POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

High accuracy Pose Estimation with Computer Vision

Supervisor Politecnico di Torino
-Prof. Alessandro RIZZO

Candidate
-Tomás Marcelo BOCCO

Supervisor FAPS
-M.Sc. Oguz KEDILIOGLU
-Prof. Dr.-Ing. Jörg FRANKE

April 2021

In partnership with:

Institute for Factory Automation and Production Systems (FAPS)
from Friedrich-Alexander University Erlangen-Nuremberg (FAU).

Abstract

This thesis aims to develop a Computer Vision system that measures the position and orientation of a sample attached to an anthropomorphic robotic manipulator with high accuracy. This system is intended to act as the primary sensor in an external control loop to improve the accuracy in the actual sample positioning system for the neutron diffractometer STREESS-SPEC at the Heinz Maier-Leibnitz center in Garching, Germany.

The experiments carried out in this facility require an error in position to be less than 50 μm in any direction, while for orientation, it cannot be higher than 0.5° in each axis. The actual positioning system consists of a 6-axis industrial manipulator with a repeatability of 50 μm, but an absolute accuracy of approximately 0.5 mm. It is assumed that the primary source of error can be explained by a deficient description of the robot's kinematics in its control system, which leads to an inaccurate pose estimation of the end-effector.

Due to this fact, it is necessary to rely on an external measuring system that can perceive deviations both in position and orientation. This information will serve as feedback to compensate the pose of the robot. This thesis is part of the RAPtOr project carried out by FAPS (Institute for Factory Automation and Production Systems) and aims to provide a solution for this problem.

Two different methods were tested for this research. The first one consists of computing the pose of a square fiducial marker based on the information provided by a set of two cameras in Stereo Vision configuration. The author proposed this fiducial and consists of an enhanced version of ArUco markers that allows subpixel algorithms for corner detection.

In order to detect corners more accurately, a Super-Resolution Deep Neural Network (SRDNN) was tested. This network had as input a corner image of 8x8 px, and returned as output an enhanced image of 64x64 px. The output training data consisted of pictures of the corners from a calibration board at different positions, with dimensions 64x64 px. They were resized to 8x8 px, and noise was added to generate the input training data. Various tests determined that the improvement due to this technique was not significant.

The second one is also based on Stereo Vision, but in this case, the fiducials are Concentric Contrasting Circular (CCC) markers. These are not internally codified, which makes more challenging the correlation process between images. Despite this, its small size is advantageous since it is more practical to attach them to complex samples under test.

For the latter approach, it was necessary to develop an algorithm that could scan a sample and generate a 3D map with the relative position between markers. As a consequence that all CCC fiducial are alike and do not have an internal codification, they need to be individually identified by their relative position with respect to each other.

A laser tracker with 10 μm accuracy was used to compare the Computer Vision algorithms' performance. Three reflectors were positioned on the manipulator's end-effector for the laser tracker to measure its position. The relative location between this instrument and the cameras had to be estimated through an optimization algorithm that aligns both measurements. Since the markers were located at a different position from the reflectors, it was also necessary to compute the relative pose between both.

The results of the measurements concluded that both methods are similar in accuracy. The cameras available for testing at FAPS resulted to be insufficient for the task at hand. A new pair of 20 Mpx cameras with 16.4 mm diagonal will be used for the target applications. In combination with 50 mm lenses, these cameras are expected to reach the required accuracy for pose estimation.

Acknowledgements

First of all, I want to express my gratitude towards my supervisors from FAPS, Oguz Kedilioglu and Prof. Dr.-Ing. Jörg Franke that give me the opportunity to carry out my thesis with them. From the beginning, it was clear the commitment that was put on this research. They have helped me to overcome the barriers that have constantly appeared.

I also want to thanks my supervisor from Politecnico di Torino, Profesor Alessandro Rizzo, who accepted being my tutor. He was also my professor and awakened my curiosity towards Robotics while laying the theoretical foundations necessary to accomplish this research.

My sincere thanks also go to Politecnico di Torino and Universidad Nacional de Córdoba that allowed me to finish my career in Italy through a Double Degree program.

Last but not least, I want to thank my parents for their unconditional support and love. Through the different stages of my life, they have provided me the skills that I need to be here today. Despite the distance, they constantly encourage me to achieve my goals and dreams.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The purpose of RAPtOr project, carried out by FAPS institute, is to find an alternative for the actual sample positioning system at the STRESS-SPEC neutron diffractometer. This instrument is part of the Heinz Maier-Leibnitz Center (HML) and is used for a wide range of applications in the field of material science [1].

It emits a beam of neutrons, part of it is reflected by the sample under test and then captured by an area detector. The properties of the reflected ray, such as its direction and intensity, depend on the internal structure of the sample material. Texture and non-destructive residual stress analyses can be carried out with the output information of this experiment. A scheme of the diffractometer is shown in Figure 1.1.

With this aim, it is necessary to position the sample with high accuracy at the focus point of the neutron source [2]. It also must be located at different orientations, in such a way that the gauge volume remains in the path of the neutron beam [3]. In Figure 1.2 it is shown a sample being hit by a neutron beam, the inner gauge is schematized as a dashed circle inside the sample.

1.2 Related Work

The research carried out by Randau in 2015 [4] described the requirements of the positioning system. The neutron diffractometer STRESS-SPEC can define a gauge volume as small as 0.5 x 0.5 x 0.5 mm^3. The accuracy of the positioning system should

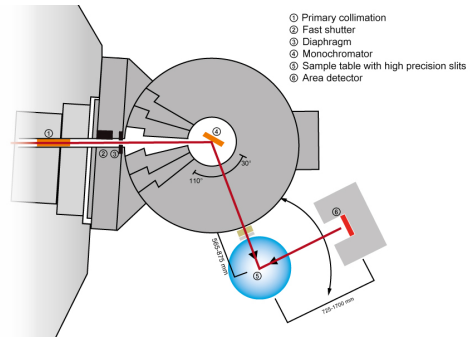


Figure 1.1: Diffractometer [1]

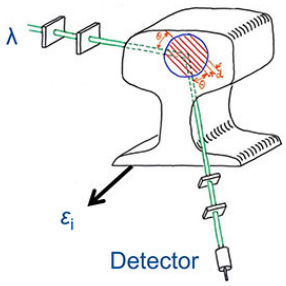


Figure 1.2: Inner Gauge (Image from Winfried Petry)

be ten times smaller than the gauge volume, resulting in a required accuracy of 50 μm.

This paper also explains that the positioning system used to be an X-Y-Z table in combination with Eulerian cradles, shown in Figure 1.3. Despite being accurate, this system limits the sample space and makes automation difficult. As a consequence, a positioning system based on the use of an industrial robot was developed (Figure 1.4).

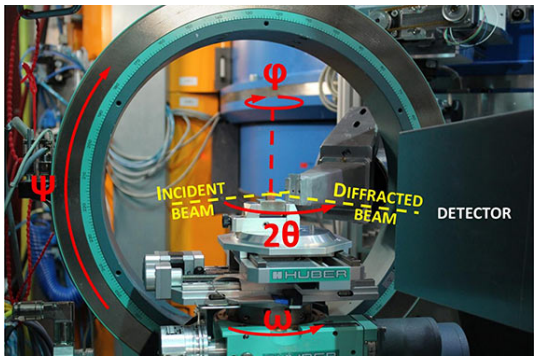


Figure 1.3: Eulerian cradle and X-Y-Z table (Image from M. Landesberger presentation)

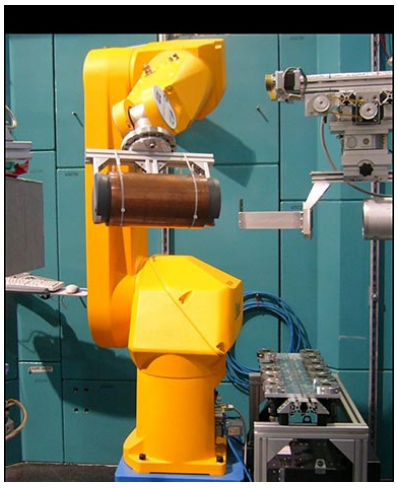


Figure 1.4: Robot positioning system at STRESS-SPEC [5]

Regardless of its good repeatability ($\pm 0.05\,mm$), its absolute accuracy was lower than desired. The study of Randau has shown that the relative accuracy on the relevant working region of the robot is $\pm 0.5\,mm$, ten times lower than required.

Due to this fact, they used an external measurement system to accurately determine the absolute position of the sample, in order to utilize that information to compensate for the pose error. This method reduced the absolute positioning error to $\pm 35\,\mu m$.

They used a Leica laser tracker to measure the absolute pose of the robot. At the end-effector, six reflectors were attached symmetrically forming a hexagon, as depicted in Figure 1.6 . In order to obtain the 6DOF pose, at least three of them had to be measured.

Despite its accuracy, the measuring system cannot be used for every orientation or sample, since the three reflectors needed may not be reachable by the laser tracker.

The aim of this thesis is to provide a solution to replace this measuring system with a more practical one.

Figure 1.5: Leica laser tracker

Figure 1.6: Robot with 6 reflector attached to its end-effector [4]

1.3 Thesis Focus

1.3.1 Control system architecture

As seen in Section 1.1 the existing robot motion control system (RMCS) is not accurate enough for placing a sample with the required specifications. One of the hypotheses of this thesis is that the main source of error is due to the inverse kinematic block on the RMCS.

This block computes the joints parameters that are necessary to achieve a certain position and orientation of the manipulator. Several modifications on this block were made, but there were no significant improvements according to Randau [4].

Nubiola [6] in 2013, achieved a significant reduction of the absolute error of an industrial robot through kinematic calibration. He used a 29-parameter model to calibrate an ABB IRB 1600 with a repeatability of $\pm 0.05\,mm$, the same as the target robot in this research. However, this was achieved with 1000 data points from a laser tracker, which

this thesis tries to replace. Furthermore, the mean error resulted to be 292 μm, which is much greater than the required specifications.

Taking this into consideration, an external control loop was proposed to improve the accuracy of the system, Figure 1.7. Because it is external, there is no need to modify the RMCS already existing. The desired pose x_d will cease being the input signal of the RMCS and would be replaced by x_n that is the output of the external controller.

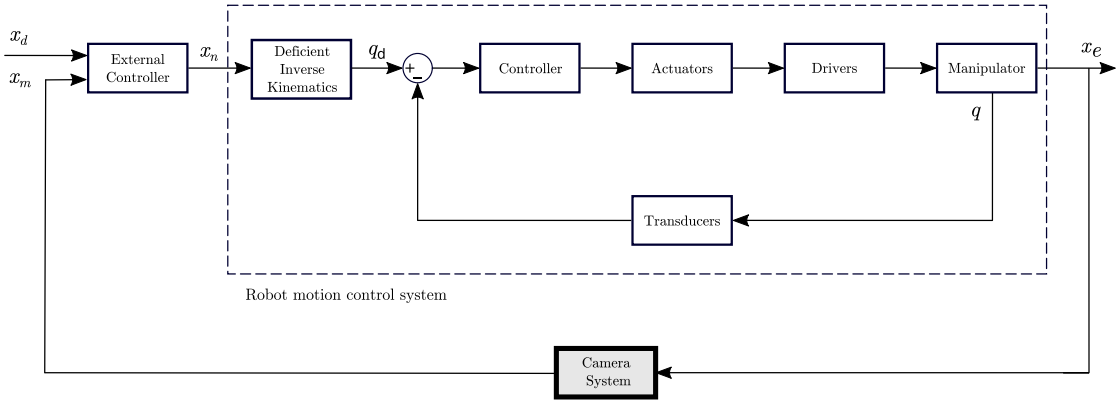


Figure 1.7: Control architecture: External feedback that compensates errors provoked by the RMCS

This controller will have x_d as input, and x_m, which is the pose measured by the external measuring system (EMS). By nature, the external control system is discrete, since before sending an instruction, it has to wait for the robot to reach the previously commanded pose.

The architecture of the controller is straightforward, and it can be seen in Figure 1.8. On each iteration, x_n will be computed as the sum between the actual pose x_d and the sum over time of the difference between x_d and x_m. The sum block that can be seen in Figure 1.8 acts as a discrete integrator. The output x_n could be interpreted as a modified desired pose that overweights the Inverse Kinematic Transformation's errors.

The architecture of this controller was tested in a simulated two planar arm manipulator, with a deficient inverse kinematics block. The position error was reduced from 70 mm to less than 50 μm in four iterations. The complete analysis of it can be seen in Appendix A.

External controllers based on Neural Network architecture were also considered. For example Chi-Tho [7] developed a Neural Network external controller in which no iteration was needed. However, its resulting mean error was around 2 mm.

The author is aware that the architecture of the controller can be enhanced to work faster. However, it is not the main objective of this thesis. It is preferable a greater accuracy rather than a short settling time. The focus is on developing an accurate

external measuring system.

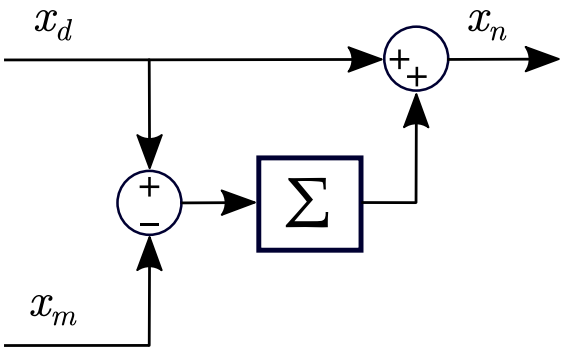


Figure 1.8: External controller

1.3.2 External measuring system

The external measuring system must have an absolute accuracy error under $50\mu m$. As described in section 1.1 the Leica laser tracker, despite reaching the accuracy goal, was not suitable for being inconvenient.

For this reason, at HML it was decided to develop a solution based on Computer Vision principles. A set of cameras would take images of the samples at STRESS-SPEC, and after processing them, the pose of it should be computed. Finding a method that achieves this with the required specifications is the main objective of this thesis.

Two different methods are proposed in the chapters Chapter 3 and Chapter 4 respectively:

1. **End-effector pose estimation with ArUco markers:** For this technique, ArUco markers are used as fiducial to determine the pose of the end-effector. With this information, the pose of the sample can be directly computed with a simple homogeneous transformation. A single ArUco marker is sufficient for computing the pose of an object in space, either with mono or stereovision. Figure 1.9.
2. **Object pose estimation with CCC markers:** This technique intends to measure the pose of a sample directly. The markers are placed on the object and not in the end-effector as the method mentioned before. At least three markers must be visible to compute its pose with stereo vision, but the number of markers increases to six with monovision. Figure 1.10.

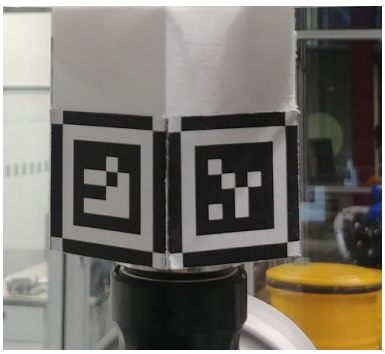


Figure 1.9: ArUco Markers

Figure 1.10: Blade with dot markers

Chapter 2

Stereo Vision: Theory and general technique

2.1 General technique

An image is a two-dimensional projection of a three-dimensional space, it inherently causes a loss in depth information. However, if two images from different perspectives are available, and the cameras' intrinsic parameters and the relative position between each other is known a priori, it is possible to triangulate the position of a common point to get its three spatial coordinates. This process is known as Stereo Vision.

Pose estimation with Stereo Vision consists mainly of 4 steps. They are represented schematically in Figure 2.1:

1. **3D Reconstruction and Calibration:** It consists of estimating the relative position between the two cameras and their relationship with the environment. It can be achieved with a calibration method, as would be seen in Section 2.2. The intrinsic parameters of the cameras can also be estimated with calibration, they would be required to correct camera lenses distortions and for the step of triangulation. (Figure 2.1 a).
2. **Feature detection and correlation:** Is necessary to identify points in an object that are visible in both pictures. This process is not straightforward since a point in an image could not necessarily be precisely identified in the other one. Some techniques try to find relevant features on objects automatically, such as SIFT [8]. However, for this thesis, it was decided to rely on fiducial markers, more in Section 2.3.1. (Figure 2.1 b).

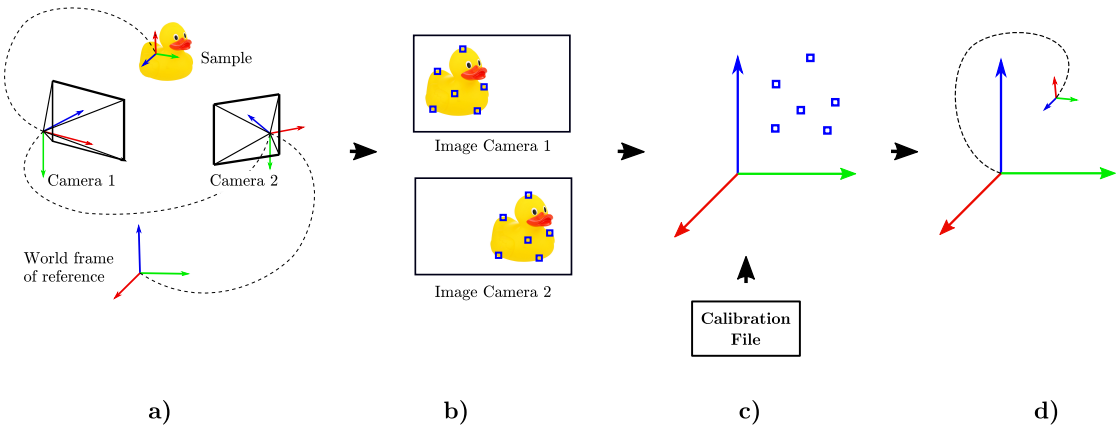


Figure 2.1: Stereo vision overview

3. **Triangulation:** With the information of the intrinsic and extrinsic parameters of the cameras and the location of the feature points in both images, their position in space can be derived. (Figure 2.1 c).
4. **Pose estimation:** To define the pose of a rigid body it is necessary to determine six parameters, as would be seen in Section 2.5. A single point is not enough. We need to know at least three points of the solid in space to compute its position and orientation with respect to the world reference frame. This can be achieved with the equation of Section 2.5.2. (Figure 2.1 d).

These four steps will be described in detail in the following subsections.

2.2 3D Reconstruction and Calibration

2.2.1 Intrinsic parameters

Pinhole camera model

The pinhole camera model consists simply of a linear transformation between a 3D space onto a 2D plane. The coordinates (X_w, Y_w, Z_w), are transformed into (u, v) through Equation 2.1 from Siciliano's Book [9]. Figure 2.2 illustrates this concept.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{\Omega \, \Pi \, T_w^c} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.1}$$

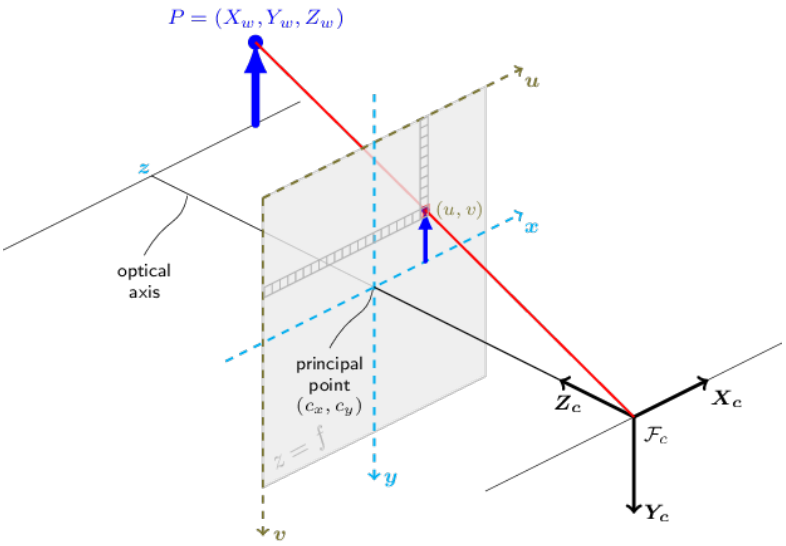


Figure 2.2: Pinhole camera model (Image from [10])

With:

$$\mathbf{\Omega} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{\Pi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

$$\boldsymbol{T}_w^c = [R_w^c \mid t_w^c]$$

As it can be seen, both coordinates are represented in homogeneous form. The parameter s is an arbitrary scale factor, while Π is an auxiliary matrix used to match matrix dimensions. The subindex w in $[X_w, Y_w, Z_w]'$, indicates that the coordinates are referred to the world reference frame. They must be first pre-multiplied by T_w^c, that is a homogeneous transformation matrix that relates the world reference frame with the camera reference frame, as shown in Equation 2.3:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = T_w^c \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.3}$$

If the word reference frame coincides with the camera reference frame, the only parameters that must be estimated for calibration are f_x, f_y, c_x, c_x , which are part of the camera matrix $\mathbf{\Omega}$.

Lenses distortion

Camera lenses distort the image causing that Equation 2.1 turns to be inadequate. There are two principal distortion effects, tangent and radial distortion, according to the documentation of OpenCV [11]. To correct the first one, we use the following equations:

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.4}$$

with:

$$r^2 = x^2 + y^2 \tag{2.5}$$

Because of this, 5 different coefficients must be estimated:

$$Distortion\ Coefficients = (k_1, k_2, p_1, p_2, k_3) \tag{2.6}$$

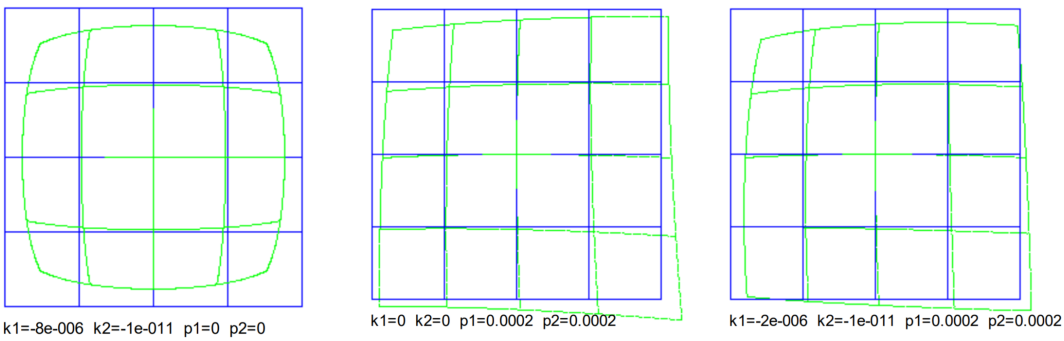


Figure 2.3: Effects of tangential and radial distortion [12]

2.2.2 Extrinsic parameters

The so-called extrinsic parameters of a stereo vision system, are simply the relative position of the cameras to each other and to the world reference system. These relationships can be expressed with only two homogeneous transformation matrices: $T_{c_2}^{c_1}$ and $T_{c_2}^{W}$.

These two matrices can be obtained through a stereo calibration process as would be described in Section 2.2.3.

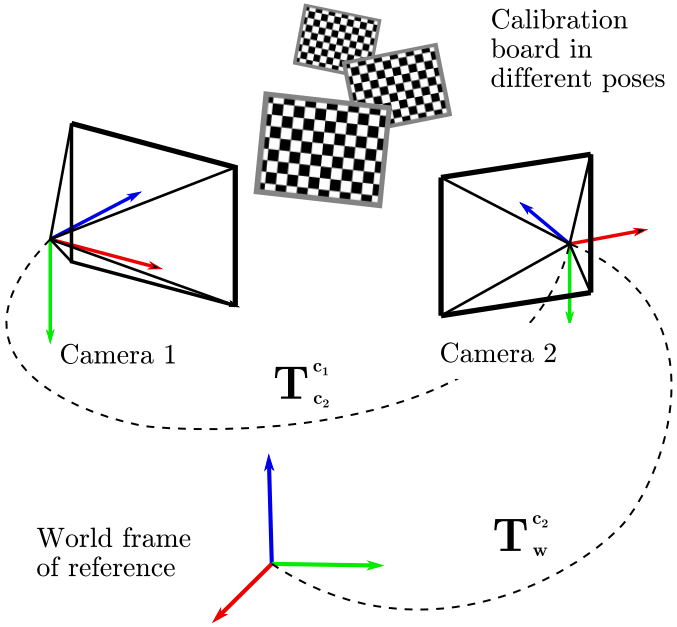


Figure 2.4: Reconstruction

2.2.3 Stereo Calibration

As seen in the last two subsections, the reconstruction of the intrinsic and extrinsic parameters requires estimating 6 different elements. They are:

- Two camera matrices.
- Two distortion coefficients vectors.
- The homogeneous transformation matrix between camera one and camera two.
- The homogeneous transformation matrix between camera two with the world reference frame.

The last item can be computed afterward if we know the first three, by measuring the position of a reference object with respect to the cameras.

Each camera matrix has 4 degrees of freedom, the transformation matrix has 6 and the distortion model is defined with 5 parameters. In total, the problem in question has 24 degrees of freedom.

All these unknowns can be computed with OpenCV `stereoCalibration` method at the same time. The calibration process consists of detecting the corners of a calibration board from both cameras' images, and then an optimization algorithm tries to find the optimal intrinsic and extrinsic parameters that produce the smallest reconstruction error.

```
ret, M1_n, d1_n, M2_n, d2_n, R, T, E, F
= cv2.stereoCalibrate( objpoints, imgpoints_l,imgpoints_r, M1, d1, M2,
                      d2, dims, criteria=crit,flags=flags)
```

Listing 2.1: Stereo calibration method in OpenCV

The corner detection is refined through a subpixel algorithm that estimates the position of the desired points on the image with higher accuracy. Apart from the images, the calibration function needs as input the geometry of the pattern.

An alternative approach consists of calibrating each camera's intrinsic parameters first, and then its extrinsic parameters. This increases the number of optimization problems to solve. Nonetheless, each of them would have a smaller amount of unknown parameters.

For the specific task at hand, a chessboard pattern was mounted on a tripod, with the two cameras focusing on it. The board was moved to different positions and the camera took images of it. The extrinsic and intrinsic parameters were computed at the same time.

In general, the intrinsic parameters of cameras do not change much over time, on the other hand, the extrinsic parameters do if there is a slight variation on the relative pose of a camera with respect to the other. These changes in position could be caused simply by changes in temperature. Because the error should be reduced at its minimum expression, it is necessary to calibrate the extrinsic parameters frequently.

The chessboard pattern is very useful for the computation of the intrinsic values since its covers almost the whole image. However, because of its dimensions, automating the calibration with such a board becomes impractical.

An automatic extrinsic calibration algorithm was developed to correct the extrinsic parameters if necessary. It detects the corners of ArUco markers stuck to the robot. In spite of its relatively small size compared with the calibration board, the markers'

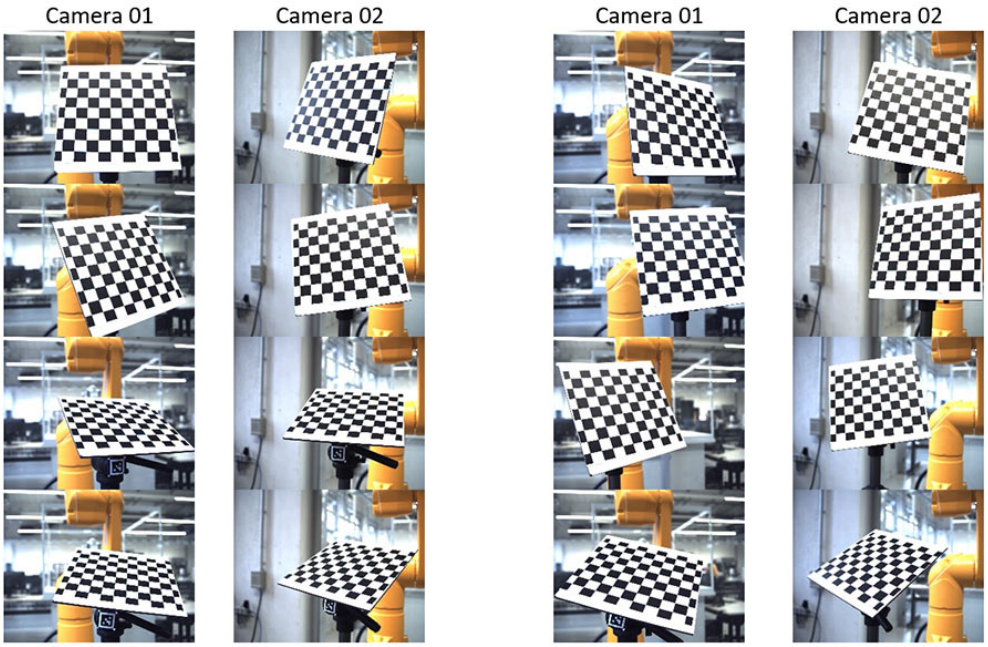


Figure 2.5: Example images for stereo calibration

position can be moved through the entire coverage area of the cameras. With images from both cameras at the same instants, the extrinsic parameters can be derived without the use of a chessboard.

2.3 Feature detection and correlation

2.3.1 Fiducial Markers

Fiducial markers are patterns that can be easily detected by a computer vision system. They are used in a wide variety of fields, just to mention a few: augmented reality, industrial systems, robot navigation, etc. For this research, it was decided to use fiducial markers for the reasons mentioned below:

- **Detection:** Markers are designed in general with the objective of being easy to detect. Well known image processing techniques can be used to locate and isolate these markers.
- **Resources already available:** The existence of libraries already available in almost any programming language, that can detect fiducial markers, is a great advantage to start working with.

- **Avoid automatic feature detection:** detecting features on images is in many cases a complicated problem and does not always lead to a precise result. It is because of this, that it was decided to define a priori the points to be detected.
- **Precision required:** Several researchers in the last year attempted to estimate the pose of an object using machine learning algorithms [13] [14] [15] [16] [17]. Neither of the papers mentioned satisfied the requirements needed for this case of study. The metric used to validate their models consist in the percentage of measurements with a mean error less than 10% of the object's diameter. In this thesis, it is intended to measure objects as large as 200 mm with 50 μm precision, which represents 0.025% of the object diameter.

2.3.2 Correlation

In a stereo vision system, it is not sufficient to localize fiducial markers on the images, it is necessary also to match corresponding detected points from both images.

If the markers are internally codified the correlation process is straightforward. When a fiducial is detected in an image the Id of it can be derived, and points with matching Ids from both pictures will be correlated. This is the case of the first approach on this thesis using ArUco markers, Figure 1.9.

If the markers do not present a codification it is still possible to correlate them. This is the case CCC markers, Figure 1.10, which is the second approach in this thesis. In the next subsection 2.3.3, it will be described how the epipolar geometry set correlation constrains based on the camera's poses.

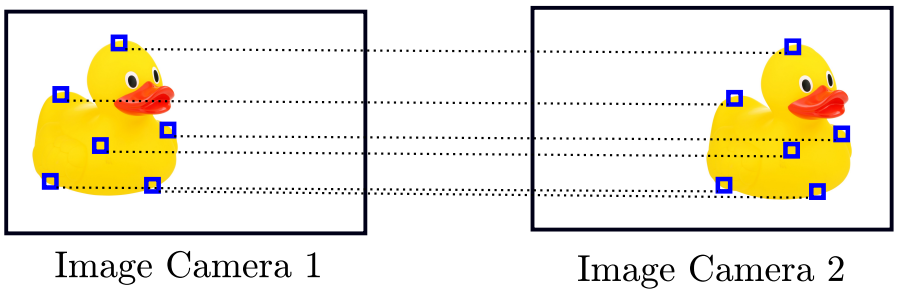


Figure 2.6: Point correlation between images

2.3.3 Epipolar Geometry

In the case of working with markers that have not an internal codification, as dot markers mentioned in Chapter 4 , it is still possible to correlate them. This can be done thanks to the known restrictions set by its epipolar geometry.

Considering a setup of two cameras as seen in Figure 2.7, let's define point O_1 and O_2 as camera 1 and 2 reference frames respectively. The point P projection onto the camera planes results in points s1 and s2 respectively. The line that passes through the points O_1 and O_2 is called the *based line*, and the intersection with the camera planes defines two points e_1 and e_2.

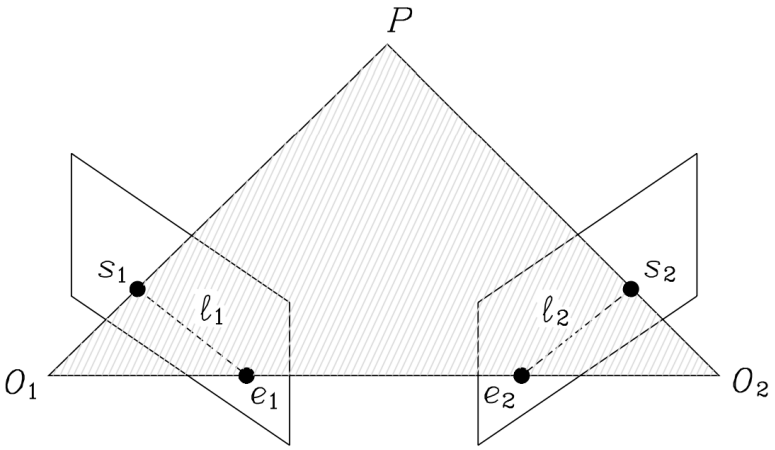


Figure 2.7: Epipolar geometry [18]

The location of point s_1 in camera one image defines immediately a constrain of s_2 in camera two images. The plane defined by points O_1, O_2, and s_1 is called the *epipolar plane* and contains point s_2. Because of this, the projection of P in the image plane of camera two, s_2, must belong to the line defined by the intersection between the epipolar plane with camera two image plane. These restrictions are express by equation 2.7.

$$s_1^T \, F \, s_2 = 0 \tag{2.7}$$

F is called the *Fundamental matrix* and can be computed with the intrinsic and extrinsic parameters. This constrain is very useful in the sense that, the correlative point from the image of one camera to another can be searched in the vicinity of the epipolar line, instead of looking at the overall image. Two correlated points, s_1 and s_2, will hardly satisfy equation 2.7 due to errors in measurements. Nonetheless, a threshold near zero can be defined to determine if two points are correlated.

This section was written based on Siciliano's book section 10.4.1 [19].

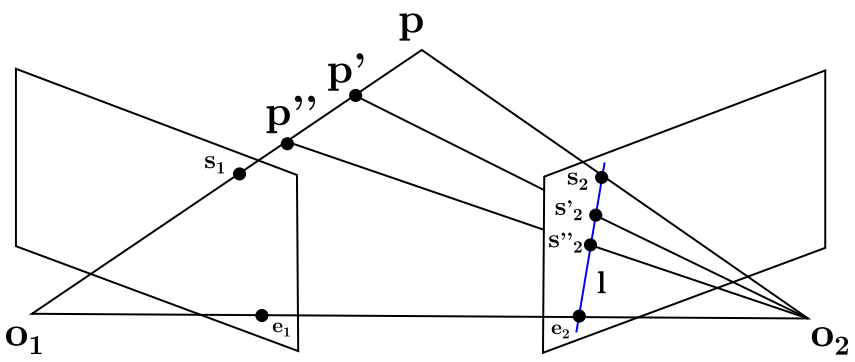


Figure 2.8: A point seen from an image can be located in any position of a line on the other image, based on [20]

2.4 Triangulation

2.4.1 Method

In the ideal case, it is possible to compute the position of a point considering its projection onto two different image planes. For the task at hand, these projections are the positions of a correlated point in the different images. Using the same nomenclature as in subsection 2.3.3, we define t as the translation vector from frame one to two, and R as the respective rotation matrix between them, these are the extrinsic parameters.

As seen in section 2.2.1, the line in space $\lambda_1 s_1$ represents all the positions of point P that could have generated the projection on image one, the same for $\lambda_2 s_2$ and image two. Now if we refer this second condition with respect to the frame from camera one, the following set of equations arises:

$$\begin{aligned} p_i &= \lambda_1 s_1 \\ p_i &= t + \lambda_2 R s_2 \end{aligned} \tag{2.8}$$

From this set of equations, p_i can be derived. However in practice, due to errors in measurements, the correlated points do not satisfy the epipolar constrain causing that the system 2.8 has no solution. From a graphical point of view, the backpropagated rays from both images will not intersect as shown in Figure 2.9

Taking this into account, an approximate solution can be derived through numerical methods based on least square algorithms. The Python library OpenCV has an internal command called `triangulatePoints`, that uses the DLT method mentioned in [20] [18].

This section was written based on Siciliano's book section 10.4.2 [19].

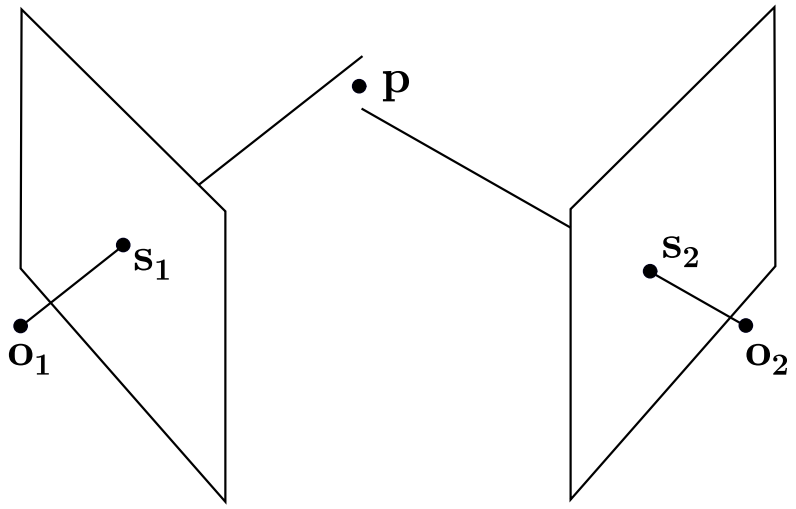


Figure 2.9: Triangulation: projections are backpropagated, but because of the measurement imperfection they do not coincide in a point, based on [18]

2.4.2 Theoretical precision

The analysis of the precision on the triangulation of a stereo vision system is in general complex. This section does not intend to provide a rigorous error analysis but instead, its aim is to find a tool that allows computing the optimal position for the cameras.

The following assumptions were made:

- Both cameras are equal and the intrinsic parameters are exactly known.
- The cameras are separated between each other on axis x a distance equal to the baseline **B**. They share the same coordinates for the other axes. Figure 2.10 a).
- The analysis would be based on a single point p, which position is shown in Figure 2.10 a). The point is located at the same radius for both cameras at a distance of z_b to the baseline. Its coordinates in X and Y are zero.
- Cameras are rotated only in the Z axis in such a way that the projection of point p on both camera images is located at its center.
- The only source of error considered is due to the camera resolution. The maximum error e_{max} represents the uncertainty of a single pixel, propagated a distance z_c from the cameras. e_{max} will be considered only as a function of z_c, which is the distance from both cameras to point p.

The uncertainty volume has a parallelepiped shape, it is shown in Figure 2.10 b) and c). It is formed as the union of the projection of a single-pixel error from both cameras. The projection of a single-pixel error is schematized in Figure 2.11.

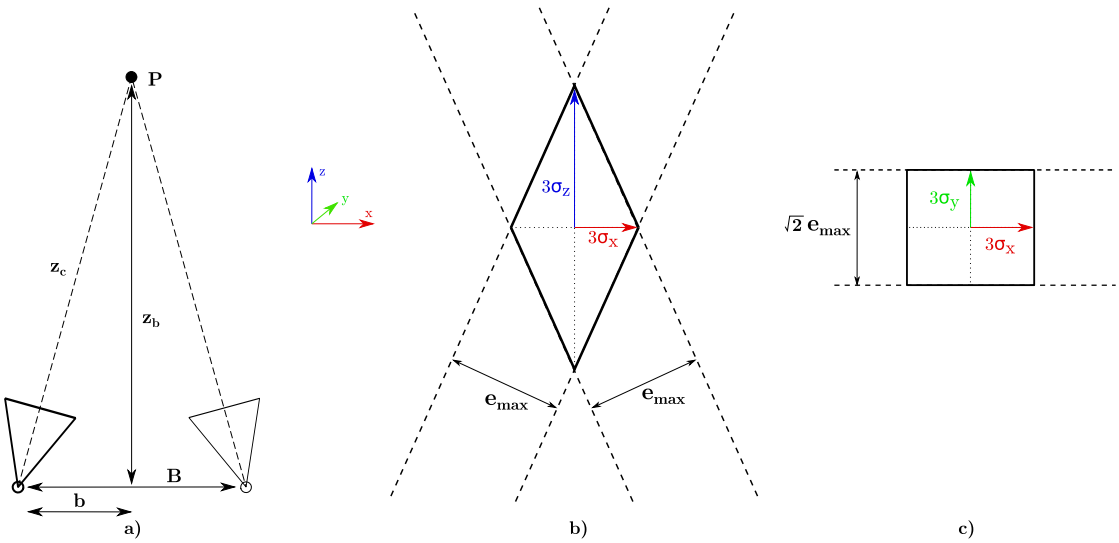


Figure 2.10: a) Configuration of the cameras and the point p to measure, b) Error volume in x-z plane defined by cameras error, c) Error volume in x-y plane

For the working condition of this thesis, e_{max} is several orders of magnitude smaller than Z_c. This causes that e_{max} remains virtually constant for all the error volume Figure 2.10 b).

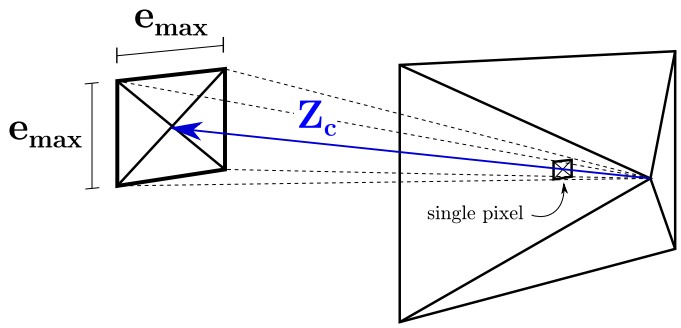


Figure 2.11: Projection of an error pixel error into space.

The parallelepiped defines the boundary of the error, but for the computation of the mean square error (MSE), it is necessary to compute the standard deviation in every main direction. The error for each axis is modeled as a gaussian distribution with a standard deviation σ_i for $i = x, y, z$, as shown in Figure 2.12. The new model is more conservative since its volume is larger than the previously defined parallelepiped.

The value of σ_i is computed as one-third of the distance from the centroid to the outer boundary of the parallelepiped in the direction of every axis. This set a confidence interval of 99,73% for every axis, and a total probability of 99,19% that a measured point lay inside the ellipsoid.

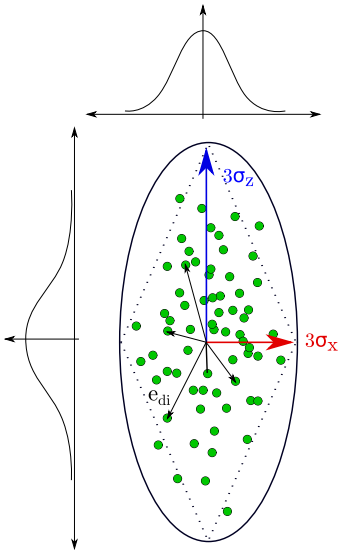


Figure 2.12: Volume error modeled as a combination of normally distributed errors in every axis, the diamond shape represents the previously considered error region.

The formula to compute each standard deviation is express in Equation 2.9, the procedure for obtaining it is explained in Appendix B.

$$\begin{aligned} \sigma_x &= \frac{z_c}{6\,z_b}\,e_{max} \\ \sigma_y &= \frac{\sqrt{2}}{6}\,e_{max} \\ \sigma_z &= \frac{z_c}{6\,b}\,e_{max} \end{aligned} \tag{2.9}$$

The only variables in the equations are z_b and B, since z_c and e_{max} are functions of these last two. The formula for obtaining e_{max} is expressed in Equation 2.10. Δe_{px} is the smallest division of the cameras, which in this case is one pixel. While f_l is the focal length of the cameras in pixels.

$$e_{max} = z_c \frac{\Delta e_{px}}{f_l} \tag{2.10}$$

In order to find the optimal position for the cameras, it is necessary to define first an objective function to be minimized. The chosen loss function was the variance of the error distance, this can be computed as express in Equation 2.11, the derivation of the

formula is explained in Appendix B.

$$\sigma_d^2 = \sigma_x^2 + \sigma_y^2 + \sigma_z^2 \tag{2.11}$$

In Appendix B it is also demonstrated that the standard deviation σ_d is equal to the Root Mean Squared Error (RMSE):

$$RMSE = \sigma_d \tag{2.12}$$

The optimal value for the baseline resulted to be approximately 1.2872 times z_b also derived in Appendix B.

$$B_{opt} = 1.2872\, z_b \tag{2.13}$$

2.5 Pose estimation

2.5.1 Pose of a Rigid Body

The complete definition of the orientation and position of a rigid body in space is known as its pose. It is always measured related to a frame of reference. For its complete description at least 6 parameters must be known: 3 for position and 3 for orientation.

Position can be expressed using an X-Y-Z Cartesian coordinates system, while orientation can be defined by its Euler angles. There are other ways of defining this last such as Rotation Matrices and Quaternions, both systems have more parameters than required, 9 and 4 respectively, however, their degrees of freedom are still 3.

Quaternions are particularly useful to solve a problem known as 'Gimbal lock' where at certain configurations one degree of freedom is lost. Because of this, all codes necessary for testing the measuring system were developed using quaternions for orientation definition.

2.5.2 Position and Orientation computation

Let us define a frame of reference with center in O_p attached to a rigid body and make that reference frame coincides with the world reference frame. P is a set of points attached to that rigid body, called a *Rigid cloud of points*.

Let us now call Q the measured cloud of points in the actual position. Finding $\phi = (\varphi, \vartheta, \psi)$ and t that satisfy the equation 2.14, would define the pose of the rigid body.

$$q_i = t + R(\varphi, \vartheta, \psi)\, p_i \quad \text{for } i = 1, \ldots, N \tag{2.14}$$

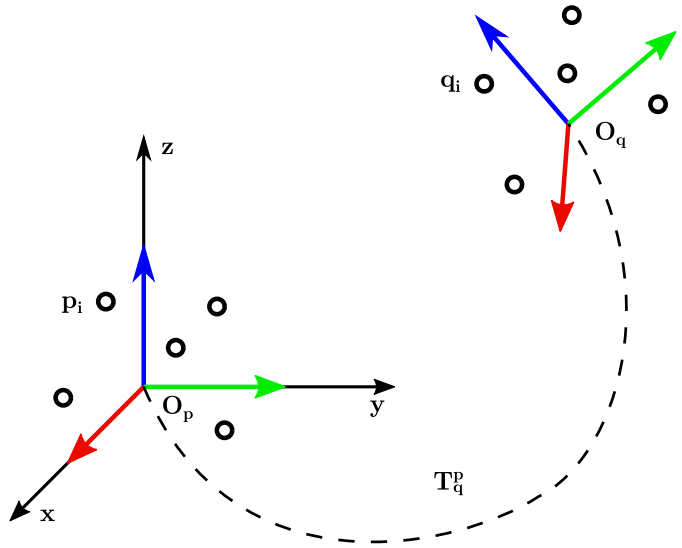


Figure 2.13: Estimating the pose of a body consist in determine the transformation that it is necessary to be applied from it zero FR, to match in position and orientation with the actual pose.

In real applications, and because of errors in measurements, it is possible that equation 2.14 could not be satisfied. Instead, an estimation of the pose of the rigid body could be obtained, computing the values of R and t that minimize the loss function E defined in expression 2.15.

$$E(R, t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||q_i - R\, p_i - t||^2 \tag{2.15}$$

Hopefully, there is a closed-form expression to compute R and t, if there exist a correspondence between points from Q and P.

$$\begin{aligned} Q &= \{q_1, ..., q_n\} \\ P &= \{p_1, ..., p_n\} \end{aligned} \tag{2.16}$$

The center of masses of each cloud point can be defined as:

$$\begin{aligned} \mu_q &= \frac{1}{N_q}\sum_{i=1}^{N_q} q_i \\ \mu_p &= \frac{1}{N_p}\sum_{i=1}^{N_p} p_i \end{aligned} \tag{2.17}$$

Extracting its respective centers of masses to Q and P we obtain Q' and P':

$$\begin{aligned} Q' &= \{q_i - \mu_q\} \\ P' &= \{p_i - \mu_p\} \end{aligned} \tag{2.18}$$

With this new sets of points we define the matrix W which definition is the following:

$$W = \sum_{i=1}^{N_p} q_i' \, p_i'^T \tag{2.19}$$

Doing the SVD decomposition of matrix W we obtain:

$$W = U\Sigma V^T \tag{2.20}$$

The values of R and t that minimize expression 2.15 are the following:

$$\begin{aligned} R &= U\,V^T \\ t &= \mu_q - R\,\mu_p \end{aligned} \tag{2.21}$$

This set of equations are based on [21].

2.5.3 Iterative closest points (ICP)

In section 2.5.2, it was explained that a closed solution exists if there is already a correspondence between the points of Q and P. If such correlation is not defined, we still can apply the iterative closest point algorithm (ICP).

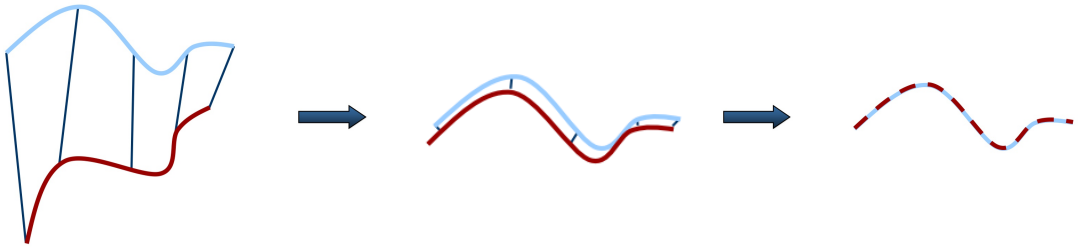


Figure 2.14: Schematic representation of a 3 step ICP process to align two curves. Image from [21]

The general idea is to correlate every point p_i with the point of Q located at the minimum distance from it. In this way, every point of P would have an initial correlated point from Q.

Then, with this initial correlation, we can compute the optimal value of R and t as seen in section 2.5.2. Points from P are rotated and translated according to these values. The closest point from p_i in Q does not guarantee a good correlation, however, the new position will probably be closer than the initial one.

The process is then repeated, a new correlation between the two sets is made and the rotation matrix and the translation vector are computed and finally, points in P are transformed again. Iterating this procedure would converge in a solution. The resulting homogeneous transformation matrix will be the combination of the n transformations produced in P.

It is not guaranteed that the result will lead to the optimal solution, the algorithm can be stuck in a local minimum. However, in many cases, this algorithm results in the optimal value.

Algorithm 1 Summary of ICP algorithm.

Require: ${}^{\mathbb{A}}\mathcal{P}$ ▷ `reading`

Require: ${}^{\mathbb{B}}\mathcal{Q}$ ▷ `reference`

Require: $\mathcal{T}_{init}$ ▷ initial transformation

${}^{\mathbb{A}}\mathcal{P}' \leftarrow \text{datafilter}({}^{\mathbb{A}}\mathcal{P})$ ▷ data filters

${}^{\mathbb{B}}\mathcal{Q}' \leftarrow \text{datafilter}({}^{\mathbb{B}}\mathcal{Q})$ ▷ data filters

${}_{i-1}{}^{i}\mathcal{T} \leftarrow \mathcal{T}_{init}$

repeat

${}^{i}\mathcal{P}' \leftarrow {}_{i-1}{}^{i}\mathcal{T}({}^{i-1}\mathcal{P}')$ ▷ move `reading`

$\mathcal{M}_i \leftarrow \text{match}({}^{i}\mathcal{P}', \mathcal{Q}')$ ▷ associate points

$\mathcal{W}_i \leftarrow \text{outlier}(\mathcal{M}_i)$ ▷ filter outliers

${}^{i+1}_{i}\mathcal{T} \leftarrow \arg\min_{\mathcal{T}} \left(\text{error}\left(\mathcal{T}\left({}^{i}\mathcal{P}'\right), \mathcal{Q}'\right)\right)$

until convergence

Ensure: ${}^{\mathbb{B}}_{\mathbb{A}}\hat{\mathcal{T}} = \left(\bigcirc_{i} {}_{i-1}{}^{i}\mathcal{T}\right) \circ \mathcal{T}_{init}$

Figure 2.15: Algorithm by Hanzhou Lu [22]

Chapter 3

Pose estimation with ArUco Markers

3.1 Marker description

ArUco markers are a type of square fiducials that possess an internal codification. This characteristic makes them very useful for correlation in stereo vision systems (SVS). Each marker can be unequivocally identified on both images of an SVS, and the corners of it can be triangulated. Another characteristic that may explain its popularity, is that a single marker provides sufficient information for estimating its pose with a single camera.

They are easily detectable through CV algorithms since they have a thick black border that contrasts with their white background. The Aruco library was first proposed by Garrido and Muñoz [23].

Another benefit of working with ArUco markers is that the Python library OpenCV offers a great number of tools for working with them. For example, there is a buit-in function for computing its pose in space. For this, it is necessary to know a priori the intrinsic parameters of the camera and the size of the marker.

Its internal codification allows a robust determination of its Id. Its binary arrangement is compared with a predefined library and a threshold of correlation can be set.

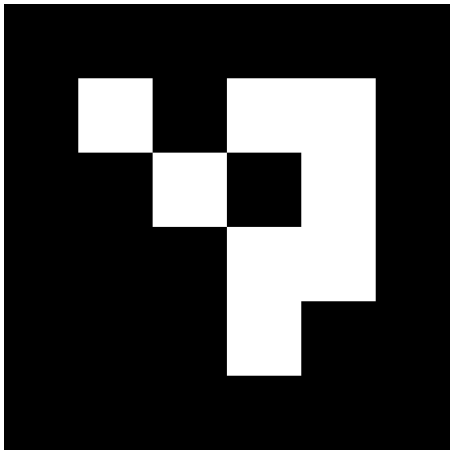


Figure 3.1: 4x4 ArUco marker with id = 0

3.2 Pose estimation with a single camera

It is possible to estimate the pose of ArUco markers relative to a camera if we know beforehand the shape and dimensions of them. Also, we need the intrinsic parameters of the camera.

According to [24], the projection of four coplanar points on an image are sufficient for estimating the pose of the object, if there is no possible combination of three of these points being collinear. This is the case of a square marker and an example of it can be found in Siciliano's Book section 10.3 Pose Estimation [24].

OpenCV has a build-in operation for this purpose:

```
1 [rvecs, tvecs] = cv.aruco.estimatePoseSingleMarkers(markerCorners,
    markerLengthSide, cameraMatrix, distCoeffs)}
```

Listing 3.1: ArUco pose estimation method with a single camera.

As it can be seen in listing 3.1, the method returns the relative position and orientation of the marker with respect to the camera. Their inputs are the position of the four corners of the marker on the image, the length of the marker side, the camera matrix, and the distortion coefficients of the camera.

As it would be shown in Section 3.6, a small error in the location of the corners on the image, affects significantly the estimation. Since this project tries to achieve the smallest possible error, a stereo vision system was proposed.

3.3 Subpixel algorithms

The location of a point in an image is determined by its position in X and Y of the image plane. In general, they are expressed in an integer quantity of pixels, since a single pixel is the minimum quantity of information in an image.

However, there exist corner detection algorithms that try to estimate the location of a point with higher accuracy. Since they aim to achieve an accuracy smaller than a pixel are called *Subpixel algorithm*.

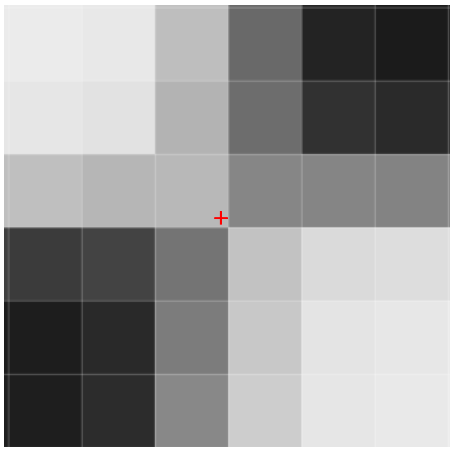


Figure 3.2: Subpixel corner detection

OpenCV counts with an internal function called **cornerSubPix()** that improves the corner detection process through the technique proposed by [25]. Despite this, its performance with ArUco markers corners is significantly inferior to chessboard-like patterns (Figure 3.3). At least this is suggested by the preliminary test shown later in this document (Section 3.6).

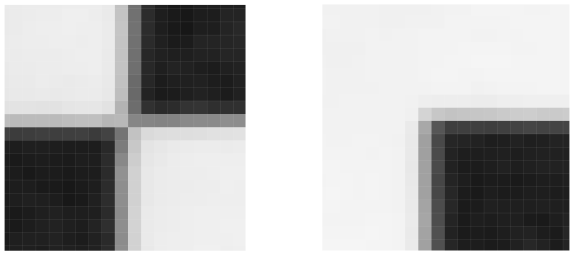


Figure 3.3: a) Chessboard corners b) ArUco marker corners

3.4 Enhanced ArUco Markers

To improve the accuracy of the detection system, it was decided to modify the standard ArUco marker to make possible the use of OpenCV corner subpixel function.

The first attempt was to modify its corners by adding little black squares to resemble a chessboard pattern, as proposed by the creator of ArUco markers Rafael Muñoz Salinas [26]. However these changes make the marker more difficult to detect, and in some cases, they were detected incorrectly as can be seen in Figure 3.4.

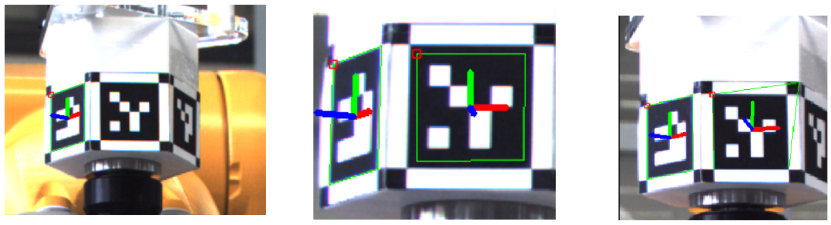


Figure 3.4: Error in detection: a) Not all the markers are detected b) The green line does not match the perimeter of the marker c) One of the corners is erroneously detected

Because of this, a new arrangement was proposed based on ChArUco calibration boards [27]. These are characterized by allowing calibration even under board occlusion, and because of the chessboard pattern, the location of its corners can be refined with subpixel algorithms.

The proposed marker is shown in Figure 3.5. It consists of a 4x4 ArUco marker inside a chessboard-like pattern. The detection algorithm for this modified marker takes a few more steps than the standard version, described as follows.

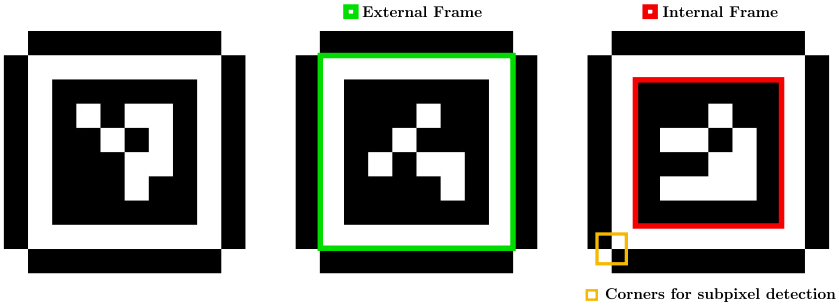


Figure 3.5: Parts of enhanced marker

First, the pose of the aruco marker is estimated as in the normal procedure, the size of the internal frame should be known. Then with the obtained information, and the camera

intrinsic matrix, a virtual square with the size of the external frame is projected on the image with the same location and orientation as the standard marker. The location of these new points is used as input for the OpenCV function `cornerSubPix` that refines the location of these corners with subpixel accuracy.

The pose of the marker is computed again with the location of these new points, the intrinsic parameter, and the size of the external frame.

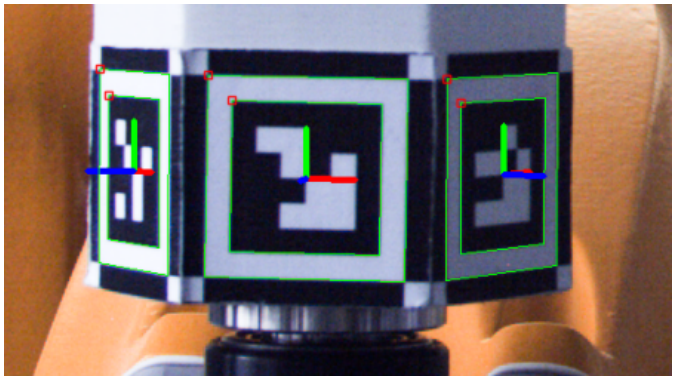


Figure 3.6: Detection of the inner and outer frame of enhanced ArUco markers

3.5 Pose estimation with Stereo Vision

The codification in these types of markers also allows locating every corner of them without ambiguity. This is very helpful for correlating the corners detected in cameras one and two. With this relation between points, the position of them in space can be triangulated as seen in Section 2.4.

It is important to highlight that, no matter the orientation of the marker in the picture, the corners will always be detected in the same order (Figure 3.7).

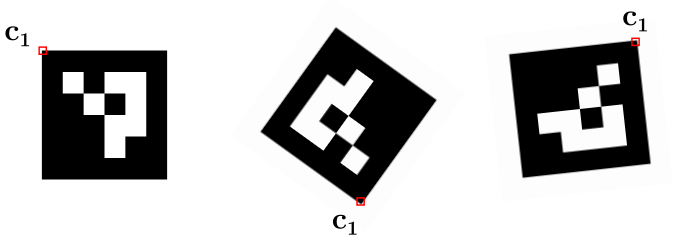


Figure 3.7: Detection of corner C1 for different orientations.

Moreover, with the information of the position of the four corners in space, the pose can be estimated with the formula seen in Section 2.5.

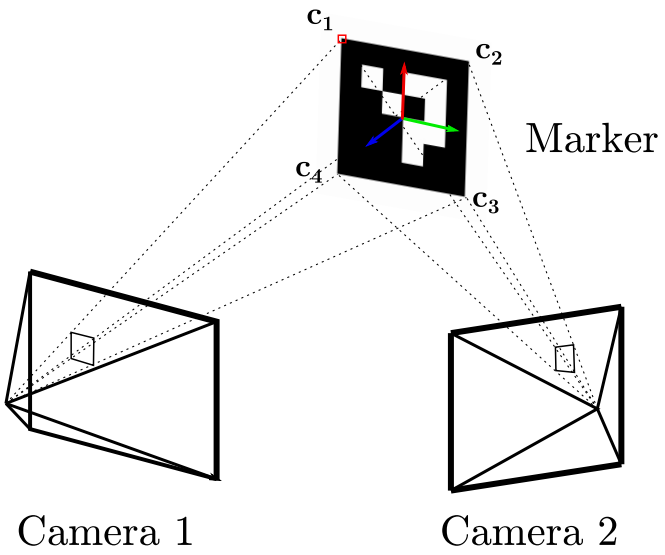


Figure 3.8: Projections of ArUco marker corners into camera planes.

3.6 Preliminary test results

To have a rough estimation of the precision of the detection of each type of marker and method, a virtual test environment was set. Using the software Blender, a set of two cameras located 300 mm apart from each other and at an angle of 7.5° on Z axis, were pointing towards an Array of ArUco markers.

Both cameras have a resolution of 1080 x 1080 px and are located 1000 mm away from the markers, its focal length is 50 mm and its sensor size is 14.2 mm x 14.2 mm.

Both of them were previously *virtually calibrated* with a chessboard in 20 different poses, located at the same distance as the markers, as described in Section 2.2.3. Some of the calibration images can be seen in Figure 3.10.

The virtual calibration is exactly the same as it would be with real images. However in this case, the calibration images are computer generated. Because we are working with an ideal model , it is possible to extract the exact intrinsic and extrinsic parameters. However, it was decided to rely on calibration images to be in accordance with the real experiment. Since the computer generated images (CGI) have no lenses distortion, it is expected that the output error will be lower than from the real application.

The test consists simply on using the cameras to measure the position of the markers to compute an estimated error. One set of images were taken to an array of standard markers, and another set to the *Enhanced* version of them. Only four images were necessary for this test, they are shown in Figure 3.11.

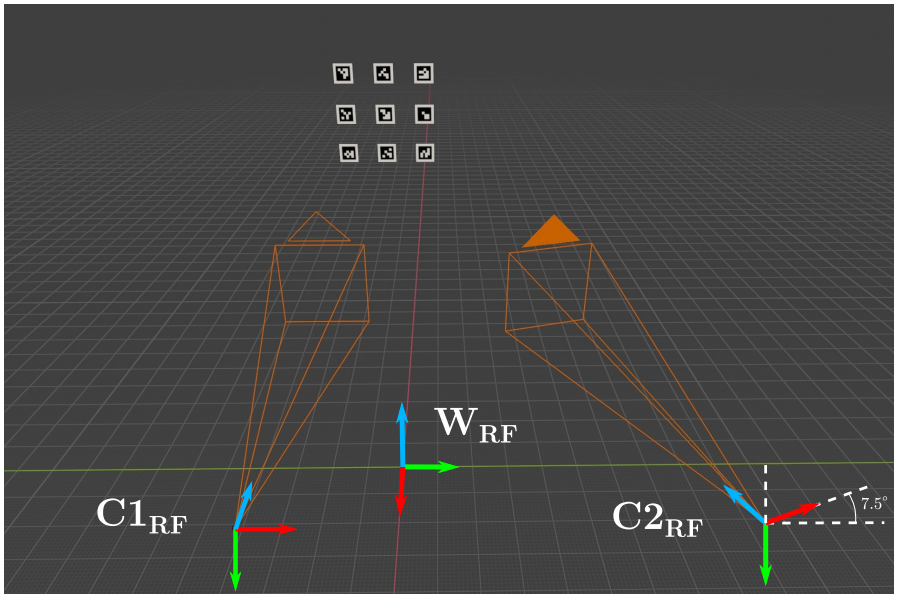


Figure 3.9: Virtual Set-up configuration.

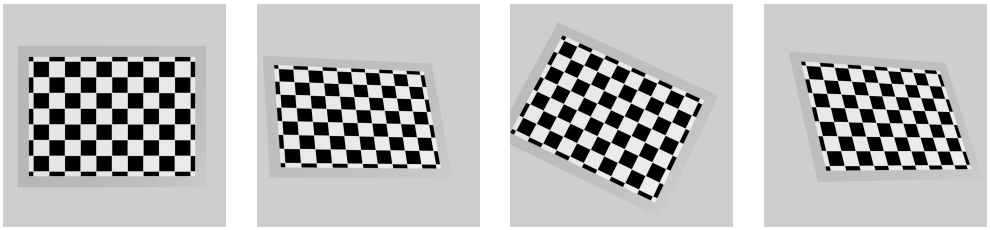


Figure 3.10: Some calibration images that were used.

The markers can be precisely located in the virtual space, so the information of the real pose is available. These values are compared with the measurements of the different CV algorithms, and the Root mean square error (RMSE) is computed for the different axes. This metric can be computed as follows:

$$RMSE_x = \sqrt{\frac{\sum_{i=1}^{N}(x_{ri} - x_{ci})^2}{N}} = \sqrt{\frac{\sum_{i=1}^{N} e_{x,i}^2}{N}} \tag{3.1}$$

N is the total number of markers (9 in this case), x_{ri} is the ground truth position in x while x_{ci} is the computed one. A similar formula can be use for the rest of the axes.

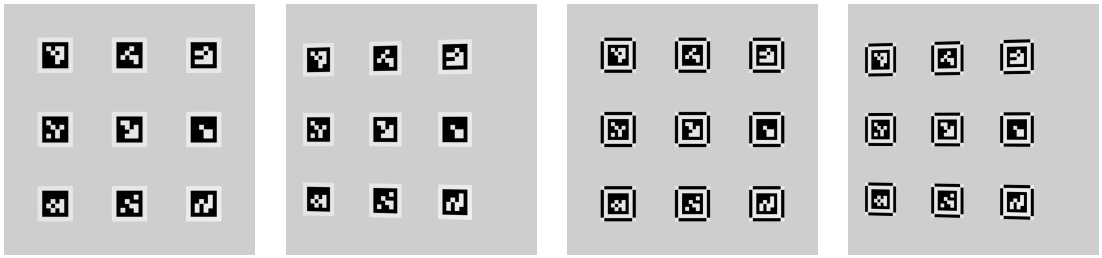


Figure 3.11: Test images: In the first two, the standard ArUco marker is present while in the last two the enhanced version of it. The first and third image are from camera 1 while the second and fourth one from camera 2.

The total root mean square error ($RMSE_T$) can be computed as follows:

$$RMSE_T = \sqrt{RMSE_x^2 + RMSE_y^2 + RMSE_z^2} \tag{3.2}$$

Table 3.1 shows the $RMSE_T$ for different combinations: with subpixel algorithms or not, using standard markers or enhanced ones, and with stereo or monovision.

Subpixel	**Marker**	**Method**	**RMSE X (mm)**	**RMSE Y (mm)**	**RMSE Z (mm)**	**RMSE T (mm)**
Yes	Enhanced	Stereo	0.1358	0.1352	0.0683	0.2035
Yes	Enhanced	Single	0.1287	0.1367	0.3436	0.3916
Yes	Standard	Single	1.2799	1.2852	18.7897	18.8771
No	Standard	Single	1.3552	1.4203	20.9826	21.0742
Subpixel	**Marker**	**Method**	**Std e_x (mm)**	**Std e_y (mm)**	**Std e_z (mm)**	**Std Tot (mm)**
Yes	Enhanced	Stereo	0.0087	0.0044	0.0720	0.0727
Yes	Enhanced	Single	0.01871	0.02652	0.33346	0.3350
Yes	Standard	Single	1.35300	1.35631	0.56851	1.9983
No	Standard	Single	1.43539	1.50186	2.75213	3.4482

Table 3.1: Preliminary test results

In monovision configuration, the subpixel algorithm improves the $RMSE_T$ by only **10%**, but the total standard deviation (std_T) a **42%**. However, the improvement with the enhanced markers is **98%**, which means that the error was reduced almost **50** times. Moreover, the standard deviation is reduced **6** times

The stereo vision system additionally reduced the error to a half, while the standard deviation was reduced to a quarter.

This test, despite being simple, provides a clear estimation of the error. Taking this into account, the research will be focused on building an accurate and reliable stereo vision system.

3.7 Super-Resolution

In a computer vision system, one of the main factors that limit the accuracy of the measurements is the camera's resolution. The most important metrics to consider for the task at hand is the area that a single pixel represents in an image, the smallest the better.

However higher resolution cameras are in general more expensive, and their output images take longer processing time. To sort these barriers, a state-of-the-art technique was proposed to increase artificially the resolution of the images.

In the last years, several investigations regarding super-resolution algorithms were published [28] [29] [30]. The majority of them are based on convolutional neural networks (CNN) that take as input low-resolution images and return as output the high-resolution version of it. They are called Super-Resolution Neural Networks (SRNN).

As for every other CNN, an input and output dataset is needed for training the network. Moreover, it is necessary to define the architecture of the network, and the objective function metric that would be used to tune the model. All of these elements will be explained in the next subsections.

3.7.1 Dataset

The most important elements in the images of the SVS are the corners of the enhanced ArUco markers. It is not necessary to increase the resolution of every part of the images, the focus will be put just on the corners. This would make the training of the SRNN less demanding since the input images would be smaller, and the input and output image diversity is significantly reduced.

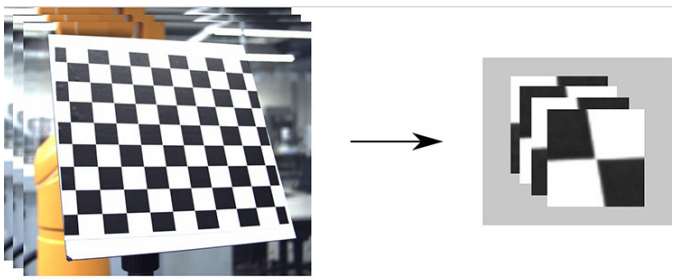


Figure 3.12: Corner extraction from calibration images

The dataset was built with calibration images where a chessboard pattern was used. First, the location of the corners was detected by the same procedure used for calibration.

After a successful localization of each corner, 64x64 pixel sections of the image were extracted. These new images contained the corners, but they were not located precisely in the center, it was decided to move the square frame randomly a few pixels apart to make the model more robust. (Figure 3.13)

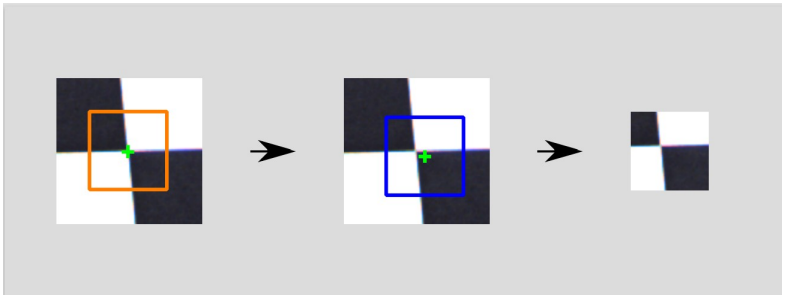


Figure 3.13: After detecting the corner a 64 by 64 pixel frame is defined centered on it. The frame is then moved apart randomly a few pixels from the original position. Finally, the image inside the latter frame is used for training data

These high-resolution (HR) corner images were used as the output y of the SRNN. For generating the low resolution (LR) images for input x, a blur kernel was applied to the HR images and was downsampled to a dimension of 16x16 pixels. With the aim of obtaining a most robust model, noise was added to the resulting images. These steps are summarized in Equation 3.3, the equation is present in [31].

$$x = (y \otimes k) \downarrow_s + n \tag{3.3}$$

The blur kernel k consists of a square matrix with all its elements equal to one, this matrix is convoluted with the corner image. The size of k varies randomly from 1x1 px to 15x15 px. After downsampling the image, a noise matrix n is added to the resulting corner image. It modifies each pixel value up to 5 points in brightness (the scale is from 0 to 255). If a pixel reaches a value outside the range of 0 to 255, its value is set to the closest limit. A graphic representation of this process is shown in Figure 3.14

The resulting dataset was 44832 corner images of 64x64 px as desired output and the same amount of input images of 16x16 px.

3.7.2 Network Architecture

The vast majority of sequential neural networks tend to reduce the amount of information from its input to its output. A very common example of it are image classification problems, where the input is a picture and the output is just a label.

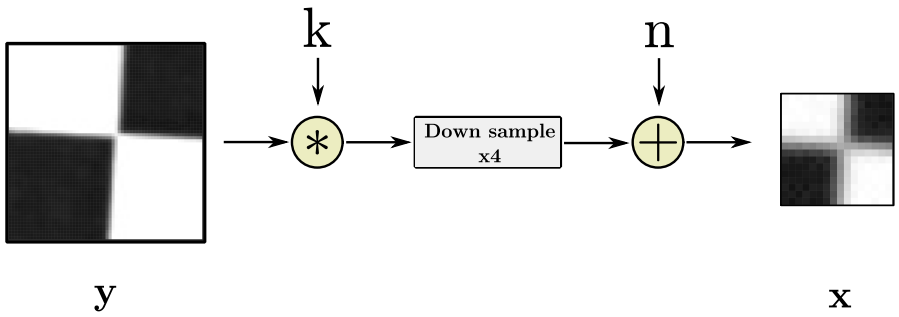


Figure 3.14: Generation of input dataset: k is a square matrix kernel, while n is a noise matrix with the size of the output image.

However, for the task at hand, the amount of information in the output is higher than on the input. It is necessary to increase four times the resolution of an image. At a first glance, it would look like this process is violating the Data Processing Inequality concept, which expresses that it is not possible to increase the amount of information with an operation. Nonetheless, the training process itself adds information to the net, which makes the augmentation in resolution possible, without being against the concept previously mentioned.

A possible approach for an SRNN is to first scale the LR image through a bicubic interpolation method making its size equal to the desired output. This modified image would then be processed through different convolutional layers with the hope that the output would be similar to the desired one. In this way, the size of the image would remain the same at every layer. However, according to [31], this process unnecessarily increase the processing time and it may introduce arbitrary smooth effects

Convolutional layers reduce or maintain the size of an image (Figure 3.15 a)). For this case the shape of the image has to increase, it is because of this that transpose convolutional layers would be used. This type of operation consists of multiplying a nxn kernel with a single element from the input, and add the result to the output (Figure 3.15 b)). If the input is expanded with strides the output will augment it's size (Figure 3.15 c)).

Taking all of this into consideration, the final architecture is plotted in Figure 3.16. It consists of 11 layers: Except for the first one, all the convolutional layers use six 3x3 kernels generating six different outputs. The first one, on the other hand, has six kernels with a size of 6x6. The only layers that are not convolutional are number 5 and 9, which are transpose convolution and have the property of augmenting the output size by 2. To avoid overfitting, before each transpose convolution a dropout step is placed.

The network was coded in Python using the Tensorflow library with Keras layers. The complete model is in Appendix C.

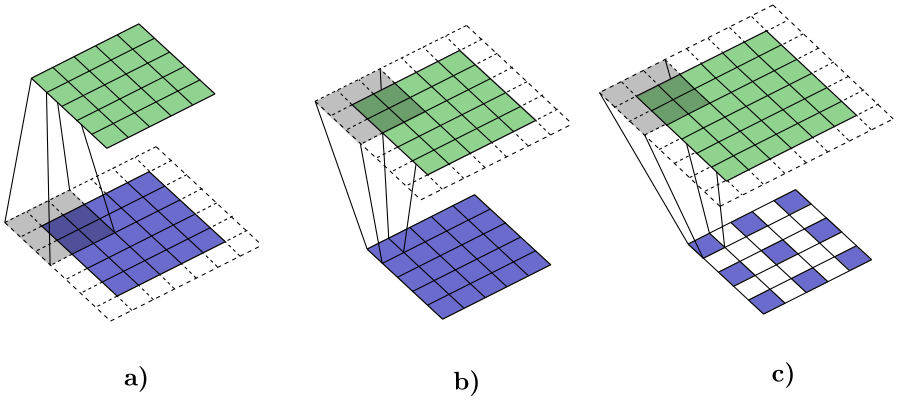


Figure 3.15: Generation of input dataset, figures based on [32]. a) Convolutional layer b)Transpose Convolution layer c) Transpose Convolution layer with input strides

The model was trained using 44832 corner images, which were divided into 404 batches. The total number of epochs for training were 4. To avoid overfitting, the training process was stopped when the validation error started to increase. (Figure 3.17)

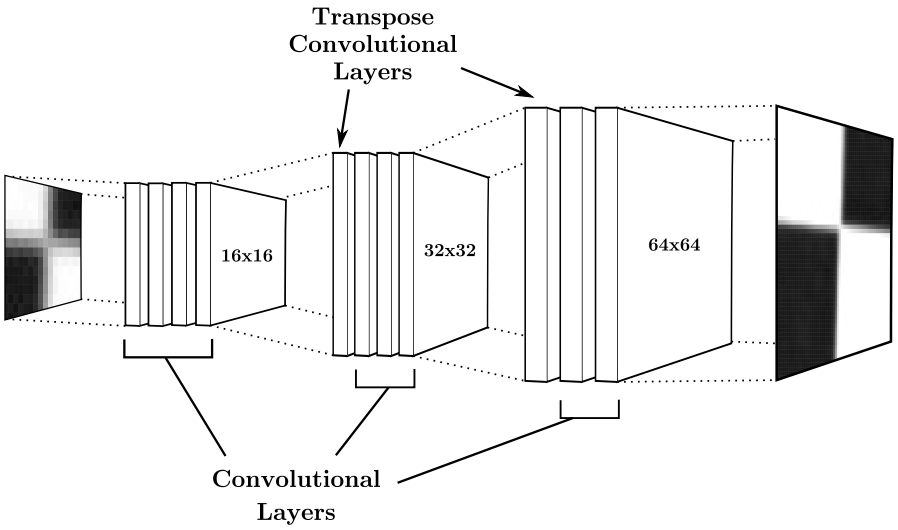


Figure 3.16: SRNN Architecture

3.7.3 Loss function

The most simple metric to consider is the mean square error (MSE). For the task at hand, each pixel in the output image is compared with its correspondent pixel in the desired output. The difference of intensity between both is squared, and the result is obtained as the average of this value for the whole image.

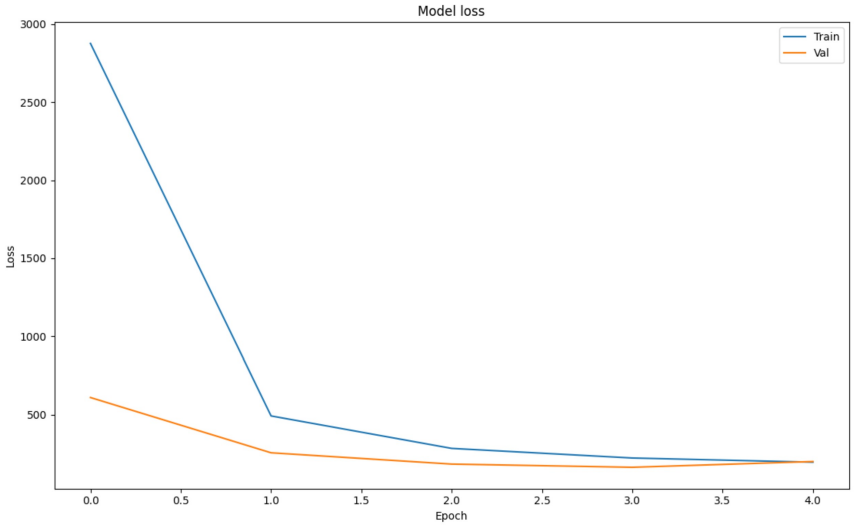


Figure 3.17: MSE error after each epoch

Nonetheless, the metrics used for training a SRNN are in general based on how real the output image looks for a human being. The most popular one is the perceptual loss, where both the output image and the desired output are transformed into feature vectors. These features are in general extracted from the first layers of pre-trained CNN. The values of the feature vectors are compared to obtain a final metric.

According to [31], the use of perceptual loss as the objective function for training a SRNN, generates more pleasant and realistic images. This is because the chosen features were from a CNN trained with a large number of real images. Therefore, the output that looks more realistic would achieve lower error.

However, perceptual loss in the search for more realistic images, leaves behind the correctness of every pixel value. The utility of SRNN for this research is not related to obtaining more realistic images, but to obtain a higher resolution image as similar as the original. It is because of this that the MSE metric would be used as the objective function.

3.7.4 Output

The SRNN was trained with a specific dataset dimension, nonetheless, since it is a convolutional network an image of any size can be an input. This is because the kernels are the trained elements and they can be used for convoluting any image size. The only condition is that the input image is larger than any kernel.

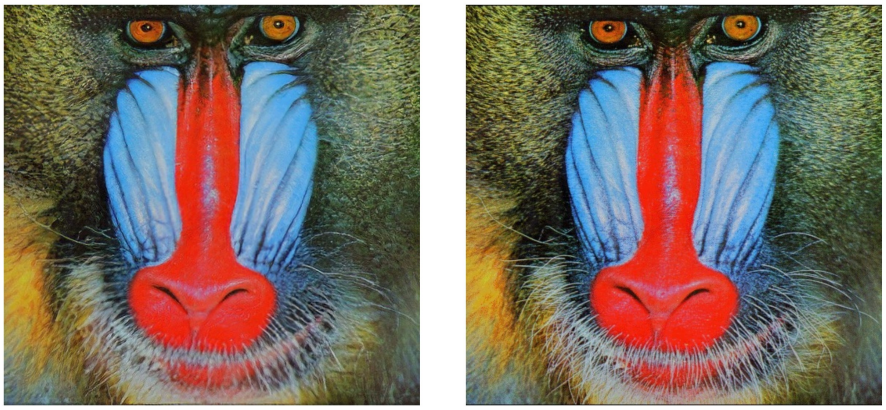


Figure 3.18: The left image was obtained with a SRNN trained with perceptual loss as it object function, the original image is the one at the right. There are part of the fur in the left image that is not present in the right one [33]

Figure 3.19 shows the result of some validation images. Figure 3.20 shows the output of a complete Enhance ArUco marker, it is important to recall that the training data were only corner images. In spite of this, the model returns, in any case, sharper images.

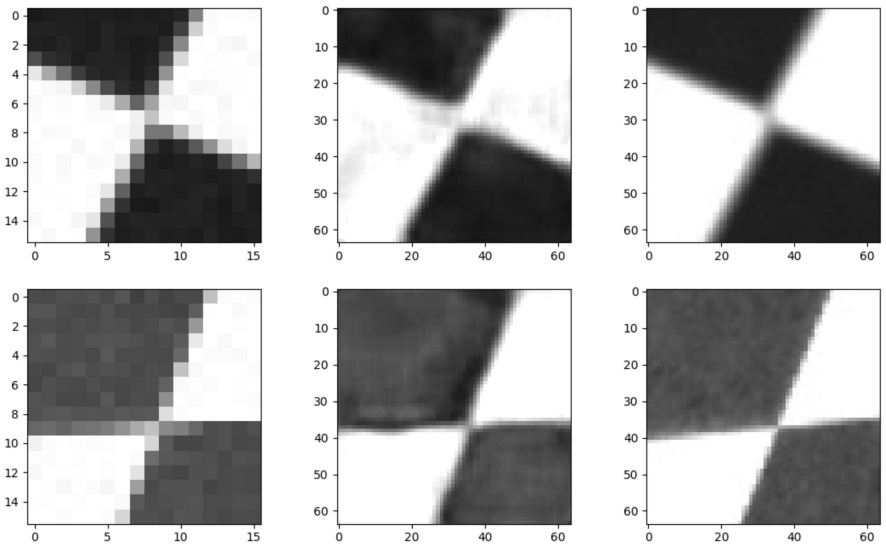


Figure 3.19: Test dataset: In the first column there are the LR images, in the middle the HR predictions, and finally in the right column the original HR images

Although the HR images are sharper than the LR images, in Chapter 6 it would be shown that this process does not improve significantly the accuracy of the system.

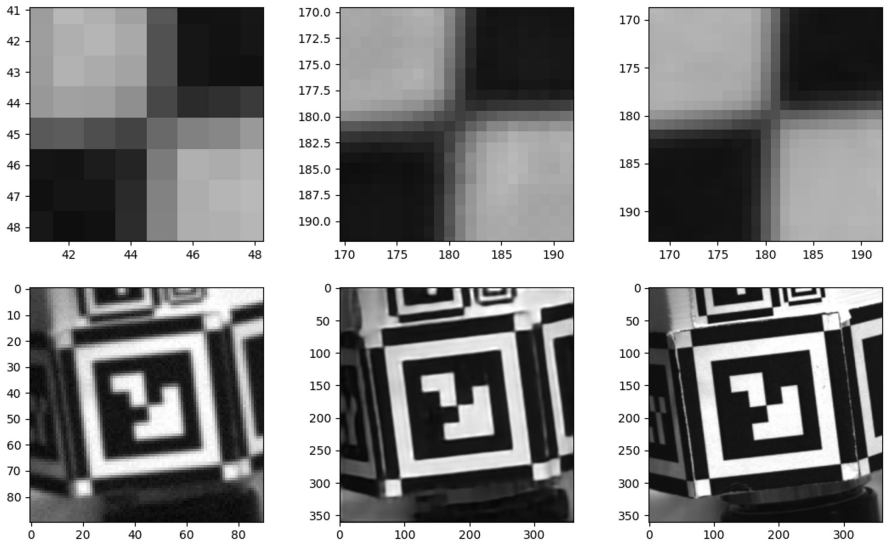


Figure 3.20: The first row is an expansion of images from the second row, which is a an image from an enhance ArUco marker. In the first column there are the LR images, in the middle the HR predictions, and finally in the right column the original HR images

Chapter 4

Pose estimation with CCC Markers

4.1 Marker description

Concentric Contrasting Circle (CCC) fiducial markers were first proposed by Gatrell in 1992 [34]. It consists simply of a solid white circle inside a black one, illustrated in Figure 4.1. They must be placed over a white background or have an additional white border.

The detection algorithm consists in 3 steps:

1. **Binarize the image:** the image is transformed to grayscale and then an arbitrary brightness threshold level is determined. The pixels with a value above this threshold will automatically set to the highest possible brightness value, while the ones whose value is under or equal to the threshold will be set to 0.
2. **Detect white and black blobs and compute its centroid:** connected pixels from the same colors from different regions are called Blobs. For each of these regions the centroid is computed.
3. **Compare the centroid of white and black blobs:** The centroids of white and black blobs are compared, if there is an almost identical value of centroid from a white and black blob that point is considered to be a marker.

In contrast with ArUco markers, a single CCC marker does not provide by its own sufficient information to compute its pose in space, neither with monovision nor with stereovision. Also, since they are not internally codified, it is harder to correlate them in

different images from stereo vision applications.

In spite of this, these types of markers are smaller, and thus, more practical to attach to a sample. Aruco markers to be identified, need to have a considerable size and be planar, so they are not useful for non-planar objects.

If we are able to detect at least 3 markers from an object in a stereo vision set-up, and we can successfully correlate them from the different camera images, it is possible to compute the pose of the sample in space. This can be done only if we know a priori the position of the markers with respect to the object.

It is because of this, that a marker scanner system was developed. It gives as output a 3D map of the relative location of every marker attached to a sample. This can be achieved using the same cameras for stereo vision, so no additional equipment is necessary.

Figure 4.1: CCC marker

4.2 Blob Detection

The term *Blob Detection* refers to the identification of regions in an image that shares some kind of common property [35]. For the particular case of CCC markers detection, It is necessary to identify connected bright or dark regions. Because these type of fiducials contain black and white regions, after the binarization of the image, they would be identified as individual blobs with almost any threshold.

In this type of circular markers, the white region is contained inside a solid black one. This allows that after binarization, the white regions remain isolated from other regions, which permits that the detection algorithm identifies each white circle as a single blob. The centroid of each blob is computed, and also some other properties that would make possible to filter out some of them.

OpenCV counts with a tool that allows to identify these regions and is called *SimpleBlobDetector*. Many parameters can be tuned for their use on special applications, in the

next subsection, it would be described the different options that can be set up.

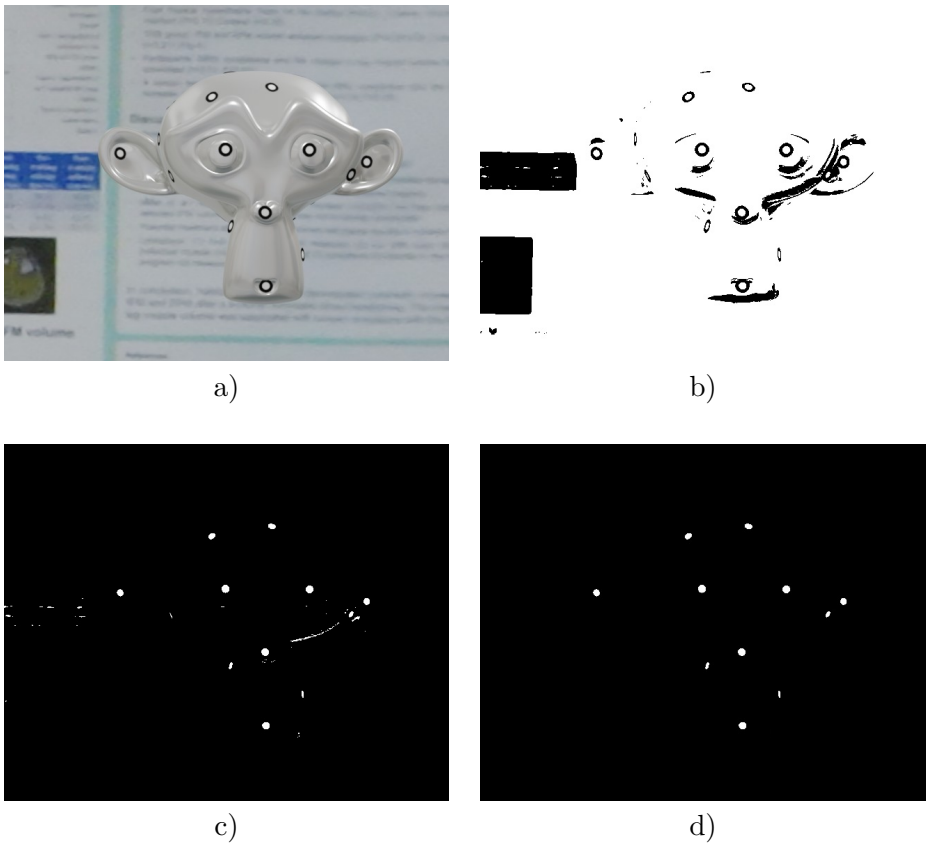


Figure 4.2: Blob detection example: a) Original image, b) Image after binarization, c) Filtered image by maximum blob area, d) Filtered image by minimum blob area.

4.2.1 Filters

Identifying the white isolated regions in an image is not enough for detecting CCC markers. If only this policy is applied, a large number of regions that are not markers would be detected, resulting in a great number of false positives. To avoid these problems, several filters can be applied, the ones that are already available with the OpenCV function are described below [36]:

- **Color Filter:** It filters out dark or bright blobs by the value at its center.
- **Area Filter:** A minimum and a maximum threshold can be set, to filter out any blob that has an area outside a certain range.
- **Circularity Filter:** The Circularity is defined as: $\frac{4\,\pi\,Area}{perimeter^2}$. As with the area

filter, we can set a range of circularity values. Any blob outside this range will be filtered out.

- **Inertia ratio Filter:** The minimum inertia of the blob is divided by the maximum inertia of the blob to obtain an *Inertia rate*. A range of valid rates can be set to filter out blobs outside these values.
- **Convexity Filter:** Convexity is defined as: $\frac{Blob\ Area}{Blob\ Convex\ hull\ Area}$. A minimum and maximum value can be set to filter out blobs outside this range.

The most relevant one resulted to be the Area filter. This range can be set computing the size of the markers with the information of the expected distance from the cameras. The value of the other filters were tuned after several trials with sample data, some of them can even be ignored. Other filters can be applied in posterior steps, and would be described in the following sections.

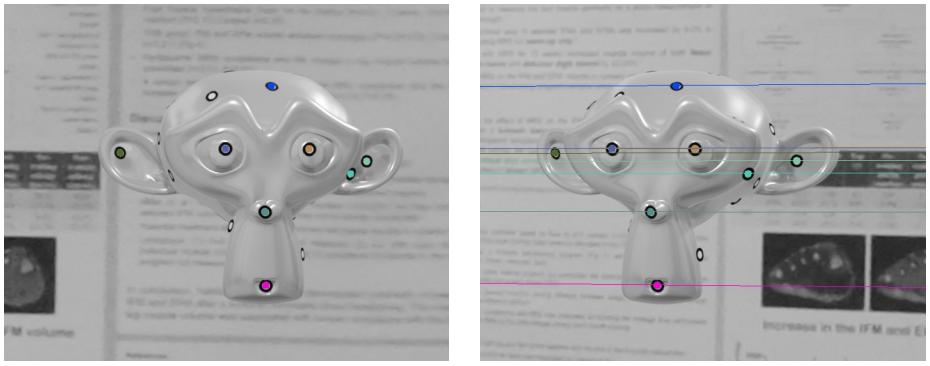
Gatrell, the creator of CCC markers, proposed another type of filter that consists of computing the centroid of the white circles and comparing them with the centroids of the black rings. This is an interesting approach since the possibility of a black and white blove sharing its centroid is very low if they are not part of the same marker. However, after several experiments, it was found that with the filters mentioned above it is not necessary to apply this latter one. By its own, a single blob detection takes the triple of time in comparison with ArUco markers detection, this type of filter would increase the detection time at least by two.

4.3 Point correlation

In order to triangulate the position of a marker in a stereo vision system, it is necessary first to identify it in both camera's images. This is not straightforward since there can be several markers visible in both pictures. Luckily, it is possible to constrain the set of possible combinations if we take into account the epipolar geometry of the stereo system.

As seen in Section 2.3.3, in a stereo vision system, the position of a point seen from the perspective of one camera, constrains immediately the position of the projection of that point on the other camera plane (Figure 4.3)

The proposed solution consists of computing the result of equation 2.7 for every combination of points from one image to the other. The result will be a matrix M_F, with size $n_x m$ with n and m the numbers of points in images one and two respectively. To compute this matrix it is necessary first to define P_{C1} and P_{C2} that are matrices containing in each row the coordinates of the detected markers for each camera frame of reference, equations 4.1 and 4.2.



a) b)

Figure 4.3: Due to the epipolar constraints, the lines in image (b) represent the possible locations of a detected marker from (a) in (b). The markers are identified with different colors that will be the same for the correspondent line.

$$P_{C1} = \begin{pmatrix} p_{x1} & p_{y1} \\ p_{x2} & p_{y2} \\ ... & ... \\ p_{xn} & p_{yn} \end{pmatrix}_{C1} \tag{4.1}$$

$$P_{C2} = \begin{pmatrix} p_{x1} & p_{y1} \\ p_{x2} & p_{y2} \\ ... & ... \\ p_{xm} & p_{ym} \end{pmatrix}_{C2} \tag{4.2}$$

Then the resulting matrix can be computed using the formula 4.3. If two points from different images, $p_{i\,C1}$ and $p_{j\,C2}$,represent the same marker in space, then they must satisfy expression 2.7, and a zero in row i and column j should be in place in matrix M_F.

$$M_F = P_{C1}^T \, F \, P_{C2} \tag{4.3}$$

However, as explained before, because of error due to measurement or calibration that number will hardly be exactly zero. This is why it is necessary to set a suitable threshold that will define if two points are correlated. If it is too high, a wrong correlation could be set that would result in triangulating an unexisting point in space. On the other hand, if it is too low, is possible that no correlation would be set. Obtaining $M_F^{|.|}$ as element-wise absolute value of M_F (Equation 4.4), the correlated points from each list will be i and j such that the element in row i and column j of matrix $M_F^{|.|}$ has a value less than the

threshold.

$$M_F^{|.|} = abs(M_F) = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1m} \\ m_{21} & m_{22} & \dots & m_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nm} \end{pmatrix} \tag{4.4}$$

There can exist situations in which the same point i from P_{C1} has more than one correlated points in P_{C2}, or vice versa. To avoid this situation, it is possible to set an additional restriction shown in expression 4.5.

$$\begin{gathered} (\boldsymbol{p}_{iC1}, \boldsymbol{p}_{jC2}) \text{ are correlated} \\ \Updownarrow \\ m_{ij} < th \ \wedge \ m_{ij} = min(Row_i(M_F^{|.|})) \ \wedge \ m_{ij} = min(Column_j(M_F^{|.|})) \end{gathered} \tag{4.5}$$

Condition 4.5 simply expresses that for p_{iC1} and p_{jC2} to be correlated points the value of matrix $M_F^{|.|}$ in row i and column j must be:

1. **Lower than the predefined threshold (th)**
2. **Equal to the minimum value of row i in matrix $M_F^{|.|}$**
3. **Equal to the minimum value of column j in matrix $M_F^{|.|}$**

Condition 1) was already explained, while conditions 2) and 3) do not permit a point from an image to have more than one correlation with other points from the other image, because the correlation will be only set with the point that has the minimum value of m.

What it is expected from this process is a set of correlated points between image one and two. In case that a marker is visible in one image but in the other not, it would not have a correlation with any other point.

With this information, it is possible to triangulate the position in space of each marker, as seen in Section 2.4, in the same way, that was done with the corners of the ArUco markers. In this stage, other filters can be applied, for example, to set a lower and upper limit from the different axis. If it is known that the detected markers should be contained in a certain range of X, Y, and Z, the triangulated points that do not belong to this volume will be discarded.

4.4 Markers mapping

Since this type of markers have not an internal codification, the identification of them is set by the distance that they have with the rest of the markers, this would be explained more in detail in Section 4.4.1. But in order to do this, it is necessary to know a priori the relationship between markers. It is imperative then, to build a 3D model that defines the relative location of them.

With this aim, a method for generating a 3d map was proposed. The main idea consists of triangulating the position of the markers for various and sequential poses of a sample. Every partial mapped section can be aligned with the next one and so on. If a sufficient number of poses are captured the entire description of the position of the markers on a sample can be set.

In figure 4.4 it can be seen an example of an object in three sequential poses. The correlated markers in the first and second columns are represented with the same color. The third column shows the triangulation of the position of the markers in space.

4.4.1 ICP algorithm

In order to sequentially align the different clouds of points, the ICP algorithm seen in section Section 2.5.3 would be used. Since the position of the markers for pose $i+1$ change very little with respect to pose i, it is straightforward to correlate points that are close to each other from one cloud to another.

If a correct correlation is set, the alignment process is direct since the rotation matrix and the translation vector between one cloud and another can be computed with the closed-form expression seen in Section 2.5.2.

Let's define Ti as the homogeneous transformation matrix between Pose i and $i+1$. In order to align our data, the following transformation has to be made to every cloud of point C_i (CoP):

$$C_i' = T_{N-1} \dots T_{i+1}\, T_i\, C_i = (\prod_{j=N-1}^{i} T_j) C_i \qquad \text{for } i = 1,2,...,N-1 \tag{4.6}$$

The last cloud is the only one not to be transformed, all others would be aligned with respect to this last one. If all the aligned clouds are combined into a new one, the map will have many repeated points, this is because for alignment two consecutive clouds of points must share at least three points in common. However, since the alignment process

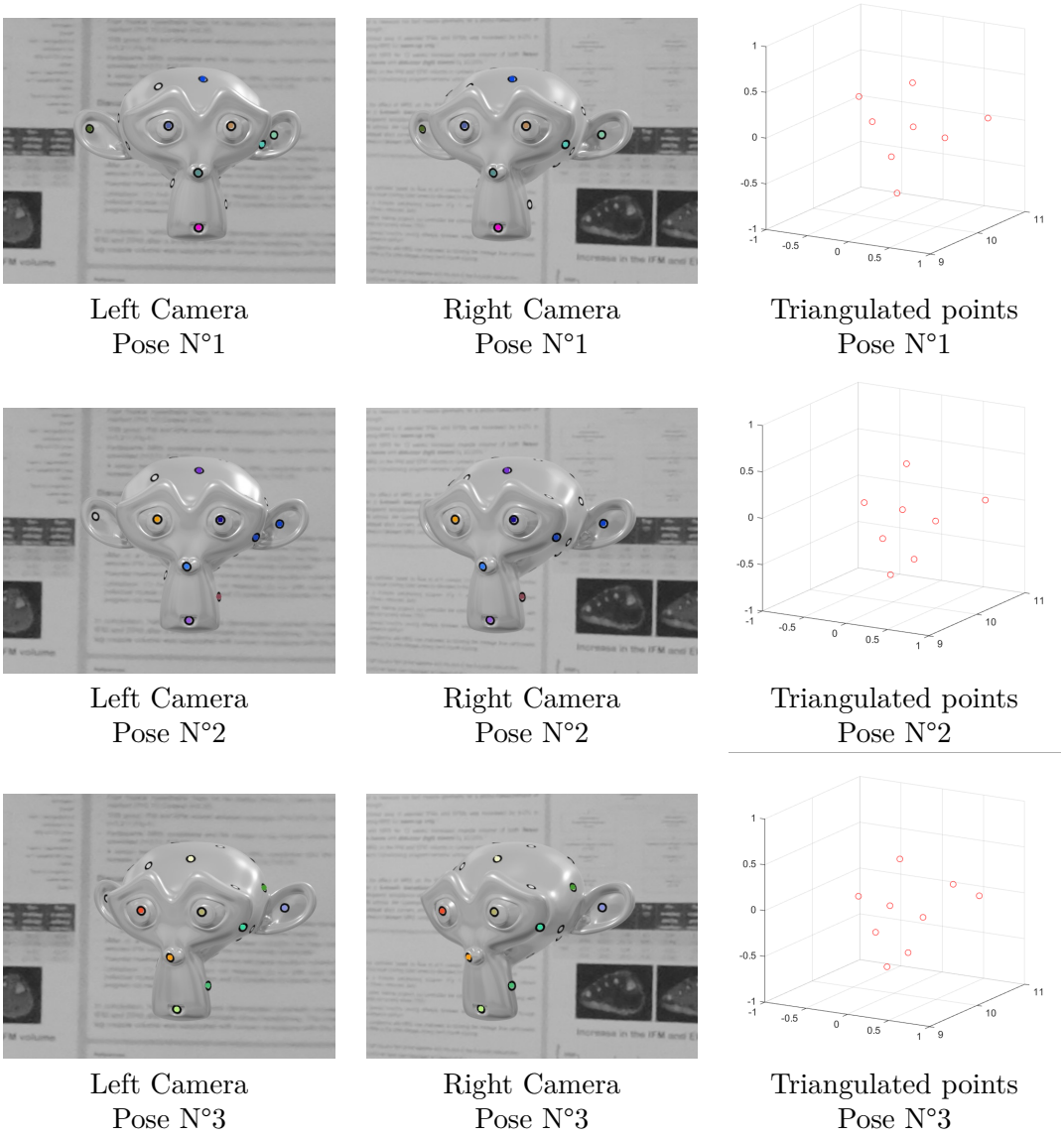


Figure 4.4: Triangulation for consecutive instances

requires first to correlate points, it is possible to keep track of the elements that represent the same marker.

The CoP for every pose is simply a list containing the coordinates of every point in the cloud. The id of a point in each cloud is determined by its index position on the list. Correlating two CoP consist simply in determine the indices between both clouds that represent the same points.

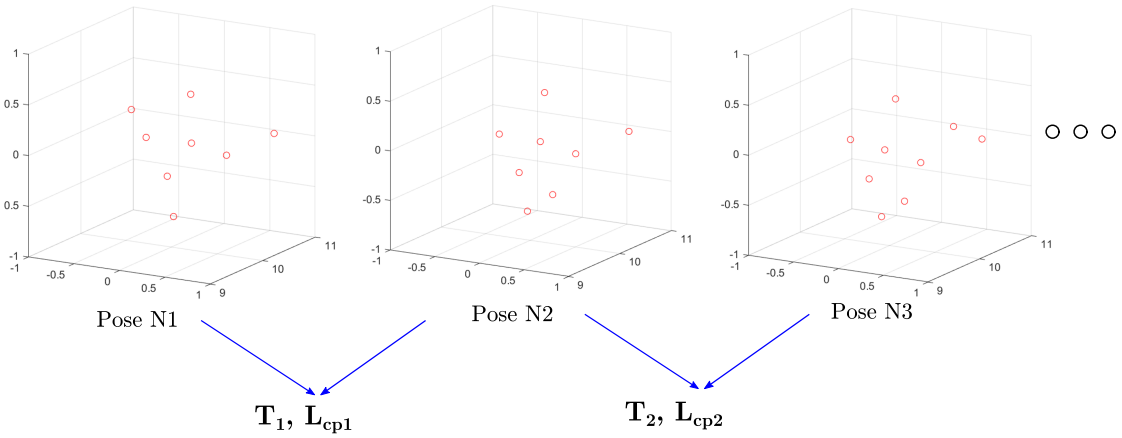


Figure 4.5: The transformation matrix T_i between two consecutive cloud of points is computed with the ICP algorithm. The process also returns as output L_{ci} that is the list of matched points from both CoP.

Figure 4.6 shows the *Matrix of correlation* of several consecutive CoP. Each row represents a different pose, while each column represents a unique marker. The numbers inside the matrix tell us the index of that point in a certain CoP. For example the CoP in poses 1 and 2 have all their points correlated, point 2 from Pose 1 is correlated with point 6 from Pose 2, and so on. Point P1 is present in all the CoP of poses from 1 to 14, and after alignment, the final position will be the result of averaging the coordinates in all these different poses.

All the points that represent the same marker are averaged to obtain a unique position. In spite of this, it is possible that during the scan process the same points appear in two different moments, if this happens it would be detected as a new one, since a sequential correlation of a marker finish when from one pose to the other a point is not detected anymore. An example of this is that points 9 and 10 from Figure 4.6. At pose 8, point 9 stop being detected and is detected again in pose 9. Despite P9 and P10 represent the same marker, they will be saved as different since their detection was no consecutive.

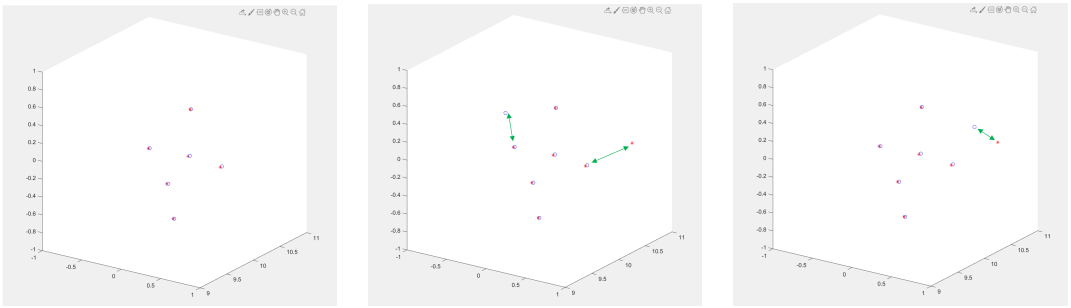
In order to avoid this, the distance between the remaining points is computed and the ones that are closer than a defined limit are considered the same point and then averaged.

4.4.2 Filters for correlation

As explained before, in order to align two clouds of points a true correlation between the elements of them has to be set. A desirable configuration should be similar to Figure 4.7 (a) (blue and red dots represent two consecutive CoP of pose i and i+1). In this setup, it is easy to make a correlation, since it is only needed to match the closest points.

Position / Point	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27
Pos 1	0	1	2	3	4	5	6																				
Pos 2	0	1	6	2	3	4	5																				
Pos 3	0	1	6	2	3	4	5																				
Pos 4	0	1	2	4	5	6	7	3																			
Pos 5	0	1	2	4	5	6	7	3																			
Pos 6	0	1	2		4	5	6	3																			
Pos 7	0	2	3		5	6	7	4	1																		
Pos 8	0	1	2		4	5	6	3																			
Pos 9	0	2	3		4	6	7	5		1																	
Pos 10	0	2	3		4	5	8	6		1	7																
Pos 11	0	2	3		4	5	8	6		1	7																
Pos 12	0	2	3		4	5	9	6		1	8	7	10														
Pos 13	0		2		3	4	8	5		1	7	6	9														
Pos 14	0		2		3	4	8	5		1	7	6	9														
Pos 15			1		2		6	3		0	5	4	7														
Pos 16			1		2		6	3		0	5	4	7														
Pos 17			1					4		0	5	3	2														
Pos 18			1					4		0	5	3	2														
Pos 19			1							0	4	3	2														
Pos 20			5							0	3	2	1	4													
Pos 21			5							0	2		1	4	3	6											
Pos 22										0	2		1	4	3	7	5	6									
Pos 23										0	3		1	5	4	7	6	2									
Pos 24										0			2	6	4	1	7	3	5								
Pos 25													2		4	1	6	3	5	0	7						
Pos 26													3		6	2	9	4	7	0	10	1	5	8			
Pos 27													2		5	1	8	4	6	0	3			7			
Pos 28													10		6	1	9	3	7	0	2			8	4	5	
Pos 29															5	1	9	3	7	0	2			8	4		6

Figure 4.6: Matrix of correlation for every pose



(a) (b) (c)

Figure 4.7: Different case scenarios

However, it is harder to do so in situations like in Figure 4.7 (b), where from instance i to $i+1$ a new point is detected and another is not detected anymore. The new red point will be correlated by closeness with a blue dot that is already correlated, and the same will happen with the blue dot that, from one instance to another, stops being detected.

A successful policy to overcome this problem is to set a correlations only if it is mutual. This condition can be expressed formally in the following way: point p_i^{C1} from cloud C_1 and point p_j^{C2} from C_2 would be correlated only if the closest point from p_i^{C1} belonging to C_2 is p_j^{C2}, and the closest point from p_j^{C2} belonging to C_1 is p_i^{C1}.

As an example, in figure 4.8 the closest blue point to r_2 is b_1 however, the closest red point to b_1 is r_1 not r_2, so there is no correlation between b_1 and r_2. Because of this, r_2

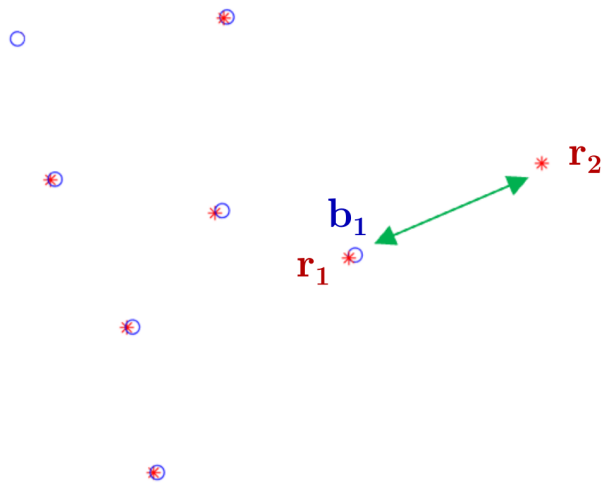


Figure 4.8: Multiple correlation case

will not be taken into account for the computation of the rotation matrix and translation vectors between the blue and red cloud.

Another situation that can occur is that two points from different clouds are mutually correlated but they do not represent the same marker, Figure 4.7 (c). If these points are considered for the computation of the rotation and translation parameters, there would be a significant increase in the error defined in equation Equation 2.15. If this error is higher than a predefined threshold, a good policy that can be applied is to recompute the optimal translation without the pair of points that produce the highest error. The reason for this is the following: If the two clouds represent consecutive poses, and the initial correlation is between the closest points, it is expected that the majority of them will be well correlated, so the pair of points that after an alignment produces the higher error probably do not represent the same marker.

An alternative approach to correlate points more robustly is described in the next section. It basically consists of matching points from different clouds, that share similar relationships with the rest of the elements of the same set. However, it requires in general more processing time, and it increases quadratically in relationship with the number of elements in a cloud.

Once we have the complete 3D map of the location of the CCC markers on sample, we need to defined where the frame of reference (FR) of the object will be located. For this purpose, it is necessary to transform the 3D map accordingly. It must be done in such a way that the chosen FoR coincides with the world FR. This step is necessary since the pose of the object in space will be determined based on its zero pose. In Figure 4.10 the FoR of the object has x, y and z coordinates equal to zero, while all of its Euler angles

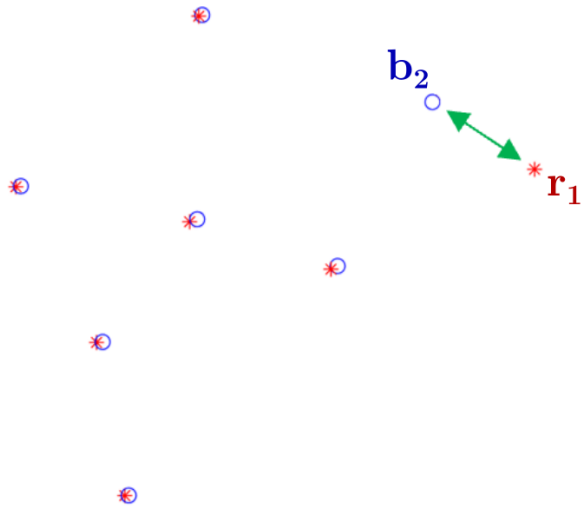


Figure 4.9: Correlation between points representing different markers

are also zero.

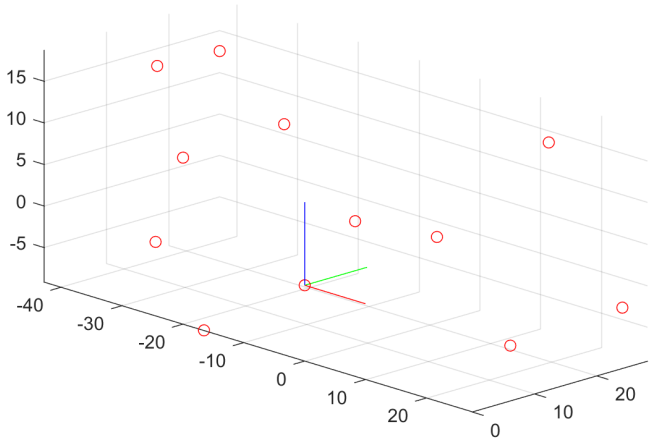


Figure 4.10: The 3D relative position between CCC markers at the end-effector of the robot. The frame of reference was chosen arbitrary, and the CoP was aligned in such a way that the object RF coincides with the world RF.

4.5 Pose estimation

As mentioned before, estimating the position of an object with CCC markers consists simply of applying the procedure described in Section 2.5.2, between the detected points of a sample in a certain pose and the 3D map of the sample's markers. But in order to apply this procedure successfully, it is necessary first to find a correct matching between the two clouds while identifying its outliers.

Several attempts were made on adapting the ICP algorithm for this particular case, however almost every time the matching solution was suboptimal. The main cause of it is the difference in the number of elements in each cloud. While the 3D map contains the position of every marker, a cloud of points for a particular pose contains only a fraction of them.

As expressed in [37], adapting the ICP algorithm to match the detected points with the ones from the 3D maps, implies solving a global optimization problem. These type of problems are in general non-convex, requiring complex algorithms to obtain the optimal solution. Even a simple one-dimension ICP problem as the one in Figure 4.11 turns out to be non-convex.

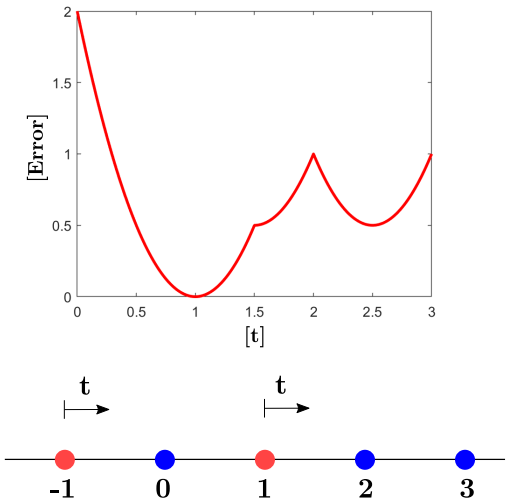


Figure 4.11: Example of a non-convex 1D ICP matching problem based on [37]: The blue and red dots represent different CoP. The red CoP can be moved a distance t from its original position. The optimization problem consists in finding the distance t that minimizes the loss function. The loss function is equal to the sum of the square distances between points from different clouds matched by closeness.

Because of this, it was decided to developed a more simple matching technique. The points from different CoP will be matched if they share similar distances with other

points in the same cloud. This method takes advantage of the fact that a same length between two points in different poses change very little because of the precision of the measuring system. With such a required precision, it is improbable that two distances between different pairs of points are equal. Also, since we are dealing with a relatively low amount of markers (less than 100), computing the distance of a point with every other in its cloud, is not a demanding task.

Let us define a "Distance Matrix" as the one containing the distance between every possible pair of points in a cloud. The dimension of it will be NxN with N equal to the number of elements of the CoP. The value of this matrix in a generic position (i,j), will be the distance of point i with respect to point j. Since one of our hypotheses is that we are working with rigid CoP, no matter the orientation, this value must remain constant.

This matrix is symmetric since the distance from i to j is the same that from j to i. It's main diagonal has all its elements equal to zero, since the distance between a point with itself is null.

The correlation technique consists of searching for elements in common between the distance matrix from the 3D map (D_{map}), and the one from the detected cloud of points (D_{det}). The Python code is shown in Listing 4.1.

```
1  def corr_3d( map_3d, det_points, th=0.0005, n_th1=3, n_th2=3):
2      # Number of points from 3D map and Detected CoP
3      l_det = len(det_points)
4      l_map = len(map_3d)
5
6      # Compute distance matrix from 3D map, and Detected CoP
7      D_det = dist_matrix(det_points,det_points)
8      D_map = dist_matrix(map_3d,map_3d)
9
10     # Create "Similarity Matrix", which will contains information of
           regarding equal lengths between D_det and D_map
11     SM = np.zeros(( l_det , l_det , l_map ))
12
13     # Iterate for every row in D_det
14     for i in range(l_det):
15         # ran is a list containing all integers from 0 to l_det the i
               value is extracted from the list since it is the same point
16
17         ran = list(range(l_det))
18         ran.remove(i)
19
20         # Iterate for every column of D_det (Except for column i)
21         for j in range(l_det-1):
```

```
22          # M_log is the "Logic matrix" that identify distance values
                from D_map that are almost equal to a distance between
                points i and ran[j] from D_det
23
24          M_log = np.abs(D_map - D_det[i,ran[j]])<th
25          SM[ran[j],i,:] = np.array(np.sum(M_log,0)>=1,dtype="bool")
                #M_map is append to SM
26
27      # Return list of correlated points that have more than n_th1 lengths
            in common
28      indices_det , indices_map = np.where(np.sum(SM,0)>=n_th1)
29
30      return indices_map ,indices_det
```

Listing 4.1: Python code for point correlation

The code in Listing 4.1 will be explained briefly in the following paragraphs. First, the function computes both distance matrices from the 3D map and the detected points, D_map respectively D_det. It then creates a three-dimension matrix with dimensions (l_det, l_det, l_map). Two consecutive *for* cycles goes to every element of D_det, those elements represent the lengths between point i and j that are the row and column index respectively. The only elements that these *for* cycles do not go through are the diagonal of D_det, since all its elements are zero because they represent the distance of a point with itself.

On line 24 from Listing 4.1, the value (i,j) from D_det is compared with all the elements of D_map. A matrix M_log is created with the same dimension of D_map. In each element of this matrix, there would be a 0 or a 1 depending if the value at the same position in D_map differs in less than th with respect to point (i,j) in D_det.

On line 25 the values from each column of M_log are summed. If the sum of one column is equal or greater than one, it would mean that the point in that column index has at least one connection with another point almost equal to the value (i,j) from D_det. A boolean list is saved in position (i,j) in SM.

Finally, in line 28 the indices of the correspondent points from the 3D map and the Detected points CoP are obtained. The points are correlated if they share at least a number of n_th1 similar lengths with the other points of their cloud.

This approach would not work if the error in measurements is large since two similar lengths could be confused. Nonetheless, the measurements for the task at hand need an accuracy of 50 μm, which makes improbable to find two equal distances with that precision.

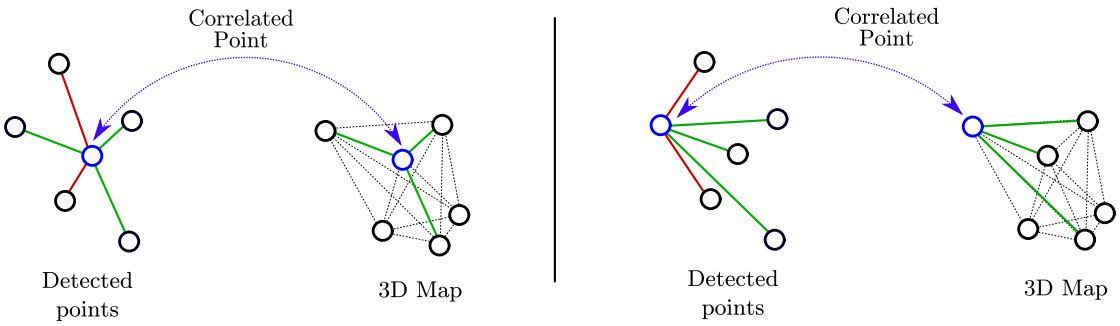
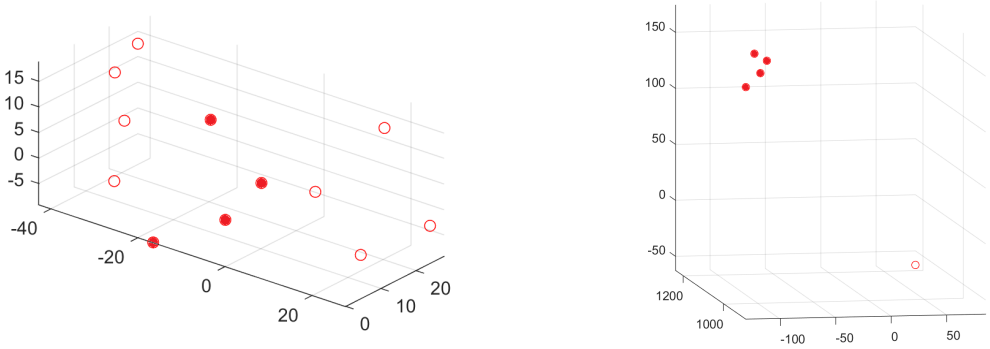


Figure 4.12: Two points of the different CoP are correlated if they share at least three lengths in common. Green lines represent lengths found in both CoP, while red lines represent lengths not found in the other cloud.

Example

For a more clear understanding, an example of how the algorithm works is shown in this section. In Figure 4.13 (a) is plotted the location of the point of the overall 3D map, while in Figure 4.13 (b) the detected points from a specific position. The latter cloud was obtaining triangulating the detected points from Figure 4.14. The red solid points in both images from Figure 4.13 represent the ground truth points in common between both clouds.



(a) (b)

Figure 4.13: Common points are plotted as solid red dots. a) 3D map CoP of the relative position of all the markers on the end-effector, b) Detected points CoP, the isolated element in the corner is an incorrect measurement.

In order to find a correlation between the two clouds of points, the Distance Matrices for the 3D map D_{map} and the detected points D_{det} are computed. They are shown in Figure 4.15 (a) and (b) respectively.

The algorithm starts from column A of D_{det}, and looks in D_{map} similar distance values. Since there is no similar distance, point A will not be correlated.

The next four points (B,C,D and E) will be correlated. Figure 4.15 (b) shows in orange the cells from D_{map} that have similar values of the element from column C in D_{det}. Column E in D_{map} has in total 3 different matches with column C in D_{det}, no other column have that quantity of points in common. This is the reason why point C from D_{det} will be correlated with point E in D_{det}. A similar plot is shown in Figure 4.15 (c) with respect to column E in D_{det}.

(a) (b)

Figure 4.14: Images of the robot at a certain pose taken from both cameras in the SVS.

	A	B	C	D	E
1	0.00	455.87	460.92	456.57	468.89
2	460.92	0.00	18.93	12.60	18.84
3	455.87	18.93	0.00	31.06	30.62
4	456.57	12.60	31.06	0.00	14.77
5	468.89	18.84	30.62	14.77	0.00

(a)

	A	B	C	D	E	F	G	H	I	J	K	L
1	0.00	13.38	30.94	20.97	15.08	29.69	39.59	33.55	50.94	47.08	60.08	57.75
2	13.38	0.00	17.57	17.99	27.13	38.22	45.98	37.64	56.89	52.69	63.64	60.01
3	30.94	17.57	0.00	27.18	44.11	52.74	58.37	48.79	68.07	63.89	71.99	67.32
4	20.97	17.99	27.18	0.00	31.11	35.36	39.27	26.68	56.44	47.31	63.58	53.98
5	15.08	27.13	44.11	31.11	0.00	18.93	31.08	30.71	39.78	37.88	50.83	51.31
6	29.69	38.22	52.74	35.36	18.93	0.00	12.62	18.92	25.00	19.29	36.64	34.10
7	39.59	45.98	58.37	39.27	31.08	12.62	0.00	14.77	23.19	9.79	32.82	24.08
8	33.55	37.64	48.79	26.68	30.71	18.92	14.77	0.00	36.37	23.00	44.19	31.30
9	50.94	56.89	68.07	56.44	39.78	25.00	23.19	36.37	0.00	16.24	13.74	24.49
10	47.08	52.69	63.89	47.31	37.88	19.29	9.79	23.00	16.24	0.00	24.20	16.81
11	60.08	63.64	71.99	63.58	50.83	36.64	32.82	44.19	13.74	24.20	0.00	22.28
12	57.75	60.01	67.32	53.98	51.31	34.10	24.08	31.30	24.49	16.81	22.28	0.00

(b)

	A	B	C	D	E	F	G	H	I	J	K	L
1	0.00	13.38	30.94	20.97	15.08	29.69	39.59	33.55	50.94	47.08	60.08	57.75
2	13.38	0.00	17.57	17.99	27.13	38.22	45.98	37.64	56.89	52.69	63.64	60.01
3	30.94	17.57	0.00	27.18	44.11	52.74	58.37	48.79	68.07	63.89	71.99	67.32
4	20.97	17.99	27.18	0.00	31.11	35.36	39.27	26.68	56.44	47.31	63.58	53.98
5	15.08	27.13	44.11	31.11	0.00	18.93	31.08	30.71	39.78	37.88	50.83	51.31
6	29.69	38.22	52.74	35.36	18.93	0.00	12.62	18.92	25.00	19.29	36.64	34.10
7	39.59	45.98	58.37	39.27	31.08	12.62	0.00	14.77	23.19	9.79	32.82	24.08
8	33.55	37.64	48.79	26.68	30.71	18.92	14.77	0.00	36.37	23.00	44.19	31.30
9	50.94	56.89	68.07	56.44	39.78	25.00	23.19	36.37	0.00	16.24	13.74	24.49
10	47.08	52.69	63.89	47.31	37.88	19.29	9.79	23.00	16.24	0.00	24.20	16.81
11	60.08	63.64	71.99	63.58	50.83	36.64	32.82	44.19	13.74	24.20	0.00	22.28
12	57.75	60.01	67.32	53.98	51.31	34.10	24.08	31.30	24.49	16.81	22.28	0.00

(c)

Figure 4.15: Distance matrices: a)Detected points distance matrix D_det, b) 3D map distance matrix D_map colored in orange for similar values of column C in D_det, c) 3D map distance matrix D_map colored in gray for similar values of column E in D_det

Chapter 5

Validation method

5.1 Laboratory equipment at testing facility

The developing process of the new external measuring system was carried out at FAPS laboratory in Erlangen, Germany.

The cameras available for testing were two Jai GO-5000C-USB with 5 megapixels each. They were equipped with Kowa lenses with 16mm focal length. Both were linked to a desktop computer by USB connection. The images from the cameras were sent as ROS 2 messages to our software for processing.

While on STRESS-SPEC the robot used for sample positioning is a 6-axis Stäubli RX160, for testing a Stäubli TX2-60L was used. Since both models are from the same company and have the same number of axes, the motion control model for the robot under test can be easily adapted to the target one. The robot commands are sent from a desktop computer through ROS 1 interface.

A Leica laser tracker AT901 was also at disposal. This instrument allows taking external measurements for a later validation of the computer vision system. It was connected to an external computer with an ethernet cable.

A graphical representation of the hardware and its interface is plotted in Figure 5.1.

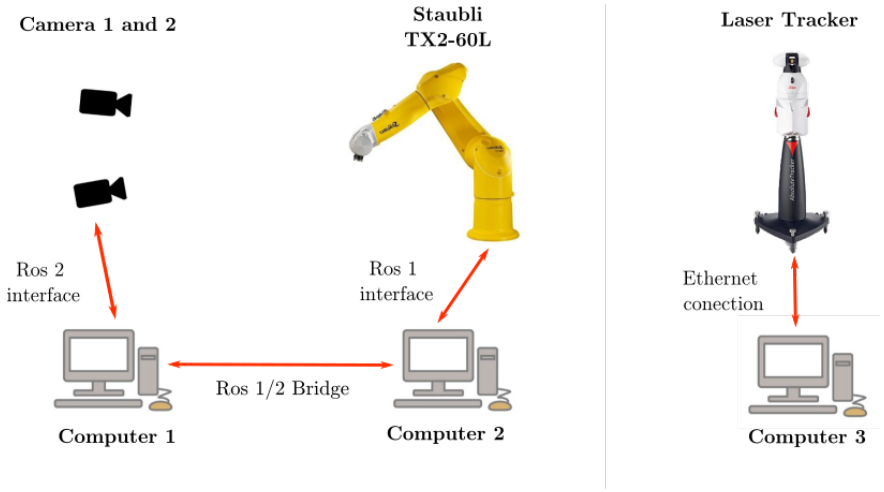


Figure 5.1: Available equipment at the testing Laboratory

5.2 Data alignment

As explained before, the measurements from the Computer Vision System (CVS) will be compared with the ones from the laser tracker, to estimate the accuracy of the former. These two sets of measurements cannot be compared directly mainly because of two reasons.

The first one is due to the fact that the zero frame of reference (FR) from the cameras is different from the one of the laser tracker, as a result, it is required to compute first the transformation between these two instruments.

The second obstacle is because the points being measured by each instrument are located in different positions in the end-effector, the cameras compute the pose of the markers while the laser tracker the pose of the reflectors (Figure 5.2). As a consequence, also the relative transformation between the frame of reference of the reflectors and the markers should be estimated (Figure 5.3).

The method for surpassing these obstacles is explained in the following sections.

5.3 Transformation estimation

In Figure 5.4 is schematically represented the different homogeneous transformations (HT) between the frame of references. $T^r_{l,i}$ represents the measurement i of the laser tracker, it can be interpreted as a transformation between the reflector and the laser

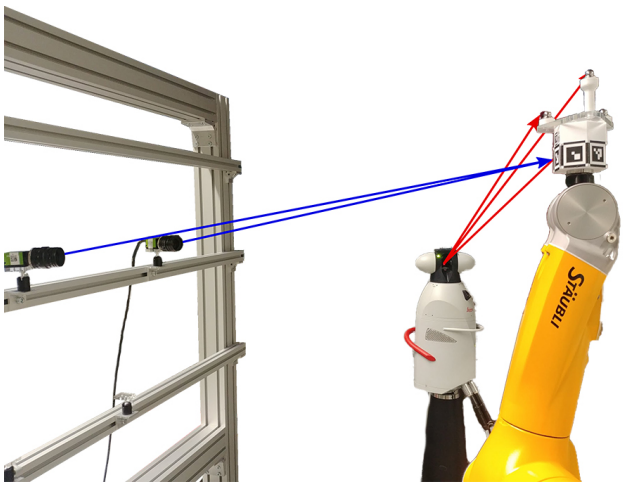


Figure 5.2: Photo of the real equipment, the blue arrows indicates the measure of the ArUco markers from the cameras, while the red arrows shows the measure from the laser tracker to the different reflectors.

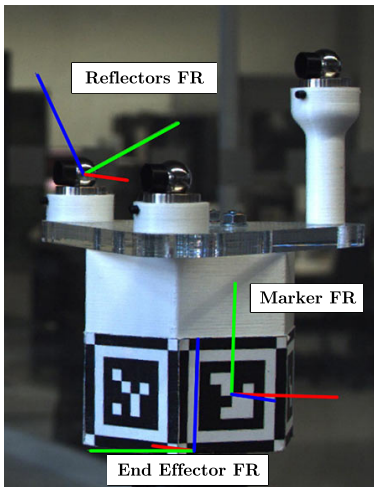


Figure 5.3: End-effector of the robot. At the top the FR of the reflectors can be seen, at the middle the one from the ArUco marker, and at the bottom the one from the robot

tracker frame of reference. In the same way, $T^m_{c,i}$ is the pose estimation i of the markers measured by the camera system. Both of this transformation varies for different poses of the end-effector.

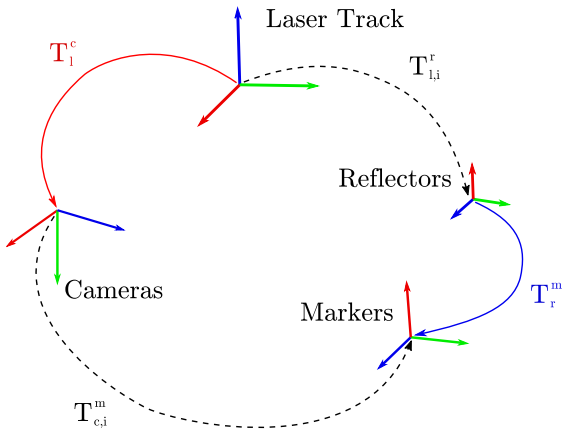


Figure 5.4: Chain of transformations

On the contrary, transformation T^c_l and T^m_r remain constant despite of the end-effector movement. The values of T^c_l stay unchanged since the cameras and the laser tracker are fixed. While for T^m_r, the reflectors and the markers are attached to the same rigid body (Figure 5.3), due to this fact, no matter the pose of the end-effector this transformation

would not change.

In the ideal case, Equation 5.1 should be valid for every pose of the end-effector. However, because of the error in measurements from both instruments, this expression does not hold.

$$T_l^c \, T_{c,i}^m \; = \; T_{l,i}^r \, T_r^m \quad \text{for } i = 1,2,...,N \tag{5.1}$$

The translation error for every measurement can be expressed also through an homogeneous transformation matrix (HTM) Te_i as shown in Figure 5.5 and computed with expression 5.2.

$$T_{ei} = \; (T_l^c \, T_{c,i}^m)^{-1} \, T_{l,i}^r \, T_r^m \tag{5.2}$$

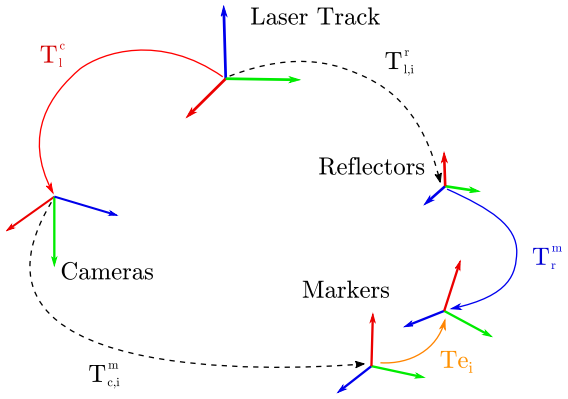


Figure 5.5: Chain of transformations with a difference $T_{e,i}$ in the final position of the markers. This is due to errors in calibration and measurements.

An estimation of T_l^c and T_r^m can be obtained by solving the optimization problem expressed in Equation 5.3. First the rotation and translation parameters of each one is computed forming the vectors p_1 and p_2. With this parameters, T_l^c and T_r^m are calculated with Equation 5.5.

$$p_1, p_2 = \underset{p_1,p_2}{\operatorname{argmin}} \sum_{i=1}^{N} (\; \alpha \; || \, E_i^{or}(p_1, p_2) \, ||_2^2 \; + \; \beta \, || \, E_i^{trans}(p_1, p_2) \, ||_2^2 \;) \tag{5.3}$$

α and β are coefficients that balance the weight of the orientation and translation error. E_i^{or} is the orientation error for measurement i, while E_i^{trans} is the translation error also for measurement i, both of them are vectors. The first one with 4 elements that represent the error in each quaternion component. While the latter one has 3 elements,

for the error in each of its coordinates (X, Y, and Z). Its value is computed according to Equation 5.4.

$$\begin{aligned} E_i^{or}(p_1, p_2) &= quat(\ T_l^c(p_1)\, T_{c,i}^m)\ -\ quat(T_{l,i}^r\, T_r^m(p_2)) \\ E_i^{trans}(p_1, p_2) &= trans(\ T_l^c(p_1)\, T_{c,i}^m)\ -\ trans(T_{l,i}^r\, T_r^m(p_2)) \end{aligned} \tag{5.4}$$

The function *quat* extracts from an HTM the rotation matrix and transforms it into quaternion form. While function *trans* extracts the translation parameters in every coordinate.

As mentioned before, p_1 and p_2 are vectors containing the rotation and translation parameters of T_l^p and T_r^m respectively (Equation 5.6). Their first four elements express the rotation in quaternions, while the last three are the translation for each axis X,Y and Z in millimeters.

$$T_b^a(p) = \left(\begin{array}{ccc|c} & R(q_1,q_2,q_3,q_4) & & \begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{5.5}$$

With:

$$p = (q_1, q_2, q_3, q_4, x_1, x_2, x_3) \tag{5.6}$$

And:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

The Matlab Code for computing the *Alignment Matrices* T_l^c and T_r^m can be found in Appendix D.

5.4 Error metrics

After obtaining T_l^c and T_r^m, it is possible to compute the difference in measurement between the cameras and the laser tracker. (Equation 5.7)

$$\begin{aligned} (\, e_{Rxi}\,,\, e_{Ryi}\,,\, e_{Rzi}\,) &= eul(\, T_l^c\, T_{c,i}^m\,) - eul(\, T_{l,i}^r\, T_r^m\,) \\ (\, e_{xi}\,,\, e_{yi}\,,\, e_{zi}\,) &= trans(\, T_l^c\, T_{c,i}^m\,) - trans(\, T_{l,i}^r\, T_r^m\,) \end{aligned} \tag{5.7}$$

From the different measured HTM the rotation matrix is extracted and transformed into Euler angles. The error value is the difference in each Euler angle from the measurement from the laser track and the cameras. If the error in some axis is larger than π, to that value is subtracted or added π n times till the error is in the range $[-\pi,\pi]$.

It is necessary only to compare the translation parameters inside the HTM from the cameras and the laser tracker to obtain the translation error in each coordinate.

The probabilistic distribution of each of these error components is assumed to be normal, so for each of them, the standard deviation will be computed. The mean is expected to be near zero since the data points were aligned in such a way that the total error was reduced to its minimum expression.

The required accuracy in position was set to $\pm$ 50 μm, while for orientation $\pm$ 0.5 degrees. The parameter that would be used to compare those values would be $\pm$ 3 standard deviations on the error for each axis, both for orientation and location.

The other type of error to consider, is the distance error e_{di} that is computed with Equation 5.8.

$$e_{di} = \sqrt{e_{xi}^2 + e_{yi}^2 + e_{zi}^2} \tag{5.8}$$

e_d is by definition always positive, so it would not be correct to model it as normally distributed. Nonetheless, e_d^2 is the sum of three normal distribution functions squared. Because of this, e_d^2 could be modeled as a Chi-squared distribution, that by definition represents the squared sum of n normal distributed variables with zero mean, and the same variance.

Since it is expected that the variance of e_z would be larger than e_x and e_y, it was decided that e_d^2 would be modeled as a Gamma distribution that is a generalization of the Chi-squared distribution. This type of distribution can be tuned with 2 continuous parameters (k and θ), in contrast with the Chi-squared distribution that has only one which has to be an integer (k). If e_d^2 follows a Gamma distribution, e_d can be modeled

as a Nakagami distribution. Every Nakagami distribution can be expressed as the root squared of a Gamma distribution.

Fitting the Nagakami distribution with the measured values of e_d would make it possible to compute the following metrics:

- $\boldsymbol{e_{50\%}}$ **:** Value of e_d for which the fitted cumulative distribution function is equal to 50 %.
- $\boldsymbol{e_{90\%}}$ **:** Value of e_d for which the fitted cumulative distribution function is equal to 90 %.

Appart from this two, also the RMSE of e_d would be computed as seen in Equation 3.1.

5.5 Pose compensation

As seen in Section 1.3.1, the stereo vision system acts as the main sensor in an external control loop that compensates the pose of the robot. This section intends to explain how the compensation process was tested.

In Figure 5.6 is schematized the flow of information through the different devices in the test set-up. First, Computer 1 receives the images from both cameras, they are processed and the actual pose of the robot is computed. It is required for pose estimation, to know a priori the calibration parameters of the stereo vision system.

Computer 2 publishes a message through a ROS topic and Computer 1 subscribes this message. The connection between computers is made through a ROS 1/ ROS 2 bridge since the code in them works with a different version of ROS. When computer 1 receives the message, it sends the actual pose T_c^m and the desired pose $T_c^{m'}$ to Computer 2 as a string. Both of these poses are measured with respect to the camera reference frame.

Computer 2 also receives the actual pose of the robot T_r^e and computes the new comanded pose $T_r^{e'}$. Computer 2 uses the information regarding the transformation between the robot FR and the camera FR T_r^c computed as in Section 5.3. Also the transformation between the end effector and the markers FR T_e^m is available. On Figure 5.7 is schematically represented all the transformations.

The new pose of the robot $T_r^{e'}$ is computed through Equation 5.9. It consists simply in computing the transformation difference T_d between the actual pose and the desired one. This transformation is obtained through the camera measurements and is used to

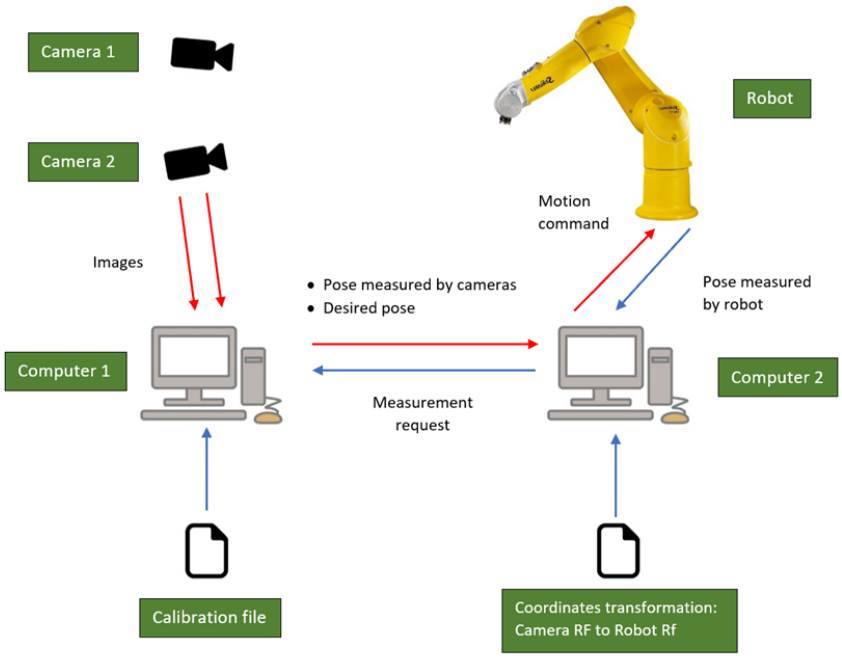


Figure 5.6: Flow of information through device's interface

calculate the new pose for the robot.

$$T_r^{e'} = T_r^e \, T_e^m \, T_d \, (T_e^m)^{-1}$$

$$with: \tag{5.9}$$

$$T_d = (T_c^m)^{-1} \, T_c^{m'}$$

This is an iterative process that would be repeated till the error stops decreasing. The results of the compensation test will be shown in the next chapter. absolute

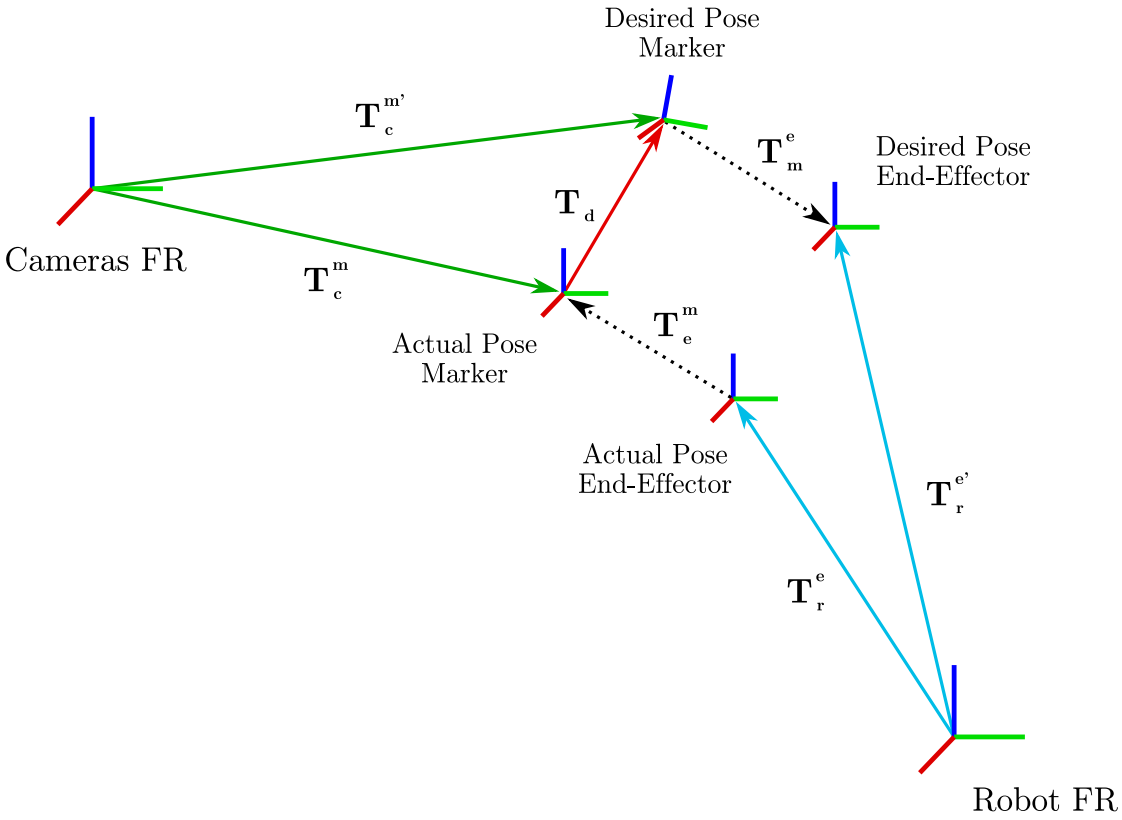


Figure 5.7: Set of transformations from cameras FR to robot FR.

Chapter 6

Test and results

In this chapter, it will be presented the results obtained from the different experiments. It is divided into three sections: The first one shows the results regarding absolute accuracy, the second one the repeatability of the measurement system, while the third section focuses on the error after compensation.

6.1 Absolute Accuracy

To compute the absolute accuracy a set of 27 different poses were chosen to be measured (Figure 6.1). The position of these points in space form a cube of 250 mm on each side, it would be called *measured volume*. The center of the cube is 775 mm apart from the camera's baseline.

The orientation of the measured points varies ± 10 degrees with respect to each other. This range was chosen in such a way that the laser tracker could still measure the position of each reflector without the need of rotating them.

It is important to highlight that not only the position of the end-effector was computed but also the orientation. The laser tracker measured the pose of the reflectors RF, while the cameras compute the pose of the markers RF. In spite of this, the transformation matrices T_l^c and T_r^m would be computed in order to relate both measurements, as seen in Chapter 5. The poses will be computed from the laser tracker RF to the markers RF, the pertinent transformations in each case are made.

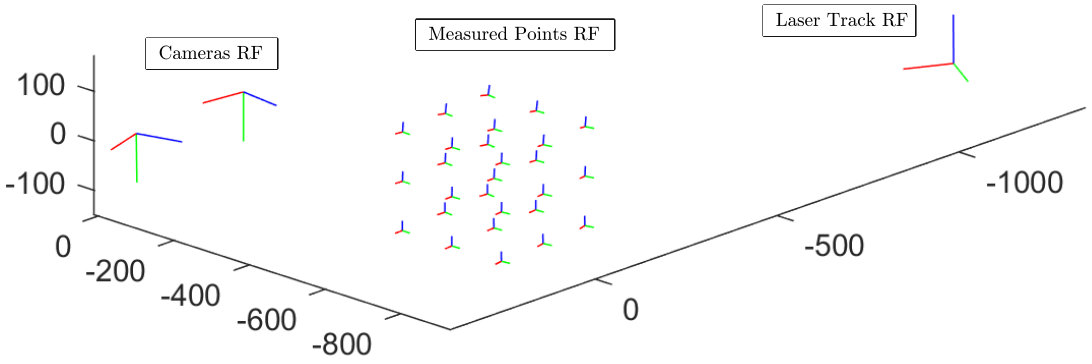


Figure 6.1: Relative position between cameras, laser tracker, and measured points

6.1.1 Method comparison

This section has the objective of comparing the absolute accuracy of CCC markers and Enhanced ArUco Marker (ArUcoE). In the case of the latter, it was also compared the measurements with one single camera, with Super-Resolution (SR) corner detection, and the standard Subpixel corner detection algorithm.

All of these measurements were carried out using the same set-up. The cameras were located 300 mm apart from each other, and its baseline at a distance of 775 mm from the center of the *measured volume*. Moreover, the calibration parameters for every method were the same, both intrinsic and extrinsic.

The matrices T_l^c and T_r^m were recomputed for each method using the 27 data points.

On Table 6.1 it is shown the standard deviation of the error in the six degrees of freedom (DoF). The measurements from the laser tracker are compared to each method, and the error in each DoF is computed for the different poses.

Method	$\boldsymbol{\sigma_x}$ (μm)	$\boldsymbol{\sigma_y}$ (μm)	$\boldsymbol{\sigma_z}$ (μm)	$\boldsymbol{\sigma_{Rx}}$ (deg)	$\boldsymbol{\sigma_{Ry}}$ (deg)	$\boldsymbol{\sigma_{Rz}}$ (deg)
ArUcoE Single Image	713.4	423.5	351.2	0.85	0.74	0.42
ArUcoE Stereo SR	143.7	141.5	206.4	0.16	0.14	0.18
ArUcoE Stereo Sub-pixel	132.1	141.4	197.7	0.13	0.14	0.14
CCC	161.6	122.1	178.1	0.07	0.08	0.12

Table 6.1: Standard deviation of the error in X, Y and Z and in the rotation angles RX, RY and RZ for the different methods

As expected, the worst performance was with the Single Image method. The errors for this method were considerably larger than the other three.

On Table 6.2 different parameters of the distance error e_d, computed as in Equation 5.8,

are shown. As explained in Chapter 5 a Nagakami function will be used to approximate the probabilistic density function (PDF) and the cumulative density function (CDF) of the error. After fitting the probabilistic model, the values of e_d 50% and e_d 90% seen in Section 5.4 can be computed.

Method	***RMSE*** e_d (μm)	e_d **50%** (μm)	e_d **90%** $(\boldsymbol{\mu m})$	e_d ***max*** (μm)
ArUcoE Single Image	884.0	794.5	1260.6	1778.2
ArUcoE Stereo SR	283.2	256.3	400.9	473.1
ArUcoE Stereo Subpixel	271.5	248.9	378.2	464.2
CCC	264.7	226.6	394.1	486.0

Table 6.2: Parameters of e_d error

It can be seen that the standard subpixel algorithm is superior to the Super-Resolution method in any metric of Table 6.2. Moreover, considering the fact that the SR method takes 5 times more processing time than the standard one (76 seconds vs 15 seconds for the 27 pairs of images), we conclude that the SR is not worth considering.

The CCC method surpasses the standard ArUcoE method in RMSE and e_d50%. On the other hand, the latter method is superior taking into account e_d90% and e_dmax metrics. It is not possible to determine in this case which method is superior.

The fitting of the Nagakami functions for e_d is shown in Figure 6.2. The scale for all stereo vision methods goes from 0 to 0.6 mm. While for the single image method the range is larger. On the other hand, on Figure 6.3 is shown the fitted CDF in contrast with the real data. The ranges are the same of Figure 6.2.

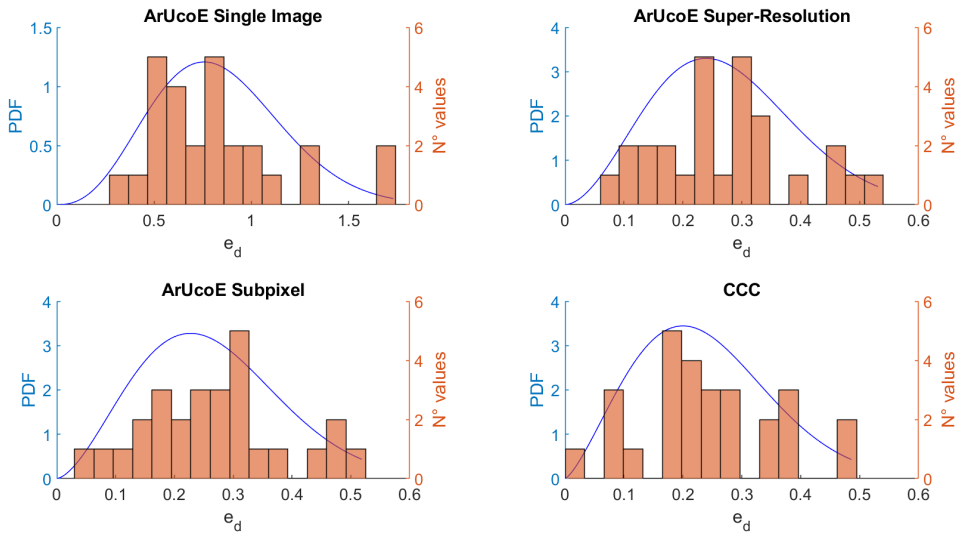


Figure 6.2: e_d histogram with its respective fitting Nagakami curve for the different methods

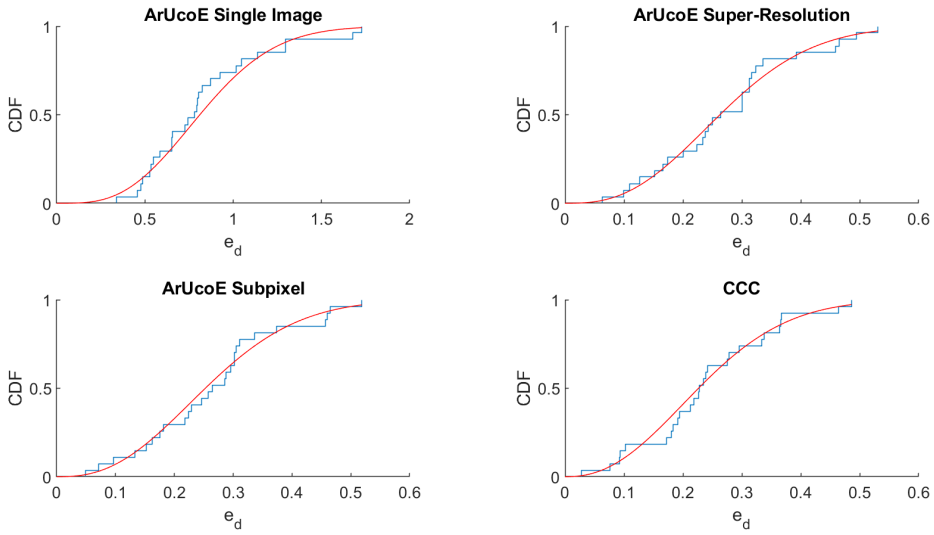


Figure 6.3: Fitting of CDF curve for e_d in every method

The PDF and CDF curves from the previous figures are plotted together on Figure 6.4. This figure makes easier the comparison between methods. Clearly, the Stereo Vision methods are superior to the single image one. Moreover, there is a little difference between the ArUcoE subpixel and the CCC method, the curve from the latter is a bit closer to zero.

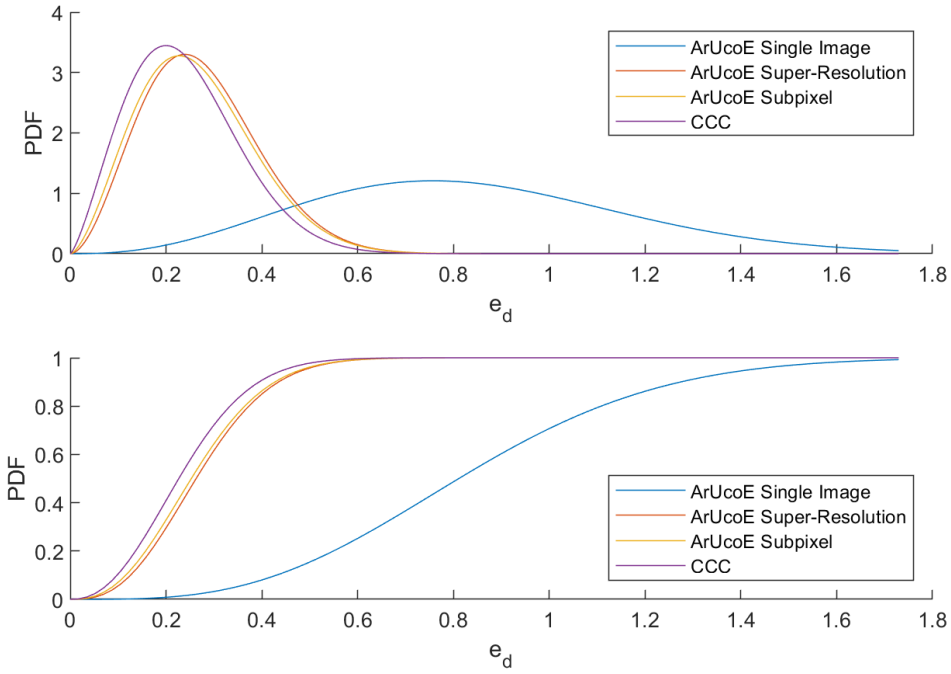


Figure 6.4: Estimated PDF and CDF curves for the different methods in an unique plot

6.1.2 Baselines comparison

This section intends to test if an increase in the baseline length truly reduced the absolute error. According to Equation 2.13 the optimal baseline distance is approximately 998 mm for an object located 775mm apart from the cameras.

In the previous section for all the experiments, the baseline was 300 mm. For practicality, instead of the optimal value, for this experiment, it was chosen a baseline of 800 mm. The expected variation for this value is very close to the optimal. In Figure 6.5 it is plotted the expected RMSE in function of x defined in Appendix B as $x = \frac{Baseline}{2 * Object\,distance\,to\,cameras}$.

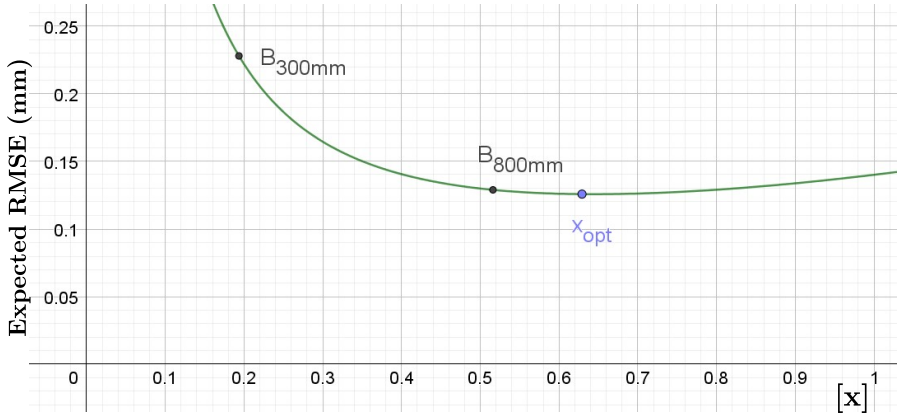


Figure 6.5: Expected RMSE in function of $x = \frac{B}{2\,z_c}$

The expected RMSE according to formula Equation 2.12 for a baseline of 300 mm is 228 μm, while for a baseline of 800 mm it is expected 129 μm. This implies a reduction of 43 % in RMSE.

On tables Table 6.3 and Table 6.4 it is shown a comparison between the error in pose estimation of CCC markers for two different baselines values. The RMSE for a baseline of 300 mm was 265 μm, and 180 μm for a baseline of 800 μm. The reduction on the RMSE, in this case, was 32 %.

Method	$\boldsymbol{\sigma_x}$ (μm)	$\boldsymbol{\sigma_y}$ (μm)	$\boldsymbol{\sigma_z}$ (μm)	$\boldsymbol{\sigma_{Rx}}$ (deg)	$\boldsymbol{\sigma_{Ry}}$ (deg)	$\boldsymbol{\sigma_{Rz}}$ (deg)
CCC (b=300mm)	161.6	122.1	178.1	0.07	0.08	0.12
CCC (b=800mm)	51.6	141.6	104.7	0.02	0.02	0.03

Table 6.3: Standard deviation of the error in X, Y and Z and in the rotation angles RX, RY and RZ for the different methods and for different baselines

Method	$\boldsymbol{RMSE\ e_d}$ (μm)	$\boldsymbol{e_d}$ **50%** (μm)	$\boldsymbol{e_d}$ **90%** $\boldsymbol{(\mu m)}$	$\boldsymbol{e_d\ max}$ (μm)
CCC (b=300mm)	264.7	226.6	394.1	486.0
CCC (b=800mm)	180.1	157.4	263.8	332.5

Table 6.4: Parameters of e_d error for different baselines

What it is important to highlight in this section, is that without an improvement on the software or hardware of the Stereo Vision System, a significant reduction in the error was achieved in every metric. In this case, the only change was on the relative position between cameras.

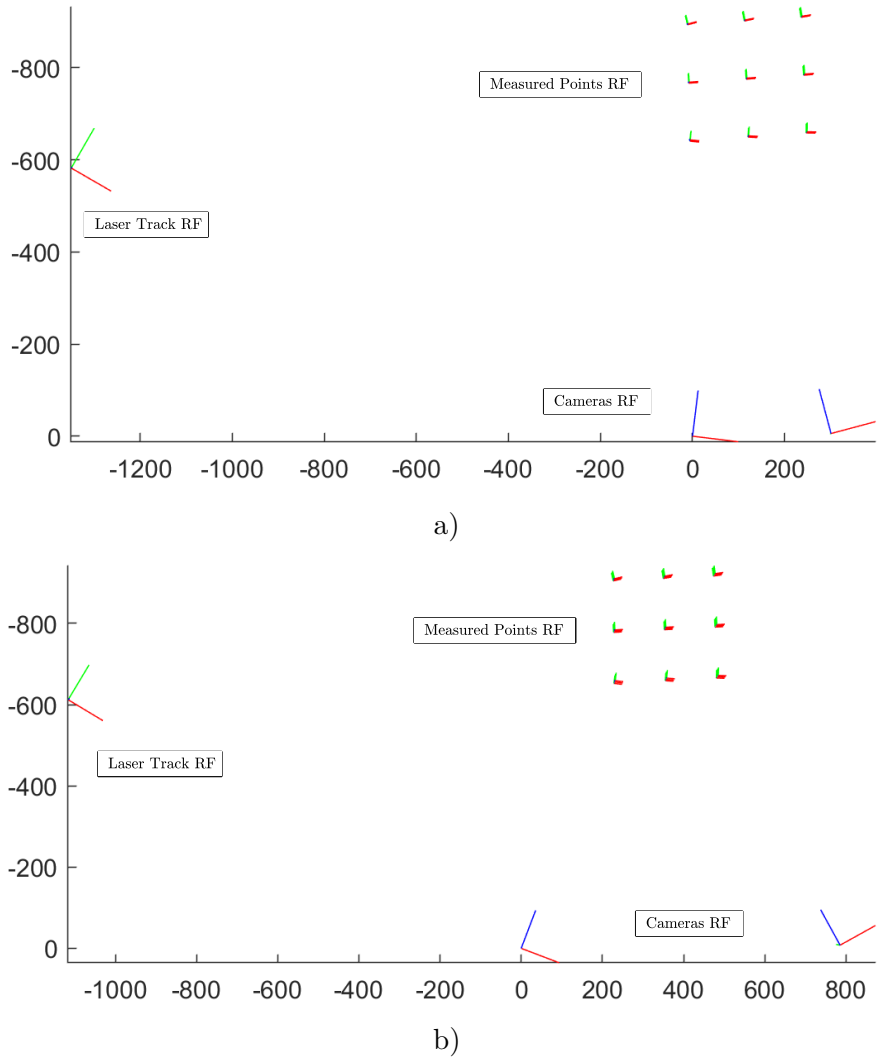


Figure 6.6: a) Experiment reconstruction with 300 mm of baseline, b) Experiment reconstruction with 800 mm of baseline

6.1.3 Poses with highest error

In order to have an estimation of which poses produce the highest errors, cross-validation was carried out. For pose i, the transformation matrices T_l^c and T_r^m were computed individually using the other 26 poses. In other words, the alignment of the data is performed without the point that is going to be measured. This guarantees that the computed e_d in the point is not affected by the optimization problem that computes the transformation matrices.

This experiment was carried out with a baseline of 800 mm, using the same data from the previous section.

The results are shown in Figure 6.7 and Figure 6.8.

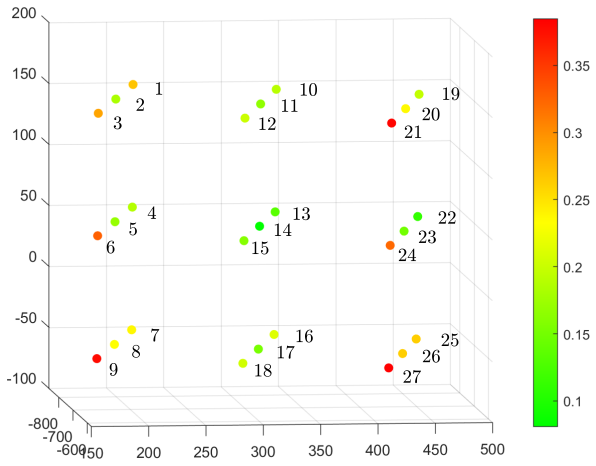


Figure 6.7: e_d in every pose after computing the alignment parameters with the other 26 poses.

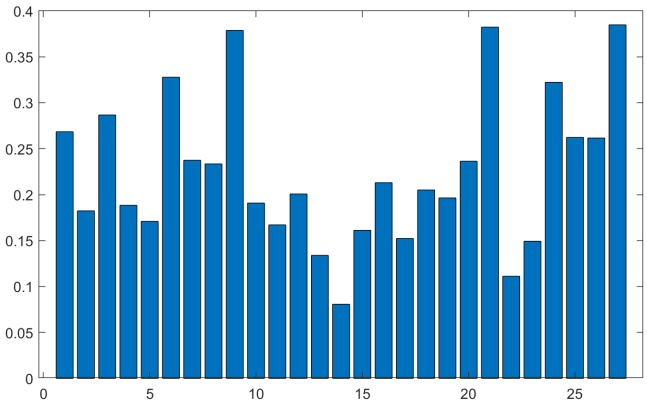
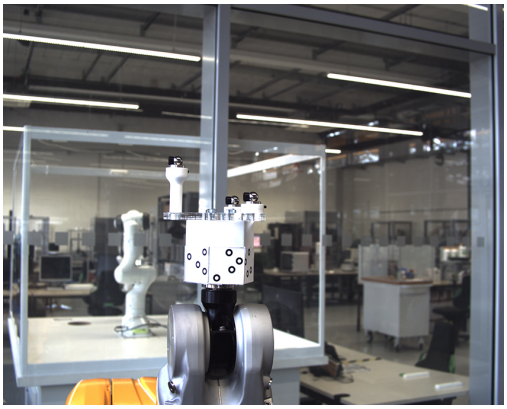


Figure 6.8: Bar chart for points in Figure 6.7

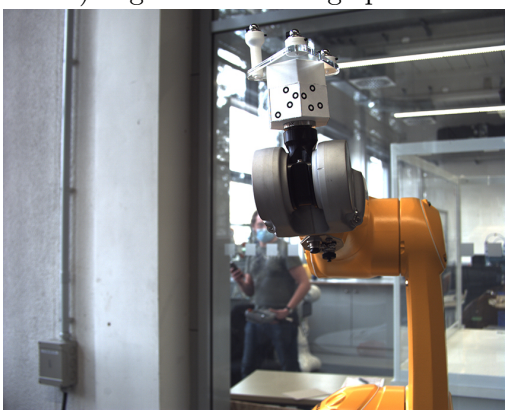
The three highest errors in positions 9, 21 and 27, coincides with the corners of the cube located closer to the cameras. The images in those three positions are shown in Figure 6.9. It is possible that these errors can be reduced with a most representative calibration, using a bigger calibration board.



a) Left camera image pose 9

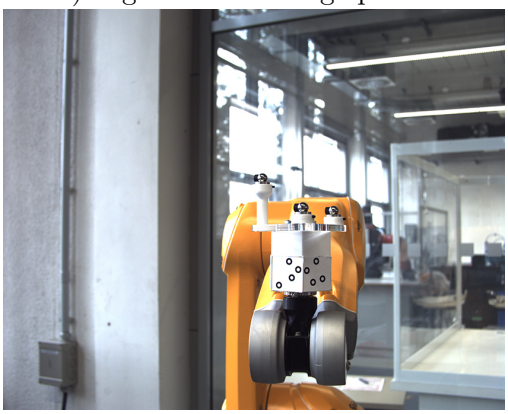
b) Right camera image pose 9

c) Left camera image pose 21



d) Right camera image pose 21

e) Left camera image pose 27



f) Right camera image pose 27

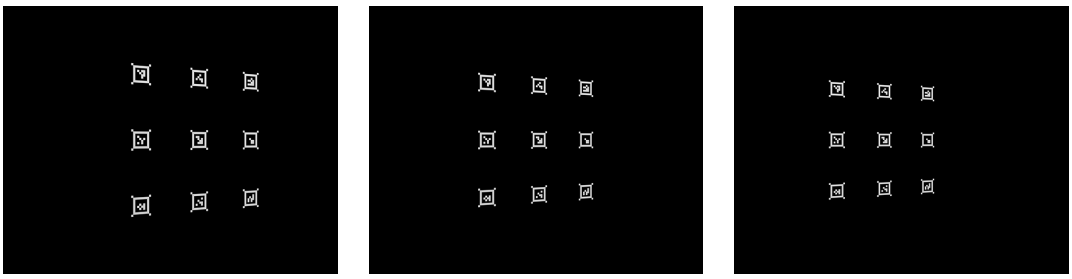
Figure 6.9: Stereo images of critical poses

6.1.4 Synthetic Data

The setup with a baseline of 800 mm was virtually reconstructed in the software Blender. The markers were located forming a cube of side 250 mm at an average distance of 775 mm from the camera's baseline. The calibration was carried out in the same manner as in Section 3.6 with synthetic calibration images. The dimension of the output images was 2560x2048 px and focal length was 16 mm, equal to the real cameras at the lab.

The difference with the real experiment is that, the rendered images from the software have no distortion, have no noise, and the markers always look sharp because of the absence ot blurriness effects. As a result, it is expected that the errors turn out to be lower.

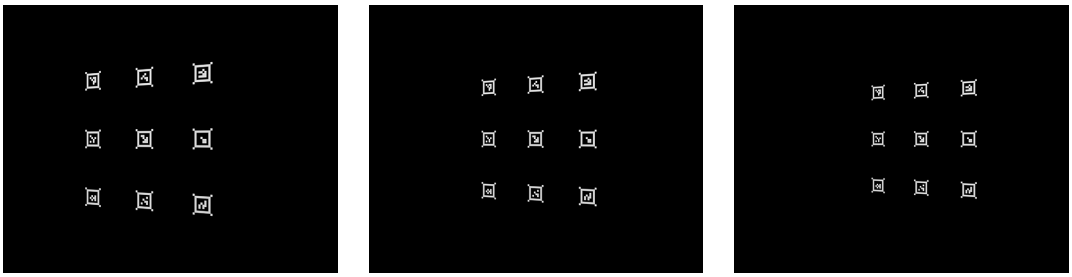
In Table 6.5 it is shown the comparison between the best performing real test (CCC b=800mm) and the computer-generated test. There is a remarkable difference in all the metrics in favor of the synthetic test. The results of this experiment can be interpreted as the maximum level of accuracy that is possible to reach with the proposed setup.



a) Left camera image 650 mm from baseline

b) Left camera image 775 mm from baseline

c) Left camera image 900 mm from baseline



d) Right camera image 650 mm from baseline

e) Right camera image 775 mm from baseline

f) Right camera image 900 mm from baseline

Figure 6.10: Computer generated images: Markers at different distances from the baseline

Method	$\boldsymbol{\sigma_x}$ (μm)	$\boldsymbol{\sigma_y}$ (μm)	$\boldsymbol{\sigma_z}$ (μm)	$\boldsymbol{\sigma_{Rx}}$ (deg)	$\boldsymbol{\sigma_{Ry}}$ (deg)	$\boldsymbol{\sigma_{Rz}}$ (deg)
CCC (b=800mm)	51.6	141.6	104.7	0.02	0.02	0.03
Syntetic Data	26.3	20.7	16.3	0.03	0.05	0.05

Method	$\boldsymbol{RMSE}$ $\boldsymbol{e_d}$ (μm)	$\boldsymbol{e_d}$ **50%** (μm)	$\boldsymbol{e_d}$ **90%** $\boldsymbol{(\mu m)}$	$\boldsymbol{e_d}$ $\boldsymbol{max}$ (μm)
CCC (b=800mm)	180.1	157.4	263.8	332.5
Syntetic Data	37.2	31.5	54.1	61.3

Table 6.5: Error comparison between synthetic and real data

6.1.5 Robot Accuracy

This section intends to estimate the absolute error of the Robot measuring system. The measured poses of the robot was extracted and aligned with the measurements from the laser tracker, as seen for the cameras system in Section 5.2.

On Figure 6.11 it is plotted the reconstruction of the experiment, with the frame of references from the different instruments. It is worth mentioning that the robot measurement was obtained from the same experiment of the CCC markers with a baseline of 800 mm. Because of this, the laser tracker measurements are the same.

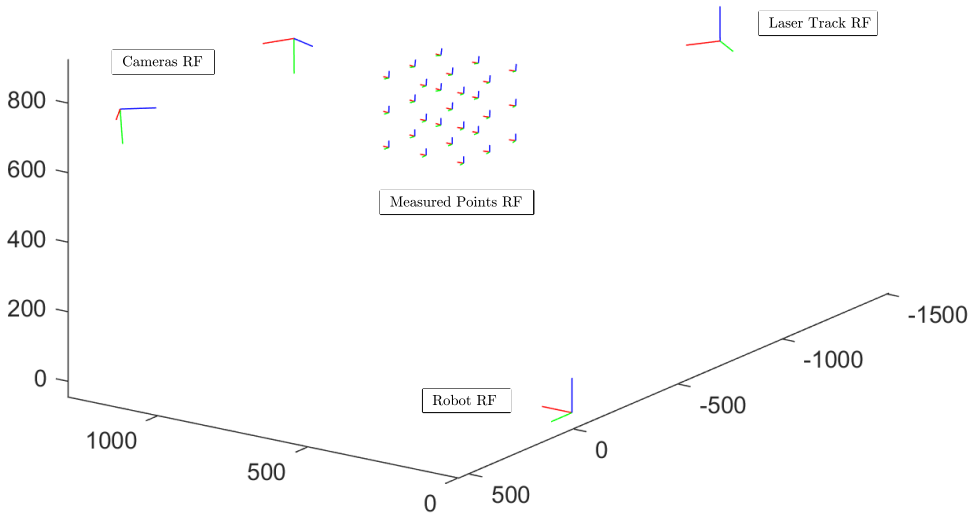


Figure 6.11: Reconstruction of the test set-up with the frame of reference from the different instruments and from the 27 measured poses.

Table 6.6 summarized the absolute error metrics for the Robot measuring system, and compare them with the best performance camera test.

Method	$\boldsymbol{\sigma_x}$ (μm)	$\boldsymbol{\sigma_y}$ (μm)	$\boldsymbol{\sigma_z}$ (μm)	$\boldsymbol{\sigma_{Rx}}$ $(deg.)$	$\boldsymbol{\sigma_{Ry}}$ $(deg.)$	$\boldsymbol{\sigma_{Rz}}$ $(deg.)$
CCC (b=800mm)	51.6	141.6	104.7	0.02	0.02	0.03
Robot	55.3	94.7	52.7	0.02	0.02	0.02

Method	$\boldsymbol{RMSE}$ $\boldsymbol{e_d}$ (μm)	$\boldsymbol{e_d}$ **50**% (μm)	$\boldsymbol{e_d}$ **90**% (μm)	$\boldsymbol{e_d}$ $\boldsymbol{max}$ (μm)
CCC (b=800mm)	180.1	157.4	263.8	332.5
Robot	121.7	100.3	180.2	244.7

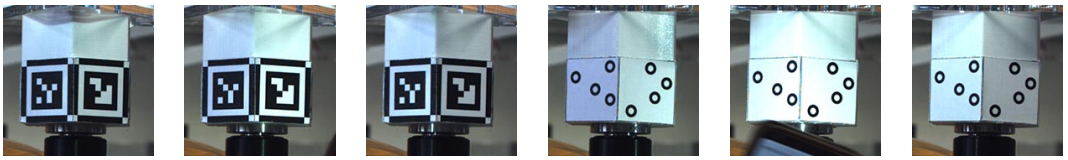
Table 6.6: Volumetric error comparison between robot and camera measurements

The experiment shows clearly that in the test environment the robot has higher absolute accuracy than the stereo camera system. Nonetheless, on the target application at Heinz Maier-Leibnitz Center, the Staubli robot RX160 has lower repeatability ($\pm\,0.05\,mm$) in comparison with the Staubli robot TX2-60L at the lab ($\pm\,0.03\,mm$). Moreover, the cameras on the real application will have a resolution of 20 Mpx in contrast with the actual cameras of 5 Mpx each.

At 1000 mm distance a single-pixel represents an area of 312x312 μm^2 for the actual cameras, while for the 20 Mpx cameras this area is reduced to 48x48 μm^2. Thus, it is expected that the error of the Stereo Vision system is reduced in the same proportion, since according to Equation 2.10 it is directly related.

6.2 Repeatability

A set of 45 image pairs for each type of marker were taken, to measure the repeatability of the stereo vision system. Those images were taken for the same pose of the robot but with different light conditions as can be seen in Figure 6.12. To generate these changes, a source of light was placed at different positions during the data collection.



a) b) c) d) e) f)

Figure 6.12: Images of the robot end-effector for different light conditions

On Table 6.7 it is shown the standard deviation in position of X, Y, and Z axes, and the same for the rotation expressed in the variation of Euler angles.

Method	σ_x (μm)	σ_y (μm)	σ_z (μm)	σ_{Rx} (deg)	σ_{Ry} (deg)	σ_{Rz} (deg)
ArUcoE Single Image	255.6	112.1	5.021	0.02	0.04	0.02
ArUcoE Stereo SR	8.5	6.4	4.7	0.03	0.03	0.02
ArUcoE Stereo Sub-pixel	9.1	5.2	4.7	0.02	0.03	0.02
CCC	9.1	4.1	3.0	0.02	0.02	0.03

Table 6.7: Standard deviation of the error in X, Y and Z and in the rotation angles RX, RY and RZ for the different methods

Table 6.8 shows the maximum position error for each axes. While Table 6.9 shows the parameters of the total distance error e_d

Method	e_{max} x (μm)	e_{max} y (μm)	e_{max} z (μm)
ArUcoE Single Image	954.9	420.7	12.4
ArUcoE Stereo SR	18.7	17.1	11.0
ArUcoE Stereo Sub-pixel	19.7	12.6	9.4
CCC	16.5	9.9	7.6

Table 6.8: Maximum position error in each axis.

Method	$RMSE$ e_d (μm)	e_d 50% (μm)	e_d 90% (μm)	e_d max (μm)
ArUcoE Single Image	276.0	162.5	465.5	1043.5
ArUcoE Stereo SR	11.5	10.5	15.9	19.8
ArUcoE Stereo Subpixel	11.4	10.2	16.4	22.3
CCC	10.3	8.9	15.2	17.4

Table 6.9: Parameters of e_d error

It is evident that the error values are much lower than for the volume accuracy test in any metric. The system thus has very high repeatability but a non-sufficient absolute accuracy.

6.3 Pose compensation

The pose compensation of the robot was carried out as seen in Section 5.5. The target position was x = 0 mm, y = 0 mm, and z= 800 mm from the left camera FR, while the target orientation was Rx = -180 deg, Ry = -25 deg, and Rz = 0 deg. After 6 iterations the robot reached a pose with an average distance error e_d equal to 17 μm and 0.05 degrees average rotation error in each axes.

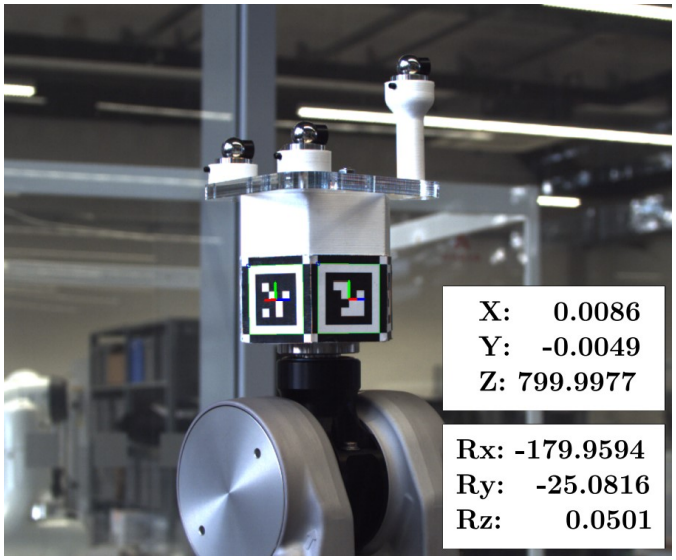


Figure 6.13: Final measurement by the Stereo Vision system

On Figure 6.14 it is shown the measured error on each axis and the total error e_d over time. Moreover on Figure 6.15 the Euler angles errors are shown over time.

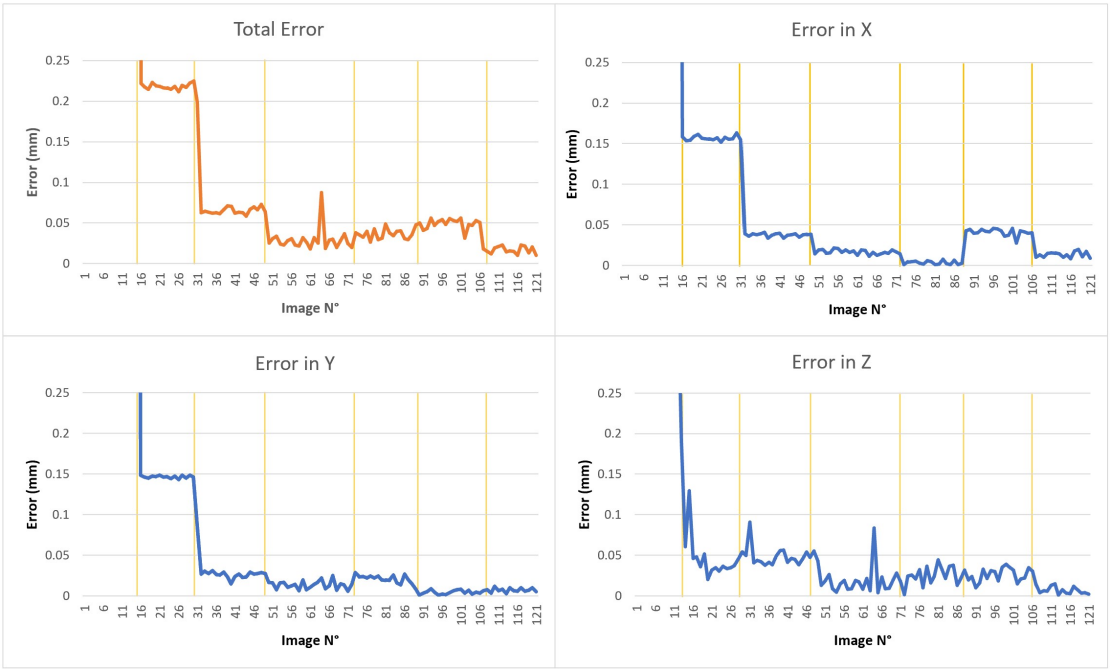


Figure 6.14: Total location error e_d and errors in each axis over time. The vertical yellow lines indicates the robot movement.

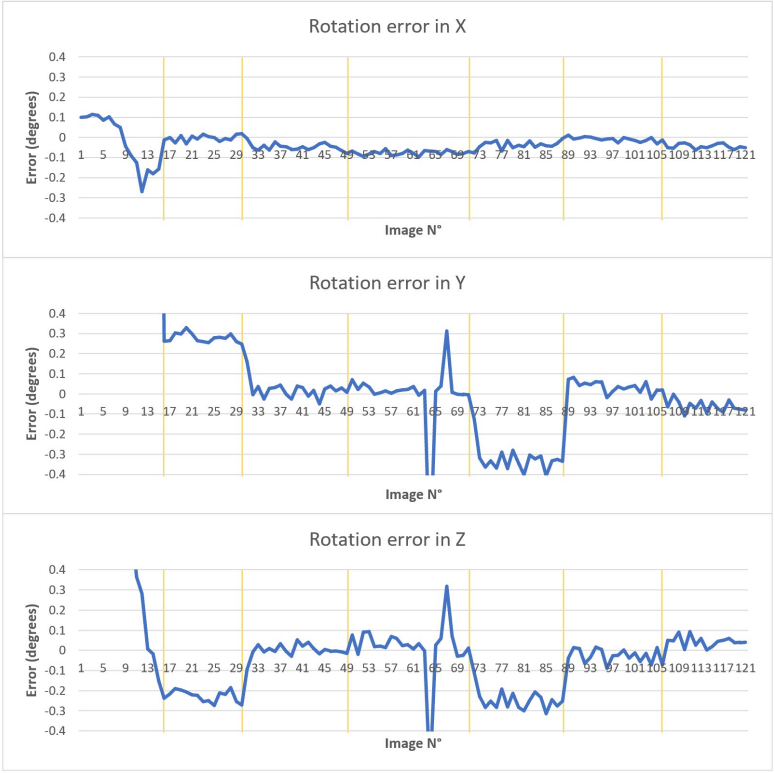


Figure 6.15: Rotation errors in the different axes. The vertical yellow lines indicates the robot movement.

It is important to highlight that the reached accuracy coincides in order of magnitude with the robot's repeatability that according to the manufacturer is 0.03 μm. What these results express is that the compensation was successful with an accuracy equal to the repeatability of the robot.

As a consequence, the final position of the robot will be accurate if measured by the camera system. However, as seen in Section 6.1 the camera system could have an absolute total error as high as 332 μm for a volume of 250x250x250 mm^3.

Nonetheless, since the robot reaches the target pose with high accuracy if measured by the cameras, by improving the stereo vision system also the robot absolute error will be reduced to the same degree. The compensation process allows the robot to have an absolute accuracy in the range of the stereo vision system, being the repeatability of the robot the accuracy limit.

Chapter 7

Conclusion

7.1 Summary

The main focus of this thesis was to develop a Computer Vision measuring system to estimate the pose of a sample in space. The required accuracy for its implementation on the STRESS-SPEC neutron diffractometer was very high, $\pm$ 50 μm in location and $\pm$ 0.5 *degrees* in orientation. Two different methods were proposed based on fiducial markers and Stereo Vision.

The first one consisted of the use of ArUco markers, which are square fiducial with an internal codification. For the shape of these markers, it was possible to estimate their pose using a single camera. These markers were modified for their use with subpixel algorithms and were called Enhanced ArUco markers (ArUcoE). These modified markers, according to the preliminary test, improved considerably the accuracy with a single camera but it was still not enough for the target application.

After that, it was decided to use a Stereo Vision system to estimate the pose of the markers. In the test environment, the RMSE error was reduced more than four times with respect to the single image method. A formula to estimate the optimal position between cameras was presented in this document, which helped to reduce the absolute error by 32 %.

In order to improve the corner position of ArUcoE on the images, a Super-Resolution algorithm was tested. The results did not improve from the standard subpixel algorithm, and the processing time was approximately 5 times higher. Taking this into consideration, it was decided not to proceed with the use of it.

The second method consisted of the use of CCC markers. Since they are not internally

codified, the point correlation between the stereo image pair was achieved using epipolar constraints. For pose estimation using these types of markers, it was necessary to build a 3D model with the relative position of the markers. This was achieved without any external equipment, and using the same cameras for pose estimation. The alignment between the different poses was possible with a modified version of the ICP algorithm, that was also able to detect outliers.

For pose estimation with CCC, it was needed to match the measured points with the ones from the 3D map. If a true correlation was set, the pose could be computed with a closed-form expression. The method for point matching was proposed by the author. This method takes advantage of the high precision of the cameras, and the relatively low number of points to be measured, for then compute the matching elements between the two clouds of points based on their relative position with respect to each other. The same method can identify outliers and it takes a single iteration to produce the output, in contrast with other proposed methods for partial CoP alignment that takes several iterations.

The repeatability and accuracy metrics were obtained using a laser tracker. Because of the experiment set-up, it was necessary to align the data from the different instruments using two translation matrices. They were computed through an optimization algorithm developed by the author.

The experiments have shown that with the use of ArUco and CCC markers a similar level of accuracy is reached, both in stereo vision configuration. With the cameras at the testing center, the required absolute accuracy was not achieved, but it is expected to do so with the new equipment. Moreover, it was shown that with the proposed control loop the robot could reach the accuracy of the camera system till the value of its repeatability.

7.2 Future work

In order to reach the target accuracy, the proposed methods have to be tested with the pair of cameras of the target application. Because of its resolution, it is expected that the error could be reduced up to six times. It is possible that at that level of precision some unconsidered effects could be present that should be properly analyzed.

The validation process with the laser tracker was carried out manually, which has required long measuring times. It was necessary almost an hour for taking a set of 27 measures. At FAPS, there is an additional laser tracker that could be automated in order to save time and be able to generate more validation data.

The author considers that there is still space to improve the accuracy of the system even with the actual equipment. The test carried out with synthetic data, and the measured precision of the system, have shown that it is still possible to keep reducing

the error. The highest errors were located at the corners of the measured volume, areas that the calibration board could not reach due to its dimensions. The calibration could be improved by using professional boards that cover almost the complete field of view of the images. Moreover, more complex calibration techniques with higher degrees of freedom could be tested.

Acronyms

AI
artificial intelligence

ArUcoE
ArUco marker Enhanced

CDF
Cumulative density function

CNN
Convolutional Neural Network

CoP
Cloud of Points

CV
Computer Vision

CVS
Computer Vision System

EMS
External measuring system

FR
Frame of reference

HR
High Resolution

HT
Homogeneous Transformation

HTM
Homogeneous Transformation Matrix

ICP
Iterative Closest Points

LR
Low Resolution

MSE
Mean Squared error

PDF
Probability density function

RMCS
Robot motion control system

RMSE
Root Mean Squared error

SRNN
Super-Resolution Neural Networks

SVS
Stereo Vision System

Appendices

Appendix A

Two link planar arm: kinematic error compensation

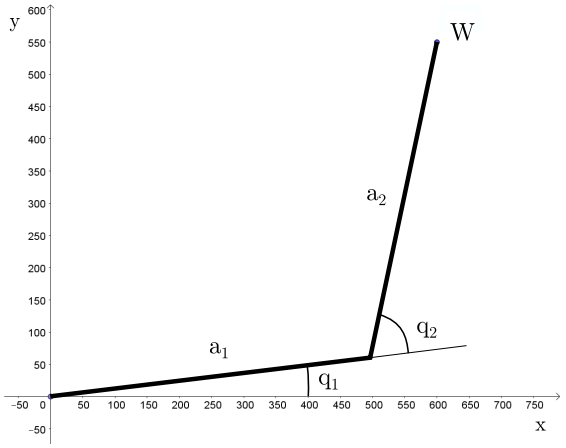


Figure A.1: Two link planar arm model

The two link planar arm is one of the most simple manipulator models, it consist in two links united by a revolute joint, and one of them bonded to the word reference frame by another revolute joint. The prelimar test for the controller was done using this model due to the fact that, the kinematics and inverse kinematics equation can be easily derived.

Expresions A.1 and A.2 describe the Kinematics and inverse Kinematics respectively.

They were obtains from the Book of Sicilianos [38].

Direct kinematics:

$$\begin{aligned} W_x &= a_1\, cos(q_1) + a_2\, cos(q_1 + q_2) \\ W_y &= a_1\, sin(q_1) + a_2\, sin(q_1 + q_2) \end{aligned} \tag{A.1}$$

Inverse kinematics:

$$\begin{aligned} q_1 &= Atan2(W_y, W_x) \\ q_2 &= \pm\, cos^{-1}\left(\frac{W_x^2 + W_y^2 - a_1^2 + a_2^2}{2a_1\sqrt{W_x^2 + W_y^2}}\right) \end{aligned} \tag{A.2}$$

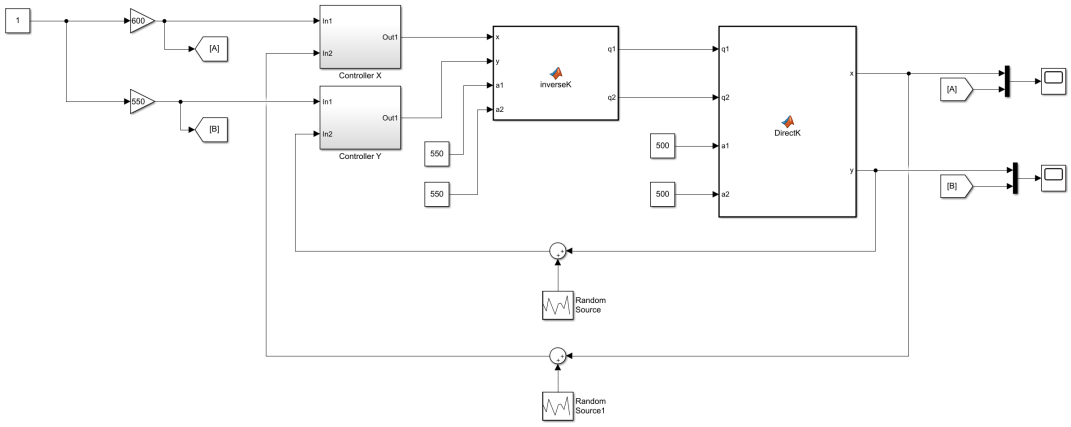


Figure A.2: Simulink model of a two link planar arm with a deficient inverse kinematic block, being compensated with an external control loop

Following the control architecture of Figure 1.7, a Simulink model was built. The desired position in x and y are the input of the control system. In the first iteration both values are feed through the controllers, and because of the architecture of them Figure A.3, they are not modified.

The inverse kinematic block (IKB) in this example, works with wrong values of a_1 and a_2. The erroneous values of q_1 and q_2 are feed through the direct kinematic block (DKB) which compute the coordinates of W with the real values of a_1 and a_2.

As it can bee seen in Figure A.2, the lengths a_1 and a_2 of the IKB are different from the DKB by a 10%. In the target application the error is expected to be much

smaller, however even with this large error the position is reach the desired range in only 4 iterations.

The desired position in x is 600 mm while on y is 500 mm. The error due to measurement is model as a gaussian noise with a standard deviation of 17 μm. The error in position should be less than $\pm 50\,\mu m$.

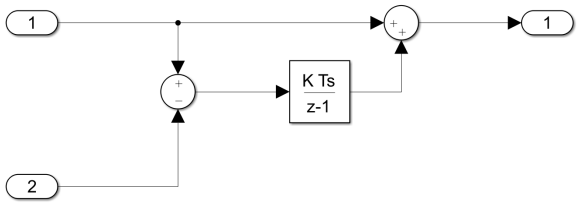


Figure A.3: External controller

The results of the simulation are shown in Figure A.5. For both axis, the error is reduced to the desired range on the fourth iteration and it is maintained throught the entire simulation. In Figure A.4 is graphically represent the poses of the manipulator for the first three iterations.

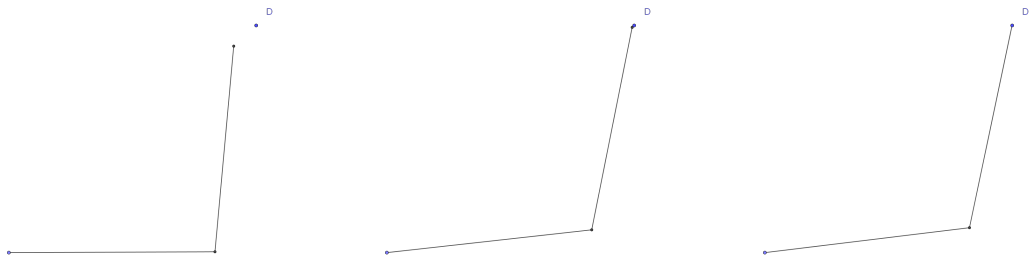
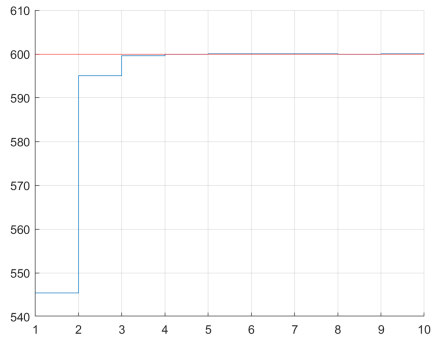
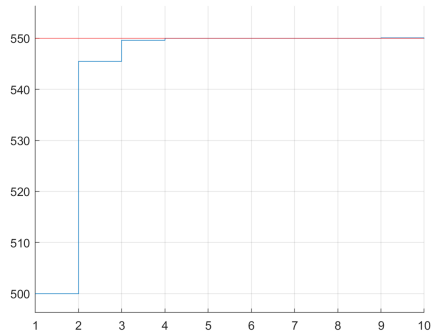


Figure A.4: Pose of the manipulator from iteration 1 to 3. Point D represent the desired position for W

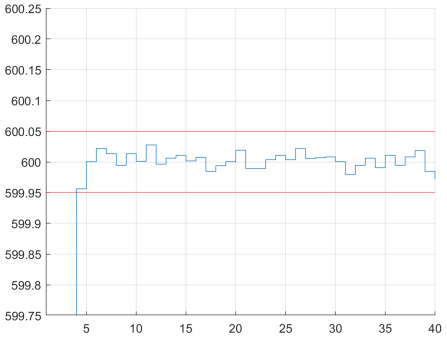
This simple simulation intends to provide preliminary results regarding the behaviour of the external controller. On the target application the number of axis will be 6 and not 2 as in this case. As explained in Chapter 1 the development of a controller is not the main focus of this thesis.



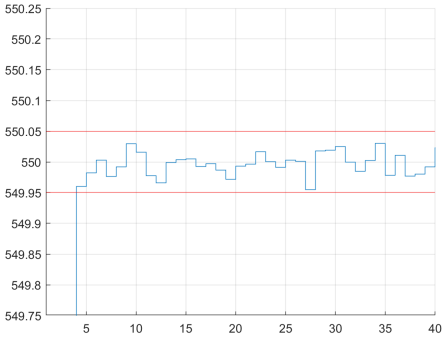
W_x through sequential iteration



W_y through sequential iteration



W_x with error range tolerance limits in red



W_y with error range tolerance limits in red

Figure A.5: Simulation output

Appendix B

Stereo Error: Theoretical Analysis

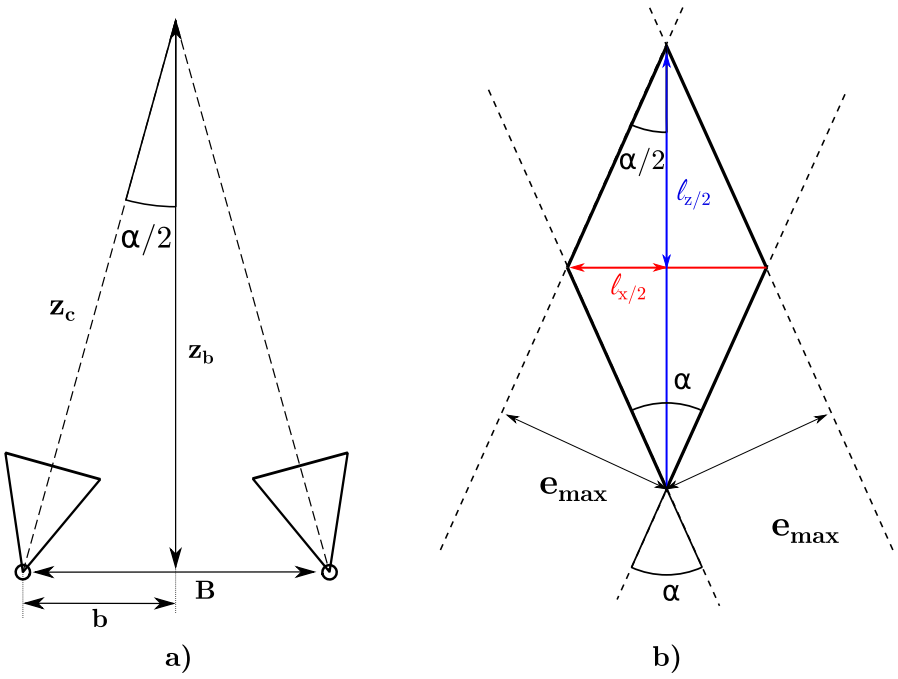


Figure B.1: a) relative pose between cameras and point to be measured b) Uncertainty volume from plane x-z perspective

From the configuration of the cameras shown in Figure B.1 a) the expressions B.1 and B.2 can be derived:

$$sin\left(\frac{\alpha}{2}\right) = \frac{b}{z_c} \tag{B.1}$$

$$tan\left(\frac{\alpha}{2}\right) = \frac{b}{z_b} \tag{B.2}$$

Also, from the geometry of the uncertainty volume are obtained the relationships B.2 and B.2. l_x and l_z are the maximum length of the uncertainty volume in the x and z-axis respectively, while α is the angle difference between the orientation of both cameras on the y-axis.

$$sin\left(\frac{\alpha}{2}\right) = \frac{e_{max}}{l_z} \tag{B.3}$$

$$tan\left(\frac{\alpha}{2}\right) = \frac{\frac{l_x}{2}}{\frac{l_z}{2}} = \frac{l_x}{l_z} \tag{B.4}$$

Combining Equation B.1 and Equation B.3, Equation B.5 can be derived. A similar procedure can be made between Equation B.2 and Equation B.4 to obtain Equation B.6.

$$l_z = \frac{z_c}{b} e_{max} \tag{B.5}$$

$$l_x = \frac{b}{z_b} l_z \tag{B.6}$$

Equation B.6 can be combained with Equation B.5 to obtained the set of expresions B.8. The procedure is shown in B.7.

$$\begin{aligned} l_z &= \frac{z_c}{b} e_{max} \\ l_x &= \frac{b}{z_b} l_z \\ &= \frac{b}{z_b} \frac{z_c}{b} e_{max} \\ &= \frac{z_c}{z_b} e_{max} \end{aligned} \tag{B.7}$$

$$\begin{aligned} l_z &= \frac{z_c}{b} e_{max} \\ l_x &= \frac{z_c}{z_b} e_{max} \end{aligned} \tag{B.8}$$

The proposed probability model consists of assuming that the error on each axis follows a normal distribution. Because of this, the parallelepiped uncertainty volume is transformed to an ellipsoid that simplifies the model. This is a conservative step since the volume of the ellipsoid will be larger than the one from the parallelepiped. The illustration of this concept is shown in Figure B.2.

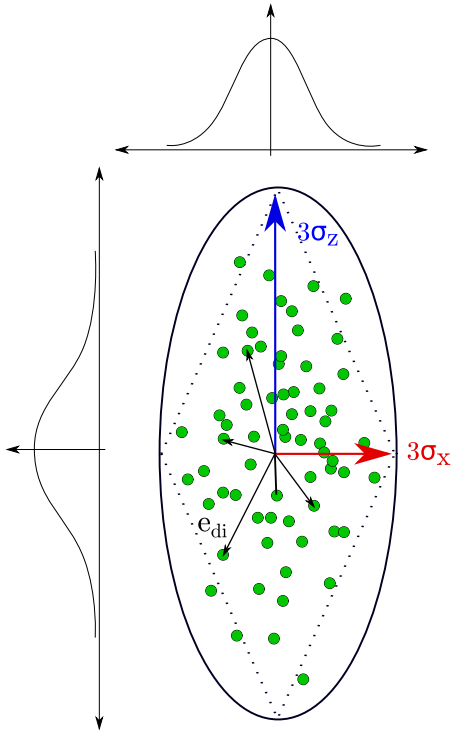


Figure B.2: Transformation of the parallelepiped error volume into an ellipsoid. The boundary on each axes is equal to three times its respective standard deviation.

The standard deviation in x and z would be defined as one-sixth of l_x and l_z respectively, Equation B.9. This condition set a probability of 99.19% that a measured point is confined inside the new uncertainty volume. This value was computed as the joint probability of each independent coordinate of the measured point laying inside the $\pm 3\,\sigma$ confidence interval of their respective axes,(0.9973^3).

$$\begin{aligned} \sigma_x &= \frac{z_c}{6\,z_b}\,e_{max} \\ \sigma_z &= \frac{z_c}{6\,b}\,e_{max} \end{aligned} \tag{B.9}$$

To compute σ_y it was considered that the error distribution is affected by the sum of the error of both cameras. The maximum error of each camera was defined as e_{max}, while its probability distribution would be considered as normal with a standard deviation

equal to a sixth of e_{max}, Equation B.10. The point is at the same distance from both cameras and e_{max} is equal for both. The value of σ_y then would be computed as expressed in Equation B.11.

$$\sigma_{ec} = \frac{1}{6}\, e_{max} \tag{B.10}$$

$$\sigma_y = \sqrt{\sigma_{ec}^2 + \sigma_{ec}^2} \;\; = \frac{\sqrt{2}}{6}\, e_{max} \tag{B.11}$$

The resulting expression for computing the standard deviation on each axis is express in B.12.

$$\begin{aligned} \sigma_x &= \frac{z_c}{6\, z_b}\, e_{max} \\ \sigma_y &= \frac{\sqrt{2}}{6}\, e_{max} \\ \sigma_z &= \frac{z_c}{6\, b}\, e_{max} \end{aligned} \tag{B.12}$$

The standard deviation on each axes provides a good description of the uncertainty in measurement. Nonetheless, it is necessary first to define a unique parameter to be minimized in order to find the optimal position for the cameras. The chosen parameter was the standard deviation of the length of the error. The error length e_{di} is computed as the distance between a measured point and the mean value as express in Equation B.13.

$$\begin{aligned} e_{di} &= \sqrt{(\hat{x} - x_i)^2 + (\hat{y} - y_i)^2 + (\hat{z} - z_i)^2} \\ &= \sqrt{e_{xi}^2 + e_{yi}^2 + e_{zi}^2} \end{aligned} \tag{B.13}$$

The variance of e_d can be computed with expression B.14. The demonstration of this equality is shown in B.15.

$$\sigma_d^2 = \sigma_x^2 + \sigma_y^2 + \sigma_z^2 \tag{B.14}$$

$$\begin{aligned}
\sigma_d^2 &= \frac{1}{n}\sum_{i=1}^{n} e_{di}^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(e_{xi}^2 + e_{yi}^2 + e_{zi}^2) \\
&= \frac{1}{n}\sum_{i=1}^{n} e_{xi}^2 + \frac{1}{n}\sum_{i=1}^{n} e_{yi}^2 + \frac{1}{n}\sum_{i=1}^{n} e_{zi}^2 \\
&= \sigma_x^2 + \sigma_y^2 + \sigma_z^2
\end{aligned} \tag{B.15}$$

The formula of σ_d^2 coincides with the formula of the Mean Squared Error. Thus, it is possible to derive Equation B.16:

$$RMSE = \sigma_d \tag{B.16}$$

The goal is to find the optimal relationship between z_b and b (Figure B.1) that minimizes σ_d. With this aim, it is required to express Equation B.14 in function of $x = \frac{b}{z_b}$. The procedure is shown in B.17.

$$\begin{aligned}
\sigma_d^2 &= \sigma_x^2 + \sigma_y^2 + \sigma_z^2 \\
&= (\frac{z_c}{6\,z_b}\,e_{max})^2 + (\frac{\sqrt{2}}{6}\,e_{max})^2 + (\frac{z_c}{6\,b}\,e_{max})^2 \\
&= \frac{e_{max}^2}{36}\left(\frac{z_c^2}{z_b^2} + 2 + \frac{z_c^2}{b^2}\right) \\
&= \frac{e_{max}^2}{36}\left(\frac{z_b^2+b^2}{z_b^2} + 2 + \frac{z_b^2+b^2}{b^2}\right) \\
&= \frac{e_{max}^2}{36}\left(4 + \frac{b^2}{z_b^2} + \frac{z_b^2}{b^2}\right) \\
&= \frac{e_{max}^2}{36}\left(4 + x^2 + 1/x^2\right)
\end{aligned} \tag{B.17}$$

e_{max} is function of z_b and b, so expression must be transform as shown in B.18.

$$\begin{aligned} e_{max}^2 &= (\, z_c \, \frac{\Delta e_{px}}{f_l})^2 \\ &= \frac{z_c^2}{f_l^2} \\ &= \frac{z_b^2 + b^2}{f_l^2} \\ &= \frac{z_b^2}{f_l^2} \left(1 + \frac{b^2}{z_b^2}\right) \\ &= \frac{z_b^2}{f_l^2} \, (1 + x^2) \end{aligned} \tag{B.18}$$

Combining B.18 and B.18 Equation B.19 is obtained.

$$\sigma_d^2 = \frac{z_b^2}{36\, f_l^2} \, (1 + x^2) \left(4 \, + \, x^2 \, + \, \frac{1}{x^2}\right) \tag{B.19}$$

Equation B.19 can be further simplified, resulting in the final expression B.20.

$$\sigma_d^2 = \frac{z_b^2}{36\, f_l^2} \, \left(\frac{1}{x^2} \, + \, 5 \, + \, 5\, x^2 \, + \, x^4\right) \tag{B.20}$$

f_l is the focal length of both cameras that is constant, while z_b is the working distance that is also constant. Because of this, finding x_{opt} that minimize σ_d^2 is equivalent to find x_{opt} that minimize $(1/x^2 \, + \, 5 \, + \, 5x^2 \, + \, x^2)$.

$$x_{opt} = \underset{x}{\operatorname{argmin}} \left(\frac{1}{x^2} \, + \, 5 \, + \, 5\, x^2 \, + \, x^4\right) \tag{B.21}$$

The solution of this problem was obtained using the web tool Wolfram Alpha [39]. There were six different solutions to the problem but only one that was positive without an imaginary part.

$$\begin{aligned} x_{opt} &= \sqrt{\sqrt{2} - 1} \\ x_{opt} &\simeq 0.6436 \end{aligned} \tag{B.22}$$

The optimal baseline can be quickly obtained as shown in B.23.

$$\begin{aligned} b_{opt} &\simeq 0.6436\, z_b \\ B_{opt} &\simeq 1.2872\, z_b \end{aligned} \tag{B.23}$$

Appendix C

SRNN Architecture in Tensorflow

```
import tensorflow as tf

def relu_advanced(x):
    return tf.keras.activations.relu(x, max_value=255)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D( 6,(6,6),input_shape=(16,16,1),
    padding="same", activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(3,3), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(3,3), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(3,3), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.1))

model.add(tf.keras.layers.Conv2DTranspose(6,(6,6),padding="same",
    dilation_rate=(1,1),strides=(2, 2), activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(6,6), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(3,3), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D( 6,(3,3), padding="same",
    activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.1))
```

```
18
19 model.add(tf.keras.layers.Conv2DTranspose(6,(6,6) ,padding="same",
       dilation_rate=(1,1),strides=(2, 2), activation=tf.nn.relu))
20 model.add(tf.keras.layers.Conv2D( 6,(9,9), padding="same",
       activation=tf.nn.relu))
21 model.add(tf.keras.layers.Conv2D( 6,(6,6), padding="same",
       activation=tf.nn.relu))
22 model.add(tf.keras.layers.Conv2D( 1,(3,3), padding="same",
       activation=relu_advanced))
23
24 model.save('DNN_model.h5')
```

Listing C.1: Neural network architecture using Keras, coded in Python

Appendix D

Alignment matrices obtention: Matlab Code

The Matlab code for computing T_l^c and T_r^m is shown in Listing D.1 and D.2. The latter one is the objective function to be minimized. The code was based on Equation 5.3.

```
1
2  % T_A and T_B are the set of transformation matrices from the different
       measurements of the cameras and the laser tracker respectively.
3  % X = [p1;p2]. X is a matrix of 2 rows and 7 columns containing the
       parameters for the transformation that we are looking for.
4  % X0 is the initial guess for X
5  % lb and ub are the lower and upper limits of the elements in X
6  % a and b are coefficient that gives weight to the angular and position
       error respectively.
7  gs = GlobalSearch;
8
9  problem = createOptimProblem('fmincon','x0',X0,
       'objective',@(x)Tot_err(T_A,T_B,X,a,b),'lb',lb,'ub',ub);
10
11 x = run(gs,problem)
```

Listing D.1: Matlab code for optimization

```
1
2  function [Err]=Tot_err(T_A,T_B,X,a,b)
3      % Separete p1 and p2 from X
```

```
    p1 = X(1,:);
    p2 = X(2,:);

    % Built the transformation matrices from p1 and p2
    T_lc = p2T(p1);
    T_rm = p2T(p2);

    % Iterate for every set of measurements
    n=size(T_A,3);
    tot_sum=0;
    for i=1:n
           % Compute Tr_1 and Tr_2 that represent the transformation
               matrices from the laser tracker to the reflectors. The
               first one measured by the cameras and the second one by
               the laser tracker.

           Tr_1=T_lc*T_B(:,:,i);
           Tr_2=T_A(:,:,i)*T_rm;

           % Parametrize transformation matrices Tr_1 and Tr_2. The
               parameters a and b scale the values of the parameters to
               balance in the next step the orientation or position
               error.
           v_1=[a*rotm2quat(Tr_1(1:3,1:3)),b*Tr_1(1:3,4)'];
           v_2=[a*rotm2quat(Tr_2(1:3,1:3)),b*Tr_2(1:3,4)'];

           % The sum of each squared element is added to the total sum,
               but first is divided by the number of measurements.
           tot_sum = tot_sum + sumsqr(v_1 - v_2)/n;
    end

    % Root of the sum of error
    Err_0 = (sqrt(tot_sum));

    % Add relaxated contrain in quaternions.
    Err = Err_0 + (abs(sumsqr(p1(1,1:4).^2)-1)+
        abs(sum(p2(2,1:4).^2)-1));

end
```

Listing D.2: Matlab error function to minimized

Bibliography

[1] *STRESS-SPEC: Materials science diffractometer*. Heinz Maier-Leibnitz Zentrum. URL: https://mlz-garching.de/stress-spec (cit. on pp. 1, 2).

[2] FAPS Institue. URL: https://www.faps.fau.eu/curforsch/raptor-automated-sample-positioning-in-neutron-diffractometry/ (cit. on p. 1).

[3] H.-G. Brokmeier et al. «Texture analysis at neutron diffractometer STRESS-SPEC». In: *Elsevier* 20 (June 2011), pp. 569–571 (cit. on p. 1).

[4] C. Randau et al. «Improved sample manipulation at the STRESS-SPEC neutron diffractometer using an industrial 6-axis robot for texture and strain analyses». In: *Nuclear Instruments and Methods in Physics Research* A 794 (2015), pp. 67–75 (cit. on pp. 1, 3).

[5] *Elastic Scattering*. Heinz Maier-Leibnitz Zentrum. URL: https://mlz-garching.de/englisch/neutron-research/experimental-methods/elastic-scattering.html (cit. on p. 2).

[6] I.A. Bonev A. Nubiola. «Absolute calibration of an ABB IRB 1600 robot using a laser tracker». In: *Robotics and Computer-Integrated Manufacturing* 29 (2013), pp. 236–245 (cit. on p. 3).

[7] Van-Phu Do Chi-Tho Cao and Byung-Ryong Lee. «A Novel Indirect Calibration Approach for Robot Positioning Error Compensation Based on Neural Network and Hand-Eye Vision». In: *App. Sci.* 9 (2019), p. 1940 (cit. on p. 4).

[8] DAVID G. LOWE. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60(2) (2004), pp. 91–110 (cit. on p. 7).

[9] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2010. Chap. 5 Actuators and Sensors, pp. 226–230 (cit. on p. 8).

[10] *Camera Calibration*. OpenCv. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html (cit. on p. 9).

[11] *Lenses Distortion*. OpenCv. URL: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html (cit. on p. 10).

[12] Varlik Kilic. «Performance Improvement of a 3D Reconstruction Algorithm Using Single Camera Images». MA thesis. Middle East Technical University, 2005 (cit. on p. 10).

[13] Qingnan Li, Ruimin Hu, Jing Xiao, Zhongyuan Wang, and Yu Chen. «Learning latent geometric consistency for 6D object pose estimation in heavily cluttered scenes». In: *Journal of Visual Communication and Image Representation* 70 (2020), p. 102790. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2020.102790. URL: https://www.sciencedirect.com/science/article/pii/S1047320320300407 (cit. on p. 14).

[14] Bo Chen et al. «End-to-End Learnable Geometric Vision by Backpropagating PnP Optimization». In: (2020). URL: https://arxiv.org/pdf/1909.06043v3.pdf (cit. on p. 14).

[15] Yannick Bukschat and Marcus Vetter. «EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach». In: (2020). URL: https://arxiv.org/pdf/2011.04307v2.pdf (cit. on p. 14).

[16] Chen Song et al. «HybridPose: 6D Object Pose Estimation under Hybrid Representations». In: (2020). URL: https://arxiv.org/pdf/2001.01869v4.pdf (cit. on p. 14).

[17] Sergey Zakharov et al. «DPOD: 6D Pose Object Detector and Refiner». In: (2020). URL: https://arxiv.org/pdf/1902.11020v3.pdf (cit. on p. 14).

[18] Richard Hartley and Andrew Zisserman. *Multiple View in Computer Vision*. Cambridge, 2004. Chap. 12 Structure Computation, pp. 311–312 (cit. on pp. 15–17).

[19] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2010. Chap. 10 Stereo Vision, pp. 433–435 (cit. on pp. 15, 16).

[20] *Triangulation*. OpenCv. URL: https://docs.opencv.org/3.4.12/d0/dbd/group__triangulation.html (cit. on p. 16).

[21] *Introduction to Mobile Robotics: Iterative Closest Point Algorithm*. George Mason University, Department of Computer Science. URL: https://cs.gmu.edu/~kosecka/cs685/cs685-icp.pdf (cit. on pp. 22, 23).

[22] Hanzhou Lu and Yujie Wei. *Parallel point cloud registration*. URL: https://hanzhoulu.github.io/Parallel-Point-Cloud-Registration/ (cit. on p. 24).

[23] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. «Automatic generation and detection of highly reliable fiducial markers under occlusion». In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2014.01.005 (cit. on p. 25).

[24] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2010. Chap. 10 Stereo Vision, pp. 418–422 (cit. on p. 26).

[25] W FORSTNER. «A fast operator for detection and precise location of distincs points, corners and center of circular features». In: *In Proc. of the Intercommission Conference on Fast Processing of Photogrammetric Data, Interlaken,* 1987 (1987), pp. 281–305. URL: https://cseweb.ucsd.edu/classes/sp02/cse252/foerstner/foerstner.pdf (cit. on p. 27).

[26] Rafael Muñoz Salinas. *Subpixel Accuracy in corner Detection. Which is the best approach?* URL: https://www.youtube.com/watch?v=OhpLCoZ4PtA (cit. on p. 28).

[27] *Detection of ChArUco Corners*. OpenCv. URL: https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html (cit. on p. 28).

[28] Minghua Wang and Qiang Wang. «Hypergraph-regularized sparse representation for single color image super resolution». In: *Journal of Visual Communication and Image Representation* 74 (2021), p. 102951. ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2020.102951 (cit. on p. 33).

[29] Y. K. Badran, G. I. Salama, T. A. Mahmoud, A. Mousa, and A. E. Moussa. «Single image super resolution based on learning features to constrain back projection». In: *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*. 2019, pp. 23–28. DOI: 10.1109/ITCE.2019.8646324 (cit. on p. 33).

[30] Y. Zhang, Q. Fan, F. Bao, Y. Liu, and C. Zhang. «Single-Image Super-Resolution Based on Rational Fractal Interpolation». In: *IEEE Transactions on Image Processing* 27.8 (2018), pp. 3782–3797. DOI: 10.1109/TIP.2018.2826139 (cit. on p. 33).

[31] W. Yang, X. Zhang, Y. Tian, W. Wang, J. Xue, and Q. Liao. «Deep Learning for Single Image Super-Resolution: A Brief Review». In: *IEEE Transactions on Multimedia* 21.12 (2019), pp. 3106–3121. DOI: 10.1109/TMM.2019.2919431 (cit. on pp. 34, 35, 37).

[32] Vincent Dumoulin and Francesco Visin. «A guide to convolution arithmetic for deep learning». In: (2016). URL: https://arxiv.org/pdf/1603.07285v1.pdf (cit. on p. 36).

[33] C. Ledig et al. «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 105–114. DOI: 10.1109/CVPR.2017.19 (cit. on p. 38).

[34] Lance B. Gatrell, William A. Hoff, and Cheryl W. Sklair. «Robust image features: concentric contrasting circles and their image extraction». In: 1992. DOI: 10.1117/12.56761 (cit. on p. 40).

[35] H. Kong, H. C. Akakin, and S. E. Sarma. «A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications». In: *IEEE Transactions on Cybernetics* 43.6 (2013), pp. 1719–1733. DOI: 10.1109/TSMCB.2012.2228639 (cit. on p. 41).

[36] *Blob Detection*. OpenCv. URL: https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html (cit. on p. 42).

[37] J. Yang, H. Li, D. Campbell, and Y. Jia. «Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (2016), pp. 2241–2254. DOI: 10.1109/TPAMI.2015.2513405 (cit. on p. 52).

[38] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2010. Chap. 2 Kinematics, pp. 93–94 (cit. on p. 89).

[39] *Wolfram Alpha*. URL: https://www.wolframalpha.com/ (cit. on p. 97).