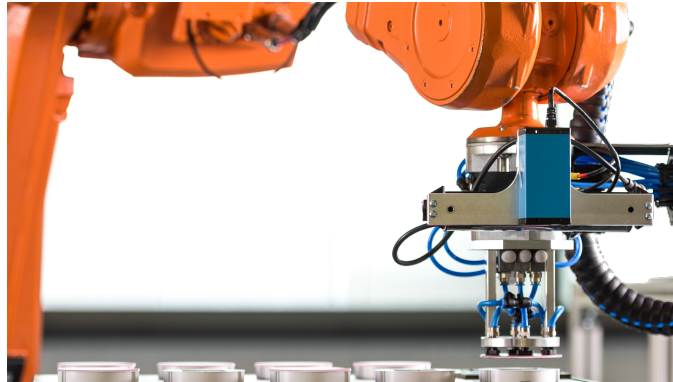




TÉCNICO
LISBOA



Vision Calibration for Industrial Robots

An Approach using Cognex VisionPro

Manuel Lemos Ribeiro

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Leonel Augusto Pires Seabra de Sousa

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo

Supervisor: Prof. Leonel Augusto Pires Seabra de Sousa

Member of the Committee: Prof. José António da Cruz Pinto Gaspar

June 2017

Per aspera ad astra.

Acknowledgments

To the company Applied Micro Electronics, I would like to express my deepest gratitude for the opportunity I was granted as well as for all the support I was given during the development of this project. To my supervisor Jasper Jansen, who continuously supported and encourage me to push further, I thank him for helping me on this most important transition to a professional career, for his professionalism, attitude and work ethics. To my colleagues, at AME, not only for their valuable advices but above all for their friendship and warm welcome.

To my supervisor Prof. Leonel Sousa and to Instituto Superior Técnico, my home for these last five years, for the best education I could ever hoped for as well as for all the amazing teachers I had the opportunity to meet. Moreover, my deepest gratitude for all the friends I had at my side during my academic journey, I dedicate them my success since without them getting here would have been impossible.

Special acknowledgments to the ones that were there since the beginning, for showing me what team work is really about and above all that some friends are for life. Therefore, I would like to personally thank my MEEC colleagues André Stielau, Bernardo Almeida, Bernardo Marques, João Franco, João Melo, João Soares de Melo, José Teixeira, Manel Ávila, João Galamba, Manel Costa, Nuno Sousa and Miguel Monteiro. And last but not least, I would like to thank Sandra Gomes and Bernardo Pontes for the two biggest friendships I could ever have asked for.

To my family, my parents Teresa Lemos and José Ribeiro, as well as my brother Francisco Ribeiro, for their patience and support throughout all my life. I dedicate all my success and happiness to them, for all the education, dedication and care that you gave me, for all the support and help I could ask for. Never have I ever been so grateful because without you, none of this would have been possible.

At last, but not least, to my partner and best friend, Sara Pereira, for all your unconditional support and dedication. Thank you for helping me in times of dissuasion, for encouraging me and pushing me forward every day. I would not be here without your backing.

Resumo

A maior parte dos processos industriais depende da assistência de braços robóticos em diversas aplicações, tais como integração robótica na linha de produção e robótica de montagem. Com a ambição de cumprir os mais elevados padrões de qualidade industrial, exatidão, precisão, flexibilidade e um maior potencial em aumentar a competitividade de mercado, algumas células de robótica industrial possuem sensores de visão. Estes sensores e câmaras associadas são usados para guiar os robôs industriais em ações de “pick and place”, colagem de peças e outras. De modo a executar estas atividades com sucesso, é necessário relacionar as projeções 2D com as posições 3D do sistema de coordenadas do robô, processo este que só é possível através de calibração. O maior foco deste projeto de tese de mestrado consiste em desenvolver uma aplicação standard capaz de automaticamente calibrar as câmaras na estação de trabalho do robô, sendo capaz de estimar a posição e a orientação exatas das câmaras em relação ao robô, ao mesmo tempo compensando a distorção não-linear das suas lentes. Ao mesmo tempo, ferramentas 3D pertinentes são propostas e introduzidas. Os procedimentos desenvolvidos e propostos não só representam uma promissora melhoria até 11 vezes, ao comparar uma câmara calibrada com uma não calibrada, mas também são responsáveis pelo decréscimo significativo da intervenção humana no processo de ensinar a posição e orientação das ferramentas ao robô. Em conclusão, os procedimentos mencionados foram também implementados e integrados com sucesso na empresa onde este projeto foi desenvolvido.

Palavras-chave: Calibração de Câmara, Calibração 'Hand-Eye', Cognex VisionPro, Robôs Industriais, Triangulação Stereo.

Abstract

Most manufacturing processes rely upon the use of industrial robot arms for different applications, such as inline robotics, system assembly robotics and machine tending robotics. In the pursuit of the highest industrial quality standards, accuracy, precision, flexibility and a larger potential to increase the competitiveness in the market, some industrial robot cells make use of vision sensors. These vision sensors and cameras are used to guide the robot cells in pick and place, perform measurements and other applications. In order to successfully execute these tasks, there is the need to relate the cameras' 2D projections with 3D positions of the robot's coordinate system, which can only be done after a calibration process. This master thesis project's main focus is to develop a standard application capable of automatically calibrating the cameras in the robot's workspace, by estimating their exact position and orientation with respect to the robot, while also compensating for the non-linear distortion of its lenses. At the same time, 3D tools are proposed and introduced. The developed calibration procedures not only show an improvement in typical applications up to 11 times, when comparing a calibrated and uncalibrated camera, but are also responsible for significantly decreasing human effort in the process of teaching the tools' positions to the robot. In conclusion, the mentioned procedures were also implemented with success at the company at which this project has been developed.

Keywords: Camera Calibration, Hand-Eye Calibration, Cognex VisionPro, Industrial Robots, Stereo Triangulation.

Contents

Acknowledgments	v
Resumo	vii
Abstract	ix
List of Tables	xv
List of Figures	xvii
Glossary	xix
1 Introduction	1
1.1 The Company	1
1.2 Motivation	2
1.3 Objectives and Deliverables	3
1.4 Thesis Outline	4
2 Fundamental Concepts	7
2.1 Robot Coordinate Systems	7
2.1.1 Quaternions	8
2.1.2 Work Object Data	8
2.1.3 Tool Data	9
2.2 Robot Calibration	9
2.3 Camera Model	10
2.3.1 Pinhole Model	10
2.3.2 Lens Distortion	11
2.4 Camera Calibration	12
2.5 Hand-Eye Calibration	13
2.6 Stereo Triangulation	14
3 Supporting Software and Interfaces	17
3.1 VisionPro	17
3.1.1 Useful Definitions	17
3.1.2 Calibration Plate	18
3.1.3 Camera Calibration	19
3.1.4 Hand-Eye Calibration	20

3.2	RobotStudio and RAPID	20
3.3	RoboVision	21
3.4	GigE Vision	21
4	Proposed Solution	23
4.1	RoboCalib	23
4.1.1	User Interface	24
4.1.2	RoboCalib's Calibration Class	26
4.2	Application's Architecture	27
4.2.1	Application's Network Architecture	28
4.2.2	The Robot's application	29
4.3	Implementation of the Calibration Tools	30
4.3.1	Camera Calibration Implementation	30
4.3.2	Hand-Eye Calibration Implementation	36
4.3.3	Stereo Triangulation Implementation	40
4.3.4	Work-space Calibration Implementation	44
5	Application's Validation	49
5.1	Robot Cell Work-space Characterization	49
5.2	Camera Calibration Results Validation	50
5.2.1	Lens Distortion Correction Evaluation	51
5.2.2	Camera Center Estimation Precision	54
5.2.3	Stereo Triangulation Accuracy using the Camera Calibrations	57
5.2.4	Camera Calibration Validation Conclusions	58
5.3	Hand-Eye Calibration Results Validation	58
5.3.1	Camera Transformation Estimation Precision	58
5.3.2	Camera Transformation Estimation Accuracy	59
5.3.3	Stereo Triangulation Accuracy using the Hand-Eye Calibrations	63
5.3.4	Hand-Eye Calibration Validation Conclusions	63
5.4	Practical Applications	64
5.4.1	Defining a <i>tooldata</i>	64
5.4.2	Defining a <i>workObject</i>	65
6	Conclusions	67
6.1	Achievements	68
6.2	Future Work	68
	Bibliography	71

A	User Interface	73
A.1	Starting Window	73
A.2	Load Calibration Configuration	73
A.3	New Calibration Configuration	74
A.4	Main Form	75
A.5	Manual Camera Calibration	76
A.6	Tool Point Triangulation	76

List of Tables

3.1	VisionPro 3D notation.	18
5.1	RMS of each <i>CogFindLine</i> tool.	52
5.2	Tool position result for the camera (2D) and for the stereo triangulation (3D) in mm.	65

List of Figures

2.1	ABB IRB 4600 and its respective axes. Courtesy of ABB.	7
2.2	Two work object coordinate frames. Courtesy of ABB.	9
2.3	Pinhole camera geometry (left) and the relationship between f and the mapped points on the focal plane.	11
2.4	Undistorted rectilinear image (left) and barrel distortion (right).	12
2.5	$AX = XB$ transformations illustration.	13
2.6	Stereo triangulation.	14
3.1	Cognex proprietary calibration plate. The coordinate system corresponds to the Phys3D space. Courtesy of Cognex.	18
3.2	The four mandatory tilted viewsets. The red axis is the mentioned axis of rotation. Courtesy of Cognex.	19
4.1	RoboCalib's main form with the corresponding calibration tools and results.	24
4.2	The UI state machine.	25
4.3	<i>Calibration</i> class's UML.	27
4.4	Top level network's architecture.	28
4.5	Camera calibration results for two cameras, displayed in RoboCalib's main form.	31
4.6	Automated camera calibration procedure.	31
4.7	Required camera transformation.	33
4.8	Hand-eye calibration results for two cameras, displayed in RoboCalib's main form.	36
4.9	Automated hand-eye calibration procedure.	37
4.10	Frames $\{a\}$ and $\{b\}$, where frame $\{b\}$ is the result of frame's $\{a\}$ rotation around all axes. v represents the distance between both frame's Z axes intersection with the calibration plate plane.	38
4.11	Stereo triangulation UI.	41
4.12	A needle position being calibrated. A vision job detects the needle position and the distance at which the needle is from the calibrated position.	44
5.1	Deployment view of the assembly robot cell.	50
5.2	<i>CogFindLine</i> result on a distorted line.	51
5.3	<i>Caliper</i> errors for distorted and undistorted lines.	52

5.4	Color progression on an image's edge.	53
5.5	Cognex's proprietary calibration plate.	53
5.6	Error across the FOV for distorted and undistorted images.	54
5.7	Measuring a part's corner distance to a reference dot.	55
5.8	Measurement inaccuracy when the camera is not centered with the object's corner.	56
5.9	Statistical results of the camera center estimation for different settings.	56
5.10	Accuracy of 3D measurements performed.	57
5.11	Statistical results of the camera transformation estimation for a constant focal length.	59
5.12	Object's displacement when it is rotated around a point that it is not its center.	60
5.13	Respective distance d , for different heights and for a calibrated and uncalibrated cameras.	61
5.14	Orientation accuracy test results for an uncalibrated camera.	62
5.15	Orientation accuracy test results for a calibrated camera.	62
5.16	Example of an usual patterned surface used in robotic applications.	66
6.1	Hand-Eye transformation for different zoom settings.	69
A.1	RoboCalib's first window.	73
A.2	RoboCalib's load calibration configuration window.	73
A.3	RoboCalib's new calibration configuration wizard.	74
A.4	RoboCalib's new calibration configuration wizard.	74
A.5	RoboCalib's new calibration configuration wizard.	75
A.6	RoboCalib's main form.	75
A.7	RoboCalib's manual camera calibration.	76
A.8	RoboCalib's tool point triangulation.	76

Glossary

AME	Applied Micro Electronics is an independent developer and manufacturer of high quality electronic products located in Eindhoven, The Netherlands.
CAD	Computer Aided Design.
CCD	Charge Coupled Device.
CNC	Computer Numerical Control.
DB	Data Base.
DOF	Degrees of Freedom.
FIFO	First In, First Out.
FOV	Field of View.
GenICam	Generic Interface for Cameras.
IP	Internet Protocol.
MP	Megapixel (one million pixels).
RMS	Root Mean Square.
ROI	Region of Interest.
SNR	Signal to Noise Ratio.
TCP	Tool Center Point.
TCP	Transmission Control Protocol.
UI	User Interface.
UML	Unified Modeling Language.

Chapter 1

Introduction

This first chapter presents an overview of the company, Applied Micro Electronics "AME" B.V., where this project has been developed, as well as an overview of the subject and the motivation behind it. In the end, the structure of this document is presented.

1.1 The Company

AME, based on Eindhoven (The Netherlands), is an independent developer and manufacturer of high quality electronic products driven by technology, striving for the best solution combining the disciplines of electrical, mechanical, software and industrial engineering. AME's vision is to develop a factory in the dark, in other words, the company has the ambition of automating all of its processes, as this is the road to increase productivity and product quality. For this ambition to be fulfilled and succeed in the most effective way, AME has decided to focus on generic automation. This approach focuses on developing generic applications that support a flexible way of working, opposed to a custom automation solution for each product that is manufactured. Developing and investing in these generic applications required higher costs and the purchase of more sophisticated robots, such as 6-axes industrial robots. Although being industrial robots known for their high repeatability, as reflected in the main industrial applications such as spot welding, they lack of accuracy. Nonetheless, with the increasing demand of automation, robots are the best solution in the direction of automation without sacrificing flexibility.

In order to achieve its goals, a relevant amount of AME's manufacturing processes rely upon the use of industrial robot cells for different applications: inline robotics, system assembly robotics and machine tending robotics. These applications have different requirements and thus use different machines and tools, suitable for each type of operation.

It is also imperative to mention, that as in for every process at AME, standardization is of extreme importance. AME has the philosophy of highly standardizing the organization in all different layers, from development processes, documents and structures to production machines. Therefore, all automation systems should be standardized and modular as well. In this case, where software will be developed for

a robot application, a computer application and vision applications, the same principle applies, meaning that this project must be compatible with all in-house robots and easily re-usable in the future.

1.2 Motivation

Being quality one of the main priorities at AME, all robotic applications must have an overall sub-millimetric accuracy. This is an extremely hard task for standard 6 degrees of freedom industrial robots, which are able to achieve an accuracy between 0.3 and 0.5mm when proper calibration is provided and robot positions are taught and fixed, according to Kubela et al. [1]. This drawback could be easily mitigated with the use of a CNC machine or any other machine with less axes designed for precise machining, instead of an industrial robot, but it would go against AME's philosophy of automation and universality, considering that an industrial robot is designed to cope and execute a wider scope of applications. Apart from the CNC machine option, it is also possible to manually teach the robot cell the exact required robot positions, which would cost a great amount of human labour, both setting it up and overseeing it and once again clash with AME's philosophy.

In the end, the solution that seems to solve this trade-off in the best way possible is the use of cameras in the robot cell's work-space. These can be used in a closed loop that informs the robot of its actual position and the distance to the targeted position.

This is the reason why most industrial robot cells, in the premises of AME, are integrated with vision systems, ensuring higher accuracy, precision, flexibility and a larger potential to increase the competitiveness in the market. Adding vision to the system improves general automation even more, as robots will not need teaching and will be able to carry on applications independently. The camera is used to guide the robots in pick and place, gluing and other applications that demand high levels of accuracy, precision and flexibility.

The camera can be mounted on its end-effector or detached from the robot cell, fixed and facing it. This project focuses mostly on cameras mounted on the end-effector given that this setup requires a more complex calibration procedure. This setup is also more versatile, since the camera is facing the manipulator's same direction, and therefore being able to do the same translational and rotational motions with it along the desired path.

The robot's IO can then trigger the camera's shutter, order the execution of a pre-defined image processing procedure over the retrieved image and process the results. These image processing procedures, also known as vision jobs, are not only useful for positioning but also for a wider set of applications, such as quality checks, estimating an object's position, calibration procedures, among others. When used for positioning, also known as vision guided motion, the robot cell requires a plane with precise physical known repetitive dimensions, e.g. a dot patterned plate, which is used to identify the robot's precise relative position over that plane. For this procedure to work the robot's camera must operate parallel and at a constant height with respect to that plane, and by doing so it can use the plate's

features to calculate its exact displacement with respect to its last position. If the result of the positioning vision job identified a difference between the actual and desired displacement, the robot moves the exact inverse amount of that difference. This procedure is repeated and iterated, over and over, until the difference between the actual and desired displacement is within an acceptable threshold.

Another important application in robotic applications is pick and place. In pick and place applications the robot must pick an object and place it somewhere else. To ensure the object is picked exactly at its center, the robot cell requires to know the precise position and orientation of the camera with respect to the robot's manipulator. The reason for such requirement, is that the only way for the robot to know where the object is positioned, is to firstly center it with its camera. If the object is not picked exactly at its center, while any translational movement adds no error to the procedure, any rotational movement of the same object will incur in a translational error.

The hand-eye calibration is the procedure that, after a sequence of calibration positions and acquired photos, is able to estimate an accurate pose - position and orientation - of the camera with respect to the robot's manipulator. Before executing the hand-eye calibration procedure, the camera's internal parameters, also known as camera intrinsics parameters (lens distortion, scale, skew and translation), must be known through a camera calibration procedure.

Once these sets of parameters are estimated, all vision dependent applications will produce results with better accuracy and precision. While the intrinsic parameters produce a corrected and undistorted image and the right image center, the extrinsic parameters allow the camera's optical axis to be more accurately oriented towards the perpendicular surface of operation and define the exact displacement to be made when moving the manipulator to the object's center after centering it with the camera.

Although being uncalibrated cameras one of the main reasons for the lack of precision during an application, influencing product quality, other issues might as well have a negative impact. This is the case of uncalibrated work-space tools. Repetitive applications tend to wear down or deform tools, e.g. glue needles, and the procedure of changing these same tools usually results in a misalignment or displacement from their original positional data, used by the robot. This is a substantial problem for highly precise applications but especially time consuming, since it requires an operator to manually align and place the tool in the correct position. This procedure is usually carried out with the robot's camera which is moved to a previously defined position facing the tool and informs the operator whether the tool is finally within acceptable boundaries or not.

1.3 Objectives and Deliverables

The lack of accuracy and efficiency of these robotic vision setups are the motivational reasons behind this project, considering that a proper and automatic calibration procedure of the robot cameras

would substantially improve the application's quality and the robot's efficiency. Therefore, three global objectives are set to be achieved in this thesis project:

- Calibration of the robot cell's cameras;
- Calibration of the tools in robot cell's workspace;
- Automation of these two latter objectives.

These are the three main goals and deliverables of this project, as these are the required core foundation for vision calibration in robotics. Nonetheless, this thesis project still proposes other deliverables, products of the latter ones. Therefore, a 3D measuring framework, using a stereo vision system, is to be developed as part of the scope of this project as well. This 3D measuring framework, essential for tasks such as quality control, uses two calibrated cameras in order to triangulate 3D points in their common field of view and measure the distance between them.

In order to accomplish these objectives, this project is to deliver an user friendly computer application with all these functionalities combined. Simultaneously, a robotic slave application is also to be developed in order to automate the process. It is also important that all of these functionalities are available independently, allowing their integration in future projects.

Even further, these objectives and deliverables must be achieved taking into account the company's guidelines, policies and processes. Consequently, any of the applications developed during the course of this project must be standard and suitable to any older or newer robot, and should not depend on a camera's type or setup.

1.4 Thesis Outline

This thesis dissertation is composed of six chapters. The first chapter, Introduction, presents the company where this project has been developed, as well as its guidelines, processes and its philosophy. It is an important chapter so that the reader understands the reasons behind some of the trade-offs and design decisions taken throughout this project. On the second chapter, Fundamental Concepts, the fundamental concepts behind industrial robot cells, industrial cameras and their respective calibration procedures are introduced. Several techniques and approaches in the scope of robot and camera calibration are presented in order for the reader to understand the complexity behind these procedures, as well as the available studies of the subjects. Chapter three, Supporting Software and Interfaces, is the chapter where the software frameworks used to support this project as well as their requirements to execute the calibration procedures are introduced. It also contains an introduction to the different protocols and interfaces used throughout the development of this project. Following, chapter four, Proposed Solution, introduces the proposed solution and its architecture, with respect to the problem of vision calibration for robotics, as well as its implementation. Chapter five, Application's Validation, proposes the required tests to validate this project's tools and the conclusion of these same tests, with respect to

accuracy and precision. Last chapter, Conclusions, summarizes the respective work done and its conclusions. Still in chapter six, future prospections of the implementation of this project in other projects are presented.

Chapter 2

Fundamental Concepts

This chapter introduces the important fundamental concepts required to be acquainted with before addressing this project's proposed solution. An overview and a detailed description of technical references of industrial robotic applications is first given, as to understand the possibilities and limitations of a robot cell. The other sections of this chapter present the camera model used in this project and addresses the state-of-the-art on camera and robot calibration.

2.1 Robot Coordinate Systems

An industrial robot commonly consists of a mechanical arm with several links and axes, as well as a computer controller responsible to govern its positions and movements. Most industrial robots have 6 DOF, three in translation and three in rotation, and therefore, but not always, six single-axis rotational joints, as the robot illustrated in figure 2.1.

For this same reason, robots use multiple coordinate frames assigned not only to each axis but also to each tool and work objects. Since these transformations are comprised of both translational and rotational displacements there is a need for a unified mathematical description of these transformations.

Let the coordinates of a point, relative to a frame $\{b\}$, rotated and translated with respect to a frame $\{a\}$, be noted as:

$${}_a p = {}^b_a R {}_b p + {}_a p^{a,b}, \quad (2.1)$$

where ${}^b_a R$ represents the rotation matrix of a frame $\{b\}$ with respect to frame $\{a\}$.

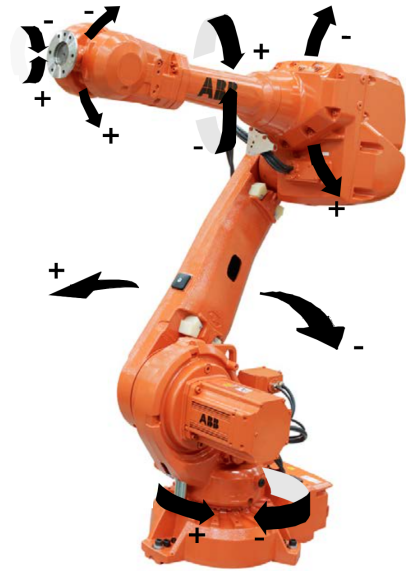


Figure 2.1: ABB IRB 4600 and its respective axes. Courtesy of ABB.

Equation 2.1 can then be noted in the form of an homogeneous transformation matrix. This transformation matrix represents the translation and rotation of a frame $\{b\}$ whose origin is ${}_ap^{a,b}$ displaced from frame $\{a\}$, and whose orientation is given the rotation matrix ${}_a^bR$.

$${}_a^bT = \begin{bmatrix} {}_a^bR & {}_ap^{a,b} \\ 0^T & 1 \end{bmatrix} \quad (2.2)$$

With this representation of a transformation is now easier to calculate the position of a point p with respect to frame $\{a\}$, ${}_ap$, knowing the point's position with respect to frame $\{b\}$, ${}_bp$:

$$\begin{bmatrix} {}_ap \\ 1 \end{bmatrix} = {}_a^bT \begin{bmatrix} {}_bp \\ 1 \end{bmatrix} \quad (2.3)$$

2.1.1 Quaternions

Both Euler angles and quaternions are used in robotic applications to define the orientation of an object. Although being Euler angles much more intuitive, they are limited by the gimbal lock phenomenon, which prevents them to define orientations with angles close to 90° . Even further, when compared to rotation matrices, quaternions are much more compact, numerically stable and more mathematical efficient.

Quaternions, invented by William Hamilton, are then a four-element vector used to encode any rotation in a 3D coordinate system and written as $q = w + xi + yj + zk$, where w, x, y and z are scalars but i, j and k are imaginary numbers. Therefore, the following conditions apply:

$$i^2 = -1, j^2 = -1, k^2 = -1,$$

$$ij = k, jk = i, ki = j, ji = -k, kj = -i, ik = -j,$$

$$|q|^2 = s^2 + x^2 + y^2 + z^2$$

Quaternions will be important later since the robot estimates its orientation in quaternions and not in Euler angles. Moreover, defining the orientation of objects and tools in the robot's application is done using quaternions, and therefore the calibration results are made available in quaternions. In this project, the w, x, y and z scalars are represented as $q1, q2, q3$ and $q4$ respectively, since this is the notation used in robot applications.

2.1.2 Work Object Data

A work object consists of a coordinate system associated with a particular object the robot processes or makes use of. Thus, work objects are both used for associating a coordinate system to a specific object, simplifying its mastering, or to define a coordinate system to a stationary holder where other objects are processed, e.g. a table. Once a work object has been defined, tools become easier to

manipulate over objects or surfaces, now that the tool's path is moved always with respect to that specific work object's coordinate system, independent of its orientation or position. In order to define the work object data, denoted as *wobjdata*, a translation and rotation, with respect to the robot's frame, are required.

Figure 2.2 depicts an usual setup of two work objects, being one the coordinate system of a surface where other objects, represented by their own work object coordinate system, are processed or manipulated. At the same time, there is another *wobjdata* which is not depicted but always defined, which represents the center of the robot's frame, *wObj0*.

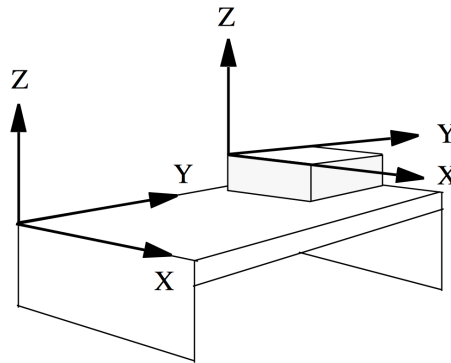


Figure 2.2: Two work object coordinate frames. Courtesy of ABB.

2.1.3 Tool Data

Tool data, denoted as *tooldata*, is used to describe the characteristics of a tool, e.g. a welding gun or a gripper, but more importantly, it defines the exact tool center point, TCP, of any attached or external stationary tool. The *tooldata* is then used to create a tool's coordinate system, centered on the TCP and oriented accordingly. Defining *tooldata* with precision is of extreme importance, since all movements of the robot are of a tool with respect to a work object.

The cameras attached to the end-effector are also represented as a *tooldata* variable, with a defined displacement and orientation from the robot's flange, also known as *tool0*.

2.2 Robot Calibration

Although not being the main focus of this project, robot calibration is still an essential subject in the field of camera calibration for robotics. In order to estimate the position of an object with a camera, one of the objectives of this project, the exact transformation between the robot's hand and the its base has to be known. If a robot cell is not calibrated, and thus its motions not behaving as expected, camera calibration accuracy itself will be deeply affected.

Robot calibration is then defined as the process of improving the robot's accuracy by changing the

robot's positioning software, rather than changing its physical design. More concretely, it is the process of identifying a better relationship between its joint readings and the actual 3D position of the end-effector.

Robot calibration differs widely in its complexity. While some authors consider only the joints readings, others involve changes in the kinematic or dynamic model of the robot. In order to facilitate differentiating different types of calibration, Roth et al. [2] separated calibration procedures into levels.

In level 1 robot calibration, a correct relationship between joint readings and the robots actual position is defined. Usually, level 1 only involves the calibration of joint sensors' mechanisms. Level 2 includes the whole robot's kinematic model, including its geometry and joint-angle relationship [3, 4]. Level 3, known as non-geometric calibration, takes into account non-kinematic errors such as joint compliance, friction, and clearance, as well as link compliance.

Independently of its level, four steps are usually taken in robot calibration: modeling, measurement, identification and compensation.

Quantitatively, and about some sources of errors, Renders et al. [5] and Becquet [6] reported flexibility in the joints and links to be responsible up to 10% of the position and orientation errors. Backlash, caused by gaps in the gears, contributes approximately up to 1% of the global error, while temperature is only responsible for 0.1%.

2.3 Camera Model

Using cameras to map the 3D world into a 2D image requires the previous understanding of camera models in order to comprehend the need for camera calibration. Therefore, this section provides an insight on the pinhole camera model, a mathematical relationship between a 3D point and its projection onto the camera's image plane, which does not account for geometric distortion or other aberrations.

2.3.1 Pinhole Model

In the pinhole model, the center of projection, also known as optical center, is the origin of an Euclidean coordinate system and the focal plane the plane $Z = f$. The Z axis, the line originated in the optical center and perpendicular to the focal plane is denoted by optical axis or principal axis, and the point where this axis meets the focal plane is called the principal point, denoted by p .

Let $T = [X, Y, Z]^T$ be a point in space, mapped on the focal plane where a line going through T and the center of projection intersects it. This point of intersection is then $(fX/Z, fY/Z, f)^T$, thus the central projection mapping from world (Euclidean 3-space \mathbb{R}^3) to image coordinates (Euclidean 2-space \mathbb{R}^2) is described by equation 2.4.

$$(X, Y, Z)^T \rightarrow (fX/Z, fY/Z)^T \quad (2.4)$$

Equation 2.4 assumes the origin of the coordinates in the focal plane to be at the principal point. This is not usually the case and therefore the equation should be modified to:

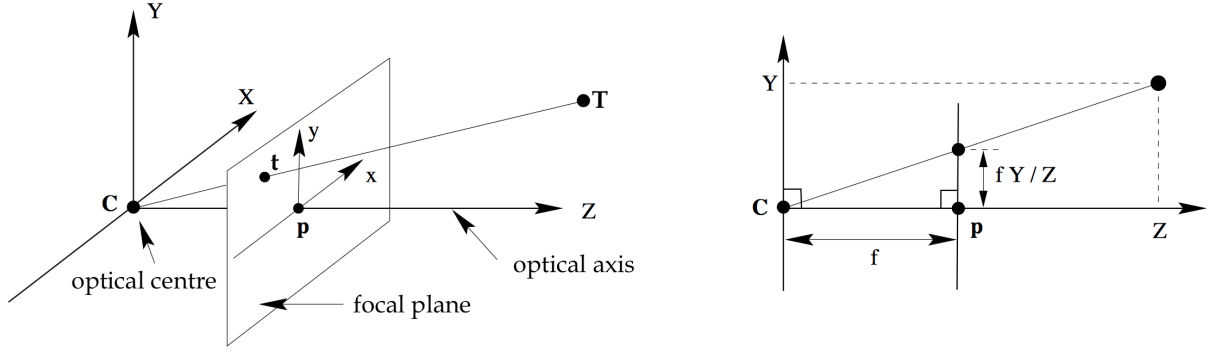


Figure 2.3: Pinhole camera geometry (left) and the relationship between f and the mapped points on the focal plane.

$$(X, Y, Z)^T \rightarrow (fX/Z + p_x, fY/Z + p_y)^T \quad (2.5)$$

where $(p_x, p_y)^T$ are the coordinates of the principal point. This transformation can also be easily expressed as a linear mapping between their homogeneous coordinates:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.6)$$

where the camera matrix for the pinhole model $P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$

Since most charge coupled device (CCD) cameras do not scale equally in both axial distances and having the camera some non-squared pixels, a scaling factor needs to be included. Let m_x and m_y be the number of pixels per unit in the x and y direction respectively, then $\alpha_x = fm_x$ and $\alpha_y = fm_y$.

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

K is the calibration matrix parametrized by [7], where x_0 and y_0 equal to $m_x p_x$ and $m_y p_y$, respectively, and where s represents axis skew caused by shear distortion.

These five parameters form the intrinsic parameters of a camera, which describe its linear behavior. Several methods used in the industry, to estimate these parameters, are presented in the section 2.4.

2.3.2 Lens Distortion

As stated by Zhuang and Roth [8], lens distortion is a composition of radial and tangential distortion, but, and as agreed among most authors including [9–11], lens distortion is totally dominated by radial

distortion.

Radial distortion is the optical aberration that deforms the image as a whole, rendering straight lines as curved lines, mostly caused by an incorrect radial curvature of the lens. Usually, for industrial camera lenses, image magnification decreases with the distance from the optical center, creating a barrel shaped effect on the image, and therefore being known as barrel distortion.

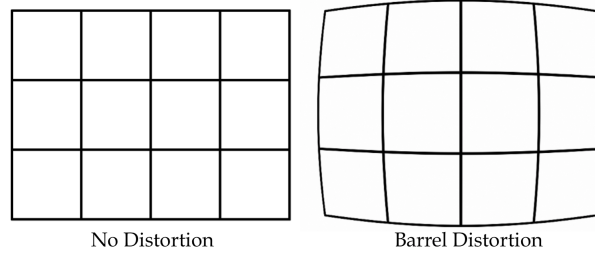


Figure 2.4: Undistorted rectilinear image (left) and barrel distortion (right).

Most of the recent radial distortion related works use the radial distortion model proposed by Slama et al. [12], where radial distortion is modeled by equation 2.8.

$$F(r) = rf(r) = r(1 + k_1r^2 + k_2r^4 + k_3r^6 + \dots), \quad (2.8)$$

where k_1, k_2, k_3, \dots are the distortion coefficients (negative for barrel distortion) and $r^2 = x^2 + y^2$. Tsai experiments show that for industrial machine vision applications only the first term needs to be considered.

Due to the non-linear nature of radial distortion, it cannot be parametrized into the pinhole model. Nonetheless, a transformation between a distorted coordinate and its corresponding undistorted ideal coordinate can be performed. Take transformation 2.4, where the undistorted point $t = (x_u, y_u) = (fX/Z, fY/Z)^T$ in the focal plane corresponded to point T of image 2.3. If distortion is not to be ignored in the pinhole model, one can write the undistorted coordinates with respect to the distorted ones:

$$(x_u, y_u)^T = (x_d + d_x, y_d + d_y)^T, \quad (2.9)$$

where $(x_d + d_x, y_d + d_y)^T$ are the real distorted coordinates on the focal plane and

$$d_x = x_d(k_1r^2) \quad \text{and} \quad d_y = y_d(k_1r^2) \quad (2.10)$$

$$r^2 = x_d^2 + y_d^2 \quad (2.11)$$

2.4 Camera Calibration

Camera calibration is the essential process of determining the internal camera geometrical and optical characteristics (intrinsic parameters) and the 3D transformation (position and orientation) between the camera and the world's coordinate system (extrinsic parameters).

The intrinsic parameters of a camera, internal and optical characteristics of a camera, are required to model the way light travels through the lenses up to the plane of the sensor. These include the five linear parameters of matrix K (focal lengths, skew and translation), equation 2.7, and the coefficients of distortion.

The extrinsic parameters are the transformation from the camera to a world's coordinate system. If the camera is fixed, then this is a simple transformation camera-to-world, but if the camera is attached to a movable robot than two transformations are required, camera-to-robot and robot-to world. The process of estimating the transformation camera-to-robot is called hand-eye calibration.

Camera calibration techniques can be classified into two categories: photogrammetric calibration and self-calibration. In photogrammetric calibration [9, 11, 13, 14] an object whose geometry in 3D is precisely known is used, while in self-calibration techniques [15–17] no object is used. Instead, simply by moving a camera in a static scene two constraints can be made in the intrinsics equation. While photogrammetric is more stable and efficient, self-calibration is more flexible, but less reliable.

It is also important to state that a camera calibration heavily depends on the camera's lenses. Therefore, any adjustment of zoom (focal length), focus, aperture or any other optical aspect of the lenses will result in an invalidated camera calibration and different intrinsic and extrinsic parameters.

2.5 Hand-Eye Calibration

Hand-eye calibration is the process required to obtain the position and orientation of the camera with respect to the robot's hand. This calibration procedure is of extreme importance, and required when the robot bases its motion on the information acquired by the camera. For many applications, where high accuracy is not required, this transformation can be estimated from CAD drawings. This method is not only unreliable due to the uncertainty of the optical center in the CAD drawings, but also due to the parameters of geometric models which can be incorrect due to measurement, machining, or assembly variances.

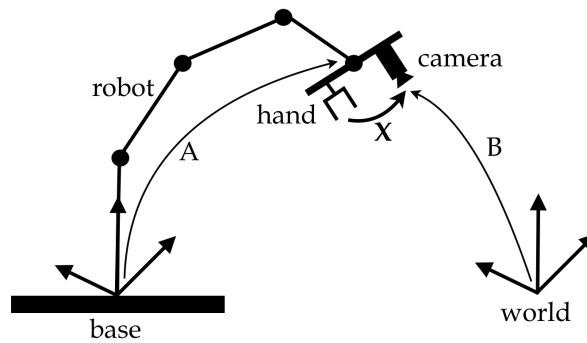


Figure 2.5: $AX = XB$ transformations illustration.

Most authors [18–21] formulate this problem with the following homogeneous matrix equation:

$$AX = XB, \quad (2.12)$$

where A is perceived as the homogeneous transformation between the robot base and the hand, B as the perceived motion of the camera with respect to a fixed calibrated target. These two are then known transformations, since the motion of the camera is extracted from the sensor and the transformation of the robot's hand given by the values of the encoders of the robot.

Shiu and Ahmad [20] were the firsts to formulate this problem as $AX = XB$, but their solution would double the linear system, every time a new frame was added. Later, Tsai and Lenz [21] formulated this problem with a fixed size system.

2.6 Stereo Triangulation

Stereo triangulation refers to the process of determining a point in 3D space knowing its projection into two images. Triangulation needs therefore two different cameras, with a shared FOV, or a robot which always acquires two images from two points of view, separated by the same transformation. While the latter is considered much more inaccurate, due to the robot's positioning errors, is still a viable way to triangulate without the need for an additional camera. For both cases, the camera's parameters must be known, including the transformation between the camera and the robot, if relating the resulting point with the robot's base is desirable.

If two cameras are used, and their intrinsic and extrinsic parameters known, then triangulating a point

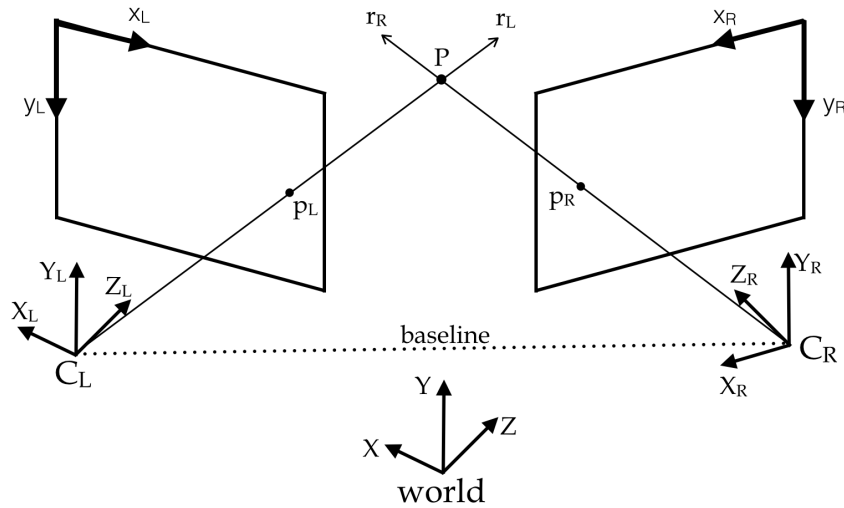


Figure 2.6: Stereo triangulation.

P is the process of intersecting two camera rays, r_R and r_L , which are the rays that go through point P projections. One of the critical issues of stereo triangulation is the problem of stereo correspondence: p_R and p_L projections must correspond exactly to the same 3D point, P .

The analytical relationship, between a 3D point and its projection, for camera i , is then given by the equation 2.13.

$$k \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha_{xi} & s_i & x_{0i} & 0 \\ 0 & \alpha_{yi} & y_{0i} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Camera } i \text{ intrinsic parameters}} \underbrace{\begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix}}_{\text{Camera } i \text{ extrinsic parameters}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \quad (2.13)$$

Camera i matrix D_i

where R_i and t_i are the rotation and translation of the homogeneous transformation which relates the camera's coordinates to the world's coordinates.

Stereo triangulation aims to evaluate the intersection $(X, Y, Z)^T$ between the two rays by solving the linear system obtained by stacking equation 2.13 for the left and right camera. In order to it, a well-known technique, linear triangulation [22, 23], is used to solve the following linear system with a least-squares technique:

$$\begin{bmatrix} x_L d_{3L}^T - d_{1L}^T \\ y_L d_{3L}^T - d_{2L}^T \\ x_R d_{3R}^T - d_{1R}^T \\ y_R d_{3R}^T - d_{2R}^T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad (2.14)$$

where the generic term d_{ni}^T is the n -th row of the camera matrix D_i .

With respect to stereo triangulation accuracy, the work of Schreive [24] is cited and analyzed. Schreive estimated the relationships between the cameras' distance, cameras' angle, focal length, camera resolution and the measurements accuracy.

He concluded that the effect of distance, from the camera center to the target, shows a strong linear trend, with error growing linearly with the distance. For the angle between cameras, the error has an even stronger influence on accuracy, rising asymptotically towards 0° and 180° , while achieving an optimal accuracy at 90° . The focal length shows a strong non-linear influence on accuracy, rising asymptotically towards 0 mm. This is due to the fact that the FOV decreases when the focal length increases, thereby decreasing the pixel quantization error. For the camera resolution relationship, and as expected, smaller pixels achieve higher accuracies.

Chapter 3

Supporting Software and Interfaces

This chapter introduces the machine vision software framework used to support this project. It gives an insight on its main functionalities and tools, used in common industrial applications, as well as an introduction of the required procedures of camera and hand-eye calibration. Moreover, relevant interfaces and protocols to this project are presented as well.

3.1 VisionPro

VisionPro is a product from Cognex, which offers a vast set of machine vision tools. This will be the framework used to develop this project, since it is used throughout other projects at AME. Even further, it offers camera and hand-eye calibration tools, as well as measuring, identification and finding tools. VisionPro offers an UI which allows an user to interface different vision tools and create a global solution. This global solution, known as a vision job, acquires an image from the camera, runs it through the vision tools and outputs the result. For more complex vision jobs, there is also a way to create scripting tools. Scripting allows an user to write code in *C#* and create inputs and outputs that interface with the other tools.

Usually VisionPro is used to create quality checks or to find certain features, using its vision tools. Although being this its main purpose, 3D vision tools are also offered but cannot be used through the UI due to its complexity. Instead of tools that can be used in the UI, VisionPro offers them as classes and methods. This section details the classes and methods made available by VisionPro, for the purpose of calibration, and their requirements. The interface between the robot cell and the way vision jobs are executed is also explained.

3.1.1 Useful Definitions

VisionPro uses its own terms and definitions with respect to coordinate systems, therefore, and in order to have a unified and clear notation of the different coordinate systems notation, table 3.1 presents the terms to be used throughout this dissertation, as well as their description.

Term	Definition
3D Pose	Position and orientation (pose) of a 3D coordinate system - comprises 6DOF.
3D Position	The position of a 3D point with respect to a 3D coordinate system.
3D Ray	Geometric object defined by a 3D position (the starting point of the ray) and orientation. A ray extends infinitely from its origin.
3D-Calibrated Camera	A camera for which a 3D calibration (both extrinsic and intrinsic) has been computed.
Raw2D Space	Left-handed 2D coordinate space based on the pixels in an acquired image.
Camera3D Space	A right-handed 3D coordinate space with its origin at the camera's optical convergence point, X- and Y-axes that are approximately parallel to and oriented in the same direction as the Raw2D coordinate system X- and Y-axes, and a Z-axis that extends along the optical axis away from the camera
Camera2D Space	The plane at $Z=1$ of Camera3D Space. When Camera2D space is viewed from the camera ($Z < 1$ and in the direction of the Camera3D positive Z axis) then Camera2D space appears as a left-handed 2D coordinate system. When Camera2D space is viewed in the direction of the Camera3D negative Z axis from a point in front of the camera (where $Z > 1$), then Camera2D Space appears as a right-handed 2D coordinate system.
Phys3D Space	A right-handed 3D coordinate space initially defined by the calibration object. The camera's extrinsic parameters are with respect to this coordinate system.
Hand3D Space	A right-handed 3D coordinate space defined by the end-effector on a robot. The position of the robot hand is reported by the robot controller as the pose of Hand3D space in RobotBase3D space.
RobotBase3D Space	A right-handed 3D coordinate space defined by the robot's base.

Table 3.1: VisionPro 3D notation.

3.1.2 Calibration Plate

VisionPro calibration is classified as photogrammetric, where, as explained in section 2.4, an object with precise known 3D dimensions is used to create a definition of the world's coordinate system. Cognex has its own proprietary calibration plates, Figure 3.1, which are to be used.

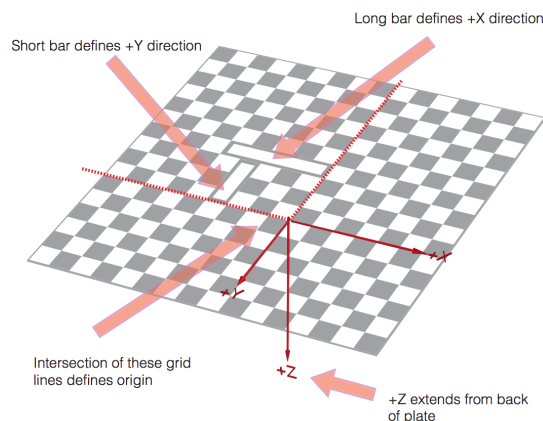


Figure 3.1: Cognex proprietary calibration plate. The coordinate system corresponds to the Phys3D space. Courtesy of Cognex.

3.1.3 Camera Calibration

VisionPro can handle the 3D calibration of one or more cameras simultaneously. In order to calibrate the cameras, VisionPro requires several different specific viewsets of the calibration plate. These are then used to extract the calibration plate's features and define the Phys3D space.

Five viewsets are mandatory and have to be acquired in order to calibrate the cameras. For better accuracy is even recommended to acquire four additional viewsets. For a single-camera calibration, all viewsets must include a well-focused view of all calibration features, but for multiple-camera calibration if some images do not include some calibration features, it is still possible to estimate an accurate calibration.

The first five mandatory viewsets are a set of four tilted viewsets and one world origin viewset. For the four tilted viewsets the calibration plate should be faced towards the cameras, centered with a rotational axis defined by the average of the optical axes of the cameras and with an approximate tilt of 20° . Each viewset should be acquired once the calibration plate has been rotated about 90° as Figure 3.2 suggests.

For the world origin viewset, the viewset required to define the Phys3D space as in Figure 3.1, the calibration plate should lay perpendicular to the axis of rotation previously referred.

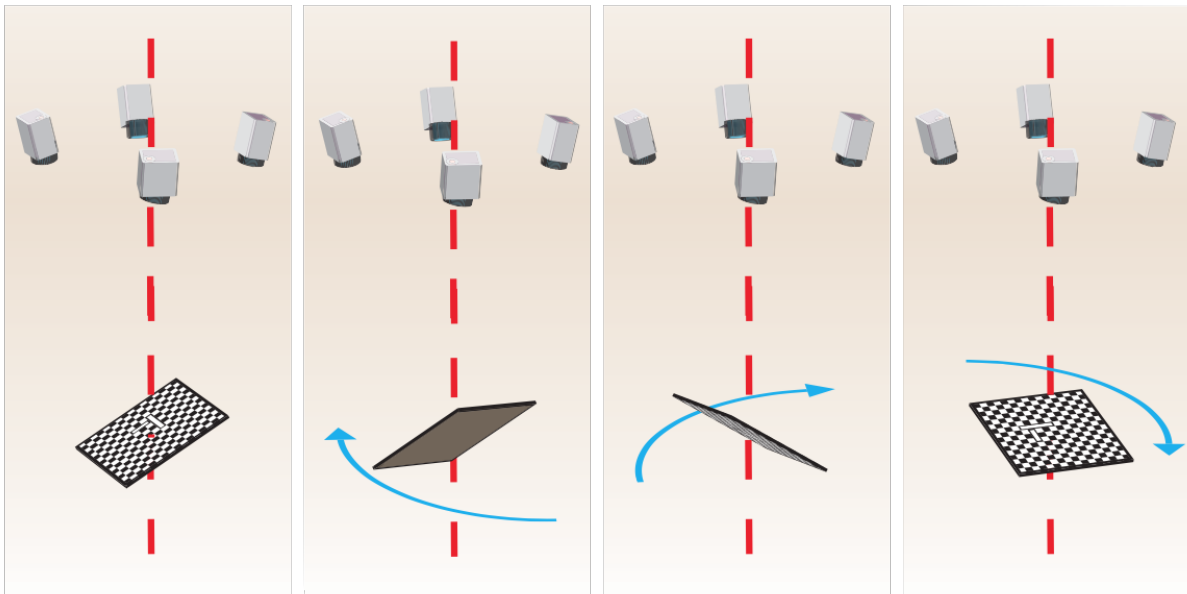


Figure 3.2: The four mandatory tilted viewsets. The red axis is the mentioned axis of rotation. Courtesy of Cognex.

As mentioned before, additional four viewsets are recommended. These are referenced as the elevated viewsets and acquired with the calibration plate precisely normal to the cameras' axis (red axis in Figure 3.1). Each viewset should be acquired at an evenly spaced height, above and below the height of the world origin viewset.

Once all viewsets have been acquired the correspondence pair features can be computed. For this

purpose, VisionPro makes the class *Cog3DCheckerboardFeatureExtractor* available. This class contains the method which creates a vector of correspondences between 3D plate locations and 2D image locations. Once this method has been executed for all viewsets of all cameras the class *Cog3DCameraCalibrator* can be used to compute the camera calibration.

The returned object is a list of *Cog3DCalibrationResult* which contains the intrinsic and extrinsic parameters of each camera. Even further, this object contains the computed residual errors of the least squares minimization procedure used to calibrate the cameras. The residual errors statistics come both in 3D (physical units) and 2D (pixels).

3.1.4 Hand-Eye Calibration

VisionPro's framework also makes hand-eye calibration tools available, for one or multiple cameras. These work similarly to the same way the ones of section 3.1.3 do: several viewsets are acquired and calibration is computed. For these viewsets the calibration plate is supposed to be fixed and motionless, while the robot moves to different stations (poses) and acquires different viewsets. The accuracy of the hand-eye calibration heavily depends on the motion between the stations, and therefore they should include the largest possible rotation (R_x , R_y and R_z) that keeps the calibration plate in the FOV of the cameras. Every time the robot moves to a new station it should acquire the robot's *Hand3D* pose with respect to *RobotBase3D*. In order to compute a successful hand-eye calibration, the robot should move to at least nine different stations.

Once the viewsets and robot poses of all stations have been acquired, the correspondence between 3D plate locations and 2D image locations needs to be estimated using the class *Cog3DCheckerboardFeatureExtractor*. *Cog3DHandEyeCalibrator* class uses then these correspondences, the corresponding *Hand3D* poses and the 3D camera calibration intrinsics to compute the hand-eye calibration for each camera.

Once hand-eye calibration has been computed, the returned object *Cog3DHandEyeCalibrationResult* contains the homogeneous transformation from the *Hand3D* to the *Camera3D*.

3.2 RobotStudio and RAPID

Being all robots in-house from ABB, as in compliance with AME's philosophy of standardization, a unique robot programming environment and language are used as well.

RobotStudio is the robot programming environment adopted to create, simulate and deploy robotic projects. It is built on the top of ABB *VirtualController*, an exact copy of the real software that runs on the robots.

It is then possible to program and debug robot applications using the RAPID high-level programming language. RAPID includes several programming functionalities such as: procedures, trap routines, functions, arithmetic and logical expressions, error handling and multi-tasking.

3.3 RoboVision

RoboVision is a computer application developed at AME which works as an interface between the robot and VisionPro applications (vision jobs). When running, it allows the robot to remotely connect to it using TCP/IP and exchange request messages. With a connection established the robot can then request RoboVision to run a specific VisionPro application from the database. After executing the vision job, RoboVision saves the input and output images as well as its results, in the database. These results and images can be immediately requested by the robot from RoboVision, through their established connection.

3.4 GigE Vision

All cameras are connected to an application using the GigE interface standard, the standard used in high-performance industrial cameras for transmitting high-speed video and related control data over ethernet networks. The benefits of GigE, when compared to other interfaces like USB, go from high bandwidth transfers up to 125 MB/s, the use of low cost CAT5 or CAT6 cables, uncompromised data transfer up to 100 meters in length and its independence of frame grabbers.

Establishing a connection with a GigE camera is also simpler, when compared with other interfaces. It is only required for the camera to be connected to a sub-net of the local area network. Once the desired video mode and video format have been chosen, an acquisition FIFO is created. When the user requires an image from the camera, the FIFO will supply it. Even further, the GigE interface allows the user to get or set features programmatically. A feature is a camera setting defined in the GenICam standard or by the camera manufacturer, such as zoom, focus, aperture or exposure.

Chapter 4

Proposed Solution

This chapter describes this project's solution, from its architecture to its implementation. Regarding the project's architecture, the conceptual models that define its solution's structure and behaviour are explained thoroughly. Following, these models' implementation and technical aspects behind it are described.

4.1 RoboCalib

RoboCalib is the name given to the main application developed under this project. It envisions to be an application suite where all calibration and related tools can be found and automatically executed. Its main focus is to facilitate the execution of calibration procedures, as well as to make all calibration results available to anyone in the network. At the same time, all results can be exported in their specific format and directly integrated in external applications. Although being all procedures automated, RoboCalib still makes it possible for a user to request a manual override of a procedure, an option suitable for situations where the robot cell is specifically constrained and does not have enough space to move freely. The manual procedures offer the user a thoroughly detailed explanation guide of the procedure, and how to get the best results from it.

RoboCalib application starts by requesting a user to load or create a new configuration, comprised of the cameras to be used, as well as its settings. The settings comprise not only the cameras' internal parameters (zoom, focus, aperture, etc.) as well as the information regarding the approximate position of the camera with respect to the robot (if the camera is attached to it), the distance at which the camera is from the calibration plate and other important details. Once the user has chosen the appropriate configuration, he will have access to the pertinent calibration tools. Every time a specific tool is executed, its results are displayed in the UI and saved accordingly in the network. The UI itself is controlled by a state machine which facilitates handling the availability of each interface resource, such as buttons or labels.

RoboCalib does not execute the calibration procedures itself (with one exception), instead, it com-

mands their execution to the robot's application and awaits for the results. The robot's application acts then as a slave application, once a TCP/IP connection has been established the robot application awaits indeterminately for a RoboCalib's command. This interaction is meticulously explained in section 4.2.

Regarding the technical aspects of the application, RoboCalib is an event-driven Windows Form application supported by Microsoft .NET Framework 4.0, written in C#. Although existing newer and more recent releases, Microsoft .NET Framework 4.0 is used in order to be compatible with VisionPro's framework.

4.1.1 User Interface

RoboCalib's UI is comprised of distinct windows, each one designed to meet a different purpose. The first form loaded, once the program is executed, is the new or load form. While the load form basically offers the user all the previously saved calibration configurations, the new form takes the user through a wizard to help him set up everything easily, including camera settings such as zoom, focus, aperture and exposure. Once a camera is selected a live video is prompted in order to facilitate the selection of the right settings for a specific application.

The screenshot displays the RoboCalib application window with the following sections:

- Menu:** Configuration, Calibration, Triangulation, Work-space Calibration.
- Configurations:**
 - Camera 1:** GigE Vision: The Imaging Source Europe GmbH: DFK Z12GP031. Settings: Zoom: 60, Focus: 365, Iris: 200, Exposure: 35.
 - Camera 2:** GigE Vision: Basler: acA1600-20gc. Settings: Zoom: 0, Focus: 0, Iris: 0, Exposure: 50.
- Calibration:**
 - Tile Size: 1,5875; Approximate Distance to Target: 260; Robot: ABB IRB 4600 - Assembly; Moving Automatically: ☒.
 - Camera(s) Calibration:** Height Poses Displacement. Calibrate button.
 - Results:**
 - Camera 1 Translation: [X, Y] = [1306.7303, 979.2319] Quality: Good.
 - Camera 2 Translation: [X, Y] = [802.8069, 610.8561] Quality: Good.
- Hand-Eye Calibration:**
 - Number of Tiles in X: 40; Number of Tiles in Y: 30. Calibrate button.
 - Results:**
 - Camera 1 Translation: [X, Y, Z] = [-185.9937, -1.7765, 27.5870] Quality: Good. Rotation: [q1, q2, q3, q4] = [0.7075307, -0.0026371, 0.0012673, -0.7066765]
 - Camera 2 Translation: [X, Y, Z] = [-172.4849, -85.5183, 120.0974] Quality: Good. Rotation: [q1, q2, q3, q4] = [0.7018494, -0.1254321, -0.1330082, -0.6884642]
- Tool Calibration:** Triangulate Point button.
- Automatic Work-space Calibration:** Create New Path, Execute Path, Results buttons.
- Status:** Robot not Connected, Camera(s) not Calibrated, Hand-Eye not Calibrated.
- Footer:** IP: 10.0.150.132 Port 8210

Figure 4.1: RoboCalib's main form with the corresponding calibration tools and results.

Both forms, once finished, lead the user to the main form. The main form, and has explained before, displays all the prompted settings as well as the results of the performed calibrations. More importantly, it makes available all calibration tools to the user. These tools might be available, or not, depending of the user's configuration and previously performed calibrations. For cameras not attached to a robot, the hand-eye and work-space calibration procedures are not available, since these require the robot's motion. At the same time, some tools might not be available because they require previously performed calibrations, this is the example of the triangulation tool, hand-eye and work-space calibration procedures, which all require camera calibration to be executed before-hand.

In order to facilitate UI updates after each procedure a state machine has been developed. This state machine is responsible for updating the correct UI elements once a relevant event has occurred. This means enabling/disabling buttons, changing labels and updating as well as saving results. Therefore, each time a calibration procedure is executed, the state machine updates its state and acts accordingly. The state machine is comprised of seven different states. Four states handle the events after a calibration procedure, remaining idle until some user input is received, and the remaining three others which disable all tools while a certain calibration procedure is being executed. Although having the same functionality, these three last states were given different separate states, instead of comprising them in a general state, for a matter of simplicity and flexibility, so that in the future different functionalities can be implemented in each one.

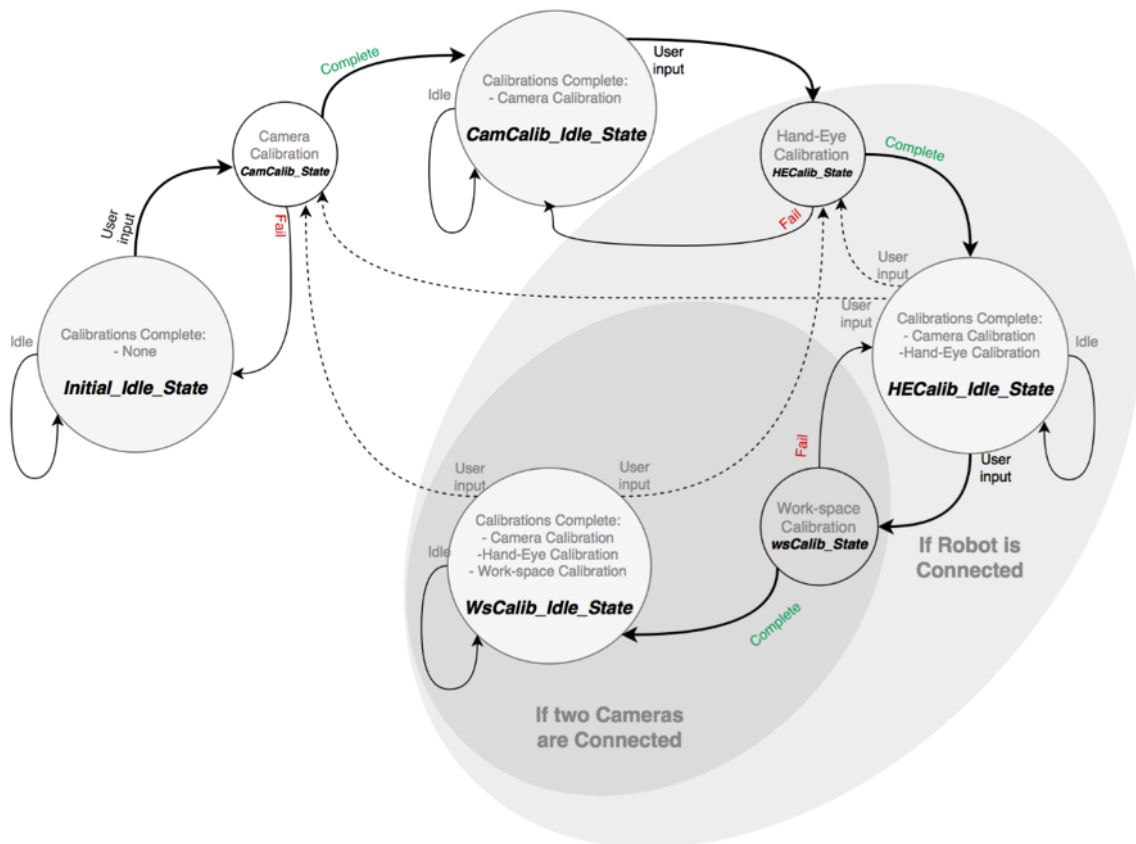


Figure 4.2: The UI state machine.

Figure 4.2 depicts the UI state machine and its state transitions, starting in the state *Initial_Idle_State*. The bold transition lines represent the application's main path, which are the steps usually taken in order to fully calibrate a robot's camera, nonetheless, the user may decide to repeat a previously executed calibration procedure, represented by the intermittent transition lines. When the application's state is one of the idle states, its current state can only be change through a user's input, but for the calibration states the state changes when the procedure is complete. If the calibration procedure was successful the application moves to next idle state, but if the procedure failed, the state is reverted to the last idle state.

Since some applications, where camera calibration is required, do not make use of a robot cell, or since some robotic applications only use one camera, some states can only be accessed if the application fulfils certain prerequisites. These restricted states are the ones represented in Figure 4.2 within a grey area.

4.1.2 RoboCalib's Calibration Class

RoboCalib's *Calibration* class is the structure that holds all the information prompted into RoboCalib, as well as all the information processed by it. The complete set of *Calibration* objects, organized in a list and stored in the network, is considered to be RoboCalib's database. It contains the result objects returned by the calibration procedures, along with all the configurations inputted by the user. It is an essential class to organize all the fundamental data, making its storage and repossession easier.

When a user wishes to create a new calibration set-up a new *Calibration* object is created. The object is constructed with the initial information provided by the user and is updated every time new calibration results are available. When the user decides to save it, RoboCalib retrieves a serialized list of previously saved *Calibration* objects from the network, and appends the new object to it.

The *Calibration* object is described by a name and a unique identification number. It contains one or two *Camera* objects, a configuration object and an object with the work-space tools information. Figure 4.3 is a representation of this class's UML.

The *Camera* object holds all the information with respect to the camera's properties (name, serial identification, video mode, zoom, focus, iris, exposure, description and MAC address) as well as a reference to each object produced by the calibration procedures. The object is created during the camera set-up and its properties cannot be changed in the future. The *CameraCalibration* and *HandEyeCalibration* classes, hold the important objects returned by the calibration procedures. These are VisionPro's objects which hold the following items:

- ***Cog3DCameraCalibrationResults***: Holds the camera calibration procedure's results and error related statistics.
- ***Cog3DCameraCalibration***: VisionPro's camera calibration object. To be used with other VisionPro procedures that require a camera calibration object, such as the hand-eye calibration or to correct an image's distortion.

- **Cog3DCameraCalibrationIntrinsics:** Camera intrinsics parameters, such as skew, translation, scale and distortion coefficients. The class also includes methods to map *Camera2D* points in *Raw2D* and vice-versa.
- **Cog3DHandEyeCalibrationResults:** Holds the hand-eye procedure results. These results include the procedure's error statistics.
- **Cog3DHandEyeCalibration:** VisionPro's hand-eye calibration, which holds the *Hand3D* to *Camera3D* transformation.

The *Configuration* class is used to save the overall inputted information by the user. Thus, it contains the size of the calibration plate tiles, the distance at which the camera or cameras are from the calibration plate, the robot used, etc. This class's information is not only used to populate the main form's labels, but also to save keep data which the calibration procedures require.

The *PositionInformation* class, which contains the work-space tools positioning information, holds the name and overall *tooldata* information of a specific tool, including the position where the robot should move to find this tool. The vision jobs required to find the tool are also saved in this class. More information on this class can be found in section 4.3.4.

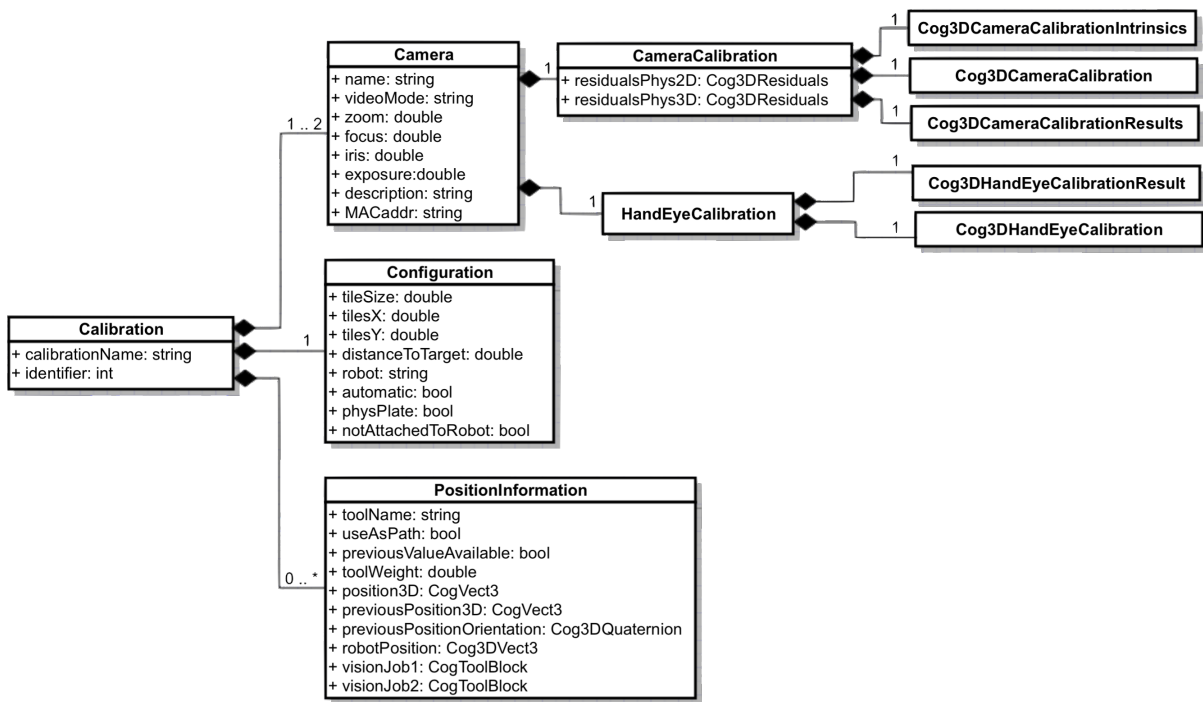


Figure 4.3: Calibration class's UML.

4.2 Application's Architecture

RoboCalib is the central link of a complex system of applications and interfaces, as well as the bridge between this system and the user. It is responsible to ensure a flawless and synced execution

of all applications and eventually handle its errors. The whole proposed system is actually comprised of three main applications: RoboCalib, the robot's application and RoboVision (the computer application responsible to run the robot's requested vision jobs, explained in section 3.3).

4.2.1 Application's Network Architecture

Figure 4.4 depicts the interfaces used to communicate between the three different applications. In this case, where RoboVision and RoboCalib are ran in the same computer as two different processes, the robot only requires to know one computer's IP address. The computer's IP address is used to establish a connection between the robot and the two server applications, RoboVision and RoboCalib. This connection is then used to send and receive TCP/IP messages. RoboCalib and RoboVision use

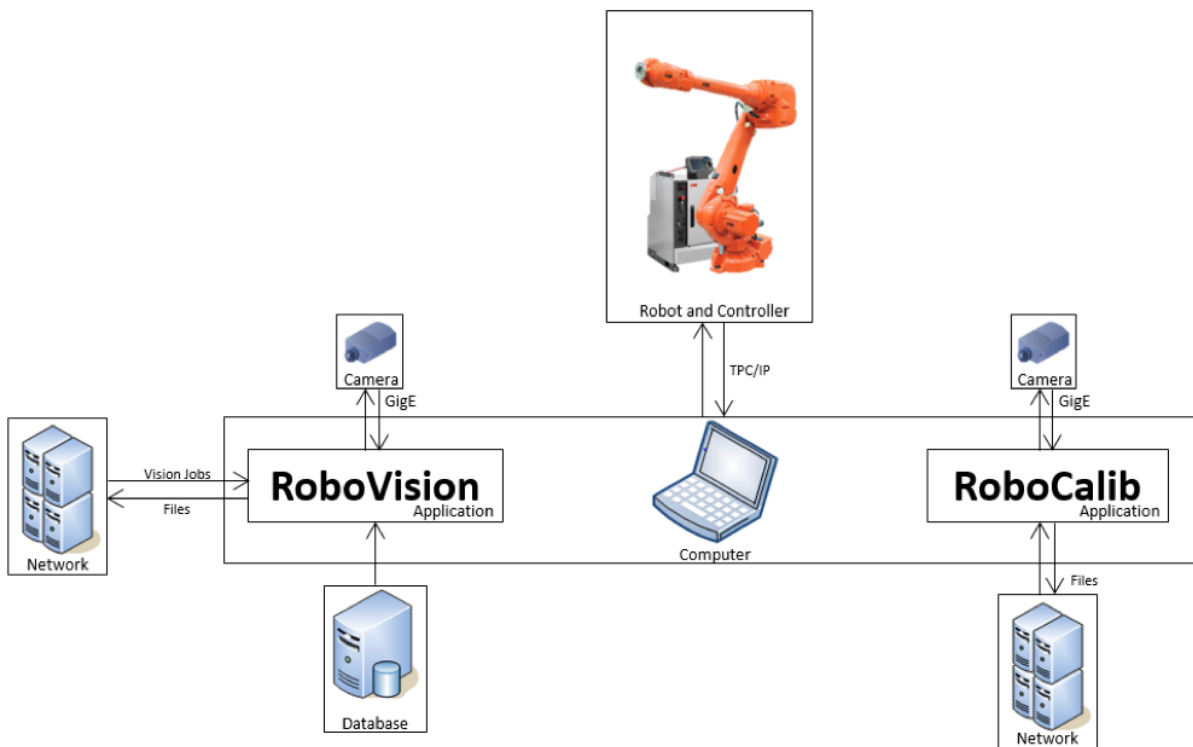


Figure 4.4: Top level network's architecture.

the company's network in order to store and retrieve files. These files are usually the vision jobs, but can be anything from temporary files, created during the execution of the vision jobs, to files where user configurations or calibration results are stored. As explained in section 4.1.2, the user configurations and calibration results are an object appended into a list which is then serialized into a file in the network.

A camera is used as a resource which cannot be shared between computers and even processes. Therefore, when a process connects to the camera, claiming it as a resource, no other process can use it until it is released. The release of the camera resource is only achievable when the process, which had previously claimed the camera, stops executing. Since both RoboVision and RoboCalib require the camera sequentially and throughout their execution, a deadlock issue arises. This means that if RoboCalib requests a camera resource, for example to set up camera settings, RoboVision will only get

access to the camera once RoboCalib's process has stopped executing.

In order to tackle this issue and to ensure that RoboCalib can run, use the cameras and hand the camera resources to RoboVision without the need of stopping its execution, RoboCalib was designed to be a set of different disposable processes. This means that when a RoboCalib's procedure requires to claim a camera, a new process is spawned. That new process starts by creating a pipe between itself and RoboCalib's main process, in order to exchange data. Once the cameras are not required anymore, the new process sends the data it collected to RoboCalib's main application and stops executing. When the user prompts the application to execute a procedure that requires the camera resources, RoboCalib checks if RoboVision is executing and asks the user to close it before continuing. At the same time, when the user prompts the application to execute a procedure that requires RoboVision to be executing, RoboCalib will launch it automatically.

RoboCalib requires to claim the camera resources for four different reasons, and therefore spawn four different processes with the intent of:

- Displaying a live feed of the selected cameras when the user is setting up a new camera configuration - *NewCalibrationConfiguration*.
- Displaying a live feed of the selected cameras during manual camera calibration - *ManualCameraCalibration*.
- Displaying a live feed of the selected cameras for the purpose of stereo triangulation - *ToolPointTriangulation*.
- Displaying a live feed of the selected cameras for the purpose of work-space calibration - *WorkSpaceCalibration*.

4.2.2 The Robot's application

The set of procedures the robot is running are here referenced as the robot's application. This is the application that awaits orders from RoboCalib and then executes the automatic calibration procedures. The robot's application is comprised of several modules, which include procedures and functions. A procedure is a set of instructions, used as a sub-program, while a function returns a value of a specific type and is usually used as an argument of an instruction.

The robot's application starts by establishing a connection with RoboCalib. This TCP/IP connection is established in order for the robot's application to receive RoboCalib's commands and send the procedures' results. The robot's calibration module goes through the sequential steps of algorithm 1. Basically, the robot's application awaits for a command, executes it, sends the results back and awaits for another RoboCalib's command. A detailed explanation of each command's procedure can be found in section 4.3.

```

1 establish connection with RoboCalib;
2 while connection is alive do
3   wait for command;
4   if command has been received then
5     establish connection with RoboVision;
6     execute respective procedure;
7     send results to RoboCalib;
8   end
9 end

```

Algorithm 1: Robot's application calibration module.

4.3 Implementation of the Calibration Tools

In this section a detailed description of the technical implementation of the calibration tools is given. These tools, camera calibration, hand-eye calibration, stereo triangulation and work-space calibration, make use of the three applications described in section 4.2. They are prompted by the user, using RoboCalib's UI, automated using the robot's application and executed by RoboVision.

All calibration procedures are automated, except for triangulation, but both camera and hand-eye calibration procedures can be executed manually if the user decides to do so. Manual calibration procedures can be necessary when some spatial constraints prevent the robot of moving freely or even to achieve better calibration results, with the drawback of a slower and more time consuming procedure.

4.3.1 Camera Calibration Implementation

Camera calibration is the foundation of all calibration procedures. It estimates the camera's intrinsic and extrinsic parameters. These are required to map 2D points into 3D rays, to use the correct camera center, to correct lens distortion and to have a transformation from the optical center of the camera to the world.

The camera calibration procedure returns a list of *Cog3DCameraCalibrationResult* objects, each corresponding to one camera, which is then used to populate each *Camera* object. RoboCalib uses these objects directly to display each camera's correct center as well as a to display a qualitative description of the procedure's performance (Figure 4.5). A new form is also made available to the user with all error statistics and results, for each viewset. Even further, once camera calibration has been executed, the user can export it as a vision job tool, which can be directly dragged to any vision job application. This tool acquires an image of that camera with the user specified parameters (zoom, focus, aperture and exposure) and corrects its radial distortion using the camera calibration object.

The camera calibration procedure can be executed in an automated fashion, using the robot, or manually. The automated mode uses the information the user made available regarding the cameras' position, with respect to the calibration plate, to simulate moving the calibration plate to the viewsets referred in section 3.1.3, without actually moving the calibration plate but the robot.

The manual mode uses a window with the cameras' live feed and an explanation of each different viewset to the user. Each time a user acquires an image of a certain viewset, the window explains the

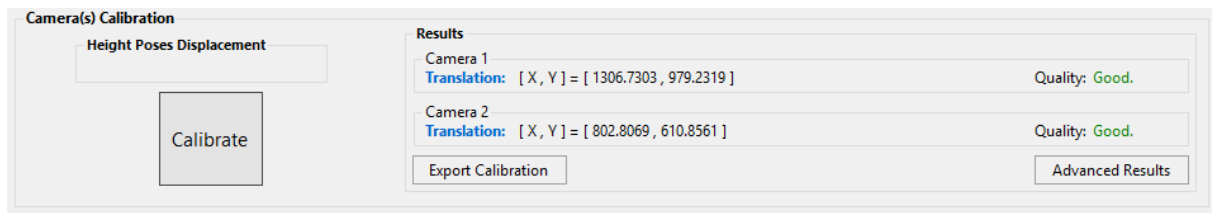


Figure 4.5: Camera calibration results for two cameras, displayed in RoboCalib's main form.

required steps to move the calibration plate to the next viewset. Once all viewsets have been acquired by the user, camera calibration is computed by this same process.

This section focuses on the automatic calibration procedure and its technical aspects. Figure 4.6 depicts all interactions between the three main applications, as well as the sequence of performed procedures.

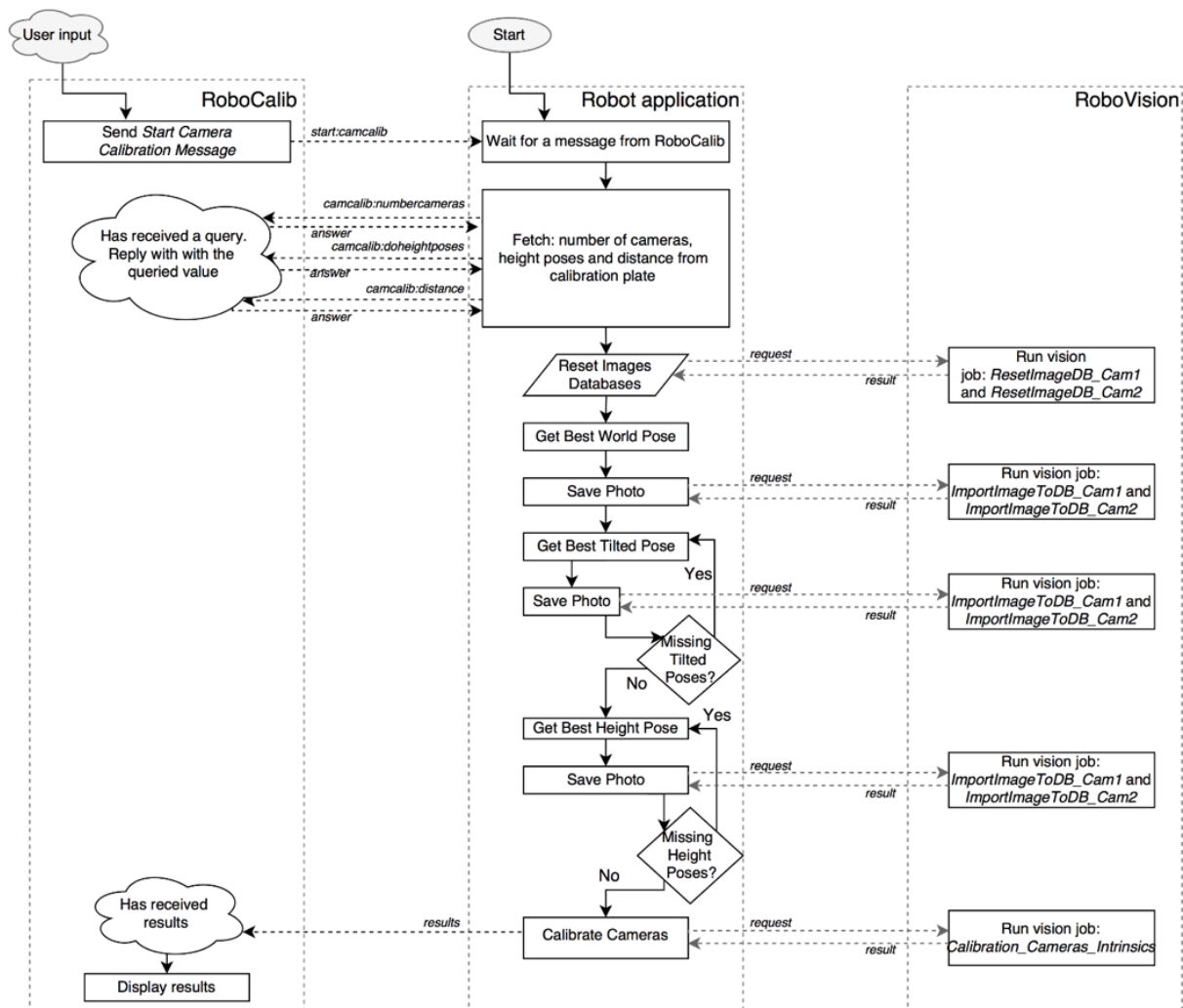


Figure 4.6: Automated camera calibration procedure.

RoboCalib is the procedure's initiator. When the user requests the camera calibration procedure to be executed, RoboCalib messages the robot's application in order to start the procedure. The robot's

application then requests essential data, such as the number of cameras to be calibrated, whether the user wants to perform height pose viewsets or not, and the vertical distance from the calibration plate.

The next step is to clear the databases (one for each camera) where the calibration images are stored. In order to do so, and considering that the robot has no write access outside its controller, the robot's application requests RoboVision to run the vision jobs responsible for clearing images in the databases. Following, the world viewset (section 3.1.3) is acquired. Since the calibration plate must be placed perpendicular to the cameras' optical axis, the robot does not change its position for this viewset, instead the user is requested to move the robot to a position where the calibration plate is visible to all cameras. Usually, there is no need to position the robot, since it was already correctly positioned when setting up the cameras.

Once the robot is ready to acquire a viewset, it requests RoboVision to execute the vision jobs responsible for acquiring and appending an image to each database.

```

1  establish a connection with RoboCalib;
2  if connection established then
3      query RoboCalib for the respective Camera object;
4      set-up video mode and format;
5      set-up an acquisition FIFO;
6      set pixel format to CogImagePixelFormatConstants.Grey8;
7      try
8          if user set zoom then
9              set user defined zoom;
10         end
11         if user set focus then
12             set user defined focus;
13         end
14         if user set aperture then
15             set user defined aperture;
16         end
17         if user set exposure then
18             set user defined exposure;
19         end
20     catch
21         cancel image acquisition;
22         inform RoboCalib and the robot's application of error;
23     end
24     try
25         get camera's pixel width and height;
26         set camera's ROI to width and height;
27     catch
28         cancel image acquisition;
29         inform RoboCalib and the robot's application of error;
30     end
31     acquire image from FIFO;
32     append image to the DB;
33 end

```

Algorithm 2: Vision job's algorithm to import images to DB.

The vision jobs which acquire and save the images, *ImportImageToDB.Cam1/2*, described as algorithm 2, start by establishing a connection and querying RoboCalib for the respective *Camera* object. Once the *Camera* object has been serialized and sent through their connection, its MAC address is

used to find that respective camera in the network. If the camera is found, then the saved video mode and format are applied and an acquisition FIFO is set-up. Since most VisionPro tools only work with 8-bit grayscale, the pixel format is always set to *CogImagePixelFormatConstants.Grey8*. As mentioned in section 3.4, it is possible to programmatically get and set *GigE* camera features, such as zoom, focus, aperture and exposure. These features might not be available for all cameras, and the camera will refuse the user's request if a specific feature is not compatible. For this reason, setting features is always done with try and catch blocks. The last feature to be set is the ROI, Region of Interest, which is always set to the whole image. Once all features have been correctly set, an image is acquired from the FIFO and saved into the respective images' DB.

The next set of viewsets are the tilted poses (section 3.1.3). VisionPro advises for the cameras to remain still and to tilt and rotate the calibration plate. Being automation one of the main objectives of this project, an algorithm was devised to simulate the calibration plate's motion using the robot's motion instead.

Figure 4.7 illustrates the original position and orientation of the camera C with respect to the calibration plate.

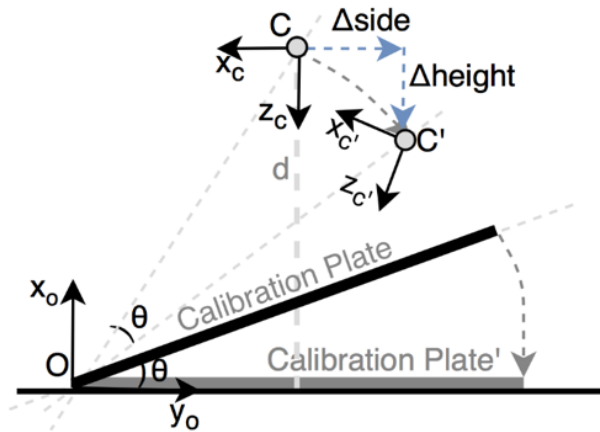


Figure 4.7: Required camera transformation.

Let O be the coordinate system in one of the sides of the calibration plate, where its Z axis runs along its edge. Then, C' is the camera after being rotated θ degrees around O_Z , which has the same position and orientation, with respect to the new also rotated calibration plate than C with the original calibration plate orientation. The required camera's displacement will then be Δ_{height} , Δ_{side} and a rotation of θ degrees around its Y axis. Now, let O' be the coordinate frame O after a θ rotation around its Z axis. Its homogeneous transformation is described by equation 4.1.

$$O' = Rot(Z, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

The position of C' center, c' , with respect to O is then calculated using equation 2.3 from section 2.1.

$$c' = O' \cdot c = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l \cdot \frac{\cos(\theta)}{2} \\ d \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} l \cdot \frac{\cos(\theta)}{2} - d \cdot \sin(\theta) \\ l \cdot \frac{\cos(\theta)\sin(\theta)}{2} - d \cdot \cos(\theta) \\ 0 \\ 1 \end{bmatrix}, \quad (4.2)$$

where l represents the width of the calibration plate. The Δ_{height} and Δ_{side} are then calculated:

$$\Delta_{height} = c'_x - c_x = l \cdot \frac{\cos(\theta)}{2} \cdot (1 - \cos(\theta)) + d \cdot \sin(\theta) \quad (4.3)$$

$$\Delta_{side} = c'_y - c_y = d \cdot (1 - \cos(\theta)) + l \cdot \frac{\cos(\theta)\sin(\theta)}{2} \quad (4.4)$$

The robot then moves the camera to the four sides of the calibration plate and acquires those four viewsets, simulating the calibration plate's motion using the equations' 4.3 and 4.4 displacements.

The next step is to acquire the height pose viewsets. For these viewsets the robot moves vertically to the positions $-2d, -d, d$ and $2d$, where d is the user defined height pose displacement. Once again, in every pose the robot acquires a viewset and saves it in the images' databases.

```

1 establish connection with RoboCalib;
2 if connection has been established then
3   query the number of cameras, tile size and height poses displacement;
4   distortion model  $\leftarrow e3ParamRadial$ ;
5   set-up the viewset types and height poses lists;
6   foreach camera do
7     create a new list of images;
8     foreach calibration image in the database do
9       add to the list of images;
10    end
11  end
12  create a new list with the lists of images;
13  create a new list of lists of features;
14  foreach calibration image do
15    extract checkerboard features(tile size);
16    add features' list to the list of features;
17  end
18  calibrate cameras(lists of features, list of viewset types, list of heights);
19  if calibration is successful then
20    save results in network;
21    inform RoboCalib that calibration is complete;
22  else
23    inform RoboCalib that the calibration was unsuccessful;
24  end
25 end

```

Algorithm 3: Camera Calibration Algorithm.

The last step of the camera calibration procedure is the actual calibration. It is executed by the vision job *Calibration_Cameras.Intrinsics* using the images in the databases. Algorithm 3 describes the sequence of instructions executed.

Establishing a connection with RoboCalib is done by fetching its IP from the network, which is saved in a file, updated by RoboCalib when it was loaded. If the TCP/IP connection is then successfully established, the first step is to query RoboCalib for the following values: number of cameras, tile size and height poses displacement.

The next step is to define the distortion model. The user has no access to modify the model since the VisionPro's chosen model, *e3paramRadial*, is the appropriate for industrial lenses, where distortion is almost not apparent to the human eye and where distortion correction can improve accuracy performance nonetheless.

The method which executes the calibration procedure requires three lists: a vector of correspondences features between 3D calibration plate locations and 2D image locations, the different heights of the height pose viewsets (if used) and a list with the order of which the viewsets were acquired. The list with the different heights of the height pose viewsets is a list of *doubles*, calculated using one of the queried values, and the list of ordered viewset types is hardcoded, since the order at which the viewsets are acquired is always the same.

The two-dimensional list of feature correspondences is returned by the method *Execute(List<List<ICogImages>>)* of the class *Cog3DCalibrationFeatureExtractorBase*, where the *List<List<ICogImages>>* are the lists with the calibration images of each camera. Once the three lists have been set-up correctly, camera calibration is executed using the class *Cog3DCameraCalibrator*.

The result of camera calibration, a *Cog3DCameraCalibrationResult* object, is then serialized and saved in the network. This object is serialized and saved in the network and not sent directly to RoboCalib so that other processes, which require a specific camera calibration result, can use it later. It is saved under the name of its respective *Calibration* object ID, a unique number assigned to all calibration configurations, so that it can be distinguished by others.

Once camera calibration has been executed it is possible to export the calibration, from RoboCalib, in a VisionPro's valid format (tool block). As explained in section 3.1, a vision job is usually developed using VisionPro's UI, by interfacing vision tool blocks with each other. The image source is usually chosen when configuring the vision job or set up by the user by using an acquisition FIFO tool. The exported calibration is a VisionPro's tool block as well, which can be dragged to the UI and interfaced with other vision tools. The exported calibration acquires an image from the camera, with the corresponding camera features defined in RoboCalib prior to calibration, and corrects the image's distortion. At the same time, this tool outputs the camera calibration, so that other custom vision tools can use it. Algorithm 4 details the implementation of this tool.

The respective camera calibration, loaded by algorithm 4, is stored in the network under its *Calibration* ID, as mentioned before. When the calibration tool is exported, RoboCalib updates its script with the respective *Calibration* ID, so that it knows what file, with the camera calibration, to search for. The class *Cog3DLensDistortionCorrector* is used to correct lens distortion when given a camera calibration object. This class also creates a new camera calibration object, since the camera intrinsic and extrinsic

```

1 load respective camera calibration from network;
2 create new Cog3DLensDistortionCorrector object;
3 train the corrector object with the camera calibration;
4 get new camera calibration from corrector;
5 acquire an image using algorithm 3;
6 if image acquisition is successful then
7     execute distortion corrector with acquired image;
8     output corrected image;
9     output new camera calibration;
10    output camera center;
11 end

```

Algorithm 4: Exported camera calibration tool.

parameters change when distortion is corrected. After acquiring a new and distorted image, from a similar procedure to algorithm 3, that image is fed into the *Cog3DLensDistortionCorrector* object, to produce a new undistorted image. In the end, if no error occurred, the tool outputs the undistorted image, the new camera calibration and the right camera center.

4.3.2 Hand-Eye Calibration Implementation

Hand-eye calibration is the second most important calibration procedure in a vision guided robot. It estimates the cameras' position and orientation (Camera3D space) with respect to the robot's hand (Hand3D space). It is then used to know exactly the distance between the camera and the robot's manipulator, and therefore knowing how much the manipulator needs to move in order to be centered with the camera's target. Hand-Eye calibration is also essential for positioning the camera's optical axis normal to the working plane.

Once the hand-eye calibration procedure has been executed, a *Cog3DHandEyeCalibrationResult* object is returned. This object contains the transformations of the cameras (Camera3D) to the robot's hand (Hand3D), as well as the error statistics of each viewset. RoboCalib makes these available in its UI, being the transformations already displayed as *tooldata*, the correct code format for the robot's application (Figure 4.8).

Figure 4.8: Hand-eye calibration results for two cameras, displayed in RoboCalib's main form.

Figure 4.9 is a representation of the automatic hand-eye calibration procedure. Just as camera calibration, the procedure is initiated by the user and can be automated or manual. Assuming that the

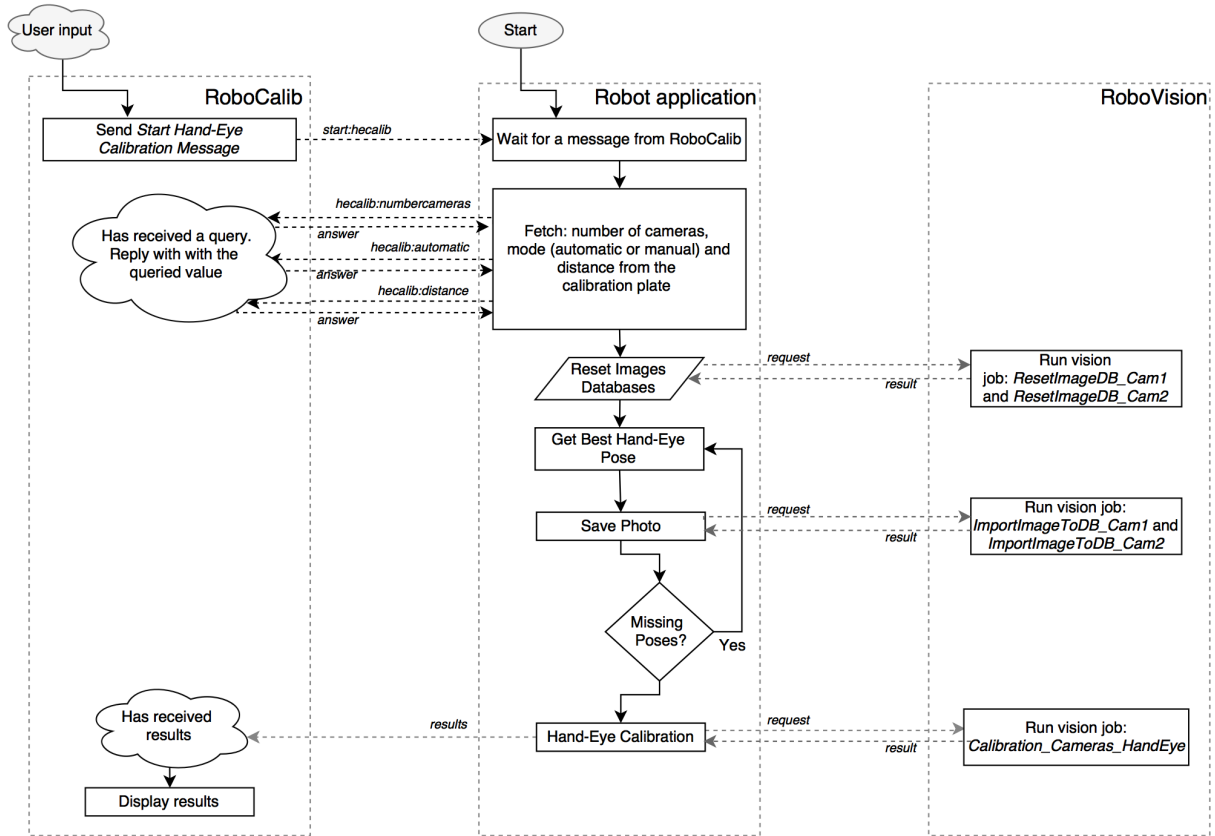


Figure 4.9: Automated hand-eye calibration procedure.

connection has already been established, RoboCalib sends an initiating message which prompts the robot's application to query the number of camera, running mode (automatic or manual) and the approximate distance to the calibration plate. Once these values have been queried, the robot's application requests RoboVision to run the vision jobs *ResetImageDB_Cam1/2*, the same vision jobs used in the camera calibration procedure, which reset the images in the databases used to store the calibration images.

After resetting the image databases, the robot is ready to start acquiring the calibration images. These calibration's requirements, presented in section 3.1.4, requires the robot to move to nine different stations, at least. Between these stations the robot's motion must rotate the camera around all three axes, and consequently change its position so that the calibration plate is visible again. This process, executed by the robot's *GetBestHand-EyePose* procedure, follows the sequential steps of algorithm 5.

- 1 define Z , Y and X nine rotations;
- 2 calculate required displacement to center the camera on the calibration plate after rotation;
- 3 rotate camera;
- 4 move camera;
- 5 estimate the robot's hand pose;

Algorithm 5: *GetBestHand-EyePose* procedure.

The Z , Y and X nine rotations are pre-defined because the robot cannot generate random numbers. Therefore, these numbers were previously randomly generated and then tuned to cover as much

different rotations as possible. The next step, the calculation of the required displacement to center the camera on the calibration plate after rotation, is computed by solving equation 2.3, where transformation b_aT is the result of the camera's rotation around the axes Z , Y and X , being $\{b\}$ the rotated frame with respect to frame $\{a\}$. A homogeneous matrix transformation resultant of a rotation around the axes Z , Y and X , respectively, is given by matrix 4.5.

$${}^b_aT = \begin{bmatrix} c_Y - c_Z & c_Z s_X s_Y - c_X s_Z & c_X c_Z s_Y + s_X s_Z & 0 \\ c_Y s_Z & c_X c_Z + s_X s_Y s_Z & -c_Z s_X + c_X s_Y s_Z & 0 \\ -s_Y & c_Y s_X & c_X c_Y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4.5)$$

where c_i and s_i represent $\cos(\theta_i)$ and $\sin(\theta_i)$, respectively, being i the axis of rotation and θ the respective angle of rotation.

Transformation 4.5 is then used to calculate the position of the point resultant of the intersection between frame's $\{b\}$ Z axis and the plane where the calibration plate is laying. The distance between this point and the point where frame's $\{a\}$ Z axis and the plane where the calibration plate lays intersect, is the distance the camera needs to move, after rotation, to center the calibration plate again. This distance is represented by v in Figure 4.10.

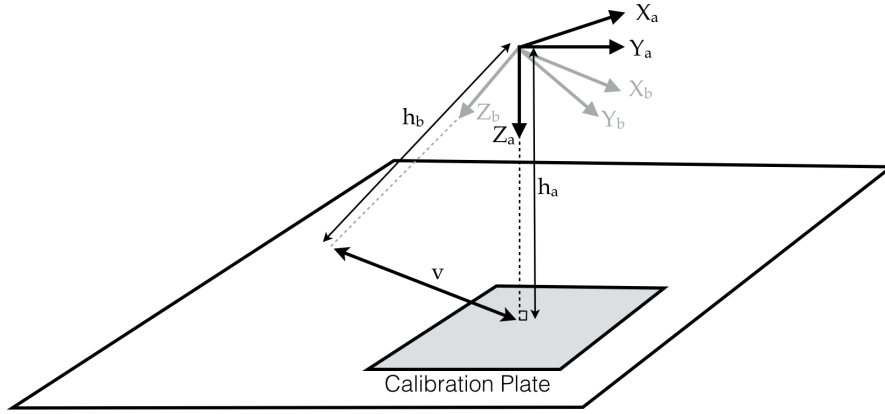


Figure 4.10: Frames $\{a\}$ and $\{b\}$, where frame $\{b\}$ is the result of frame's $\{a\}$ rotation around all axes. v represents the distance between both frame's Z axes intersection with the calibration plate plane.

Being the calibration plate normal to frame's $\{a\}$ Z axis, v can be expressed in X_a and Y_a coordinates (v_x, v_y) . Moving the opposite of this distance, before rotating the camera, will ensure having the calibration plate once again centered on the camera after rotation.

After moving and rotating the camera another problem arises. The distance between the camera and its Z axis intersection with the calibration plate, ΔZ_b , will be different than the previous distance. Thus, this distance must also be calculated and compared with the previous distance, so that it is corrected.

Equation 4.6, derived from equation 2.3, solves these three variable $(v_x, v_y$ and $\Delta Z_b)$, by intersecting Z_b axis with the plane of the calibration plate, and by representing this intersection's point in frame's $\{a\}$

coordinates.

$$\begin{bmatrix} v_x \\ v_y \\ height \\ 1 \end{bmatrix} = \begin{bmatrix} c_Y - c_Z & c_Z s_X s_Y - c_X s_Z & c_X c_Z s_Y + s_X s_Z & 0 \\ c_Y s_Z & c_X c_Z + s_X s_Y s_Z & -c_Z s_X + c_X s_Y s_Z & 0 \\ -s_Y & c_Y s_X & c_X c_Y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \Delta Z_b \\ 1 \end{bmatrix}, \quad (4.6)$$

where *height* is the known distance between the camera (before rotating) and the calibration plate. The solution can then be expressed as:

$$\begin{cases} v_x = \frac{height}{c_X c_Y} \cdot (c_X c_Z s_Y + s_X s_Z) \\ v_y = \frac{height}{c_X c_Y} \cdot (-c_Z s_X + c_X s_Y s_Z) \\ \Delta Z_b = \frac{height}{c_X c_Y} \end{cases} \quad (4.7)$$

The robot then calculates these parameters, moves $-v_x$ and $-v_y$, rotates θ_Z , θ_Y and θ_X , and moves $\Delta Z_b - height$ in the Z_b direction. Once the robot has stopped moving, each camera acquires an image, using the same method for acquisition as in algorithm 2. Before moving to the next station the robot estimates its *tool0* (Hand3D) position and orientation, with respect to the robot's base. This pose is then sent to RoboCalib, which is saving them in a list, to be later used by the hand-eye calibration.

While this algorithm only considers the primary camera, the secondary camera will also be pointed at the calibration plate, after the rotation, if both cameras focal point is the same. This is the reason why users, when setting up the cameras at a certain *height*, are informed that both cameras' center should focus the same point.

If the user decides to move the camera manually, the robot will stop its execution and let the user jog the robot to the desired station. From there the user can restart the robot's application execution to acquire the images and to stop once more for the user to jog the robot to the next station, until the nine stations are complete. Once again, each time the robot restarts its execution, *tool0*'s position and orientation are estimated and sent to RoboCalib.

Manually or automatically the last step is the hand-eye calibration carried out by the vision job *Calibration_Cameras_HandEye*. This is a similar method to the camera's calibration method, but instead of using different viewsets, the hand-eye calibration method uses the robot poses together with the images' feature correspondences.

Once the hand-eye calibration procedure has been executed successfully, RoboCalib fetches the *Cog3DHandEyeCalibrationResult* object, which had been stored in the network by the vision job *Calibration_Cameras_HandEye*, and adds it to its *Camera* objects. The Camera3D to Hand3D transformation is then retrieved from each *Camera* object and displayed on RoboCalib's UI in the format of *tooldata*.

The transformation is displayed using RAPID's notation, which is comprised of a position and an

orientation (with quaternions) between squared brackets, and only needs to be copied to the robot's application to be implemented in any project.

4.3.3 Stereo Triangulation Implementation

Stereo triangulation, as introduced in section 2.6, is the estimation of the position of a 3D point using two cameras, with a shared FOV. The point's position estimation is done by intersecting two 3D Rays, from these two cameras.

Triangulation has many practical applications in the industry, from offering the perception of depth, in some extent, to a robot, to the possibility of performing accurate 3D measurements using two triangulated points. In this project, the stereo triangulation is proposed to be used as a positioning estimator as well as for precise 3D measurements. It is offered in two different ways, as a VisionPro's UI tool block and through an UI, integrated in RoboCalib. While the first one can only triangulate points in the Phys3D and/or Hand3D space, the second one triangulates a point in Phys3D, Hand3D and RobotBase3D.

The principle behind both applications is the same, their only difference is that the second application is able to request the robot's current pose, thus being able to calculate the triangulated point in RobotBase3D. Therefore, for the first case camera and hand-eye calibrations are required, but for the second one, an established connection to the robot is required as well, since the following transformations are necessary:

$$\begin{aligned} &Phys3D \rightarrow Camera3D \text{ (Camera calibration),} \\ &Camera3D \rightarrow Hand3D \text{ (Hand-eye calibration)} \\ \text{and } &Hand3D \rightarrow RobotBase3D \text{ (Robot's hand pose)} \end{aligned}$$

As mentioned in section 4.2, RoboCalib's integrated stereo triangulation tool is executed as a different process, since it requires the cameras' resources. This means that RoboVision cannot be running at the same time. Usually, since RoboVision is used for the hand-eye calibration, it would be open when the user executes the triangulation tool. Therefore, RoboCalib always checks if RoboVision is running before launching the new process with the stereo triangulation tool. If it is running, then RoboCalib will ask the user if he wants it to kill the process.

Once RoboVision has ceased its execution, RoboCalib creates a pipe server, launches the tool and waits for it to connect. This connection is then used to exchange and request data, including the cameras' calibration objects and the robot's poses.

Triangulation is achieved using the *Cog3DTriangulator* class and by executing its *Execute(List<Cog3D CameraCalibration>, List<Double>)* method. The points used as parameters (*List<Double>*), in this method, are either the result of a vision job or the pixel coordinates selected manually by the user. The

second option can only be used in RoboCalib, not in the VisionPro's UI tool block, since it requires the user to pin-point that position on the cameras' displays. The cameras' displays are not directly connected to the cameras' acquisition FIFOs, instead its image distortion is first corrected. To do so, and to have an undistorted live feed of the cameras, a timer is set. When the timer's interval is over, an event is raised. In this event RoboCalib de-activates the timer and uses the *Cog3DLensDistortionCorrector* class, together with the cameras' calibration objects, to correct the images' distortion. Once the correction is executed, the timer is set again.

If the user decides to input the points to be triangulated manually, then the points will be fixed independently of the image, but if the user loads or creates a new vision job to detect a point then another timer is set. When the timer's interval is over, the raised event runs the vision job associated with each point and camera.

To facilitate the point's management, a class with the points' information was created. This class holds the points' coordinates, respective vision job (if existent) and a flag to discern the point's mode (manual or automatic). Four objects of this class are created as soon as the triangulation tool is loaded, and are modified by the user's input.

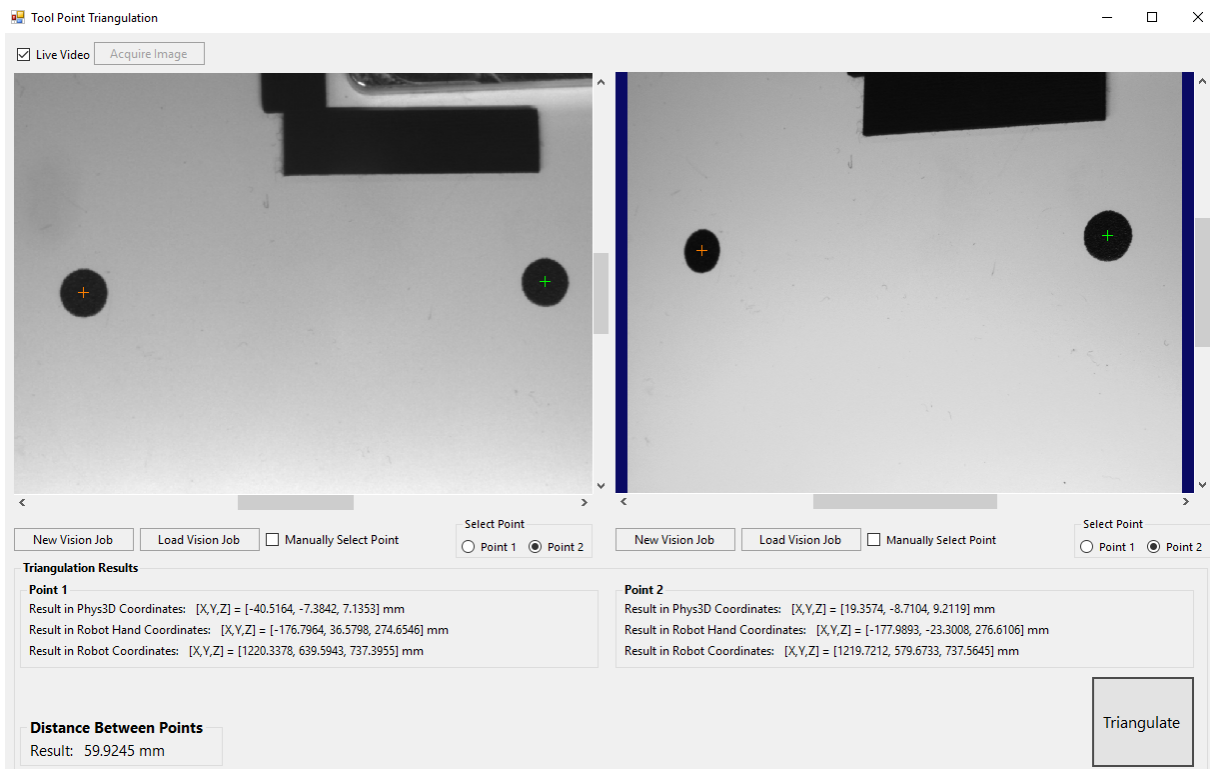


Figure 4.11: Stereo triangulation UI.

Since the user might decide to load or create a new vision job, while that respective vision job is being executed, or vice versa, a mutex was implemented. Therefore, before using a vision job object, the mutex is always checked. The method *TryEnter(Object)* of the static class *Monitor* attempts to acquire an exclusive lock on the specified vision job. If the vision job is locked then the program will not wait but

move on to the next instruction. If it manages to acquire the vision job object, then the program will lock it and modify it. In the latter case a *try/catch/finally* block is always used to ensure that, independently of any complication, the vision job lock will always be freed using the *finally* block.

The triangulation method returns a list of *Cog3DVect3* (3D points), depending on the number of input points, in the Phys3D space (the common cameras' space defined after camera calibration). The coordinates of a point in this space are not very useful from a robotic's application point of view, unless when used for estimating a distance between two 3D points.

Nonetheless, the results are always displayed in the Phys3D space, but if any other calibration procedures, besides camera calibration, have been executed, then the results will be displayed in other spaces. If the hand-eye calibration is available, then the *Camera3D* \rightarrow *Hand3D* transformation can be used to estimate the triangulated point in the Hand3D space. A 3D point in the Hand3D space has no use in any application, but it is required in order to estimate the point's position in the Robot3D space, which is useful to estimate tools' or parts' positions with respect to the robot. Therefore, the triangulation tool requests RoboCalib, holder of the robot's connection, to request the robot's application an estimation of its hand pose. The returned pose, position and orientation, is then used to create a new *Cog3DTransformLinear* object, which can map points from Hand3D to RobotBase3D. In order to detail this process even further, algorithm 6 is presented.

Another utterly important functionality of the triangulation tool, is the ability of quickly defining a new *workObject* or calibrating an older one. A *workObject* is defined by three points, its frame center, a point on its X-axis and a point on its XY-plane. Usually, the process of calibrating or creating a new *workObject* is very time consuming and requires expensive and very precise tools. It makes use of two very precise needle points and a sliding cylinder. While one needle point needs to be firmly attached to the *workObject*'s plane, the second needle point is attached to the flange of the robot. Firstly, the robot's needle point needs to be calibrated so that its *tooldata* is known precisely, then this needle is moved until it touches the tip of the other needle. The next step is to orient the robot's needle so that both needles' orientation match. This can be inferred by sliding the cylinder from one needle to the other. If the needle points are touching and perfectly oriented, the cylinder will slide effortlessly from one needle to the other. Then, the robot's needle position and orientation is saved.

This process needs to be repeated three times, so that three points are acquired and used to define the *workObject*. Since these needles have a distinct height from the *workObject*, the *workObject*'s height will be different from reality, therefore requiring the user to change the *workObject*'s height manually.

Using this triangulation tool, a user can achieve about the same results in a much faster and efficient way. More than that, the proposed solution offers a more flexible approach, since it just requires the user to triangulate three points in the surface of the *workObject* and define it using them. Conclusions about the accuracy of this procedure, when compared to the previous one, can be found in section 5.4.2.

```

1 estimate the new camera calibration for both cameras after distortion correction;
2 create a new list with the two camera calibrations;
3 if point 1 has coordinates assigned in both displays then
4     execute triangulation (list of camera calibrations, list of the point 1 coordinates in each display);
5     display the triangulation results in Phys3D;
6     if hand-eye calibration has been executed then
7         get one of the transformations  $Phys3D \rightarrow Camera3D$  from camera calibrations;
8         map the triangulated point in Camera3D space using its transformation;
9         get the corresponding transformation  $Hand3D \rightarrow Camera3D$  from hand-eye calibrations;
10        map the Camera3D point in the Hand3D space using the inverse transformation;
11        display the triangulation point result in Hand3D;
12        if the robot is connected then
13            send a pipe message to RoboCalib requesting the robot's pose;
14            wait for reply and set a timeout;
15            if reply was received before timeout then
16                create a new transformation object using the robot's pose;
17                map the Hand3D point in the RobotBase3D space using the transformation;
18                display the triangulation point result in RobotBase3D;
19            end
20        end
21    end
22    if point 2 has coordinates assigned in both displays then
23        repeat point 1 procedure;
24    end
25    if both points were triangulated successfully then
26        calculate the distance between them using their Phys3D coordinates;
27        display the distance;
28    end
29 end

```

Algorithm 6: Triangulation algorithm.

4.3.4 Work-space Calibration Implementation

Work-space Calibration is the last calibration procedure, which requires all the other camera calibrations in order to be executed. Its main goal is to close the automated calibration gap and to ensure a fully automated calibration set-up. Being all camera calibration procedures automated, there is a demand for an automated tool calibration procedure. This calibration procedure should assure the robot that the tools, which it needs to use, are exactly where their defined *tooldata* is.

This procedure has to be executed as a different process as well, *WorkSpaceCalibration*, since it requires the cameras' resources. Therefore, RoboCalib creates a pipe between it and the new process, the same way as for the triangulation tool, in order to exchange calibration data and requests.

Usually, in order to ensure the right position of the tools, the robot is moved to a pre-defined position where the tool is discernible and noticeable to the camera. Once there, the robot continuously runs a vision job that detects the 2D position of a tool and calculates its distance from a previously defined calibrated tool position. If the distance is bigger than a defined threshold, the vision job will run again and a message will be displayed to the user warning him that the tool is still not in its calibrated position. Once the tool's position is adjusted to be within a certain threshold, the application can proceed.

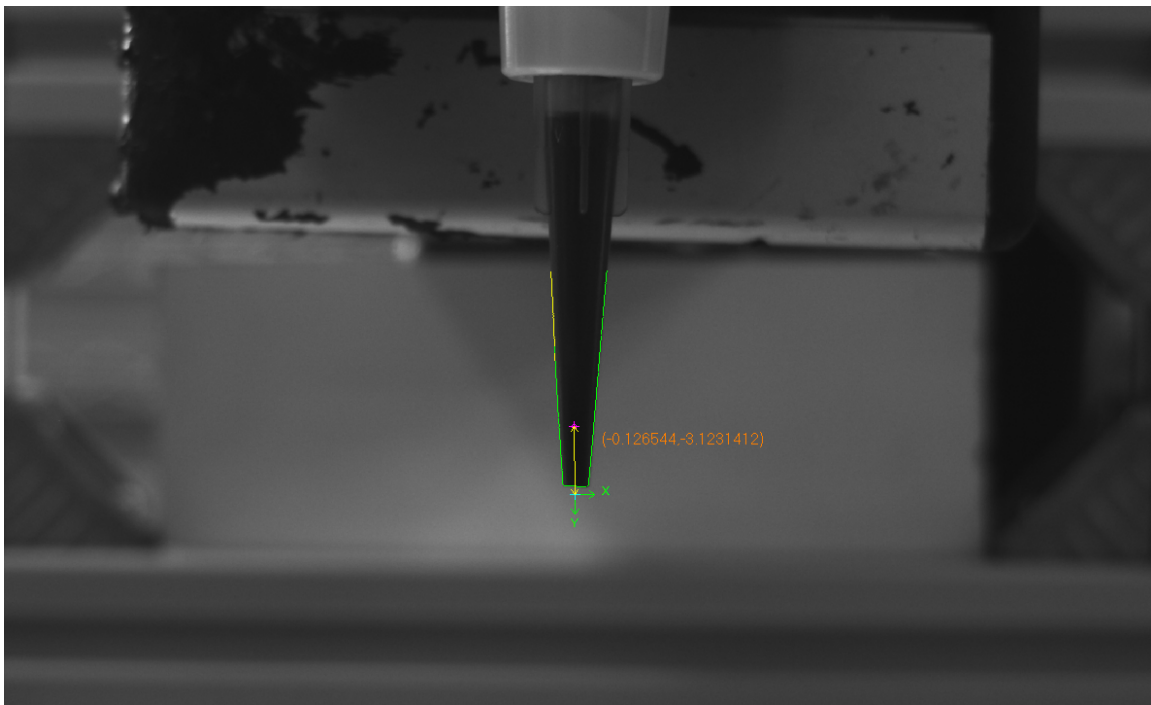


Figure 4.12: A needle position being calibrated. A vision job detects the needle position and the distance at which the needle is from the calibrated position.

Work-space calibration is the proposed solution to optimize this procedure. It uses the triangulation method introduced in the previous section to triangulate the tools' position, thus requiring a stereo system as well. Its purpose is to automatically update the tools' *tooldata* in the robot's application. In order to do so, it needs to move to poses where the tools are visible and discernible in both cameras. Since

the robot does not know priorly where to find the tools, this procedure has two phases: the teaching phase and the execution phase.

The teaching phase requires the user to define a robot path by jogging the cameras to the respective positions of the tools. RoboCalib presents an UI with both cameras' undistorted images, corrected the same way as the images used in the triangulation tool. Opposed to the triangulation tool, the work-space calibration cannot be used manually. This means that the points to be triangulated cannot be inserted manually, but have to be a result of a vision job. Vision jobs work the same ways as in the triangulation tool, and therefore, the user can load or save vision jobs. With respect to their implementation, vision jobs are implemented in the same way as in the triangulation tool, including the use of a timer and mutex to handle them.

Since the path from one tool to another might be constrained, the user has the option to set-up auxiliary poses. In these poses no vision job is ran and the robot will not be stopped. The position is just an intermediary step before moving to the next tool, to avoid any path constraint between two consecutive tools.

When the user intends to save a pose, auxiliary or a tool's pose, RoboCalib queries the robot's application for the robot's hand current pose. Once again, since the connection is established between RoboCalib and the robot, and not between the *WorkSpaceCalibration* process and the robot, the *WorkSpaceCalibration* process has to request the robot's pose to RoboCalib, which then requests the robot's pose directly. Once the pose has been sent to RoboCalib, and received by this process, it is saved in a *PositionInformation* object, introduced in section 4.1.2. This object contains a tool's information, such as name and weight, the pose where the robot moved in order to focus the tool, the vision jobs used by each camera to identify the position of the tool, and the results of the triangulation of the position of the tool.

This object is then saved in a list of *PositionInformation* objects, being each object a pose for the robot, which can either be a tool pose or a path pose, depending on the value of the flag *useAsPath*.

When the user finishes the path with all the tools he wants to calibrate and their corresponding path poses, the list of *PositionInformation* is sent to RoboCalib through their pipe connection.

The execution phase executes the previously taught path, as described in algorithm 7. The robot will move from pose to pose and run its vision jobs in the tools' poses. The vision jobs are executed until they find its respective tool and triangulation is successful. The tool's 3D position is then saved as its *position3D* in the corresponding *PositionInformation*.

In the end, after successfully finding all tools, RoboCalib will display the results. These results are displayed in a table where the user has write-access to each cell, and where each row represents a different tool. The parameters for each tool are the same parameters used to define a *tooldata* variable:

- Position of the tool (3D position triangulation result);

- Orientation of the tool (default - no rotation);
- Weight of the tool (0 kg).

Other parameters are required to parametrize a *tooldata* variable, such as center of gravity or moments of inertia, but these are usually set to a constant value. Therefore, these parameters can only be changed in the robot's programming environment.

```

1 foreach PositionInformation in List<PositionInformation> do
2   request RoboCalib to move the robot to next pose;
3   wait for RoboCalib's messages informing that the robot has reached its position;
4   if !PositionInformation.useAsPath then
5     while both vision jobs haven't been successful do
6       execute vision jobs;
7     end
8     triangulate the tool's position;
9   end
10 end
11 display all tools' estimated positions;
12 if desired update the corresponding tooldata in the robot's application;
```

Algorithm 7: Work-space calibration execution phase.

As previously mentioned, the user has then the option of modifying any of these parameters, including a tool's position result. There is also an option to select a set of tools and then update their values in the robot's code. Once again, since the work-space tool has no established connection with the robot, the tools' objects are sent to RoboCalib through their shared pipe. RoboCalib then informs the robot's application that will send a list of tools. Since the robot's connection can only handle strings with a maximum of 80 characters, the objects have to be sent one by one, as well as its variables. Thus, the robot starts by receiving the tool's name, and parses the next received items to a tool with that name. Once all items have been received the robot will either wait for another tool's name or receive an *END* command, informing it that all tools' information has been sent.

Tooldata variables cannot be created during the robot's application execution, and therefore it is not possible to directly assign the received *tooldata* parameters to a new *tooldata*. A *tooldata* does not contain any parameter which could be used to differentiate it from other *tooldata* variables, but its declared name. This means that the only way of assigning the new *tooldata* variables is to create a new data structure, and store them there.

This new structure *ToolData*, a RAPID *RECORD*, has exactly the same parameters as a *tooldata* variable and a name identification. The name is then used to differentiate the different tools. In the beginning of the execution of the robot's application, an array of *ToolData* variables is initialized. This array is of type *PERS*, which means that its values are persistent and will not be erased or modified when the execution has ended.

Since *ToolData* is not a variable of type *tooldata* it cannot be used directly in the robot's application. To tackle this issue, a function that gets the name of the required tool and returns a *tooldata* variable with the parameters of the required tool, was created. This function is essentially a search function, since it

looks in the *ToolData* array for the *ToolData* with the input tool's name and copies its parameters to a *tooldata*, which is then returned.

Chapter 5

Application's Validation

This chapter introduces, describes and discusses different strategy evaluation metrics and methods for the calibration tools proposed in chapter 4. It dedicates one section to each tool with the respective tests and conclusions, regarding its precision, accuracy and performance. Before heading into the tests of each tool, this chapter presents a characterization of the robot and vision setup used to carry out these same tests.

5.1 Robot Cell Work-space Characterization

In this section, the robot's cell workspace and vision elements, used to validate the results, are characterized. Therefore, a complete explanation of the setups is provided.

The robot cell used to carry out these tests is an ABB IRB 4600, capable of handling objects up to 45 kg and with a maximum reach of 2.05 m. It is used for assembling tasks, such as gluing different parts of a product.

Its manipulator, comprised of suction cups, is used to grasp and manipulate the parts to be assembled. These parts are usually settled in a set of stacked bins that can be easily manipulated by the robot. After picking up the parts from the bins, the robot uses a carrier to perform highly precise assemblies. This carrier usually contains a precise pattern, such as dots, which allows the robot to perform measurements with a better precision, using the dots as references.

Apart from the bins and the carrier, the robot is still sided by a set of pertinent tools, such as glue dispensers. Figure 5.1 is a representation of the complete robot's work-space.

The main camera, responsible for acquiring the images for the vision jobs, is a DFK Z12GP031 GigE camera capable of 12xZoom and 5 MP. This camera is firmly attached to the robot's arm and oriented the same way as the manipulator.

Another camera, a DMK Z12GP031 GigE camera, is also used as an external fixed camera. The aim of this camera is to retrieve feedback from procedures where the part to be examined is not visible to the

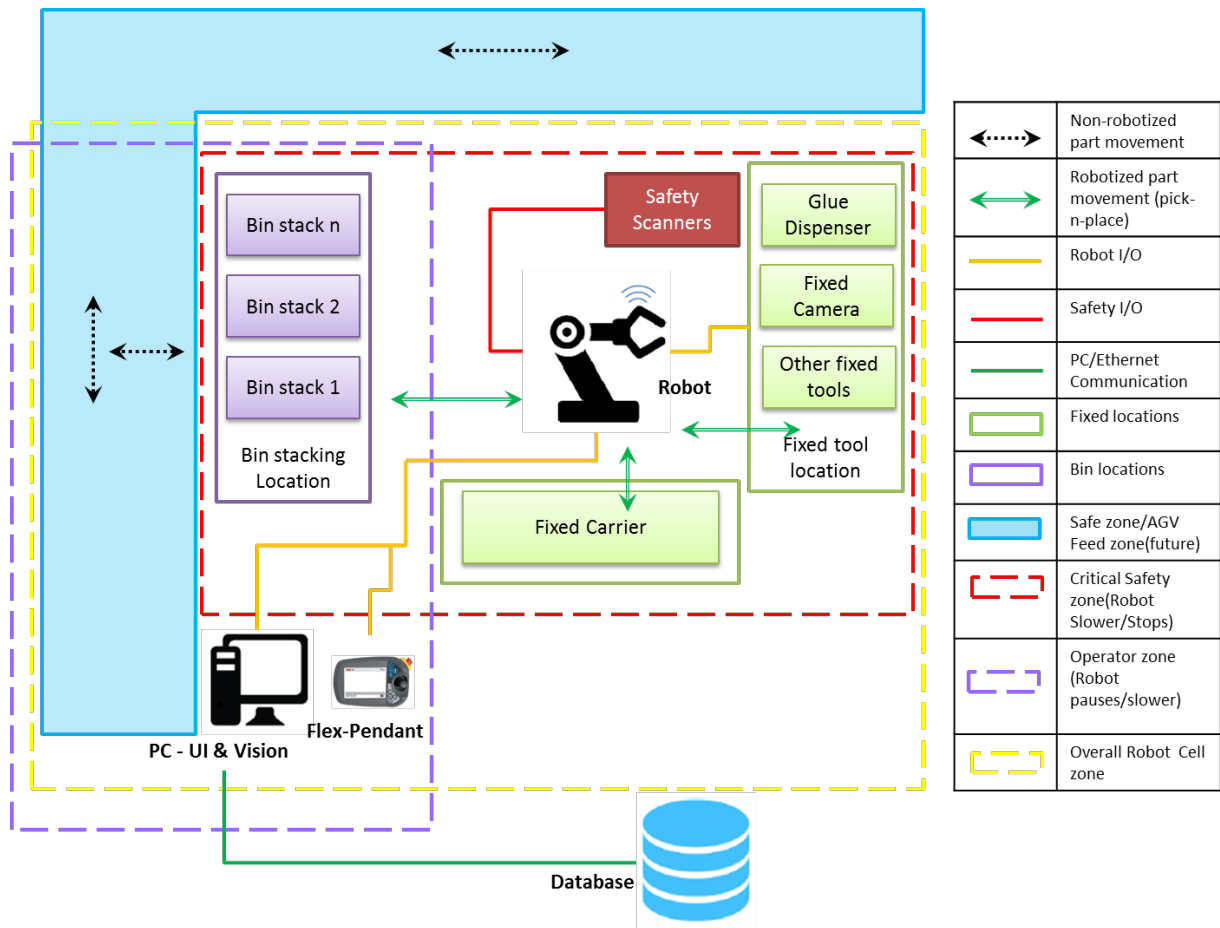


Figure 5.1: Deployment view of the assembly robot cell.

main camera. Such procedures include, for example, making sure the part has been picked correctly and examine the quality of its glue lines.

Most of the following camera calibration and stereo triangulation validation tests will be carried out with two Basler acA1600-20gc (2 MP), while the hand-eye and work-space validation tests will be accomplished with the DFK Z12GP031 as well. The reason why the tests are carried out with different cameras is both to test the application's performance for different resolutions, as well as for a matter of simplicity, since the stereo triangulation procedure should be evaluated with the use of two equal cameras.

5.2 Camera Calibration Results Validation

Concerning the developed RoboCalib application, the camera calibration procedure proposed in section 4.3 has three main practical purposes: to correct the non-linear radial lens distortion of the camera lenses, to define the real center of the image and to estimate the relative position of two cameras, required to perform 3D measurements using the stereo triangulation tool.

This section envisions to evaluate the precision, accuracy and reliability of the camera calibration

procedure with respect to its deliverables, for different situations and conditions.

Using the exported calibration tool, from RoboCalib, undistorted images were used and compared with the original distorted images. In order to quantitatively evaluate the improvements achieved with camera calibration, two tests are proposed. The first test uses the VisionPro's *CogFindLine* tool which looks for a line within a search region. This tool uses a line of VisionPro's *Caliper* tools which find the best edge point inside their search regions. Once all *Calipers* have found their edge point, the *CogFindLine* tool fits the best line through them. The distances of each point to the fitted line are also made available by the *CogFindLine* tool. These distances can be then used to quantitatively measure the present radial distortion, since lines are never straight when distortion is present.

Due to the impossibility of physically evaluating the relative position of two cameras and compare it with the results of the camera calibration procedure, the results of the stereo triangulation, which depend on the relative position of the cameras, are evaluated instead. These will indirectly give an insight on the accuracy of the positioning of the two cameras.

The 3D measurements evaluation, using the stereo triangulation tool, is accomplished by relating the results of 3D measurements with the real precise measurements.

5.2.1 Lens Distortion Correction Evaluation

The following results were achieved using the Basler acA1600-20gc with HF9HA-1B 1:1.4/9mm Fujinon lenses, at an approximate height of 100mm.

With a barrel type distortion, found in this camera and in most industrial cameras, a horizontal or vertical line which goes through the center is not severely affected by distortion, but a horizontal or vertical line close to the borders of the image is deeply affected by it. To measure the curvature of a distorted line, the Cognex's *CogFindLine* was used. Once the *Calipers* have returned the set of points along the edge, the *CogFindLineTool* fits the best line through the points, using a linear regression. An example of this tool's result is illustrated in Figure 5.2, where the green line is the best fitted line for the set of green points returned by the *Calipers*.



Figure 5.2: *CogFindLine* result on a distorted line.

As previously mentioned, the *CogFindLine* also estimates the RMS error of the fitting and the distance from every *Caliper* point to the fitted line. These values are used to quantitatively validate the inverse distortion model used to correct the images. The plot on Figure 5.3 plots the error of each *Caliper* on a distorted horizontal line, distorted vertical line, undistorted horizontal line and undistorted vertical line.

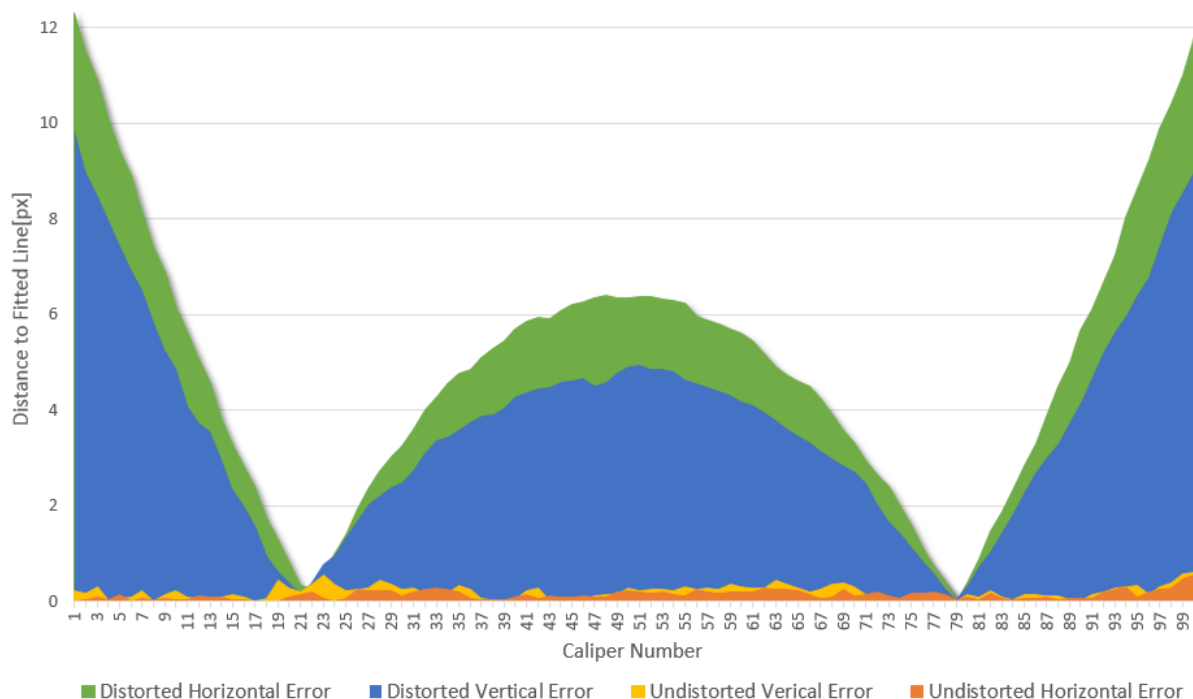


Figure 5.3: *Caliper* errors for distorted and undistorted lines.

A quick visual inspection the plot would lead to a fallacious conclusion, since being the distortion error for a horizontal line bigger, one would consider distortion more severe horizontally, which is not true. This is due to the fact that the *CogFindLine* used to find the horizontal line was larger than the one used vertically, due to the ratio of the image. This causes the fitted line to significantly move down and to consequently increase the error of the *Calipers*.

More importantly, the plot on Figure 5.3 validates one of the most important deliverables of distortion correction, making lines straight opposed to curved. The eliminated error does not correspond directly to the amount of eliminated distortion, but it is implicit that a perfect straight line corresponds to no distortion in the image. Table 5.1 quantitatively summarizes the RMS error of each *CogFindLine* tool.

	Horizontal Line RMS [px]	Vertical Line RMS [px]
Distorted	5.6557	4.3329
Undistorted	0.1626	0.2341
Eliminated Percentage of Error	97.1247%	94.5972%

Table 5.1: RMS of each *CogFindLine* tool.

It is concluded, after analyzing these results, that the tool eliminated distortion successfully and that the approximately 2.9% and 5.4% missing are due to noise in the images, considering that the edge transition usually consists of more than one pixel as illustrated in Figure 5.4. These transitions are not clear enough to define a transition with a better accuracy, having therefore an error of 0.16 and 0.23 pixels, respectively.

Even with these promising results, it is not possible to correctly validate and evaluate the perfor-

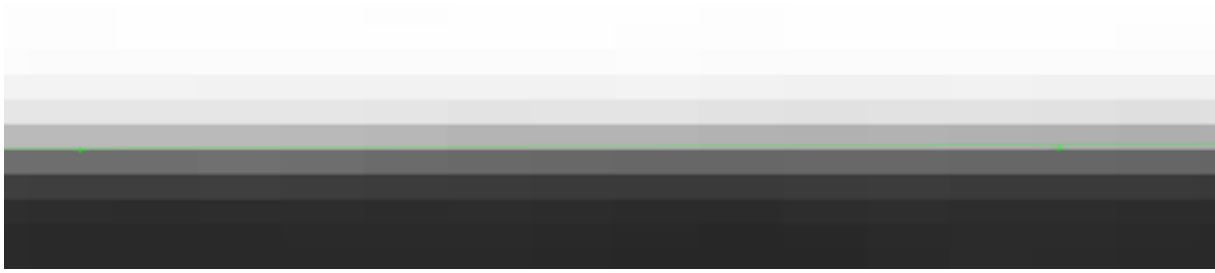


Figure 5.4: Color progression on an image's edge.

mance of the distortion correction tool, since there is no metric to evaluate if the image's scale and skew was corrected and not affected. Thus, one more test where 2D measurements are estimated and compared between a distorted and undistorted image, is proposed. Using an object with precise known dimensions, after knowing an approximate relation between the pixels' size and a physical unit, such as mm, it is possible to characterize the performance of the distortion correction across the FOV, by making linear consecutive measurements from the center to an image's corner.

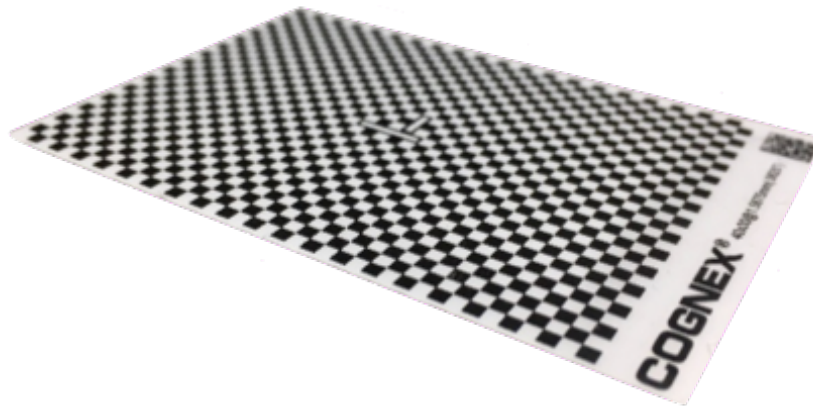


Figure 5.5: Cognex's proprietary calibration plate.

In order to estimate the relation between the pixels' size and a physical unit, one of Cognex's proprietary calibration plate was used. Together with the *CogCalibCheckerboard* tool, which uses one image of the calibration plate and the size of each of its squares, to create a new coordinate system with physical dimensions, rather than pixels' size. For the undistorted image, this will produce a good result, since all squares of the calibration plate will have the same size, but for the distorted images, since all squares will vary in size, due to the radial distortion, the tool will produce an inaccurate transformation. This is not an issue, since the focus of this test is not to evaluate the progression of the error, across the FOV, for the distorted images, but only for the undistorted ones. Therefore, the only purpose of the measurements done on the distorted image, is to analyze the exponential effect of distortion in an image.

Results once again validate the improvements of the camera calibration procedure, with respect to distortion correction. It is shown that for distorted images, the measuring error increases exponentially along with the distance from the center, as predicted. In this case, with a FOV of about 120 cm^2 , distortion can cause measuring errors up to about 2 mm, which is approximately 2.5% of the image's diagonal.

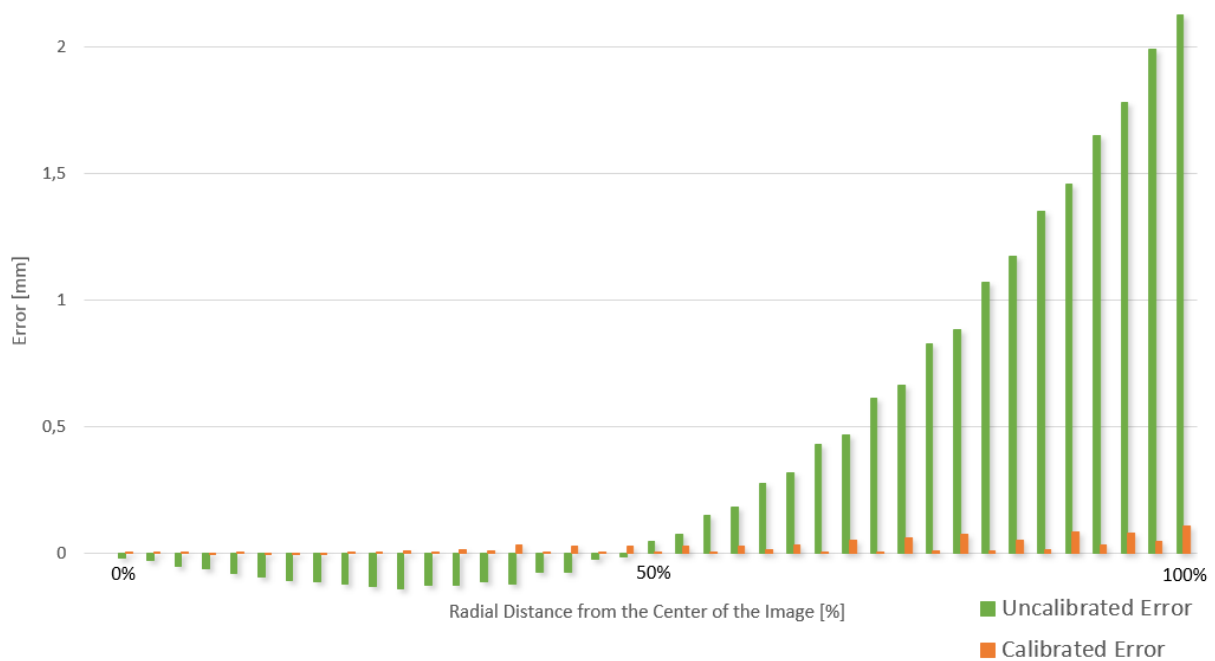


Figure 5.6: Error across the FOV for distorted and undistorted images.

For highly precise applications, such as the ones carried out at AME, where for some applications an overall accuracy of 0.05 mm is required, this error would produce a non-conforming product assembly and most probably damage it during the assembly.

RoboCalib's solution improved the measurements' accuracy by an average factor of 23, when compared with the distorted images. Furthermore, the calibrated camera measurements had a maximum error of 0.1 mm, which was only measured close to the image's corner, where distortion is more severe.

Improvements aside, it is important to note that even for undistorted images, the measuring errors increase exponentially, although not significantly. Nonetheless, this behavior was expected, considering that the distortion correction process always produces worse results close to the image borders. This is due to the fact of information being severely condensed near the borders of the image, due to distortion. Therefore, when correcting the image, there will be less information to re-construct the image where distortion was present. Thus, the undistorted image tends to become blurrier along the FOV, which consequently makes the process of precisely finding a feature less accurate and reliable.

5.2.2 Camera Center Estimation Precision

The center of an image ($width/2$, $height/2$) is not always the actual intersection between the focal plane and the optical axis. Mechanical tolerances and misalignments, as well as imperfections in the lenses tend to change the orientation of the optical axis, consequently misaligning the camera's center with the sensor's center.

When normal lenses are used, instead of telecentric lenses, where image magnification is independent of an object's distance or position in the FOV, knowing the camera's exact center is crucial. The

first reason of its tremendous importance is the need to center the robot's camera with an object. In a robotic's application that requires to manipulate a certain object, and to pick it up exactly in its center, the camera's optical center must intersect the object's center and be perfectly perpendicular to its surface. This is required, since the only known transformation between the camera and the manipulator, given by the hand-eye calibration, is between its optical center and the manipulator. Thus, for an object to be precisely picked up in its center, the camera center needs to be centered with the object's center as well.

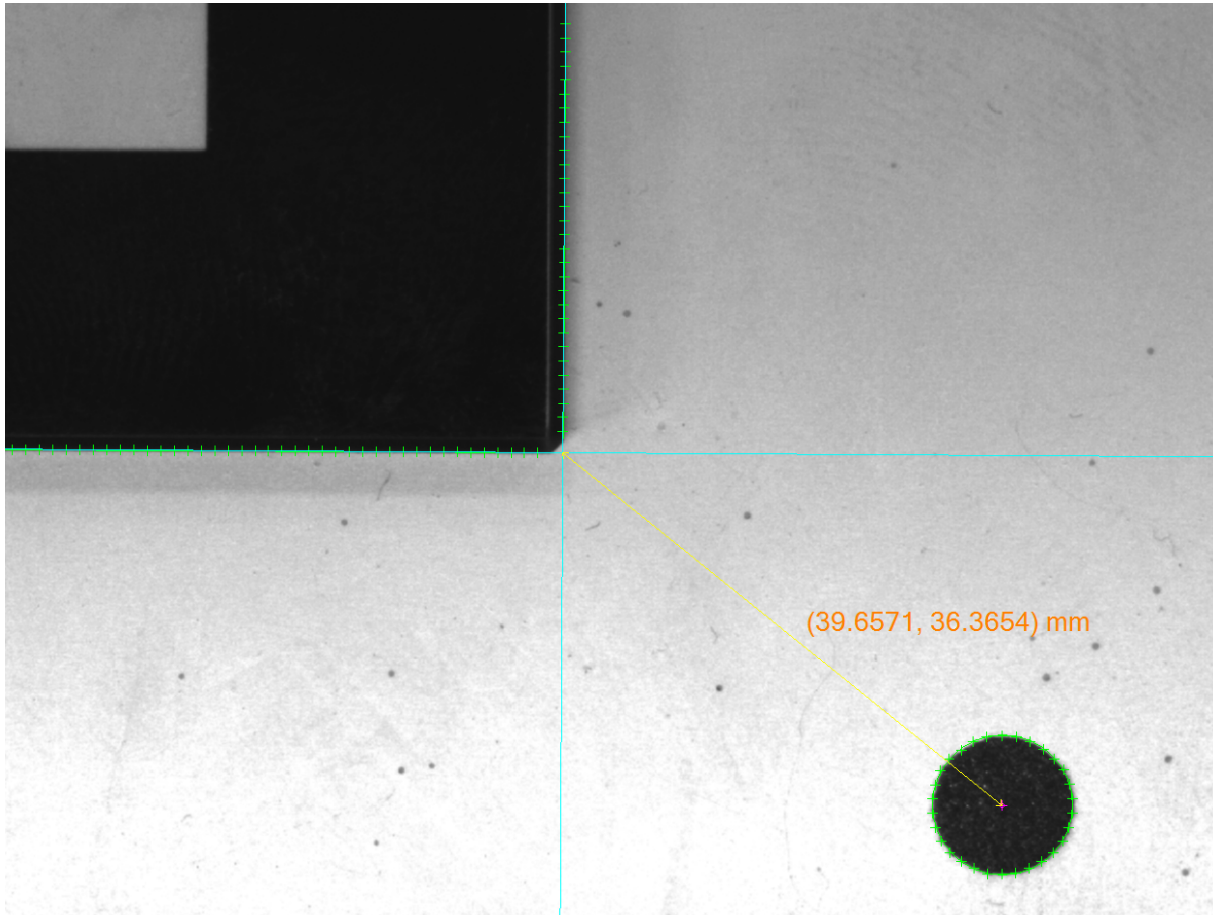


Figure 5.7: Measuring a part's corner distance to a reference dot.

In some applications, such as the one illustrated in Figure 5.7, measuring the distance between an object's corner and a reference point is required. Due to the fact of some of these objects having a considerable height a measurement will not be accurate, unless the correct camera center is over the object's corner. Otherwise, its height will affect the results due to perspective and magnification, such as the example illustrated in Figure 5.8.

This section does not test the accuracy of the camera center estimation, but its precision. The camera center accuracy can only be tested once the transformation between the camera and the robot is known, product of the hand-eye calibration. Therefore, section 5.3.2 gives a better insight on the accuracy of the camera center estimation.

In this proposed test, two cameras (a DFK with 5 MP and a Basler with 2 MP) were used with different focal lengths in order to test the precision of the camera center estimation for different settings.

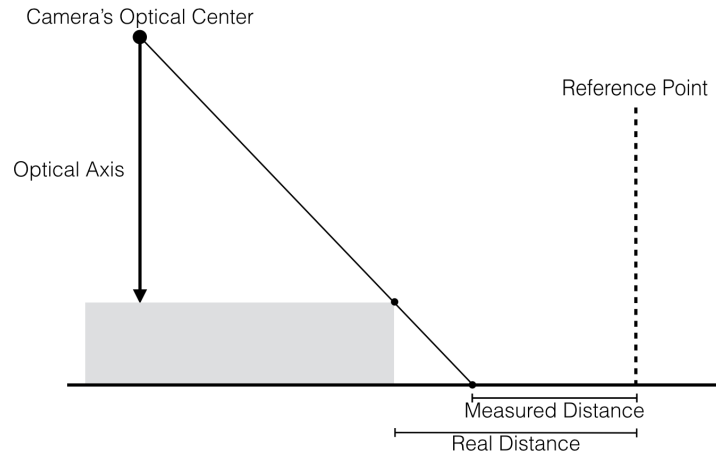


Figure 5.8: Measurement inaccuracy when the camera is not centered with the object's corner.

Although having used different camera resolutions and focal lengths, the size of the FOV for all settings was set to be the same, by tweaking the cameras' height. The statistical results of these tests, illustrated on Figure 5.9, show that the bigger the resolution, the better the precision.

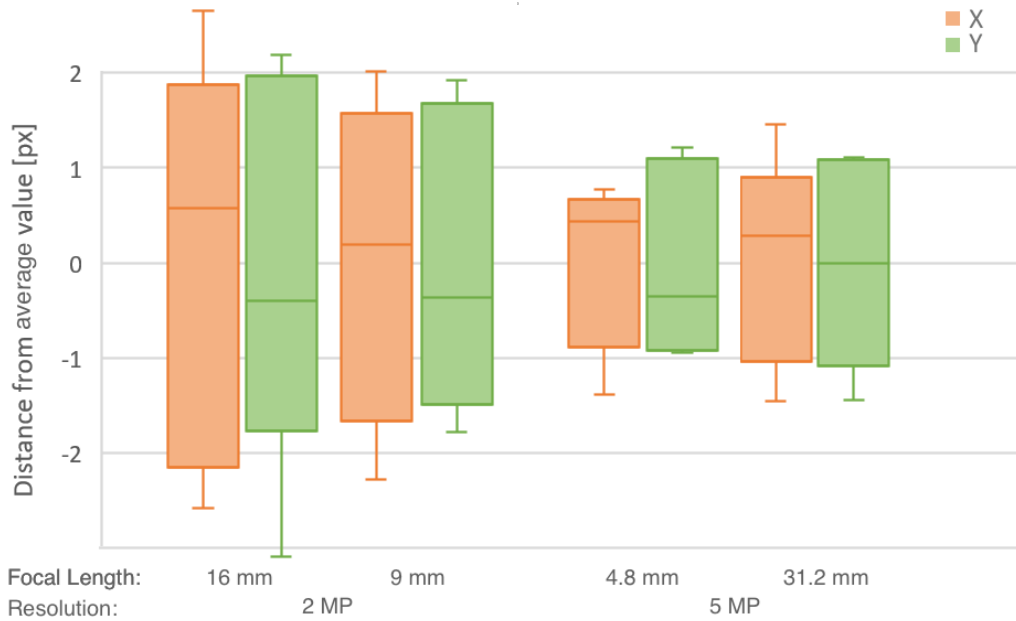


Figure 5.9: Statistical results of the camera center estimation for different settings.

These results were once more expected, since a camera with a higher resolution is able to acquire images with more detail and information, resulting in a more accurate and precise calibration. Thus, the results of the 5 MP camera were expected to be an improvement of around $\sqrt{2.5}$ times (pixel ratio) when compared to the 2 MP camera results. This was not exactly the result, since the ratio between the RMS of the results of the two cameras was of approximately 1.83. This difference of about 0.25, between the expected result and the actual result, is the result of other variables that are difficult to predict. These variables are the result of the different characteristics of the two cameras, such as SNR (Signal to Noise

Ratio) or lens quality.

Moreover, it is also proved that different focal lengths do not affect significantly the precision of the camera center estimation. Since the same lenses were used, and only the focal length changed, the image quality is not expected to be affected.

5.2.3 Stereo Triangulation Accuracy using the Camera Calibrations

The last deliverable of a camera calibration procedure is the possibility of triangulating a point in 3D using two calibrated cameras, in a stereo setup. Even further, stereo triangulation introduces the possibility of measuring the distance between two points with a bigger accuracy than before, including points spaced at different heights. The only requirement of stereo triangulation is having the camera calibration performed for each camera.

This section presents the behavior of the 3D measurements's accuracy, along the cameras' FOV. For these tests, only camera calibration is used, since distances are always calculated using the Phys3D space. The results are expected to behave as described in section 2.6 and as Schreve [24] has detailed. Moreover, the results are also expected to behave accordingly to the results presented in Figure 5.6, where accuracy slightly decreases along the FOV.

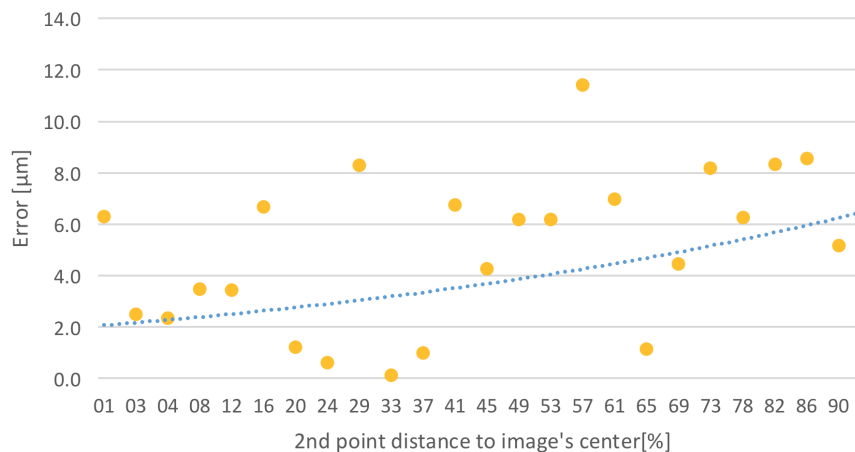


Figure 5.10: Accuracy of 3D measurements performed.

In order to characterize the accuracy of stereo triangulation along the FOV, the same method used in section 5.2.1 is used. Measurements of the Cognex's proprietary calibration plate were performed and analyzed for a stereo setup with the two Basler cameras (both 2 MP). The calibration plate is first centered in both cameras' FOV and placed so that it is focused in both cameras. Following, measurements are performed along the diagonal of the FOV.

Although being the cameras' resolution smaller (2MP) than the DFK (5MP), the measurements' accuracy turned out to be much better. The DFK camera alone was able to perform the same measurements with an average error of approximately 22 μm , while the two Baslers performed them with an average

error of 5 μm . It is then concluded, that even for smaller camera resolutions, stereo triangulation is capable of achieving better measuring results than a single camera.

5.2.4 Camera Calibration Validation Conclusions

The camera calibration was proven to accurately and precisely offer its deliverables. Lens distortion was efficiently mitigated, with up to 97% of the distortion corrected, resulting in measurement improvements with an average factor of 23. The camera center was proven to be always estimated with an adequate precision. Its accuracy results, while not studied in this section, can be found in section 5.3.2. Moreover, stereo triangulation, using only the camera calibrations, demonstrated to be far more accurate than single camera measurements, with an average accuracy of 5 μm .

5.3 Hand-Eye Calibration Results Validation

The hand-eye calibration procedure, as proposed in section 4.3.2, has one main deliverable: the estimation of the transformation between Camera3D and Hand3D spaces. This transformation is then used for four main applications: defining the cameras' correct *tooldata* in the robot's application, positioning the camera's optical axis exactly perpendicular to an object's surface, offsetting the camera's position in the direction of its X and Y pixel coordinates and relating a camera's point to the robot's frame.

This section's aim is to evaluate the accuracy and precision of the hand-eye calibration results.

5.3.1 Camera Transformation Estimation Precision

In this section, the precision of the camera's transformation estimation is studied, and its results presented statistically. The camera transformation, position and orientation with respect to the Hand3D space, was therefore calibrated and estimated repeatedly for a fixed focal length of the DFK camera. The results are illustrated in Figure 5.11.

The precisions associated to these tests, using an average deviation, were of ± 0.022 mm, ± 0.007 mm, ± 0.015 mm for X, Y and Z, respectively. With respect to the orientation, the precisions were calculated to be $\pm 1.71\text{E-}05$, $\pm 1.67\text{E-}05$, $\pm 7.31\text{E-}05$ and $\pm 1.74\text{E-}05$ for q1, q2, q3 and q4 respectively.

Both the position and the orientation's precision are considered to be sufficient and adequate for industrial applications, since their uncertainty is in the same magnitude of the camera's pixel size and robot's accuracy. Furthermore, the same test was carried on with the Basler camera, and the ratio (approximately 1.83) found in section 5.2.2, between the 2 MP and 5 MP cameras, was again obtained. This demonstrates that the calibration procedures' results are linearly affected by resolution.

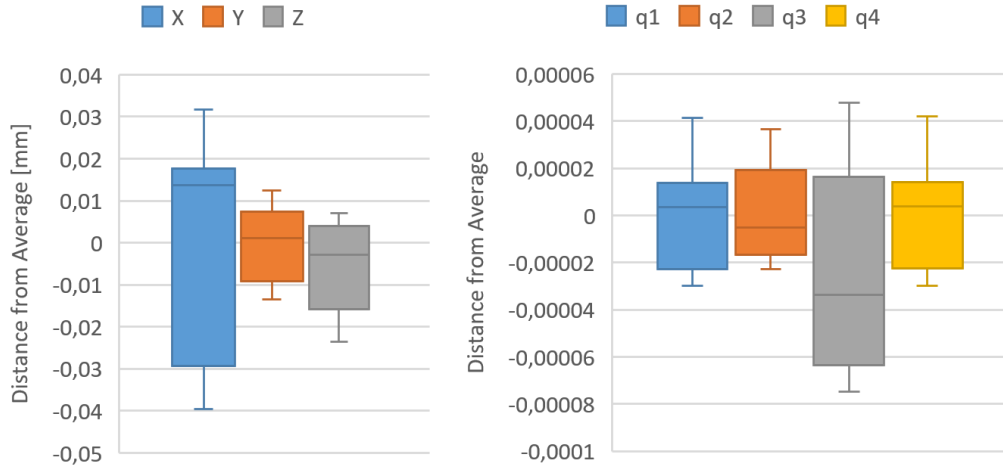


Figure 5.11: Statistical results of the camera transformation estimation for a constant focal length.

5.3.2 Camera Transformation Estimation Accuracy

Although not being possible to precisely quantitatively evaluate the accuracy of the hand-eye results, it is possible to characterize it. This section proposes two tests to compare an uncalibrated camera with a calibrated one.

The first proposed test, developed to characterize the position's accuracy of the hand-eye calibration, proposes rotating an object over its center, using the robot's suction cups. The displacement of that object, after the rotation, quantifies the error of the position of the camera, since the camera was used to center the object.

The second proposed test, which aims to characterize the orientation's accuracy of the hand-eye calibration, rotates the camera over its Z axis. In an ideal situation, all points in a perpendicular plane to the camera should rotate around the camera's center. Therefore, the camera is oriented to be exactly perpendicular to a plane and then rotated 180° . Every 4° a vision job is executed to calculate the position of a fixed feature, with respect to the camera center. For an ideal calibration, that feature should rotate around the camera's center at a constant distance.

Hand-Eye Position Accuracy

The first step is to center an object with the camera, which means using a vision job to detect its center and exactly move the camera to overlap the camera's center (result of camera calibration) with the vision job's estimated object center. Once the camera is centered with the object, the robot moves its manipulator to the exact position where the camera was previously positioned. This movement's accuracy totally relies on the transformation's accuracy, meaning that for a perfect hand-eye calibration result, the robot's manipulator would be exactly centered with the object and that for an inaccurate hand-eye calibration the object would be slightly displaced.

After positioning the manipulator on the camera's last position, the robot moves it in the direction of

the object until it is able to grasp it using its suction cups. The object is now slightly elevated, rotated 90° degrees and laid again. The camera is moved once more to its previous position.

The vision job to detect the object's center, with respect to the camera center, is executed again. For an exact hand-eye calibration result, the object would have been rotated but not moved, but for an inaccurate hand-eye calibration the object will not only be rotated but moved as well, because the position where the object was picked by the suction cups, was not its center.

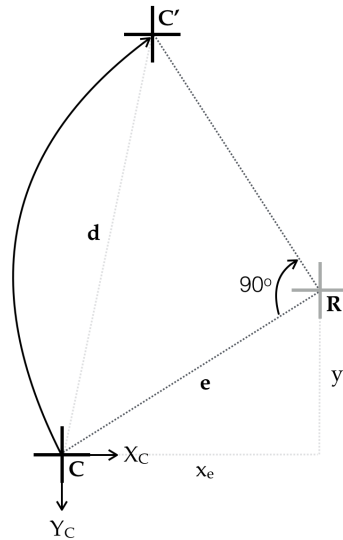


Figure 5.12: Object's displacement when it is rotated around a point that it is not its center.

It is then possible to quantitatively relate the inaccuracy of the hand-eye calibration result with the displacement of the object. Take d , of Figure 5.12, the distance between the camera center (where the object's center C was positioned before being rotated around the center of rotation R) and the current center of the rotated object C' . The distance between the rotation center and the camera's center is approximately the calibration error. It is only an approximation since the optical axis would have to be perfectly perpendicular to the object's surface for the error to be exactly equal to d . This error, equal to $d/\sqrt{2}$, can even be split into x_e and y_e , which then can be used to improve the camera's estimated position.

The test was performed at different heights, with the purpose of comparing the uncalibrated camera's *tooldata*, which is a rough estimation of the camera's center from a CAD drawing, and a calibrated camera's *tooldata*. The results illustrated in Figure 5.13 show improvements of up to 6 times.

Although expecting the calibrated camera not to produce any error, several factors influence the accuracy of this test. Among them, and most importantly, is the accuracy of the vision job responsible for estimating the object's center, the robot's motion accuracy and the suction cup's accuracy.

It is also important to mention that this error has no influence in a usual assembly application, if and only if the picked up object is not rotated. If the object is picked up, moved to the top of its respective

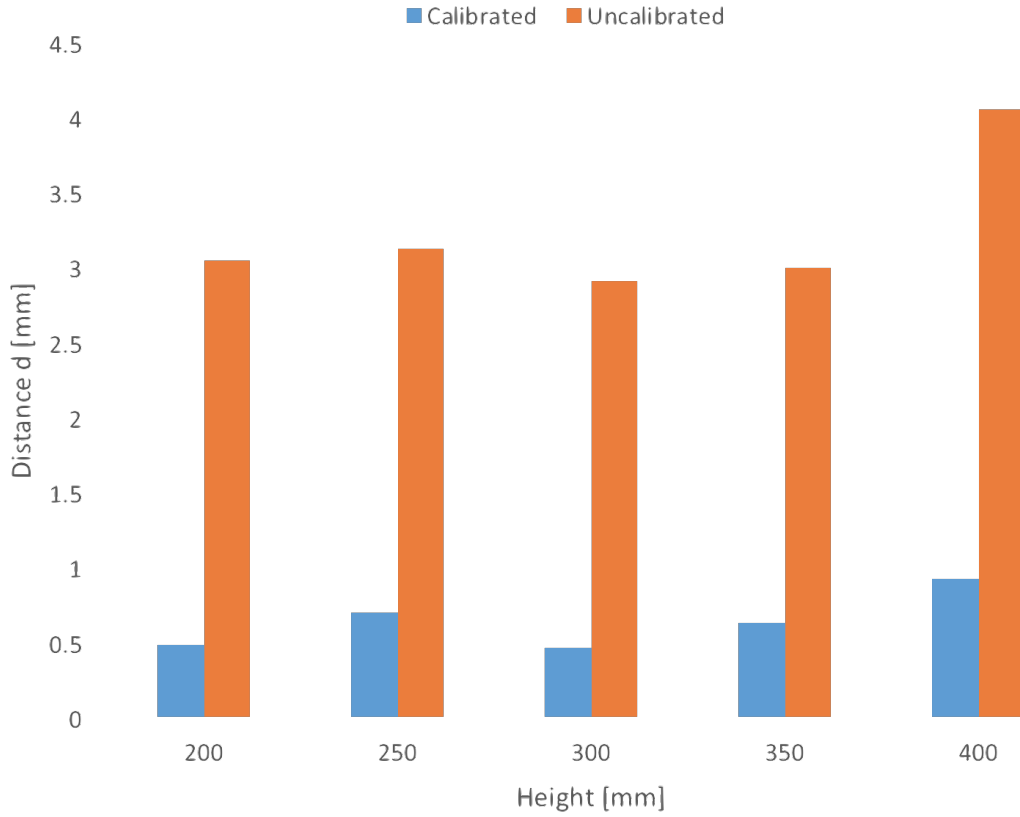


Figure 5.13: Respective distance d , for different heights and for a calibrated and uncalibrated cameras.

part and assembled, without the need to rotate it, the alignment of the two parts' center will be flawless. Nonetheless, this situation is very unlikely to happen, since the two parts are rarely aligned. Usually it is required to rotate the part about 1° to 2° . The displacement error, can be calculated using formula 5.1, which means that for the uncalibrated camera, and for a part which requires to be rotated 2° , the displacement error will be around 0.081 mm, but for the calibrated camera it is only of 0.017 mm. Some assembly applications require up to 0.05mm of an overall accuracy, meaning that an uncalibrated camera would produce a non-conforming part, just from rotating the object slightly.

$$2\sin\left(\frac{\theta}{2}\right)\frac{d}{\sqrt{2}} \quad (5.1)$$

Hand-Eye Orientation Accuracy

While the first test focuses on the accuracy of the position of the camera's transformation, with respect to the Hand3D space, a second test is proposed to evaluate the orientation of the hand-eye results. For an exact hand-eye result, any rotation around the camera's Z axis should not move the camera center from its relative position in the image. In other words, for exact hand-eye results, any point in the FOV of the camera, should move circularly around the camera center when the camera is rotating. The second proposed test therefore consists of rotating the camera around its Z axis and log the position of a feature with respect to the camera center.

If the orientation result of the hand-eye calibration is not exactly the camera's orientation, the feature

will not move circularly around the camera center. Moreover, if the camera's axis is not perfectly perpendicular to the feature's surface, the movement will also not be perfectly circular. Despite this fact, the aim of this test is again to relate the performance of a calibrated camera with an uncalibrated one, and therefore having a perfect normal surface to the camera's optical axis is not strictly required.

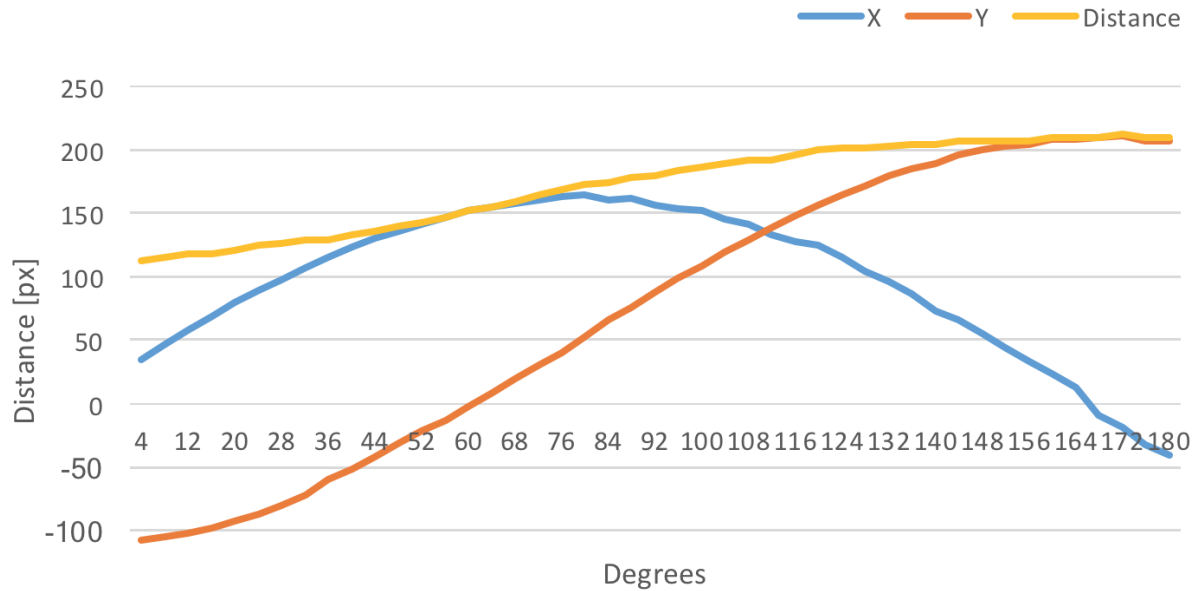


Figure 5.14: Orientation accuracy test results for an uncalibrated camera.

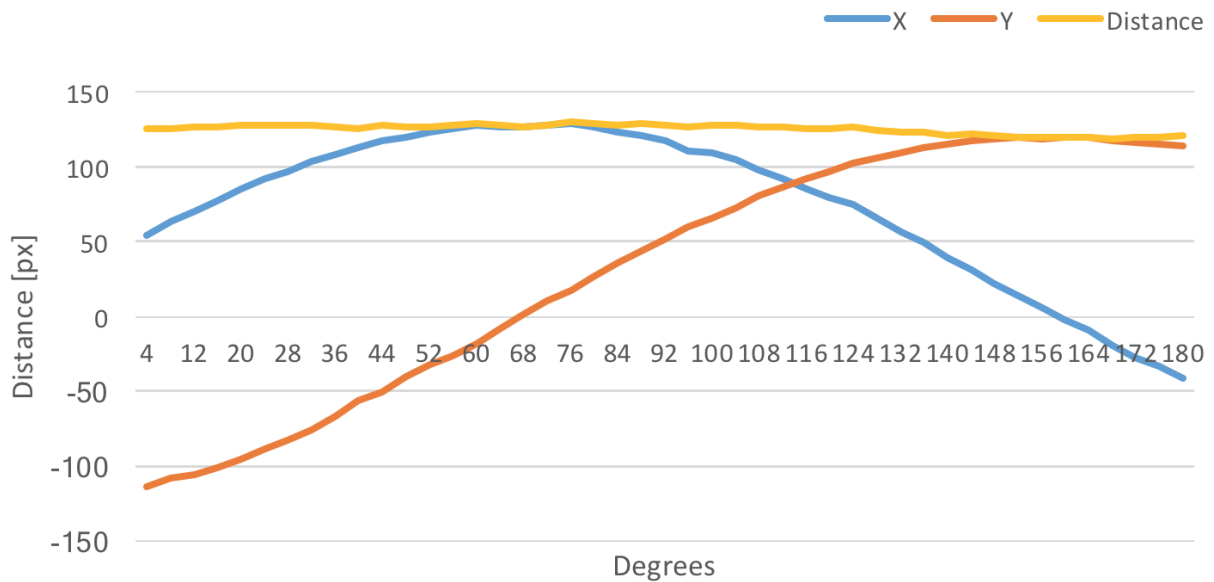


Figure 5.15: Orientation accuracy test results for a calibrated camera.

The camera was rotated 180° in steps of 4° . Each step, a vision job was executed to find the feature's position. The results for the calibrated and uncalibrated camera were then plotted in Figures 5.15 and 5.14, respectively. In order to evaluate this test's performance, the X and Y values of the position of the feature were logged, and its distance to the camera center calculated. The better the calibration the more constant is this distance, since the feature should move circularly around the camera center.

The results show that the uncalibrated camera test did not result in a circular movement, but the calibrated camera did, approximately. The distance at which the feature was from the camera center varied by 99 pixels for the uncalibrated camera, but only 10 pixels for the calibrated one. Furthermore, their average error was of 29.97 pixels and 2.69 pixels, respectively.

These results represent an improvement by a factor of 11, when comparing the orientation of a calibrated and uncalibrated camera. This improvement was also expected to be bigger than the improvement on the accuracy of the position of the camera, since the uncalibrated position of the camera was deducted from a CAD file, but the orientation was simply set to be the same as the camera's casing.

Moreover, these results are also important to validate the accuracy of the camera center estimation, product of the camera calibration. Even with a correct orientation result, if the camera center was incorrect, the results would not have been as precise as the ones obtained.

5.3.3 Stereo Triangulation Accuracy using the Hand-Eye Calibrations

Stereo triangulation using the hand-eye calibrations has the same accuracy as the results on section 5.2.3, with respect to 3D measurements' accuracy. Although, this section does not focus on the accuracy of 3D measurements performed with the hand-eye calibrations, instead it aims to evaluate the accuracy of the triangulated 3D points in the RobotBase3D space.

Therefore, a very precise tool, shaped as a needle, was attached to the robot. This tool is used to touch certain positions in space and easily retrieve their 3D positions in the RobotBase3D. Then, the stereo triangulation tool was used in order to estimate these same positions in 3D, through triangulation. The results pointed to an average error of 0.269 mm. Although being a considerable error, it is acceptable for most applications, since this tool is mostly used to define the tools' position.

Among the sources of this error, is the process of retrieving the positions with the precise needle tool, which is aligned by eye, and the inaccuracy of the robot.

5.3.4 Hand-Eye Calibration Validation Conclusions

Hand-eye calibration was proven to accurately and precisely offer its deliverables. Its position estimation was demonstrated to be precise up to 7 μm and its accuracy to produce results six times better than an uncalibrated set-up. Regarding the orientation results, improvements regarding its accuracy were even greater, improving the camera's real orientation by up to eleven times. Regarding stereo triangulation, it was shown that the tool is adequate to be used to estimate work-space tools with a good accuracy.

5.4 Practical Applications

This section presents some practical applications of the camera and hand-eye calibration in industrial applications, as well as its benefits and comparison between the previous methods' results and the new results.

5.4.1 Defining a *tooldata*

As mentioned in previously presented sections, the process of defining and/or calibrating a tool's *tooldata* is excessively inaccurate. It relies on finding an approximate position of the tool, and then through a process of trial and error updating its position until it suits the accuracy of its assigned purpose. Furthermore, most tools wear down due to an extensive use, or simply are required to be exchanged from time to time. Therefore, the tool's *tooldata* has to be constantly updated.

At the moment, the tool's calibration methods rely on positioning the robot's camera where it has an unobstructed and discernible view of the tool to be calibrated. Two different methods can then be used:

1. Ask the operator to tweak the tool's position, by making fine adjustments to it, until the tool's 2D position in the camera is within a pre-defined threshold, with respect to its reference position.
2. Update the camera's *tooldata* with the displacement values obtained by the camera.

Both methods fit their purposes, but the first method is very time-consuming and the second method has many flaws, such as only detecting the tool's displacement in two dimensions. Moreover, both methods require to be calibrated using the Cognex *CogCalibCheckerboard* tool in order to be able to relate pixels with mm. This means that the calibration plate must be positioned exactly at the same plane as the tool, which can be an obstacle difficult to overcome, as well as unreliable.

Stereo triangulation is the proposed solution to this obstacle. Once the two cameras have been calibrated (both camera and hand-eye calibration), the stereo triangulation UI can be easily used to retrieve the new position of the tool in the RobotBase3D. To facilitate the process, it is possible to load the vision jobs that detect the tool's position, which had been previously created and saved. Although considerably slower, than the second method, it is a much more accurate process.

To have a better understanding on how stereo triangulation and method 2 perform, a test is proposed. In this test, a needle is used as the tool to be triangulated. The needle is moved twice. For the first time, the needle is slightly unscrewed from its fixture. The second time the needle is pushed in the opposite direction of the camera. Therefore, for the first situation the cameras should detect a small displacement in the needle's height, and for the second situation a small displacement in the opposite direction of the main camera.

Table 5.2 includes the three points acquired during the proposed test, being the first the position before the two needle displacements. In this situation, y_{camera} is approximately oriented in the same direction as the negative RobotBase3D's Z axis and x_{camera} in the same direction as RobotBase3D's X axis.

x_{camera}	y_{camera}	X	Y	Z	d_{camera}	D
-49.249	13.116	1044.683	1151.593	1522.788	3.577	3.767
-49.289	16.692	1044.702	1152.262	1519.080	0.665	1.642
-49.398	16.036	1044.628	1153.895	1518.9272		

Table 5.2: Tool position result for the camera (2D) and for the stereo triangulation (3D) in mm.

Thus, for the first needle displacement, it was expected for the value of the needle to increase in y_{camera} and to decrease in Z. The needle is also prone to move in other directions when it is being unscrewed, and for that reason there was also a displacement of about 0.6 mm in the Y direction, which was not detected by the camera 2D measurements.

The second needle displacement, illustrates this issue even further. The needle was pushed only in the Y direction, which for the camera consists in a small displacement of approximately 0.7 mm in height, when in reality, the movement was of 1.6 mm in another direction.

In conclusion, the developed stereo triangulation tool not only improves the accuracy of this procedure, but also its reliability. It enables to correct the tools' positions in three dimensions, rather than just two and comes with only one trade-off - the need of an extra camera.

5.4.2 Defining a *workObject*

Defining a *workObject* requires a point and an orientation. Usually, defining a *workObject* consists of finding three point of its surface. This procedure is required to be very precise, since it is imperial for the orientation of a *workObject* to exactly correspond to the respective surface's orientation. If not, the robot will not be able to place the suction cups exactly parallel to an object (which ensures the object will not be displaced when picked up) and the calibrated camera optical axis will not be normal to the surface. These two aspects had been previously discussed, but other problems arise when the *workObject*'s orientation is not the same as the surface:

- Robot XY movements over the surface will not be parallel to the surface, incurring in an erroneous travelled distance.
- Too much or too less pressure, or stress, might be applied to an object, due to the misalignment between the suction cups and the object's surface.

Even further, when using a patterned surface, such as in Figure 5.16, it is important for the XY directions of the *workObject* to be aligned with the pattern.

In order to precisely calibrate a *workObject*, special tools are required. One of these tools, is a sharp needle which is then attached to the robot's arm. Three other needles are attached to the surface that requires to be calibrated. The procedure requires an operator to precisely move the robot's needle over the other three stationary needles and orient it so that the needles have the same orientation. Checking that the needles have the same orientation is done by sliding a cylinder from one needle to the other.

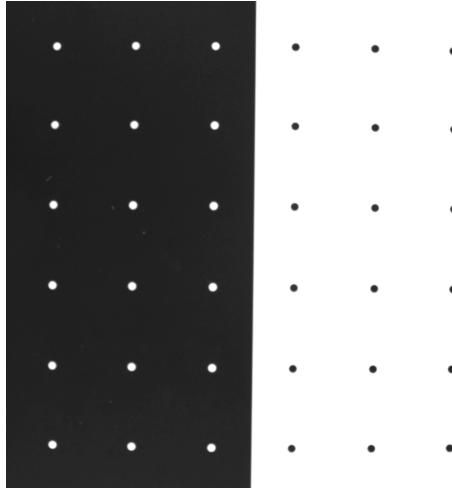


Figure 5.16: Example of an usual patterned surface used in robotic applications.

This cylinder's inside has a very small tolerance, and will only slide if both needles have the same orientation.

This method is very precise, but excessively time consuming, and therefore most times avoidable. This project proposes once more the use of the stereo triangulation tool to facilitate and to shorten the duration of this process. More importantly, it rules out the need of special tools.

The stereo triangulation tool can be used to triangulate the three required points on the *workObject*'s surface. In addition to reduce the procedure's duration from about half an hour to minutes, it also does not require to adjust the *workObject*'s height after being computed, due to the height of the needles.

The angle between the planes of the resulting *workObjects*, for the needle and the stereo triangulation procedure, was only of 0.017° . This small difference proves that the stereo calibration tool can be used for the purpose of *workObject* calibration, with a good accuracy and a huge improvement in the procedure's duration.

Chapter 6

Conclusions

This chapter finalises this report and presents the conclusions of this project, as well as the achievements this project was able to obtain. Future work and objectives, relative to this project's scope, are proposed and discussed.

The main goal of this thesis project was to develop a software suite comprised of camera calibration tools. The ability of easily calibrating industrial cameras is a must for precise robot applications, like the ones carried at AME, and without it robots are not able to precisely relate its camera points to their own coordinate system. Many works in this field have focused on camera and hand-eye calibration procedures, but there is a lack of a unified solution. Moreover, this project also focused on other calibration tools that make use of camera and hand-eye calibrations, such as the stereo triangulation and work-space tool. These tools allow this project to close the gap on the automatic calibration scope of industrial vision guided robot cells.

RoboCalib, the proposed application, was developed taking into account all specifications, processes and philosophies of the company. From standardization and universality, ensuring that the application is able to calibrate any camera in any robot, to modularity, which makes use of an object oriented programming language to easily structure the program and allow other users to easily implement RoboCalib's code in their projects. Furthermore, modularity is also about offering this project's deliverables as standalone programs or applications that can also be easily implemented in other projects.

Additionally, this project's three main objectives have been delivered, as this project's solution offers automatic camera, hand-eye and work-space calibration procedures.

Moreover, new robotic applications, that are now being developed, take into consideration the work of this project and rely on it to calibrate any necessary cameras. Tools that need to be precisely calibrated use stereo triangulation in order to estimate its position, and the automatic work-space calibration tool is used to update the tools' position before starting a production cycle.

6.1 Achievements

This project's solution demonstrated that all calibration procedures were able to improve the current situation.

The camera calibration tests demonstrated that this particular calibration is able to mitigate radial distortion up to 97% and improve 2D measurements by up to a factor of 23, when compared to measurements carried out with distortion. More importantly, it determined that 3D measurements, using only each camera's camera calibration, have an average accuracy of 5 μm , with two cameras of 2 MP. As a reference to be compared with, a camera with 5 MP alone is only capable of measuring the same points (although in 2D) with an average accuracy of 22 μm .

The hand-eye calibration tests reported its position estimation to be precise up to 7 μm and its accuracy to produce results six times better than an uncalibrated set-up. Its orientation results, were proven to be an even bigger improvement, by improving the camera's real orientation by up to eleven times. Additionally, the transformation of a triangulated 3D point from the Phys3D space to the robot's RobotBase3D was shown to have an average accuracy of at least 0.269 mm. This accuracy, although smaller than the other results, it is more than acceptable, if used to estimate the work-space tools' position.

In addition, this project's solution proposes a new procedure to calibrate a *workObject*. This new procedure proved that using the stereo triangulation tool it is possible to define a *workObject* 15 times faster, and with an error of only 0.017° between each *workObject*'s planes.

6.2 Future Work

Despite the successful development and accomplishment of all the deliverables intended for this project, there is still some planned future work within this project's scope.

Firstly, there is a lack of different sized calibration plates. Some calibration procedures are affected by this because their cameras' FOV is not completely filled by the calibration plate. Having custom sized calibration plates would improve most calibration tools, such as distortion correction, since all areas of the image would be corrected using an exact and not estimated model. Another desirable and advantageous aspect of the calibration plates that could be customizable, is its tiles' size. Due to the calibration plate's required high printing accuracy, purchasing different calibration plates for each different application is not viable. Therefore, it was proposed to develop a mobile application capable of running on tablets, with a high resolution screen, that would allow an user to customize the calibration plate size as well as the dimensions of its tiles.

Besides, this application would be connected to RoboCalib as well. This connection would be used to orient the calibration image during the calibration procedures, sparing the operator's time and effort.

Moreover, and taking into consideration that the calibration procedures still take some time to be performed, it would be desirable to have a tool that could estimate different calibration results for different camera settings, without going through a calibration procedure. This would allow users to choose any settings of a zoom camera, for example, and instantaneously retrieve an estimated camera and hand-eye calibration for that specific setting. The camera calibration would be approximately the same, since the distortion model does not change much with different zoom or focus settings. Although, the camera center should be estimated, since it has been proven to change for different camera settings. But more importantly, the transformation between the camera and the Hand3D space could be estimated with a good accuracy.

Figure 6.1 shows how the hand-eye transformation result behaves for different zoom settings. The observable line is an approximation of the camera's optical axis. With a proper fitting algorithm, it would be possible to define the optical axis with just some different zoom settings. The estimated line could then be used to estimate the camera's center and pose with respect to the Hand3D space, for applications with relatively low accuracy requirements, without having to go through any calibration procedure.

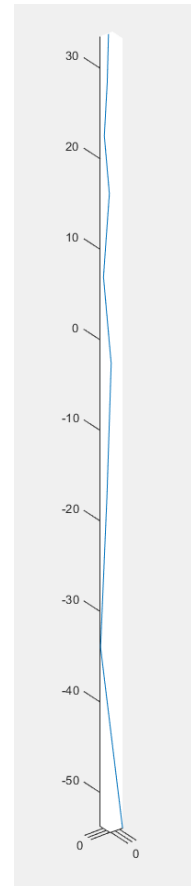


Figure 6.1: Hand-Eye transformation for different zoom settings.

Bibliography

- [1] T. Kubela, A. Pochyly, and V. Singule. Investigation of position accuracy of industrial robots and online methods for accuracy improvement in machining processes. In *Electrical Drives and Power Electronics (EDPE), 2015 International Conference on*, pages 385–388. IEEE, 2015.
- [2] Z. Roth, B. Mooring, and B. Ravani. An overview of robot calibration. *IEEE Journal on Robotics and Automation*, 3(5):377–385, 1987.
- [3] S. A. Hayati. Robot arm geometric link parameter estimation. In *Decision and Control, 1983. The 22nd IEEE Conference on*, volume 22, pages 1477–1483. IEEE, 1983.
- [4] H. Stone, A. Sanderson, and C. Neuman. Arm signature identification. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 41–48. IEEE, 1986.
- [5] J.-M. Renders, E. Rossignol, M. Becquet, and R. Hanus. Kinematic calibration and geometrical parameter identification for robots. *IEEE Transactions on robotics and automation*, 7(6):721–732, 1991.
- [6] M. Becquet. Analysis of flexibility sources in robot structure. In *IMACS/IFAC. Symp. Modeling and Simulation of Distributed Parameters, Hiroshima, Japan*, pages 419–424, 1987.
- [7] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.
- [8] H. Zhuang and Z. S. Roth. *Camera-aided robot calibration*. CRC press, 1996.
- [9] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, 1987.
- [10] G.-Q. Wei and S. De Ma. Implicit and explicit camera calibration: Theory and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):469–480, 1994.
- [11] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. Ieee, 1999.

- [12] C. C. Slama, C. Theurer, and S. W. Henriksen. *Manual of photogrammetry*. American Society of photogrammetry, 1980.
- [13] R. K. Lenz and R. Y. Tsai. Techniques for calibration of the scale factor and image center for high accuracy 3-d machine vision metrology. *IEEE Transactions on pattern analysis and machine intelligence*, 10(5):713–720, 1988.
- [14] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages 432–437. IEEE, 1999.
- [15] S. J. Maybank and O. D. Faugeras. A theory of self-calibration of a moving camera. *International Journal of Computer Vision*, 8(2):123–151, 1992.
- [16] R. I. Hartley. An algorithm for self calibration from several views. In *Cvpr*, volume 94, pages 908–912. Citeseer, 1994.
- [17] Q.-T. Luong and O. D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of computer vision*, 22(3):261–289, 1997.
- [18] J. C. Chou and M. Kamel. Finding the position and orientation of a sensor on a robot manipulator using quaternions. *The international journal of robotics research*, 10(3):240–254, 1991.
- [19] H. H. Chen. A screw motion approach to uniqueness analysis of head-eye geometry. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 145–151. IEEE, 1991.
- [20] Y. C. Shiu and S. Ahmad. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form $ax = xb$. *IEEE Transactions on Robotics and Automation*, 5(1):16–29, 1989.
- [21] R. Y. Tsai and R. K. Lenz. A new technique for fully autonomous and efficient 3d robotics hand/eye calibration. *IEEE Transactions on robotics and automation*, 5(3):345–358, 1989.
- [22] R. I. Hartley and P. Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.
- [23] O. Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.
- [24] K. Schreve. How accurate can a stereovision measurement be? In *Research and Education in Mechatronics (REM), 2014 15th International Workshop on*, pages 1–7. IEEE, 2014.

Appendix A

User Interface

A.1 Starting Window

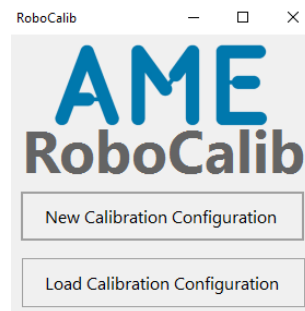


Figure A.1: RoboCalib's first window.

A.2 Load Calibration Configuration

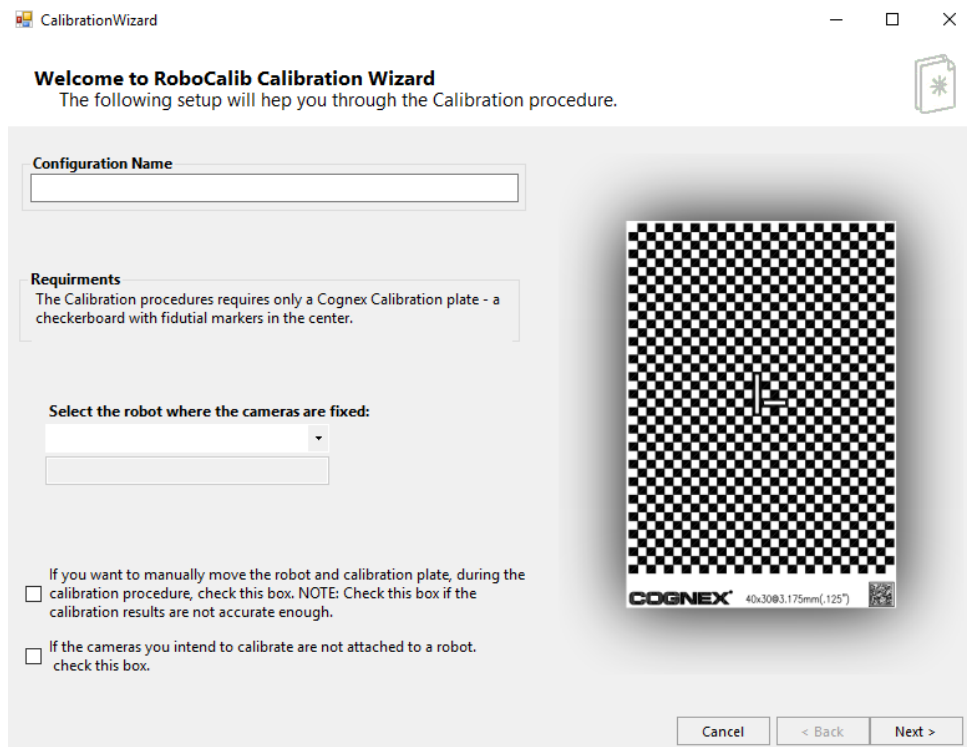
Load Calibration Configuration									
	Configuration Name	Robot	Focus Distance	Camera 1	Zoom Camera 1	Focus Camera 1	Iris Camera 1	Exposure Camera 1	Camera 1
	wer	ABB IRB 4600 - ...	260	GigE Vision: The...	60	365	200	35	N/A
	0	ABB IRB 4600 - ...	170	GigE Vision: The...	0	300	200	35	N/A
	10	ABB IRB 4600 - ...	170	GigE Vision: The...	10	330	200	30	N/A
	20	ABB IRB 4600 - ...	170	GigE Vision: The...	20	300	200	30	N/A
	30	ABB IRB 4600 - ...	170	GigE Vision: The...	30	260	200	30	N/A
	40	ABB IRB 4600 - ...	170	GigE Vision: The...	40	260	200	30	N/A
	50	ABB IRB 4600 - ...	280	GigE Vision: The...	50	380	200	30	N/A
	60	ABB IRB 4600 - ...	280	GigE Vision: The...	60	380	200	35	N/A
	70	ABB IRB 7600 - T...	280	GigE Vision: The...	70	380	200	35	N/A
	80	ABB IRB 4600 - ...	565	GigE Vision: The...	80	535	200	60	N/A
	rotation	ABB IRB 4600 - ...	200	GigE Vision: The...	20	325	200	20	N/A
	rot300	ABB IRB 4600 - ...	270	GigE Vision: The...	40	380	200	20	N/A
	rot400	ABB IRB 4600 - ...	400	GigE Vision: The...	60	470	200	35	N/A
	rot400_2	ABB IRB 4600 - ...	405	GigE Vision: The...	50	475	300	35	N/A
	rot250	ABB IRB 4600 - ...	250	GigE Vision: The...	30	360	200	20	N/A
	rot300_2	ABB IRB 4600 - ...	300	GigE Vision: The...	40	420	200	30	N/A
	rot 350	ABB IRB 4600 - ...	250	GigE Vision: The...	47	450	200	20	N/A

Camera GigE Vision: Basler: acA1600-20gm not found!

Load Configurations

Figure A.2: RoboCalib's load calibration configuration window.

A.3 New Calibration Configuration



CalibrationWizard

Welcome to RoboCalib Calibration Wizard
The following setup will help you through the Calibration procedure.

Configuration Name
[Text Field]

Requirements
The Calibration procedures requires only a Cognex Calibration plate - a checkerboard with fiducial markers in the center.

Select the robot where the cameras are fixed:
[Dropdown Menu]

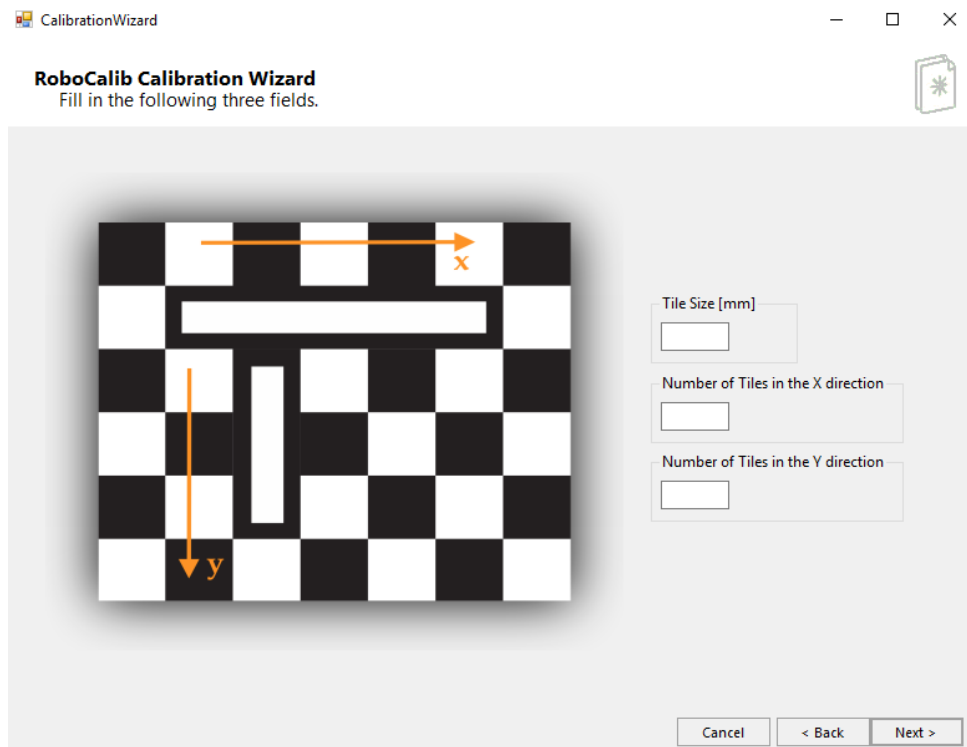
☐ If you want to manually move the robot and calibration plate, during the calibration procedure, check this box. NOTE: Check this box if the calibration results are not accurate enough.

☐ If the cameras you intend to calibrate are not attached to a robot, check this box.

COGNEX 40x30(83.175mm(.125'))

[Cancel] [< Back] [Next >]

Figure A.3: RoboCalib's new calibration configuration wizard.



CalibrationWizard

RoboCalib Calibration Wizard
Fill in the following three fields.

Tile Size [mm]
[Text Field]

Number of Tiles in the X direction
[Text Field]

Number of Tiles in the Y direction
[Text Field]

Cancel [< Back] [Next >]

Figure A.4: RoboCalib's new calibration configuration wizard.

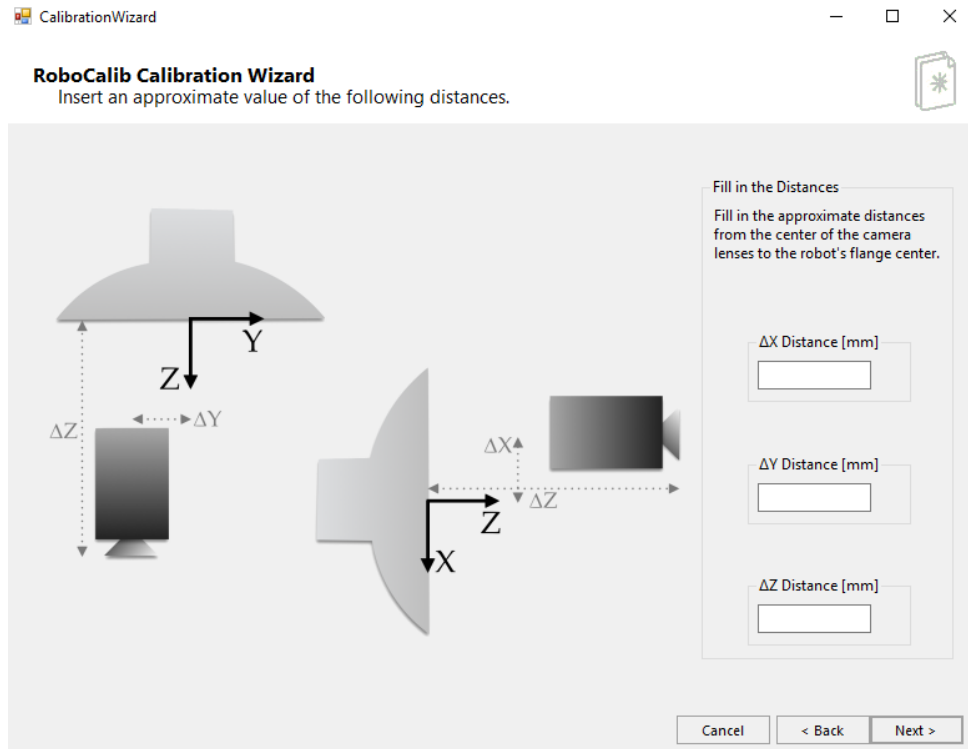


Figure A.5: RoboCalib's new calibration configuration wizard.

A.4 Main Form

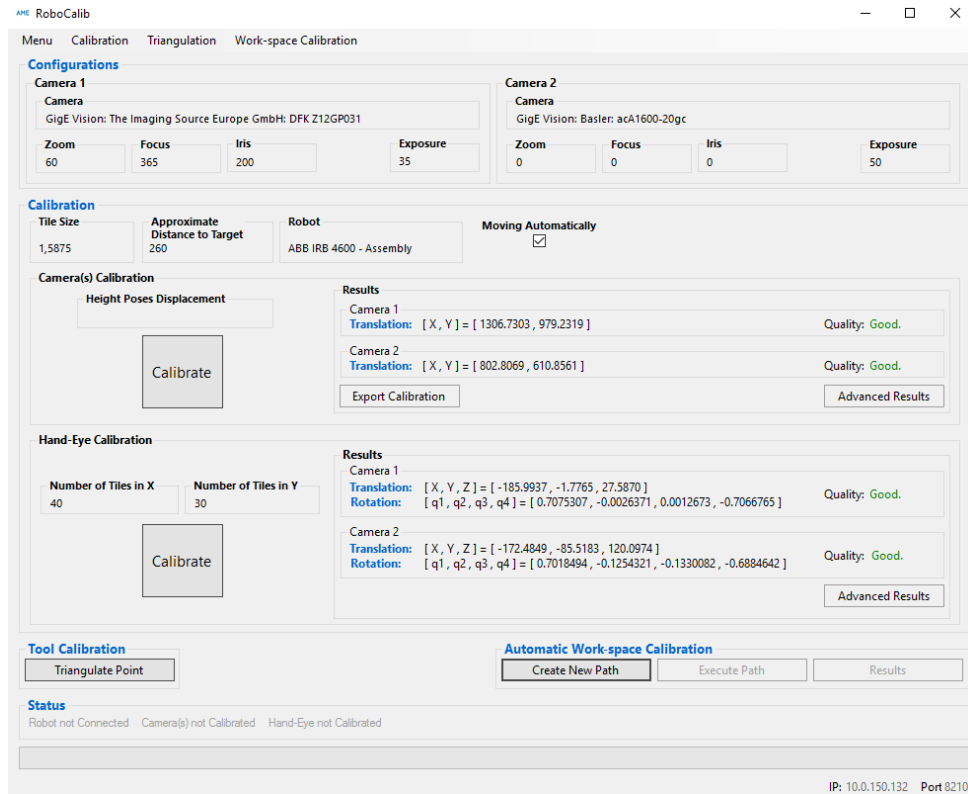


Figure A.6: RoboCalib's main form.

A.5 Manual Camera Calibration

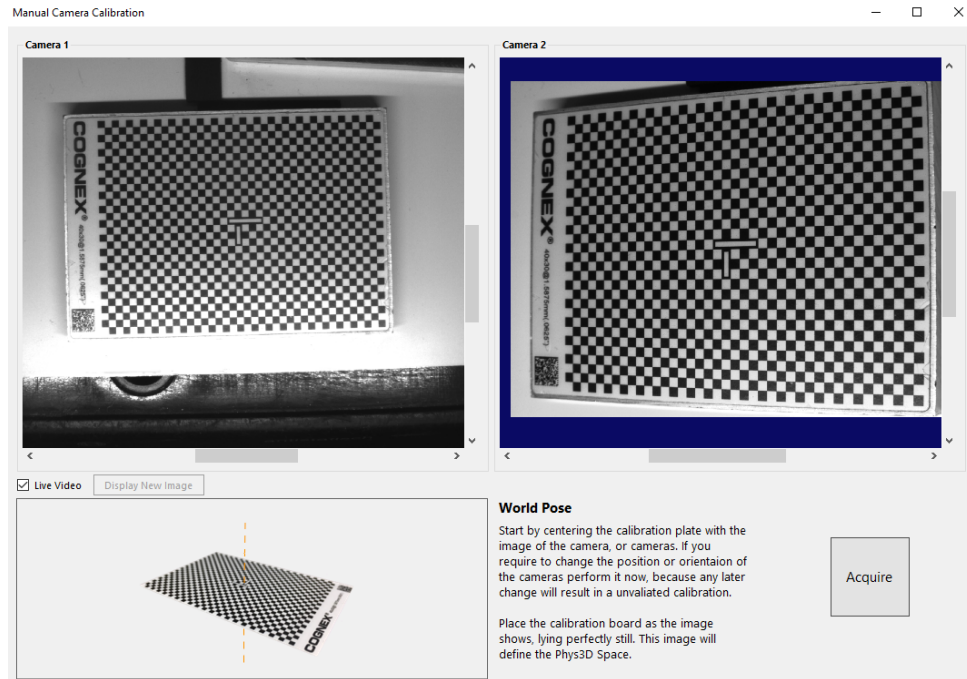


Figure A.7: RoboCalib's manual camera calibration.

A.6 Tool Point Triangulation



Figure A.8: RoboCalib's tool point triangulation.