partial \mathbf{x}_n}{\partial \mathbf{a}}$, $\frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\alpha}}$, $\frac{\partial \mathbf{x}_n}{\partial \mathbf{d}}$, and $\frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\theta}}$ represent the $(6 \times n)$ matrices which contain the partial derivatives of the $\mathbf{x}_n$ components with respect to the single parameters. In particular, these matrices are the Jacobians of the transformations between the joint space and the task space. For any given parameters (for example, the link lenghts) the Jacobian is calculated using Equation (2.12):

$$\mathbf{J}_a = \frac{\partial \mathbf{x}_n}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\delta x}{\delta \mathbf{a}_1} & \frac{\delta x}{\delta \mathbf{a}_2} & \cdots & \frac{\delta x}{\delta \mathbf{a}_n} \\ \frac{\delta y}{\delta \mathbf{a}_1} & \frac{\delta y}{\delta \mathbf{a}_2} & \cdots & \frac{\delta y}{\delta \mathbf{a}_n} \\ \frac{\delta z}{\delta \mathbf{a}_1} & \frac{\delta z}{\delta \mathbf{a}_2} & \cdots & \frac{\delta z}{\delta \mathbf{a}_n} \\ \frac{\delta u}{\delta \mathbf{a}_1} & \frac{\delta u}{\delta \mathbf{a}_2} & \cdots & \frac{\delta u}{\delta \mathbf{a}_n} \\ \frac{\delta r}{\delta \mathbf{a}_1} & \frac{\delta r}{\delta \mathbf{a}_2} & \cdots & \frac{\delta r}{\delta \mathbf{a}_n} \\ \frac{\delta w}{\delta \mathbf{a}_1} & \frac{\delta w}{\delta \mathbf{a}_2} & \cdots & \frac{\delta w}{\delta \mathbf{a}_n} \end{bmatrix} \tag{2.12}$$

Grouping the parameters in the vector $\boldsymbol{\zeta} = [\mathbf{a}^T\, \boldsymbol{\alpha}^T\, \mathbf{d}^T\, \boldsymbol{\theta}^T]$, defining the deviation between the real parameters and nominal ones as $\boldsymbol{\Delta}\boldsymbol{\zeta} = \boldsymbol{\zeta}_m - \boldsymbol{\zeta}_n$, and grouping the partial derivatives matrices in the $(6 \times 4n)$ matrix $\boldsymbol{\Phi} = [\frac{\partial \mathbf{x}_n}{\partial \mathbf{a}}\, \frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\alpha}}\, \frac{\partial \mathbf{x}_n}{\partial \mathbf{d}}\, \frac{\partial \mathbf{x}_n}{\partial \boldsymbol{\theta}}]$, Equation (2.11) can be compactly rewritten as:

$$\boldsymbol{\Delta}\mathbf{x} = \boldsymbol{\Phi}\boldsymbol{\Delta}\boldsymbol{\zeta} \tag{2.13}$$

To perform the identification process, it is desired to compute $\boldsymbol{\Delta}\boldsymbol{\zeta}$ starting from the knowledge of $\boldsymbol{\Delta}\mathbf{x}$, $\mathbf{x}_n$, and $\mathbf{x}_m$ measurements [2]. Equation (2.13) is a system of six equations with $4n$ unknowns, therefore enough measurements have to be acquired to obtain a system of at least $4n$ equations [2]. If l poses are recorded, Equation (2.13) yields:

$$\overline{\boldsymbol{\Delta}\mathbf{x}} = \begin{bmatrix} \boldsymbol{\Delta}\mathbf{x}_1 \\ \vdots \\ \boldsymbol{\Delta}\mathbf{x}_l \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_1 \\ \vdots \\ \boldsymbol{\Phi}_l \end{bmatrix} \boldsymbol{\Delta}\boldsymbol{\zeta} = \overline{\boldsymbol{\Phi}}\boldsymbol{\Delta}\boldsymbol{\zeta} \tag{2.14}$$

As regards the nominal values needed for the computation of the matrices $\boldsymbol{\Phi}_i$, the geometric parameters are always constant, while the joint values depend on the configuration of the robot at the pose i.

To avoid ill-conditioning of the matrix $\overline{\mathbf{\Phi}}$, it must be chosen l poses such that $lm \gg 4n$. Then the Equation (2.14) can be solved using an ordinary least-square technique: the solution of (2.14) is:

$$\mathbf{\Delta\zeta} = (\overline{\mathbf{\Phi}}^T\overline{\mathbf{\Phi}})^{-1}\overline{\mathbf{\Phi}}^T\mathbf{\Delta}\overline{\mathbf{x}} \tag{2.15}$$

where $(\overline{\mathbf{\Phi}}^T\overline{\mathbf{\Phi}})^{-1}\overline{\mathbf{\Phi}}^T$ is the pseudo-inverse of the matrix $\overline{\mathbf{\Phi}}$. The DH parameters are updated by using the following equation:

$$\boldsymbol{\zeta}' = \boldsymbol{\zeta}_n + \mathbf{\Delta\zeta} \tag{2.16}$$

The explained procedure should be iterated until $\mathbf{\Delta\zeta}$ converges to within an arbitrary threshold.

At the first iteration, $\boldsymbol{\zeta}_n$ corresponds to the nominal values of the kinematic model parameters. As the iteration progresses, the calibration matrix $\overline{\mathbf{\Phi}}$ has to be updated with the parameters obtained via Equation (2.16) at the previous iteration: therefore $\boldsymbol{\zeta}_n$ is equal to $\boldsymbol{\zeta}'$ of the previous iteration and $\boldsymbol{\zeta}_n$ is used to compute the $\mathbf{\Phi}_i$ matrices. Similarly, $\overline{\mathbf{\Delta x}}$ has to be calculated as a difference between the measured values for the l end-effector positions and the corresponding poses computed by the direct kinematics function with the values of the parameters at the previous iteration [2]. As a result, more accurate estimates of the real robot geometric parameters, as well as possible corrections to make on the joint transducers measurements, are found. The flowchart of the identification process is shown in Figure 2.12

As mentioned in Subsection 2.2.2, the calibration in the KUKA joint values is not performed, because the joint sensors' readings are considered precise so that the matrices $\mathbf{\Phi}_i$ do not contain the Jacobian regarding the θ values.

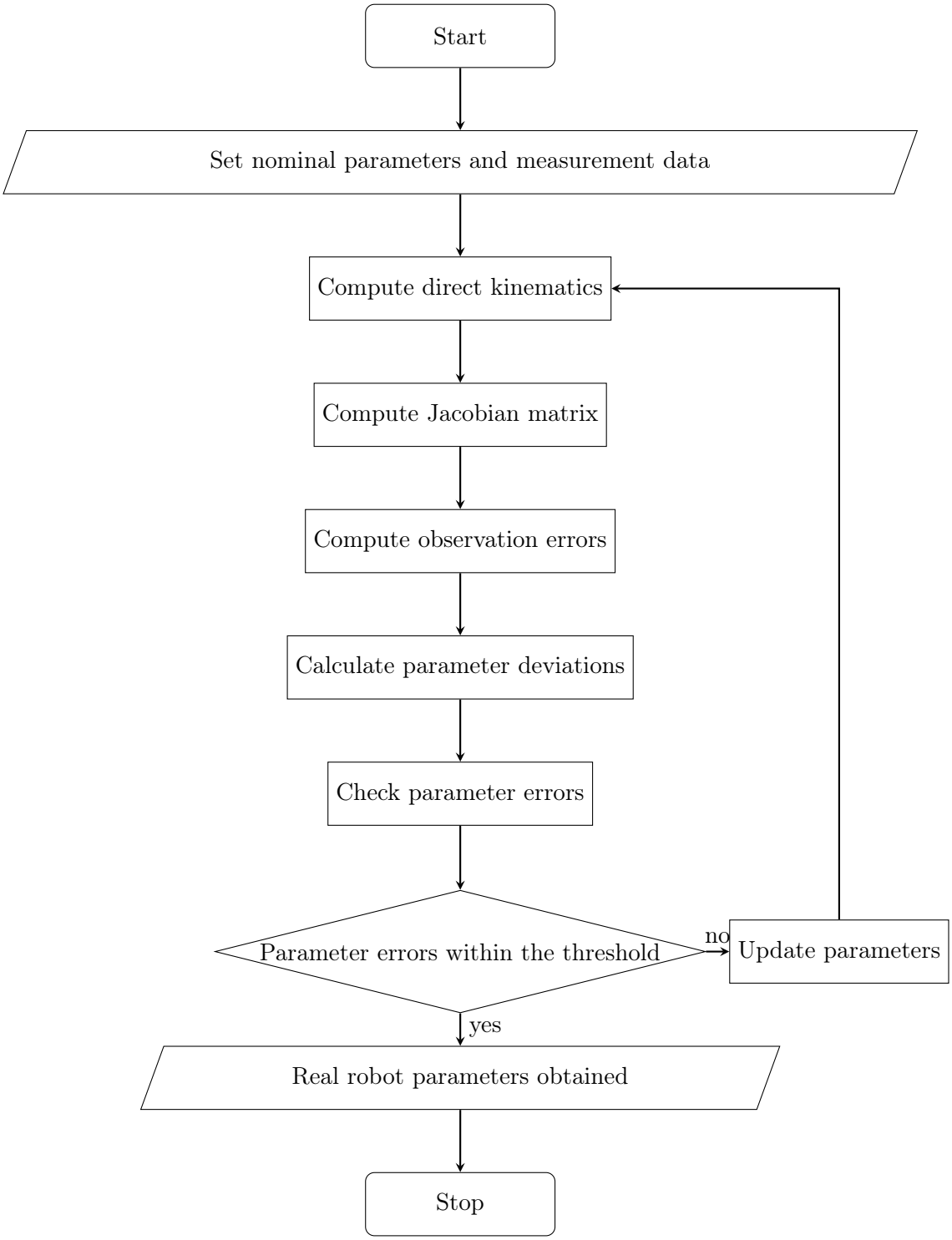


Figure 2.12: Identification process flowchart

Chapter 3

The System

The approaches and techniques described in Chapter 2 can be applied to a great variety of serial-robots to increase their accuracy. For this thesis purpose, an open-loop calibration method is performed to guarantee a precision positioning performance of a seven revolute joints serial-manipulator KUKA iiwa LBR 7R 800 (KUKA). The measurement system employed is the tracking system Atracsys fusionTrack 500 (Atracsys), which is needed to record the manipulator tool's pose. In particular, using the Robot Operating System (ROS) framework, the robot is programmed to move from one position to another, to record the joint angles readings and the end-effector information, once the pose is reached. In the following chapters, the algorithms developed to move the robot and perform the calibration are explained, as well as the results obtained from the algorithms.

In this chapter, the manipulator and tracking system characteristics are illustrated and an overview of the ROS framework is described, to have a better understanding of the next chapters' contents.

3.1 KUKA iiwa LBR

The KUKA iiwa LBR 7 R800 is a robotic arm with seven revolute joints, currently employed for manufacturing applications and research projects. It is a sensitive and Human-Robot Collaboration (HRC) compatible manipulator. In particular, LBR stands for lightweight robot and iiwa for intelligent industrial work assistant [17]. It is designed to work together with humans in close cooperation (Figure 3.1).

(a) *The robot [18]* (b) *Human and robot collaboration [17]*

Figure 3.1: The KUKA iiwa LBR

KUKA Motion Range	
Joint	*Range [deg]*
A1	[-170, +170]
A2	[-120, +120]
A3	[-170, +170]
A4	[-120, +120]
A5	[-170, +170]
A6	[-120, +120]
A7	[-175, +175]

Table 3.1: KUKA motion range table

Every robot's axis has multiple sensors that provide signals for robot control, such as position, velocity, impedance and torque controls, and they are used as a protective function of the robot. The sensors that the axis contains are:

- Axis range sensors: they ensure that the axis range is adhered to. The range of motion of each axis is shown in Table 3.1.
- Torque sensors: they ensure that axis loads are not exceeded. In particular, this type of sensors makes the KUKA sensible and quick reacting. Thanks to the torque sensors, the LBR iiwa can detect contact immediately and consequently reduce its level of force and speed instantly.
- Temperature sensors: they monitor the thermal limit values of the electronics. When the thermal limit values are exceeded, the KUKA immediately shuts down.

The kinematic system of the KUKA is of redundant design due to its seven axes and consists of the following principal components as shown in Figure 3.2:

- 1 indicates the two-axis in-line wrist with two motors located in the last two-axis, A6 and A7.
- 2 represents the joint modules. They consist of an aluminum structure. Moreover, they contain and link the drive units to one another.
- 3 is the base frame, which is the base of the robot.

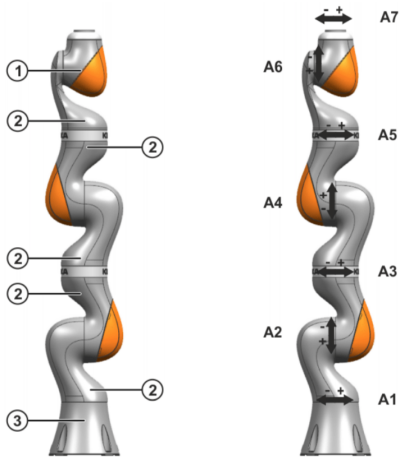


Figure 3.2: Main assemblies and robot axes. Axes are enumerated from A1 to A7 [6].

KUKA Nominal Modified DH Parameters				
Link	*theta [deg]*	*d [m]*	*a [m]*	*alpha [deg]*
1	0	0.340	0	0
2	0	0	0	-90
3	0	0.400	0	90
4	0	0	0	90
5	0	0.400	0	-90
6	0	0	0	-90
7	0	0.126+0.035	0	90

Table 3.2: The nominal DH parameters of the KUKA iiwa LBR 7R 800

In particular, the redundant design guarantees great flexibility of the manipulator's use, since it allows the robot to reach inaccessible places by selecting the most favorable configuration, given the position and orientation of the end effector.

The kinematic model of the KUKA is based on the mathematical model described in Subsection 2.1.3 and makes use of the parameters illustrated in Subsection 2.1.4. The theoretical values of the variables are presented in Table 3.2, where 0.035 m indicates the offset length between the KUKA last flange and the electrical end-effector mounted on the top of the robot (Figure 3.3): this height is included in link 7.

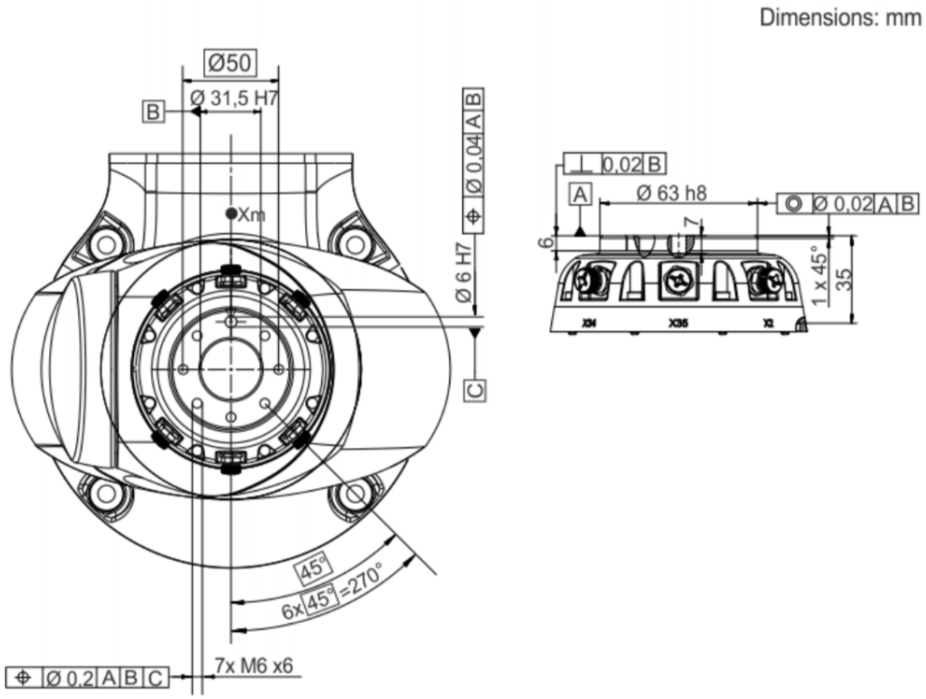


Figure 3.3: Dimensions, media flange electrical. The media flange is represented with axis 7 in the zero position. The symbol **Xm** depicts the position of the locating element in the zero position [7]

Figure 3.4: The Atracsys fusionTrack 500

3.2 Atracsys fusionTrack 500

The fusionTrack 500 is a passive and active, real-time optical pose-tracking system specially designed to detect and track reflective spheres, disks and IR-LEDs in real-time within a specified volume [8]. In particular, the reflective spheres or disk are called active fiducials, while the IR-LEDs are referred to as passive fiducials. The Atracsys system is represented in Figure 3.4.

The device is composed of two cameras that can observe passive or/and active fiducials simultaneously and it uses triangulation to calculate their positions with high precision (90 µm RMS at distances up to 2 m, as shown in Figure 3.5). Since the Atracsys can detect more fiducials at the same time, it can also track fiducials placed in a specific geometry. These fiducials are attached to a marker: therefore the system can determine the marker's position and orientation. A marker is an object on which geometry of fiducials is affixed. However, the fiducials on the marker must have an asymmetrical arrangement, because the fusionTrack recognizes the fiducials only if the distances between them are different from one another. Hence, each distance between two fiducials is unique and can be easily recognized by the two-cameras device, thus the marker pose based on this geometry is determined. The Atracsys can detect simultaneously up to sixteen markers, each marker containing 4 fiducials.

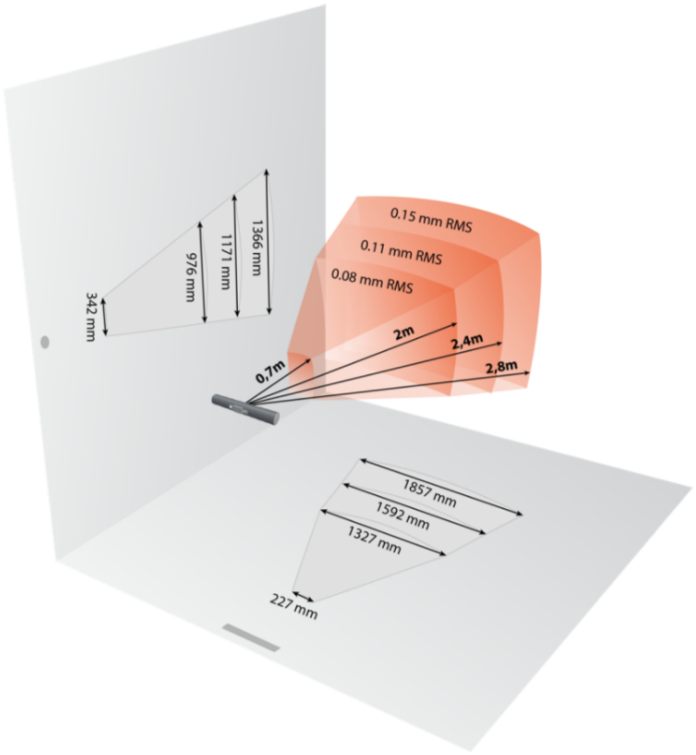


Figure 3.5: The fusionTrack tracking volume [8]

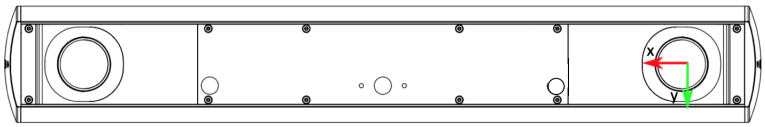


Figure 3.6: The coordinates system, front view (modified image from [8])

The next lines explain how the fusionTrack tracks the fiducials [8]:

- The Atracsys illuminators emit infrared (IR) light.
- The IR light reflects on passive markers or triggers active markers which then emit IR light.
- The fusionTrack tracks the reflected or emitted IR light and transmits the information to the host computer to which is connected and on which the Atracsys Software Development Kit (SDK) is installed or the libraries of the SDK are used.
- The host computer extracts the position of each detected spot, computes the 3D position of each detected source, then tries to match them to known marker geometries.
- The information is displayed to the user, with status information on each piece of data, allowing the user to do additional processing (display, filtering, collection, etc.).

Pose data of the markers are reported with respect to the Atracsys right-handed coordinates system, placed by default in the optical center of the left camera, as shown in Figure 3.6. Moreover, in order to be tracked by the fusionTrack device, the markers must lie inside the working volume illustrated in Figure 3.5.

Furthermore, the SDK provides libraries and applications for tracking the markers according to the needs of the tasks it has to perform. Then, these libraries and applications are used to develop a ROS package in order to make possible the exchange of pose data between the Atracsys and the algorithm that processes the data. This will be explained in the following chapter (Chapter 4).

3.3 ROS: Robot Operating System

The Robot Operating System (ROS) is an open-source and flexible software framework for writing complex robot applications. ROS provides a hardware abstraction layer, where developers can build robotics programs without worrying about the underlying hardware [9]. Moreover, it offers tools and libraries for obtaining, building, writing and running codes, as well as visualizing and processing robot data, across different robotics platforms and multiple computers, to encourage collaborative program development [19].

ROS philosophy can be summarized in three main principles:

- ROS is a peer-to-peer framework: it allows the individual programs to communicate with each other in different ways, synchronously or asynchronously, according to the needs.
- It is multi-language: ROS can be implemented in different programming languages, such as C, C++, Python, and Matlab.
- Program sharing: one of ROS goals is to make the coding part as easy as possible by sharing fundamental codes.

The core of the ROS framework is a message-passing middleware in which processes, called nodes, can communicate and exchange data with each other even when running from different machines [9]. Nodes are capable of sending and receiving messages from every node. To understand how the communication between nodes takes place, some definitions have to be described:

- **ROS Master**: ROS Master manages the communication between nodes. Every node registers at startup with the master.
- **ROS Node**: a ROS node is a process that performs computation. It can be individually computed, managed and executed. Moreover, it is organized in packages.
- **Topic**: nodes are combined into a graph and communicate with one another using streaming of topics. Topics are streams of messages which the nodes send to each other. In particular, nodes can publish and subscribe to a topic. Normally there is one node that operates as a publisher and multiple nodes that act as subscribers. The publisher sends a message on the topic by publishing the message, while the subscriber receives the message published on the topic by listening to this topic. All nodes could work as both publishers and subscribers at once.
- **ROS Package**: The software in ROS is mainly organized in ROS packages. A package consists of ROS nodes, datasets, and configuration files, organized in a single module [9].

An example of how the communication works on ROS can be seen in Figure 3.7. There are two nodes, one named talker and the other listener. The talker node publishes a string message containing "Hello World" into a topic called /talker, and the listener node subscribes to this topic. The communication is divided into stages, that are marked as (1), (2) and (3).

(1) Before running any codes, the ROS master must be started. After it has been initialized, it waits for nodes. When the talker node starts running, it first connects to the master and then it begins to exchange the publishing topic details, such as topic name and message type, with the master. The master maintains tables of the publisher connected to it. Whenever a publisher's details change, the table updates automatically [9].

(2) When the listener node is started, it connects to the master and exchanges the details of the node, such as the topic it subscribes to and its message type. The master also maintains a table of subscribers, similar to the publisher.

(3) When there are a publisher and a subscriber that employ the same specific topic, the master node exchanges the publisher details with the subscriber. This helps nodes to connect and exchange data. After they have connected, there is no role for the master. The data is not flowing through the master; instead, the nodes are interconnected and exchange messages [9].

In conclusion, this framework is a powerful tool for robotics applications and allows the management of different systems, such as the one used for the kinematic calibration. During the measurement phase, the robot and the Atracsys must be used simultaneously and ROS creates a perfect environment where each node controls a specific piece of the equipment. For this thesis purpose, ROS must be installed and some packages creation and use are required. The procedure to install ROS Kinetic on Ubuntu 16.04 is described in Appendix A.

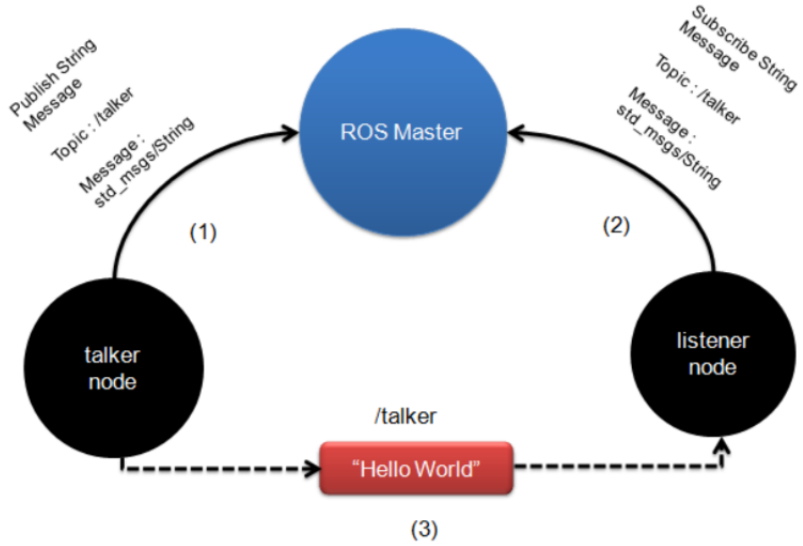


Figure 3.7: Communication between ROS nodes using topics [9]

Chapter 4

A KUKA iiwa LBR Calibration Method

In this chapter, a method to calibrate the KUKA iiwa LBR is described in detail, based on the techniques explained in Chapter 2. To understand the developed algorithms, the first step aims to illustrate the ROS packages used to control the KUKA and to register the robot's data. Then the modeling process for the KUKA is presented and the DH parameters used to build the kinematic model of the robot are shown. Afterward, the two codes are depicted: the first algorithm allows the KUKA to move to a random position, with the end-effector always turned toward the tracking system. Furthermore, it registers the robot's joint angles and the position of the end-effector with respect to the base frame. The second algorithm computes the calibrated DH parameters based on the data acquired from the measurement phase, according to the process described in Section 2.2.5. Finally, to verify the results, a test is performed: a comparison between the data obtained from the Atracsys and the outputs of the kinematic model with the updated parameters is done.

4.1 ROS Packages

The following two ROS packages are used to perform the measurement phase of the calibration process.
One package is available on `https://github.com/jonaitken/KUKA-IIWA-API`, and allows to move the robot's end-effector to a specific location, and it can also read the manipulator's joint angles. The other package is needed to acquire the robot's data from the Atracsys. In particular, it is based on the Atracsys SDK program that observes a particular marker and computes its position and orientation.

4.1.1 ROS-integrated API for the KUKA LBR iiwa collaborative robot by Sheffield Robotics

Sheffield Robotics presents an Application Programming Interface (API) for the KUKA iiwa LBR. The API is designed to be simple and to interface to ROS to provide an easy platform development [10].

The KUKA iiwa LBR is provided in an industrial robot system, that consists of the following components (Figure 4.1):

- Manipulator;
- KUKA Sunrise Cabinet robot controller;
- KUKA smartPAD control panel;

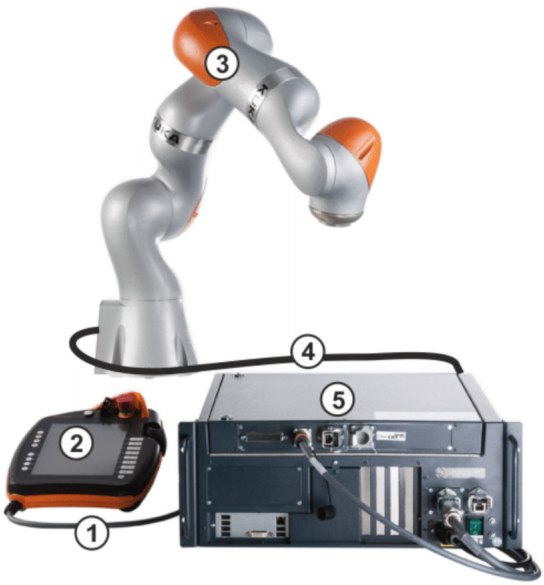


Figure 4.1: Overview of robot system [6]. 1 is the connecting cable to the smartPAD, 2 is the KUKA smartPAD control panel, 3 is the manipulator, 4 is the connecting cable to KUKA Sunrise Cabinet robot controller, and 5 is the KUKA Sunrise Cabinet robot controller

- Connecting cables;
- Software;
- Options and acessories, such as the media electric flange.

The API architecture focuses on breaking out the functionality that would normally be available within the KUKA Sunrise controller run on the Smartpad [10]. It aims to extend the capability of the KUKA, using the structure shown in Figure 4.2.

The KUKA daemon is a program in Java, included in the Sunrise operating system of the KUKA Sunrise Cabinet robot controller, that handles some generic and controlling tasks, such as collision detection. This program is necessary to move the KUKA safely while using the library developed by Sheffield Robotics. It is provided by the GitHub

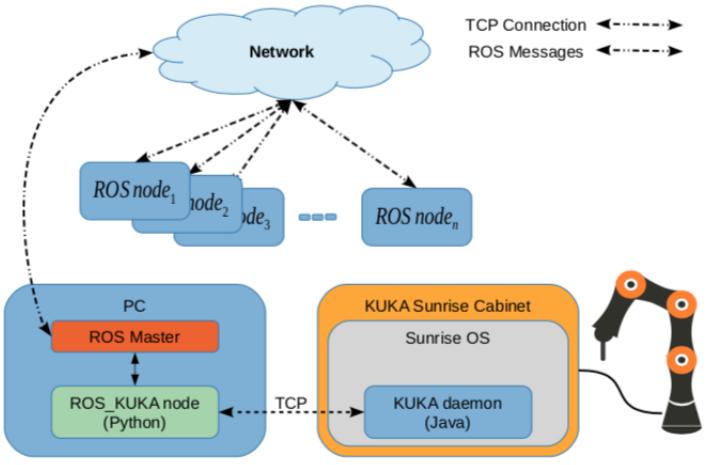


Figure 4.2: Components of the KUKA ROS interface architecture [10]

Topic	**Data Structure**
JOINT POSITION	Joint Position [A1, A2, A3, A4, A5, A6, A7, A8] time-stamp
TOOL POSITION	Tool Position [x, y, z, A, B, C] time-stamp
TOOL FORCE	ToolForce [F_x, F_y, F_z] time-stamp
TOOL TORQUE	ToolTorque [τ_x, τ_y, τ_z] time-stamp
JOINT ACCELERATION	JointAcceleration [a_1,a_2, a_3, a_4,a_5,a_6, a_7] time-stamp
JOINT VELOCITY	JointVelocity [v_1, v_2, v_3, v_4, v_5, v_6, v_7] time-stamp
IS COMPLIANCE	isCompliance bool time-stamp
IS JOINT COMPLIANCE	isCompliance bool time-stamp
IS COLLISION	isCollision bool time-stamp
IS JOINT OUT OF RANGE	isJointOutOfRange bool time-stamp
OPERATION MODE	OperationMode type time-stamp

Table 4.1: Kuka LBR iiwa data available through ROS topics

repository on which the package can be found. A ROS KUKA node is also implemented in the Python language: it is a ROS node that plays as an intermediate between the KUKA daemon and the ROS master. It subscribes to the topics containing commands published by other ROS nodes and passes them to the KUKA daemon. In the same way, it receives status information of the KUKA from the Java application and publishes it on specific topics.

By running the ROS node on an external pc, rather than KUKA Sunrise OS, it is possible to preserve robot's safety protocols, because no modification of the KUKA Sunrise Cabinet is required. Essentially more features are enhanced and added instead of altered.

The main topics and data structure of ROS messages are shown in Table 4.1.

The "Joint Position" topic is a string array, containing the joint positions expressed in degrees. It is also possible to obtain the tool position through the "Tool position" topic: it gives a string array in which x, y and z are the end-effector cartesian positions expressed in millimeters, and A, B and C are the values of the orientation angles in degrees. Moreover, the torque and force values measured at the robot's tool are available. The former is expressed in Newton and the latter in Newton meter. Axis-specif joint acceleration and velocity are published in "Joint acceleration" and "Joint Velocity" topics respectively.

In addition to the standard position control mode, it is possible to use either the Cartesian Impedance or Joint Impedance control modes. In the "Is Compliance" topic a boolean variable is published. When the value is true, the robot is in Cartesian Impedance control mode. Similarly, in the "Is Joint Compliance" topic, a string array containing a boolean value is published. Also in this case, if the value is true, the robot is operating in Joint Impedance control mode.
Boolean values are also published in both "Is Collision" and "Is Joint Out of Range" topics. The first one contains a true value if a collision is detected, the second one publishes a true value when any joint is out of range. Finally, in the "Operation Mode" topic, it is possible to read in which mode the robot is working. The possible operating modes are manual reduced velocity or T1 mode, manual high velocity or T2 mode and automatic mode or AUT.

As described before, nodes can send several commands to the robot through the ROS KUKA node. Hence, from the nodes side, the command string must be published in the "Kuka_Command" topic. The string array is sent to the robot with a specific codification for each command that can be found in Table 4.2.

Command Description	**Command Structure**
SET JOINT VELOCITY	setJointVelocity v
SET JOINT ACCELERATION	setJointAcceleration a
SET JOINT JERK	setJointJerk j
SET CARTESIAN VELOCITY	setCartesianVelocity v
SET JOINT POSITION	setPosition A1 A2 A3 A4 A5 A6 A7
SET CARTESIAN POSITION	setPositionXYZABC x y z A B C ptp/lin
MOVE END EFFECTOR WITH POINT TO POINT MOTION	MoveXYZABC x y z A B C
MOVE END-EFFECTOR WITH CIRCULAR MOTION BETWEEN TWO POINTS	MoveCirc x_1 y_1 z_1 A_1 B_1 C_1 x_2 y_2 z_2 A_2 B_2 C_2
SET COMPLIANCE	setCompliance x y z A B C
RESET COMPLIANCE	resetCompliance
RESET COLLISION	resetCollision
FORCE STOP	forceStop
SET WORKSPACE	setWorkspace x_{min} y_{min} z_{min} x_{max} y_{max} z_{max}

Table 4.2: Main Commands for KUKA LBR robot

The interface allows setting the joint velocity, as well as the acceleration, and the tool's velocity. It can also set the robot position by defining the target position in both joint and cartesian space. In particular, for the cartesian space, the end-effector can be moved from one point to another using either linear, point-to-point or circular motions. In the linear movement, the end-effector is moved linearly, while in the point-to-point one, the tool follows the fastest path to reach the new position.

The KUKA robot can be operated in two different control types: Position Control mode and Compliance Control mode. The first mode aims to execute the programmed path with maximum position accuracy and without path deviation. In the second one, the controller is modeled on a virtual spring-damper system. With the "Set Compliance" command, the robot compliance mode with a particular stiffness in each x, y, z, A, B, C is activated. To deactivate the robot compliance mode, the "Reset Compliance" command must be sent.

The "Reset Collision" command resets a collision if any collision has been detected, while the "Force Stop" command acts as an emergency stop, hence it stops the robot and removes all the robot motion queue waiting to be executed. Finally, the "Set Workspace" command defines a cubic end-effector workspace boundaries.

For this thesis purpose, the KUKA is operated in automatic mode and makes use of the position control mode. The KUKA is moved to random positions inside a predefined workspace: then, each position is recorded by the Atracsys. To use the commands and topics explained above, a package is built inside the ROS workspace according to Appendix A with the name `kuka`. Inside its `src` folder, a folder with the name `python` is generated. This folder contains the ROS KUKA node and the library provided by the Sheffield Robotics. Moreover, the KUKA daemon program must be uploaded to the KUKA Sunrise operating system and selected on the smartPAD so that the communication between the ROS KUKA node and the KUKA can take place.

4.1.2 The Atracsys ROS Package

The Atracsys package aims to define a ROS node for the Atracsys fusionTrack 500. As mentioned in Section 3.2, the Atracsys is a tracking system able to define the positions and orientations of multiple markers with respect to its coordinates frame. The node, based on the libraries and a simple program provided by the Atracsys SDK, detects a marker and publishes its pose on a topic, which has the same marker name and geometry ID, a unique number that labels this marker. As a result, if more markers need to be tracked, more topics with the markers' names and IDs can be seen: in particular, each topic contains the cartesian positions and quaternion of the respective marker.

In order to use the Atracsys, the device must be installed on the computer (see Appendix B) and the geometry of the desired marker to be tracked has to be defined. The marker geometry must be uploaded in the `geometry` folder of the package. It is a `.ini` file and it comes in the following form:

```
; Example of geometry file
[geometry]
count=3
id=3
[fiducial0]
x=0.000000
y=26.870000
z=3.000000
[fiducial1]
x=-23.690000
y=-22.770000
z=3.000000
[fiducial2]
x=26.120000
y=-27.140000
z=3.000000
[pivot]
x=0.000000
y=0.000000
z=0.000000
```

The name of the `.ini` file is the name of the geometry. The `[geometry]` section contains `count` and `id`. Count defines the number of fiducials used on the marker (at least 3, at most 6). Id is the unique number arbitrary given to the geometry. Then the cartesian position of each fiducial with respect to an arbitrary pivot must be determined. The Atracsys computes the position and orientation of the marker reference frame centered on the pivot, in accordance with the fiducials' positions.

The Atracsys node is written in C++ and uses the libraries provided by the Atracsys SDK. Moreover, it makes use of some functionalities defined in the sample program "stereo2_AcquisitionBasic" provided by the SKD such as initializing the fusionTrack, uploading the geometry, tracking the geometry and closing the driver connection.

First the ROS node is initialized: its name is "AtracsysTrackingNode". Then the name of the publisher is defined in the following line:

```
std::vector<ros::Publisher> Atrapublisher_;
```

`Atrapublisher_` publishes a vector of messages.
Afterward, the device is initialized and the geometry is loaded. In particular, the loading geometry process provides a string, which contains the name of the geometry and its id, as an output. In the case of multiple markers to detect, the loading process outputs a vector of strings, containing the names and the IDs of the markers. The vector of strings is associated

with the publisher, so that `Atrapublisher_` publishes a `geometry_msgs::PoseStamped` message-type on topics that have the same names and IDs of the markers to be tracked. This procedure can be seen in the following code:

```
std::vector<string> topics_ = tmp->loadGeometries();

for(auto i=0;i<topics_.size();i++){
Atrapublisher_.push_back(n.advertise<geometry_msgs::PoseStamped>
(topics_[i],1));
}
```

Then the tracking phase is started. In this phase, the attributes of the marker are derived, such as position, orientation, and the ID. Then, the `PoseStamped` geometry message is defined. In particular, the message presents the x, y and z coordinates of the marker, as well as its orientation, expressed by the quaternion. Later, the algorithm checks the ID of the tracked marker and publishes the message in the topic that contains this ID. This is an important step: in case more than one marker needs to be tracked, the algorithm guarantees that the acquired data is published in the right topic.
Moreover, the node stops publishing on the generated topic if the marker is not seen by the camera. When the node is shut down, the algorithm allows to stop the tracking phase and end the connection with the device.

This package is fundamental for the measurement phase in the calibration process. It allows tracking markers that can be fixed on the robot, in order to acquire data for the error minimization between the outputs of the theoretical kinematic model and the real robot positions.

4.2 The KUKA iiwa LBR Model

Since the measurement process aims to track the end-effector poses, and since the Atracsys can track multiple geometries of fiducials, it is needed to design markers to be attached to the robot. Three markers are created: two for the robot base and one for the KUKA's end-effector. The platforms can be seen in Figure 4.3.

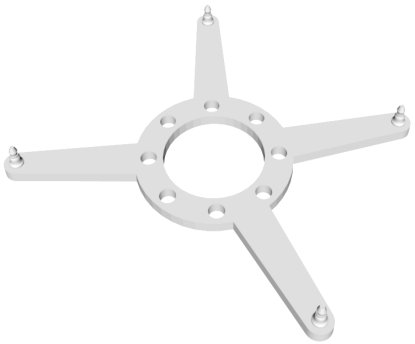
The two base markers are used to track the base frame of the robot, while the third marker is used to track the robot end-effector. The base markers are placed on the base screws as shown in Figure 4.4. The end-effector is positioned on the top of the media electric flange and fixed through M6 screws (Figure 4.5). Reflective spheres are attached to the markers through designed supports available on the platforms.

As stated in Subsection 4.1.2, the geometries of the markers have to be uploaded in the package's `geometry` folder in the `.ini` files form, such that the Atracsys can track them. In particular, the `.ini` files contain the positions and orientations of the markers reference frames to be detected. These frames of reference are computed from the CAD drawings of the markers.

For the measurement phase, only one of the base supports is used, depending on the position of the camera and one particular base support the camera can track. Atracsys can detect the reference frames with respect to its coordinate system, but the pose of the end-effector marker needs to be measured with respect to the coordinate system of one of the base platforms. Therefore, the data of the reference frames acquired by the Atracsys are processed in order to obtain the relative position of the end-effector marker with respect to the base supports, as explained in the following sections. As a result, the acquired robot data represents the position and orientation of the end-effector marker with respect to the base platform. In order to compare the pose of the end-effector support with the outputs of the direct kinematics, the manipulator's theoretical model must contain the two additional reference frames given by the base and the end-effector markers. The reference frames of

(a) *Left base marker*

(b) *Right base marker*



(c) *End-effector marker*

Figure 4.3: The designed markers for the KUKA iiwa LBR

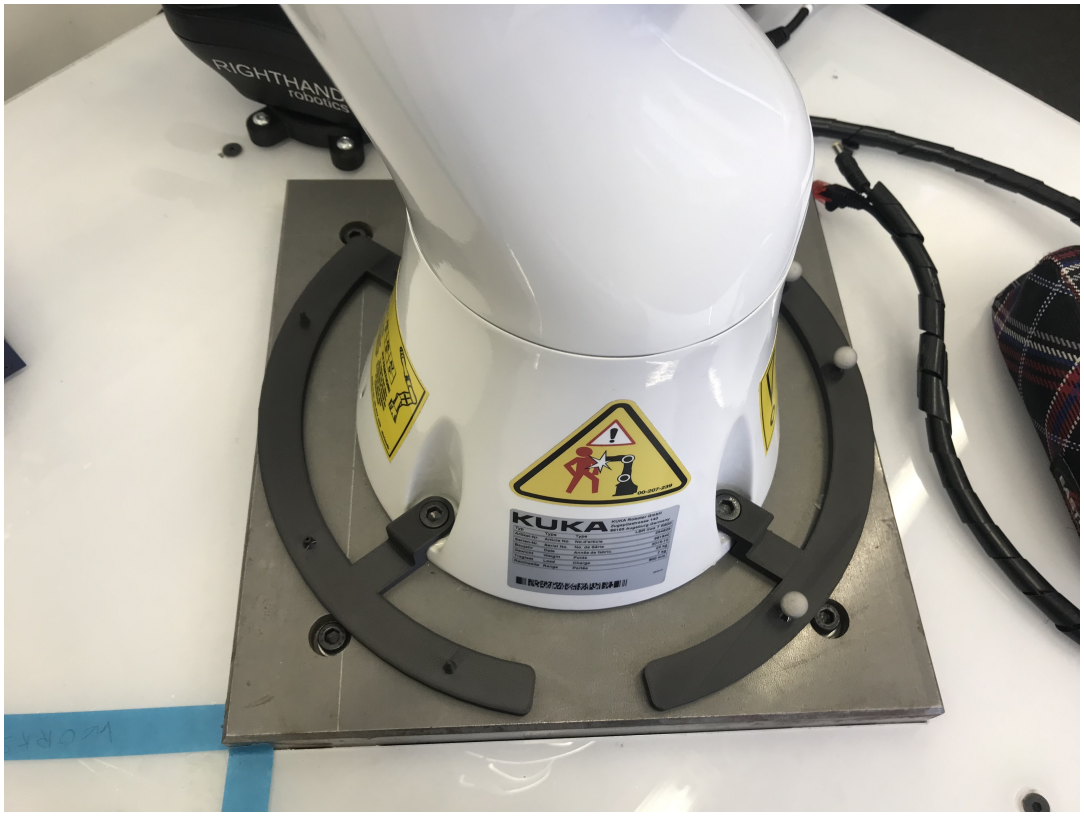


Figure 4.4: Base markers

the base supports are made coincident with the robot base frame, while the origin of the end-effector platform coordinate system is translated of 5mm with respect to the flange reference frame. Therefore, to make the comparison between data and model consistent, the only modification in the nominal model is made on link 7, on which 5mm are added to represent the end-effector support reference frame. The used kinematic model for the KUKA calibration method is represented in Table 4.3.

4.3 `movingKuka` Algorithm

`movingKuka` is a Matlab algorithm that allows the implementation of the open-loop measurement phase. In this program, the robot is randomly moved inside a predefined workspace: then, each end-effector pose is tracked by the camera and saved in an Excel file. Later, these measures are used to update the parameter estimates using the least-

KUKA Nominal Modified DH Parameters				
Link	*theta [deg]*	*d [m]*	*a [m]*	*alpha [deg]*
0	0	0.340	0	0
1	0	0	0	-90
2	0	0.400	0	90
3	0	0	0	90
4	0	0.400	0	-90
5	0	0	0	-90
6	0	0.126+0.035+0.005	0	90

Table 4.3: The Craig parameters for the KUKA kinematic chain

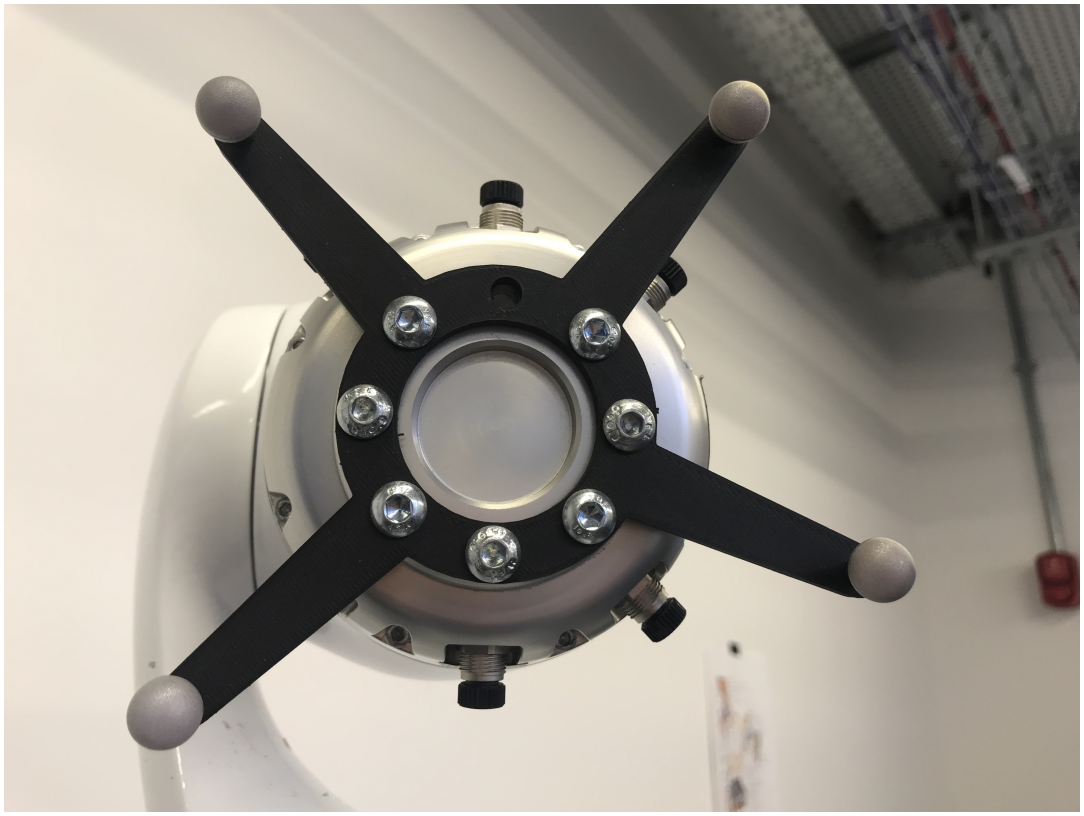


Figure 4.5: End-effector marker

square identification technique described in Subsection 2.2.5.

The algorithm consists of four parts:

- Initialization;
- Definition of end-effector position and orientation;
- Safety features;
- Measurement process;

The first step is to initialize ROS inside the algorithm. As explained in Section 4.1, the algorithm makes use of two ROS packages to complete the measurement process. In particular, the publisher to send the commands to the robot, the subscribers to listen to the base and end-effector supports topics, and the subscriber to read the robot joint angles are set. Then the variable n is specified: it represents the desired number of poses the robot will execute. Afterward, the robot is moved into a chosen starting position, to easily allow the robot reaching the first pose. The joint angles of the starting position are 0 0 0 -90 0 0 0 degrees.

The second step is to generate a random position of the end-effector inside a defined cubic volume. The tool position has to respect the limits of the KUKA working envelope, which is represented in the surface between the two circles in Figure 4.6. Then, the end-effector orientation must be computed. In particular, the orientation is set such that the KUKA always faces the camera. If the camera is moved while the code is running, the algorithm is able to determine the position of the camera and to compute the orientation of the end-effector: the KUKA can dynamically follow the Atracsys. Therefore, the KUKA is always turned towards the tracking system, allowing the Atracsys to have a clear view of the marker attached to the end-effector. To reach this purpose, the base support topic

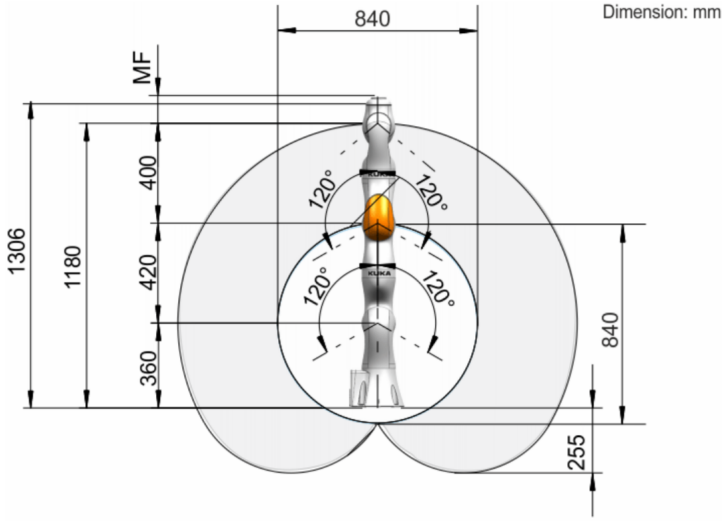


Figure 4.6: KUKA working envelope [6]

must be subscribed to. The platform used in this experiment is the right one, that can be positioned on the KUKA base screws in two ways: either on the side of the robot base, which is the position is designed for, or on the front of the robot. The two positions can be seen in Figure 4.7.
In the nominal position, the reference frame of the base support is coincident with the robot's base one; in the second case, the reference frame is rotated of -90 degrees with respect to the robot base coordinates system. The base topic is subscribed ten times. If the messages published on the base marker topic are empty, i.e the Atracsys cannot track the base marker, the robot moves to the starting position. The same happens if the ten checked messages are equal to each other, hence the marker is not visible.
So the algorithm waits until the base marker is visible, such that it can compute the orientation of the end-effector. If the data carried by the messages are different to each other, they are saved in two matrices: one containing ten positions of the base support with respect to the Atracsys coordinates system, and the other containing ten quaternion vectors. Then, the mean values of the positions and the quaternions are computed. The resulting quaternion is transformed into a rotation matrix through the Matlab function `quat2rotm`, and it is put together with the mean position: the outcome is a transformation matrix $\mathbf{T}^a_{mb}$ that expresses the homogeneous transformation between the Atracsys reference frame and the base platform one.
To compute the end-effector orientation, it is desired to know the position and orientation of the Atracsys with respect to the base marker, whose coordinates frame must be coincident with the robot's reference system. If the marker is positioned on the robot's base side, $\mathbf{T}^a_{mb}$ represents also the transformation matrix from the Atracsys to the robot's base frame, therefore $\mathbf{T}^a_{mb} = \mathbf{T}^a_b$. In case the marker is located on the front of the KUKA, an additional rotation of 90 degrees around the Z axis of the marker base reference frame must be post-multiplied to $\mathbf{T}^a_{mb}$ in order to represent the transformation between the Atracsys and KUKA's base frame, hence $\mathbf{T}^a_b = \mathbf{T}^a_{mb}\mathbf{Rot}(90, Z)$.
To represent the Atracsys with respect to the robot's base frame, the inverse of the transformation matrix from the Atracsys frame to the robot base one must be calculated, as expressed in Equation (4.1):

$$\mathbf{T}^b_a = (\mathbf{T}^a_b)^{-1} = \begin{bmatrix} (\mathbf{R}^a_b)^T & -(\mathbf{R}^a_b)^T\mathbf{t}^a_b \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4.1}$$

(a) *Marker on side or nominal position* (b) *Marker in front or moved position*

Figure 4.7: Positions of the right base support

where $-(\mathbf{R}_b^a)^T\mathbf{t}_b^a$ represents the position of the Atracsys with respect to the base frame. Since the end-effector and the Atracsys positions are known with respect to the robot's base coordinates system, it is possible to compute the distance of the camera with respect the tool in the base reference frame using the following subtraction (Equation 4.2):

$$\mathbf{p}_a^{EF} = \mathbf{t}_a - \mathbf{t}_{EF} \tag{4.2}$$

The unit vector $\mathbf{v}$ is calculated from $\mathbf{p}_a^{EF}$. The representation of $\mathbf{v}$ in the base reference frame is shown in Figure 4.8.
$\mathbf{v}$ is described by two angles: *beta* that represents the angle that the vector creates with the Z axis, and *alpha*, the angle between the vector projection on the XY plane ($\mathbf{v_{xy}}$) and the X axis, in particular (Equation (4.3)):

$$beta = \arccos(\frac{v_z}{\sqrt{v_x^2+v_y^2+v_z^2}}) \quad \text{and} \quad alpha = \arctan(\frac{v_y}{v_x}) \tag{4.3}$$

where v_x, v_y and v_z are the components of the vector respectively along X, Y and Z axes. To position the end-effector towards the camera, the Z axis must be coincident with the

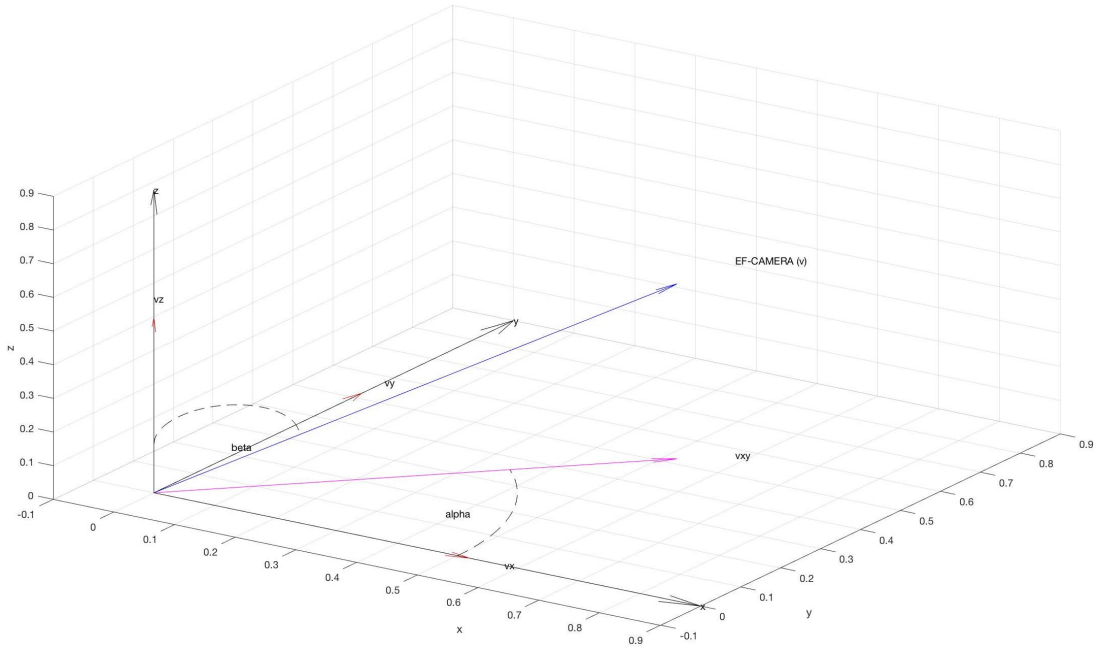
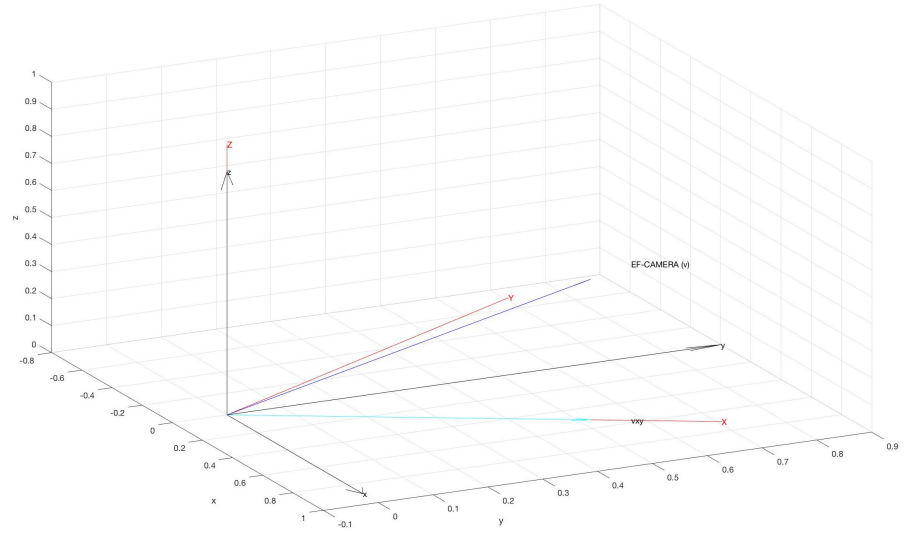
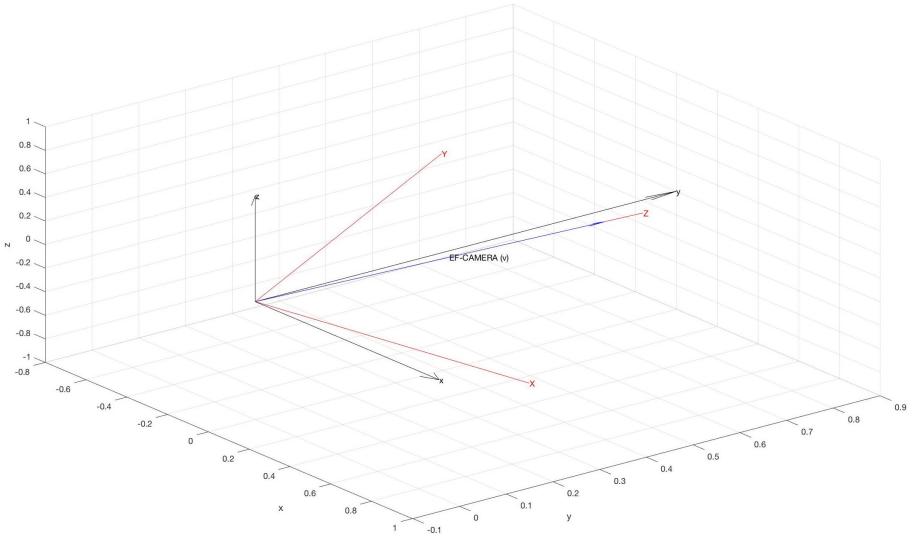


Figure 4.8: Representation of the unit vector $\mathbf{v}$ in the base frame coordinates system



(a) *First rotation*



(b) *Second rotation*

Figure 4.9: Needed rotations to compute the end-effector orientation. In red the rotated reference frame

unit vector $\mathbf{v}$. So first a rotation around the Z axis is computed, such that the X axis lies along the vector projection $\mathbf{v_{xy}}$ (Figure 4.9(a)). Then, a rotation around the Y axis of the rotated reference frame is performed. As a result, the Z axis is positioned along the $\mathbf{v}$ direction (Figure 4.9(b)). Equation (4.4) shows the computed rotation process:

$$\mathbf{R} = \mathbf{Rot}(alpha, Z)\mathbf{Rot}(beta, Y) \tag{4.4}$$

If the randomly generated position of the end-effector is out of the limits established by the KUKA working envelope, a new end-effector pose is generated. Otherwise, the transformation matrix $\mathbf{T}_t$ containing the $\mathbf{R}$ rotation matrix, obtained by Equation (4.4), and the end-effector position can be computed. $\mathbf{T}_t$ is used to calculate the joint angles to locate the end-effector in the desired position and orientation. In order to reach this purpose, the inverse kinematics function `gen_InverseKinematics` offered by Safeea and Pedro Neto in their KUKA Sunrise Toolbox is used. The position and orientation definition process is illustrated by the flowchart presented in Figure 4.10.

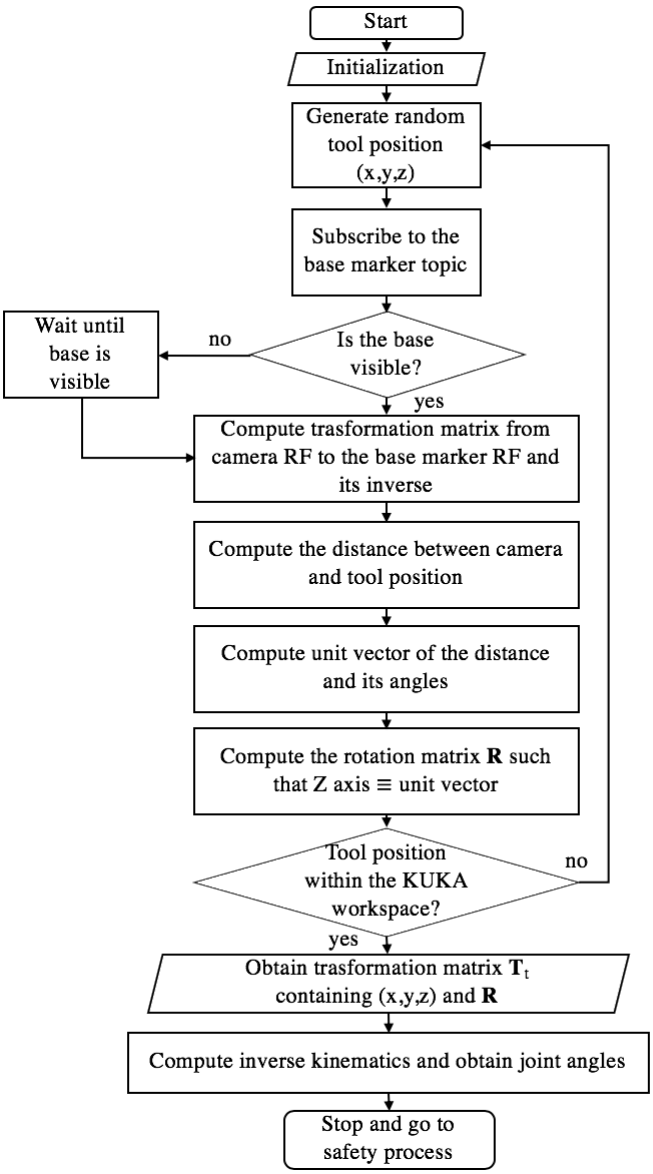


Figure 4.10: Position and orientation definition process. RF stands for reference frame and EF stands for end-effector

The third step aims to add safety features to the algorithm. In particular:

- It checks if the joint angles obtained from the inverse kinematics function are within the robot's joint limits (Table 3.1). If not so, a new random position is generated;
- It checks if the direct kinematics, based on the robot nominal model and using the joint angles obtained from the inverse kinematics, coincides with the randomly generated position. If the difference between the translation vector, obtained from the direct kinematics, and the random position vector is greater than a given error tolerance of 1mm, a new position is randomly generated;
- It computes the direct kinematics for links 3 and 4 and checks their z positions, such that the robot does not hit the table it is fixed to while moving. If their heights are less than 100mm, a new random position is generated.

If these conditions are fulfilled, the joint angles resulted from the inverse kinematics are used to move the robot. The flowchart representing this third step is shown in Figure 4.11.

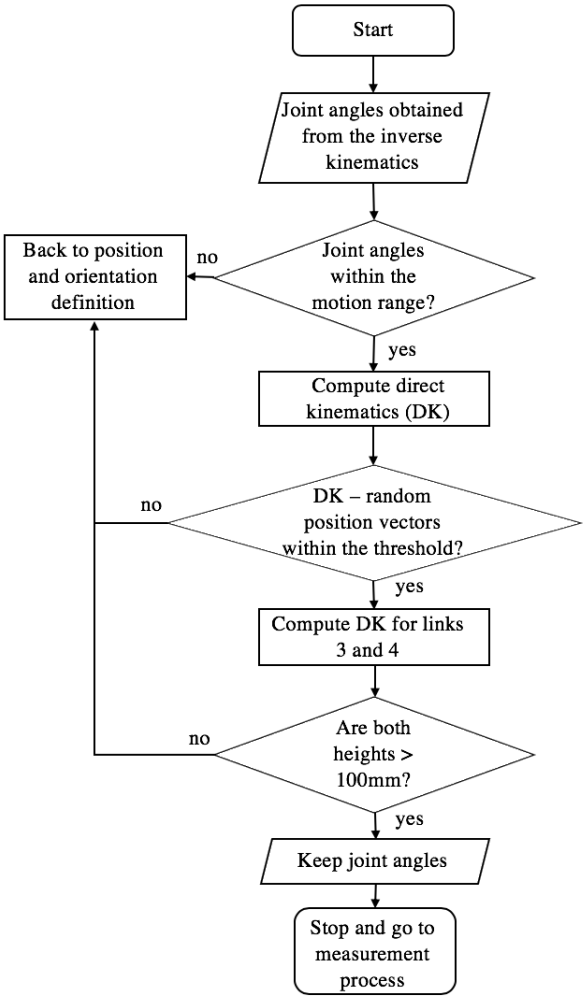


Figure 4.11: Safety process

The fourth and final step allows the robot to move to the desired joint angles and to record the end-effector pose. First, the joint angles are sent to the KUKA through the command "setPosition". Then the algorithm waits five seconds to allow the robot to move to the desired position. After this break, a condition is inserted to check whether the robot has reached the desired joint angles or not. To do it, a subscriber to read the robot's joint angles is used. If the joints are not in the desired position, the algorithm waits five seconds.
Later, the subscriber to the end-effector marker is employed. As said in Section 4.1.2, the Atracsys node stops publishing on the markers topics if the platforms are not visible. Therefore the algorithm checks the last fifteen messages published on the end-effector marker topic. If the last messages are empty so that the marker is not visible, or the data are the same, hence the last fifteen messages refer to a previous recording, a new pose is generated.
In case the end-effector marker is visible, the data are processed in the same way are processed the base marker data: hence, the transformation matrix from the Atracsys reference frame to the end-effector marker one is obtained. This matrix is named $\mathbf{T}^a_{EF}$. Then the position and orientation of the end-effector marker frame with respect to the

base platform coordinates system are computed (Equation (4.5)):

$$\mathbf{T}_{EF}^{b} = \mathbf{T}_{a}^{b}\mathbf{T}_{EF}^{a} \tag{4.5}$$

The transformation matrix $\mathbf{T}_{EF}^{b}$ is saved inside an Excel file. Finally, the robot joint angles are again read by the subscriber and saved inside a Text file. Later, these angles are used for the identification algorithm. The measurement process flowchart is shown in Figure 4.12.

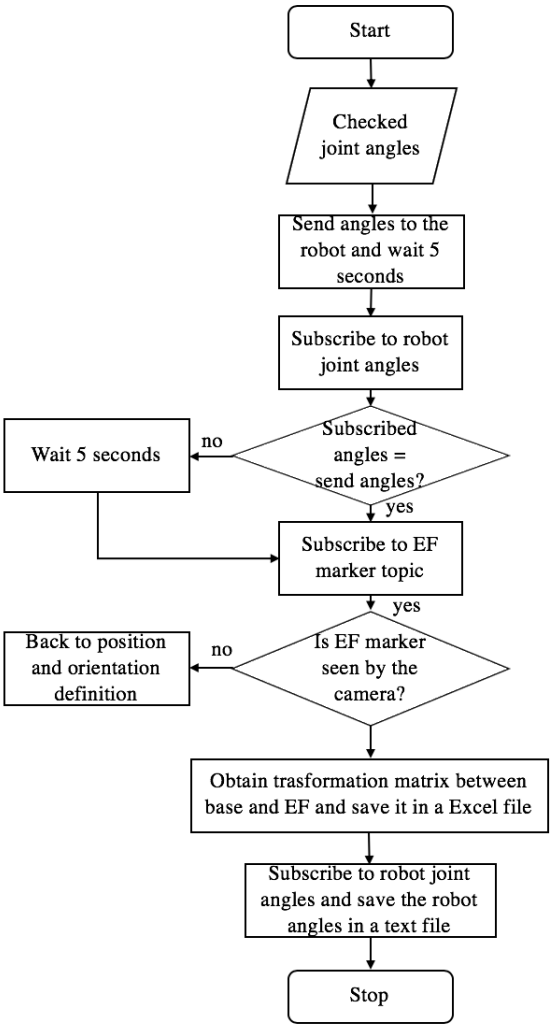


Figure 4.12: Measurement process

The last three steps of the algorithm are repeated for n poses. In the end, the program will give as outputs n Excel files, each containing the transformation matrix of the recorded pose, and a Text file, that includes a matrix of n rows and 7 columns, where 7 represents the joints of the robot.

4.4 `robotCalibration` Algorithm

`robotCalibration` is an algorithm developed in Matlab used to minimize the error between the real pose of the end-effector and the theoretical one, computed from the nominal kinematic model. As a result, better estimates of the robot parameters are obtained. This algorithm is based on the identification process described in Subsection 2.2.5.

First, the symbolic parameters that will be used in the algorithm are initialized. Then, a matrix containing all the nominal parameters of the robot, as shown in Table 4.3, is defined. This matrix is reshaped into a (28×1) vector called **DHpar**. After, the vector of the parameters that have to be calibrated is set: it contains all the offset lengths, link lengths and twist angles of the robot. The joint angles are not included because the joint sensors' readings are considered precise. In the same way, a symbolic matrix containing all the symbolic parameters of the robot is defined and reshaped into a vector. Also, in this case, a vector of the symbolic parameters that will be iterated is set. Afterward, the direct kinematics is performed based on the symbolic matrix and the symbolic end-effector pose vector is derived. To perform the calibration, the data obtained from the measurement process must be included in the algorithm. Therefore the joint angles are imported and saved into a $(n \times 7)$ matrix, where n represents the number of executed poses. The same is done to the transformation matrices whose poses are saved in a matrix named $\mathbf{X}_{mes}$. After defining a threshold of $1 \cdot 10^{-6}$ and a $\boldsymbol{\Delta err}$ greater than the threshold, the calibration algorithm can start. As long as $\boldsymbol{\Delta err}$ is greater than the threshold the following steps have to be performed:

- For the number of executed poses n, the first seven elements of the **DHpar** vector must be equal to the n-row of the matrix containing the imported joint angles. Then the symbolic values of the DH parameters are substitued with the values of **DHpar** in the end-effector symbolic vector. The numeric end-effector is saved into a matrix called $\mathbf{X}_{nom}$. Moreover, the Jacobian matrix is computed and saved in a matrix referred to as ***phi***.

- The error between the matrix containing the measured poses and the matrix containing the theoretical poses is computed: $\boldsymbol{\Delta X} = \mathbf{X}_{mes} - \mathbf{X}_{nom}$

- The $\boldsymbol{\Delta err}$ is calculated through the line of code:

```
dCP =  pinv(phi)*deltaX
```

  where `dCP` is $\boldsymbol{\Delta err}$ and `pinv` is the Matlab function for calculating the pseudo-inverse of a matrix.

- Then the new DH parameters are updated through the lines:

```
prevCP = DHpar(a);
CP=prevCP+dCP;
DHpar(a) = CP;
```

  where `DHpar(a)` with `a = [8:28]` represents the vector of parameters that needed to be iterated.

The parameters updating is repeated until all the elements of `dCP` are lower than the threshold.

In the next chapter, three tests are computed and three different sets of parameters are obtained. The first test is performed with the base marker in the nominal position and makes use of an high number of poses. The second and third tests are executed using the base marker on the front of the KUKA: the first two tests make use of an high numbers of poses, while the second employs a low number of poses.

Chapter 5

Experiments and Results

5.1 Experiments

The calibration method illustrated in Chapter 4 is applied to the KUKA iiwa LBR to improve its positioning accuracy. Three tests are performed to verify the validity of this process.

The first test consists of locating the right base marker on its nominal position, i.e. on the right side of the robot's base, and moving the end-effector of 100 poses inside a cubic volume defined as the following lines of code:

```
%% Random x y z values
% Random x value [mm]
a_1 = -100; %lower x limit
b_1 = 800;  %upper x limit
x = (b_1-a_1).*rand(1,1) + a_1;
% Random y value [mm]
a_2 = -400; %lower y limit
b_2 = 800;  %upper y limit
y = (b_2-a_2).*rand(1,1) + a_2;
% Random z value [mm]
a_3 = 200;  %lower z limit
b_3 = 800;  %upper z limit
z = (b_3-a_3).*rand(1,1) + a_3;
```

The second test is performed positioning the right base marker on the front of the robot and allows to move the manipulator of 100 poses inside the same workspace of the first test.

The last and third test is executed using the base marker in front of the robot and aims to move the robot's tool of 50 poses inside a smaller volume, defined as $x = [20, 550]$, $y = [-260, 300]$ and $z = [500, 800]$, all expressed in millimeters.

The sets of poses executed by the robot in these three tests are shown in Figure 5.1: the points in yellow are the completed positions in the first test, the ones in blue represent the poses executed in the second test and the ones in red describe the poses performed in the third test. Moreover, as stated in Section 4.3, in these tests the KUKA is always directed towards the Atracsys camera, such that the tracking system has a clear view of the marker attached to the end-effector, as shown in Figure 5.2.

The reason the base marker is positioned in two different ways is that it is hard to find an Atracsys' location in which the camera can easily track the base marker in the nominal setting. As a consequence, the marker positioned on the front of the robot can slightly move, due to manufacturing reasons: the base marker was designed to be fixed on the side of the KUKA's base.

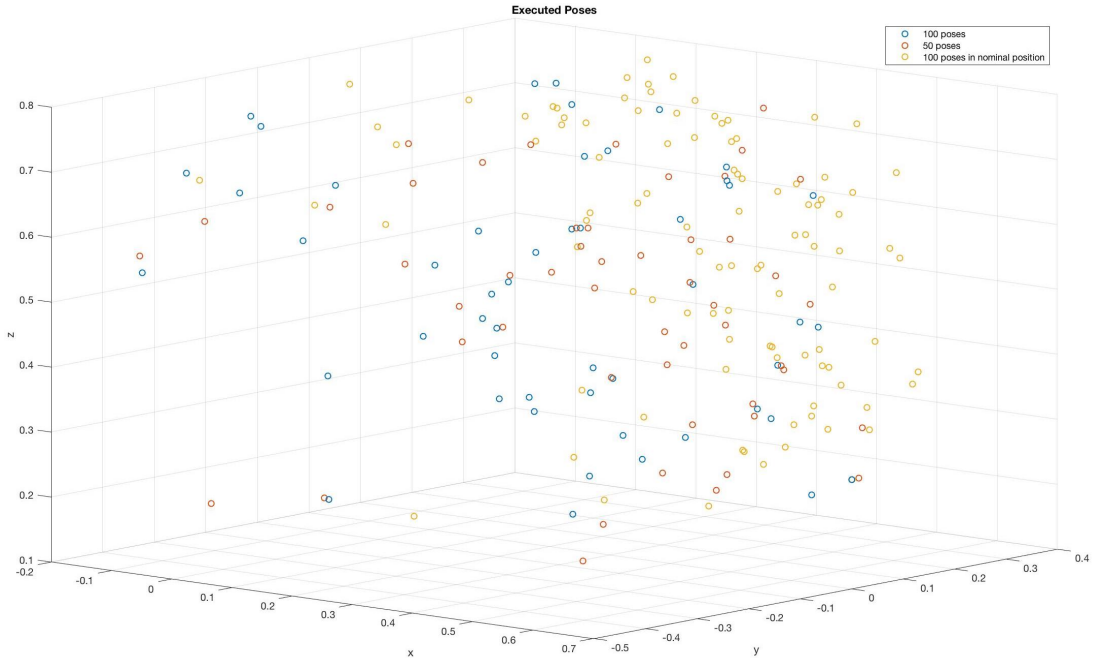


Figure 5.1: Executed poses

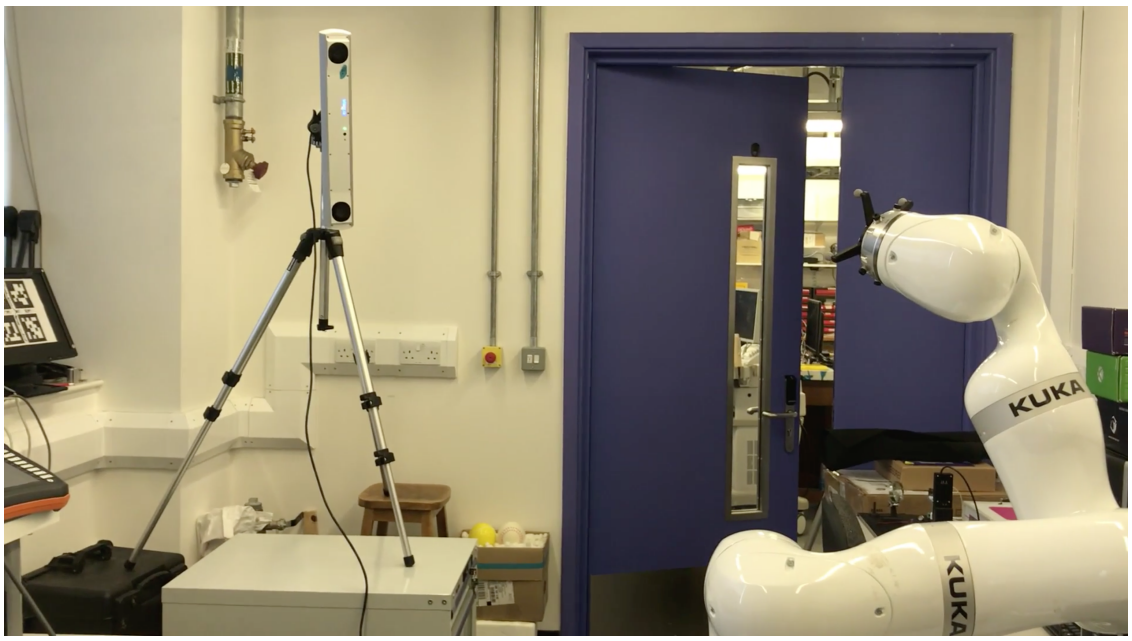


Figure 5.2: KUKA's end-effector towards the camera

5.2 Results

The sets of data that are acquired from the three experiments are used to find the real DH parameters of the robot.

For the first test, the obtained DH parameters are shown in Table 5.1. The nominal poses obtained from the updated parameters (in red) and the measured poses tracked by the Atracsys (in blue) trends are shown in Appendix C in Figure C.1. To verify the accuracy of the coefficients, the root-mean-square errors between the poses measured by the camera and the outputs of the updated model are computed. The average value of

these errors is 0.0018 m. The trends of the root-mean-square errors and the average value are shown in Figure D.1 in Appendix D.

First test: updated DH parameters				
Link	*theta [rad]*	*d [m]*	*a [m]*	*alpha [rad]*
1	0	0.3374	0.0221	0.0009
2	0	0.0007	-0.0003	-1.5698
3	0	0.4005	0.0009	1.5708
4	0	0.0030	0.0066	1.5670
5	0	0.3967	0.0024	-1.5779
6	0	0.0005	0.0031	-1.5658
7	0	0.1227	-0.0034	1.5707

Table 5.1: Updated parameters for the the first test

The updated DH parameters of the second test are shown in Table 5.2. The trends of the poses with updated DH parameters and measured poses are presented in Appendix C in Figure C.2, while the trends of the root-mean-square errors between measured poses and nominal ones are illustrated in Appendix D in Figure D.2. The average value of these errors is 0.0023 m.

Second test: updated DH parameters				
Link	*theta [rad]*	*d [m]*	*a [m]*	*alpha [rad]*
1	0	0.3360	0.0196	0.0061
2	0	0.0005	-0.0009	-1.5653
3	0	0.4018	0.0035	1.5665
4	0	0.0041	0.0059	1.5812
5	0	0.3977	0.0050	-1.5808
6	0	-0.0033	0.0038	-1.5965
7	0	0.1269	-0.0019	1.5713

Table 5.2: Updated parameters for the second test

Finally, the updated DH parameters for the last test are shown in Table 5.3. The trends of the measured and nominal poses are shown in Figure C.3 in Appendix C. The average error is 0.0032 m and the trends of the root-mean-square errors are shown in Figure D.3 included in Appendix D.

Third test: updated DH parameters				
Link	*theta [rad]*	*d [m]*	*a [m]*	*alpha [rad]*
1	0	0.3257	-0.0123	0.0217
2	0	0.0006	0.0014	-1.5690
3	0	0.4008	-0.0004	1.5732
4	0	0.0324	0.0030	1.5913
5	0	0.4037	-0.0053	-1.6346
6	0	0.0042	0.0045	-1.5385
7	0	0.1502	-0.0059	1.5701

Table 5.3: Updated parameters for the third test

A comparison between the first and second tests should be considered. The DH parameters of the second experiment are slightly different from the coefficients obtained in the first one, although the selected workspaces, as well as the executed poses, are the same.

This is due to the marker base position, as stated in Section 5.1. The base marker is designed to be placed on the side of the robot, in such a way that the base is still when the robot is operated. When the base is positioned on the front, it can slightly move while the robot is running. Therefore, errors occur during the measurement phase and the DH parameters computation is consequently affected. Moreover, also the noise presented on the measured data, joint compliance and link deflections, that depend on the position of the robot, can lead to different parameter values.

A difference between the second test coefficients and the third ones can be observed. The number of poses that the robot executes is different in the two experiments. The result is that the average error of the second test is lower than the one in the third test. Therefore, the larger is the number of poses executed by the robot, the higher is the accuracy of the kinematic algorithm: the error between the real pose and the nominal one using calibrated DH parameters is reduced.

A final remark can be made regarding the volume used to generate random positions. The higher is the space investigated by the robot, the higher is the accuracy. This depends not only on the definition of the workspace boundaries, in which the random positions are generated, but also on the tracking volume of the Atracsys: the farther the position of the camera, the higher is its tracking volume. Increasing the robot workspace improves the precision of the kinematic calibration. The best way to reach accuracy is to make the robot move in a big workspace with a large amount of poses.

The described kinematic method is developed for any tool attached to the robot's flange: the tool must have at least three markers on the top and its height has to be included in the last link. Anytime the tool is changed, the kinematic process should be repeated to keep a high level of accuracy.

Chapter 6

Conclusion

6.1 Conclusions

In this work, a strong calibration procedure is presented and discussed, in order to increase the positioning accuracy of a seven-revolute joints manipulator KUKA iiwa LBR 7R 800. First, a theoretical background needed to have a better understanding of the kinematic calibration is described. The general mathematical model used to represent a manipulator is reported, along with the Denavit-Hartemberg conventions and parameters used in this thesis. Several methods for a level 2 calibration are investigated and discussed. On the basis of this study, an open-loop measurement method is adopted to obtain the end-effector pose data that, thereafter, are elaborated with an ordinary least-square technique. In order to obtain an higher accuracy, ROS packages are developed and used to manage the KUKA robot and the Atracsys tracking system on the ROS framework. In specific, the `movingKuka` algorithm defines a node that coordinates the arm manipulator and the camera in the measurement phase, while the `robotCalibration` program provides the identication process to obtain a realistic representative model of the KUKA. Both algorithms are developed in Matlab. The acquired results are considered positive, therefore the proposed calibration method is valid.

6.2 Recommendations

In this section, some recommendations are described to perform the measurement process:

1. It is better to position the base marker on its nominal position to reduce the movement of the marker while the KUKA is running. This ensures more accurate computation of the DH parameters.

2. The experiment should be performed in an environment with minimum light intensity and reflection. Light and its reflection can affect the tracking markers phase, making it difficult to measure the position and orientation of the markers.

3. The KUKA can run autonomously, and therefore run overnight without oversight. This allow a very long measurement phase to run, hence higher number of poses can be saved to make more precise the calibration method. However, it is recommended to always check the system for safety reasons.

Appendix A

First Appendix: How to Install ROS

To install ROS on Ubuntu 16.04, it is needed to navigate on the ROS website (`http://www.ros.org/`) and install the desired ROS distribution, that is, for this thesis purpouse, ROS Kinetic.

A.1 Getting Started with the Installation

ROS is going to be installed on Ubuntu from the ROS package repository. In order to use ROS package repository, the Ubuntu one's options have to be configured. The first step is to go to the "Software & Updates" folder and enable all the Ubuntu repositories. Then, ROS packages have to be allowed from the ROS repository server called `packages.ros.org` through the following command:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb\_release-sc main" > /etc/apt/sources.list.d/ros-latest.list'
```

When a new repository is added to Ubuntu, the keys should be added to validate the origin of the packages, so the next command has to be typed:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
recv-key 0xB01FA116
```

After these steps ROS is ready to be installed.

A.2 Installing ROS

The ROS packages should be installed on Ubuntu. First of all, the list of packages on Ubuntu has to be updated. In order to do this, the following command is written in the terminal:

```
$ sudo apt-get update
```

After updating the list, the entire ROS package is installed using the following command:

```
$ sudo apt-get install ros-kinetic-desktop-full
```

This line allows installing ROS Kinetic on Ubuntu. To conclude the installation `rosdep` tool has to be initialized: it allows to install dependencies of packages that will be compiled. These commands must be run on the terminal:

```
$ sudo rosdep init
$ rosdep update
```

A.3 Getting rosinstall

The source trees for particular ROS packages have to be installed through a ROS command-line tool, called `rosinstall`. The tool is based on Python and can be installed through the command line:

```
$ sudo apt-get install python-rosinstall
```

Since ROS is completely installed, it is needed to check if the installation was successful. Open a terminal and type the `roscore` command:

```
$ roscore
```

Then open another terminal and run the `turtlesim` command:

```
$ rosrun turtlesim turtlesim_node
```

If the installation is proper, the turtle is displayed, as shown in Figure A.1.

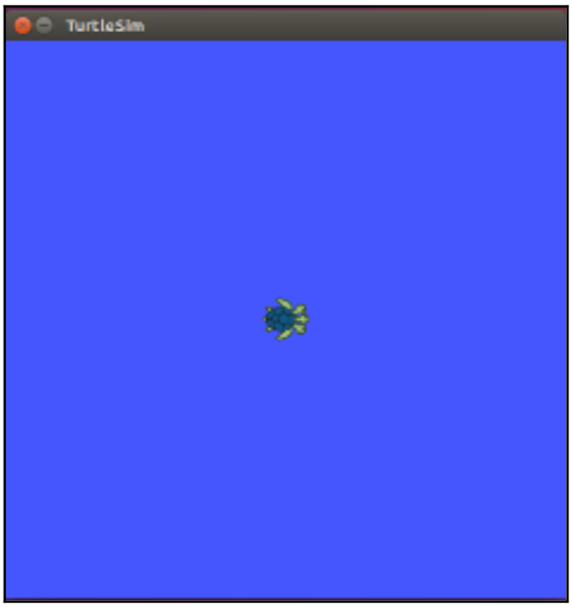


Figure A.1: The turtlesim mode [9]

A.4 Setting ROS Workspace

The last step is to create a ROS workspace, which is a place where ROS packages are kept. To build the ROS workspace the next instructions must be followed.

- Open a terminal and type:

  ```
  $ mkdir -p ~/catkin_ws/src
  ```

  This command allows to create an empty workspace folder named `catkin_ws`, and another one called `src`, that will contain the ROS packages.

- Open a terminal in the `src` folder, or go inside the folder through this line:

  ```
  $ cd ~/catkin_ws/src
  ```

- Initialize the catkin workspace in the `src` folder by:

```
$ catkin_init_workspace
```

- After the initialization, the packages can be built inside the workspace using the following commands:

```
$ cd  ~/catkin_ws/
$ catkin_make
```

  The workspace can be built even without any packages.

- Three folders can be found inside `catkin_ws`: `build`, `devel` and `src`.

A.5 Creating a ROS Package

The cakin packages are created in order to allow the communication between nodes. This step is crucial for enabling the conversation between the Atracsys, the KUKA and the algorithm that processes and saves the data. The next steps explain how to generate a catkin package.

- Go to the `src` folder inside the `catkin_ws` one by writing:

```
$ cd  ~/catkin_ws/
```

- The command `catkin_create_pkg` is used to make a new package:

```
$ catkin_create_pkg <package_name> std_msgs rospy roscpp
```

- Therefore the package folder with the name you given is generated inside `src`. It contains two files: `package.xml` and `CMakeLists.txt`

- The new package has to be built inside the ROS workspace through the commands:

```
$ cd  ~/catkin_ws/
$ catkin_make
```

- To add the workspace to the ROS environment, the generated setup file must be sourced:

```
$ source devel/setup.bash
```

When a package is built inside a workspace, it is better to use the command `catkin build` instead of `catkin_make`. In order to migrate from `catkin_make` to `catkin build`, the next steps must be followed:

- Go inside `catkin_ws` folder and open a terminal. Then write:

```
$ rm -rf build/ devel/ install/ src/CMakeLists.txt
$ cd src catkin_init_workspace
```

- Then the `catkin build` command can be used:

```
$ catkin build
```

- Remember to source the package every time a change is done:

```
$ source devel/setup.bash
```

Appendix B

Second Appendix: Installation of the Atracsys fusionTrack 500

The fusionTrack uses a Gigabit Ethernet 1000BASE-T (IEEE 802.3ab) interface to communicate with the computer. The fusionTrack factory setting uses 172.17.1.7 as the static IP address. The fusionTrack must be properly powered and connected to a PC [8].

The Ethernet adapter of the host PC must be configured as indicated in Table B.1.

Address assignment	**Static**
IP address	172.17.1.100
Subnet mask	255.255.255.0
Jumbo frame	Enabled (at least 8500 bytes for maximal performance)

Table B.1: Default network settings [8]

Appendix C

Third Appendix: Poses Trends

C.1 First Test

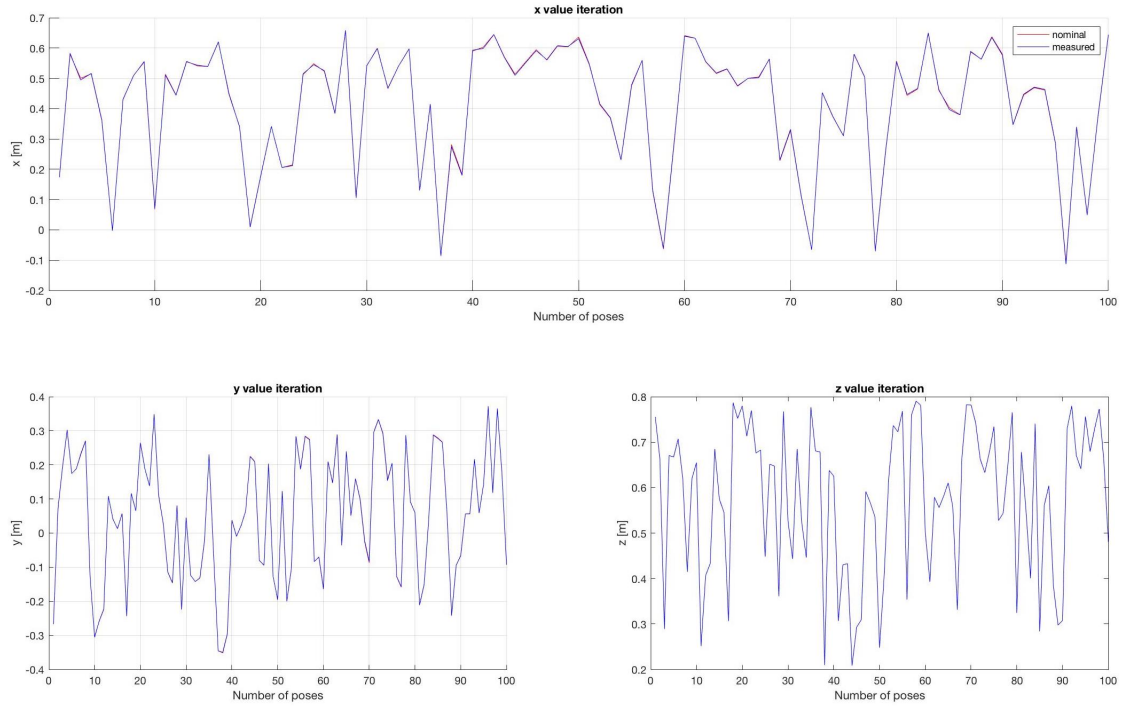


Figure C.1: Trends of nominal and measured poses in the first test

C.2 Second Test

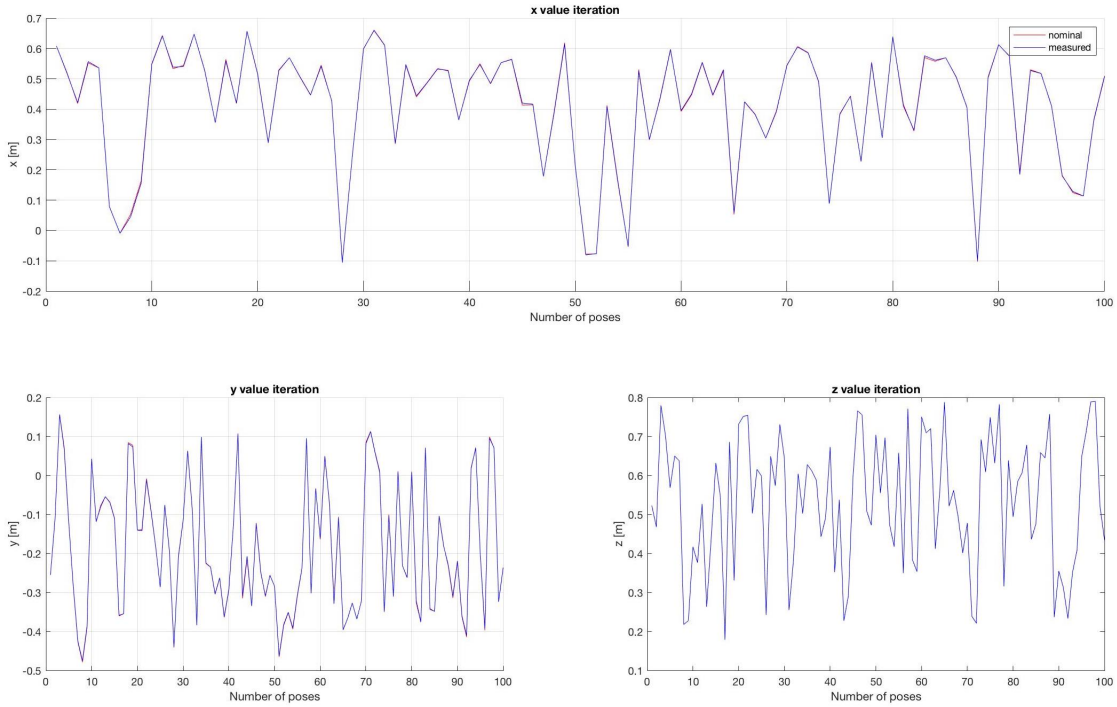


Figure C.2: Trends of nominal and measured poses in the second test

C.3 Third Test

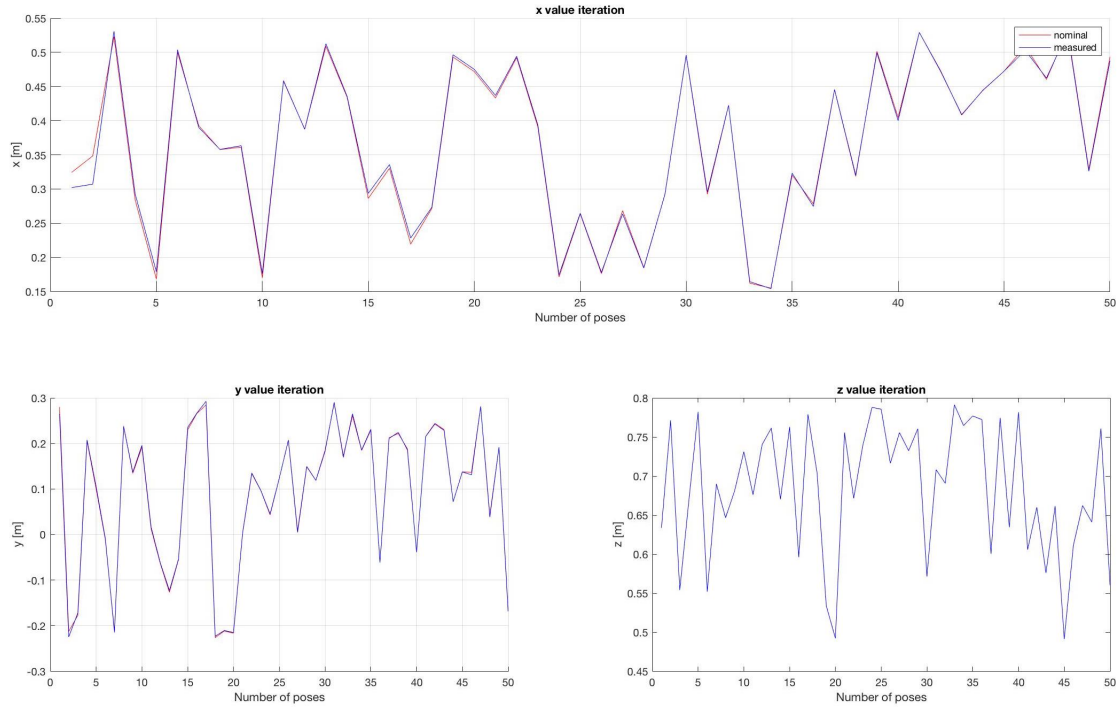


Figure C.3: Trends of nominal and measured poses in the third test

Appendix D

Fourth Appendix: Errors Trends

D.1 First Test

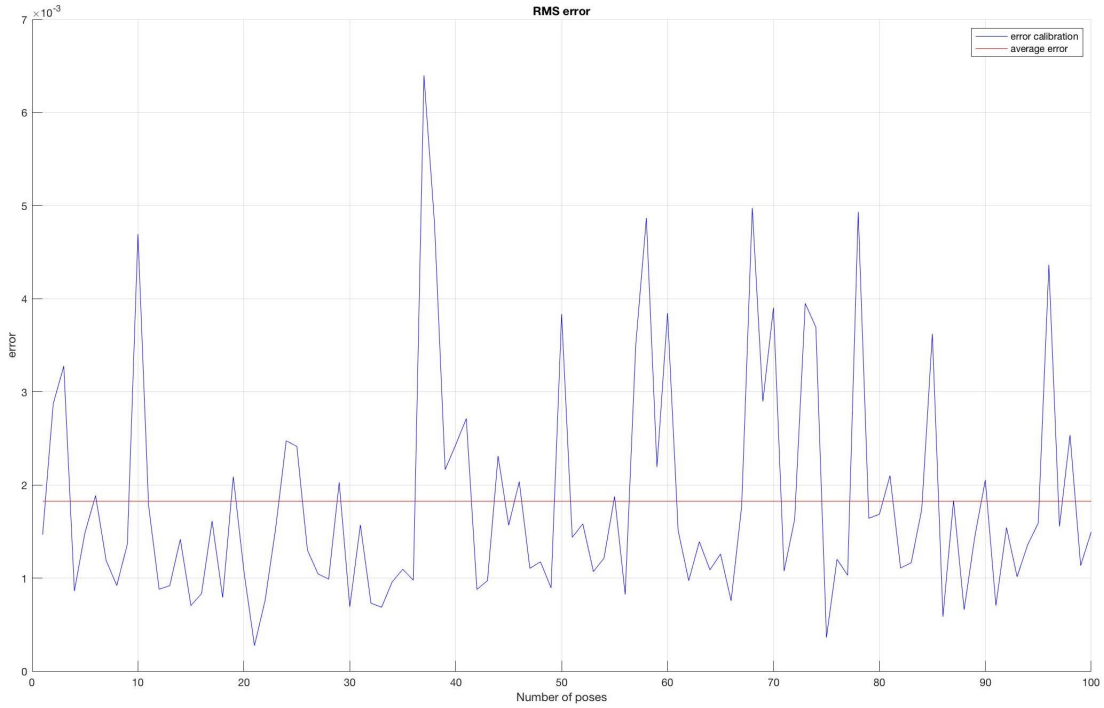


Figure D.1: First test root-mean-square and average errors

D.2 Second Test

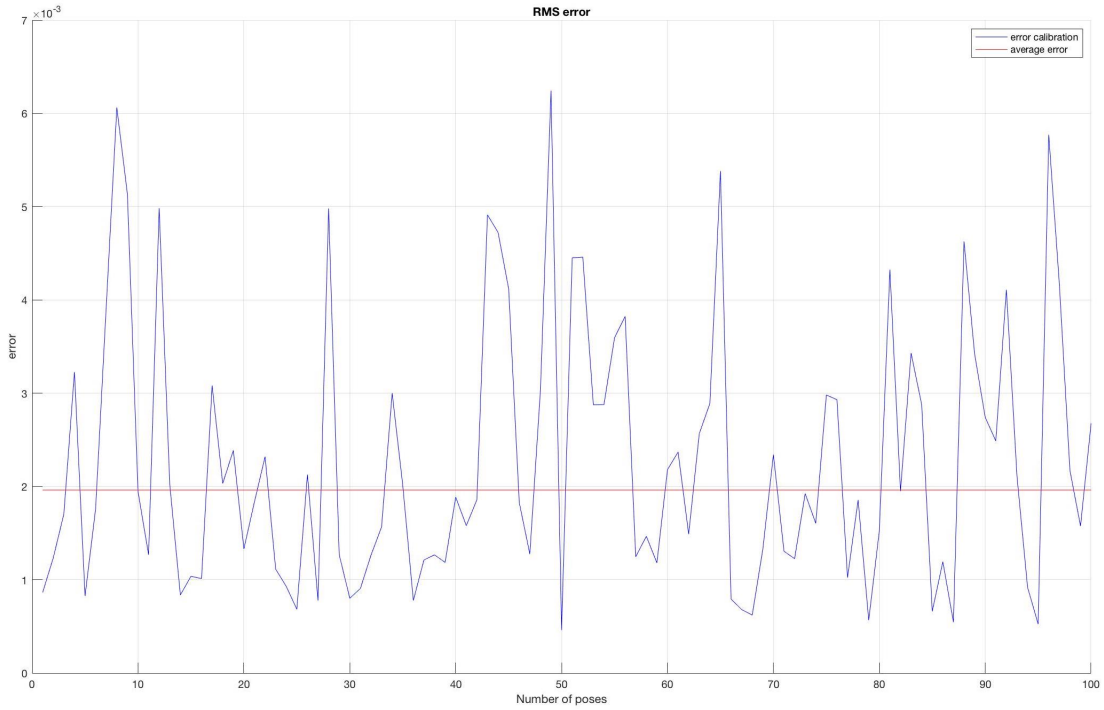


Figure D.2: Second test root-mean-square and average errors

D.3 Third Test

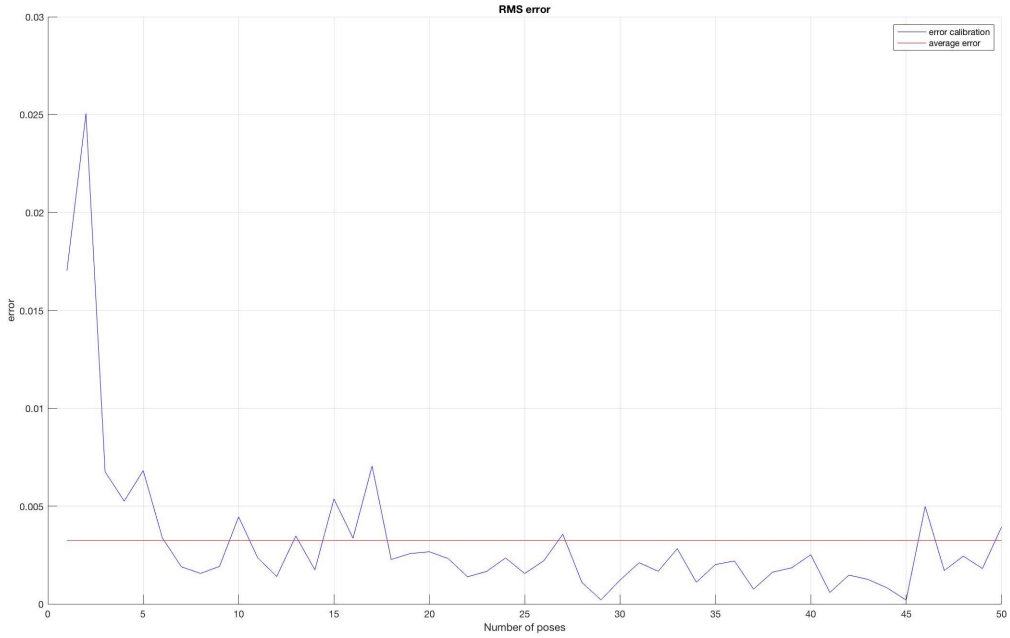


Figure D.3: Third test root-mean-square and average errors

Bibliography

[1] B. Bona, "Robotics," published in PoliTO Robotics course, 3 2017. [Online]. Available: http://www.ladispe.polito.it/corsi/meccatronica/01PEEQW/2016-17/slides.html

[2] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2010. [Online]. Available: https://books.google.co.uk/books?id=jPCAFmE-logC

[3] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] M. R. Driels and W. E. Swayze, "Automated partial pose measurement system for manipulator calibration experiments," *Robotics and Automation, IEEE Transactions on*, vol. 10, pp. 430 – 440, 09 1994.

[5] B. W. Mooring, Z. Roth, and M. R. Driels, "Fundamentals of manipulator calibration," pp. V–VIII, 4–5, 01 1991.

[6] K. R. GmbH, *LBR iiwa*, 7th ed., KUKA Laboratories GmbH, Zugspitzstraße 140 D-86165 Augsburg Germany, 5 2016.

[7] ——, *Media Flange*, 7th ed., KUKA Laboratories GmbH, Zugspitzstraße 140 D-86165 Augsburg Germany, 2 2016.

[8] A. LLC, *fusionTrack 500 User Manual*, 4th ed., Atracsys LLC, Route du Verney 20 1070 Puidoux Switzerland, 4 2019.

[9] L. Joseph, *ROS Robotics Project*, 1st ed. Livery Place 35 Livery Street Birmingham B3 2PB, UK: Packt Publishing Ltd, 3 2017.

[10] J. L. Saeid Mokaram, Jonathan M. Aitken, *ROS-integrated API for the KUKA LBR iiwa collaborative robot*, 1st ed., University of Sheffield, Western Bank, Sheffield S10 2TN, 3 2017.

[11] D. Deblaise and P. Maurine, "Effective geometrical calibration of a delta parallel robot used in neurosurgery," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug 2005, pp. 1313–1318.

[12] K. Nancy, "Kinematic calibration of a serial robotic arm using a linear movement constraint," Master's thesis, Carleton University, 8 2016.

[13] A. Chennakesava Reddy, "Difference between denavit - hartenberg (d-h) classical and modified conventions for forward kinematics of robots with case study," 12 2014.

[14] J. Denavit and R. Hartenberg, "A kinematic notation for lower-pair mechanisms," *ASME Journal of Applied Mechanics*, vol. 22, pp. 215–221, 01 1955.

[15] H. Lipkin, "A note on denavit-hartenberg notation in robotics," 01 2005.

[16] J. H. Jang, S. H. Kim, and Y. K. Kwak, "Calibration of geometric and non-geometric errors of an industrial robot," *Robotica*, vol. 19, no. 3, p. 311–321, 2001.

[17] Kuka iiwa lbr official website. [Online]. Available: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

[18] Kuka iiwa lbr 7 r800 robotworx. [Online]. Available: https://www.robots.com/robots/lbr-iiwa-7-r800

[19] Robot operating system (ros) official website. [Online]. Available: http://wiki.ros.org/ROS/Introduction

Acknowledgements

Al termine di questo importante percorso, volevo ringraziare tutti coloro che mi hanno supportato (e sopportato) in questi anni di studio.

Un generoso ringraziamento al mio relatore Prof. Marcello Chiaberge e al mio supervisor Dott. Riccardo Secoli, che mi hanno seguito in questi mesi, offrendomi le nozioni necessarie per portare a termine il progetto. Li ringrazio soprattutto per la loro disponibilità e pazienza che hanno dimostrato nei miei confronti.

Ringrazio i ricercatori del MiM Lab, che hanno reso piacevoli le giornate lavorative, che mi hanno sostenuto durante i momenti più difficili e hanno contribuito a dare un valore aggiunto al lavoro svolto. Un grazie in particolare ad Alex, Sukhi, Arnau, Stephen, Sam, Marlene, Eloise, He, Xue, Tom, Silvia, Abdul, Ferdinando, Vani, Hisham e Fabio.

Grazie ai miei compagni di corso Fabio, Andrea, Filippo e Davide, che oltre ad essere colleghi e compagni di studio, sono ottimi amici. Mi hanno regalato momenti di leggerezza e risate, facendo scorrrere veloce le giornate di lezione e di sessione.
Grazie anche a Giacomo, Mickael, Zakaria, Laurent, Alexis, Joshua, Enni, Luca e Johana, che hanno fatto lo stesso durante i mesi di Erasmus in Francia e mi hanno accompagnato nella mia prima esperienza all'estero.

Un grazie alle amiche che hanno vissuto con me a Londra, Faduma, Chiara e Keyla. Ringrazio soprattutto Faduma per le giornate post-lavorative e i weekend, il sostegno morale che mi ha dato e per aver coltivato questa splendida amicizia con me.

Grazie ai miei migliori amici, che per quanto lontani in questo ultimo anno, hanno sempre fatto in modo di essere nella mia vita, venendomi a trovare, chiamandomi e facendosi sentire. Grazie ai miei amici "simbiotici" Alma e Nick, che si fanno sempre in quattro per starmi accanto. Grazie di cuore a Isabella che ha sempre un po' di spazio per me. Grazie a Marta per avermi regalato questa bellissima amicizia. Grazie ad Elena, Mattia e Irene, i miei compagni e vicini di casa di una vita. Grazie a Stefano che mi ha dato momenti di allegria.

Un caloroso grazie alla mia famiglia e soprattutto ai miei genitori, che oltre ad avermi dato il sostegno economico per completare i miei studi, si sono sempre sacrificati perché io potessi ricevere sempre il meglio. Senza di voi nulla poteva accadere. Siete e sarete sempre il pilone portante della mia vita, e saró sempre grata per tutto quello che mi avete dato. Siete tutto.

E infine, grazie a Fabio, che mi ha sempre spronato a dare il meglio di me stessa. Che mi é stato accanto a Torino, a Limoges e a Londra. Che mi ha spinta oltre ai miei limiti. Grazie per avermi reso una persona migliore.

Kinematic Calibration of a Seven Revolute Joints Serial Manipulator

Coordinator:
Prof. Marcello Chiaberge

Student:
Jennifer Chimento