Faculdade de Engenharia da Universidade do Porto

Automated Eye-in-Hand Calibration Process for FANUC Robots using a Cognex Vision System

Nuno Xavier Fernandes Alves

Dissertação realizada no âmbito do
Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Paulo Portugal

30/06/2025

Resumo

A integração de sistemas de visão computacional com robôs industriais é fundamental para alcançar alta precisão e eficiência nos processos de automação modernos. Um dos principais desafios nesta integração é a calibração precisa dos sistemas de coordenadas entre o sistema de visão e o robô, referida como calibração mão-olho. Os métodos tradicionais de calibração manual são morosos, propensos a erros humanos e ineficazes em ambientes industriais dinâmicos. Esta dissertação aborda esses desafios através do desenvolvimento de um sistema automatizado de calibração mão-olho para sincronizar um sistema de visão Cognex com um robô FANUC, implementado numa célula robótica industrial na JPM Industry.

A solução proposta resulta num sistema de calibração totalmente automatizado, capaz de determinar com precisão a transformação espacial entre o atuador final do robô e o sistema de visão acoplado. Ao eliminar a necessidade de medições manuais ou de suposições mecânicas pré-definidas, o sistema melhora a precisão, a repetibilidade e a adaptabilidade em tarefas de calibração industrial. A implementação final permite a recalibração dinâmica, aumenta a robustez do sistema em condições reais de operação e facilita a sua integração em fluxos de trabalho automatizados.

Este estudo estabelece a base para um algoritmo concebido para reduzir o tempo de calibração, minimizar erros, melhorar a disponibilidade dos sistemas robóticos e de visão em ambientes industriais e diminuir a necessidade de intervenção humana.

Palavras-chave: Robô, Câmara, Calibração, Calibração mão-olho, Automação industrial.

Abstract

The integration of computer vision systems with industrial robots is critical for achieving high precision and efficiency in modern automation processes. A key challenge in this integration is the accurate calibration of the coordinate systems between the vision system and the robot, commonly referred to as hand-eye calibration. Traditional manual calibration methods are time-consuming, prone to human error, and inefficient in dynamic industrial environments. This dissertation addresses these challenges by developing an automated hand-eye calibration system for synchronizing a Cognex vision system with a FANUC robot, implemented in an industrial robotic cell at JPM Industry.

The proposed solution delivers a fully automated calibration system capable of accurately determining the spatial transformation between the robot's end-effector and the attached vision system. By eliminating the need for manual measurements or predefined mechanical assumptions, the system enhances precision, repeatability, and adaptability in industrial calibration tasks. The final implementation enables dynamic recalibration, improves system robustness in real-world operating conditions, and supports seamless integration into automated workflows.

This study establishes the foundation for an algorithm designed to reduce calibration time, minimize errors, improve the availability of robotic and vision systems in industrial environments, and decrease the need for human intervention.

Keywords: Robot, Camera, Calibration, Hand-eye calibration, Industrial automation.

United Nations Sustainable Development Goals (SDGs)

1. **SDG 8: Decent Work and Economic Growth**
   Automation and the reduction of human intervention in technical calibrations improve productivity in industries, creating safer working environments and reducing the likelihood of errors and rework.

2. **SDG 9: Industry, Innovation, and Infrastructure**
   This work contributes to the development of innovative technologies and automated processes, promoting efficiency and accuracy in industrial automation. The synchronization between vision systems and robots enhances the robustness of industrial infrastructures and facilitates the adoption of smart solutions.

3. **SDG 12: Responsible Consumption and Production**
   By increasing efficiency and minimizing waste in industrial processes, this project promotes more sustainable production practices, contributing to the more responsible use of resources.

ODS	Meta	Contribution	Performance indicators and metrics
8	1	Reduce calibration time	Calibration time
	2	Reduce calibration errors	Maximum and average error
	3	Safe work for the workers	Reduction of human involvement in calibration
9	1	Tecnological inovation in the industrial sector	Number of technologies implemented
	2	Update of the infastructure	Increased automation
12	1	Waste reduction	Increased process efficiency

Table 1- SDG´s

Table 1 outlines the relationship between the proposed automated calibration system and the Sustainable Development Goals (SDGs) defined by the United Nations, by identifying specific contributions of the project, their alignment with SDG targets (referred to as "Meta"), and the corresponding performance indicators used to evaluate impact.

Under **SDG 8 – Decent Work and Economic Growth**, the project contributes in three key ways:

- **Meta 1:** Reducing calibration time directly enhances productivity by accelerating the deployment and reconfiguration of robotic systems. The associated performance indicator is the actual *calibration time*, which can be objectively measured and compared against manual procedures.

- **Meta 2:** Minimizing calibration errors improves process reliability and reduces rework or production downtime. This is assessed through the *maximum and average error* obtained during calibration tests.
- **Meta 3:** By automating the calibration process, the system reduces the need for manual intervention in technical tasks, thereby contributing to a safer working environment. The relevant performance metric here is the *reduction of human involvement in calibration*, which can be measured by the number of manual steps eliminated.

For **SDG 9 – Industry, Innovation and Infrastructure**, the system supports:

- **Meta 1:** Encouraging technological innovation through the development and application of advanced calibration methods that improve precision and adaptability in automated systems. The *number of technologies implemented* serves as a measurable indicator of innovation, reflecting the integration of novel processes and approaches.
- **Meta 2:** Modernizing industrial infrastructure by introducing an automated and scalable calibration solution, thereby improving the responsiveness and flexibility of manufacturing systems. This is measured by *increased automation* within the robotic cell.

Lastly, under **SDG 12 – Responsible Consumption and Production**:

- **Meta 1:** The system contributes to reducing material and energy waste by improving the efficiency and precision of robotic operations. As fewer calibration errors lead to less scrap and rework, the relevant performance indicator is *increased process efficiency*.

Together, these contributions demonstrate that the proposed calibration system not only addresses technical challenges but also supports broader sustainability and innovation objectives, aligning with global efforts to improve industrial productivity, safety, and resource efficiency.

Acknowledgements

I would like to express my deepest gratitude to my faculty supervisor, Professor Paulo Portugal, for his guidance and support throughout the development of this work. His expertise and availability were essential for the completion and assessment of this report.
I also extend my thanks to the Faculdade de Engenharia da Universidade do Porto for the knowledge and resources provided throughout my academic journey, which proved to be indispensable for the execution and study of this work.
Finally, a special thank you to JPM and all those involved in this project for their availability and support during the *Research Initiation* course and throughout the dissertation. I would like to express particular appreciation to Tiago Nunes and Ricardo Pimenta for the opportunity to carry out this work at JPM, as well as for their continued guidance and insight, which were fundamental in the practical application and development of this dissertation.

Contents

List of figures

List of tables

List of Acronyms

ACK	Acknowledgment
DH	Denavit-Hartenberg
DI	Digital Input
DO	Digital Output
FB	Function Block
FEUP	Faculdade de Engenharia da Universidade do Porto
FK	Forward Kinematics
FPS	Frames per Second
GI	Group Input
GO	Group Output
HMI	Human Machine Interface
IDE	Integrated Development Environment
IK	Inverse Kinematics
LM	Levenberg-Marquardt
OPCUA	Open Platform Communications Unified Architecture
PC	Personal Computer
PLC	Programmable Logic Controller
ROI	Regions Of Interest
SVD	Singular Value Decomposition
TCP	Tool Center Point
TIA	Total Integrated Automation
TP	Teach Pendant

Chapter 1

Introduction

This chapter gives a brief introduction to the company JPM Industry, as well as an introduction to the proposal, objectives, and motivations behind it.

1.1 - Context

JPM is a leading provider of industrial automation solutions, specializing in the integration of technologies such as industrial robots and computer vision systems to optimize manufacturing processes. The company focuses on delivering turnkey automation systems that enhance precision, reduce manual intervention, and increase production efficiency. At the core of its operations is the continuous development of advanced, scalable solutions that improve the reliability and performance of robotic systems in dynamic industrial environments. This dissertation is directly aligned with JPM's strategic focus, aiming to address one of the company's critical challenges: automating the calibration process between robots and vision systems, also known as Hand-Eye Calibration. At JPM, the calibration process is carried out using a robotic cell composed of a robotic arm, a camera mounted on the robot, a calibration board, and a programmable logic controller (PLC), as shown in ***Figure 1-1***. This setup is specifically designed for calibration validation and testing purposes. In real client applications, similar robotic cells are implemented with additional components and functionalities tailored to the specific needs of each industrial environment. This type of setup is particularly important in scenarios where objects arrive at undetermined positions, requiring the vision system to detect their exact location so that the robotic arm can perform precise pick-and-place operations.

For this coordination to be possible, Hand-eye calibration plays a crucial role, as it allows the robot to accurately associate the position of the object detected by the camera with its own reference frame — enabling the conversion of visual data into actionable coordinates within the robot's coordinate system.

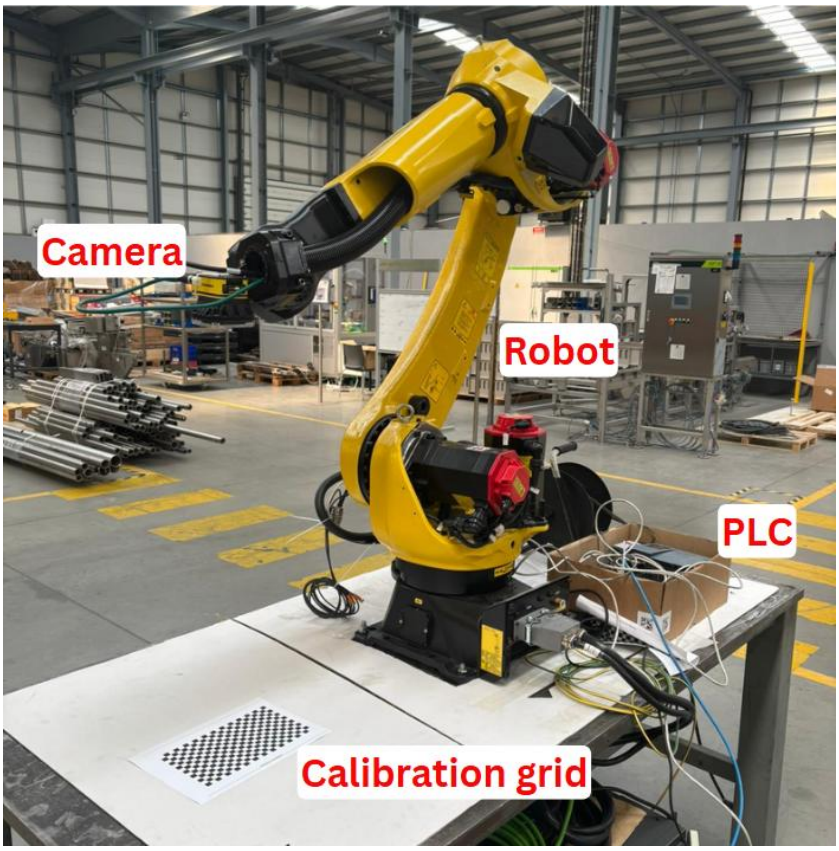

Figure 1-1 Robotic cell

These components of the robotic cell will collaboratively execute a Hand-Eye Calibration procedure. The primary objective of this calibration is to accurately determine the spatial relationship between the coordinate frames of the robot's end-effector and the camera. This relationship is mathematically described by two transformation vectors, which represent the translation and rotation needed to align the camera frame with the end-effector frame. Obtaining these transformations is crucial for precise positioning and alignment in tasks involving visual feedback, as it ensures that the robot can interpret visual data within the context of its own coordinate system.

1.2 - Motivation

In modern automated manufacturing, the integration of vision systems and robots is essential for enabling intelligent behaviors such as object recognition, defect inspection, and adaptive motion. However, the effectiveness of this integration depends heavily on the precise spatial relationship between the camera and the robot.

When the camera is mounted on the robot arm (*eye-in-hand*), this relationship becomes dynamic and more sensitive to changes in the environment and robot motion. Manual calibration in such setups is not only labor-intensive but also prone to frequent errors and misalignments due to mechanical shifts or configuration changes.

This limitation represents a significant bottleneck in achieving reliable and adaptive automation, especially in sectors served by JPM, where flexibility, accuracy, and reduced downtime are critical. By automating the calibration process for eye-in-hand systems, the proposed solution offers several advantages:

- **Faster deployment of robotic systems**, as calibration can be completed in minutes rather than hours.
- **Consistent and repeatable results**, eliminating variability introduced by human intervention.
- **Reduced operational costs**, as automated calibration decreases the need for expert labor and frequent maintenance.
- **Enhanced system resilience**, as the robot can periodically recalibrate itself in response to environmental changes or task reconfigurations.

1.3 - Problem Statement

Prior to the development and implementation of the automated Hand-Eye calibration system, the calibration procedure between the robot and vision system at JPM was performed manually. In the manual approach, the camera was mounted on a custom-designed mechanical support fixed to the robot's gripper. This support was fabricated to ensure a predetermined spatial relationship between the camera and the robot end-effector. These simplifications allowed the transformation between the robot coordinate system and the camera coordinate system to be defined using constant, pre-calculated values.
Calibration under this method involved only calculating the X and Y distances between the end effector frame and the camera frame. This was done by manually positioning the robot so that the camera was visually aligned with a known calibration grid or reference board. Operators would manipulate the robot until specific features in the camera's image aligned with designated points on the calibration pattern. This type of calibration was very dependent on the initial setup and forced manual movement of the robot.
This method was based on static assumptions and rigid mechanical configurations, which limited its flexibility, accuracy, and long-term reliability within an industrial setting.

1.4 - Objectives

The central goal of this dissertation is to design and implement a fully automated calibration system that aligns the coordinate frames of an industrial robot and a vision system—specifically, a FANUC robot and a Cognex vision system used at JPM Industry.

This work proposes an integrated solution that consists of a PC application, a PLC program, and a robot controller program that works synchronously for a solution capable of performing self-

calibration, significantly reducing the reliance on human input and enhancing the adaptability of the robot-vision system.

The calibration process will be embedded into a robotic cell, where the camera is mounted directly onto the robot. This setup enables dynamic calibration adjustments in response to real-world variables such as vibration, mechanical drift, thermal expansion, and changes in component positioning.

The system will use a target-based approach where the robot moves to a series of known positions, and the vision system captures corresponding image data. This information is then used to compute a transformation matrix that aligns the coordinate systems of the robot end-effector and the camera. The solution will integrate aspects of geometric computer vision (e.g., intrinsic and extrinsic calibration, pose estimation) and robot kinematics to ensure high precision in the calibration process.

This process will be implemented using appropriate programming interfaces on the FANUC controller and Cognex Insight-Explorer software to interface with the camera. Communication between the robot and vision system will be handled via industrial protocols such as PROFINET and OPC-UA to ensure reliable data exchange and command execution.

Ultimately, the goal is to develop a robust, reusable, and scalable solution that can be easily adapted to various robot-vision configurations, minimizing future integration efforts for similar systems.

Objectives:

1. **Automate Eye-in-Hand Calibration:**
   Develop a fully automated solution to compute the transformation between the robot's end-effector frame and the camera's coordinate system, specific to the eye-in-hand configuration.
2. **Reduce Calibration Time and Complexity:**
   Minimize the time required for calibration and eliminate the need for manual measurements or expert supervision by using an automated process integrated with the robot's motion system.
3. **Enhance Accuracy and Precision:**
   Improve the precision of spatial alignment between the robot and vision system, enabling accurate 3D positioning for tasks like object detection, inspection, and part handling.
4. **Ensure Operational Robustness:**
   Create a solution that maintains calibration accuracy even in challenging industrial conditions.
5. **Design for Reusability and Scalability:**
   Build the calibration algorithm as a reusable and adaptable module that can be applied to other robots and camera systems, making it scalable across multiple industrial configurations

1.5 - Structure of the document

Chapter 1: Introduction
This chapter introduces the context, motivation, and objectives of the proposed work. It outlines the challenges in current calibration techniques. The chapter also discusses the significance of the work and provides an overview of the document structure.

Chapter 2 – Background
This chapter provides a comprehensive overview of the robotic cell. It details its hardware and software components, the communication architecture between devices, and the calibration procedures employed prior to the proposed improvements. This contextual background serves to highlight the limitations of the existing system and establish a baseline for the enhancements introduced later in the dissertation.

Chapter 3 – Theoretical Background and State of the Art
This chapter presents the theoretical foundations and current methodologies relevant to robotic vision systems. It introduces essential concepts such as coordinate transformations, forward and inverse kinematics, and camera calibration. Emphasis is placed on hand–eye calibration, including both its geometric formulation and practical implementations. A detailed review of state-of-the-art techniques, highlighting their strengths, limitations, and applicability to industrial environments.

Chapter 4 – Proposed Architecture
This chapter outlines the system architecture designed to support automated hand–eye calibration. It defines the functional and technical requirements and describes the communication framework established between the industrial robot, PLC, camera, and PC. Detailed descriptions of the software design for each element (robot, PLC, and PC) are provided, illustrating the control logic and process coordination necessary for the calibration routine.

Chapter 5 – Implementation
In this chapter, the implementation of the solution is detailed. This includes the development of a Python-based program responsible for capturing images, acquiring robot poses, and performing camera and hand–eye calibration using the OpenCV library. The chapter also covers the robot controller routines and PLC function blocks developed to support calibration, ensuring precise execution of movements and synchronization with the vision system.

Chapter 6 – Results and Validation
This chapter presents the experimental results and evaluates the performance of the

proposed calibration system. Key metrics such as reprojection error and transformation consistency are used to assess accuracy and robustness. The influence of various factors—such as lighting conditions, number of calibration poses, and background quality—is analyzed.

Chapter 7 – Conclusions
The final chapter summarizes the contributions and outcomes of the dissertation. It reflects on the strengths and limitations of the proposed system and suggests directions for future research and improvements, particularly in the context of extending scalability and adaptability for broader industrial use cases.

Chapter 2

Background

This chapter provides a comprehensive overview of the robotic cell, detailing its constituent hardware and software components, communication architecture, and the calibration procedures employed prior to the system's enhancement. By establishing a baseline understanding of the system's original configuration and its associated challenges, this chapter sets the stage for the improvements proposed in later sections.

2.1 - Overview of the Robotic Cell

Prior to the development undertaken in this dissertation, the robotic cell consisted of a FANUC industrial robot, a Cognex vision system, a calibration board, and a Siemens programmable logic controller (PLC). Communication between the robot, the vision system, and the PLC was established via the PROFINET over an Ethernet network, enabling synchronized operation and data exchange among the components. This robotic cell was specifically designed for Hand-Eye Calibration. In this setup, the robot does not perform a production task. Instead, the focus is solely on calibrating the spatial relationship between the camera and the robot. Once the calibration is completed, the system can be adapted for various applications that require coordinated use of vision and robotics. One typical example is a pick-and-place operation, where the robot uses the calibrated data to locate and manipulate objects detected by the camera.

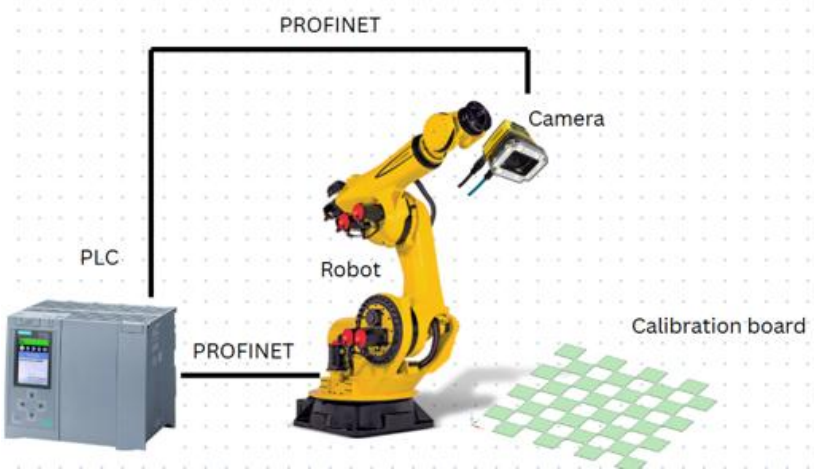

Figure 2-1 Robotic Cell

2.2 - Components

FANUC Robot

The robotic arm used in the cell is a FANUC Arc Mate 120id/35 [29], known for its high precision, reliability, and wide deployment across industrial automation settings. It features six degrees of freedom, which enables complex and flexible positioning in three-dimensional space. The robot is responsible for moving the camera to various known poses during the calibration process.

Figure 2-2 Fanuc Arc Mate 120id/35 [29]

The motion and operation of the robot are governed by a robot controller, which is a computational unit responsible for executing motion planning, trajectory control, and task coordination. The controller processes commands, handles sensor inputs, and precisely actuates the robot's joints to achieve the desired end-effector positions and orientations.
For this robot, the control system is the FANUC R-30iB Plus controller [32], which is designed for high-performance industrial applications. It provides real-time motion control, high-speed

processing, and a user-friendly interface for configuration and diagnostics. The controller also supports advanced functionalities such as collision detection, coordinated motion with external axes, and a variety of safety and communication protocols, including Ethernet/IP, PROFINET, and Modbus TCP. These capabilities make it suitable for complex tasks in automated manufacturing environments, including both high-speed operations and precision applications such as sensor calibration.

Figure 2-3 FANUC R-30iB Plus controller [32]

Fanuc Teach Pendant

The FANUC Teach Pendant [33] is an interface device used in industrial robotics for programming, operating, and troubleshooting FANUC robotics systems. It serves as the primary human-machine interface (HMI), allowing operators to interact with the robot, set up motion sequences, and diagnose issues efficiently.

Figure 2-4 Fanuc Teach Pendant

The main functionalities of the pendant are:

1. **Jogging and Motion Control**
    - Operators can manually jog the robot in various coordinate modes (joint, world, tool, and user frame) to position it accurately.

- The speed and acceleration settings can be adjusted for precise movement control.

2. **Program Creation and Editing**
   - Programs can be created using TP programming. TP programming (*Teach Pendant programming*) is the native language used to program FANUC robots via their teach pendant. It's relatively simple and menu-driven, designed for easy creation of motion and logic routines.
   - Users can edit existing programs, insert motion commands, conditional logic, and I/O interactions.
3. **Error Diagnosis and Troubleshooting**
   - The teach pendant provides real-time error messages, fault codes, and suggested corrective actions.
   - Log files and system diagnostics can be accessed for maintenance and debugging.
4. **Integration with External Systems**
   - It supports communication with PLCs (*Programmable Logic Controllers*), vision systems, and other automation components via Ethernet/IP, Modbus TCP, and PROFINET.
5. **Simulation and Testing**
   - Programs can be tested in teach mode before execution in production mode to minimize errors and optimize efficiency.

Cognex Vision System

The Cognex In-Sight 7600 camera [30] is a compact, high-performance industrial vision system designed for demanding automation tasks such as robotic guidance, inspection, and alignment. In this dissertation, the camera is mounted directly onto the robot's end-effector, enabling it to move with the robot and capture images of calibration targets from various perspectives. This placement ensures accurate calibration and real-time feedback during operation.
It features a 1/1.8-inch CMOS sensor with a resolution of up to 800 × 600 pixels and a pixel size of 4.5 µm, enabling precise imaging even during fast robotic movements. The camera supports frame rates up to 165 fps (monochrome) and 100 fps (color) at full resolution.

Figure 2-5 Cognex Insight 7600 [30]

Calibration Target

A fixed calibration target (checkerboard pattern) is placed within the robot's working volume. This target provides reference features used for extracting camera parameters and computing spatial transformations. The target must remain stationary during calibration to ensure accuracy. The calibration grid used in this dissertation has a grid spacing of 10mm.

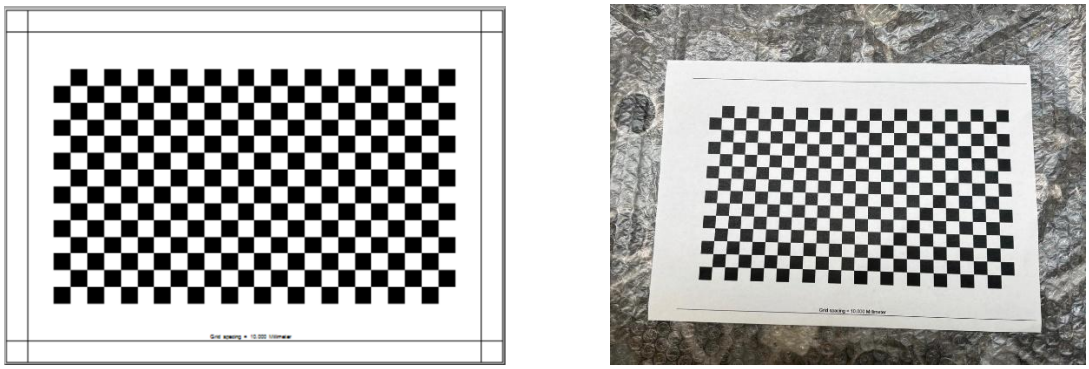

Figure 2-6 Cognex Calibration grid [25]

Siemens PLC

The Siemens S7-1517 PLC [31] is a high-performance programmable logic controller from the Siemens SIMATIC S7-1500 series, designed for demanding automation applications.

Figure 2-7 Siemens PLC S7-1517 [31]

In the robotic cell, the S7-1517 acts as the central control unit, orchestrating communication and coordination between the robot, the vision system, and the PC. Its powerful CPU and fast processing capabilities ensure reliable, real-time control of all system components.
The PLC manages the logical control of the robotic cell, executing predefined sequences of operations critical to the calibration process. This includes activating the robot’s movement routines, triggering the vision system for image acquisition, and synchronizing data collection.

The PLC's role is to ensure precise sequencing of each step, reducing the risk of misalignment or data loss during the calibration process.
The PLC's programmable logic is implemented using TIA Portal [34], Siemens' engineering framework, which provides a user-friendly environment for developing, testing, and deploying control logic. Within this environment, the calibration sequence is designed as a structured program, incorporating state-based logic to handle various calibration stages, including robot movements, image capture, and data transmission.

2.3 - Software

Tia Portal V18

The Totally Integrated Automation (TIA) Portal V18 [34] is an engineering framework developed by Siemens, designed to facilitate the programming, configuration, and commissioning of industrial automation systems. In this project, the TIA Portal V18 will serve as the primary environment for configuring and programming a Siemens PLC, which will act as the central controller for the entire automation process.
TIA Portal's integrated simulation tools will enable testing and validating PLC logic before deployment, reducing commissioning time and minimizing potential errors in the production environment.

Cognex Insight-Explorer

Cognex In-Sight Explorer [25] is a specialized software application designed for configuring and simulating the operations of Cognex vision systems. It plays a critical role in this dissertation by facilitating the setup and calibration of the camera, as well as performing advanced image processing tasks.
Running on a personal computer, this software will allow users to:

- **Camera Configuration:** Establish communication with the vision camera, set parameters like exposure, focus, and image resolution, and verify live image feeds.
- **Calibration and ROI Definition:** Define regions of interest (ROI) and perform camera calibration to map pixel coordinates to real-world measurements. This calibration is essential for accurate hand-eye coordination with the robot.
- **Image Acquire:** It enables saving the images on the computer to make the calibration outside the Insight-Explorer software.
- **Image Analysis:** Process captured images to detect features, measure object positions, and calculate displacements between the detected grid center and the desired image center.
- **Communication with PLC:** Transmit processed image data to the PLC, providing the necessary feedback for closed-loop control of the robot's movements.
  The ability to simulate vision tasks within In-Sight Explorer ensures that calibration and image processing algorithms can be fine-tuned in a virtual environment, streamlining the integration with physical hardware.

The In-Sight Explorer software is used to configure and develop programs executed by the vision system. In the robotic cell, prior to the development of the automated Hand-Eye Calibration procedure, this software was run on a personal computer solely for the purpose of configuring and programming the camera. It was not required to be active during the operation of the robotic cell, as the camera executed its tasks independently once configured.

Fanuc Roboguide

Roboguide [35] is an advanced simulation and programming tool developed by FANUC, designed for modeling, programming, and simulating robot operations. In this dissertation, Roboguide will be instrumental in validating robot behavior, allowing for a virtual test environment that closely mirrors real-world scenarios.
Key functionalities of Roboguide include:

- **Robot Calibration:** Simulating the calibration routine to align the robot's coordinate system with the camera's coordinate system, a crucial step in hand-eye calibration.
- **Trajectory Planning:** Programming and visualizing robot movements between reference points, with precise control over speed, acceleration, and tool orientation.
- **Process Flow Validation:** Testing the complete interaction between the robot, camera, and PLC to verify the logical flow of commands and data, identifying and resolving potential issues before physical deployment.
- **Collision Detection and Safety Checks:** Ensuring safe robot movements by detecting potential collisions in the simulated environment, helping to prevent damage to equipment and ensuring operator safety.

After testing and validating the program within the RoboGuide simulation environment, it can be exported and deployed to the physical robot for execution.
By leveraging Roboguide's powerful simulation capabilities, the entire robotic workflow can be refined and optimized without requiring immediate access to physical hardware.

2.4 - Communications

Although PROFINET was the primary communication protocol utilized in the system prior to the development of this dissertation, the architecture also supported alternative communication standards such as Modbus TCP and OPC UA. The selection of a specific protocol dependes on the application requirements, system integration constraints, and additional considerations such as software licensing, device compatibility, and performance needs.

PROFINET

PROFINET (Process Field Net) [36] is an open industrial Ethernet standard developed by PI (PROFIBUS & PROFINET International) for automation systems. It is designed to facilitate real-time communication between industrial controllers, field devices, and other automation

components over standard Ethernet networks. PROFINET supports both cyclic data exchange for real-time control and acyclic communication for diagnostics and configuration.
In this robotic cell setup, PROFINET is employed primarily for acyclic communication between the PLC, the FANUC robot, and the Cognex vision system. Acyclic communication is event-driven and used for the transmission of commands, status updates, and diagnostic data, rather than for continuous, time-critical control. This mode of operation is particularly suitable for tasks such as initiating image capture, receiving calibration status, or coordinating discrete events in the cell.
Moreover, because acyclic communication does not require tightly bounded cyclic update rates, it reduces network load and provides greater flexibility for the PLC to handle additional communication tasks. Specifically, it enables the PLC to interact with other system components using different industrial protocols, such as Modbus TCP, OPC UA, or EtherNet/IP, without compromising the timing or reliability of the primary automation tasks. This capability supports interoperability and system scalability, which are essential in modern, heterogeneous automation environments.

Modbus TCP

Modbus TCP [37] is an industrial communication protocol that operates over standard TCP/IP networks and is part of the broader Modbus family, originally developed by Modicon (now Schneider Electric). It is widely used in automation and control systems for communication between programmable logic controllers (PLCs), human-machine interfaces (HMIs), sensors, and actuators.
Modbus TCP implements the traditional Modbus protocol on Ethernet, encapsulating Modbus frames within TCP/IP packets. This allows for easy integration into modern Ethernet-based networks without requiring specialized hardware or infrastructure.
This protocol uses a client-server communication model, where the PLC typically acts as the client, initiating read and write requests to other devices, such as the camera and the robot, which function as servers by responding to these requests.

OPC-UA

OPC UA (Open Platform Communications Unified Architecture) [38] is a platform-independent, service-oriented communication protocol widely used in industrial automation for secure and standardized data exchange. It supports both client-server and publish-subscribe models, enabling interoperability between heterogeneous devices such as PLCs, robots, and vision systems. With features such as built-in security, scalability, and rich information modeling, OPC UA is well-suited for applications ranging from real-time control to integration with higher-level systems in Industry 4.0 environments.

2.5 - Calibration method

The synchronization model between the robot and the vision system is based on a structured calibration procedure with specific requirements. Both the robot and the vision system must support the same communication protocol, or alternatively, a gateway device must be employed to enable interoperability between them.
In this method, the camera was mounted on a custom-designed mechanical support fixed to the robot's gripper (**Figure 2-8**). This support was fabricated to ensure a predetermined spatial relationship between the camera and the robot end-effector. Specifically, the Z-offset—representing the linear distance from the robot's gripper to the camera along the Z-axis—was known. These simplifications allowed the transformation between the robot coordinate system and the camera coordinate system to be defined using constant, pre-calculated values. This type of simplification can be seen in (**Figure 2-8**), where the x, y, and z frames have a rotation of 0 degrees between the camera and the robot end-effector frames, and the Z distance is known due to a fixed value camera support. This type of calibration solely determines the translational offsets in the X and Y directions between the coordinate systems of the camera (xc, yc) and the robot (xr, yr).

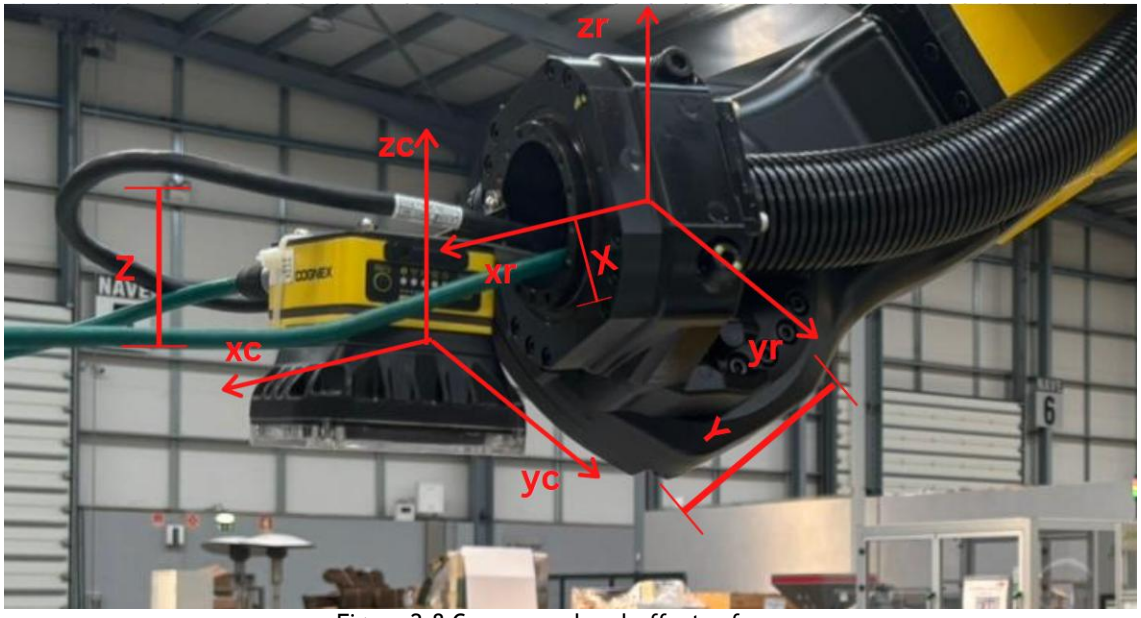

Figure 2-8 Camera and end-effector frames

To determine the X and Y offsets between the camera and the robot's end-effector, two calibration poses were used. In the first pose, the robot was manually positioned so that a predefined intersection of lines on the calibration grid aligned with the grid's origin. The coordinates of a secondary, predefined reference point on the grid were then recorded from the camera image.
In **Figure 2-9**, the line intersection on the left was perfectly aligned with the origin of the calibration grid, and the intersection on the right corresponds to the point (-93.5 ; 0)mm.

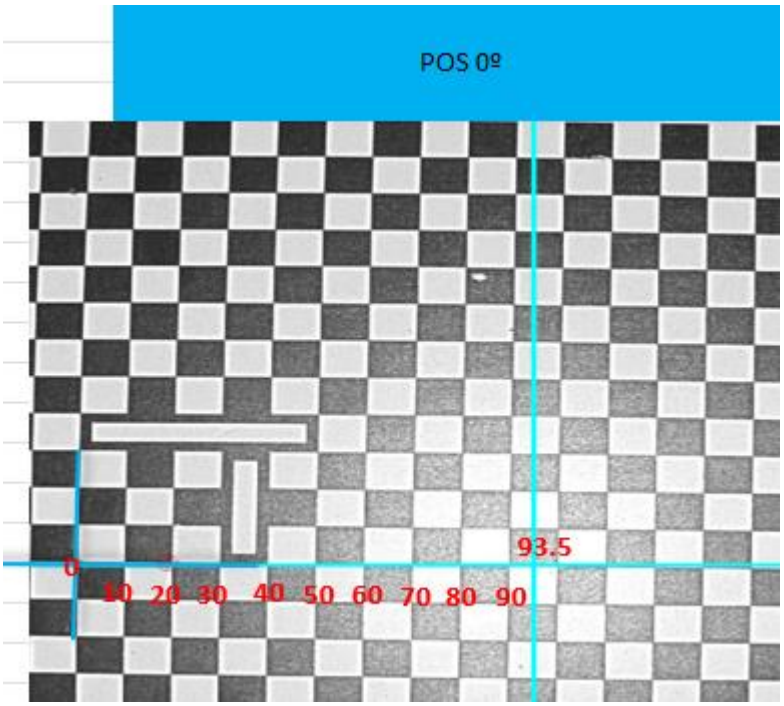

Figure 2-9 First position

Subsequently, the robot moved to the second pose, which involved rotating the camera 180 degrees around the same grid while maintaining the same height and position relative to the grid plane. The coordinates of the same reference point were again recorded.
In **Figure 2-10**, the intersection of the corresponding reference point after rotation is located at (−95 ; 3) mm.

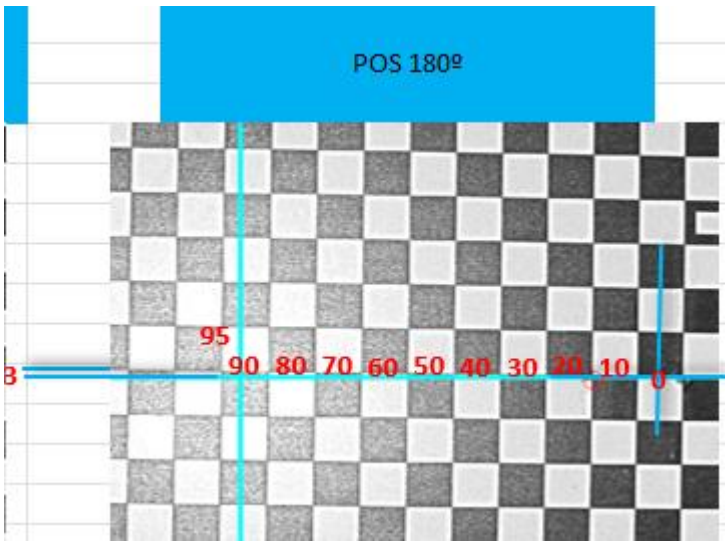

Figure 2-10 Second position

The relative displacement between the two observations of the reference point was then computed by subtracting the X and Y values obtained in the first pose from those in the second pose. This difference represents the interpolated shift caused by the camera's rotation. Finally, the X and Y offsets of the camera relative to the robot's end-effector were obtained by

summing this interpolated shift with the known X and Y coordinates of the robot's tool center point.
For the interpolation calculation, the difference between these points is computed as:

$$(x_i; y_i) = (x_1; y_1) + (x_2; y_2) \quad (2.1)$$

Where

- $(x_i; y_i)$ are the coordinates of the interpolation.
- $(x_1; y_1)$ are the coordinates of the intersection in the first pose.
- $(x_2; y_2)$ are the coordinates of the intersection of the second pose.

For the example:

$$(-93.5;\ 0) + (-95; 3) = (1.5; -3)mm$$

Thus, the position of the camera relative to the robot's end-effector is obtained by adding the vector (1.5;−3) mm to the end-effector's current position.
The Z offset of the camera was determined independently. It corresponds to the Z coordinate of the robot's end-effector plus the known Z offset introduced by the physical support structure on which the camera was mounted.

2.6 - Limitations

Although this calibration method can yield high accuracy, it presents several limitations. It requires a camera support with a known and fixed geometry, and the entire process is manual, demanding an experienced operator to execute the procedure correctly. Furthermore, the calculated offsets must be manually introduced into the robot controller or vision system, which increases the risk of human error.
Additionally, the method is inherently rigid and lacks adaptability. Any misalignment of the camera due to mechanical wear, impact, or maintenance interventions necessitates the fabrication of a new custom support and the repetition of the entire manual calibration process. This can result in increased downtime and reduced system flexibility in dynamic industrial environments.
These constraints highlight the need for a more robust, automated, and scalable calibration approach—one that reduces reliance on manual procedures and predefined mechanical configurations. Addressing these challenges is essential to improving the overall reliability, efficiency, and maintainability of the robotic vision system.
The calibration system implemented in this dissertation is fully automated and does not rely on predefined or fixed values related to the camera mount. It autonomously computes the transformation between the robot end-effector and the camera from scratch, allowing for flexible deployment. Additionally, the system supports recalibration as needed, ensuring adaptability and maintaining accuracy over time. This approach significantly reduces calibration time and minimizes the need for manual intervention, thereby improving efficiency and repeatability in robotic system setup.

2.7 - Conclusions

This chapter has presented a detailed overview of the robotic cell utilized for Hand-Eye Calibration, including its core hardware and software components, communication protocols, and the calibration methodology employed prior to the developments introduced in this dissertation. By examining the original system architecture—comprising a FANUC industrial robot, Cognex vision system, Siemens PLC, and associated simulation and programming tools—this chapter has established a clear understanding of the cell's foundational configuration.

The initial calibration approach, while capable of delivering accurate results, was highly dependent on manual operations and fixed mechanical assumptions, such as known offsets and rigid camera mounts. These constraints limited the system's flexibility, increased the risk of human error, and introduced inefficiencies during recalibration or system modifications.

To overcome these limitations, this dissertation proposes an enhanced calibration system that is fully automated, mount-agnostic, and capable of computing the transformation between the robot and camera dynamically. This new method enables recalibration without physical reconfiguration, significantly reducing calibration time and manual intervention. The improvements outlined here lay the groundwork for a more adaptable, scalable, and efficient vision-guided robotic platform—paving the way for versatile industrial applications, including high-precision pick-and-place operations.

Chapter 3

Theorical Background and State of the Art

Robotic systems that integrate computer vision rely heavily on precise geometric and mathematical models to interpret the world, localize objects, and interact with their environment. A comprehensive understanding of the spatial relationships between different components, such as cameras, robot joints, and end-effectors, is essential to enable tasks like visual servoing, object manipulation, and autonomous path planning. This chapter presents the theoretical and technical foundations necessary for developing and implementing such capabilities, with a particular focus on the calibration and coordination of vision and motion systems.

The discussion begins with the concept of coordinate systems and frames in robotic manipulators, emphasizing how positions and orientations are defined and transformed between reference points. This leads naturally into the definition of the Tool Center Point (TCP) and the mathematical formulations behind forward and inverse kinematics, which are central to determining and controlling the robot's pose.

Subsequently, the chapter introduces the fundamentals of camera calibration, including the modeling of lens distortion and the estimation of intrinsic and extrinsic parameters. These steps are essential to accurately relate the 2D image plane to the 3D world, enabling meaningful visual data to be used in control and decision-making processes.

A major focus is placed on hand–eye calibration, the procedure by which the spatial transformation between the robot end-effector and a mounted or external camera is estimated. Accurate hand–eye calibration is critical for tasks where visual perception and robot action must be precisely coordinated. The chapter covers both the geometric formulations of the problem and the methods used for pose acquisition, parameter estimation, and optimization.

Finally, the chapter concludes with a detailed review of the state of the art relevant hand–eye calibration methods and tools. This includes classical approaches such as the Tsai–Lenz algorithm, more recent formulations using dual quaternions, and software implementations available in both commercial platforms (e.g., MATLAB, Cognex VisionPro) and open-source

libraries (e.g., OpenCV). Each is analyzed in terms of accuracy, computational complexity, and practical applicability to industrial and research contexts.
Together, the theoretical principles and contemporary advancements presented in this chapter establish the foundational knowledge for the system developed in this dissertation and motivate the design decisions made throughout its implementation.

3.1 - Tool center point

The Tool Center Point (TCP) of a robotic arm refers to the specific point on the end-effector that is used for motion planning, control, and task execution. Understanding its position and orientation relative to the robot base is critical in industrial robotics, kinematics, and automation.

In this dissertation, it is essential to understand that the Tool Center Point (TCP) represents the position and orientation of any tool or component attached to the robot. When the TCP is not explicitly defined, it is typically represented by the default value (0, 0, 0). This default point corresponds to the geometric center of the robot's end-effector, as shown in **Figure 2-6**.

The objective of this dissertation is to determine the translation and rotation vectors between the end-effector's coordinate frame and the camera's coordinate frame.

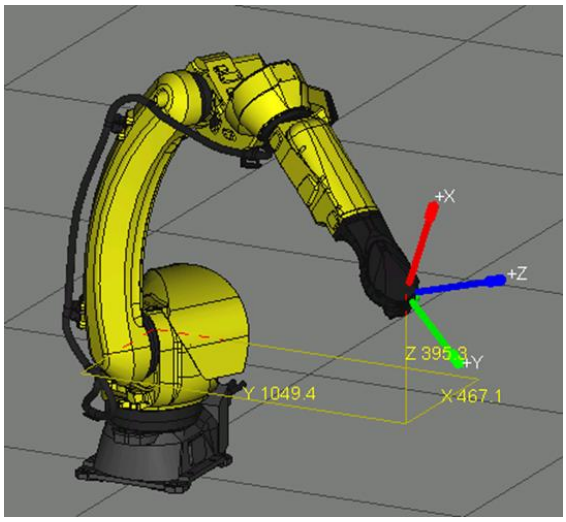

Figure 3-1 Robot end-effector frame. [24]

3.2 - Coordinate Systems

Several coordinate systems are involved in hand-eye calibration:

- **Robot Base Frame** – fixed reference frame for all robot movements.
- **Tool Center Point** – defined at the robot's end-effector.

- **Camera Frame** – located at the optical center of the mounted camera.
- **Target (World) Frame** – anchored to the calibration board.

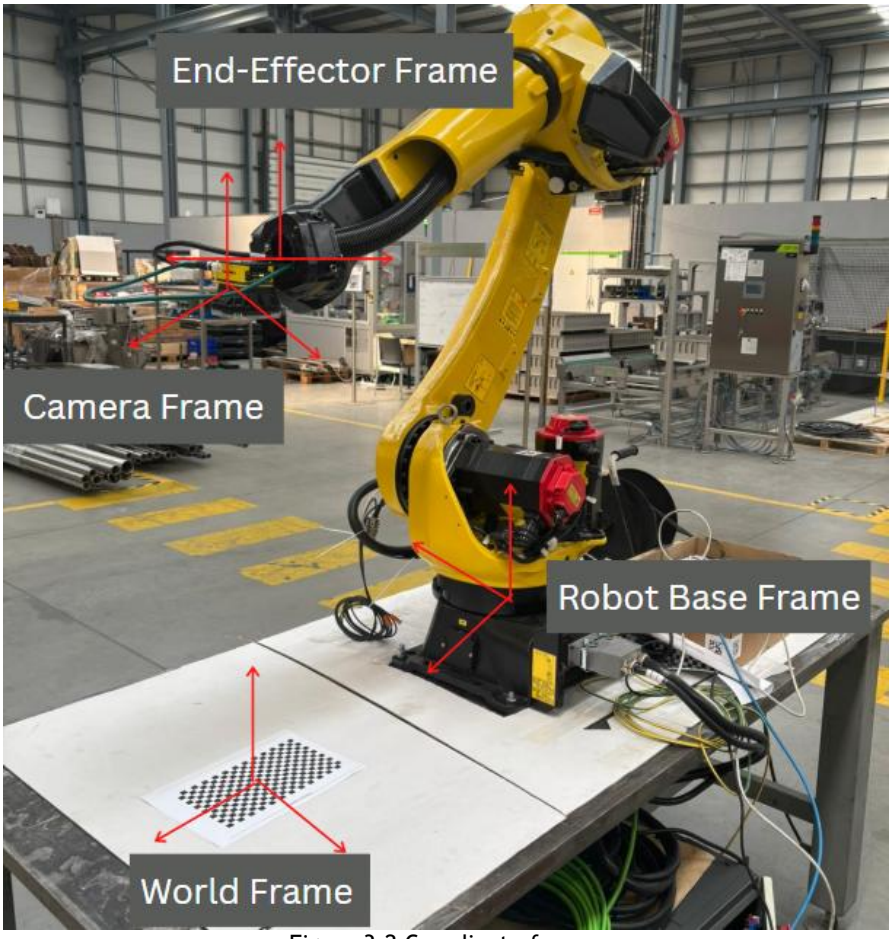

Figure 3-2 Coordinate frames

When discussing camera calibration, it refers to the determination of the transformation vectors between the world frame and the camera frame. In contrast, robot calibration focuses on calculating the transformation vectors between the robot base frame and the end-effector frame. Lastly, hand-eye calibration addresses the transformation vectors between the end-effector frame and the camera frame.

3.3 - Coordinate Frames in a Robotic Arm

In robotic systems, particularly articulated manipulators, the precise control and understanding of motion rely heavily on the proper definition and manipulation of the robot frames. These frames form the foundation for describing the robot's geometry, movement, and interaction with external systems such as cameras or tools.

1. Coordinate Frame Definitions
To systematically describe the spatial configuration of each component of a robotic arm, the following coordinate systems are defined:

- **Base Frame (F_0):**
  This is the global or world coordinate frame, fixed at the base of the robot. All other coordinate frames are ultimately referenced with respect to this frame. It serves as the origin for calculating the absolute position and orientation of the robot's components.

- **Joint Frames (F_1, F_2, ..., F_n):**
  Each joint of the robot has an associated coordinate frame, typically defined using the Denavit–Hartenberg (DH) convention [23] or similar kinematic modeling frameworks. These frames move as the robot's joints articulate and are essential for computing the relative transformations between links.

- **End-Effector Frame (F_EE):**
  This frame is rigidly attached to the robot's end-effector, representing its spatial configuration. It defines where the robot tool is located, excluding any tool-specific offsets.

- **Tool Center Point (TCP) Frame (F_TCP):**
  The TCP frame is defined relative to the end-effector, accounting for the actual tool being used (e.g., gripper, welding torch, camera). It represents the precise point of interaction between the robot and its environment. Proper calibration of the TCP is crucial for accuracy in task execution. This frame is not presented in **Figure 3-3** since the TCP can be defined for every moving part attached to the robot. By default, the F_TCP is the same as the F_EE.

Figure 3-3 Robot Frames

3.4 - Forward and Inverse Kinematics in Robotic Systems

In robotic systems, kinematics refers to the mathematical relationships between the joint parameters of a robot and the position and orientation of its end-effector. Two fundamental types of kinematic computations are forward kinematics (FK) and inverse kinematics (IK).

Forward Kinematics (FK)

Forward kinematics involves determining the pose (position and orientation) of the robot's end-effector based on known joint parameters. This is essential for understanding where the robot is in space when the angles or displacements of its joints are given. Mathematically, forward kinematics is expressed as:

$$T_{end\ effector} = f(\theta_1, \theta_2, \ldots, \theta_n) \quad (3.1)$$

where θ_i represents the individual joint variables and $T_{end\ effector}$ is the resulting transformation matrix representing the end-effector's pose in the robot base frame.

Inverse Kinematics (IK)

Inverse kinematics determines the necessary joint parameters to achieve a desired end-effector pose. This is a more complex process, as it may yield multiple solutions or no solution at all, depending on the robot's geometry and constraints:

$$(\theta_1, \theta_2, \ldots, \theta_n) = f^{-1}(T_{end\ effector}) \quad (3.2)$$

Computing the Forward Kinematics Function

The function $f(\theta_1, \theta_2, \ldots, \theta_n)$ calculates the pose of the robot's end-effector based on the joint values. This is typically done using the Denavit–Hartenberg [23] convention and homogeneous transformation matrices.

Robot Model and Parameters

The Denavit–Hartenberg [23] convention models each joint and link using four parameters:
- θ_i: Joint angle (variable for revolute joints)
- d_i: Offset along the previous z-axis
- a_i: Link length
- α_i: Twist angle between z-axes

DH Transformation Matrix

Each joint transformation is represented as a 4×4 homogeneous transformation matrix A_i, given by:

$$A_i = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

To determine the pose of the end-effector, the individual A_i matrices are multiplied in sequence:

$$T_0^n = A_1 \times A_2 \times \ldots \times A_n \quad (3.4)$$

This final matrix T_0^n represents the pose of the end-effector relative to the robot base frame. The resulting transformation matrix contains:
- A 3×3 rotation matrix in the top-left (describing orientation)
- A 3×1 translation vector in the top-right (describing position)

This complete representation provides all spatial information needed to control the robot's end-effector.

Importance in Hand-Eye Calibration

Precise computation of the transformation matrix from the base to the end-effector is crucial not only for robot control but also for achieving accurate hand-eye calibration. It ensures that the sensor's observations can be correctly interpreted in the robot's coordinate system, enabling tasks such as visual servoing, inspection, and autonomous manipulation with high precision.

3.5 - Theoretical Background on Camera Calibration

Camera calibration is a fundamental process in computer vision, enabling the determination of a camera's internal and external parameters. This process establishes the mathematical relationship between the three-dimensional (3D) world and its two-dimensional (2D) image projections.

Intrinsic Parameters

Intrinsic parameters define the internal characteristics of a camera — essentially, how the camera transforms 3D points in the real world into 2D points in an image based on its own geometry and optical properties.
These parameters are independent of the camera's position or orientation in space (those are extrinsic parameters). Instead, they describe how the camera projects the world onto its vision sensor.
Intrinsic parameters are defined in matrix **K**:

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

where:

- **fx** and **fy** represent the focal length of the camera in pixels along the x-axis and y-axis, respectively. These values determine how much a point in 3D space is scaled when projected onto the image plane. The focal length in pixels is derived from the physical focal length of the camera lens and the pixel density of the image sensor.
- **cx** and **cy** denote the coordinates of the principal point, which is the intersection of the camera's optical axis with the image plane. Ideally, this point lies at the center of the image sensor, but due to manufacturing imperfections, it may be slightly offset. These coordinates are critical for accurately mapping the captured image data to the camera's coordinate system.

Extrinsic Parameters

Extrinsic parameters define the pose of the camera in the world — in other words, where the camera is and how it's oriented relative to a known coordinate system (like a robot's base frame or a calibration pattern in space).
They describe the rigid transformation from the world coordinate system to the camera coordinate system.
The extrinsic parameters are described in the following homogeneous transformation matrix:

$$T_{world}^{camera} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \quad (3.6)$$

More explicitly, this can be written as a 4×4 matrix:

$$T_{world}^{camera} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

- **Rotation matrix (R_{ij})**: encodes the orientation of the camera relative to the world coordinate system. It defines how the camera is rotated in three-dimensional space, aligning the camera's local coordinate axes with those of the world frame. Specifically, each column corresponds to one of the camera's local axes expressed in world

coordinates: the first column represents the camera's x-axis, the second the y-axis, and the third the z-axis.

- **Translation vector (t_i)**: defines the position of the camera's optical center within the world coordinate frame. The components of **t**, namely t_1, t_2, and t_3, specify the displacement along the world's x, y, and z axes, respectively. The units of **t** correspond directly to those used to define the world coordinate system — for example, millimeters — providing a physical measurement of the camera's location.

Together, these parameters enable the transformation of points from the world to the camera coordinate system:

$$Pc = R * Pw + t \quad (3.8)$$

Where **Pw** is a point in the world frame and **Pc** is a point in the camera frame, both 3D points.

Camera Model

The most used model for camera calibration is the pinhole camera model [2]. This model is a mathematical framework used to describe the projection of 3D points onto a 2D image plane. It assumes an idealized camera with no lens distortions, where light rays travel in straight lines through a single small aperture (the pinhole) and project onto an image plane as we can see in **Figure 3-4**:

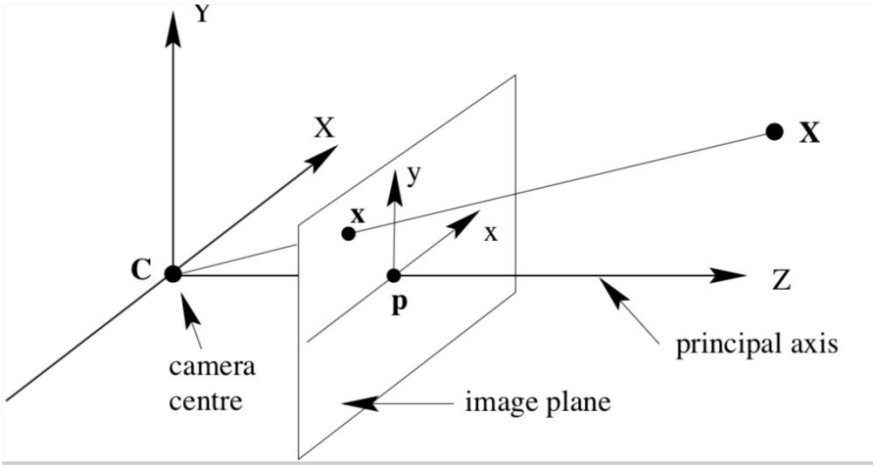

Figure 3-4 Pinhole camera model illustration. [2]

A point in the world, represented as:

$$Pw = [x\ y\ z]^T \quad (3.9)$$

Where $[x\ y\ z]$ are the coordinates in mm of the point in the world that is projected onto the image plane at:

$$Pi = [px\ py]^T \quad (3.10)$$

Where $[px\ py]$ are the coordinates in pixels of the point in the image.

Having the world points and the image points, the intrinsic and the extrinsic parameters of the camera can be estimated using the following equation:

$$\begin{bmatrix} px \\ py \\ 1 \end{bmatrix} = K[R\ T] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.11)$$

where:

- **K** is the intrinsic camera matrix.
- **[R T]** represents the extrinsic parameters.

Estimating the intrinsic and extrinsic camera parameters requires employing an iterative nonlinear optimization process, as the comprehensive camera projection model results in a system of nonlinear equations that cannot be solved analytically.

This estimation process will be discussed in detail later in this chapter.

The pinhole camera model does not account for lens distortion, which is inherently present in real-world cameras. Therefore, during camera calibration, distorted image points must be transformed into undistorted coordinates in order to conform to the assumptions of the pinhole projection model.

3.6 - Lens distortion

Lens distortion is a critical concept in computer vision and optical systems, especially in applications like hand-eye calibration, where accurate spatial alignment between a camera and a robotic system is essential. Distortion arises from the inherent imperfections in lens design and manufacturing, causing the captured image to deviate from the true geometry of the scene.

Types of Lens Distortion

1. **Radial Distortion:** This is the most common form of distortion, where light rays bend as they move away from the optical axis. It manifests as:
   Barrel Distortion: Lines appear to bulge outward, resembling the shape of a barrel. It occurs when the magnification decreases with distance from the optical center.
   Pincushion Distortion: Lines bow inward, forming a pinched effect. This happens when magnification increases with distance from the center.

Mustache Distortion: A combination of barrel and pincushion distortions, creating a complex, wavy deformation.

2. **Tangential (Decentering) Distortion:** Caused by lens elements not being perfectly aligned, resulting in an asymmetrical displacement of image points.
3. **Thin Prism Distortion:** A less common form, where the lens system behaves like a weak prism, causing a slight shift of points in a particular direction.

The most common types of lens distortion are the barrel distortion and the pincushion Distortion that are observed in the **Figure 3-5** :

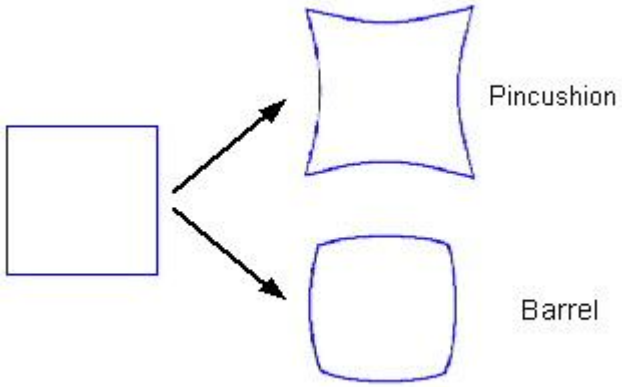

Figure 3-5 Barrel and Pincushion distortion compared to the undistorted lens[26]

Mathematical Modeling

To align distorted image points with the pinhole model, we must apply the inverse distortion model [1]. Given distorted pixel coordinates (px_d, py_d), the goal is to find the corrected normalized coordinates (px, py).

$$px_{\text{dist}} = px \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (3.12)$$

$$py_{\text{dist}} = py \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (3.13)$$

Where:

- $r^2 = px^2 + py^2$
- k_1, k_2, k_3 are the distortion coefficients used to mathematically describe the lens-induced deviations from the ideal pinhole camera model.

By replacing the pixel coordinates (px, py) into the distortion equations to obtain the distorted coordinates (px_d, py_d), the extended camera model can accurately estimate not only the intrinsic and extrinsic parameters of the camera, but also the distortion coefficients that describe the lens-induced deviations from the ideal pinhole projection.

Theory behind the camera calibration process

The calibration process follows a structured approach that builds a set of correspondences between known physical points and their observed positions in the camera image. The process begins by selecting a calibration target with a known and easily detectable geometric structure. A commonly used target is a checkerboard pattern like Figure 3-5, consisting of alternating black and white squares arranged in a grid. The corners where the squares meet are used as reference points due to their high contrast and sharpness, which allows for precise detection in images.
To obtain robust calibration results, the checkerboard is placed at various positions and angles relative to the camera. This ensures that the captured images provide a wide range of perspectives and depths, allowing for accurate parameter estimation.
A fixed 3D model of the checkerboard grid is defined, where each corner has a known position in a flat plane (commonly with Z = 0). These world coordinates are paired with the 2D pixel coordinates detected in the images, creating a dataset of 3D-to-2D correspondences for each calibration image.

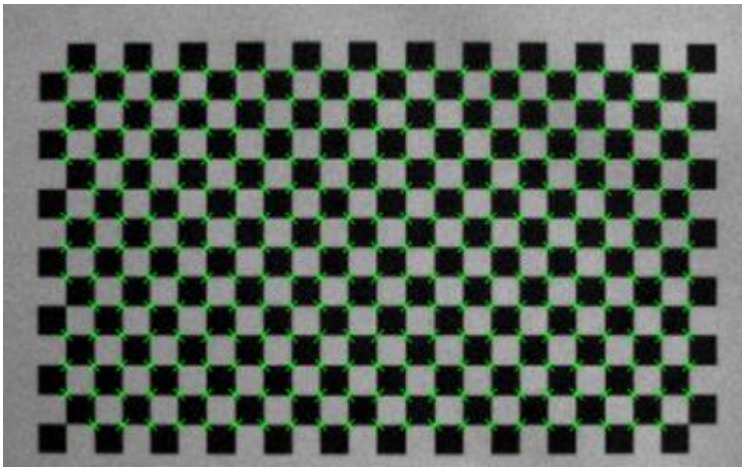

Figure 3-6 Calibration grid with marked points

Assuming that the spacing between adjacent corners is 10 mm, then each grid intersection corresponds to a known 3D world coordinate with a fixed metric distance. The top-left corner of the grid is often defined as the origin of the world coordinate system, i.e., (0,0,0) mm. Each subsequent point on the grid can then be assigned coordinates in millimeters based on its relative position (e.g., 10,0,0), (20,0,0), etc.). These known 3D points are then matched to their corresponding 2D image projections (px, py) in pixel units as detected in the image. This set of correspondences between world coordinates and image coordinates forms the input data for estimating the camera's intrinsic, extrinsic, and distortion parameters.

3.7 - Parameter Estimation

Using the correspondence between object points and image points, the calibration process estimates intrinsic and extrinsic parameters by minimizing the reprojection error (E):

$$E = \sum_{i=1}^{N} \left\| Pi - \widetilde{Pi} \right\|^2 \quad (3.14)$$

where:

- **Pi** are the observed image points.
- $\widetilde{\boldsymbol{Pi}}$ are the reprojected points computed from the estimated camera model.

The reprojected point is the predicted 2D image coordinate of a known 3D point, calculated using the current estimates of the camera parameters (intrinsic, extrinsic, and distortion).

This nonlinear least squares problem is solved using the Leveneberg–Marquardt algorithm presented in the next section.

Optimization

The calibration process requires iterative optimization techniques to minimize reprojection errors and refine parameter estimates.

Levenberg-Marquardt Algorithm:
The Levenberg-Marquardt algorithm [39] is a nonlinear optimization technique used to iteratively refine intrinsic and extrinsic parameters by minimizing the sum of squared differences between observed and projected image points.
LM Combines Two Methods:

1. **Gradient Descent**
    - Moves parameters in the direction of steepest decrease of the error function.
    - Very stable but can be slow near the minimum.
2. **Gauss-Newton Method**
    - Uses a second-order approximation assuming the error function behaves like a quadratic near the minimum.
    - Faster convergence close to the solution but can be unstable far from it.

LM's adaptive approach ensures both stability and fast convergence, making it ideal for refining parameters by minimizing reprojection error.

3.8 - Hand-eye Calibration

Hand-eye calibration enables the robot to accurately interpret visual data, translating image coordinates into real-world positions. Hand-eye calibration is a term that is equivalent to the synchronization of a coordinate system between a camera and a robot. The result of this calibration is a translation and rotation vector between the end-effector of the robot frame and the camera frame.

$$HandEye\ translation = [x, y, z] \quad (3.15)$$

$$HandEye\ rotation = [w, p, r] \quad (3.16)$$

Where x, y, z are the distances in mm between the end-effector of the robot and the camera in the respective frames, and w, p, r are the rotations in degrees of the frames of the camera in relation to the end-effector frames.

AX=XB Formulation

For a moving robot with a camera attached to its end-effector:

$$A_i X = X B_i \quad (3.17)$$

In this formulation:

- $\boldsymbol{A_i}$ is the known homogeneous transformation matrix that describes the pose (rotation and translation) of the robot end-effector relative to the robot base frame. Each A_i corresponds to one of several robot poses captured during calibration. The number of such matrices depends on the total number of poses used.
- $\boldsymbol{B_i}$ is the corresponding known transformation matrix representing the pose of the calibration target (or camera) relative to the world coordinate system (typically the calibration object's frame). These are derived from the camera's extrinsic parameters, also one per calibration pose. The number of B_i matrices must match that of the A_i matrices to maintain pose correspondence.
- $\boldsymbol{X}$ is the unknown constant transformation matrix that defines the rigid-body relationship between the camera frame and the robot end-effector frame. It encodes
- both the rotation and translation necessary to express coordinates from the end-effector frame into the camera frame.

This equation system is solved to estimate X, often using least-squares optimization.

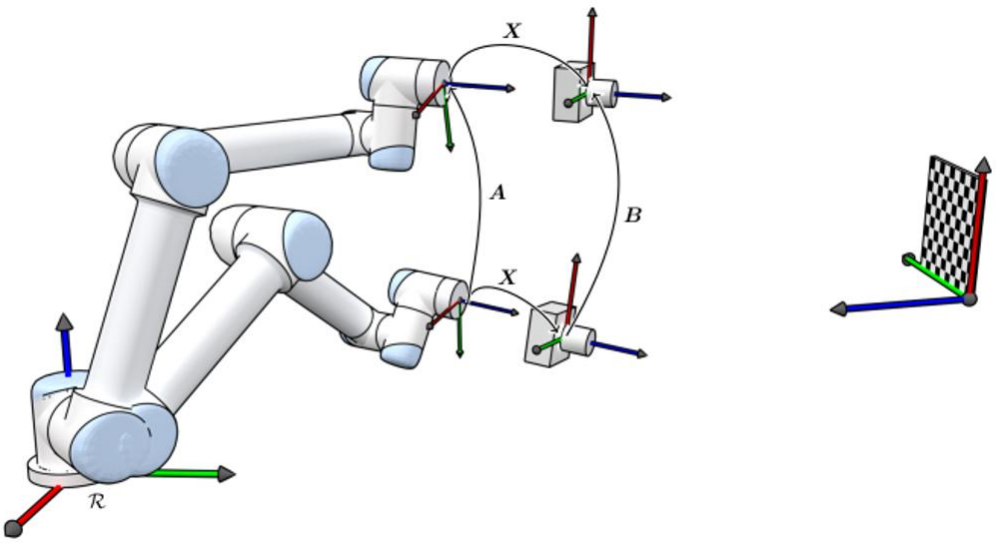

Figure 3-7 Hand-Eye AX=XB illustration. Adapted from [27]

Optimization

Similar to the pinhole camera model, the hand–eye calibration problem involves nonlinearities due to the rotation and translation components in the transformation matrices. As a result, the problem must be either linearized or solved using an iterative optimization approach. To estimate the unknown transformation X, the following reprojection error is minimized:

$$E = \sum_i \|A_i X - X B_i\|^2 \quad (3.18)$$

This objective equation represents the discrepancy between the transformed robot motions and the corresponding camera motions across all calibration poses. To minimize this error, the Levenberg–Marquardt algorithm, previously discussed in the context of the pinhole camera model, is commonly employed. It enables efficient nonlinear least-squares optimization by balancing between gradient descent and the Gauss–Newton method, thus ensuring stable convergence even in the presence of complex transformation dynamics.

3.9 - Hand-eye poses

In the context of hand-eye calibration, the choice of poses is crucial to ensure accurate calibration results. The poses refer to the specific positions and orientations of the robot and the camera used during the calibration process. Different systems may require different numbers of poses, but a common approach is the use of multiple poses to gather sufficient data for solving the transformation between the camera and robot frames.
One well-known methodology for hand-eye calibration is proposed by Cognex, which recommends a set of four mandatory poses for the calibration process. These poses are designed to cover a wide range of orientations and positions of the camera relative to the robot, ensuring that the system is calibrated in a variety of real-world scenarios. The key aspects of these poses are as follows:

1. **Variety of Orientations**: The four poses should cover different orientations of the robot and the camera, ensuring that the transformation between the two frames is not biased by any particular alignment. This includes both rotational and translational variations.
2. **Displacement of the Camera**: In practice, the camera is often mounted on the robot's end effector. The poses should ensure that the camera's position changes, thus providing a broad spectrum of data to help establish the accurate transformation matrix.
3. **Ensuring Geometric Diversity**: The poses should be chosen such that they are geometrically diverse, meaning that the positions should not be collinear or coplanar. This diversity helps avoid errors and ambiguities in the calibration results, leading to a more accurate relationship between the robot and the camera.
4. **Capture of Singularities**: Some calibration methods may encounter singularities or situations where the system's solution becomes undefined due to the geometry of the poses. By utilizing a set of diverse poses, such singularities can be avoided, allowing for robust and reliable calibration.

5. **Accuracy and Repeatability**: In the hand-eye calibration process, ensuring the accuracy and repeatability of the poses is paramount. Each pose should be executed with high precision, as small deviations could lead to errors in the final calibration. The four poses should be measured with high accuracy using appropriate sensors or methods.

Cognex's use of four mandatory poses has been proven to be effective for ensuring optimal calibration. It provides a balanced approach by using a limited number of poses that together cover the essential degrees of freedom needed for accurate calibration. The set of four poses enables the system to gather enough information to compute the hand-eye transformation matrix without being overly redundant or computationally expensive. For optimal results, however, Cognex suggests providing a total of nine plate views, which includes the four plate views mentioned earlier plus five more. While the four essential poses provide a strong starting point for hand-eye calibration, adding five more plate views extends the geometric coverage, enhances precision, and ensures robustness against environmental factors and errors. This set of nine plate views maximizes the accuracy of the transformation between the camera and robot frames, which is crucial for high-quality, real-time robotic vision tasks.

The **Figure 3-8** shows the four mandatory poses (1-4) plus the five more recommended (5-9):

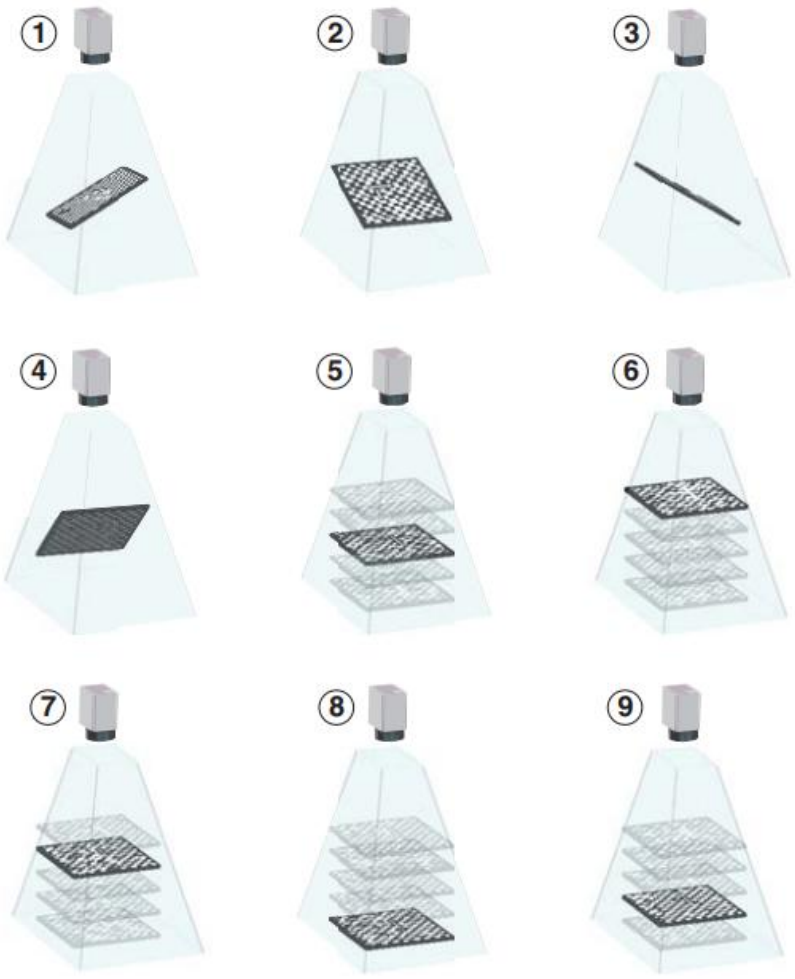

Figure 3-8 Camera Poses for Hand-Eye Calibration.[28]

3.10 - State of the art

Hand–eye calibration is a fundamental procedure in robotic vision systems, enabling precise spatial alignment between a robot's end-effector and an externally mounted or fixed camera. This transformation is crucial for tasks such as visual servoing, 3D reconstruction, object manipulation, and autonomous navigation, where the accuracy of robot–camera coordination directly impacts system performance.

The core challenge lies in determining the rigid-body transformation between the coordinate frame of the camera and that of the robot's end-effector, typically formulated through the previously discussed equation $A_iX = XB_i$.

Over the years, a wide range of methods have been developed to address this problem, evolving from classical closed-form solutions to modern iterative and optimization-based approaches.

Additionally, the emergence of high-resolution imaging sensors, machine learning techniques, and integrated simulation platforms has further enhanced the applicability and performance of hand–eye calibration algorithms across industrial and research domains.

This section presents a comprehensive overview of some influential and widely adopted methodologies in hand–eye calibration, outlining their mathematical formulations, computational characteristics, advantages, and limitations.

Foundational Methods in Hand–Eye Calibration

Tsai and Lenz Method

One of the most influential and widely cited works in the field of hand–eye calibration is the method introduced by Tsai and Lenz in 1989 [3]. Their technique addressed the problem of determining the rigid transformation X between a camera (or sensor) mounted on a robot end-effector and the end-effector itself. This transformation is crucial for enabling precise vision-guided robotic manipulation.

Tsai and Lenz formalized the calibration problem through the now-standard and previously discussed:

$$A_iX = XB_i$$

Their approach decomposes the problem into two sub-problems:

1. **Rotation estimation**: Solved using a linear method based on axis-angle representation.
2. **Translation estimation**: Solved using a least-squares method after determining rotation.

This two-step decoupled approach significantly reduced computational complexity and remains a baseline against which newer methods are often compared. Although newer techniques offer improved accuracy or robustness in noise-prone environments, the Tsai–Lenz algorithm remains foundational due to its simplicity, efficiency, and interpretability.

Daniilidis Dual Quaternion Method

Building upon the limitations of matrix-based methods, Daniilidis proposed a novel approach in 1999 using dual quaternions to represent both rotation and translation in a unified algebraic

framework [4]. Dual quaternions extend classical quaternions by including a dual part, allowing compact representation of rigid body transformations.
This formulation offers several advantages:

- **Compactness**: A dual quaternion has only eight parameters compared to the sixteen in a 4×4 transformation matrix.
- **Avoids decoupling**: Unlike the Tsai–Lenz method, Daniilidis' approach jointly solves for rotation and translation, leading to potentially higher numerical stability.
- **Robustness**: Dual quaternions inherently avoid singularities and ambiguities that can arise in matrix decomposition techniques.

Dual quaternions are an extension of quaternions that simultaneously encode both rotation and translation in a compact and algebraically convenient way, making them particularly suitable for 3D rigid transformations.
The dual quaternion representation is particularly effective in solving the equation:

$$\widehat{A_\iota} \otimes \hat{X} = \hat{X} \otimes \widehat{B_\iota} \quad (3.20)$$

where $\widehat{A_\iota}$, $\widehat{B_\iota}$, and $\hat{X}$ are dual quaternions, and $\otimes$ denotes the quaternion multiplication operator.
where:

- $\widehat{A_\iota}$ encodes the relative motion of the robot end-effector between two time steps (i.e., the transformation from pose i to pose i+1 with respect to the robot base),
- $\widehat{B_\iota}$ represents the corresponding relative motion of the camera (i.e., the transformation between two camera poses with respect to the observed world frame),
- $\hat{X}$ is the unknown transformation from the robot end-effector frame to the camera frame — the goal of the hand–eye calibration.

To solve this equation, Daniilidis reformulated it into a homogeneous linear system, which is a set of linear equations that can be expressed in matrix form. He then used a mathematical technique called eigendecomposition, which breaks down a matrix into a set of special vectors (called *eigenvectors*) and corresponding values (called *eigenvalues*). In simple terms, these eigenvectors indicate directions in which a transformation stretches or compresses space, and the eigenvalues indicate how much stretching or compression occurs. This method selectes the eigenvector associated with the smallest eigenvalue because it corresponds to the solution that introduces the least amount of error in the system, thereby producing the most accurate transformation.
This method is especially advantageous because it provides a closed-form solution that naturally handles the combined nature of rotation and translation through the dual quaternion representation.
The dual quaternion approach has influenced a broad spectrum of recent work in robot vision, particularly in cases where compact, robust formulations are required, such as in embedded systems or optimization pipelines.

Automatic and System-Level Hand–Eye Calibration

Antonello et al. [15] proposed a fully automatic hand–eye calibration method designed for streamlined integration in industrial environments. Their system combines robot motion planning, computer vision, and pose estimation in a fully autonomous calibration pipeline, removing the need for manual supervision during data acquisition.
A key component of their approach is the use of ArUco markers, which are square, 2D visual patterns similar to QR codes. These markers are easy to detect and track using a camera and are used as reference points to estimate the camera's position and orientation (pose) relative to the marker. By attaching ArUco markers to a calibration board and capturing images from different robot positions, the system gathers the necessary data for hand–eye calibration.
The novelty of Antonello's method lies in its high degree of automation and its robustness to noise. The system reduces the impact of calibration errors introduced by imperfect robot kinematics or visual noise by capturing multiple measurements and applying robust optimization techniques. This makes the solution especially attractive for industrial applications where robots may need to be frequently recalibrated without interrupting production. Its plug-and-play design supports seamless deployment in dynamic manufacturing environments where speed and repeatability are critical.

Han et al. [19] introduced a new mathematical formulation of the hand–eye calibration problem, expressed as $A_iX = YB_i$. Unlike the classical $A_iX = XB_i$ formulation, which solves for a single unknown transformation X between the robot end-effector and the camera, their model introduces a second unknown Y, representing the transformation from a global reference system (such as the world frame) to the robot base. This dual-transformation framework enables decoupled estimation of the camera-to-end-effector and world-to-robot-base transformations.
This new model is particularly useful in multi-sensor environments, such as those incorporating iGPS (indoor Global Positioning System). iGPS is a high-precision indoor localization system that uses infrared beacons and sensors to track the positions of objects in a workspace. Unlike traditional calibration setups that rely solely on robot encoders or external cameras, iGPS offers a flexible way to track both the robot and camera independently, which is helpful when one or both are mobile or not rigidly fixed.
The proposed method increases calibration flexibility and accuracy, especially in complex or dynamic setups. Experimental validation using an iGPS system showed that the dual-model formulation improves the reliability of calibration results and remains compatible with classical hand–eye models under certain conditions. This makes it a valuable extension of the standard framework and a strong candidate for advanced industrial and research applications.

Enebuse et al. [20] provided a systematic comparison of multiple hand–eye calibration methods under uncertain environmental conditions. Their study evaluates classical algorithms (e.g., Tsai–Lenz) and newer optimization-based techniques by simulating disturbances such as noisy kinematics, uncalibrated cameras, and sensor drift.
Key findings include:

- Some methods are more resilient to random noise but fail under systematic bias.
- Optimization-based approaches such as nonlinear least-squares minimization and bundle adjustment offer better accuracy in noisy scenarios.

- The initial guess in iterative solvers plays a crucial role in convergence and precision.

Their results are relevant for real-world deployments where perfect measurements are rarely guaranteed. The work emphasizes the need to select calibration methods based on specific operating conditions and performance trade-offs.

M. L. Ribeiro [13] investigates the integration of machine vision systems in industrial robotic environments, with a particular focus on the hand–eye calibration process. Among the surveyed literature, this study stands out due to its methodological similarity to the present research, both in terms of scope—robot-camera calibration—and the industrial application context. Ribeiro employs the Cognex VisionPro software for implementing the calibration routines, which include functionalities for estimating the hand–eye transformation. However, while the tool offers robust built-in algorithms, it introduces significant constraints: the calibration procedures are tightly coupled to the Cognex ecosystem, making the solution less adaptable to other vision systems or third-party hardware. Moreover, the hand–eye calibration capabilities within VisionPro require a specific license, posing cost and accessibility challenges for broader industrial or academic use. These limitations underline the importance of developing more flexible and hardware-agnostic calibration methods, as pursued in the present work.

MATLAB-Based Implementation

Another contribution to the field of hand–eye calibration comes from the commercial implementation provided by MathWorks titled *"Estimate Pose of Moving Camera Mounted on a Robot"* [21]. This resource outlines a complete workflow for calibrating a camera mounted on a robot end-effector using MATLAB's Computer Vision Toolbox. The methodology described is particularly valuable due to its flexibility, allowing users to integrate various types of vision sensors and robotic systems by leveraging MATLAB's extensive computational environment. The solution is based on the classic $A_iX = XB_i$ formulation and supports both estimation of camera intrinsics and extrinsic hand–eye calibration through an accessible, script-based interface. Unlike proprietary vision software, this implementation is versatile and does not depend on a specific hardware brand, making it more adaptable to heterogeneous setups often encountered in academic and research environments. However, it is important to note that while powerful, this approach requires a solid understanding of MATLAB programming and access to licensed toolboxes, which may present a barrier in cost-sensitive applications or open-source-focused projects. Nonetheless, it serves as a strong foundation and reference for developing robust and reproducible hand–eye calibration pipelines.

OpenCV Tools

An important open-source contribution to hand–eye calibration is provided by OpenCV [40], a widely adopted computer vision library offering robust and accessible tools for camera and sensor calibration. Specifically, a function that supports multiple calibration methods—such as

Tsai–Lenz and Daniilidis' dual quaternion algorithm—allowing flexible implementation depending on the specific requirements of the system configuration.
The major advantage of the OpenCV approach is its open-source nature, making it freely available and adaptable for both academic and industrial use. In contrast to proprietary tools, OpenCV places no restrictions on hardware compatibility, enabling seamless integration with a variety of cameras and robot platforms. Due to these strengths, the OpenCV library was selected and adapted in this dissertation as the core framework for implementing the hand–eye calibration routine. The adaptation process involved integrating OpenCV's calibration function with pose data extracted from a robot controller and a calibrated vision system, allowing for a fully custom and reproducible calibration pipeline suited to the experimental setup of this work. This choice ensures portability, cost-efficiency, and future extensibility, especially in research environments where flexibility and transparency are key.

3.11 - Conclusions

The field of hand–eye calibration has matured significantly, evolving from classical closed-form solutions like Tsai–Lenz to advanced formulations leveraging dual quaternions, optimization strategies, and sensor fusion. As highlighted, each method offers specific trade-offs between accuracy, computational efficiency, ease of implementation, and hardware dependence. While commercial solutions such as MATLAB and Cognex VisionPro offer powerful toolkits, they often come with constraints related to licensing and ecosystem compatibility. Conversely, open-source alternatives—particularly OpenCV—provide flexible, transparent, and adaptable tools that align with the demands of modern academic and industrial research.
The diversity of methods also reflects the varying operational contexts in which hand–eye calibration is applied—from tightly controlled manufacturing lines to dynamic, sensor-rich environments.

Chapter 4

Proposed architecture

This chapter presents a comprehensive overview of the system architecture developed to support the hand-eye calibration process. It begins by outlining the functional and technical requirements of the system, followed by a detailed explanation of the communication framework established between the robot, programmable logic controller (PLC), industrial camera, and personal computer (PC). The chapter highlights the specific roles and responsibilities of each component within the calibration workflow, emphasizing how they interact to ensure synchronized and reliable operation. Furthermore, the architectural design of the PLC, robot, and PC software is described in depth, including the control logic, data flow mechanisms, and inter-process coordination required to successfully execute the calibration routine.

4.1 - Requirements

The only requirement defined by the company for the hand-eye calibration process was that the final estimated transformation error should remain below 2 mm. No further constraints were imposed regarding the methodology or tools to be used. However, for the successful execution of the calibration procedure, several system-level requirements were necessary. These included ensuring that the robot's end-effector performed at least the nine mandatory poses previously discussed, the availability of an external PC to store the captured images and perform the hand-eye calibration computations, and maintaining a controlled environment with consistent lighting conditions to ensure image quality, specifically, good illumination and minimal visual noise.

4.2 - Updated robotic cell

Due to the computational complexity involved in hand-eye calibration, performing this process directly on the PLC was deemed unsuitable. Hand-eye calibration typically relies on iterative

optimization algorithms to minimize the transformation error between the robot and the camera coordinate systems. These iterative estimations demand substantial processing power and numerical capabilities, which exceed the real-time control-oriented architecture and limited computational resources of most PLCs. As a result, an external PC was integrated into the robotic cell to handle the hand-eye calibration procedure. This allowed the use of more sophisticated mathematical libraries and optimization routines, ensuring accurate and efficient calibration without overloading the PLC.

Communication between the PC and the PLC is established via the OPC-UA protocol. This protocol was selected due to several key advantages that align well with the system requirements. Firstly, OPC UA is fully compatible with asynchronous communication frameworks such as PROFINET, making it suitable for integration into industrial environments where real-time deterministic communication is not strictly required. Secondly, OPC UA provides a standardized, platform-independent interface that simplifies access to PLC variables. By operating as an OPC UA server, the PLC exposes its internal variables to the PC, which can read and write data directly without relying on proprietary drivers or complex configurations. This enables seamless and reliable data exchange, ensuring effective coordination between the calibration process and the robot's control logic.

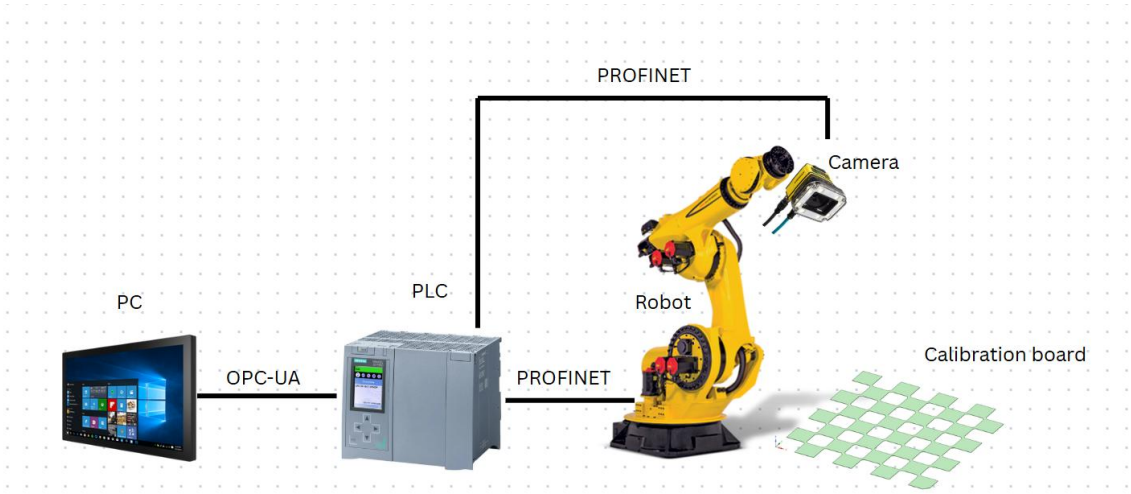

Figure 4-1 Updated robotic cell

4.3 - Operation of the Robotic Cell

The control architecture of the robotic cell is structured around the PLC, which serves as the central coordination unit for all system components: the industrial robot, the vision system (camera), and the host PC. The PLC is responsible for managing all communication and synchronization tasks within the system. This includes coordinating the exchange of commands and status signals between the robot, camera, and PC. In addition to its control role, the PLC serves as a central repository for global system variables, such as the current robot pose and the results of the calibration process. This centralized management ensures consistent data availability and reliable operation across all system components.

The PLC is responsible for issuing movement commands to the robot, instructing it to sequentially move to a series of predefined calibration poses.
Due to limitations in the communication protocol between the robot and the PLC, it is not possible to transmit floating-point (real-number) data directly. To overcome this, the robot encodes its position and orientation—normally represented with decimal values—into a format that can be transmitted using only integers or bits (explained further in this chapter). This encoded data is then sent to the PLC, along with a signal indicating that the robot has reached its desired position. Upon receiving the data, the PC decodes it back into its original numerical format and stores the robot's position and orientation. This process ensures that accurate positional data is shared between the robot, the PLC and the PC, despite the communication limitations.
Simultaneously, the PLC sends trigger commands to the camera, instructing it to capture calibration images. The camera returns an acknowledgment once the image acquisition is complete. This ensures synchronization between image capture and robot positioning.

The host PC connects to the PLC via an OPC-UA server. The PLC's OPC-UA server exposes several variables for read and write access, including the current robot pose components, the *Start Calibration* and *Calibration Done* signals, the *PC command*, an acknowledgement that the PC has saved the robot pose, and the resulting hand-eye calibration translation and rotation vectors. With the robot poses and the acquired images, two critical calibration procedures are done:

1. **Camera Calibration**: The PC associates the 2D image coordinates (captured by the camera) with the corresponding 3D robot world coordinates. This process allows the estimation of the camera's extrinsic parameters relative to the robotic cell.
2. **Hand-Eye Calibration**: Using the known robot poses and the estimated camera extrinsics, the PC computes the transformation between the robot's end-effector coordinate system and the camera coordinate system.

Once the calibration is complete, the resulting transformation matrix is written back to the PLC through the OPC-UA interface. This allows subsequent operations within the robotic cell to make use of the calibrated spatial relationship between the robot and the camera.

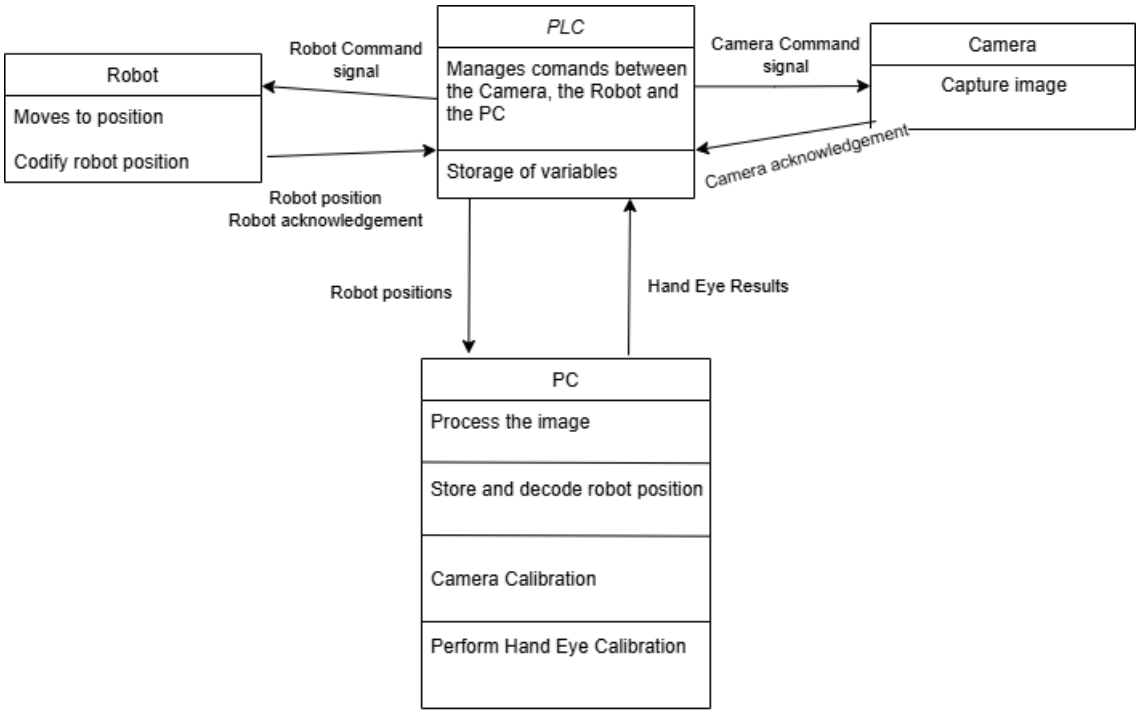

Figure 4-2 Functioning of the Robotic Cell

4.4 - PLC

The calibration process is managed by two main **PLC ladder programs**, developed as part of this dissertation, each with a distinct role. The first program oversees the overall calibration sequence, coordinating the high-level flow of operations. The second program handles pose-specific tasks by managing the sequence of commands that must be executed for each individual robot pose.

Calib Function Block

The first PLC program (*Calib Function Block*) is responsible for managing the overall **calibration sequence** and coordinating the interaction between the robot and the PC. Initially, the PLC remains in a waiting state until it receives a signal indicating that the calibration program on the PC has started (*Calibration Start signal*). Once this signal is detected, the PLC triggers the robot to begin executing the calibration poses explained further in this chapter (*Program 1*). For each pose, the PLC calls the second program (*Sequence Function Block*), which manages the specific commands for the camera, the robot, and the PC. This cycle continues until the PC signals that the calibration process is complete (*Calibration Done signal*). At that point, the PLC encodes the calibration results to be sent to the robot and transitions into an operation mode. In this mode, it verifies whether a specific object is present in the robot's predefined observation pose. If the object is detected, the PLC instructs the robot to go to the object. If the PC starts another calibration, the process is repeated.

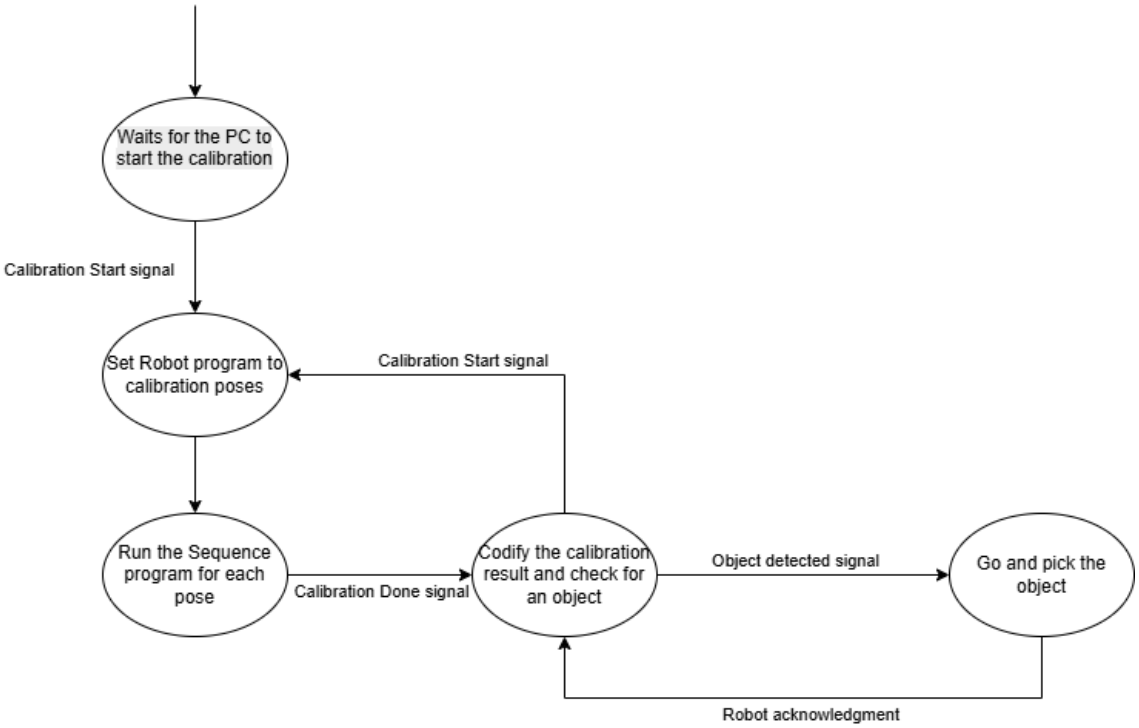

Figure 4-3 PLC Calib Function Block

Sequence Function Block

The second PLC program is responsible for managing the sequence of operations associated with each individual **calibration pose** (*Sequence Function Block*). The sequence begins with the PLC verifying that the camera is online. Once confirmed, the PLC sends a trigger command to the camera to capture an image (*trigger*) and waits for an acknowledgment (*Camera acknowledgement*) confirming that the image has been taken. Following this, the PLC sends a command to the PC (*PC command*), instructing it to retrieve and store the current robot pose. It then waits for an acknowledgment from the PC indicating that the pose has been successfully recorded (*PC acknowledgement*). Once this step is complete, the PLC issues a command to the robot to move to the next predefined position. It first waits for an initial acknowledgment (*Acknowledgement 1*) from the robot confirming that the command was received, followed by a second acknowledgment (*Acknowledgement 2*) indicating that the robot has reached the target position. Upon receiving both confirmations, the PLC repeats the entire process for the next pose in the sequence.

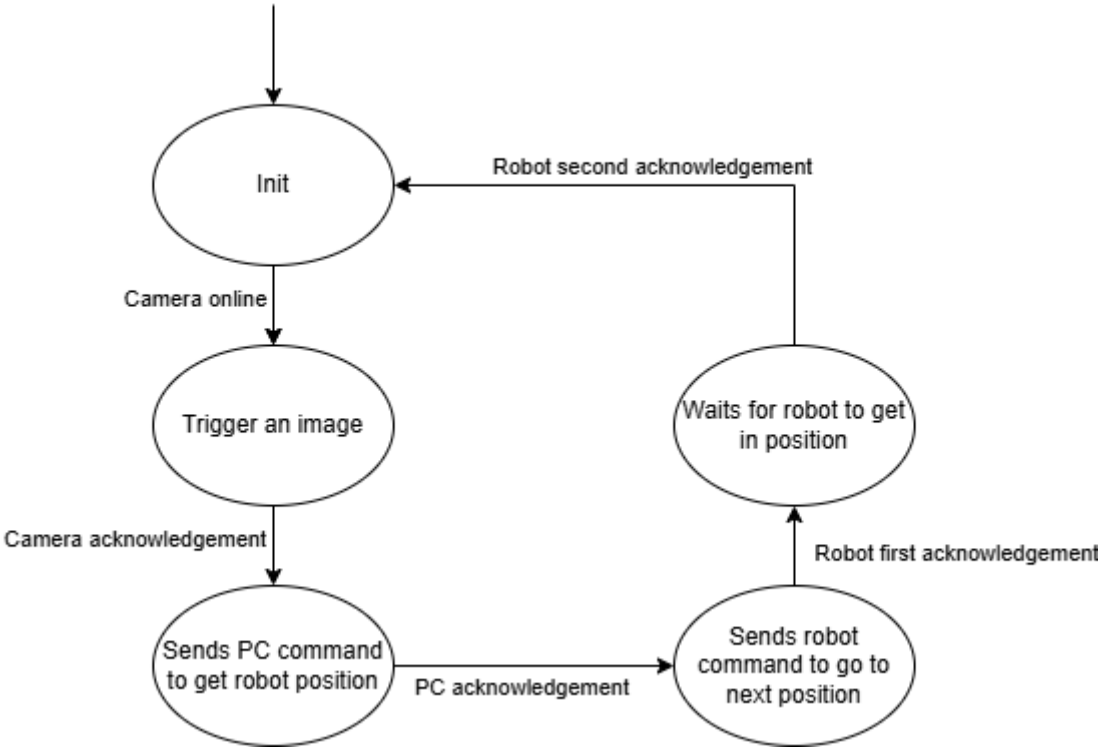

Figure 4-4 PLC poses sequence

This second PLC program operates in a cyclic manner and does not terminate on its own. It is only executed when explicitly called by the first PLC program and continues running during that specific step. Once the first program advances to the next step, the second program is effectively paused until it is called again.
The division of responsibilities between the two PLC programs enables efficient coordination and control of the calibration process, ensuring synchronization among the robot, camera, and PC, while allowing for systematic data acquisition and operational flexibility throughout the procedure.

4.5 - Robot architecture

The robot control system is organized into four main programs, developed for this dissertation: the ***Main Program***, ***UPDATETOOL Program***, ***GOTOPOINT Program***, and ***Program1***. These programs are executed directly on the robot controller and are written using **TP (Teach Pendant) language**, FANUC's standard programming environment.
The main program oversees general robot operations. The ***UPDATETOOL*** program decodes the hand-eye calibration results and updates a specific robot tool accordingly. The ***GOTOPOINT*** program commands the robot to move to the location of the detected object. Lastly, ***Program 1*** is responsible for guiding the robot through the calibration poses and encoding the necessary data during the calibration process.

Robot Main program

The main program executed by the robot follows a structured sequence designed to accommodate two primary operational modes: system calibration and object manipulation. The program begins by invoking the ***UPDATETOOL*** program, which ensures the current tool parameters and configurations are correctly initialized.
Following this initialization step, all acknowledgment flags exchanged with the PLC are reset to zero, ensuring a clean start for the upcoming operations. Once the reset is complete, the robot moves to a predefined **observation pose**, which serves as both the initial position for the calibration routine and the default monitoring position during regular operation. This pose is carefully selected to ensure that the camera—mounted on the robot's end-effector—has a clear and unobstructed view of the calibration grid, enabling accurate image capture for corner detection. Additionally, this pose is strategically chosen to provide optimal visibility of the workspace, allowing the camera to detect the presence of predefined objects or features. As such, the observation pose serves a dual purpose: it is the starting point for the hand-eye calibration procedure and the vantage point for object detection during system runtime. From this pose, the robot enters a standby state, awaiting a command from the PLC to proceed with the next operation.
At this point, the robot monitors two input signals from the PLC:

- **Start Calibration Signal**: If this signal is set to TRUE (logical 1), the robot starts the calibration routine by executing Program 1. This program controls the sequential movement of the robot through a set of predefined calibration poses used for system calibration tasks.
- **Object Detected Signal**: If this signal is instead set to TRUE, the robot bypasses the calibration routine and proceeds to execute the **Go To Point** program. This routine commands the robot to move (jog) to a specified location corresponding to the position of a detected object.

After either one of the programs runs, the main program starts again, repeating the cycle.

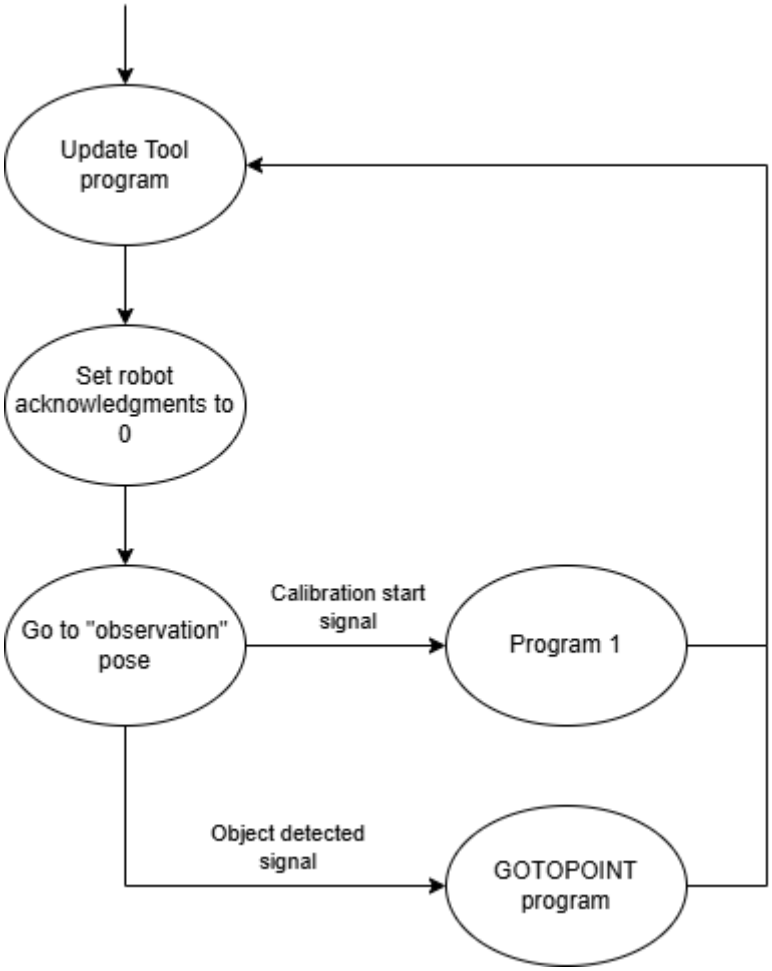

Figure 4-5 Robot Main

Robot Program 1

Program 1 is responsible for executing the robot's calibration routine, where the robot moves sequentially through a series of predefined calibration poses. The program operates in a looped structure, allowing each pose to be handled in a synchronized and deterministic manner.
For each calibration pose, the program begins by waiting for a command signal from the PLC (*Robot command*). Once this command signal is set to TRUE, the robot sets *Acknowledgment 1* to TRUE, indicating that it has successfully received the command to proceed. Following this acknowledgment, the robot jogs to the specified calibration pose.

Upon reaching the target pose, the robot encodes its Cartesian position before sending it to the PLC, due to a communication constraint that prevents the direct transmission of real-number data.
In the FANUC system, communication with the PLC is handled through:

- **DI (Digital Inputs)** and **DO (Digital Outputs)**: Binary signals used to represent discrete on/off states or control flags.
- **GI (Group Inputs)** and **GO (Group Outputs)**: Used to transmit 16-bit integers, enabling the exchange of numeric data in a compact format.

To overcome the limitation of not being able to send floating-point values, *PROGRAM 1* uses a custom encoding method. Each Cartesian coordinate is split into two parts: the integer portion and the decimal portion. One **GO** is used to send the integer part, while a second **GO** carries the decimal part. A **DO** is reserved as a flag to signal if the value is a positive or negative number. This allows the PC to reconstruct the full numeric value using only the available integer-based communication channels. The encoding method used for this transformation is detailed in the next chapter.
After encoding the position, the robot sets *Acknowledgment 2* to TRUE, signaling to the PLC that it has both reached the desired pose and successfully encoded its position.
This sequence — receiving the command, moving to the target pose, encoding the position, and sending acknowledgments — is repeated iteratively until all calibration poses have been completed.

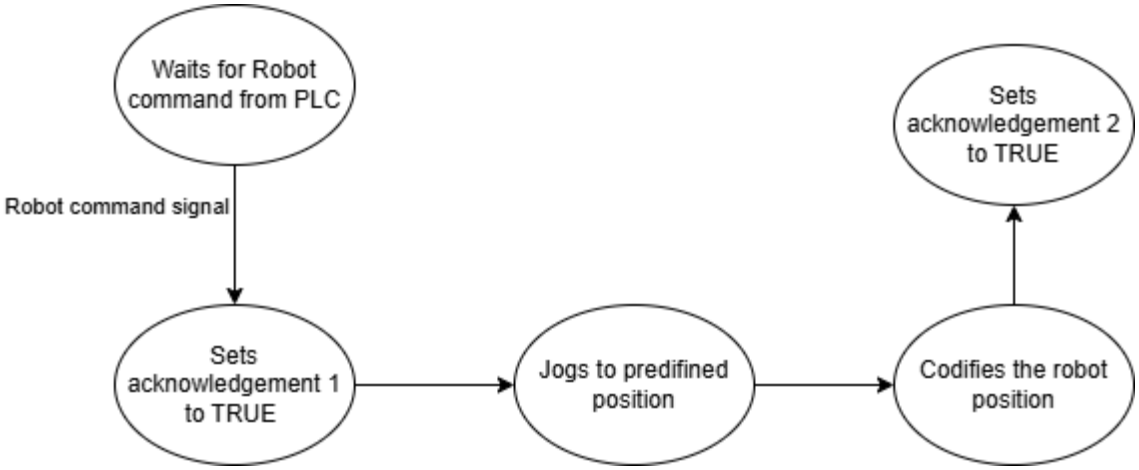

Figure 4-6 Robot Program 1

This program is executed repeatedly for each calibration pose in the sequence.

Robot GOTOPOINT program

The ***GOTOPOINT*** program is responsible for executing the robot's movement toward a detected object location, based on input provided by the PLC. Upon program start, the robot sets *Acknowledgment 1* to TRUE to indicate that the command has been received and the execution process is starting.
The robot then proceeds to decode the target position sent by the PLC. This decoding process is the inverse of the encoding method previously applied for robot-to-PLC communication, and ensures accurate reconstruction of the object's position in Cartesian space.
Once the target position has been successfully decoded, the robot jogs to the specified location corresponding to the object's position. This operation allows the robot to precisely reach the object for visual confirmation of the hand-eye results.

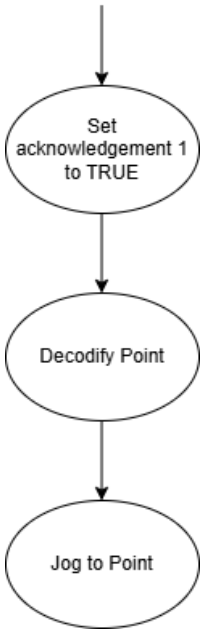

Figure 4-7 Robot GOTOPOINT program

Robot UPDATETOOL program

The ***UPDATETOOL*** program is responsible for configuring the robot's tool transformation based on the result of the hand-eye calibration. Specifically, the program decodes the transformation values (x, y, z, w, p, r) obtained from the hand-eye calibration process—previously stored and encoded within the PLC—and applies this transformation to update the corresponding tool frame in the robot's controller.

By updating the tool definition with the accurate hand-eye transformation, this program ensures that subsequent robot motions are properly aligned with the calibrated camera coordinate system.

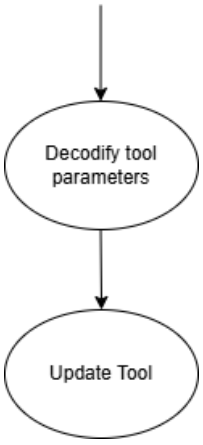

Figure 4-8 Robot UPDATETOOL program

4.6 - PC program architecture

On the PC side, the calibration process is handled by a program in Python developed for this dissertation. The program begins by clearing all previously stored calibration images, ensuring that no residual data from earlier sessions interferes with the current calibration process. Once the storage is reset, the PC sets the *Start Calibration* signal to TRUE, notifying the PLC that a new calibration sequence is being initiated.

Following this, the PC waits for a command signal from the PLC (*PC command*), indicating that the robot has reached a calibration pose and that the corresponding position data is ready to be recorded. Upon detecting this signal, the PC retrieves and stores the robot's position—as made available through the OPC-UA server—and then sends an acknowledgment back to the PLC, confirming that the data has been successfully captured.

This cycle of waiting for the command, storing the position, and sending acknowledgment is repeated for each calibration pose. Once all calibration poses have been completed and the corresponding positions saved, the PC proceeds to the image processing stage.

At this point, the PC retrieves all images captured during the calibration procedure and detects the grid points within each image. Using these image points and the known coordinates of the calibration grid, the PC establishes correspondences between 2D image points and 3D world points.

With these correspondences, the program performs a camera calibration, estimating the extrinsic parameters of the camera for each pose. These extrinsic parameters, which define the camera's position to the world frame, are then paired with the corresponding robot poses to compute a hand-eye calibration.

The result of the hand-eye calibration—namely, the translation and rotation vectors between the robot end-effector and the camera coordinate system—is then transmitted back to the PLC via the OPC-UA interface, completing the calibration workflow.

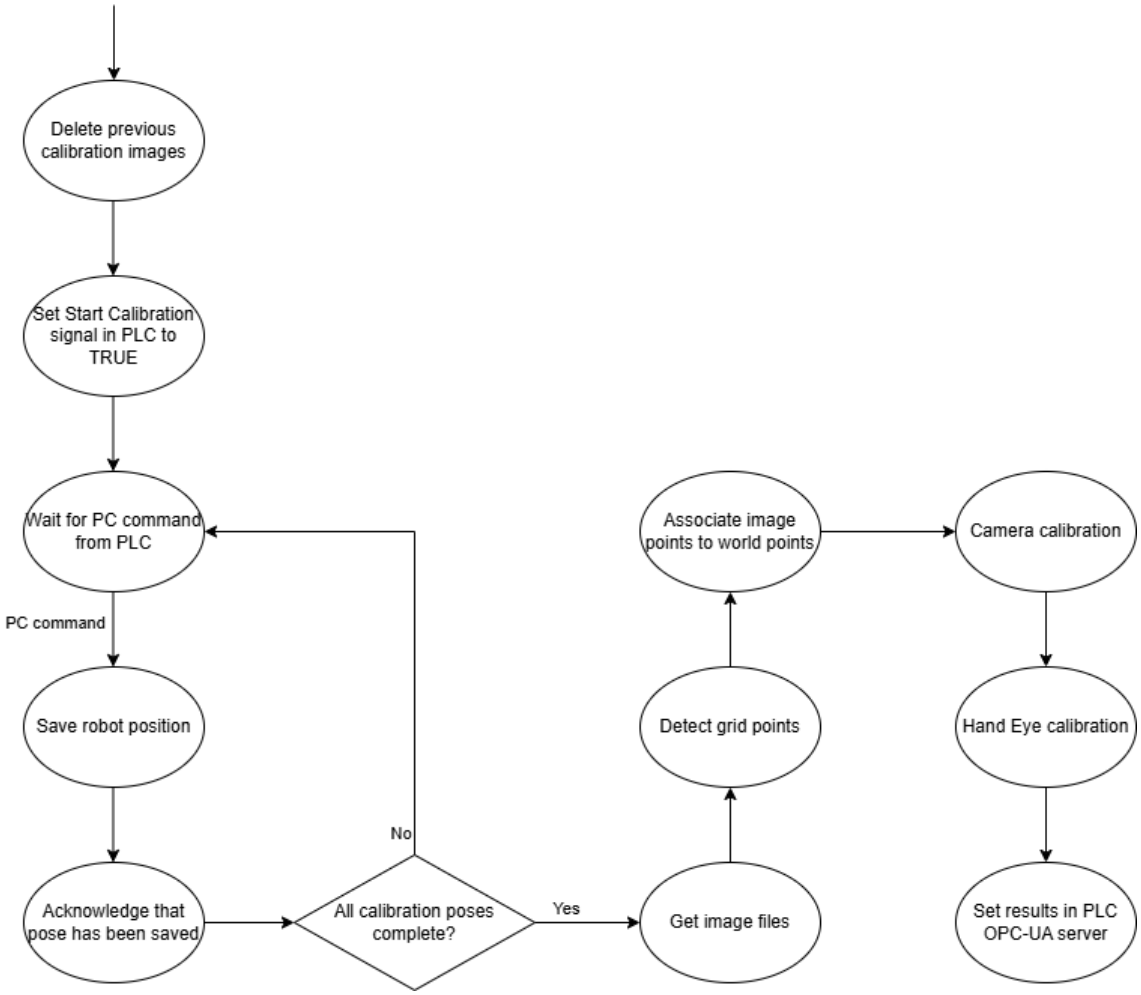

Figure 4-9 PC calibration program

The Python PC program, named ***Hand_Eye_Auto***, is responsible for performing the hand-eye calibration procedure. Its inputs consist of the robot end-effector poses—captured at specific calibration positions—and the corresponding images acquired by the camera at those same positions. Using these images, the program estimates the pose of the camera with respect to the world coordinate system.

Given the known robot poses relative to the robot base and the estimated camera poses relative to the world, the program computes the transformation between the camera and the robot end-effector. The output is a result expressed as a translation vector and a rotation vector, which together describe the rigid-body transformation from the camera frame to the robot end-effector frame.

The retrieval and decoding of the robot pose are handled by a custom Python function called *get_robot_position()*. This function reconstructs the original real-number pose values from the encoded data sent by the robot. The encoding method, described in a previous section, uses three separate signals per axis: one for the integer part, one for the decimal part, and a binary flag indicating the sign of the number. Specifically, the final decoded value is computed by

summing the integer part (*x*) and the decimal part (*x_dec*), with the sign determined by the signal flag (*x_signal*), where 1 represents a negative value and 0 represents a positive value.
After decoding, the complete robot pose (*X, Y, Z, W, P, R*) is stored in two matrixes (*robot_translation_vectors and robot_rotation_vectors*) that accumulates the translation (*X,Y,Z*) and the rotation (*W,P,R*) vectors for each calibration step.

The detection of the calibration grid points is performed using the open-source computer vision library OpenCV [40], specifically the *cv2.findChessboardCorners()* function. This function automatically detects the inner corners of the chessboard pattern in the input image and returns their pixel coordinates, ordered from left to right and top to bottom.
Once the image points are detected, they are associated with the corresponding real-world coordinates of the calibration grid. In this case, the calibration grid used features a square spacing of 10 mm. The X and Y coordinates are calculated by multiplying the grid indices by the square size (10 mm). This generates a set of world points in the world coordinate system.

Having established the correspondence between the world and image points, the camera pose in relation to the world is estimated using the previously explained ***Pinhole camera model** [2].* To estimate the two unknown parameter matrices—**intrinsic** and **extrinsic**—an iterative optimization process is required. This estimation is performed using the *cv2.calibrateCamera()* function from the OpenCV library [40]. The function takes as input the previously defined sets of corresponding 3D world points and 2D image points and applies nonlinear optimization to minimize the reprojection error. As a result, it outputs the camera's intrinsic parameters and the extrinsic parameters, which describe the pose of the camera relative to the world. For the purpose of hand-eye calibration, only the **extrinsic parameters**—representing the camera's pose with respect to the calibration grid—are used in subsequent steps.

Having obtained the camera poses relative to the world frame and the corresponding robot end-effector poses relative to the robot base, the program proceeds to perform hand-eye calibration. This calibration step involves solving the previous explained $AX = XB$ formulation, originally introduced by Tsai and Lenz [3]. Since the solution to this formulation requires iterative estimation methods, the OpenCV library function *cv2.calibrateHandEye()* [40] is employed. This function takes as input the series of camera-to-world and robot-to-base transformations and estimates the rigid transformation X, which represents the camera's position and orientation relative to the robot end-effector.
The resulting transformation is expressed as a rotation vector and a translation vector, together defining the spatial relationship between the camera and the robot's end-effector. Finally, these results are transmitted to the PLC via the OPC UA server, using the OPC-UA Python library to create a client connection and write the estimated values to the appropriate nodes in the PLC's address space.

4.7 - Conclusions

The architecture of the updated robotic cell demonstrates the feasibility of performing accurate hand-eye calibration in an industrial environment by offloading computational tasks

to an external PC and assigning coordination to the PLC. Separating control and processing allowed each component to operate within its performance constraints while maintaining system-level synchronization.
The use of OPC-UA for communication enabled flexible data exchange without requiring tight integration between platforms.
The architecture confirmed that reliable calibration can be achieved by adding an external PC. This design can be extended or adapted for other calibration or vision-based automation tasks, supporting scalability and modular integration in similar robotic applications.

Chapter 5

Implementation

This chapter describes the implementation of the Hand_Eye_Auto system, a Python-based program that performs automatic hand-eye calibration between a robot and a vision system. The program coordinates image capture and robot pose acquisition, computes camera and hand-eye calibration parameters using OpenCV, and communicates with a PLC via OPC-UA.
It also covers the supporting robot controller programs and PLC function blocks that manage robot movements, synchronization signals, and calibration steps.

5.1 - PC program

The Python program, is a calibration procedure involving a robotic arm and a camera. The calibration involves the use of images of a chessboard pattern for camera calibration and measuring the transformation between the camera and the robotic tool.

Main Program

The Main Program executes in the following order:

Initialization:
The initial part of the code sets several configurations for the calibration process:

- The location in the PC of the images to be used for calibration (*image_folder*).
- The PLC OPCUA server url (*url*).
- The **grid size** (*grid_size*) and **square size** (*square_size*) used for the chessboard detection.
- The camera resolution (*camera_resolution*) in pixels.
- Calculates the grid origin (in this case, it is the calibration board center).

```
# --- Initialization ---
url = "opc.tcp://192.168.10.10:4840"
client = Client(url)
image_folder = "C:\\Users\\NunoAlves\\Desktop\\Results\\"  # Path to the images folder
image_files = [f"Image{str(i).zfill(5)}.jpg" for i in range(9)]  # List of images
grid_size = (23, 13)  # Number of squares in the calibration grid
square_size = 10.0  # Size of the squares in mm
camera_resolution=(800,600) # Camera resolution in mm
```

Figure 5-1 Program initialization

Deleting Old Images:

The code scans the specified folder for image files (with extensions like .png, .jpg, etc.) and deletes all images. This step clears the directory before starting a fresh calibration process.

```
image_extensions = ("*.png", "*.jpg", "*.jpeg", "*.bmp", "*.tiff")

for ext in image_extensions:
    for image in glob.glob(os.path.join(image_folder, ext)):
        os.remove(image)  # Apaga cada imagem encontrada
```

Figure 5-2 Deleting Old Images

Triggering the Calibration Process:

The OPC-UA client (PC) is used to trigger the start of the calibration process by writing a True value to a specific OPC-UA node. This triggers the calibration to begin on the robot or system you're working with.

```
client.connect()
node = client.get_node("ns=4;i=104")  #start calibration
node.set_value(ua.DataValue(ua.Variant(True, ua.VariantType.Boolean)))
client.disconnect()
```

Figure 5-3 Start Calibration

Data acquisition loop:

Within the Main Program, a loop iterates over each calibration pose (9 for this specific example) and performs the following sequence of operations:

Waiting for the Robot's Signal:

The loop enters a while start == False loop, waiting for the PLC to signal that it's ready to capture the robot position.

Robot Position Registration:

The function *get_robot_position()* explained further in this chapter is called to fetch the current position and orientation of the robot. This will be used as part of the hand-eye calibration process. The program appends these values to the lists *robot_rotation_vectors* and *robot_translation_vectors*.

Acknowledging the PLC:

After collecting the data, the loop communicates back with the OPC-UA server, signaling that the data has been captured successfully.

```
#Cycle for the number of positions
for i in range(9):
    while start == False:
        client.connect()
        node = client.get_node("ns=4;i=86") #PC command
        start = node.get_value()
        client.disconnect()
    start = False
    #Register robot position
    robot_translation,robot_rotation=get_robot_position()
    robot_rotation_vectors.append(robot_rotation)
    robot_translation_vectors.append(robot_translation)
    print("\nCalculating calibration number ", i + 1)
    #Acknowledge the PLC that the robot position was registed
    client.connect()
    node = client.get_node("ns=4;i=85")
    node.set_value(ua.DataValue(ua.Variant(True, ua.VariantType.Boolean)))
    client.disconnect()
```

Figure 5-4 Data acquisition loop

Image Loading and Processing:

After all the robot positions have been recorded, the program searches for all image files in the folder (*image_files*) and sorts them by modification date.

The script then loads the 9 images (assuming the directory contains at least 9 images), processes each one, and attempts to detect the chessboard corners using the OpenCV [40] function *cv2.findChessboardCorners()*.

Once the pixel coordinates of the calibration grid points are detected in the image, the program associates each image point with a corresponding 3D world point based on the known geometry of the calibration grid, which features a fixed square spacing of 10 mm. This association assumes a consistent, top-left to bottom-right ordering of points. The image coordinates are stored in the image_points vector, while the corresponding 3D coordinates—defined relative to the grid's center—are stored in the world_points vector. This correspondence provides the necessary input for camera calibration.

```python
# Search for all the images in the image folder
image_files = []
for ext in image_extensions:
    image_files.extend(glob.glob(os.path.join(image_folder, ext)))

# Sort the images for modification date (oldest first)
image_files.sort(key=os.path.getmtime)

# Select the 9 oldest images
image_files = image_files[:9]

# --- Load and process every image ---
for image_file in image_files:
    image_path = image_file
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error loading the image: {image_file}")
        continue

    # Detect Calibration Grid points
    ret, corners = cv2.findChessboardCorners(image, grid_size, None)

    if ret:
        # Add image points for calibration
        image_points.append(corners.reshape(-1, 2))

        # Generate world points with origin to the grid center
        world_points = np.zeros((grid_size[0] * grid_size[1], 3), np.float32)
        for i in range(grid_size[1]):
            for j in range(grid_size[0]):
                x = j * square_size - center_x
                y = center_y - i * square_size
                world_points[i * grid_size[0] + j] = [x, y, 0]  # Position (X, Y, Z=0)

        object_points.append(world_points)
```

Figure 5-5 Image processing

Camera Calibration:

After all nine images have been processed and their corresponding image and world points have been collected, the camera calibration is performed using the *cv2.calibrateCamera()* function from the OpenCV library [40]. This function implements an iterative optimization algorithm to estimate the parameters of the pinhole camera model [2]. Specifically, it uses the correspondences between 2D image points and their associated 3D world points to solve for the camera's intrinsic matrix (*K*), distortion coefficients (*dist*), and the extrinsic parameters—namely, the rotation and translation vectors (*rvecs*, *tvecs*) that describe the pose of the camera with respect to the world coordinate system. The result is an estimation of the camera's internal characteristics and its spatial relationship to the observed calibration grid.

```python
# Camera Calibration
ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(object_points, image_points, camera_resolution, None, None)
```

Figure 5-6 Camera Calibration

Hand-Eye Calibration:

The hand-eye calibration is carried out using OpenCV's *cv2.calibrateHandEye()* [40] function, which estimates the rigid transformation between the robot's end-effector (gripper) and the camera mounted on it. This function solves the $AX = XB$ formulation originally proposed by Tsai and Lenz [3], where $\boldsymbol{A}$ and $\boldsymbol{B}$ represent the poses of the robot and camera, respectively, and $\boldsymbol{X}$ is the transformation from the camera frame to the gripper frame. The solution is obtained through iterative optimization, using pairs of poses collected during the calibration procedure—specifically, the robot poses relative to its base and the camera poses relative to the calibration grid (world frame).

The output of the algorithm is composed of a rotation matrix (*R_cam2gripper*) and a translation vector (*t_cam2gripper*) that together define the transformation between the camera and the robot end-effector.

```
#Calibração mão olho
R_cam2gripper,t_cam2gripper=cv2.calibrateHandEye(
    robot_rotation_vectors,robot_translation_vectors,rvecs,tvecs,cv2.CALIB_HAND_EYE_TSAI)
```

Figure 5-7 Hand Eye calibration

Rotation matrix to rotation vector:

Since the output of the hand-eye calibration algorithm provides the rotation component as a 3×3 matrix (*R_cam2gripper*), and the robot controller requires rotational information in the form of a rotation vector (**w, p, r**) expressed in degrees, it is necessary to convert the rotation matrix into a rotation vector. This transformation is typically performed using the Rodrigues formula, which converts a rotation matrix into its axis-angle representation, allowing for the extraction of the rotation vector in a format compatible with the robot's control system. In practice, this is done using OpenCV's [40] built-in function *cv2.Rodrigues()*, which takes a rotation matrix as input and returns the corresponding rotation vector. This result is then converted to degrees and formatted to match the input requirements of the robot controller.

```
r_cam2gripper,_= cv2.Rodrigues(R_cam2gripper)

r_cam2gripper_deg= np.degrees(r_cam2gripper)
```

Figure 5-8 Convertion of a rotation matrix to a rotation vector

Final Step - Setting Values in OPC-UA Server:

After all the calibration is done, the transformation vectors (*R_cam2gripper, t_cam2gripper*) are sent back to the OPC-UA server, allowing the robot to know the results of the calibration.

```
# Translation vector t_cam2gripper
node = client.get_node("ns=4;i=163")
node.set_value(ua.DataValue(ua.Variant(t_cam2gripper[0], ua.VariantType.Float)))
node = client.get_node("ns=4;i=164")
node.set_value(ua.DataValue(ua.Variant(t_cam2gripper[1], ua.VariantType.Float)))
node = client.get_node("ns=4;i=165")
node.set_value(ua.DataValue(ua.Variant(t_cam2gripper[2], ua.VariantType.Float)))

# Vetor de rotação r_cam2gripper
node = client.get_node("ns=4;i=229")
node.set_value(ua.DataValue(ua.Variant(r_cam2gripper_deg[0], ua.VariantType.Float)))
node = client.get_node("ns=4;i=230")
node.set_value(ua.DataValue(ua.Variant(r_cam2gripper_deg[1], ua.VariantType.Float)))
node = client.get_node("ns=4;i=231")
node.set_value(ua.DataValue(ua.Variant(r_cam2gripper_deg[2], ua.VariantType.Float)))

client.disconnect()
```

Figure 5-9 Send Hand Eye results

Ending the Calibration:

Once all the values have been written to the OPC-UA nodes, the *Calibration Done* signal is sent to the server, marking the end of the calibration process.

```
client.connect()
node = client.get_node("ns=4;i=226")  #calibration done
node.set_value(ua.DataValue(ua.Variant(True, ua.VariantType.Boolean)))
print("\nCalibration Done\n")
client.disconnect()
```

Figure 5-10 Calibration Done signal

Robot Position

The *get_robot_position()* is a developed function for this dissertation that registers the robot's position by communicating with the PLC over the OPC UA protocol. It retrieves robot position and orientation values like x, x_dec, x_signal, and decodes them, returning the translation and rotation matrices representing the robot's pose.

```
def get_robot_position():

    client.connect()
    node = client.get_node('ns=4;i=172') #x
    x = node.get_value()
    node = client.get_node('ns=4;i=173') #x_dec
    x_dec = node.get_value()
    node = client.get_node('ns=4;i=166') #x_signal
    x_signal = node.get_value()
```

Figure 5-11 Get robot position

The process illustrated in Figure 5-11 is repeated for each of the remaining robot pose parameters: y, z, w, p, and r.

Once all parameters are retrieved, the decoding process is executed. This involves dividing the decimal part by 1000 to scale it appropriately, and then adding it to the corresponding integer part.

Finally, the signal bit is evaluated to determine the sign of the number. A value of 1 indicates that the number is negative, while 0 denotes a positive value. This reconstruction ensures that the original real-valued robot pose data is accurately recovered for use in the calibration process.

```
x_dec=x_dec/1000
y_dec=y_dec/1000
z_dec=z_dec/1000
w_dec=w_dec/1000
p_dec=p_dec/1000
r_dec=r_dec/1000

x=x+x_dec
y=y+y_dec
z=z+z_dec
w=w+w_dec
p=p+p_dec
r=r+r_dec

if(x_signal==True):
    x=-x

if(y_signal==True):
    y=-y

if(z_signal==True):
    z=-z

if(w_signal==True):
    w=-w

if(p_signal==True):
    p=-p

if(r_signal==True):
    r=-r
```

Figure 5-12 Position decoding

5.2 - PLC Program Blocks Overview

The PLC program blocks manage the calibration procedure by acting as an interface between the *Hand_Eye_Auto* program, the robot, and the camera system. Below is an overview of the two primary function blocks discussed in the previous chapter (***Calib Function Block*** and ***Sequence Function Block***), along with two additional function blocks that are invoked within them. The first of these auxiliary blocks is the ***Codify Function Block***, which processes the hand-eye calibration results and updates them via the OPC-UA interface. The second is the ***Transform Point Function Block***, which converts a point defined in the world coordinate frame into the corresponding point in the robot base coordinate frame.

Calibration Function Block

The ***Calib_FB*** (Calibration Function Block) is responsible for handling robot operations in response to commands from the PC. It operates in a sequence of steps, transitioning through different network stages:

1. **Network 1: Initialization and Program Start**

The PLC waits for the PC to trigger the *Start_Calibration* signal.
Once triggered, the PLC sets *offsetCrippa* to 1, initiating ***Program 1*** in the robot, which drives the robot through a sequence of 9 calibration positions.
After completing ***Program_1***, the system resets calibration variables and moves to **Network 2**.

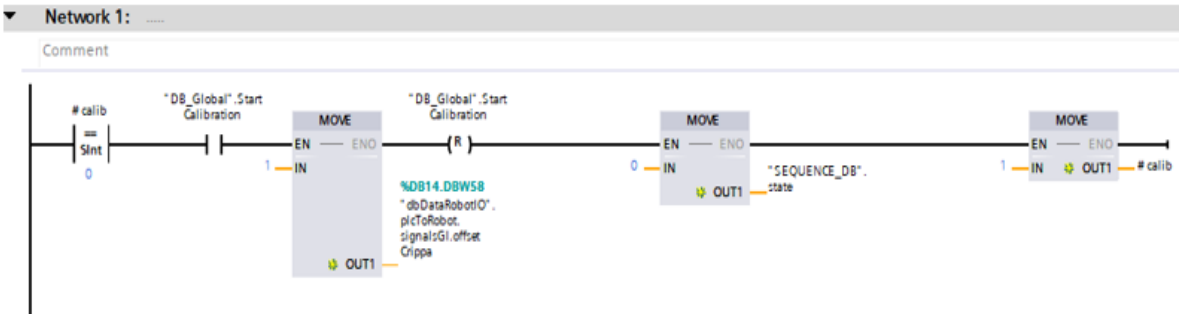

Figure 5-13 Calibration initialization in ladder.

2. **Network 2: Sequence Management**

The ***SEQUENCE_FB*** is activated to handle image capturing and coordinate updates.
The PLC triggers the vision system to capture an image, sends the robot to the next position, and requests the PC to save robot position data.
After receiving a confirmation (*Calibration_Done*), the PLC transitions to **Network 3**.

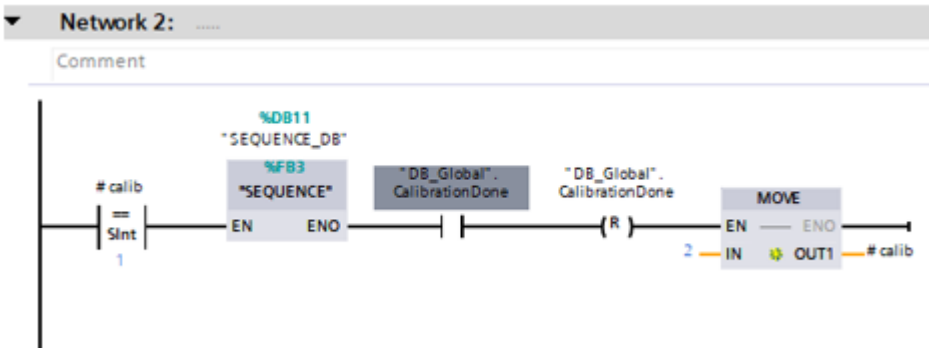

Figure 5-14-Sequence management in ladder.

3. **Network 3: Main Program Execution**

The PLC sets *offsetCrippa* to 0, signaling the robot to execute its ***Main Program***.
This program includes moving the robot to an "observation" position and updating the camera TCP (Tool Center Point).
The ***Codify_FB*** is executed to encode the camera TCP for robot use.
The process then returns to **Network 1** for potential reactivation if the *Start_Calibration* variable is triggered again.

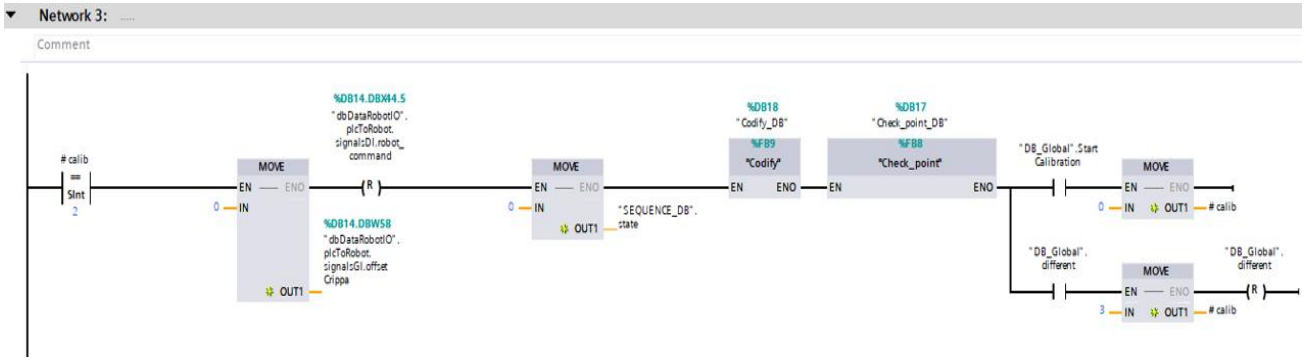

Figure 5-15 Main program execution in ladder.

4. **Optional: Checkpoint and Point Transformation**

The ***Check_point FB*** compares the current object position with the previous one. If a change is detected, the PLC moves to **Network 4**.
In **Network 4**, the ***Transform_point FB*** converts the camera's observed position into robot base coordinates. The robot is then directed to move to this point.
The PLC also sets *offsetCrippa* to 2, signaling the robot to run the ***GOTOPOINT*** program, which decodes the position and moves the robot accordingly.

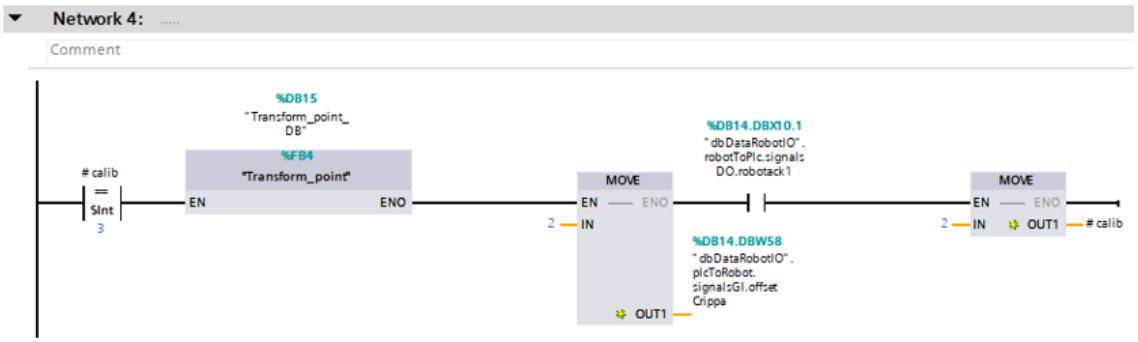

Figure 5-16 Checkpoint and point transformation in ladder.

This part is optional since the calculation of the point will be heavy on the PLC processing, being not ideal for small processing power PLC's. To avoid this, the TCP of the camera is corrected using *Codify* from **Network 3**. By correcting the Camera TCP the robot can change it's position to the previous calibrated "observation" position.

Sequence Function Block

The ***Sequence_FB*** (Sequence Function Block) manages the execution of calibration tasks for each position in the sequence.

1. **Network 1: Camera Online Check**

The PLC checks if the camera is online. Once confirmed, it transitions to **Network 2**.

Network 1:

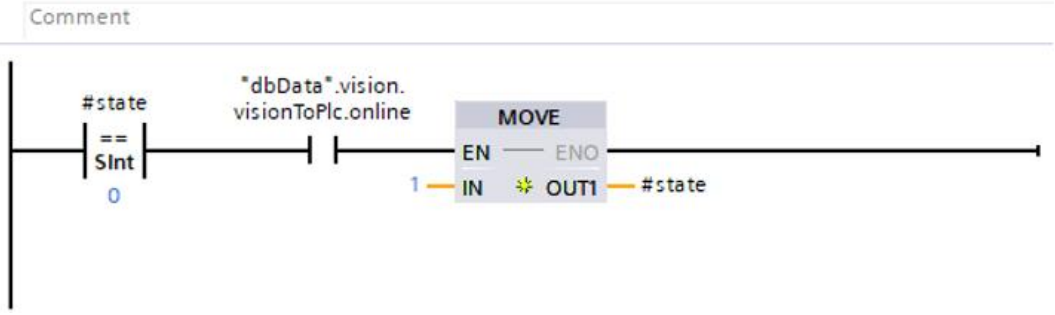

Figure 5-17 Camera online check in ladder.

2. **Network 2: Image Triggering**

The PLC sends a command to the vision system to capture an image.
The PLC waits for an acknowledgment from the vision system before transitioning to the next Network.

Network2:

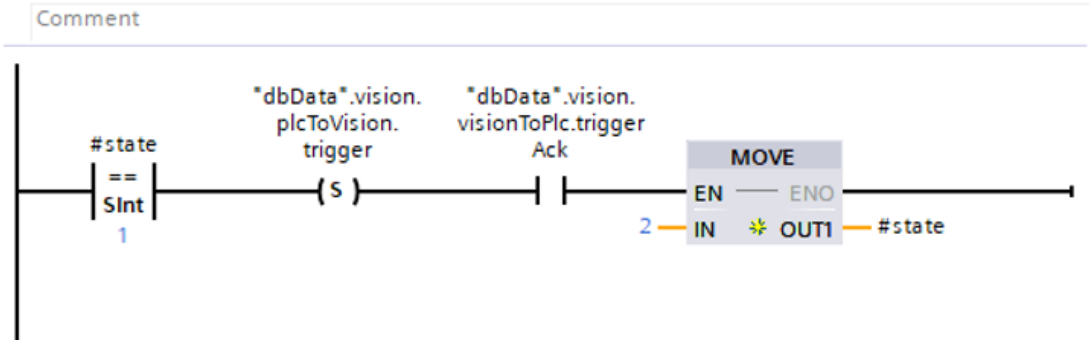

Figure 5-18 Image triggering in ladder.

3. **Network 3: Robot Position Capture**

After resetting the image trigger, the PLC commands the PC to capture the robot's position. The PLC waits for confirmation from the PC before moving to **Network 4**.

Network 3:

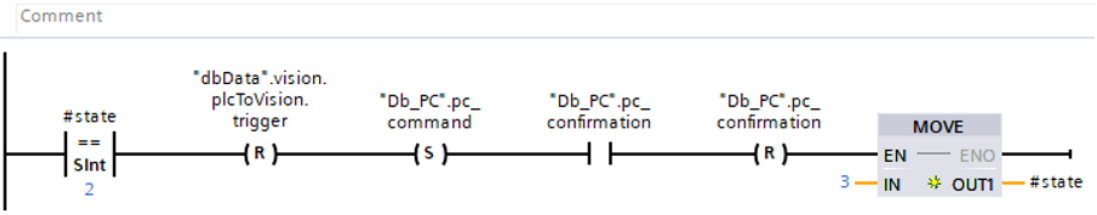

Figure 5-19 Robot Position Capture in ladder.

4. **Network 4: Robot Command and Acknowledgment**

The PLC issues a command to the robot and waits for acknowledgment that the command was received.

The value of *offsetCrippa* is set to 0, as the robot will resume its ***Main Program*** after the calibration sequence.

Once acknowledged, the PLC moves to **Network 5**.

Network 4:

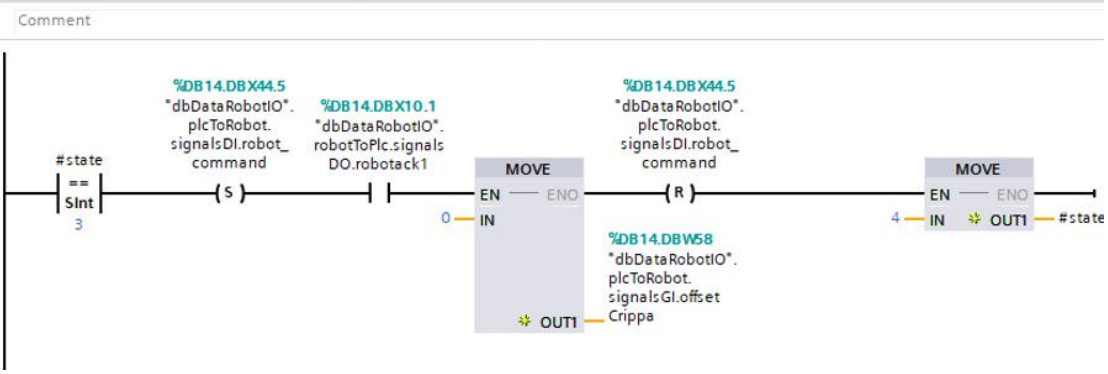

Figure 5-20 Robot command and acknowledgment in ladder.

5. **Network 5: Robot Arrival Confirmation**

The PLC waits for the *Acknowledgement 2* (robot*ack2*), confirming that the robot has reached the designated position.
After receiving this confirmation, the sequence restarts at **Network 1** and continues this loop until the *Calibration Done* signal is received from the PC.

Network5:

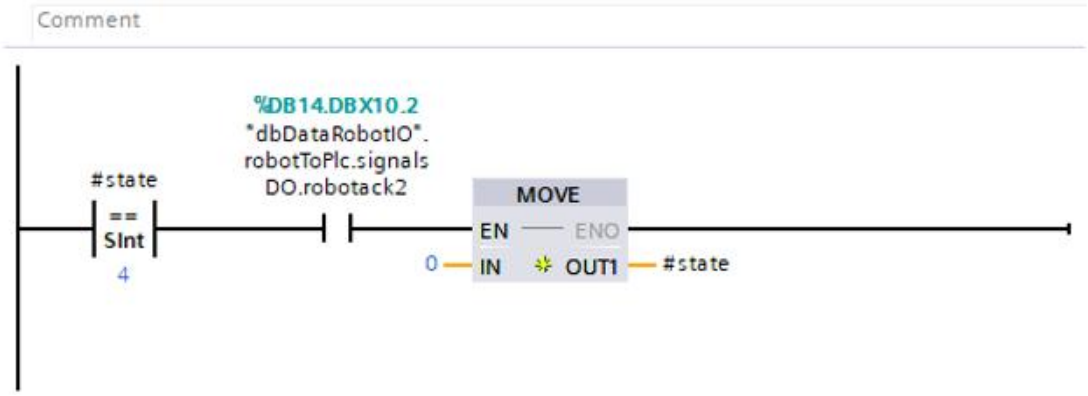

Figure 5-21 Robot arrival confirmation in ladder.

Codify Function Block

The ***Codify_FB*** (Codification Function Block) handles the encoding of the camera's translation and rotation in relation to the gripper, specifically the *t_cam2gripper* and *r_cam2gripper* values, and updates the OPC-UA interface with the codified values .

1. The function checks the polarity of the translation and rotation values.
    - If negative, a specific DI (Digital Input) is set to 1; otherwise, it is set to 0.
    - Negative values are converted to positive.
2. The integer part of the values is stored in a GI (Global Input).
3. The decimal part is multiplied by 1000 and stored in another GI.

Transform point Function Block

The ***Transform_point FB*** (Transformation Function Block) is responsible for converting the observed point in the camera's coordinate system to a corresponding point in the robot's base coordinate system.

1. **Step 1: World to Camera Transformation**

The coordinates in the world frame (P_{world}) are transformed into the camera frame (P_{cam}) using a rotation matrix ($R_{world2cam}$) and translation vector ($t_{world2cam}$).
The general transformation equation in this case would be:

$$P_{cam} = R_{world2cam} \times P_{world} + t_{world2cam} \quad (5.1)$$

Where:

- P_{world} is the position of the object in the world frame. $P_{world} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- $R_{world2cam}$ is the rotation matrix that describes the orientation of the camera relative to the world frame. $R_{world2cam} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ where r represent the rotation of the camera in relation to the world.

- $t_{world2cam}$ is the translation vector that describes the position of the camera relative to the world frame. $t_{world2cam} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

This allows the robot to understand the object's position relative to the camera.

2. **Step 2: Camera to Gripper Transformation**

The camera frame coordinates (P_{cam}) are further transformed into the gripper frame ($P_{gripper}$) using the rotation matrix ($R_{cam2gripper}$) and translation vector ($t_{cam2gripper}$). The general transformation equation here would be:

$$P_{gripper} = R_{cam2gripper} \times P_{cam} + t_{cam2gripper} \quad (5.2)$$

Where:

- P_{cam} is the position of the object in the camera frame. $P_{cam} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- $R_{cam2gripper}$ is the rotation matrix that describes the orientation of the gripper relative to the camera. $R_{cam2gripper} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ where r represent the rotation of the gripper in relation to the camera.

- $t_{cam2gripper}$ is the translation vector that describes the position of the gripper relative to the camera. $t_{cam2gripper} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

This transformation aligns the object's position with the gripper's frame, which is used for robotic manipulation.

3. **Step 3: Gripper to Base Transformation**

Finally, the gripper frame coordinates ($P_{gripper}$) are converted into the robot base frame (P_{base}) using the rotation matrix ($R_{gripper2base}$) and translation vector ($t_{gripper2base}$):

$$P_{base} = R_{gripper2base} \times P_{gripper} + t_{gripper2base} \quad (5.3)$$

Where:

- $P_{gripper}$ is the position of the object in the gripper frame. $P_{gripper} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- $R_{gripper2base}$ is the rotation matrix that describes the orientation of the gripper relative to the robot's base. $R_{gripper2base} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ where r represent the rotation of the base in relation to the gripper.

- $t_{gripper2base}$ is the translation vector that describes the position of the gripper relative to the robot's base. $t_{gripper2base} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

At this point, the object's position has been fully transformed into the robot's base frame, where it can be used to control the robot's actions (e.g., moving the arm to the object).
The process of transforming object coordinates between frames is a critical aspect of robot kinematics. The rotations and translations provide a mathematical foundation for mapping the object's position and orientation in one reference frame to another. These transformations are essential for ensuring that the robot can interact with objects accurately based on the data received from sensors like cameras. By chaining these transformations (camera → gripper → robot base), the robot can perform tasks like pick-and-place or manipulation effectively.

5.3 - Robot Controller Functionality and Communication with PLC

The robot controller is structured around a main function that handles multiple tasks essential for proper operation and communication with the Programmable Logic Controller (PLC). Initially, the controller performs tool updates when necessary and resets both acknowledgment signals (*Acknowledge 1 and Acknowledge 2*) to zero. It then moves the robot to the predefined "observation" pose, if required, and enters a waiting state to receive further instructions from the PLC.
The PLC can issue one of the following commands:

- Keep the robot in the "observation" pose (*GI[8]=0*).
- Execute a sequence of calibration points (*GI[8]=16*).
- Move the robot to a specific point sent by the PLC (*GI[8]=32*).

The acknowledgment signals play a crucial role in synchronizing operations between the PLC and the robot:

- ***Acknowledge 1*** (*DO[1078])* indicates that the robot has received a movement command.
- ***Acknowledge 2*** (*DO[1079])* indicates that the robot has reached the target position.

```
MAIN

   1:  LBL[1]
   2:  CALL UPDATETOOL
   3:  UTOOL_NUM=1
   4:  DO[1078]=OFF
   5:  DO[1079]=OFF
   6:J  P[1] 100% FINE
   7:  IF GI[8]=0,JMP LBL[1]
   8:  IF GI[8]=16,CALL PROG_1
   9:  IF GI[8]=32,CALL GOTOPOINT
  10:J  P[33] 100% FINE
  11:  JMP LBL[1]
[End]
```

Figure 5-22- Main program of the robot.

Calibration Sequence: PROG_1 to PROG_x

When the PLC triggers *PROG_1*, the robot initiates a series of calibration programs, sequentially executing *PROG_1, PROG_2, ..., up to PROG_x*. Each program corresponds to a specific calibration point.

In PROG_2, the process is as follows:

1. ***Acknowledge 1*** is reset to 0, and the robot waits for a movement command from the PLC (DI[1068]).
2. Once the command is received, the PLC sets ***Acknowledge 1*** to 1 and ***Acknowledge 2*** to 0.
3. The robot moves to the designated position.

```
PROG_2

   1:  DO[1078]=OFF
   2:  LBL[2]
   3:  IF DI[1068]=OFF,JMP LBL[2]
   4:  DO[1079]=OFF
   5:  DO[1078]=ON
   6:J  P[2] 100% FINE
   7:  DO[1080]=OFF
   8:  DO[1081]=OFF
   9:  DO[1082]=OFF
  10:  DO[1083]=OFF
  11:  DO[1084]=OFF
  12:  DO[1077]=OFF
  13:  PR[1]=P[2]
  14:  R[1]=PR[1,1]
  15:  R[2]=PR[1,2]
  16:  R[3]=PR[1,3]
  17:  R[4]=PR[1,4]
  18:  R[5]=PR[1,5]
  19:  R[6]=PR[1,6]
  20:  IF R[1]>0,JMP LBL[1]
```

Figure 5-23 Program for one robot pose.

4. The position data (X, Y, Z) and orientation (W,P,R) are encoded:

- All Digital Outputs (DOs) responsible for variable signaling are initially reset to 0.
- Position and orientation values are saved in individual registers.
- If any value is negative, its corresponding DO is set to 1 and the value is converted to positive.
- Each variable's integer part is stored in one Group Output (GO), and the decimal part in another GO.

This encoding process is repeated for every point in the sequence.

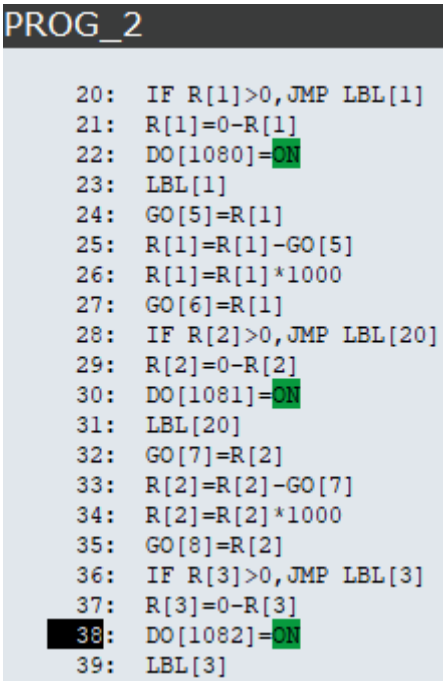

Figure 5-24 Codification of the robot position.

After reaching the position and completing the encoding, ***Acknowledge 2*** is set to 1, notifying the PLC that the robot is in position. This triggers the PC to capture the robot's location. The sequence continues by calling the next calibration program.

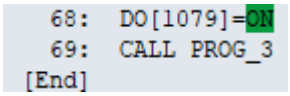

Figure 5-25 Calling of the next program (pose).

After *PROG_x* is completed, the robot returns to the main function.

GOTOPOINT Program

The *GOTOPOINT* program decodes the position data sent from the PLC and moves the robot accordingly. The sequence is as follows:

1. ***Acknowledge 1*** is set to 1 to signal that a new command is being processed.
2. Group Inputs (GIs) containing the target point data from the PLC are stored in registers.
3. If a corresponding Digital Input (DI) indicates a negative value, the value is inverted.

4. The two parts (integer and decimal) are combined to define the Position Register (PR) for X, Y, Z, and orientation.

```
GOTOPOINT
  1:  DO[1078]=ON
  2:  R[1]=GI[2]/1
  3:  R[2]=GI[3]/1000
  4:  IF DI[1031]=OFF,JMP LBL[1]
  5:  R[1]=0-R[1]
  6:  R[2]=0-R[2]
  7:  LBL[1]
  8:  PR[1,1]=R[1]+R[2]
  9:  R[1]=GI[4]/1
 10:  R[2]=GI[5]/1000
 11:  IF DI[1032]=OFF,JMP LBL[2]
 12:  R[1]=0-R[1]
 13:  R[2]=0-R[2]
 14:  LBL[2]
 15:  PR[1,2]=R[1]+R[2]
 16:  R[1]=GI[6]/1
 17:  R[2]=GI[7]/1000
 18:  IF DI[1033]=OFF,JMP LBL[3]
 19:  R[1]=0-R[1]
 20:  R[2]=0-R[2]
```

Figure 5-26 Decodification of the position.

Once the target position is reconstructed, the robot jogs to it and sets ***Acknowledge 1*** to 0. It then returns to the main program.

```
 26:J  PR[1] 100% FINE
 27:  DO[1078]=OFF
[End]
```

Figure 5-27 Jog to the position.

The *UPDATETOOL* program functions similarly to *GOTOPOINT*, with one key difference: instead of moving the robot, it updates the tool configuration with the data sent from the PLC.

```
 43:  UTOOL[2:Eoat2]=PR[1]
[End]
```

Figure 5-28 Update of the tool.

GOTOPOINT is useful when the PLC computes a point in world coordinates based on camera input. The robot can then move to that calculated position.
UPDATETOOL is beneficial when the camera is defined as a tool. In such cases, updating the tool allows the robot to maintain a consistent observation pose, even when the camera position or orientation has shifted. By recalculating the camera-to-gripper relationship, the system ensures observation consistency across different setups.

5.4 - Communications

The two communication protocols used in this implementation were **PROFINET** and **OPC UA**.
For **PROFINET** communication, integration was achieved by importing a GSD (General Station Description) file into the PLC TIA PORTAL program. A GSD file is an XML-based configuration file provided by the device manufacturer that describes the device's capabilities, including supported data types, communication parameters, and I/O configuration. Once the GSD file was added, the device—such as the camera or robot—was configured by specifying its IP address and PROFINET device name. With this setup, the PLC automatically handled the asynchronous data exchange with the device, allocating memory areas for input and output data according to the definitions in the GSD.
For **OPC UA** communication, an OPC UA server interface was created directly in the PLC. This allowed external clients—such as a PC program—to browse, read, and write PLC variables over a standardized, platform-independent protocol.

5.5 - Conclusions

The implementation of the ***Hand_Eye_Auto*** system successfully automates the hand-eye calibration process by integrating Python scripts, robot controller programs, and PLC function blocks. Although it relies on OpenCV library functions for both camera and hand-eye calibration computations, this solution remains practical, modular, and adaptable. It can be deployed across various robot platforms and camera types, offering a flexible and efficient approach to achieving accurate robot-vision alignment in industrial applications.

Chapter 6

Results and Validation

This chapter presents the results obtained from the hand-eye calibration experiments and validates their effectiveness in real-world scenarios. The evaluation focuses on quantifying the accuracy, repeatability, and robustness of the calibration process using key performance indicators such as reprojection error and transformation deviations. Several factors influencing calibration quality—including the number of poses used, lighting conditions, pose variation, and background quality—are systematically analyzed. Additionally, practical validation was carried out in an industrial setting to show how recalibration corrects positional discrepancies. The insights gained from these analyses inform best practices for achieving reliable calibration in robotic vision systems.

6.1 - Reprojection error

The ***reprojection error*** is a key metric used to evaluate the accuracy of a camera calibration process. After estimating the camera intrinsic and extrinsic parameters, the 3D points from the calibration pattern are projected back onto the 2D image plane using these parameters. The reprojection error is defined as the average distance in pixels between these projected points and the actual detected image points. It quantifies the discrepancy between the observed and predicted image locations, reflecting the quality of the calibration. A lower reprojection error indicates a more precise calibration and better alignment between the model and the real camera observations. To assess the accuracy, a function developed for this dissertation, named ***calculate_reprojection_error()***, was implemented to compute the reprojection error.

```python
def calculate_reprojection_error(world_points_array, image_points_array, K, dist, rvecs, tvecs):

    total_error = 0
    for i in range(len(world_points_array)):
        image_points_projected, _ = cv2.projectPoints(world_points_array[i], rvecs[i], tvecs[i], K, dist)
        # Calculates the error for every point
        error = np.linalg.norm(image_points_array[i] - image_points_projected.reshape(-1, 2), axis=1)
        total_error += np.sum(error)  # Sums all the errors

        mean_error=np.mean(error)

        # Prints the reprojection error
        print(f"Average reprojection error for pose {i+1}: {mean_error}")

    # Average error
    mean_error = total_error / len(world_points_array) / len(world_points_array[0])
    print(f"\nAverage reprojection error: {mean_error}")

    return total_error, mean_error
```

Figure 6-1 Reprojection error function

For each world point represented in the calibration board, the function projects it onto the image plane using the Open CV function *cv2.projectPoints()* [40] *and* compares the result with the actual observed image points.
The function accepts the following inputs: the world coordinates of the calibration points, the corresponding detected image points, the camera intrinsic matrix *K*, distortion coefficients *dist*, and the rotation and translation vectors (*rvecs,tvecs*) that describe the camera's pose relative to the world coordinate system. It computes and prints the reprojection error for each individual point and returns both the total and the mean reprojection error over all points.
It serves to validate the previously estimated intrinsic and extrinsic parameters of the camera system.

6.2 - Hand Eye program output

During the calibration process, the terminal output of the Hand Eye program displays the robot's position and orientation at each step, along with the current calibration number. Upon completion of all nine calibration positions, the message "Calibration Done" appears, followed by the final calibration results. These results include the average reprojection error for each position as well as the overall average reprojection error.
Typically, a reprojection error of less than 1 pixel is considered excellent, while an error between 1 and 2 pixels is acceptable for most practical applications. Significantly higher errors can suggest issues such as inaccuracies in the camera model, errors in feature detection, or improper observation of the calibration pattern.
Several factors can influence the reprojection error and should be considered when interpreting calibration quality:

1. **Quality of the Calibration Pattern**: A checkerboard or other calibration pattern must be flat, well-lit, and clearly visible.
2. **Number of Calibration Points**: A greater number of accurately detected points generally leads to better calibration.
3. **Camera Lens Distortion**: If not properly modeled, radial and tangential distortions, especially in wide-angle lenses, can raise the error.

4. **Camera Resolution and Image Quality**: High resolution and low noise improve point detection accuracy.
5. **Distribution of Points**: Calibration points should be spread across the entire image, including the edges.
6. **Environmental Conditions**: Lighting, reflections, and occlusions can negatively impact the visibility and detection of calibration points.

```
Todas as imagens foram apagadas.
Requested session timeout to be 3600000ms, got 30000ms instead
[1150.829   13.942  394.024] [-103.925  -66.127    6.111]

Calculating calibration number  1
[1438.826 -293.005  255.746] [-72.479 -45.61  -17.694]

Calculating calibration number  2
[1490.984   10.284  210.565] [-101.71   -43.081    3.465]

Calculating calibration number  3
[1411.94   350.427  210.535] [-130.194  -41.392   31.136]

Calculating calibration number  4
[1152.435  502.064  210.511] [-156.682  -48.457   52.855]

Calculating calibration number  5
[825.096 502.021 182.019] [168.011 -58.538  81.842]

Calculating calibration number  6
[682.37  106.832 135.61 ] [124.148 -76.993 134.074]

Calculating calibration number  7
[ 772.594 -264.393  135.601] [   6.544  -73.736 -100.983]

Calculating calibration number  8
[1092.912 -282.625  226.457] [-55.749 -63.343 -40.621]

Calculating calibration number  9
Erro médio de reprojeção para a posição 1: 0.15107831358909607
Erro médio de reprojeção para a posição 2: 0.21714062988758087
Erro médio de reprojeção para a posição 3: 0.22475074231624603
Erro médio de reprojeção para a posição 4: 0.22723643481731415
Erro médio de reprojeção para a posição 5: 0.22913089394569397
Erro médio de reprojeção para a posição 6: 0.24310356378555298
Erro médio de reprojeção para a posição 7: 0.22195245325565338
Erro médio de reprojeção para a posição 8: 0.2417726069688797
Erro médio de reprojeção para a posição 9: 0.18809527158737183

Erro médio de reprojeção: 0.21602898836135864
```

Figure 6-2 Hand Eye program Terminal Output

As illustrated in **Figure 6-2**, the program outputs to the terminal the estimated pose of the robot end-effector relative to its base frame, along with the average reprojection error (measured in pixels) for each calibration pose. Additionally, it provides the overall average reprojection error computed across all poses.

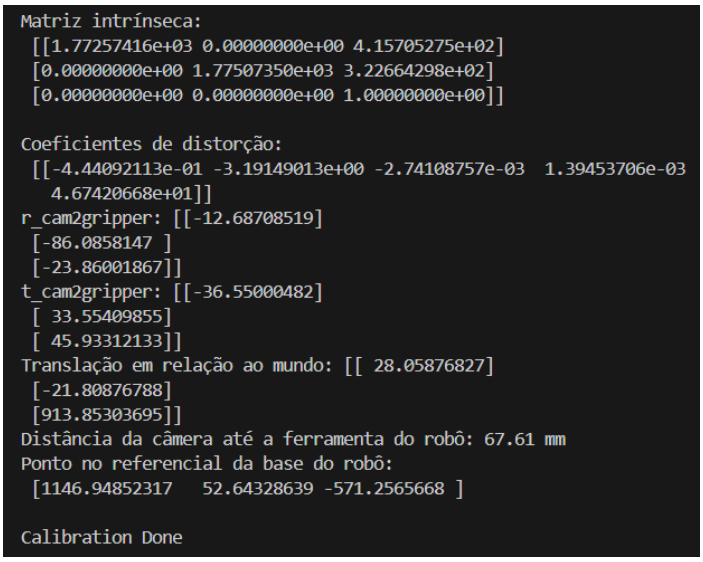

Figure 6-3 Hand Eye Terminal Output

Figure 6-3 shows additional calibration outputs, including the intrinsic camera matrix, distortion coefficients, camera-to-gripper transformation (*r_cam2gripper and t_cam2gripper)*, the camera's position relative to the chessboard origin, the distance of the camera to the gripper and the world origin expressed in the robot's base reference frame. Although explicitly knowing them is not strictly required for the calibration to be performed, they are useful for visually validating the results. Furthermore, they enhance the reusability of the estimated parameters in other industrial applications. For instance, the intrinsic or extrinsic parameters may be required in separate vision-based tasks, such as object detection, pose estimation, or robotic guidance systems.

6.3 - Validation Approach and Testing Procedure

To validate the accuracy of the calibration, the results were tested in a real-world scenario provided by JPM Industry. In this case, a shift in the camera's physical position occurred due to robotic movement, leading to a discrepancy between the calibrated and actual camera poses. In **Figure 6-4**, the camera is shown in the "observation" pose, with green crosses indicating the square intersections based on a previous calibration.

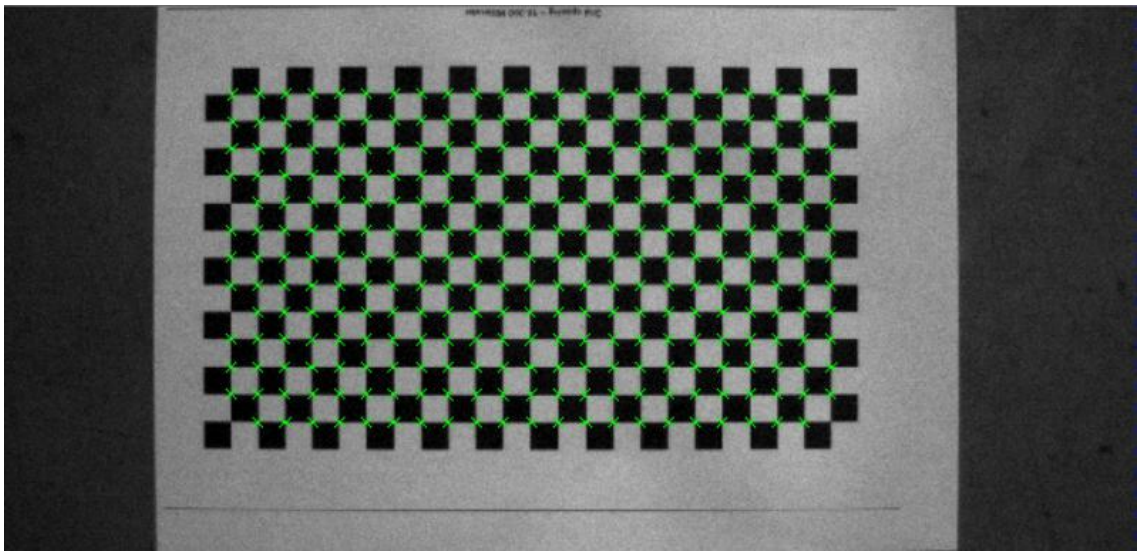

Figure 6-4 Calibrated pose

To simulate a real-world scenario in which the camera's position relative to the robot changes due to external factors, the camera was manually rotated with respect to the robot. This deliberate adjustment created a positional shift, replicating the effect of potential disturbances that may occur during robotic operation, such as mechanical vibrations, accidental impacts, thermal expansion, or wear and tear of mounting components. **Figure 6-5** reveals that the green crosses are misaligned by approximately 5 to 6 cm.

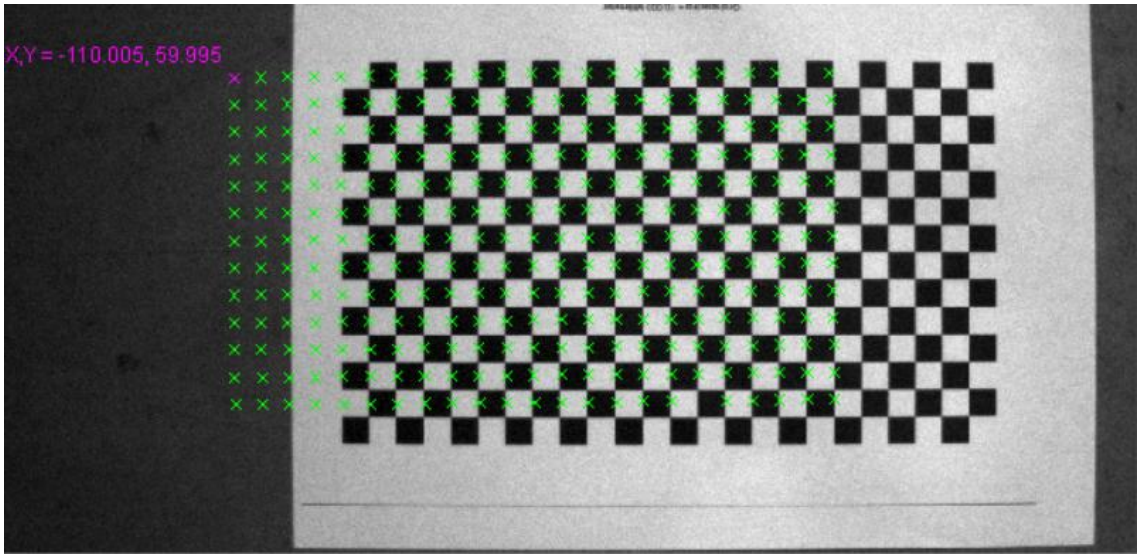

Figure 6-5 Shifted pose

To correct this, the previously saved "observation" pose was recalibrated with the Hand Eye program. The updated camera-to-gripper transformation was then uploaded to the robot via the *UPDATETOOL* program in the robot controller. After applying the updated tool data, the robot adjusted its pose so the camera could return to its original "observation" pose, as illustrated in **Figure 6-6**.

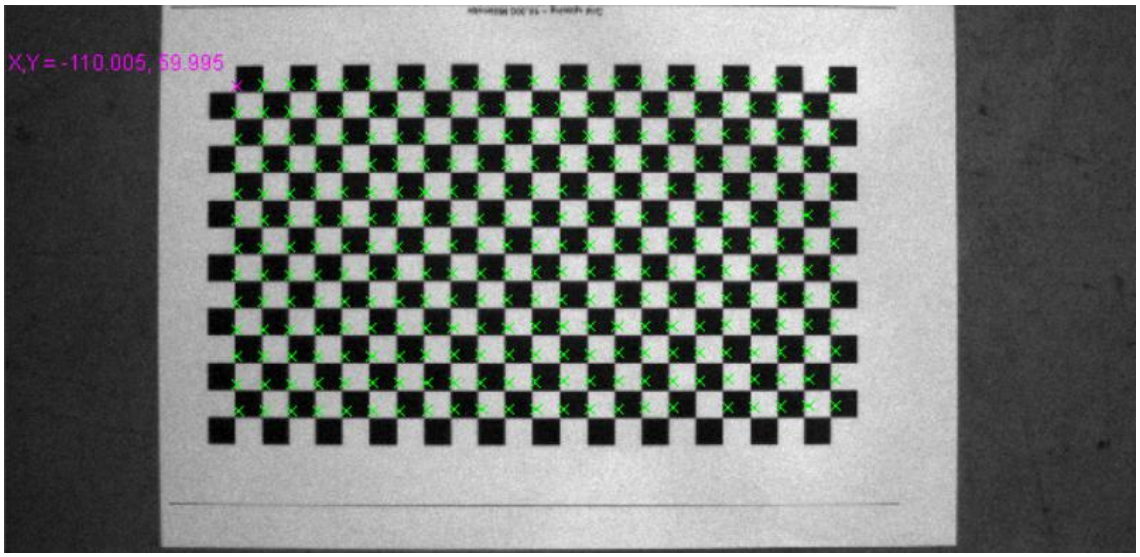

Figure 6-6 Recalibrated observation pose

With the updated calibration, the green crosses were closely aligned to the actual square intersections. This level of precision is sufficient for robotic applications involving objects with dimensions as small as 1-2 cm, confirming the effectiveness of the calibration and the reliability of the Hand Eye program system.

6.4 - Number of poses

To assess the influence of the number of poses used during the calibration process on the resulting accuracy, a systematic experiment was conducted. The experiment involved performing 20 camera-to-gripper calibrations under similar environmental and setup conditions for different number of poses. The calibration aimed to estimate the relative transformation between the camera and the robotic gripper, expressed in terms of:

- **Dist**: Euclidean distance from the camera to the gripper in mm,
- ***x***, ***y***, ***z***: translation components in 3D space, representing the position of the camera relative to the gripper in mm,
- ***w***, ***p***, ***r***: rotation angles (in degrees) representing the orientation of the camera with respect to the gripper in degrees, around the *x*, *y*, and *z* axes (commonly referred to as roll, pitch, and yaw).

The experiment was conducted using three different datasets comprising 5, 9, and 20 poses for the calibration process. The set of 5 poses corresponds to the minimum required poses as specified by Cognex [28], while the 9-pose set aligns with Cognex's recommended configuration for improved calibration accuracy. The 20-pose set was used to evaluate the impact of a higher number of calibration poses on the overall accuracy and robustness of the Hand-Eye Calibration.

For each pose set (5, 9, and 20 poses), the experiment was repeated 20 times to assess the consistency and precision of the Hand-Eye Calibration results under repeated trials.

The results of this test will be expressed in terms of the deviation observed in the Hand-Eye Calibration results. This is because the number of calibration poses primarily affects the precision of the results rather than their absolute accuracy.

These tests were conducted under consistent conditions, using the same hardware components and shop-floor lighting throughout. The set of 9 poses was constructed by extending the initial 5 mandatory poses with 4 additional ones. Similarly, the 20-pose configuration included all poses from the 9-pose set, supplemented with 11 further poses to increase the data volume for calibration.

6.4.1.1 - Calibration using 5 Poses

In the first part of the experiment, the calibration was performed using only 5 robot poses per trial. **Figure 6-7** presents the deviations in the estimated transformation parameters across 20 independent calibration trials.
The average deviation observed in this setup was approximately **5 mm** for the translation components and **0.5 degrees** for the rotation components.
The high deviations in this configuration highlights the impact of limited data on the calibration process. With fewer poses, the system becomes underconstrained, making it more susceptible to noise, pose estimation errors, and ambiguities in the spatial relationships being modeled.

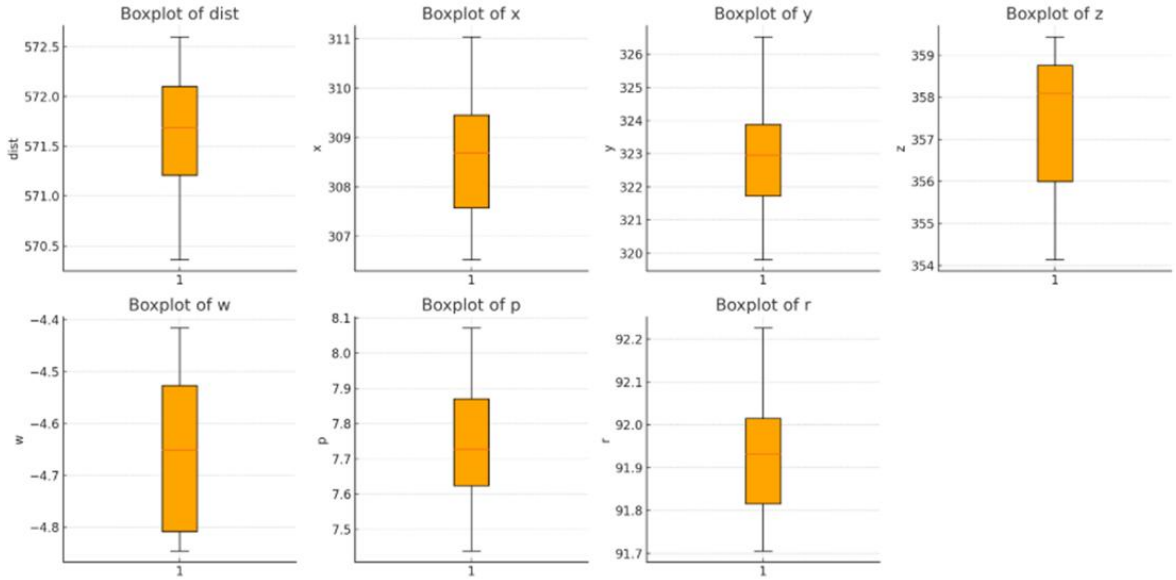

Figure 6-7 5 Poses deviation

6.4.1.2 - Calibration Using 9 Poses

In the second part of the experiment, 20 independent calibration trials were conducted, each using 9 different robot poses. The results of these calibrations are presented in **Figure 6-8**.

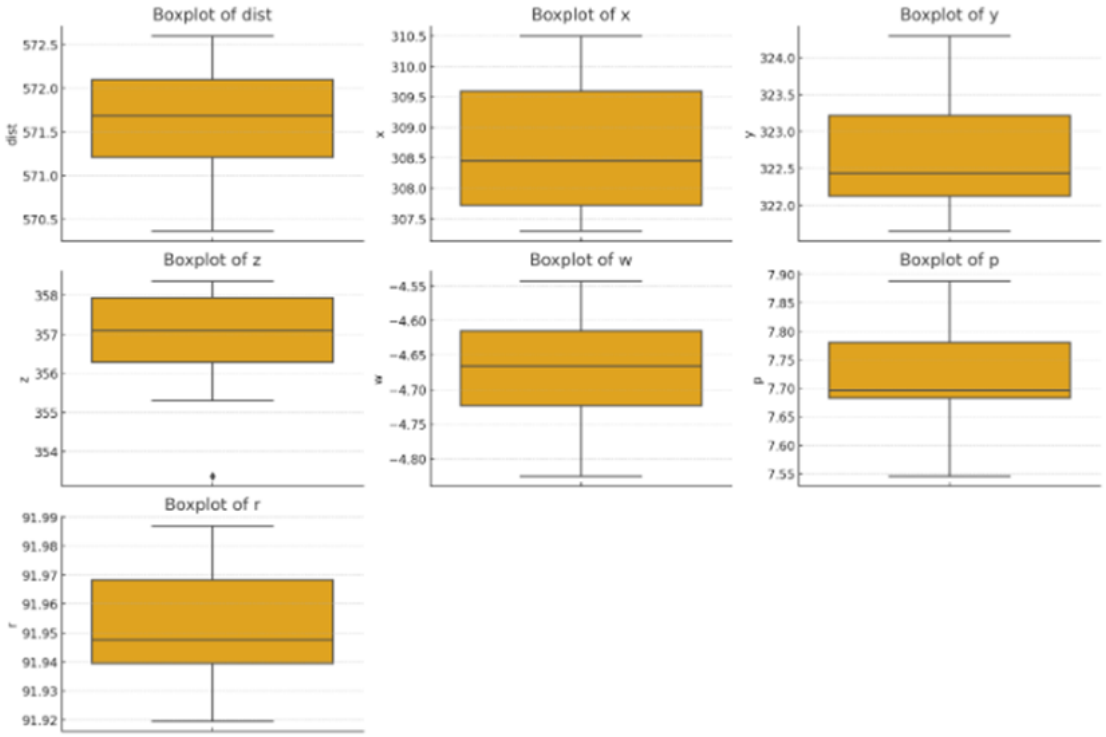

Figure 6-8 9 Poses deviation

The observed average deviation across the 20 trials was approximately **2.5 mm** in translation (i.e., the *x, y, z* components) and **0.35 degrees** in rotation (i.e., the *w, p, r* components). These deviations reflect the variability in calibration results caused by small errors in pose estimation, sensor noise, or minor variations in image data across calibration runs.

6.4.1.3 - Calibration Using 20 Poses

In the third part of the experiment, the same calibration procedure was repeated, this time using **20 robot poses** per calibration trial. As shown in **Figure 6-9**, this larger and more diverse set of poses led to a significantly improved calibration outcome. The deviation across 20 trials was reduced to approximately **1 mm** for translations and **0.1 degrees** for rotations.

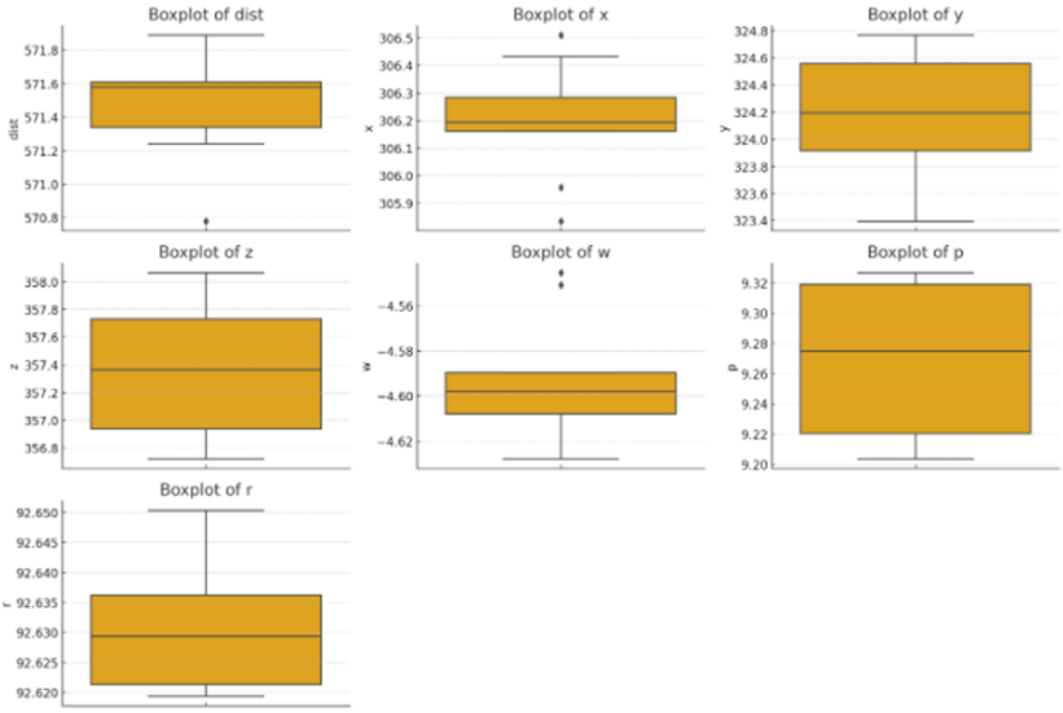

Figure 6-9 20 Poses deviation

6.4.1.4 - Conclusions

The experimental results clearly show that the number of calibration poses has a significant impact on the accuracy, repeatability, and robustness of the estimated camera-to-gripper transformation. As the number of poses increases, the deviations in both translation and rotation parameters decrease consistently, indicating improved calibration quality.

Using only 5 poses leads to the highest observed deviation, approximately 5 mm in translation and 0.5 degrees in rotation. This level of variation highlights the limitations of under-constrained calibrations, where the small dataset is insufficient to suppress the effects of sensor noise, pose estimation errors, or image inconsistencies. As a result, calibrations using 5 poses are not recommended for tasks requiring high precision.

Calibrating with 9 poses strikes a practical balance between calibration effort and accuracy. With an average deviation of 2.5 mm and 0.35 degrees, this setup offers a good trade-off between reduced calibration time and acceptable accuracy, making it suitable for many typical applications where speed and efficiency are important, but extreme precision is not critical.

For applications that demand highly accurate and stable calibration results, increasing the number of poses beyond 9 is essential. The 20-pose configuration yielded the best results, with deviations reduced to approximately 1 mm in translation and 0.1 degrees in rotation. This substantial improvement can be attributed to the following factors:

- Increased Redundancy: More poses introduce greater constraints into the optimization, making the results less sensitive to individual measurement errors.
- Enhanced Pose Diversity: A broader sampling of the robot's workspace allows better resolution of geometric relationships and minimizes ambiguity in the transformation.
- Error Mitigation: Larger datasets enable averaging of random errors and help identify systematic biases more effectively.

- Greater Numerical Stability: More data points improve the conditioning of the calibration problem, leading to more stable and accurate solutions.

In summary, while 5 poses are insufficient and lead to unreliable calibration, 9 poses provide a good compromise between speed and result quality. However, when precision is a priority, such as in fine manipulation or visual servoing tasks, using at least 20 diverse poses is strongly recommended to achieve consistent and high-quality calibration performance.

6.5 - Lighting

For the lighting experiment, a wide variety of poses (20 in total) were used to evaluate how effectively the hand-eye calibration program could identify the calibration grid, with and without the embedded lighting from the Cognex camera. The same conditions were maintained for both tests. One test relied solely on the ambient natural light available on the shop floor, while the other utilized the camera's built-in lighting.
In the first test, using only natural light, the calibration grid could not be identified in 4 of the images due to poor lighting conditions. As a result, the hand-eye calibration error ranged from 1.5 to 2 cm. The calibration result can be seen in **Figure 6-10**:

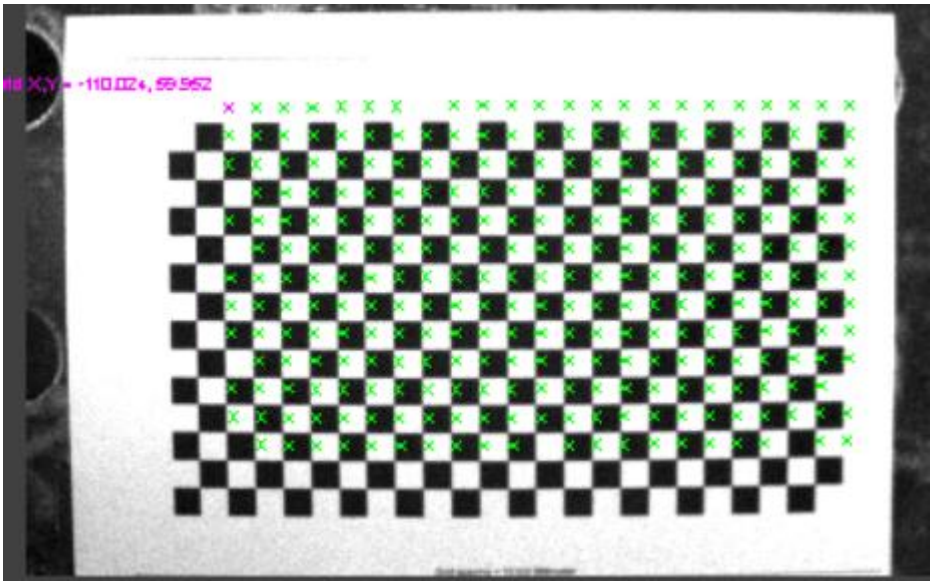

Figure 6-10 Calibration with natural lighting

In the test using the camera's built-in lighting, the calibration grid was successfully identified in all 20 poses, in contrast to the test without lighting, where the grid could not be detected in 4 poses. The hand-eye calibration error decreased to a range of 1.0 to 1.5 cm. The calibration result is seen in **Figure 6-11**:

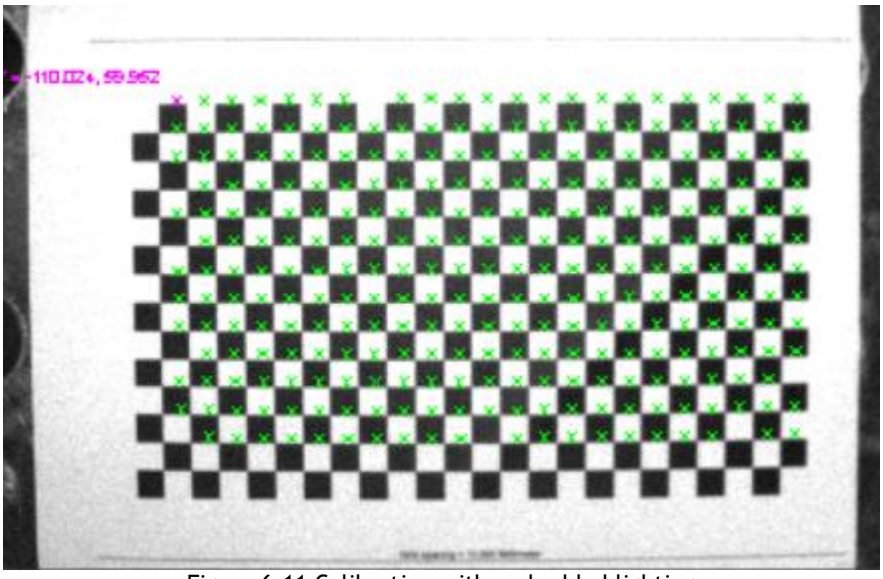

Figure 6-11 Calibration with embedded lighting

Figure 6-12 compares the reprojection errors from the two tests—with and without lighting. Although the reprojection error is primarily related to the accuracy of the camera calibration rather than the hand-eye calibration itself, a more precise estimation of the camera's extrinsic parameters contributes to improved hand-eye calibration results.

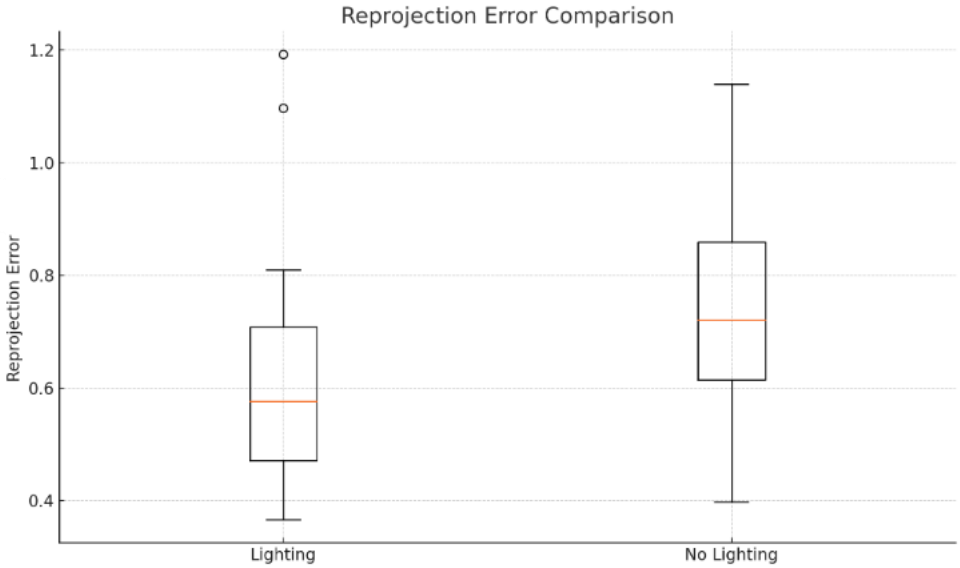

Figure 6-12 Reprojection error with different lighting

Figure 6-13 shows that lighting has a significant impact on the quality of hand-eye calibration. The results indicate that using the embedded lighting from the camera leads to more accurate and reliable calibration outcomes. However, due to material constraints, it was not possible to test additional lighting setups, such as external fixed lighting or isolating the setup from ambient shop floor lighting.

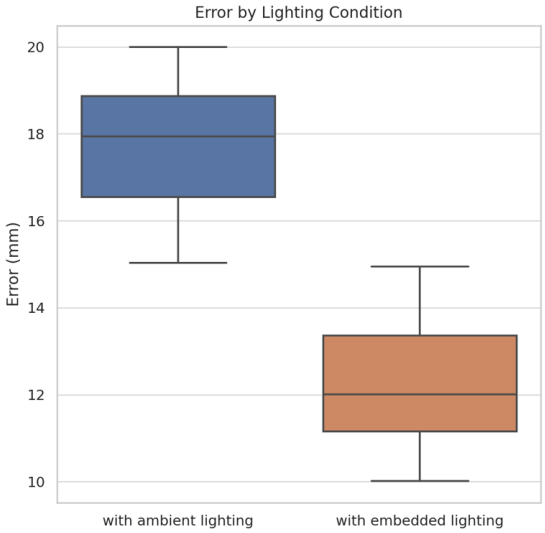

Figure 6-13 Hand-Eye Calibration error by lighting conditions

6.6 - Pose Variation

The experiments were carried out under consistent testing conditions, with the primary variable being the range of positional variations applied to the calibration grid. Three levels of variation were tested: low, medium, and high. These variations involved changes in the position and orientation of the grid relative to the camera, aiming to evaluate how different degrees of movement affect the robustness and accuracy of the hand-eye calibration process.
For the low variation scenario, poses were spaced approximately 150 mm apart. In the medium variation scenario, the spacing increased to 400 mm, while the high variation configuration used pose distances of approximately 700 mm. These variations aimed to assess how the spatial distribution of poses impacts the accuracy of the Hand-Eye Calibration.

Low variation

Low variation between poses results in insufficient spatial diversity in the collected data, which negatively impacts the accuracy of the hand-eye calibration. When the range of motion is too limited, the calibration algorithm lacks the necessary geometric information to reliably estimate the transformation between the camera and the robot. Consequently, the calibration process becomes less robust, often leading to increased errors. In **Figure 6-14**, limited pose variation led to hand-eye calibration errors in the range of 1.0 to 1.5 cm.

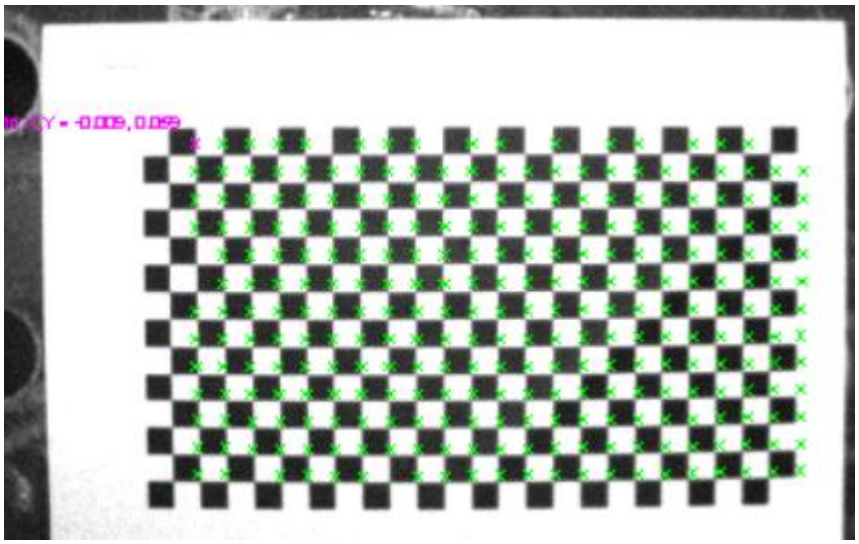

Figure 6-14 Calibration with low variation poses

Medium variation

With a medium level of pose variation, the hand-eye calibration can be performed effectively. This level of variation provides sufficient spatial diversity for accurate transformation estimation, while still maintaining image quality that allows the calibration grid to be consistently detected by the system. As a result, this approach offers a good balance between data richness and detection reliability, leading to improved calibration performance. In **Figure 6-15**, the hand-eye calibration error was reduced to 7–8 mm.

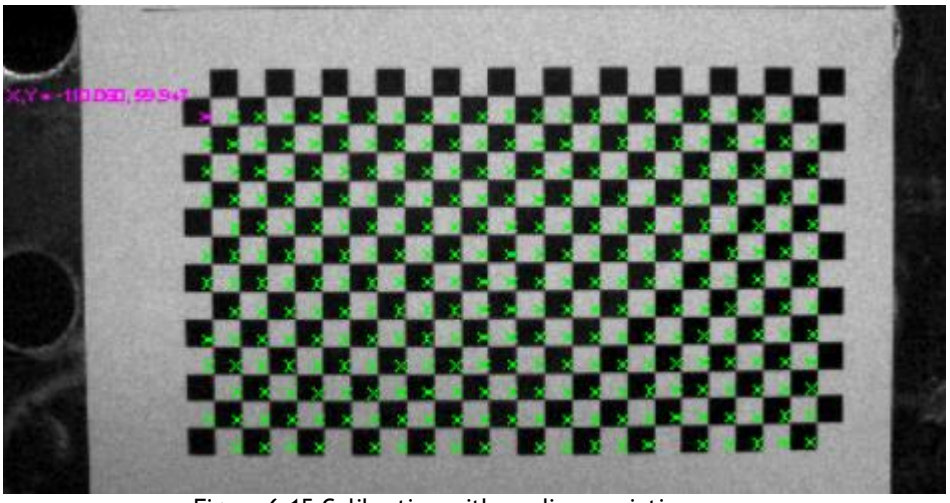

Figure 6-15 Calibration with medium variation poses

High Variation

Although hand-eye calibration tends to be more accurate when there is greater variation in the poses, excessive variation can negatively impact image quality. Large positional changes can make it difficult to capture high-quality images, especially when all images are acquired using the same focus and exposure settings. This degradation in image quality can, in turn, affect

the accuracy of the camera calibration, ultimately limiting the precision of the hand-eye calibration. In **Figure 6-16**, the calibration error ranged from 1.0 to 1.5 cm.

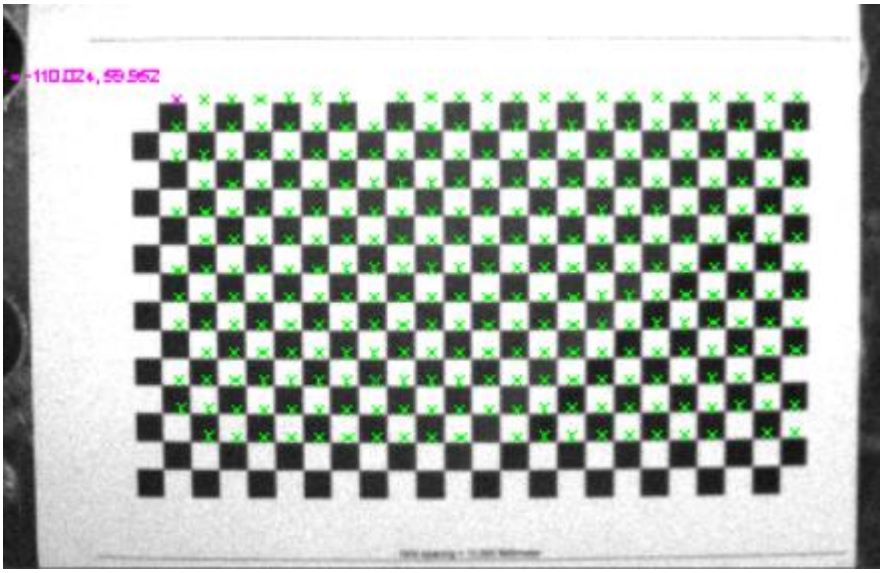

Figure 6-16 Calibration with high variation poses

Figure 6-17 illustrates the impact of pose variation on the hand-eye calibration error:

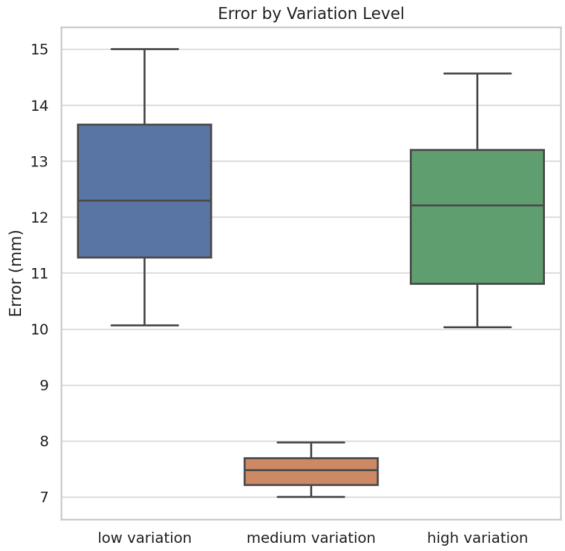

Figure 6-17 Hand Eye calibration error by variation level

The analysis of reprojection errors across different pose variation levels shows that medium variation consistently yields the lowest and most stable errors, indicating higher calibration accuracy. Low variation results in moderately higher errors due to insufficient spatial diversity, while high variation introduces greater error variability and higher average values, likely due to compromised image quality from extreme angles. The results in **Figure 6-18** suggest that medium pose variation offers the most favorable conditions for minimizing reprojection error in camera calibration processes.

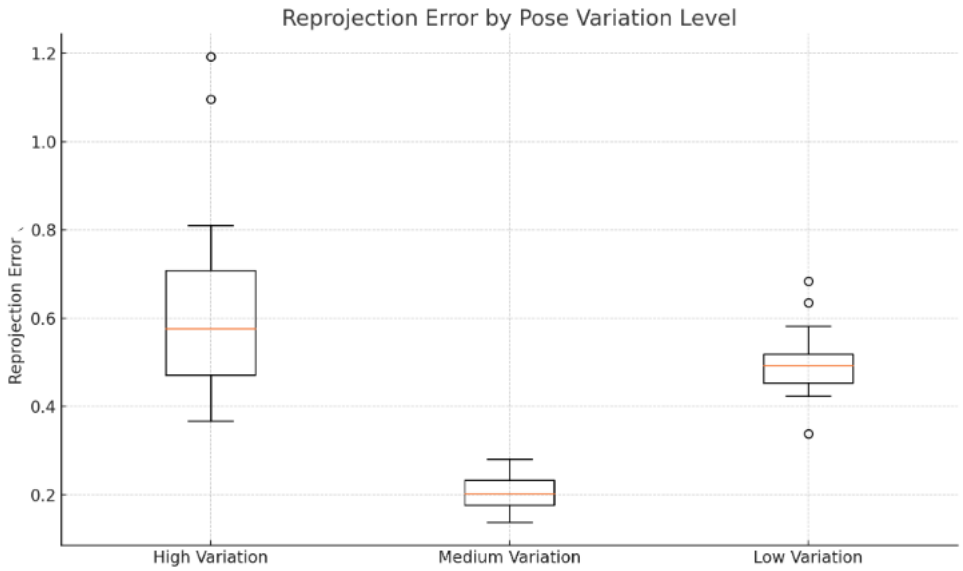

Figure 6-18 Reprojection Error by Pose Variation

6.7 - Background Effect

The background, referring to the surface on which the calibration grid is placed, plays a significant role in the accuracy of hand-eye calibration. Variations in the background, such as inconsistent textures or visual noise, can introduce errors by affecting the detection and recognition of the calibration pattern. A cluttered or non-uniform background can cause the vision system to misidentify grid points or reduce the precision of feature extraction, ultimately degrading the calibration quality. Therefore, ensuring a clean, high-contrast, and uniform background is critical to minimizing errors and improving the robustness and reliability of hand-eye calibration results.

In the case of a problematic background, such as the one shown in the figure below, reflective surfaces can cause light to bounce toward the camera, leading to glare or unwanted reflections. This can negatively affect the camera's focus and image quality, making it more difficult to accurately identify the calibration points. Additionally, these reflections and light distortions may confuse the feature detection algorithms, resulting in errors during the calibration process.
In **Figure 6-19**, the calibration error ranged between 1 and 1.2 cm, highlighting the significant impact a poor background can have on accuracy.

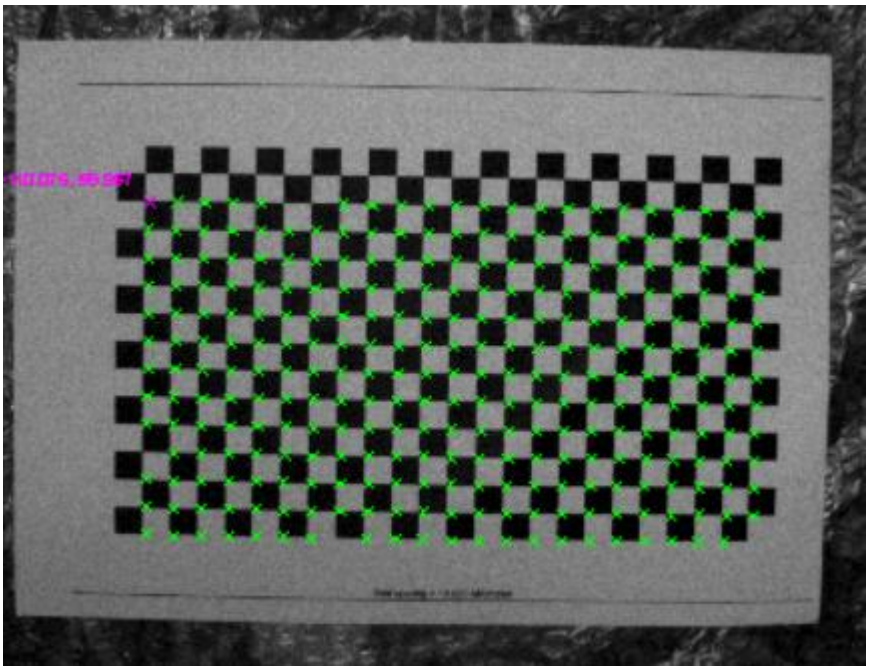

Figure 6-19 Calibration with a bad background

In contrast, a good background—simple, free of visual noise, and without reflections—can significantly reduce hand-eye calibration errors. For example, in the **Figure 6-20**, the calibration error was reduced to between 1 and 3 mm, demonstrating how a clean background improves the accuracy and reliability of the calibration process.

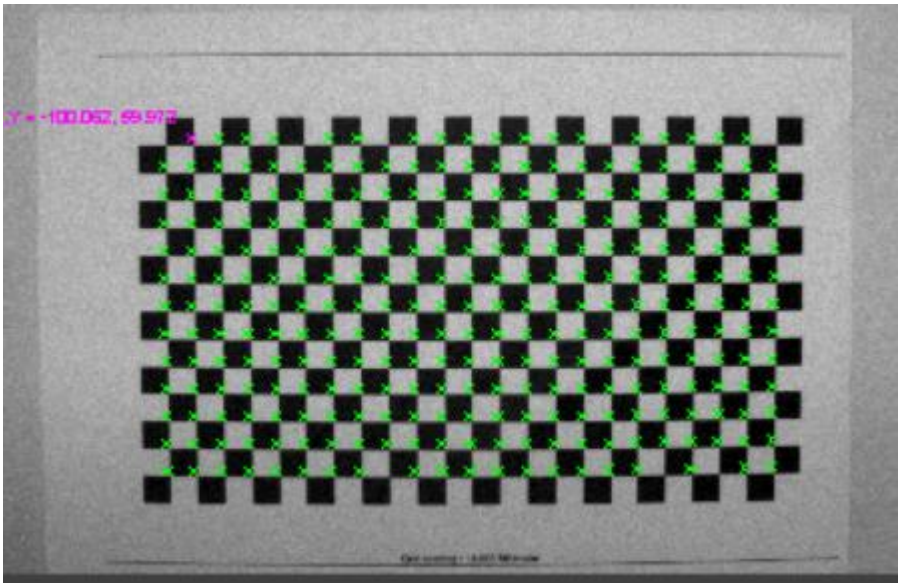

Figure 6-20 Calibration with a good Background

Figure 6-21 illustrates the impact of the background on the hand-eye calibration error:

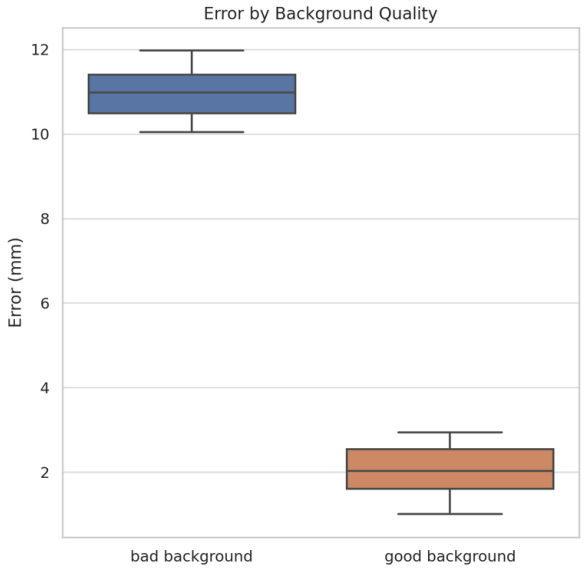

Figure 6-21 Hand Eye calibration error by background quality

6.8 - Conclusions

The experimental results presented in this chapter highlight the critical factors that impact the performance and accuracy of hand-eye calibration. Low reprojection errors (as low as 0.21 pixels) indicate a high-quality calibration setup.

A key finding is the strong correlation between the number of calibration poses and result consistency. While five poses proved insufficient for reliable calibration, nine poses provided a practical compromise between efficiency and accuracy. Twenty poses, however, yielded the highest precision and lowest variance. Additional tests confirmed that moderate pose variation, embedded lighting, and a clean background further enhance calibration robustness and precision.

Collectively, these findings provide a solid foundation for optimizing hand-eye calibration protocols in vision-guided robotic systems, especially in applications where accuracy, repeatability, and environmental adaptability are paramount.

Chapter 7

Conclusions

This chapter evaluates the performance and practical impact of the automated Hand-Eye calibration system developed during this dissertation. The discussion begins by outlining the key contributions of the solution, particularly its modular design and adaptability to different hardware configurations. It then compares the proposed method with the manual calibration procedure previously used, highlighting improvements in precision, flexibility, and ease of use. In addition, this chapter identifies the system's current limitations, including environmental dependencies and hardware constraints, which may affect its accuracy or scalability in certain industrial settings. Lastly, potential directions for future work are proposed, aimed at improving robustness and facilitating broader deployment. Together, these elements provide a comprehensive analysis of the system's capabilities and its relevance to real-world robotic applications.

7.1 - Contribuitions

Although the estimation of the Hand-Eye transformation parameters relied on existing functions from the OpenCV library—thus not contributing to the methodology in terms of the underlying mathematical computation—the proposed system demonstrates innovation through its scalable and adaptable architecture. The developed solution for the Hand-Eye calibration process was implemented using a Cognex camera and a Fanuc industrial robot, however, its design is hardware-agnostic, allowing for straightforward adaptation to various types of cameras and robotic platforms. This flexibility makes it particularly suitable for deployment in dynamic industrial environments, such as those encountered in JPM Industry, where integration with diverse robotic arms and vision systems is often required. By abstracting the calibration pipeline and ensuring modularity in the software implementation, the system supports broader interoperability and reusability, which are critical aspects in scalable industrial automation solutions.

7.2 - Comparison to manual calibration

Before the implementation of the automated Hand-Eye calibration process, the calibration of the camera in the system was performed manually. This manual approach relied heavily on mechanical precision and human estimation and was only viable due to the specific constraints and assumptions that applied to the initial setup.

In the manual method, the camera was mounted on a custom-designed mechanical support that rigidly fixed its position and orientation relative to the robot gripper. The geometry of this support was known in advance, particularly the Z-offset (distance from the gripper to the camera along the tool's Z-axis), and the rotational alignment of the camera was defined as being zero in all three axes (0°, 0°, 0°). These assumptions simplified the transformation from the robot's Tool Center Point (TCP) to the camera frame, allowing the camera's pose to be modeled with fixed, known values. As a result, calibration could be done by manually rotating and translating the robot so that the camera aligned visually with a calibration grid, and known reference points could be matched to the grid origin.

However, this approach had significant drawbacks. First, it was highly rigid and not resilient to change. Any minor displacement or rotation of the camera—whether due to vibration, robot movement, thermal expansion, or mechanical wear—would invalidate the original calibration. Since the system relied on fixed geometric assumptions rather than actual measurement, it lacked the flexibility to adapt to environmental changes. As a result, recalibration required a manual process that was time-consuming, error-prone, and reliant on skilled operators.

Second, the manual alignment process itself introduced potential inaccuracies. The robot had to be manually moved so that certain known points in the camera image aligned with features on the calibration board, and this required visual estimation by the operator. Even small human errors in positioning could propagate into the camera's extrinsic parameters, reducing the overall precision of the system.

The new calibration method, based on the ***Hand_Eye_Auto*** program, overcomes these limitations by providing a fully automated and adaptive calibration process. Instead of assuming fixed mechanical parameters, the system computes the complete camera-to-gripper transformation (*cam2gripper*) through a sequence of controlled robot movements and image captures. This allows the calibration to be established from scratch, without prior knowledge of the camera's mounting orientation or distance. The system estimates both translation and rotation, enabling it to operate correctly even if the camera is mounted with a non-zero or unknown orientation relative to the gripper.

Most importantly, the automated approach allows the system to recalibrate dynamically in case of a shift in the camera's position. For example, if the camera is accidentally bumped or its mount is modified, the system can re-run the calibration routine and update the TCP of the camera in the robot's controller automatically. This adaptability is critical in real-world industrial environments, where mechanical shifts are inevitable over time.

Additionally, because the process does not rely on assumptions of mechanical rigidity, it can be applied to different robots and camera setups without redesigning custom brackets or pre-measuring offsets. It also reduces operator workload and training, since the calibration can be performed using a standard routine rather than relying on manual alignment and estimation.

In summary, while the manual calibration process offered a basic level of functionality under tightly controlled conditions, it lacked robustness, scalability, and precision in dynamic industrial environments. The automated calibration developed in this dissertation represents a

significant step forward in terms of flexibility, reliability, and usability, making it more suitable for long-term deployment in real-world robotic applications.

7.3 - Limitations

While the Hand Eye calibration system demonstrated high accuracy and reliability, several limitations were identified during its implementation and testing that should be considered for both future development and industrial deployment.

Visibility of the Calibration Grid:
A fundamental requirement of the calibration process is that the full calibration pattern (checkerboard) must be clearly visible in every image acquired. If the pattern is partially occluded, cropped, or not detected in the image, the calibration algorithm fails or returns inaccurate results. This imposes strict constraints on the positioning of the robot and camera, especially in constrained or cluttered environments.

Dependence on External PC Processing:
The calibration and interpolation routines used by the ***Hand_Eye_Auto*** system require computational resources that are not suitable for a typical industrial Programmable Logic Controller (PLC). As a result, the entire calibration pipeline must be executed on an external PC, which introduces dependencies on third-party hardware and software and may not be ideal for fully embedded, real-time industrial applications.

Trade-off Between Precision and Field of View:
Images captured from a closer distance to the calibration board yield higher reprojection accuracy due to the increased pixel coverage of the pattern. However, this increases the sensitivity of the system to positional shifts: even small changes in the camera's pose can cause the calibration board to move out of the field of view, invalidating the process. This creates a trade-off between calibration precision and tolerance to positional variation.

Dependency on In-Sight Explorer for Image Capture:
To capture images during the calibration process, the *In-Sight Explorer* software must run in parallel with the calibration system. This software is responsible for acquiring and storing images on the PC, meaning that the process cannot be decoupled from Cognex's proprietary ecosystem. It also introduces an additional point of failure and dependency that must be maintained throughout the calibration procedure.

Pose Adjustments Due to Environmental Constraints:
In real-world industrial setups, physical obstacles such as machine components, fixtures, or safety barriers can obstruct ideal robot poses for calibration. As a result, some of the planned poses may need to be manually adjusted to avoid collisions or visibility issues. These changes can impact the uniformity and spatial distribution of calibration data, potentially reducing calibration accuracy if the new poses do not sufficiently cover the camera's full field of view.

7.4 - Future work

Although the results obtained from the hand-eye calibration process demonstrated high accuracy—achieving errors within the range of 1 to 3 mm—the system did not consistently meet the stringent requirement of maintaining an error of less than 2 mm at every calibration point. This limitation is likely attributed to a combination of environmental lighting conditions on the shop floor and hardware constraints, particularly the quality of the camera used during the calibration process.

Ambient lighting, especially when uncontrolled or variable, can introduce light noise, reflections, and inconsistent illumination across different calibration images. These issues can negatively impact the visibility and detection accuracy of the calibration grid, ultimately affecting the calibration precision. Similarly, limitations in camera resolution or lens quality may hinder the system's ability to detect feature points with the granularity required for sub-millimeter accuracy.

To address these limitations, future work should focus on testing the hand-eye calibration program under optimized conditions. This includes using a higher-quality industrial camera with improved resolution and optics, as well as implementing a controlled lighting environment. One proposed solution is the construction of an enclosed calibration booth or light-isolation box, within which the robot can perform calibration routines free from external light interference. By standardizing the lighting conditions, the system could achieve more stable and accurate point detection, thus reducing calibration errors.

Additionally, further development of the hand-eye calibration algorithm should aim to increase its robustness under less-than-ideal conditions. This may involve integrating advanced image preprocessing techniques, adaptive exposure control, or even machine learning-based feature detection, all of which could enhance performance without requiring significant hardware modifications.

Finally, a natural progression of this work would be to deploy and validate the algorithm in a real-world industrial application. By integrating the calibrated vision system into an actual robotic solution—such as bin picking, precision assembly, or quality inspection tasks—the practical effectiveness of the calibration process can be fully assessed. Such implementation would not only verify the algorithm's robustness but also highlight any additional improvements needed for operational deployment in dynamic and demanding environments.

7.5 - Conclusion

This dissertation presented the development and evaluation of an automated Hand-Eye calibration system designed to enhance precision, flexibility, and usability in industrial robotic applications. The system's modular and hardware-agnostic architecture enables easy adaptation to diverse cameras and robots, addressing the limitations of traditional manual calibration methods that rely on fixed mechanical assumptions and operator skill. The

automated approach significantly improves robustness by allowing dynamic recalibration in response to mechanical shifts, reducing downtime and operator workload.

Despite its demonstrated accuracy and practical benefits, the system faces challenges related to environmental dependencies, such as strict visibility requirements for the calibration grid, reliance on external PC processing, and constraints imposed by industrial settings. These factors highlight the need for further refinement to enhance system robustness and independence.

Future work should focus on optimizing hardware and environmental conditions, such as using higher-quality cameras and controlled lighting setups, alongside algorithmic improvements including advanced image processing and machine learning techniques. Ultimately, deploying the system in real industrial tasks will provide critical insights into its operational effectiveness and pave the way for broader adoption in scalable robotic automation solutions.

Together, these findings underscore the system's potential to transform Hand-Eye calibration from a labor-intensive, error-prone task into a reliable, flexible, and user-friendly process suited for dynamic industrial environments.

References

[1] Z. Zhang, "A Flexible New Technique for Camera Calibration," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, pp. 1330–1334, 2000.

[2] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, 2nd ed. Cambridge University Press, 2003.

[3] 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, vol. 5, no. 3, pp. 345–358, 1989.

[4] K. Daniilidis, "Hand-Eye Calibration Using Dual Quaternions," The International Journal of Robotics Research, vol. 18, no. 3, pp. 286–298, 1999.

[5] G. H. Golub and C. F. Van Loan, Matrix Computations, 3rd ed. Johns Hopkins University Press, 1996.

[6] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 9, no. 5, pp. 698–700, 1987.

[7] M. I. A. Lourakis and A. A. Argyros, "SBA: A Software Package for Generic Sparse Bundle Adjustment," ACM Transactions on Mathematical Software, vol. 36, no. 1, pp. 1–30, 2009.

[8] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment – A Modern Synthesis," Proceedings of the International Workshop on Vision Algorithms, pp. 298–372, 2000.

[9] J. J. Craig, Introduction to Robotics: Mechanics and Control. Pearson, 2005.

[10] B. Siciliano and O. Khatib, Springer Handbook of Robotics. Springer, 2016.

[11] R. P. Paul, Robot Manipulators: Mathematics, Programming, and Control. MIT Press, 1981.

[12] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, Robotics: Modelling, Planning and Control. Springer, 2010.

[13] M. L. Ribeiro, Vision Calibration for Industrial Robots: An Approach using Cognex VisionPro, Master's thesis, Instituto Superior Técnico, Lisbon, Portugal, 2017.

[14] Z. Roth, B. Mooring, and B. Ravani. An overview of robot calibration. IEEE Journal on Robotics and Automation, 3(5):377–385, 1987.

[15] M. Antonello, A. Gobbi, S. Michieletto, S. Ghidoni and E. Menegatti, "A fully automatic hand-eye calibration system," 2017 European Conference on Mobile Robots (ECMR), Paris, France, 2017.

[16] Zijian Zhao and Yuncai Liu, "Integrating camera calibration and hand-eye calibration into robot vision," 2008 7th World Congress on Intelligent Control and Automation, Chongqing, China, 2008.
[17] O. Kroeger, J. Huegle and C. A. Niebuhr, "An automatic calibration approach for a multi-camera-robot system," 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 2019.
[18] L. Villaverde, D. Maneetham and T. Rabgyal, "Camera Calibration Algorithm for Industrial Robot," 2022 IEEE 8th Information Technology International Seminar (ITIS), Surabaya, Indonesia, 2022.
[19] R. Han, T. Dunker, R. Xu, E. Trostmann and Z. Xu, "Research on New Mathematical Model Aix=Ybi for Hand-to-Eye Calibration and Application Scenery of This Model Based on iGPS," in IEEE Sensors Journal, vol. 24, no. 23, pp. 39677-39689, 1 Dec.1, 2024.
[20] I. Enebuse, B. K. K. Ibrahim, M. Foo, R. S. Matharu and H. Ahmed, "Accuracy Assessment of Hand-Eye Calibration Techniques in Uncertain Environments for Vision Guided Robots," 2023 IEEE International Conference on Mechatronics (ICM), Loughborough, United Kingdom, 2023.
[21] MathWorks, "Estimate pose of moving camera mounted on a robot." [Online]. Available: https://www.mathworks.com/help/vision/ug/estimate-pose-of-moving-camera-mounted-on-a-robot.html.
[22] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," IEEE Transactions on Robotics and Automation, vol. 12, no. 5, pp. 651-670, Oct. 1996.
[23] J. Denavit e R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," Journal of Applied Mechanics, vol. 22, pp. 215–221, 1955.
[24] FANUC Corporation, RoboGuide Simulation Software. FANUC Corp. [Software]. Available: https://www.fanuc.eu
[25] Cognex Corporation, In-Sight Explorer Software. Cognex Corp., Natick, MA, USA. [Software]. Available: https://www.cognex.com
[26] Cognex Corporation, Radial and Perspective Transformations, VisionPro Users Guide. [Online].
Available:https://support.cognex.com/docs/vpromx_1000/web/en/visionpro/Content/Topics/users-guide/vision-tools-guide/fixtures-calibration/coordinate-spaces-radial-distortion.htm
[27]T. Myhre, *Robotic Camera Calibration – An Overview*, Torstein Myhre's Webpage. [Online]. Available: https://www.torsteinmyhre.name/snippets/robcam_calibration.html
[28] Cognex Corporation, *VisionPro 3D Guide*, Rev. 9.0, Cognex Support, [Online]. Available: https://support.cognex.com/docs/cvl_900/support/EN/3d_guide.pdf
[29] FANUC Europe Corporation, "ARC Mate 120iD/35 - For narrow and difficult spaces," FANUC, [Online]. Available: https://www.fanuc.eu/eu-en/product/robot/arc-mate-120id35.
[30] Cognex Corporation, "Sistemas de Visão In-Sight 7000," *Cognex*, [Online]. Available: https://www.cognex.com/pt-pt/products/machine-vision/2d-machine-vision-systems/in-sight-7000-series.
[31] Siemens AG, "SIMATIC S7-1500, CPU 1517-3 PN," *Industry Mall - Siemens Portugal*, [Online]. Available: https://mall.industry.siemens.com/mall/pt/pt/Catalog/Product/6ES7517-3AQ10-0AB0.
[32] FANUC Corporation, *R-30iB Plus Controller Product Overview*, FANUC, 2023. [Online]. Available: https://www.fanuc.eu/~/media/files/pdf/products/robots/brochures/mbr-00119-ro-product-overview/v-23/robot-brochure-en.pdf

[33] FANUC Corporation, "iPendant Touch," FANUC Europe, 2023. [Online]. Available: https://www.fanuc.eu/eu-en/accessory/controller/pendant-touch
[34] Siemens AG, *Totally Integrated Automation (TIA) Portal V18*, Siemens AG, 2022. [Software]. Available: https://new.siemens.com
[35] FANUC Corporation, *ROBOGUIDE Simulation Software*, FANUC Corp., 2022. [Software]. Available: https://www.fanucamerica.com
[36] PROFIBUS & PROFINET International (PI), *PROFINET – The Standard for Industrial Ethernet*, PI International, Karlsruhe, Germany. [Online]. Available: https://www.profibus.com
[37] Schneider Electric, *Modbus Messaging on TCP/IP Implementation Guide v1.0b*, Schneider Electric, North Andover, MA, USA, 2006. [Online]. Available: https://modbus.org
[38] OPC Foundation, *OPC Unified Architecture Specification*, OPC Foundation, Scottsdale, AZ, USA, 2021. [Online]. Available: https://opcfoundation.org
[39] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.
[40] OpenCV Team, "Hand-Eye Calibration," *OpenCV Documentation*, [Online]. Available: https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga84aa1241e3eace3eadaf7d5c209b2e6b.