الجمهورية الجزائرية الديمقراطية الشعبية

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة محمد البشير الابراهيمي – برج بوعريريج

Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculté des Sciences et de la Technologie

Département Electromécanique

MÉMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Automatique

Spécialité : Automatique et Informatique Industrielle

Par:

- Bellache Mohamed Said
- Nacef Moncef

Sujet

Optimisation et déploiement de modèles d'intelligence artificielle embarqués sur Raspberry Pi pour des applications en temps réel

Soutenu publiquement, le / /, devant le jury composé de :

Mr.Bensidhoum Tark	MAA	**Univ-BBA**	Président.
Mr.Zaoui Fares	MCB	**Univ-BBA**	Examinateur.
Mr.Khenfer Riad	MCA	**Univ-BBA**	Encadrant.

Année Universitaire 2024/2025

Dedication

My heartfelt appreciation goes to everyone who stood by me through every phase of this journey.
I express my gratitude to those who showed me how patience, determination and faith should be understood.
I dedicate this work to my father and mother who have been both my crown and source of warmth.
My simple work comes from my profound thankfulness for your endless love and dedication.

To my silent companion,
With innocent eyes and gentle presence,
My beloved cat, Shiro…
I appreciate how your calm presence and silence provides me with peace and comfort.

And to my wonderful friends,
You who reduced my daily burdens and made my journey memorable through laughter that will always stay with me.
You were true companions and loyal support—
This journey was enriched by your beautiful presence and I am deeply thankful for it.

Mohamed Said

Dedication

I dedicate this work to everyone who planted the seeds of ambition within me and carefully cultivated them with love and patience.
To my dear parents who stand as my strength and inspiration.
Through every stage of my journey my siblings gave me support and guidance.
This work belongs to my friends who joined me on our shared path of laborious nights and joyful experiences.
I express my appreciation to those who believed in me and offered continuous support and encouragement throughout my journey.

This work represents years of persistent effort and accumulated knowledge,
With all my love and gratitude.

Moncef

Abstract

This project aims to develop an intelligent system based on a robotic arm capable of recognizing various geometric shapes (such as circles, squares, and triangles) through an integrated camera, and then automatically pointing to the desired shape based on its color and geometric features. This system serves as a simplified model of computer vision applications in automated control, combining image processing with motion-based decision-making in an interactive environment.

The system is implemented using a Raspberry Pi 4 as the central processing unit, with all functionalities programmed in Python. The OpenCV library is used to process the images captured by the camera, extracting geometric shapes and analyzing color characteristics using the HSV color space. This is followed by edge detection and shape recognition using contour analysis algorithms. Based on the target shape, commands are sent to the robotic arm to move its joints via stepper motors toward the shape's position, enabling precise and realistic pointing motion.

Keywords: Computer vision, Geometric shape recognition, Image processing, HSV color space, Robotic arm, Raspberry Pi 4, OpenCV, Stepper motors.

ملخص

يهدف هذا المشروع إلى تطوير نظام ذكي يعتمد على ذراع آلية قادرة على التعرف على الأشكال الهندسية المختلفة (مثل: الدائرة، المربع، والمثلث) عبر كاميرا مدمجة، ومن ثم الإشارة إلى الشكل المطلوب تلقائيًا بناءً على خصائصه اللونية والشكلية. يُعد هذا النظام نموذجًا مبسطًا لتطبيقات الرؤية الحاسوبية في التحكم الآلي، ويجمع بين المعالجة الصورية واتخاذ القرار الحركي في بيئة تفاعلية.

تم تنفيذ النظام باستخدام وحدة Raspberry Pi 4 كوحدة معالجة مركزية، مع برمجة جميع الوظائف بلغة Python تُستخدم مكتبة Open CV لمعالجة الصور الواردة من الكاميرا، حيث يتم استخلاص الأشكال الهندسية وتحليل خصائص اللون باستخدام الفضاء اللوني HSV، تليها عمليات كشف الحواف والتعرف على الأشكال عبر خوارزميات تحليل الكنتور (Contours) بناءً على الشكل المطلوب، تصدر أوامر إلى الذراع الآلية لتحريك مفاصلها عبر محركات الخطوة نحو موقع الشكل المستهدف، لتنفيذ حركة إشارة دقيقة وواقعية.

الكلمات المفتاحية: رؤية حاسوبية، التعرف على الأشكال الهندسية، تحليل الصور، HSV، الذراع الآلية، Raspberry Pi 4، OpenCV، محركات خطوة، اتخاذ القرار الحركي.

Résumé :

Ce projet vise à développer un système intelligent basé sur une main robotique capable de reconnaître différentes formes géométriques (comme le cercle, le carré et le triangle) à l'aide d'une caméra intégrée, puis de pointer automatiquement vers la forme souhaitée en se basant sur ses caractéristiques de couleur et de forme. Ce système constitue un modèle simplifié des applications de la vision par ordinateur dans le contrôle automatisé, en combinant le traitement d'image et la prise de décision motrice dans un environnement interactif.

Le système est mis en œuvre à l'aide d'un Raspberry Pi 4 en tant qu'unité centrale de traitement, avec toutes les fonctionnalités programmées en Python. La bibliothèque Open CV est utilisée pour traiter les images capturées par la caméra, en extrayant les formes géométriques et en analysant les caractéristiques de couleur à l'aide de l'espace colorimétrique HSV. Ensuite, la détection des bords et la reconnaissance des formes sont effectuées à l'aide d'algorithmes d'analyse de contours. En fonction de la forme ciblée, des commandes sont envoyées à la main robotique pour déplacer ses articulations via des moteurs pas à pas vers la position de la forme, permettant ainsi un geste de pointage précis et réaliste.

Mots clés : Vision par ordinateur, Reconnaissance de formes géométriques, Traitement d'image, Espace colorimétrique HSV, Main robotique, Raspberry Pi 4, OpenCV, Moteurs pas à pas.

Summary

Dedication I
Abstract III
Summary V
General Introduction 1
CHAPTER I: Embedded Industrial Artificial Intelligence 3
1. Introduction: 3
2. Definition of Artificial Intelligence (AI): 3
3. The Different forms of Artificial Intelligence: ANI, AGI, and ASI: 3
3.1 Artificial Narrow Intelligence (ANI): 4
3.2 Artificial General Intelligence (AGI): 4
3.3 Artificial Super intelligence (ASI): 5
4. Industrial Artificial Intelligence 5
5. Embedded system 6
6. Integration of AI and Cameras in Factories 6
7. Robotic arm Movement Using Artificial Intelligence 7
8. Artificial Intelligence Programming Languages for Industrial Environment Programming 8
8.1. Python 9
8.2 C/C++ 9
8.3. Java 9
8.4. MATLAB 9
8.5. Rust and Go (Emerging) 9
9. Conclusion 10
CHAPTER II: System Architecture (Hardware and Software Components) 11
1. Intoduction 11
2. Robotic arm BCN3D Moveo (modify) 11
3. Motors 12
3.1 NEMA 17 STP-42D221-03 12
3.2 NEMA 23 JK57HS76-2804 13
3.3 NEMA 17 JK42HS48-1684 13
4. Electronic cards used in arm control 14
4.1. Control Unit (Raspberry Pi 4 Model B) 14
4.2. DRV8825 17

5. Software Environment and Tools Used 18
5.1. Operating System (Raspberry Pi OS) 18
5.2. Programming Language (Python and Integration with a Raspberry PI) 18
5.3. Thonny IDE on Raspberry Pi 19
5.4. Software Libraries Utilized in the Project 20
6. Conclusion 21
CHAPTER III: Application and Optimization of a Robotic arm 22
1. Introduction 22
2. Mechanical Part of the Arm 22
2.1. Arm Preparation 22
2.2. Integration of Stepper Motors 23
2.3. Mounting on the Wooden Platform and Defining the Workspace 23
3. Raspberry Pi 4 Model B (Getting Started) 23
3.1. Installing Raspberry Pi OS on microSD 23
3.2. Initial Setup for Raspberry Pi 4 24
3.3. Configuring Raspberry Pi 4 for Python Project Execution 25
3.4. Installing Required Python Libraries 26
4. Camera Calibration and Perspective Mapping 27
5. Controlling NEMA Stepper Motors via Raspberry Pi Using DRV8825 Modules 28
5.1. DRV8825 Module (Wiring and Functionality) 28
5.2. Power Supply Configuration 29
6. Experimental Step-to-Angle Calibration 31
7. Geometric Shape Recognition with OpenCV 32
8. Color Detection and Integration with Shape Recognition 32
9. Converting Camera Coordinates to Real-World Coordinates Using Homography 34
10. Converting Real-World Coordinates to Robotic Arm Angles (Inverse Kinematics) 34
10.1. Mechanical Protection of The Arm Within the Code 35
11. Converting Angles to Steps and Controlling Stepper Motors 36
12. Coordinated Motion Loop and System Reset 36
13. Conclusion 38
General conclusion 39

List of figures

Figure I-1 The Three Stages of Artificial Intelligence: ANI, AGI, and ASI [2]
Figure I-2 AI-Powered Visual Inspection on Production Lines [5]
Figure I-3 Real-Time PPE (Personal Protective Equipment) Detection Using Computer Vision [3]
Figure I-4 AI-Driven Camera System for Monitoring Robotic Arm Activity [7]
Figure II-1 Robotic arm BCN3D Moveo [13]
Figure II-2 NEMA 17 STP-42D221-03 and its diagrame
Figure II-3 NEMA 23 JK57HS76-2804 and its diagrame
Figure II-4 NEMA 17 JK42HS48-1684 and its diagrame
Figure II-5 Raspberry Pi 4 Model B 8GB
Figure II-6 Labeled Diagram of Raspberry Pi 4 Model B (8GB RAM Version) [16]
Figure II-7 DRV8825
Figure II-8 LOGO Python
Figure II-9 LOGO Thonny python IDE
Figure II-10 LOGO OpenCv
Figure III-1 Arm Preparation
Figure III-2 Raspberry Pi imager
Figure III-3 desktop interface
Figure III-4 sudo apt -get update
Figure III-5 sudo apt-get upgrade
Figure III-6 sudo apt-get install python3-venv
Figure III-7 cd /project-folder
Figure III-8 python3 -m venv myenv
Figure III-9 source myenv/bin/activate
Figure III-10 pip3 install opencv
Figure III-11 Pip3 install RPi.GPIO
Figure III-12 pip3 install numpy
Figure III-13 v4l2-ctl --list-formats-ext
Figure III-14 Python code to operate the camera
Figure III-15 Elecronic Assembly in reality
Figure III-16 Elecronic Assembly
Figure III-17 Example Code Used for Manual Step Testing
Figure III-18 Detecting the color and shapes
Figure III-19 Some Images from the Practical Aspect of the Project"

List of tables

Table II-1 Dimensions and Characteristics of the Joints
Table II-2 Specifications of Motors Used in the Robotic Arm
Table II-3 Hardware Component Overview of Raspberry Pi 4 Model B [15]
Table II-4 Raspberry Pi 4 Model B Starter Kit [15]
Table III-1 I max and Vref
Table III-2 200 Steps \x(Degree)

GENERAL INTRODUCTION

General Introduction

Robotics, deployed together with computer vision and artificial intelligence, represents one of the biggest growth sectors today. The capability of detecting shapes, recognizing objects, and overtly interacting with the environment underpins many innovations that we see in fields like industry, healthcare, logistics, and education. The requirement of integrating these capabilities into an embedded system like the Raspberry Pi is an extremely appealing technical challenge. In the context of this project, we have created an intelligent robotic system(robotic arm) that can, in real time, diagnose geometric shapes with a camera. The work is comprised of visual shape detection, image processing, path calculation, and controlling stepper motors on a lightweight platform like the Raspberry Pi 4 Model B.

This thesis is in the larger milieu of optimizing and deploying embedded artificial intelligence models on constrained hardware, which will be demonstrated within the context of Intelligent Robotics. The demonstration will provide evidence of the utility of libraries like as OpenCV and NumPy in conjunction with the high-level programming language Python, to allow for the development of an end-to-end system of vision and autonomous actuation.

The two main objectives of this project are to firstly evidence the reality of building embedded intelligent systems that are capable of performing precise robotic tasks and secondly, provide a scalable architecture in which advanced projects can take advantage of for expanding into applications such as complex object recognition, human machine interaction, or automated production lines.

The thesis consists of three chapters:

Chapter One presents the theoretical foundation of artificial intelligence, its types, and its areas of applications, computer vision. Additionally, it explains the relationship between cameras and robotic arms in intelligent systems.

Chapter Two presents the elements of the robotic arm that we used for the project. It gives detailed descriptions of the mechanical components and electronic components (Nema17 motors, DRV8825 drivers, Raspberry Pi 4...), and the software tools (Python language, Thonny IDE, and important libraries).

Lastly, Chapter Three presents the applications in practice. It explains how to build the whole system. It also provides explanation for Python code that connects computer vision and control of the automation, from the step of identifying the shape, determining its coordinates, converting

coordinates into angles, and finally converting angles into motor steps, to achieve the correct precision while reaching the target.

The project demonstrates the potential of embedded AI as it pertains to robotic applications. It can be taken further to include more advanced functions such as automated sorting, autonomous navigation, and even assistance by a human in semi-structured settings. It serves as a solid basis for ongoing work in intelligent cyber-physical systems.

CHAPTER I: EMBEDDED INDUSTRIAL ARTIFICIAL INTELLIGENCE

CHAPTER I: Embedded Industrial Artificial Intelligence

1. Introduction:

This chapter provides an overview of Artificial Intelligence (AI) and its main forms: ANI, AGI, and ASI. It then focuses on the use of Industrial AI in enhancing automation and efficiency in manufacturing. The chapter also explores the integration of AI with embedded systems and industrial cameras, in addition to its application in robotic arm movement. Lastly, it outlines the key AI programming languages used in industrial settings.

2. Definition of Artificial Intelligence (AI):

Artificial Intelligence (AI) is the discipline of computer science and engineering that concentrates on inventing systems that can simulate human intelligence. AI in engineering is linked to machines that can perform tasks traditionally done by humans, such as identifying patterns, decision-making, and optimizing data. These intelligent systems using computer science established algorithms and models to enable independent execution and analysis of information. Main areas of artificial intelligence are: Machine Learning; Neural Networks; Computer Vision; and Natural Language Processing [1].

3. The Different forms of Artificial Intelligence: ANI, AGI, and ASI:

Artificial Intelligence (AI) can take on a few different formations. At the most basic level, Artificial Intelligence can be explored through three categories of intelligence and capability levels. Each category represents a version of artificial intelligence, and it also identifies the difference in terms of what is occurring in the real world as opposed to where the real world is heading theoretically.

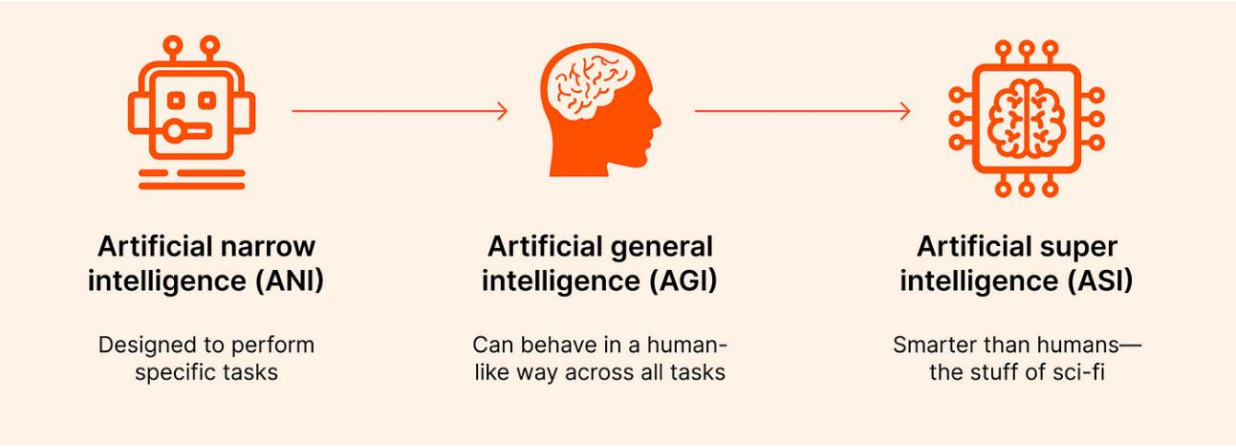


Figure I-1 The Three Stages of Artificial Intelligence: ANI, AGI, and ASI [2]

3.1 Artificial Narrow Intelligence (ANI):

Also referred to as Weak AI, this type of artificial intelligence is the most frequently encountered and currently existing area of artificial intelligence. In ANI, systems are built and trained to fulfill a specific task (or limited set of tasks) as accurately as they can. ANI systems only operate to a specified extent and cannot generalize; they cannot utilize their knowledge outside of the specific domain they were built for [3].

ANI is not conscious, self-aware, or sapient. ANI AI performs on whatever data sets are available to it, it draws conclusions through pattern detection, and it executes algorithmic output that ranges from processing speed to task-specific design. Furthermore, there are many instances of ANI are already part of our everyday lives, such as digital voice assistants (Siri and Alexa) [3].

- **Example:** Our project devised a system with a camera that can detect geometric shapes and their Colors and figure out where they are in the picture. When a shape is identified, the robotic arms move automatically in the direction of the shape based ONLY on what the camera sees. That is considered an example of Artificial Narrow Intelligence because the system is limited to a distinct and narrow task in one category—recognizing/responding to an object in its environment, as opposed to the expansive range of ways humans recognize objects and respond to their environment.

3.2 Artificial General Intelligence (AGI):

Also called Strong AI, AGI is the theoretical point where an AI system would be able to function in an equivalent cognitive capacity to a human being. Essentially, this would mean the system would be able to understand, learn, and apply knowledge to many tasks in many areas/situations just as people do, which includes the ability to adapt to novel situations [3].

An AGI would be capable of normal human reasoning, emotional intelligence, abstract reasoning, and autonomous learning across domains of content and context. AGI would not be restricted to one function but would be able to transfer knowledge from learned tasks into a different but apparently related task, like humans do. However, AGI is a concept that is under heavy research and has not been realized yet. There are a number of significant technological, ethical, and philosophical questions surrounding creating such a system [3].

3.3 Artificial Super intelligence (ASI):

Artificial Superintelligence is a proposed AI that can perform better and faster than humans in all aspects of cognition - scientific creativity, general wisdom, social intelligence, etc. ASI could generate its own plans, insights, and forms of problem-solving to complex global problems that can be measured against human intellect [3].

Although it is entirely hypothetical, ASI comes up in future-oriented research because of its potential. If real, ASI could transform science, medicine, and technology at unprecedented levels. However, because it would be beyond human understanding and potential control, it raises serious questions around safety, regulation, and overall impacts on society on a long-term basis [3].

4. Industrial Artificial Intelligence

Industrial Artificial Intelligence, or Industrial AI, uses powerful artificial intelligence techniques, such as machine learning, deep learning, and computer vision, to boost efficiency, accuracy, and decision making in an industrial environment. Industrial AI systems can operate continuously in real time, monitoring data from machines and sensors and making changes as processes fluctuate in real time, while predicting maintenance actions and changes in operating conditions [4]. As a result, Industrial AI can minimize the time equipment is not running, enhance the quality of products, and build more agile and sustainable manufacturing systems. In addition to working with real time data, a key feature of Industrial AI is computer vision. For example, camera systems that are equipped with artificial intelligence can automatically note product specifications such as size, color, texture, and surface imperfections, when the item is moving down a conveyor line .

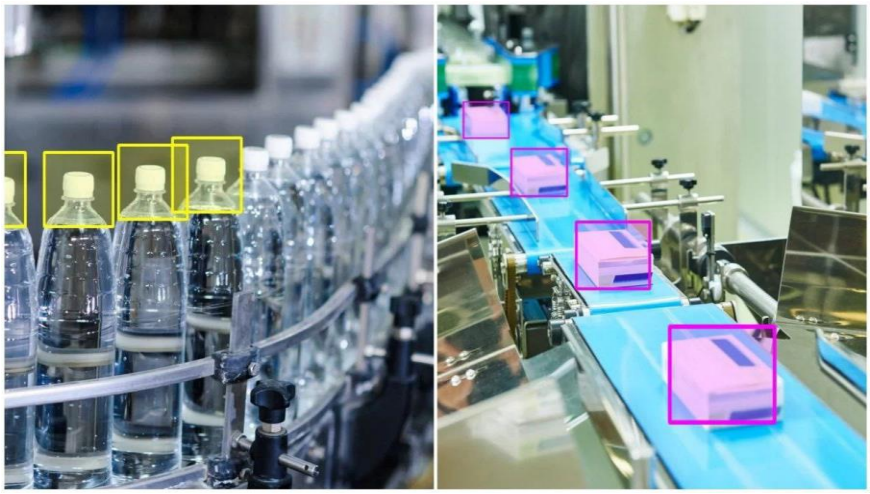


Figure I-2 AI-Powered Visual Inspection on Production Lines [5]

5. Embedded system

An embedded system, or embedded computer, is a computer system with a processor, memory, and input/output units that is designed to carry out a dedicated function as part of a larger mechanical or electrical system. Embedded systems typically are physically smaller, more efficient, and have constraints imposed by size, power consumption, expenses, or weight. Embedded systems are normally contained in large devices (hence the name), although they can be used as stand-alone systems [6].

For our project, we employed an Embedded Artificial Intelligence (AI) setting, where AI capabilities such as object detection and spatial awareness are used directly inside the embedded space. Using a Raspberry Pi 4 Model B as a processing unit, the embedded system captures visuals from a camera and uses this sensory data to determine the shape and location of an object and subsequently controls a robotic arm to interact with the object. This shows how, in the real world, artificial intelligence combined with embedded systems has an unrivaled capacity to perform real-time perception, decision, and control inside a compact and efficient computing environment.

6. Integration of AI and Cameras in Factories

Linking artificial intelligence (AI) and camera systems will reshape modern factories in serious ways and lead to real advances in the areas of automation, quality control, and safety. Cameras have become the eyes of the smart factory. They are constantly providing high-definition data from the factory floor, production lines, equipment, and observing worker behavior. When AI algorithms, especially in the computer vision area, are applied to the visual data input from the cameras, it becomes a powerful tool to impact real time decisions and changes on the factory floor.

AI vision systems can recognize complex patterns, defect product, check assembly, and monitor safety, at a scale and speed no human can match. For example, a camera-system can detect small variances in color, shape, or surface texturing indicating a flaw in a manufactured part. Once identified, action can take place immediately to resolve the issue. AI vision systems can also impact safety by monitoring compliance against workplace safety protocols (in example: identifying whether workers were their helmets or other protective apparel, notifying when non-compliance occurs Figure 1-3). In addition, AI interprets camera inputs to control and dynamically update production line robotic arms and automated machinery and make small fine-tune adjustments based on live environmental feedback data.

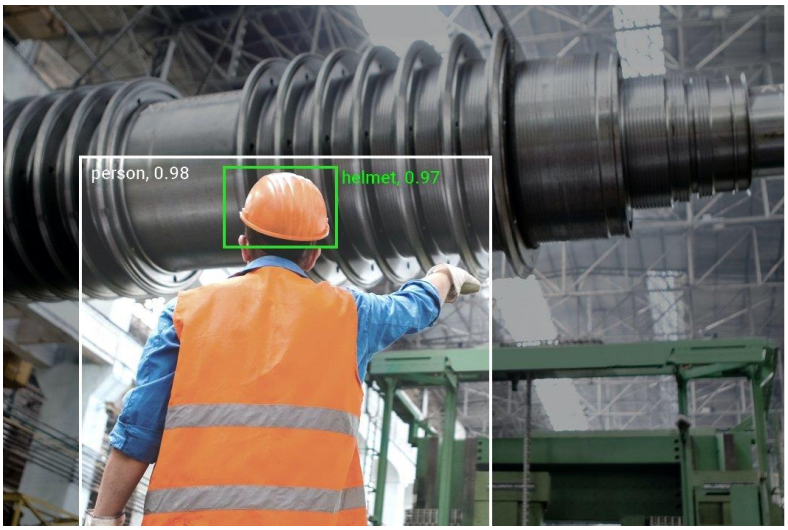


Figure I-3 Real-Time PPE (Personal Protective Equipment) Detection Using Computer Vision [3]

Beyond improving product quality and operational efficiency, this integration supports predictive maintenance by detecting early signs of equipment degradation, helping prevent unexpected failures. As manufacturing continues to evolve toward highly automated and connected ecosystems, the synergy between AI and visual sensing technologies is central to enabling factories that are not only intelligent, but also adaptive and worker-aware.

7. Robotic arm Movement Using Artificial Intelligence

In industrial and research applications, artificial intelligence (AI) has greatly enhanced the control and coordination of robotic arms allowing them to accomplish tasks that were previously beyond their reach of human skill and perception. Previously, traditional robot systems were rigid and limited to specified actions, since their movements were pre-programmed. However, AI allows robotic arms to learn from a database, understand sensor data, and make intelligent decisions in real time - therefore, these hands are more useful in real-world dynamic and unstructured environments.

This capability is fundamentally rooted in machine learning and deep learning algorithms that facilitate robotic systems in reviewing the massive amounts of sensor data- involving joint angles, pressure sensors, or visual data- to adapt their motion accordingly. For instance, a robotic arm could utilize computer vision, process camera images to derive an object's size, shape, orientation, and texture, and determine the optimal strategy to grasp the object. Such systems can even observe the operation of multiple robotic arms at once; a camera and AI could recognize which robotic arm is actively engaged and which one is paused so that the task is coordinated, and this would not add a burden to the system (figure 1-4). By developing correspondingly precise object manipulation

capabilities, these systems would impart a degree of intelligence to not only operate the robotic arms, but to also coordinate the operation of multiple robotic units working on a production line.

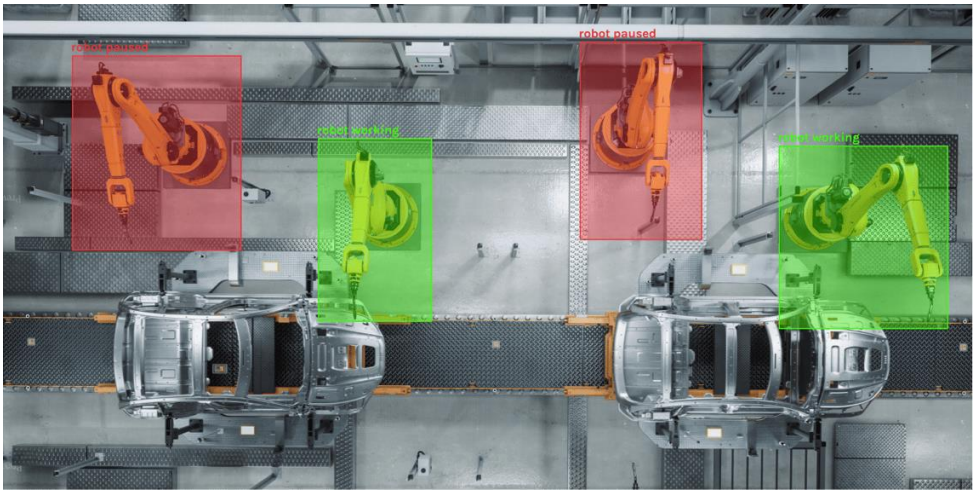


Figure I-4 AI-Driven Camera System for Monitoring Robotic Arm Activity [7]

Furthermore, AI allows robotic arms to improve in a similar manner as reinforcement learning in the simulation or in real life through continued trial and error through exposure. An AI robotic arm may continuously improve so that eventually it knows when to change grasp strength, and how to manipulate multiple fingers/coordinate finger joint angles depending on the context. The flexibility of AI robotic arms with respect to assembling items/picking and packing items and in collaborative robotics (cobots) or when working with humans offers enormous value given the importance of flexibility, physical safety, and agility.

AI robotic hands are also capable of real-time responsiveness; for example, when picking up an object, their ability to reposition in response to unexpected change in an object position or change grips when collaborating with a human. Therefore, with AI and advanced robotics working together, manufacturers are creating systems that are not just automated, but capable of intelligent awareness, contextual sensitivity and mimicking certain functions of human hand use. Certainly, the evolution of robotic systems to be able to mimic human hand function is critical to manufacturers seeking improvements in precision, productivity and flexibility in meeting manufacturing demands of increasing complexity.

8. Artificial Intelligence Programming Languages for Industrial Environment Programming

Programming artificial intelligence (AI) systems for industrial applications requires languages that can deliver high-speed performance, scalability, support real-time processing, and deploy onto embedded hardware or robot platforms. Further, the level of programming is orchestrated by the application (i.e., computer vision, robotic control, predictive maintenance, or embedded AI).

8.1. Python

Python is by far the most popular programming language in the AI development space, thanks to its readable syntax, large community of developers, and extensive libraries including TensorFlow, PyTorch, OpenCV, and Scikit-learn, which are particularly powerful in developing models for machine learning, deep learning and computer vision. Even though Python is not natively a real-time programming language, it remains widely used to prototype AI systems that are eventually implemented to run in real-time industrial applications [8].

8.2 C/C++

C and C++ are important when designing industrial systems that require real-time operation, efficient memory footprint and hardware access at a low level. This situation is often found with embedded systems and robotics. For example, libraries such as OpenCV, and platforms like the Robot Operating System (ROS), span across development libraries that support developing and running AI applications in real-time for manufacturing and robotic automation [9].

8.3. Java

Java is also widespread used in industrial environments, especially given the fact it is portable and reliable for enterprise-scale systems. Java is not widely used in AI development compared to languages like Python or C++, and some frameworks that support Java like Deeplearning4j. Java is often used in industrial IoT (internet-of-things) systems for backend integration, and real-time monitoring applications such as temperature and/or humidity [10].

8.4. MATLAB

MATLAB is a commonly used platform in academia and engineering for learning, simulation, and prototyping. It has excellent toolboxes for dealing with machine learning, control system development, and signal processing. Its compatibility with Simulink is an added bonus for modeling and simulating automation systems prior to building and testing in an industrial automation setting [11].

8.5. Rust and Go (Emerging)

Rust and Go are newer languages characterized by their respectable performance and memory safety for concurrency. They are gaining traction in edge AI and industrial Internet of Things use cases, especially in safety-critical systems that require reliability and direct control of devices at a low-level [12].

9. Conclusion

Artificial Intelligence has played a transformational role in the modern industrial environment by enabling machines, learning and wisely working. Its integration in production systems increases efficiency, accuracy and adaptability. Industrial AI combines datapowered algorithms with domain expertise to customize processes in real time, while embedded systems provide the necessary computational foundation required to make localized decisions.

The use of cameras and AI-operated computer vision systems improves quality control and activist safety, enabling real-time visual analysis and automation. Robotic arms directed by AI are capable of performing complex manipulations with increased flexibility and coordination. These intelligent systems require languages that support high-demonstration computing, real-time accountability and spontaneous integration with industrial hardware. Together, these technologies make the origin of intelligent, associated and adaptive industrial systems.

CHAPTER II: SYSTEM ARCHITECTURE (HARDWARE AND SOFTWARE COMPONENTS)

CHAPTER II: System Architecture (Hardware and Software Components)

1. Intoduction

Since we chose to construct an intelligent robotic arm that can recognize shapes and colors with artificial intelligence methods and navigate toward them, it was important to do diligent research and select the proper technical components, which allowed the system to accomplish the intended tasks successfully.

In this chapter, we will explain all of the hardware and software components used in building the project in detail, starting with the 3D-printed robotic arm which is based on an open-source design, the Nema stepper motors which were chosen based on meeting the movement requirements, and the Raspberry Pi 4 Model B board which provided the system's brain.

We will explain the operating system (Raspberry Pi OS), which was used, the development environment (Thonny IDE) used, and the main programming libraries, specifically Python, OpenCV, and RPi.GPIO, which formed the basis of the software developed for recognition and control.

2. Robotic arm BCN3D Moveo (modify)

The BCN3D Moveo is an open-source, 3D-printed robotic arm developed by BCN3D Technologies in collaboration with the Ministry of Education of the Government of Catalonia. Designed for educational use, the original Moveo features five degrees of freedom driven by stepper motors, providing a cost-effective platform for learning mechanical design, automation, and industrial programming.



Figure II-1 Robotic arm BCN3D Moveo [13]

In this project, we used the original Moveo design but assembled only the first three axes: the base, shoulder, and elbow. Each axis is actuated by a stepper motor, enabling precise motion control. This partial assembly maintains the core educational purpose of the project while focusing on foundational mechanical concepts and motion control.

According to the Moveo BCN3D design specifications, the approximate dimensions of the parts (base, shoulder, elbow) are as follows:[14]

Table II-1 Dimensions and Characteristics of the Joints

Joint	Approximate Length	Primary Function
Base	23.2 cm	Supports the arm and allows horizontal rotation (θ1)
Shoulde	22.1 cm	Enables vertical movement of the arm (θ2)
Elbow	13 cm	Extends the arm toward the target (θ3)

- **Note:** Only the main elbow segment was considered, excluding the end-effector (gripper), in order to simplify the kinematic calculations and focus on the core structure of the robotic arm.[14]

3. Motors

In our project, the robotic arm is powered by three stepper motors**,** each controlling a specific joint: base, shoulder, and elbow. NEMA 17 motors are used for the base and elbow, while a NEMA 23 motor is used for the shoulder to provide higher torque. This configuration ensures reliable and precise movement Tableau 3-1.

Table II-2 Specifications of Motors Used in the Robotic Arm

Joint	Motor Model	Current (A)	Torque (Kg·cm)
Base	JK42HS48-1684 (NEMA 17)	1.68 A	5.1 Kg·cm
Shoulder	JK57HS76-2804 (NEMA 23)	2.8 A	19.26 Kg·cm
Elbow	STP-42D221-03 (NEMA 17)	~1 A	~ 4.0 Kg·cm

3.1 NEMA 17 STP-42D221-03

The NEMA 17 STP-42D221-03 (EM-284) is a unipolar hybrid stepper motor with a step angle of 1.8° (200 steps per revolution). Each phase draws approximately 1.0 A at 5–12 V, delivering a holding torque of 0.314 N·m (approximately 4.0 kg·cm).

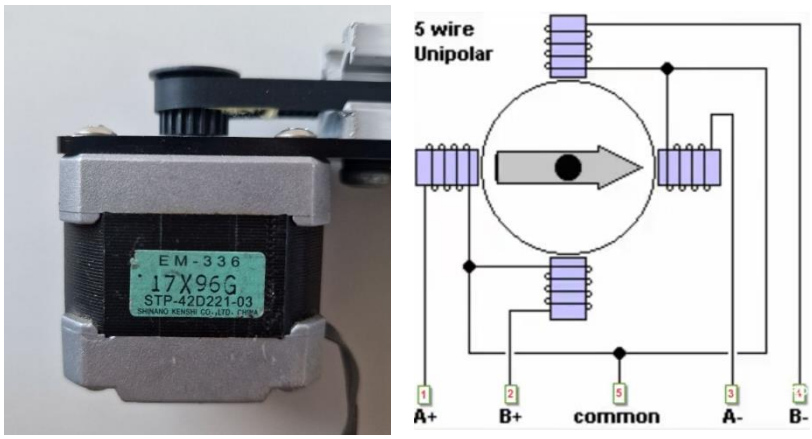


Figure II-2 NEMA 17 STP-42D221-03 and its diagrame

3.2 NEMA 23 JK57HS76-2804

The NEMA 23 JK57HS76-2804 is a bipolar hybrid stepper motor with a step angle of 1.8°, corresponding to 200 steps per revolution. It operates at a rated current of 2.8 A per phase and a nominal voltage of 3.36 V. The motor delivers a holding torque of 1.89 N·m (approximately 19.26 kg·cm), making it highly suitable for applications requiring high torque and positioning precision, such as and robotic arms.

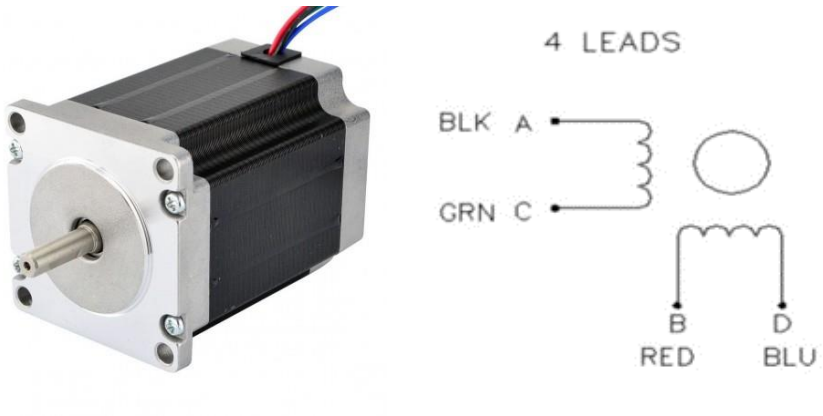


Figure II-3 NEMA 23 JK57HS76-2804 and its diagrame

3.3 NEMA 17 JK42HS48-1684

This is a bipolar hybrid stepper motor with a step angle of 1.8° (200 steps per revolution). It operates at 1.68 A per phase with a rated voltage of 2.8 V, and provides a holding torque of 4.2 kg·cm.

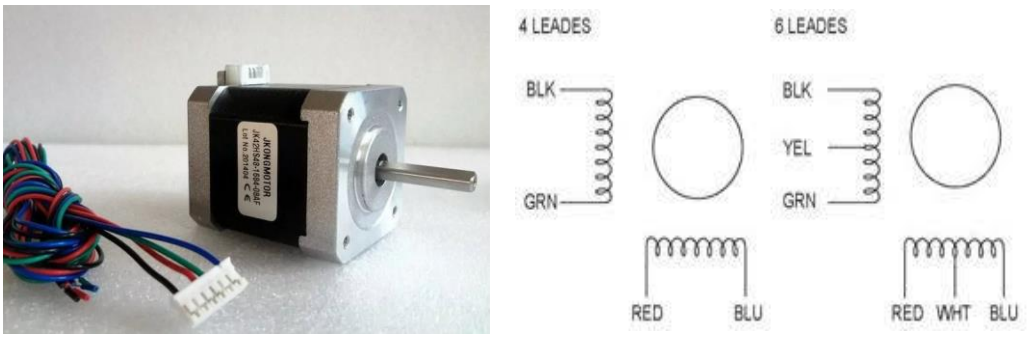


Figure II-4 NEMA 17 JK42HS48-1684 and its diagrame

4. Electronic cards used in arm control

In our project, we used the Raspberry Pi 4 Model B as the central control unit responsible for managing the operation of the robotic arm. The Raspberry Pi executes software routines that send control signals to the DRV8825 stepper motor driver modules, which in turn power and control the movement of the stepper motors. Communication between the Raspberry Pi and the DRV8825 drivers is achieved via GPIO pins, using step and direction signals. This configuration allows for accurate positioning and smooth motion of each joint in the arm. Additionally, the system can be programmed in Python, making it flexible and easy to adapt for educational or prototyping purposes.

4.1. Control Unit (Raspberry Pi 4 Model B)

The Raspberry Pi Foundation developed the Raspberry Pi 4 Model B as a major upgrade in their single-board computer (SBC) series. When released in June 2019, the Pi 4 introduced significant improvements in processing power, memory capacity, connectivity options, and multimedia capabilities compared to previous models. This device provides near-desktop performance within a compact and affordable form factor, making it suitable for both simple electronics applications and more advanced embedded systems. For these reasons, we used the 8 GB RAM version of the Raspberry Pi 4 Model B in our project .



Figure II-5 Raspberry Pi 4 Model B 8GB

4.1.1. Main Components of Raspberry Pi 4 Model B

Here is a comprehensive list of the main components of the Raspberry Pi 4 Model B, along with their technical descriptions and functions:

Table II-3 Hardware Component Overview of Raspberry Pi 4 Model B [15]

Label on Diagram	Component Name	Description
4 Cores @ 1.5 GHz	BCM2711 CPU	Quad-core ARM Cortex-A72 CPU @ 1.5 GHz with 1MB shared L2 cache.
RPi GPIO	40-pin GPIO Header	General-purpose I/O pins used for hardware interfacing and control.
DDR4 SDRAM	LPDDR4 RAM	High-speed RAM available in 2GB, 4GB, or 8GB LPDDR4 variants.
Power Management	PMIC	Power management IC distributing voltages across the board.
USB-C Power	USB-C Power Input	Supplies 5V/3A power (Power Delivery not supported).
2× HDMI	Micro HDMI Ports	Supports up to dual 4K@30Hz or single 4K@60Hz display output.
UART	UART Header	Serial interface for console output or debugging via GPIO pins.
Analog Video	Composite Video Output	Provided via 4-pole AV jack, also delivers stereo audio.
CAM or Display	MIPI CSI & DSI Connectors	Connect official Raspberry Pi camera and display modules.
Ethernet Gigabit	Gigabit Ethernet Port	True gigabit Ethernet over a dedicated controller (not over USB like on Pi 3).
2× USB 2	USB 2.0 Ports	For connecting keyboard, mouse, and other peripherals.
2× USB 3	USB 3.0 Ports	High-speed USB ports (up to 5 Gbps) for data-intensive peripherals.
PoE Conn.	PoE Header	Supports optional Power over Ethernet HAT.
Wireless	WiFi + Bluetooth	Dual-band 802.11ac Wi-Fi and Bluetooth 5.0 onboard.
Mounting Holes	Mounting Points	Physical holes for attaching heatsinks, fans, or enclosures (HAT compatible).

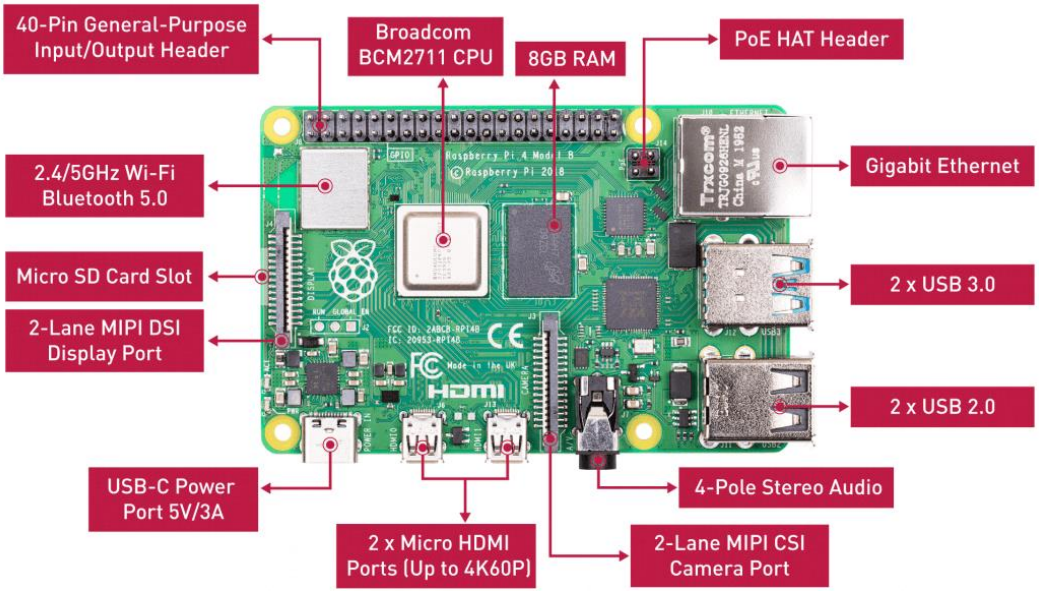


Figure II-6 Labeled Diagram of Raspberry Pi 4 Model B (8GB RAM Version) [16]

4.1.2. Raspberry Pi 4 Model B Starter Kit: Contents and Purpose

A Raspberry Pi 4 Model B Starter Kit is a bundle designed to give users everything needed to get up and running quickly with their Raspberry Pi 4. These kits are ideal for beginners and enthusiasts.

Table II-4 Raspberry Pi 4 Model B Starter Kit [15]

Item	Purpose	**Image**
Raspberry Pi 4 Model B (2GB/4GB/8GB)	Main computer board for running software and connecting hardware components.	
Official Power Supply (5V 3A USB-C)	Provides stable and adequate power required for the Raspberry Pi 4's operation.	
microSD Card (32GB+ with Raspberry Pi OS)	Serves as the main storage medium with the operating system and data.	
Protective Case (with fan or heatsink)	Protects the board and aids in heat dissipation.	
Micro HDMI to HDMI Cable	Connects the Raspberry Pi 4 to a monitor or TV for video and audio output (up to 4K).	

USB Keyboard and Mouse	Input devices used to control and interact with the Raspberry Pi 4.	
Heatsinks and/or Cooling Fan	Helps cool the CPU and other chips to prevent thermal throttling during heavy use.	
CSI/DSI Camera and Display Cables	Connects Raspberry Pi compatible cameras and displays via CSI and DSI interfaces.	
Display Monitor	Visual interface for accessing the desktop environment and running graphical applications.	

4.2. DRV8825

The DRV8825 is a microstepping stepper motor driver developed by Texas Instruments, designed for precise control of bipolar stepper motors. It supports up to 1/32 microstepping resolution, operates within a voltage range of 8.2V to 45V, and can provide up to 2.2 A per coil with sufficient cooling. The driver includes several built-in protection features, such as over-temperature shutdown, overcurrent protection, and undervoltage lockout, making it a reliable choice for robotics and automation projects.

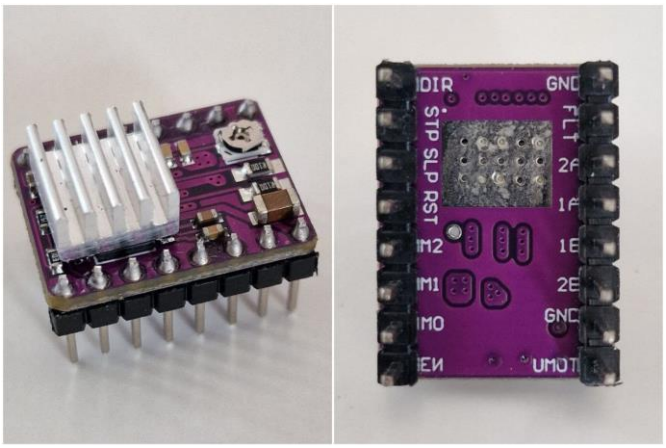


Figure II-7 DRV8825

To adjust the current on the DRV8825, a digital multimeter (multimeter) is used to measure the Vref voltage on the small potentiometer located on the driver board. First, power the board and

place the multimeter's black (COM) probe on GND and the red probe on the metal screw of the potentiometer. While observing the voltage reading, gently turn the screw with a small screwdriver: clockwise to increase the current, and counterclockwise to decrease it. Once the desired Vref voltage is reached (based on the target current using the formula Imax= 2. Vref) stop adjusting.

5. Software Environment and Tools Used

5.1. Operating System (Raspberry Pi OS)

Raspberry Pi OS (formerly Raspbian) is the official operating system from the Raspberry Pi Foundation for Raspberry Pi single-board computers. It is a Debian-based Linux distribution available for free, optimized to work efficiently on ARM architecture, and further more on the hardware utilized in all Raspberry Pi models. Raspberry Pi OS comes in a variety of versions including full desktop version with recommended software (e.g., LibreOffice, Chromium browser), a Lite option for headless installations (no desktop), and a minimal image for special application. It carries education software and development environments including Python, Scratch, and Thonny and is therefore optimized for learning, experimenting, and developing embedded systems. Ongoing updated and supported by a large community, Raspberry Pi OS values stability, simplicity, and educational functionality. The OS can be installed via the official Raspberry Pi Imager tool and supports Raspberry Pi models from the first to the Raspberry Pi 4 Model B [17].

5.2. Programming Language (Python and Integration with a Raspberry PI)

Python is a high-level, open-source programming language that has straightforward clean syntax and is easy to learn. Python is quickly gaining notoriety and acceptance due to its the programming language's ability to solve real world problems in elegant ways, and works not only for novice programmers, but also skilled professionals. Python is an interpreted programming language created by Guido van Rossum, and released in 1991. It is a multi-paradigm programming language that appropriately supports object-oriented programming and functional programming development. Python is a Reasonably versatile programming language, and you can find it in almost all domains these days including artificial intelligence, machine learning, data analysis, robotics, and web development. You have an amazing standard library, and the infinite third-party libraries online, such as NumPy, OpenCV, and TensorFlow, which means, you can build very complex systems in just hundreds of lines of code, when the equivalent lower-level programming languages, such as C or Java may take thousands and thousands of lines of code to recreate.[16]

Python is the default and officially supported language in the Raspberry Pi ecosystem. It has been installed with Raspberry Pi OS, giving developers the ability to program hardware through GPIO using a library like RPi.GPIO or gpiozero. This allows the programmer to control motors, sensors, and other electronic modules fully. Additionally, Python works seamlessly with position-specific software design environments like Thonny IDE, which is also preconfigured on Raspberry Pi OS for writing, testing, and debugging code. The essence of Python and Raspberry Pi biodiversity provides the development community with a useful package for rapid prototyping and educational activities. This project attempts to use Python to bridge the high-level computer vision algorithms, to low-level hardware control, and ultimately foster an environment for the framework of an intelligent robotic system that encompasses visual recognition and autonomous actuation.[18]



Figure II-8 LOGO Python

5.3. Thonny IDE on Raspberry Pi

Thonny IDE is an integrated development environment which has been developed explicitly to streamline the learning and development stages of python programming, particularly for novices. Thonny is packaged with Raspberry Pi OS, meaning it is the default environment for Python programming on this platform. Thonny has a simple, easy to use, user interface with basic features such as syntax highlighting, code completion, an integrated debugging, as well as a step through operation to inspect variable values. The significant feature which sets Thonny apart is the ability to manage Python packages. As it is built directly into the IDE, it is easier to set up development projects on the Raspberry Pi, as you can easily create and remove virtual environments.[19]

From a Raspberry Pi standpoint, Thonny is low footprint and has direct access to GPIO libraries, hence Thonny commands a significant role in physical computing and embedded systems development as it bridges the gap between the code development and interaction with hardware components such as motors, sensors and cameras using python scripts. The usability and focus on education is likely the reason why Thonny is the most commonly used IDE in schools when integrating projects and STEM education. In this project, Thonny IDE was the environment of

choice for writing and debugging the Python code that controlled the robotic arm, and grabbing the camera inputs in real time.[19]



Figure II-9 LOGO Thonny python IDE

5.4. Software Libraries Utilized in the Project

This project used some of the most advanced Python libraries for image processing, math, and hardware control. These libraries were dedicated to performance and offered the project a level of accuracy in performing the system's tasks.

5.4.1. OpenCV

Open cv is one of the most powerful libraries for computer vision; it has about 2,500 in-built functions. OpenCV supports a wide variety of capabilities including image and video processing, edge detection, shape recognition, camera calibration, and object tracking. In this project, OpenCV was vital for the analysis of the camera frames and detecting the gray circles accurately to retrieve the coordinates the robotic arm needed to move. [18]



Figure II-10 LOGO OpenCv

5.4.2. NumPy

NumPy is a specialized library for arrays and numerical computations, which was used for the accelerated processing of image data and converting pixel coordinates to real world coordinates. NumPy also provided ease of the math needed for motor control.[18]

5.4.3. RPi.GPIO

This library provides a straight forward interface to the Raspberry Pi GPIO pins. This library was used to program and send digital control signals to the stepper motors that controlled the robotic arm movement.[18]

5.4.4. math

The math library was utilized for trigonometric and geometric calculations, especially in the inverse kinematics algorithms. It allowed functions like cos(), sqrt(), and conversion to degrees for angles.[18]

5.4.5. time

The time library managed the timings of when signals were being transmitted to the motors, ensuring a synchronized and smooth movement of the robotic system in real time.[18]

6. Conclusion

The second chapter of the project is the foundation for understanding how the intelligent robotic system is constructed. It goes into real detail about the technical considerations surrounding not only the architecture but also the software architecture of the system. We started from the point when we had finished constructing the robotic arm, then we started to cover how the motors are handled, specifically, the Nema 17 stepper motors and how they are driven using DRV8825 drivers. The drivers are controlled by providing a pulse to the motors, thus allowing mechanical movements to correspond with the pulses, apart from any other considerations.

We then covered the function of the main control unit, the Raspberry Pi 4 Model B, which is essentially the brain of the system. It processes images and makes decisions, which it communicates to the motors. We also outlined the overall software environment based on the Python programming language, including the primary libraries we used: OpenCV, NumPy, and RPi.GPIO...

It was in this symbiosis of hardware and software that we could design a completely integrated system that could interact with its environment intelligently and with a high degree of precision.

CHAPTER III: APPLICATION AND OPTIMIZATION OF A ROBOTIC ARM

CHAPTER III: Application and Optimization of a Robotic arm

1. Introduction

In this chapter, we will concentrate on a key aspect of the intelligent system that uses computer vision and links mechanical control to its operational cycle – the robotic arm's operational cycle. In this section, we describe how the camera identifies shapes and colors, how coordinates derived from the identified objects are converted to real-world spatial location, how those locations are converted to motion angles, ultimately leading to motor commands. We will also discuss how we imposed angles limits, communicated with the stepper motors (Nema 17) through the DRV8825 drivers, and the sequence of events we took when found our target. This whole process gives us a closed loop system that joins perception, computation and control, all of which will together help ground the development of responsive and efficient robotic systems.

2. Mechanical Part of the Arm

2.1. Arm Preparation

In the university laboratory, we found a partially 3D-printed robotic arm labeled BCN3D Moveo. After conducting research on this model, we found that it is an open-source project, freely available on GitHub, allowing it to be used and modified for educational and research purposes.

Upon inspecting the available components, we noticed that only three main parts had been printed and prepared: the base, the shoulder, and the elbow. So, we proceeded to the assembly phase, relying on the technical guide and instructions provided in the official project documentation, which can be found at: https://github.com/BCN3D/BCN3D-Moveo

This guide offered detailed instructions for assembling joints, connecting mechanical axes, and configuring the arm for smooth and stable movement.



Figure III-1 Arm Preparation

2.2. Integration of Stepper Motors

As part of the robotic arm's mechanical assembly, the stepper motors were precisely installed in the locations assigned to them during the design phase. Each motor was positioned within a dedicated compartment integrated into the arm's structure, ensuring optimal alignment with the system's kinematic requirements.

A NEMA 17 stepper motor was positioned at the base joint to enable rotation around the vertical axis. Another NEMA 17 was mounted at the elbow joint, where it manages the controlled bending and extension of the arm segment. The shoulder joint was equipped with a NEMA 23 motor, selected for its higher torque to accommodate the vertical lifting of the arm.

The transmission of motion is carried out through synchronous timing belts, which provide smooth, precise, and backlash-free movement. All motors were securely fixed using custom-designed mounts integrated into the 3D-printed chassis of the robotic arm. This setup not only guarantees mechanical stability but also enables the system to perform with three degrees of freedom, sufficient for executing targeted and coordinated tasks.

2.3. Mounting on the Wooden Platform and Defining the Workspace

Once assembled, the robotic arm was mounted on a flat wooden platform that served as a test workspace. On this platform, we drew a Cartesian coordinate system (X, Y) with a unit scale of 1 centimeter along each axis. This allowed us to define the work area precisely and easily convert the coordinates of the detected objects from camera frames into real-world positions.

We established the center of the base as the origin point (0, 0) of the coordinate system, making it the reference for all inverse kinematics calculations. It is important to note that the radius of the robotic base is approximately 7.5 cm, which defines a circular dead zone where the arm cannot operate. This zone was taken into account when determining the effective working range of the arm.

3. Raspberry Pi 4 Model B (Getting Started)

3.1. Installing Raspberry Pi OS on microSD

To install Raspberry Pi OS on a microSD card, download Raspberry Pi Imager from the official website and install it on your computer. Insert a microSD card (16GB or larger). Open the Imager, select "Raspberry Pi 4" under "Choose Device", then choose the desired OS version under "Choose OS" (e.g., Raspberry Pi OS 64-bit). Select your microSD card under "Storage", then click "Write". After the process completes, eject the card and insert it into the Raspberry Pi 4.

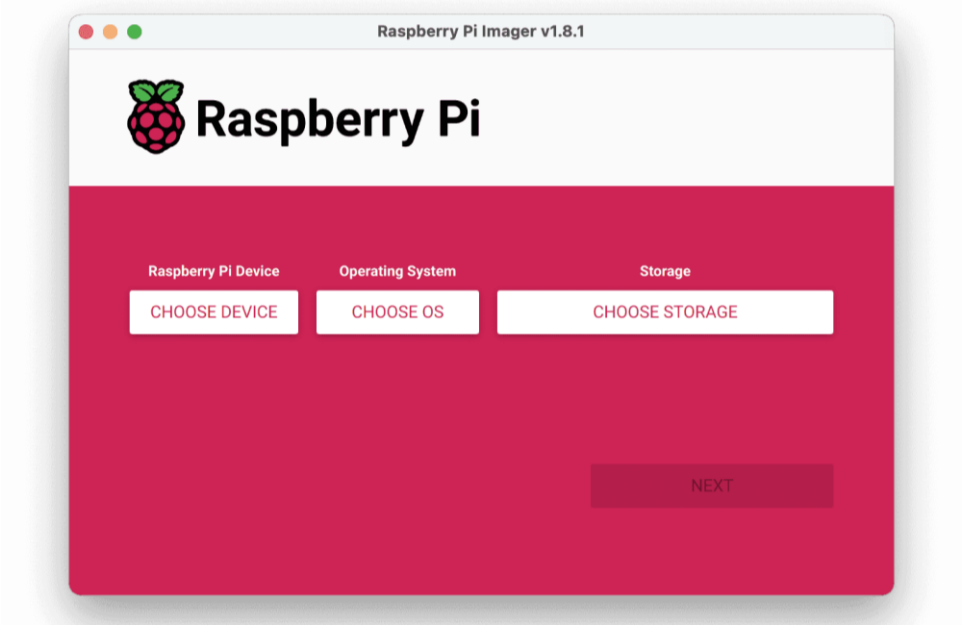


Figure III-2 Raspberry Pi imager

3.2. Initial Setup for Raspberry Pi 4

To start using the Raspberry Pi 4 Model B, you need a microSD card of 16GB or larger, preloaded with the Raspberry Pi OS. Insert the microSD card into the slot on the underside of the board. Then, connect a 5V 3A USB-C power supply, a monitor via a micro-HDMI cable, and a keyboard and mouse via USB ports. Once powered on, the Raspberry Pi will boot into the operating system, and you can begin interacting with the desktop interface, writing code, and configuring settings. This basic setup is essential to start development or hardware control using the Raspberry Pi.

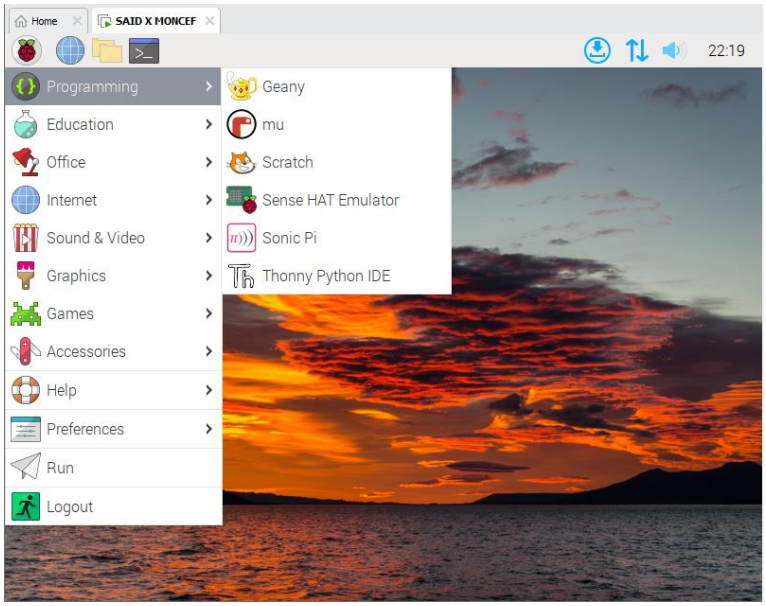


Figure III-3 desktop interface

3.3. Configuring Raspberry Pi 4 for Python Project Execution

Before setting up the virtual environment, we updated the Raspberry Pi OS to ensure system stability and access to the latest package versions. We used the following commands in the Terminal:

```
saidmoncef@raspberry:~ $ sudo apt-get update
Hit:1 http://deb.debian.org/debian bullseye InRelease
Hit:2 http://archive.raspberrypi.org/debian bullseye InRelease
Hit:3 http://deb.debian.org/debian-security bullseye-security InRelease
Hit:4 http://deb.debian.org/debian bullseye-updates InRelease
Reading package lists... Done
```

Figure III-4 sudo apt -get update

```
saidmoncef@raspberry:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  chromium chromium-common chromium-sandbox libjavascriptcoregtk-4.0-18
```

Figure III-5 sudo apt-get upgrade

After updating, we created a Python virtual environment to isolate the project's dependencies from the system-wide Python installation. This prevents package conflicts and keeps the development environment organized. The steps were:

- Install the virtual environment module:

```
saidmoncef@raspberry:~ $ sudo apt-get install python3-venv
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-venv is already the newest version (3.9.2-3).
python3-venv set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
```

Figure III-6 sudo apt-get install python3-venv

- We created a new folder (name: saidxmoncef) to contain the project and moved into it:

```
saidmoncef@raspberry:~ $ cd /home/saidmoncef/saidxmoncef
saidmoncef@raspberry:~/saidxmoncef $
```

Figure III-7 *cd /project-folder*

- Create the virtual environment inside the folder:

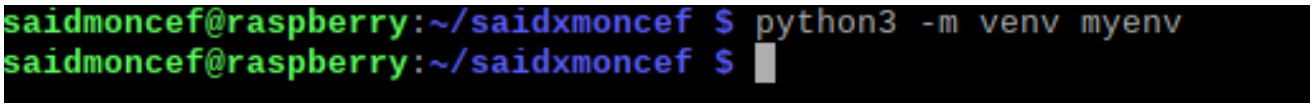


Figure III-8 python3 -m venv myenv

- Activate the virtual environment:

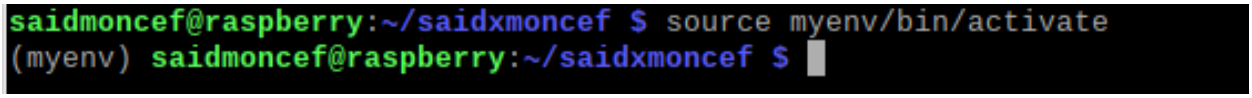


Figure III-9 *source myenv/bin/activate*

Once activated, the environment was ready for installing required libraries using pip.

3.4. Installing Required Python Libraries

3.4.1. Installing the OpenCV

LibraryTo enable image processing and computer vision features in our project, we installed the OpenCV library within the virtual environment. After activating the environment, we used the following command in the Terminal:

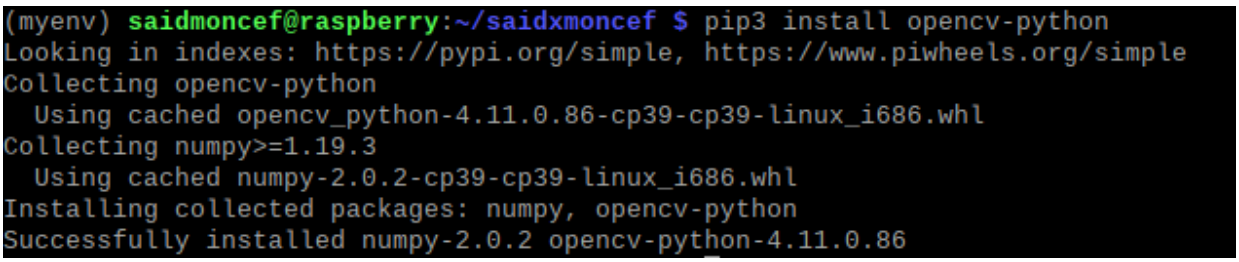


Figure III-10 pip3 install opencv

This installed the core OpenCV package required for capturing, processing, and analyzing images from the camera connected to the Raspberry Pi.

3.4.2. Installing the RPi.GPIO Library

To control the Raspberry Pi's GPIO (General Purpose Input/Output) pins, we installed the RPi.GPIO library inside the virtual environment. After activating the environment, the following command was executed in the Terminal:



Figure III-11 Pip3 install RPi.GPIO

3.4.3. Installing the NumPy Library

To support numerical operations and array processing within the project, we installed the NumPy library inside the virtual environment. After activating the environment, we ran the following command:

```
(myenv) saidmoncef@raspberry:~/saidxmoncef $ pip3 install numpy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: numpy in ./myenv/lib/python3.9/site-packages (2.0.2)
```

Figure III-12 *pip3 install numpy*

4. Camera Calibration and Perspective Mapping

To enable the robotic system with visual perception capabilities, we first initialized and tested the USB camera using the Raspberry Pi board. Compatibility and supported display resolutions were verified by executing the following command in the terminal:

v4l2-ctl --list-formats-ext

```
saidmoncef@raspberry:~ $ v4l2-ctl --list-formats-ext
```

Figure III-13 v4l2-ctl --list-formats-ext

This allowed us to determine the supported formats and resolutions. We selected a resolution of 320x240 pixels, as it ensured lightweight processing and prevented issues such as display flickering or lag.

Following this, the camera functionality was tested using a basic Python scrip:

```python
import cv2
# Open the default camera (camera index 0)
cap = cv2.VideoCapture(0)
# Check if the camera opened successfully
if not cap.isOpened():
    print("❌ Cannot open camera")
    exit()
# Loop to read and display frames
while True:
    ret, frame = cap.read()
    if not ret:
        print("❌ Failed to grab frame")
        break
    cv2.imshow("Camera", frame)
# Exit on pressing 'ESC'
    if cv2.waitKey(1) & 0xFF == 27:
        break
# Release the camera and close windows
cap.release()
cv2.destroyAllWindows()
```

Figure III-14 Python code to operate the camera

Once the camera was confirmed to be functioning correctly, we mounted it directly above the robotic arm, oriented vertically downwards to capture the wooden platform where the arm operates. This top-down configuration provided a complete view of the working area, facilitating accurate object tracking.

To associate image coordinates with real-world coordinates, a homographic transformation was performed. We manually selected four reference points in both the image space and the physical space. The relationship established was as follows:

- [-29, 50] corresponds to the pixel (0, 0)
- [28, 50] corresponds to the pixel (319, 0)
- [-24, 8] corresponds to the pixel (0, 239)
- [23, 8] corresponds to the pixel (319, 239)

These reference points were later used in a Python script with the OpenCV library to compute the transformation matrix, which enabled accurate projection of any detected object from the camera frame to its real position on the workspace platform.

5. Controlling NEMA Stepper Motors via Raspberry Pi Using DRV8825 Modules

During the development of our robotic system, precise control over the motion of a 3-DOF robotic arm was essential. To achieve this, we utilized three stepper motors of the NEMA family:

- A NEMA17 motor for the base, providing horizontal rotation,
- A NEMA23 motor for the shoulder, enabling vertical lift and lowering,
- A NEMA17 motor for the elbow, extending or retracting the arm forward.

To interface these motors with the Raspberry Pi 4 and control them programmatically, we employed three DRV8825 modules, one for each motor. These modules are efficient drivers that allow control over step count, direction, and current for each stepper motor.

5.1. DRV8825 Module (Wiring and Functionality)

Each DRV8825 module was connected as follows:

- The STEP and DIR pins were connected to GPIO pins on the Raspberry Pi.
- The VMOT pin was connected to a 12V external power supply.
- The A1, A2, B1, B2 pins were wired to the motor coils.

- A common GND was established between the Raspberry Pi and the power supply.

The STEP pin receives pulses corresponding to the number of steps the motor should move, while the DIR pin controls the rotation direction (clockwise or counter-clockwise).

To prevent overheating or damaging the motors, we manually adjusted the current limit on each DRV8825 module using the built-in potentiometer. The maximum current was calculated using the following formula:

- $Vref = Imax \cdot 5 \cdot Rs$ *(1)*
- $Imax = \frac{Vref}{5 \cdot Rs}$

Where:

- Rs: is the onboard sense resistor Rs= ~ 0.1
- 5: constant that represents the internal current sense gain of the DRV8825 driver.
- *Vref :* was measured and adjusted using a digital multimeter.

The current limits were set as follows:

Table III-1 I max and Vref

Motor	Type	*Imax*	Computed *Vref*
Base (NEMA17)	NEMA17	1.5A	0.75V
Shoulder (NEMA23)	NEMA23	2.8A	1.4V
Elbow (NEMA17)	NEMA17	1.5A	0.75V

5.2. Power Supply Configuration

Given the varying power requirements of the motors, we utilized two separate power sources:

- A 12V / 2A power supply was dedicated to the NEMA17 base motor.
- A 12V / 5A power supply was shared between:
  - The NEMA23 shoulder motor (2.8A),
  - The NEMA17 elbow motor (1.5A),

totaling 4.3A, which is within the 5A supply limit.

We initially aimed to use a 12V / 10A power supply for all motors, but due to availability constraints in the lab and limited time at the end of the project, we opted for the dual-source setup described above.

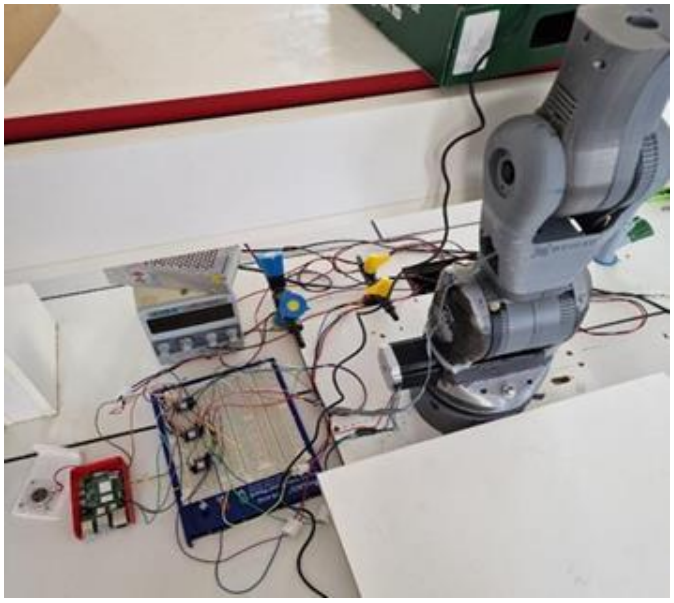


Figure III-15 Elecronic Assembly in reality

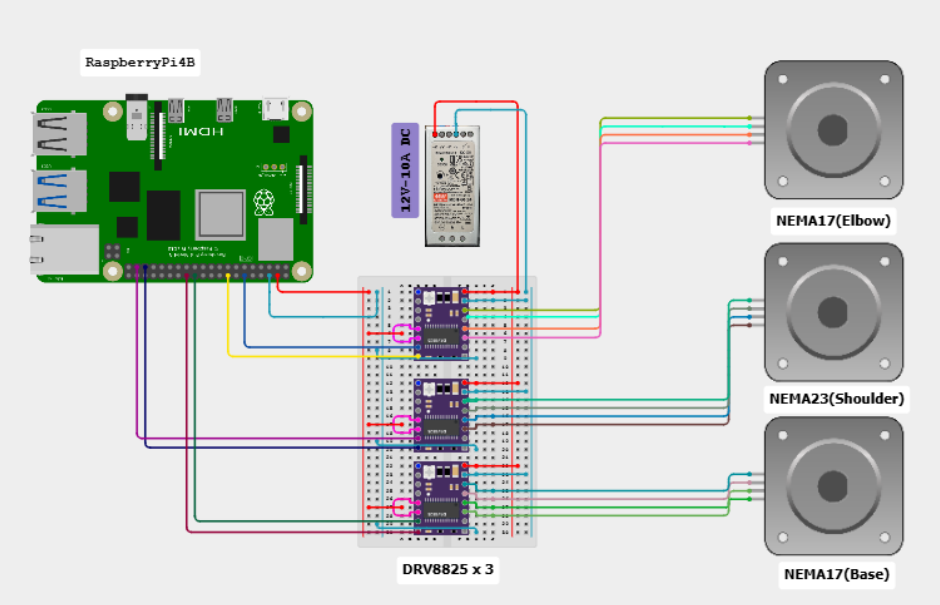


Figure III-16 Elecronic Assembly

6. Experimental Step-to-Angle Calibration

To establish a reliable relation between the number of steps and the actual rotation angle for each motor, we conducted manual calibration experiments using a digital angle finder. We rotated each motor by exactly 200 steps and measured the resulting angular displacement.

The following approximations were established:

Table III-2 200 Steps \x(Degree)

Motor	Type	200 Steps \x(Degree)
Base	NEMA17	200 Steps\35Degree
Shoulder	NEMA23	200 Steps\40Degree
Elbow	NEMA17	200 Steps\40Degree

These values were later incorporated into the Python control script to convert the computed inverse kinematics angles into the exact number of steps needed for each actuator.

The following Python snippet was used during early testing phases to verify motor movement and validate angle calculations:

```
import RPi.GPIO as GPIO
import time

# Setup GPIO pins
STEP = 7
DIR = 8
GPIO.setmode(GPIO.BCM)
GPIO.setup(STEP, GPIO.OUT)
GPIO.setup(DIR, GPIO.OUT)

# Set direction and step count
GPIO.output(DIR, GPIO.HIGH)  # Set rotation direction
steps = 200

for _ in range(steps):
    GPIO.output(STEP, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(STEP, GPIO.LOW)
    time.sleep(0.001)

GPIO.cleanup()
```

Figure III-17 Example Code Used for Manual Step Testing

This code played a crucial role in calibrating the system and confirming the relationship between motor steps and angular displacement.

7. Geometric Shape Recognition with OpenCV

During our project, we learned how to detect geometric shapes using the OpenCV library. We started by converting the image from the camera into a grayscale image to simplify processing, then applied the Canny algorithm to detect edges. After that, we used the findContours function to extract the boundaries of objects, and then approximated each shape using approxPolyDP to determine the number of vertices.

All these functions and algorithms are available within the OpenCV library, and we simply invoked them in our program.

The number of vertices was the main criterion for determining the type of shape:

Three vertices → Triangle

Four vertices with a length-to-width ratio ≈ 1 → Square, otherwise → Rectangle

More than 7 vertices → Circle

Note: The minimum number of vertices used to detect a circle can be increased (e.g., more than 15) to improve accuracy, but since the shapes in our case did not exceed 7 vertices, we preferred this threshold for better simplicity and smooth performance.

Using this approach, we were able to programmatically classify shapes within the camera frame and link this system with the robotic arm's movement, so that it only responds when a specific shape, like a circle, is detected. This development enabled us to build an intelligent system capable of interacting with its environment based on visual data.

8. Color Detection and Integration with Shape Recognition

We also learned how to detect the colors of objects in an image using the HSV color space (Hue, Saturation, Value), as it provides higher accuracy and flexibility compared to the BGR space, especially under varying lighting conditions. We converted the captured image from the camera to the HSV color space using the cv2.cvtColor() function, then created a mask for each object using the cv2.drawContours() function to accurately isolate the color region.

After isolating the object's region, we extracted the average values of Hue, Saturation, and Value using the cv2.mean() function. These functions and algorithms are readily available through the OpenCV library.

We relied mainly on the Hue value to classify colors, following these intervals:

- $H \in [0,10]$ or $[160,180]$ → Red

- H ∈ [11,25] → Orange
- H ∈ [26,34] → Yellow
- H ∈ [35,85] → Green
- H ∈ [86,125] → Blue
- H ∈ [126,160] → Purple

By also analyzing saturation and brightness, we were able to distinguish between white, gray, and black.

After mastering color detection, we integrated the color detection system with the shape detection module we had previously developed. This allowed our system not only to identify the shape (e.g., circle or square) but also to recognize its color (e.g., gray, red, etc.). As a result, we could program the robotic arm to respond only when a specific shape with a specific color was detected — for example, a gray circle.

This integration of both shape and color recognition enabled us to design a more intelligent and responsive system, capable of making decisions based on a broader set of visual features — an essential step towards developing more autonomous and effective robots.

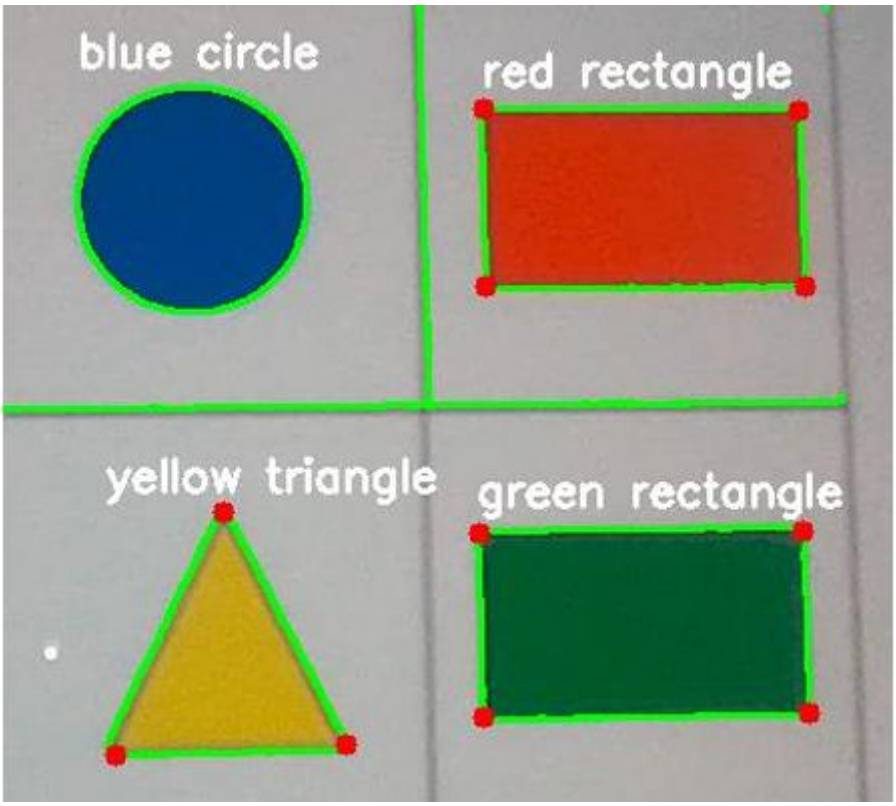


Figure III-18 Detecting the color and shapes

9. Converting Camera Coordinates to Real-World Coordinates Using Homography

We faced a key requirement: converting the detected object's position in the camera image (2D pixel coordinates) into real-world coordinates that the robotic arm can use for movement. To achieve this, we used a mathematical technique called Homography, which is a projective transformation used to map points between two different planes—such as the camera frame and the physical workspace.

We began by selecting four reference points from the camera image (typically the corners of the visible frame) and matched them with their corresponding real-world positions. Then, using OpenCV's cv2.findHomography() function, we calculated a 3×3 transformation matrix that mathematically describes the relationship between the two planes.

Once an object like a circle was detected in the camera frame, we used cv2.perspectiveTransform() to convert its center coordinates from the image space to accurate real-world coordinates. This allowed the robotic arm to understand the object's actual location and move precisely toward it in the real world.

This technique is essential in computer vision and robotics, as it enables intelligent interaction between what the camera sees and how the robot reacts. It makes the system more autonomous and efficient when interacting with its surrounding environment.

10. Converting Real-World Coordinates to Robotic Arm Angles (Inverse Kinematics)

In robotic arm systems, determining the angles of joints based on a target position is accomplished using inverse kinematics. In our project, once the real-world coordinates (X, Y) of the detected object are obtained, we calculate the necessary joint angles—base rotation (θ_1), shoulder angle (θ_2), and elbow angle (θ_3)—to reach that target point.

1. Base Rotation (θ_1):
   Since the camera captures images in a 2D plane, the base angle is calculated proportionally to the object's horizontal position in the frame using a linear mapping:

- $\theta^1 = \left(\frac{cx-160}{159}\right) \cdot 90$ *(2)*

where cx is the x-coordinate of the object in the image.

2. Distance to Target (d):
   We calculate the straight-line distance from the base to the target using the Pythagorean theorem:

- $x = \sqrt{X^2 + Y^2}$ *(3)*
- $y = H$
- $d = \sqrt{x^2 + y^2}$ *(4)*

where H is the vertical offset of the arm's base.

3. Elbow Angle (θ_3):

   Using the Law of Cosines, we determine the angle at the elbow joint:

- $cos(\theta_3) = \frac{L_1^2 + L_2^2 - d^2}{2 \cdot L_1^2 \cdot L_2^2}$ *(5)*

The result is clamped between -1 and 1, then converted to degrees.

4. Shoulder Angle (θ_2):

   Also using trigonometry, we calculate the shoulder angle as the sum of two angles:

   α: The angle from the base to the target.

   β: The angle formed between the first segment and the line to the target.

- $\alpha = tan^{-1}\left(\frac{y}{x}\right)$ *(6)*
- $\beta = cos^{-1}\left(\frac{L_1^2 + d^2 - L_2^2}{2 \cdot L_1 \cdot d}\right)$ *(7)*
- $\theta 2 = \alpha + \beta$ *(8)*

10.1. Mechanical Protection of The Arm Within the Code

We defined strict maximum angle limits for each joint to ensure safe movement and prevent mechanical collisions or motor overloading. Specifically, we set the following constraints:

- Shoulder angle (θ_2): must not exceed 100°
- Elbow angle (θ_3): must not exceed 80°

These constraints were embedded into the code logic. After calculating the elbow angle θ_3 using the law of cosines:

- $cos(\theta_3) = \frac{L_1^2 + L_2^2 + d^2}{2 \cdot L_1 \cdot L_2}$ *(9)*

we check whether θ_3 exceeds the limit of 80°. If it does, the system fixes θ_3 at 80° and recomputes the shoulder angle θ_2 to compensate. This is done by adjusting the calculation of θ_2 using:

- $cos(\beta) = \frac{L_1^2+d^2-L_2^2-2\cdot L_1^2\cdot L_2^2\cdot cos(\theta_3)}{2\cdot L_1\cdot d}$ *(10)*

Then we calculate:

- $\theta^2 = \alpha + \beta$ *(11)*

If this adjusted θ_2 still exceeds the maximum limit of 100°, the movement is aborted, and the system disregards the target point as it lies beyond the reachable workspace of the arm.

This approach ensures safe and flexible operation, allowing the system to intelligently adapt to its environment while protecting the mechanical integrity of the robotic arm.

11. Converting Angles to Steps and Controlling Stepper Motors

After calculating the required joint angles for the robotic arm (θ_1, θ_2, θ_3) using inverse kinematics, it was necessary to convert these angles into a number of steps that the stepper motors (Nema 17) could physically execute.

Through prior testing, we discovered the relationship between the number of steps and the angle of rotation for each joint:

- 200 steps for the base motor correspond to a 35° rotation
- 200 steps for both the shoulder and elbow motors correspond to a 40° rotation

Thus, to convert an angle θ to steps (s), we used the following equations:

- $s_1 = \theta_1 \cdot \left(\frac{200}{35}\right)$ *(12)*
- $s_2 = \theta_2 \cdot \left(\frac{200}{40}\right)$ *(13)*
- $s_3 = \theta_3 \cdot \left(\frac{200}{40}\right)$ *(14)*

After converting the angles to steps, we sent commands to the motors using the move_motor function. This function sets the rotation direction (DIR) and generates the required number of pulses (STEP) to move the motor. This implementation allowed the robotic arm to move accurately to the computed angles.

12. Coordinated Motion Loop and System Reset

After reaching the target, the system pauses for a short duration (time.sleep(5)) to allow any mechanical interaction, such as touching the object or verifying the position.

This duration can be adjusted by changing the number 5 to any desired number of seconds.

Then, reverse commands are sent with the same number of steps but in the opposite direction, in order to return the robotic arm to its original position (home point).
Once the arm returns, the system is re-initialized to analyze the image again and wait for the detection of a new stable shape.

This sequence represents the core of the interactive system, where the loop between computer vision and mechanical motion is closed in a structured and precise manner.

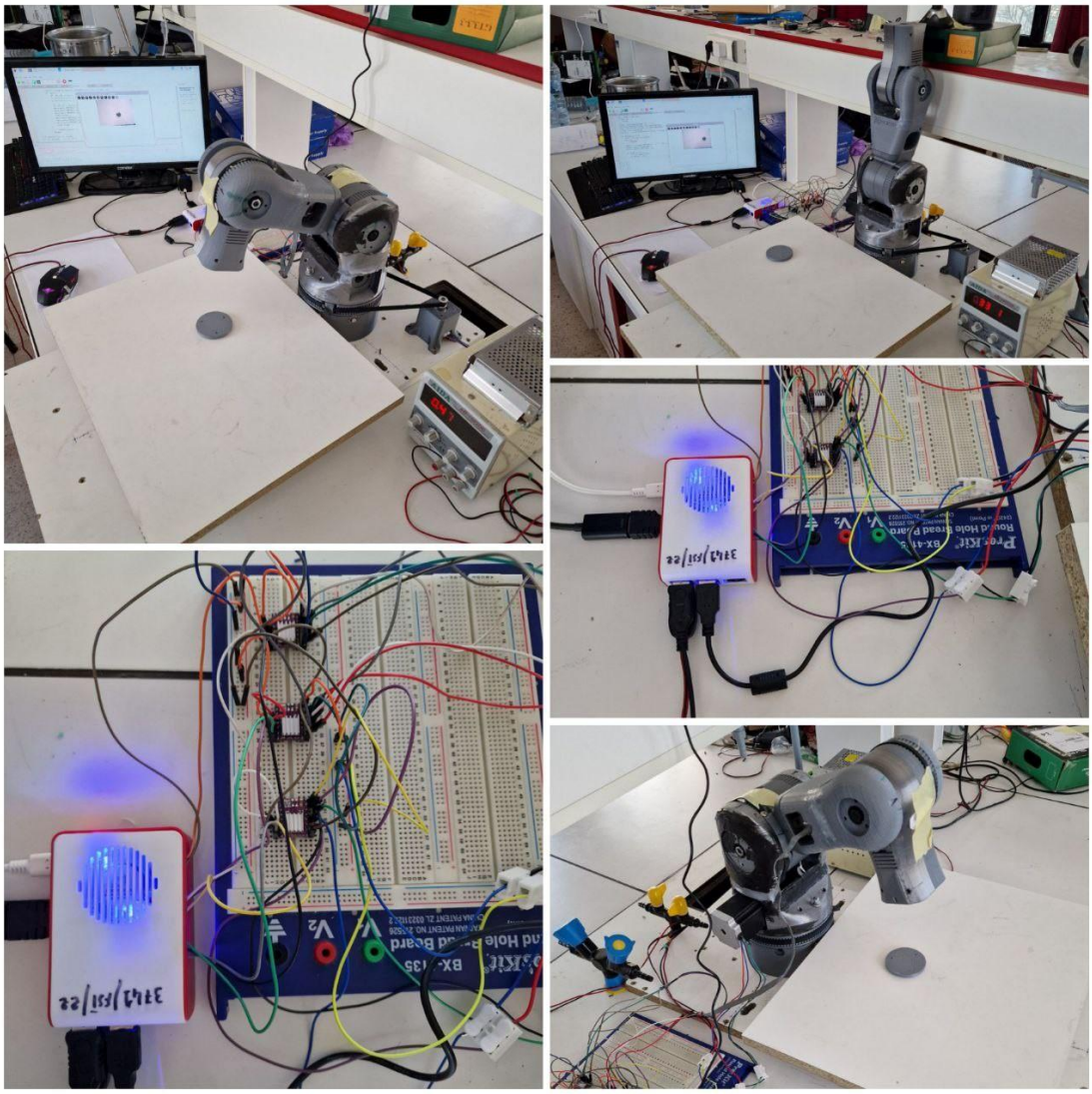


Figure III-19 Some Images from the Practical Aspect of the Project"

13. Conclusion

In summary, this chapter has provided a thorough overview of the working paradigm of the robotic arm from shape being visually recognized to the motor commands being executed in the real world. The application of computer vision through sophisticated software, combined with kinematic calculations and computer-controlled motors has demonstrated how a smart system can dynamically agent the world around it. Both the combination of image processing, coordinate transforming, angle calculating, and motor executing allows our robotic system to demonstrate an easy and efficient service. Overall, we have demonstrated both the functionality of our robotic design, and established a basis for autonomous and adaptive robotic systems for research to extend upon in the future.

GENERAL CONCLUSION

General conclusion

As modern technology continues to advance rapidly, the demand for embedded intelligent systems capable of real-time operating increases across numerous sectors, including robotics. This project is a practical implementation in this area, as an intelligent robotic arm was developed with image analysis capabilities that can visual recognize shapes and interact with them, all based on artificial intelligence (AI) and computer vision (CV) technologies, in an embedded environment using a Raspberry Pi.

We deployed optimized AI models designed for hardware constraints (memory and processing power) to perform a complex task autonomously, but intelligently. The combination of camera-based visual processing with stepper motor control contributed to the realization of a complete system that interacts with its environment, semi-autonomously. In order to reach the final functional prototype that succeeds in its intended goals, many technical and software challenges needed to be addressed.

We were able to combine the theoretical piece (artificial intelligence and computer vision), with practical piece (hardware control), through this experience, and we enjoyed the process of understanding how to design and realize embedded intelligent systems in real time. We learned how to analyze images, calculate coordinates, convert coordinates to angles, and translate angles to actual motor commands.

Not only does this project signify the end of a graduation thesis but also the start of an ongoing journey of research or work full of possibilities. The project could be expanded in many ways such as:

Improving the vision algorithms to detect more complex shapes and colours,

Integrating more advanced machine learning methods such as deep neural networks,

Or creating a multi-task robotic platform to operate in industrial or service-based environments.

What has been developed in this thesis represents the actual possibility of embedded artificial intelligence working in robotics and opens the possibility of smarter, more interactive applications in the near future.

bibliographie

[1] Interaction Design Foundation - IxDF. (2016, December 19). What is Artificial Intelligence (AI)?. Interaction Design Foundation - IxDF. https://www.interaction-design.org/literature/topics/ai

[2] Innovate Forge. (2023, November 23). *Understanding ANI, AGI, and ASI in artificial intelligence*. Medium. https://medium.com/@InnovateForge/understanding-ani-agi-and-asi-in-artificial-intelligence-6265d01091a1

[3] Viso.ai. (n.d.). *The 3 types of artificial intelligence: ANI, AGI, and ASI(February 13th, 2024)* . https://viso.ai/deep-learning/artificial-intelligence-types/

[4] Precog.co. (n.d.). *Industrial AI*. https://precog.co/glossary/industrial-ai/

[5] SparkIntelligence Ltd. (n.d.). *Computer Vision Applications*. https://sparkintelligence.co.uk/computer-vision-applications/

[6] Ashtari, H. (2023, October 12). *What are embedded systems?* Spiceworks. https://www.spiceworks.com/tech/tech-general/articles/what-are-embedded-systems/

[7] Siddell, K. (2023, October 31). *Computer vision will be essential to the Industry 5.0 era*. alwaysAI. https://alwaysai.co/blog/industry5.0/

[8] Van Rossum, G., & Drake, F. L. (2009). *The Python language reference manual*. Network Theory Ltd https://dl.acm.org/doi/10.5555/1593511

[9] Stroustrup, B. (2013). *The C++ programming language* (4th ed.). Addison-Wesley. https://dl.acm.org/doi/10.1145/234286.1057836

[10] Arnold, K., Gosling, J., & Holmes, D. (2005). *The Java programming language* (4th ed.). Addison-Wesley. https://doi.org/10.5555/1076806

[11] MathWorks. (2020). *MATLAB and Simulink documentation*. MathWorks. https://www.mathworks.com/help/matlab/

[12] Klabnik, S., & Nichols, C. (2019). *The Rust programming language* (2nd ed.). No Starch Press. https://doc.rust-lang.org/book/

[13] BCN3D. (n.d.). *BCN3D MOVEO – A fully OpenSource 3D printed Robot Arm* [Thing #1693444]. Thingiverse. https://www.thingiverse.com/thing:1693444

[14]BCN3D Technologies. (2016). *BCN3D Moveo - A fully Open Source 3D printed robot arm*. GitHub. Retrieved from https://github.com/BCN3D/BCN3D-Moveo

[15]Raspberry Pi Foundation. (n.d.-a). *Raspberry Pi 4 Model B*. Retrieved from https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[16] Wu, E. (2020, May 28). *Meet the brand new Raspberry Pi 4 8GB*. Seeed Studio. https://www.seeedstudio.com/blog/2020/05/28/meet-the-brand-new-raspberry-pi-4-8gb-ram/

[17]Raspberry Pi Foundation. (n.d.-b). *Raspberry Pi OS*. Retrieved from https://www.raspberrypi.com/software/

[18]GeeksforGeeks. (2025, June 25). *Python Tutorial | Learn Python Programming Language*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python/python-programming-language-tutorial

[19]RaspberryTips. (2023). *How to Use Thonny IDE on Raspberry Pi?* Retrieved from https://raspberrytips.fr/thonny-ide-raspberry-pi/