THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

CAPSTONE PROJECT

MAJOR: MECHATRONICS ENGINEGERIN

PROJECT TITLE:

USING 4-DOF ROBOTIC ARM TO CLASSIFY FAULTY ELECTRONICS CIRCUITS AND APPLYING MACHINE LEARNING TO DETECT FAULTS

Supervisor : **Dr. DOAN LE ANH**
Co-Supervisor : **NGUYEN KHUONG QUYNH**
Reviewer : **Dr. PHAM ANH DUC**
Students : **NGO TRUONG HOANG TIEN**
DOAN CONG TIN
Student ID : **101200248**
101200293
Class : **20CDT1**
20CDT2

Da Nang, 6/2025

SUMMARY

Topic: Using 4-Dof robotic arm to classify faulty electronic circuits and applying Machine Learning to detect faults

Students: Ngo Truong Hoang Tien – Doan Cong Tin

Class: 20CDT1 – 20CDT2

Supervisor: Dr. Doan Le Anh

Reviewer:

1. Actual demand:

In the context of automation and efficiency enhancement in electronics manufacturing, the rapid and accurate inspection and classification of faulty electronic circuits is crucial. Traditional manual inspection methods are often time-consuming, prone to errors, and struggle to meet the increasing demands for product quality. Particularly for small and medium-sized enterprises, accessing complex and expensive automated inspection systems poses a significant challenge.

Therefore, the research, design, and fabrication of a system utilizing a 4-Degree-of-Freedom (4-Dof) robotic arm to automate the inspection and classification of faulty electronic circuits, combined with Machine Learning algorithms to enhance the accuracy and efficiency of fault detection, is a highly practical and widely applicable approach.

2. Scope of research topic

- Research on the operating principles and applications of a 4-Dof robotic arm in electronic component inspection and manipulation tasks.
- Investigation of suitable Machine Learning algorithms for classifying different types of faults in electronic circuits based on collected data.
- Mechanical design and integration of the robotic arm with sensor systems and necessary auxiliary components for the inspection process.
- Development of control interfaces and software for data acquisition, Machine Learning model training, and robotic arm control.
- Establishment of an automated fault inspection and classification process.
- Evaluation of the system's accuracy, speed, and efficiency in detecting and classifying faulty electronic circuits.

3. The content of the topic has been done

- Survey of existing methods for inspecting and classifying electronic circuit faults, including both manual and automated approaches.

- Research on various industrial robotic arms and selection of a 4-Dof robotic arm suitable for the application.
- Investigation and selection of appropriate Machine Learning algorithms for the fault classification problem (e.g., Support Vector Machines, Neural Networks, Decision Trees, etc.).
- Preliminary system design, including the integration of the robotic arm, sensor systems (e.g., camera, current/voltage sensors, etc.), and other components.
- Development of a data acquisition process for collecting data on faulty and non-faulty electronic circuits for Machine Learning model training.
- Development of a robotic arm control program to perform inspection tasks (e.g., scanning, contacting, etc.).
- Building and training a Machine Learning model to classify different types of faults.
- Conducting experiments and evaluating the system's performance.

4. Results

- A prototype system utilizing a 4-Dof robotic arm for performing electronic circuit inspection tasks has been developed.
- A Machine Learning model capable of classifying faulty electronic circuits with high accuracy has been successfully developed (specify the expected or achieved accuracy).
- The system demonstrates the potential to perform fault inspection and classification faster than manual methods.
- An automated process for inspecting and classifying electronic circuit faults has been proposed, with potential for application in real-world manufacturing.
- The (expected) cost of building the system is potentially more competitive than existing commercial automated inspection systems (if applicable).
- The model has the potential for expansion to detect more types of faults or integrate other features (e.g., detailed fault reporting, test data storage, etc.).

THE UNIVERSITY OF DANANG
University of Science and Technology
Faculty of Mechanical Engineering

SOCIALIST REPUBLIC OF VIETNAM
Independent – Freedom - Happiness

TASK DISTRIBUTION FOR CAPSTONE PROJECT

Full.	Student's name	Students ID	Class	Major
1	Ngo Truong Hoang Tien	101200248	20CDT1	Mechatronics engineering
2	Doan Cong Tin	101200293	20CDT2	Mechatronics engineering

1. *Topic: Using 4-Dof robotic arm to classify faulty electronic circuits and applying Machine Learning to detect faults*
2. *Topics covered by:* ☐ *Having signed an intellectual property agreement for the implementation results*
3. *Initial figures and data:*
4. *Contents of explanations and calculations:*

a. General part:

Full.	Student's name	Contents
1	Ngo Truong Hoang Tien	Research the topic, choose ideas and options, find reference sources, present the content as required and edit the content
2	Doan Cong Tin	

b. Private part:

Full.	Student's name	Contents
1	Ngo Truong Hoang Tien	Chapter 1 Overview of the topic Chapter 2 Analysis and select design option Chapter 4 Control system design Chapter 6 Conclusions and future works

2	Doan Cong Tin	Chapter 2 Analysis and select design option Chapter 3 Components in the system Chapter 4 Mechanical design and Manufacturing Chapter 5 Control system design

5. Drawings, graphs (specify types and sizes of drawings):

a. General part:

Full.	Student's name	Contents	Quantity
1	Ngo Truong Hoang Tien	Overall system drawing	01:A0
2	Doan Cong Tin		

b. Private part:

Full.	Student's name	Contents	Quantity
1	Ngo Truong Hoang Tien	Electrical diagram drawing	01:A0
		Algorithm flowchart drawing	01:A0
2	Doan Cong Tin	Product structure drawing	01:A0
		Detail drawing & Detail Breakdown	03:A0
		Dynamic diagram drawing	01:A0

6. Supervisor: Dr. Doan Le Anh

7. Project assignment date:

8. Project completion date:

Da Nang, May.....,2025

Head of Department ……. **Supervisor**

Dr. Doan Le Anh

PREFACE

In the context of increasing automation and the need to enhance efficiency in electronics manufacturing, the application of automated systems for inspecting and classifying faulty electronic circuits has become extremely important. Traditional manual inspection methods are often time-consuming, prone to errors, and struggle to meet the growing demands for product quality. Optimizing the fault inspection and classification process to ensure speed, accuracy, and cost reduction is a crucial requirement.

Stemming from this practical need, the project "**Using 4-Dof robotic arm to classify faulty electronic circuits and applying Machine Learning to detect faults**" was carried out to research and develop an automated system for inspecting and classifying electronic circuit faults. In this context, the integration of a 4-Degree-of-Freedom (4-Dof) robotic arm for manipulation and data acquisition, combined with Machine Learning algorithms for analyzing and identifying different types of faults, plays a pivotal role in enhancing the system's efficiency and accuracy.

The project focuses on the main contents including: Research on the application of a 4-Dof robotic arm in electronic circuit inspection, establishing a fault data acquisition process, developing and training a Machine Learning model for fault classification, system integration, and performance evaluation through experiments. Through this, the project not only helps students access advanced technologies in robotics and artificial intelligence but also brings practical value in automating the quality control process for electronic products.

Despite the significant efforts made during the implementation process, due to limitations in time and knowledge, the project may inevitably have shortcomings. We sincerely look forward to receiving comments from our teachers and peers to further improve the project.

We would like to sincerely thank the faculty members in the Faculty and Department of Mechatronics, especially Dr. Doan Le Anh for their dedicated guidance and support throughout the implementation of this project.

Sincerely,

Ngo Truong Hoang Tien – Doan Cong Tin

Class 20CDT – Faculty of Mechanical Engineering

ASSURE

We would like to assure you that the entire content of the project "Using 4-Dof robotic arm to classify faulty electronic circuits and applying Machine Learning to detect faults" is the result of our own research and work, under the guidance of Instructor Dr. Doan Le Anh and instructor at AUE Company. Mr. Nguyen Khuong Quynh, Engineer of Technical Department.

In the process of implementing the project, we have referred to relevant documents and research works. Contents, images, and data cited from other sources are clearly stated in the References section.

We would like to take full responsibility for the honesty and accuracy of this project. If there are any errors or violations related to intellectual property rights, we would like to take full responsibility before the university and the applicable regulations.

Implementation group

Ngo Truong Hoang Tien – Doan Cong Tin

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATION LIST

Abbrevitation	Full Form (if known)	Description
DOF	Degree of Freedoom	Number of independent movements of a system
PWM	Pulse Width Modulation	Method to control signal by varying pulse width
DC	Direct Current	Electrical current that flows in one direction only
AC	Alternating Current	Electrical current that changes direction periodically
STL	Stereolithography	File format used for 3D printing
3D	Three-Dimensional	Having three dimensions: length, width, and height
YOLOv8	You Only Look Once version 8	Real-time object detection AI model
IoU	Intersection over Union	Overlap between predicted box and ground truth (Area of Overlap ÷ Area of Union)

CHAPTER 1 TOPIC OVERVIEW

1.1 Introduction of topic:

1.1.1 History of formation and development:

The term "Robot" originates from the Czech word "Robota," which means "menial labor," and was first introduced in the 1921 play Rossum's Universal Robots by Karel Čapek. In this play, Rossum and his son created human-like machines to serve mankind. This could be seen as an early inspiration for engineers to invent mechanisms and machines that mimic human muscular actions.

In the early 1960s, the American company AMF (American Machine and Foundry Company) introduced a versatile automatic machine, which they named the "Industrial Robot." Today, the term "industrial robot" refers to devices that resemble a human arm and are automatically controlled to perform specific manufacturing operations.

Technically, modern industrial robots originated from two earlier fields: remote control mechanisms (teleoperators) and numerically controlled machine tools (NC machines). Teleoperators—also known as master-slave devices—were widely developed during World War II for the purpose of handling radioactive materials. The operator was separated from the hazardous area by a wall with one or more observation windows for visibility.

These remote-control systems replaced the operator's arm with a gripper mechanism (slave) on the inside and two control handles (master) on the outside. The gripper and handles were connected through a six-degree-of-freedom mechanism, allowing for free positioning and orientation of the arm and gripper. This mechanism enabled the gripper to follow the movement of the control handles.

Around 1949, numerically controlled (NC) machine tools were developed to meet the needs of the aerospace industry. The first industrial robots were essentially a combination of the mechanical structure of teleoperators with the programmable capabilities of NC machines.

Below is a brief overview of some key milestones in the development of industrial robots. One of the first industrial robots was the Versatran robot developed by AMF in the United States. Around the same time, the Unimate-1900 robot also appeared in the U.S. and was first used in the automotive industry. Following the U.S., other countries began manufacturing industrial robots: the UK in 1967; Sweden and Japan in

1968 under U.S. licensing; West Germany in 1971; France in 1972; and Italy in 1973. The performance of robots has continually improved, especially in recognition and processing capabilities.

In 1967, researchers at Stanford University (USA) developed a robot model based on the "eye-hand" coordination model. This robot could recognize and orienting the gripper to the position of an object using sensors. In 1974, the American company Cincinnati introduced the T3 robot (The Tomorrow Tool), which was computer-controlled and capable of lifting objects weighing up to 40 kg.

It can be said that robots are a fusion of the flexible functionality of teleoperators with the increasingly advanced "intelligence" of numerically controlled systems, along with sensor manufacturing technology, programming methods, and developments in artificial intelligence and expert systems. In recent years, the functional capabilities of robots have continued to evolve. Modern robots are equipped with a variety of sensors to detect their environment, and the rapid advances in information technology and electronics have led to the creation of robotic generations with more specialized features, lower costs, and greater deployment. As a result, industrial robots have become an essential part of modern production lines.

1.1.2 Development trends:

A robot is a product that integrates both science and technology with high complexity. Designing and building a robot requires knowledge in mathematics, mechanics, physics, electronics, control theory, computational science, and many other fields.

In the next 20 years, everyone will need a personal robot just as they need a computer today. Robots will be at the center of a major technological revolution after the Internet. With this trend, along with traditional applications of robots in industry, healthcare, education, entertainment, and especially in national security and defense, the robot market and related services will be immensely large.

Robots have made significant progress over the past half-century. The first robots were applied in industry in the 1960s to replace humans in performing heavy, dangerous tasks in hazardous environments. Due to the increasing demand for use in complex manufacturing processes, industrial robots now need to have more flexible and intelligent capabilities.

Figure 1.1 Robot applications in automated production.

Today, in addition to the initial application of robots in machine manufacturing, other applications such as in healthcare, medical care, agriculture, shipbuilding, construction, national security and defense, and household use are seeing increasing demand, driving the development of terrain robots and service robots.

Figure 1.2 Robots in Auto Manufacturing.

1.1.3 Classification of industrial robots:

Classification by Mechanical Structure:

Based on the structure of the manipulator, industrial robots can be categorized into the following types:

- Cartesian coordinate robots.
- Cylindrical coordinate robots.
- Spherical coordinate robots.
- Articulated (or angular coordinate) robots.
- SCARA (Selective Compliance Articulated Robot Arm) robots.

Classification by Drive System:

Common types of drive systems include:

- **Electric Drive Systems:** These typically use DC motors (Direct Current) or stepper motors. Electric drives are compact and easy to control, making them widely used in various applications.
- **Hydraulic Drive Systems:** These systems can deliver high power output and are suitable for heavy-duty tasks. However, they tend to be bulky and nonlinear in behavior, which complicates control strategies.
- **Pneumatic Drive Systems:** Pneumatic systems are lighter and more compact due to the absence of return lines, but they require a centralized compressed air source. These systems are typically used for low to medium power applications and have limited precision. They are often applied in pre-programmed robots performing simple tasks such as pick-and-place operations (Point to Point – PTP).

Figure 1.3 Robot Scara LS10-B803S / RC90-B.

Classification by Application:

Depending on their use in industrial production, robots can be classified as:

- Painting robots
- Welding robots
- Assembly robots
- Material handling robots
- And more

Classification by Control Method and Characteristics:

Robots can also be distinguished based on their control systems:

- **Open-loop control robots:** These systems operate without feedback mechanisms.
- **Closed-loop (or servo-controlled) robots:** These utilize sensors and feedback loops to improve control accuracy and flexibility.

1.1.4 Objectives of the project:

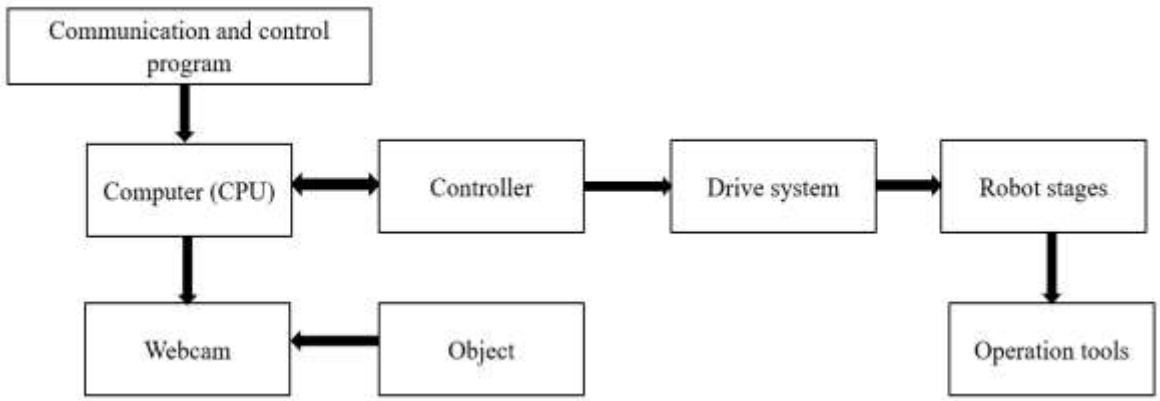

Figure 1.4 Block diagram of the system.

The system will have the following main components:

- A camera to capture images of the objects
- A conveyor belt long enough to transport the products
- A robotic arm to place the objects into the corresponding slots after they have been classified
- A computer and central controller to analyze the data from the camera and control the robotic arm

The task is to design and build a robotic arm model to classify objects on the conveyor belt. The computer, through the camera, will capture images of the objects on the conveyor belt, process the data to extract the necessary information about their shape, and then send this data to the controller. The controller will control the actuators, including the motors of the robotic arm, the conveyor motor, and other motors, to move the objects into the corresponding slots.

Some software that our student group will use for the project:

- SolidWorks 2022: used for 3D design of the system
- AutoCAD 2021: used for drawing 2D details, used in the cutting of acrylic
- Pycharm Community Edition 2024: used for processing the images sent from the camera
- Arduino IDE: used for programming the actuators.

1.1.5 Scope of Study

- Research on design options for a mechanical structure of a 4-DOF robotic arm capable of object manipulation and classification.
- Integration of object detection using camera and machine learning techniques (e.g., image classification) to identify faulty electronic circuits.
- Application of Arduino Mega for controlling the robotic arm instead of PLCs, utilizing stepper/servo motors, limit switches, and serial communication.
- Design and implementation on a small-scale educational model with potential to scale or adapt to industrial applications.
- The study does not focus on advanced human-machine interface (HMI) design or industrial communication protocols outside the core classification and control scope.

1.1.6 Scientific and Practical Significance

- Scientifically, the project consolidates and applies theoretical knowledge in embedded systems, electronics, image processing, machine learning, and robotic control. It reinforces interdisciplinary skills essential in modern automation, especially in vision-based classification systems and real-time robotic manipulation.

- Practically, the ability to classify defective electronic circuits using a robotic system integrated with machine learning can significantly improve quality control in electronics manufacturing. The model's low cost, flexibility, and educational value make it suitable for applications in academic laboratories or as a prototype for small production lines.

1.1.7 Basis for Topic Development

- Fundamental knowledge in Arduino programming, sensors, actuators, and mechanical design.
- Understanding of machine learning techniques, particularly image classification using models like CNNs (Convolutional Neural Networks).
- Skills in designing, building, and programming an integrated automation system involving computer vision, control electronics, and mechanical motion.

1.2 Overview of Related Studies and Similar Devices:

1.2.1 PCBNet – Lightweight Convolutional Neural Network for Defect Inspection in SMT:

Wu et al. (2022) introduced PCBNet, a lightweight convolutional neural network (CNN) specifically designed for defect inspection and detection in Surface Mount Technology (SMT). This model aims to balance performance, accuracy, and processing speed, making it suitable for embedded systems or devices with limited computational resources.

System architecture:

- Image preprocessing: Raw PCB images are processed to extract Regions of Interest (ROI), cropped and normalized in size, and noise is removed.
- Defect detection with CNN: ROI data is input into PCBNet, which employs multiple convolutional layers with a small number of parameters. The network learns to classify defects such as misplacement, soldering defects, missing components, or reversed placement.
- Result display and reporting: Detected defects are presented on a user-friendly Human–Machine Interface (HMI).

Results:

- Accuracy: PCBNet achieved superior accuracy on real-world SMT production dataset samples.

- Speed: Low image processing time, suitable for real-time applications.
- Main advantage: Compact model, easy to deploy, low hardware requirements, suitable for low-cost defect inspection applications.

Evaluation:

This study offers a practical and applicable solution in industrial environments, especially for small to medium enterprises lacking the resources to deploy complex AI models or powerful GPU devices.

1.2.2 Accurate Detection of High-Density PCB Components Based on Improved YOLOv8:

In cases where electronic PCBs have high component density, traditional methods or standard CNNs face limitations, especially regarding resolution and the ability to distinguish small, closely packed objects. The research team led by Si Shen et al. (2025) proposed the GS-YOLOv8s model, focusing on accurate defect detection on PCBs with high speed.

Model and main improvements:

- Network structure: The model is based on YOLOv8s but has been refined to reduce parameter count and enhance small object detection capability.
- Dual-model strategy: Two separate models are trained in parallel to learn different defect features. An integration module then combines their results to improve detection reliability.
- Dataset used: Provided by Lite-On Technology Co., Ltd, including images of PCBs with high component density and various defect types such as misplacement, missing components, and soldering defects.
- Experimental results:
- Defect detection accuracy: ≈ 95.6%
- Recall rate: ≈ 94.6%
- mAP@0.5: ≈ 97.4%
- Processing time per image: only 0.027 seconds — well-suited for real-time production line processing.
- Total model parameters: significantly reduced compared to original YOLOv8, suitable for deployment on edge devices.

Evaluation:

This model demonstrates a breakthrough in performance and speed, especially effective in environments with stringent production requirements such as the Electronics

Manufacturing Services (EMS) industry. The improvements from the original YOLOv8 also show the potential for scalable automatic inspection systems operating in real time without expensive hardware.

1.2.3 Mitsubishi RV-4FR (4 DOF Robot Series):

Table 1.1 Mitsubishi RV-4FR specifications.

Specification	Manufacturer	Payload	Reach	Repeatability
Parameter	Mitsubishi Electric (Japan)	4 kg	515 – 649 mm	±0.02 mm

Description:

The Mitsubishi RV-4FR robot series is an industrial robot with a 4-degree-of-freedom configuration, specially designed for automation applications requiring high precision and fast speed. The RV-4FR is suitable for tasks such as small assembly, part inspection, product handling, and operations requiring flexible manipulation in confined spaces. The robot can be mounted in various ways (floor, ceiling, wall) and easily integrated into modern production lines thanks to the advanced CR800-D controller. With high repeatability and durable operation, the RV-4FR meets the demands of light to medium industrial applications.

Advantages:

- High precision and stable operation
- Fast speed with short cycle times
- Compact design, flexible installation
- Industrial-grade environmental protection with IP40/IP67 rating
- Advanced controller supporting programming and integration

Disadvantages:

- Limited payload of only 4 kg, unsuitable for heavy tasks
- Medium to high cost, suitable for industrial applications requiring quality

Figure 1.5 Mitsubishi RV-4FR (4 DOF Series).

1.2.4 ABB IRB 1200 (4 DOF Customized Version):

Table 1.2 ABB IRB 1200 specifications

Specification	Manufacturer	Payload	Reach	Repeatability
Parameter	ABB (Switzerland)	5–7 kg	900 mm	±0.02 mm

Description:

The ABB IRB 1200 is a well-known industrial robot series recognized for its precision and durability. The customized 4 DOF version is widely used for simple tasks such as pick-and-place, polishing, inspection, and spot welding. This robot is suitable for production lines demanding high accuracy and quality. The RobotStudio software supports programming and simulation, facilitating easy integration into production systems.

Advantages:

- Very high accuracy.
- Robust and durable industrial design.
- Professional programming software support.

Disadvantages:

- Relatively high cost.
- Typically used only when very high quality is required.

Figure 1.6 ABB IRR 1200.

CHAPTER 2 ANALYSIS AND SELECTION OF DESIGN OPTIONS

2.1 Object Handling System Design Options:

2.1.1 Option 1: Manual Placement of Circuit Boards:

2.1.1.1 Operating Principle:

- The operator manually places each circuit board onto the conveyor belt. The image acquisition and classification tasks are then handled automatically.

2.1.1.2 Advantage:

- Extremely simple setup, no need for additional hardware.
- Low cost, easy to implement for student or lab projects.
- Allows focus on machine vision and classification algorithm development.

2.1.1.3 Shortcoming:

- Relies on human input; inconsistent placement may affect image accuracy.
- Not scalable for industrial production.

2.1.2 Option 2: Robot-Assisted Object Feeding:

2.1.2.1 Operating Principle:

- A secondary robot arm or automatic pusher places each circuit board onto the conveyor when space is detected via sensors.

2.1.2.2 Advantage:

- Partial automation reduces human error.
- Feeder can be synchronized with conveyor motion.

2.1.2.3 Shortcoming:

- Requires additional motors, sensors, and control logic.
- Increased complexity and cost for non-core functionality.

2.1.3 Option Selection: Option 1 – Manual Placement of Circuit Boards:

- Given the project's academic and experimental scope, Option 1 is selected for its simplicity, low cost, and suitability for focusing on the core objective—developing and testing the vision-based classification system. This option avoids the need for additional actuators or complex mechanical design, allowing the team to concentrate on software, image processing, and algorithm integration.

- However, the system will be designed with modularity in mind to allow future upgrades. In future stages or industrial applications, the manual placement can be replaced with automated feeding (Option 2) or other advanced handling mechanisms to enhance throughput and reduce human involvement.

2.2 Conveyor Transport Mechanism:

2.2.1 Option 1: DC Motor without Feedback:

2.2.1.1 Principle:

- The DC motor runs based on predefined ON/OFF commands or PWM (Pulse Width Modulation) control for speed regulation. There is no feedback to verify the motor's actual position or speed during operation.

2.2.1.2 Advantage:

- Inexpensive and simple.
- Easy to build and integrate.

2.2.1.3 Shortcoming:

- No speed/position control.
- Not suitable for synchronizing with robotic pick-and-place timing.

2.2.2 Option 2: Stepper Motor Drive:

2.2.2.1 Principle:

- The stepper motor moves in discrete steps based on input pulse signals. Each pulse corresponds to a fixed angular movement, allowing precise open-loop control of position and speed.

2.2.2.2 Advantage:

- Good control over position and speed without needing feedback.
- Easily interfaced with Arduino Mega and compatible libraries (e.g., AccelStepper).
- Allows for synchronized motion with other axes of the robotic arm.

2.2.2.3 Shortcoming:

- Lacks feedback mechanism, so it may lose steps under high load or acceleration.
- Not suitable for high-speed or high-torque applications.
- Efficiency may decrease due to constant current draw even at rest.

2.2.3 Option 3: Servo Motor with Encoder Feedback:

2.2.3.1 Operating Principle:

- A servo motor uses a built-in encoder to continuously measure position and speed. A closed-loop control system adjusts motor operation in real time to match desired setpoints.

2.2.3.2 Advantages:

- High-precision control of both speed and position.
- Closed-loop feedback ensures accurate movement, even under varying loads.
- Ideal for synchronized operation with the robotic arm or coordinated motion tasks.

2.2.3.3 Shortcomings:

- Higher cost compared to stepper or simple DC motors.
- Requires specific drivers and careful PID tuning for optimal performance.
- May increase system complexity in student-level or prototype projects.

2.2.4 Compare Option:

Table 2.1 Comparison of transportation conveyor options

Criteria	**Regular engine**	**Stepper Motor**	**Servo Motor (encoder)**
Precise speed control	Low	Average	Very high
Engine synchronization capability	Not suitable	Can	Very suitable
Speed/Position Response	Without	Without	With encoder
Suitable for research models	Cheap, simple	Easy to use	Very suitable
Cost	Cheap	Average	Higher

2.2.5 Option selection: Option 1: DC Motor without Feedback

- For the initial phase of the project, Option 1 is selected due to its simplicity, ease of implementation, and low cost—factors that are critical for student-level or proof-of-concept systems. The absence of feedback is acceptable for non-critical motion tasks, such as activating the vacuum pump or other binary control components.
- This choice allows the team to allocate more resources and time to essential subsystems like vision processing, object classification, and robotic arm coordination. Nevertheless, the system design remains flexible, and future versions can be upgraded to include stepper motors or servo motors with feedback if higher accuracy or motion reliability becomes necessary.

2.3 Classification and Sorting Mechanism:

2.3.1 Option 1: Static Camera with Image Classification:

2.3.1.1 Principle:

- A fixed camera above the conveyor captures images of circuit boards as they pass underneath. The computer runs a machine learning model (e.g., YOLOv8) to detect defects.

2.3.1.2 Advantage:

- High speed and non-contact detection.
- Easily trained and updated with new data.

2.3.1.3 Shortcoming:

- Requires careful lighting and positioning.
- Sensitive to motion blur if conveyor speed is too high.

2.3.2 Option 2: Robot Arm with Vision-Guided Manipulation:

2.3.2.1 Principle:

- A 4-DOF robotic arm receives classification results and picks up the board, placing it into the correct bin (OK or Defective).

2.3.2.2 Advantage:

- Automated sorting with precise placement.
- Visually intuitive and demonstrable.

2.3.2.3 Shortcoming:

- Requires accurate inverse kinematics and synchronization.

2.3.3 Selected Option: Fixed camera + 4-DOF robotic arm sorting

- A hybrid approach combining Option 1 and Option 2 is selected. The fixed camera handles image classification using machine learning, while the robotic arm is responsible for executing the sorting based on classification results. This division of labor improves system modularity, simplifies debugging and development, and allows independent enhancement of the vision and control subsystems. It also aligns well with the educational goals of integrating computer vision with robotic manipulation.

2.4 Control System Architecture:

2.4.1 Option 1: PLC-Based Control:

2.4.1.1 Operating Principle:

- A Programmable Logic Controller (PLC) is used to manage the sequence of operations, typically programmed using ladder logic or structured text. It handles I/O operations, timers, and logic control.

2.4.1.2 Advantage:

- Industrial-grade reliability and durability.
- Widely used and recognized in industrial automation environments.
- Excellent for deterministic control tasks like timing and safety interlocks.

2.4.1.3 Shortcoming:

- Limited flexibility for integrating computer vision or AI-based decision-making.
- Not well suited for high-level programming tasks or interfacing with modern machine learning frameworks.

2.4.2 Option 2: Arduino Mega-Based Control:

2.4.2.1 Operating Principle:

- Arduino Mega controls the robotic arm, motors, and actuators via I/O pins, while receiving classification results from a PC through serial communication. It executes control logic written in C/C++.

2.4.2.2 Advantage:

- Flexible and easy to program in C/C++.
- Well-supported in academic and hobbyist environments.
- Compatible with a variety of motor drivers, sensors, and communication protocols.

- Low cost, making it ideal for prototyping and student projects.

2.4.2.3 Shortcoming:

- Less robust compared to PLCs in industrial conditions.
- Lacks real-time operating system; precise timing for simultaneous multi-axis motion requires careful software design.
- Limited native support for advanced communication protocols (e.g., EtherCAT, Modbus-TCP).

2.4.3 Summary of Design Choices:

Component	Selected Option	Rationale
Object Feeding	Manual Placement	Simplicity, cost-saving, focus on vision system
Conveyor Drive	Stepper Motor	Sufficient control for small-scale, affordable
Vision System	Fixed Camera + ML (YOLOv8)	Fast, accurate, and flexible
Sorting Mechanism	4-DOF Robotic Arm	Precise, demonstrative, and programmable
Control System	Arduino Mega	Flexible, easy integration with Python + ML pipeline

2.4.4 Option Selection: Arduino Mega-Based Control:

The Arduino Mega is selected as the main controller for this graduation project due to its suitability for academic environments. It offers a good balance between functionality and simplicity, allowing students to focus on system integration, robotic motion control, and communication with the computer vision module. Its low cost, community support, and compatibility with stepper/servo motors make it a practical and accessible choice for prototyping and demonstrating key concepts in automation and intelligent systems.

CHAPTER 3 COMPONENTS IN THE SYSTEM

3.1 Driver TB6600:

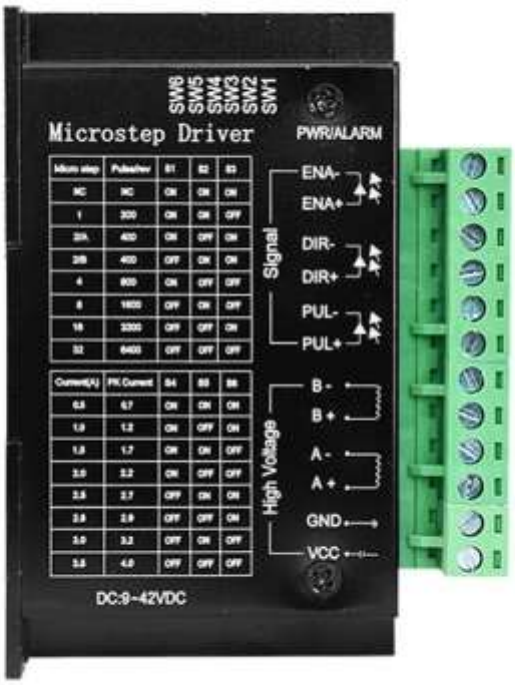

Figure 3.1 TB6600 Stepper Motor Driver.

Table 3.1 TB6600 Stepper Motor Driver specification sheet

Parameter	**Specification**
Operating Voltage	9V – 24V DC
Output Current	Adjustable from 0.5A to 4.0A
Motor Type	2-phase Stepper Motor
Control Signals	Step (Pulse), DIR (Direction), EN (Enable – Optional)
Microstepping Modes	Full, 1/2, 1/4, 1/8, 1/16, 1/32
Maximum Pulse Frequency	200kHz
Logic Signal Voltage	3.3V – 24V

Protections	Overheat, Overcurrent, Short Circuit
Cooling Method	Aluminum heatsinks
Operating Temperature	$-10°C$ to $+ 45°C$
Storage Temperature	$-40°C$ to $+ 70°C$

- The TB6600 is a versatile and high-performance stepper motor driver designed to control 2-phase stepper motors with precision and reliability. It supports pulse (STEP), direction (DIR), and enable (EN) control signals, allowing easy integration with microcontrollers like Arduino or industrial controllers such as PLCs. The driver responds to each input pulse by moving the motor one step or microstep, depending on the configured microstepping mode. Direction signals determine the rotation direction, while the optional enable signal allows users to turn the motor output on or off as needed.
- TB6600 features adjustable current output up to 4.0A, which can be configured via DIP switches. This ensures sufficient torque delivery while minimizing heat and vibration. The driver also supports various microstepping resolutions—ranging from full step to 1/32 microstep—allowing smoother and more accurate motion control. Its wide logic voltage range (3.3V to 24V) makes it compatible with both low-voltage and industrial control systems.
- To ensure safe operation, the TB6600 includes built-in protection features such as overcurrent protection, overheating shutdown, and short-circuit prevention. These protections make it suitable for demanding applications like CNC machines, 3D printers, and robotic arms, where reliability and precision are essential.

3.2 Stepper Motor Nema 17:

Figure 3.2 NEMA 17 Stepper Motor.

Table 3.2 Stepper Motor Nema 17 Specification Sheet.

Parameter	**Specification**
Motor Type	Bipolar Stepper Motor
Step Angle	1.8° per step (200 Steps per revolution)
Rated Voltage	2.8V – 3.6V (typical)
Rated Current	1.0A – 2.0A per phase (depends on model)
Holding Torque	40N.cm – 59N.cm (depends on model)
Number of Phases	2
Resistance per Phase	1.1 – 3.5 ohms
Shaft Diameter	5 mm
Motor Size	42 x 42 mm (1.7 x 1.7 inches)
Weight	280g – 350g
Wiring Configuration	4-wire (bipolar) or 6-wire (unipolar)
Max Temperature Rise	80℃
Motor Length	34 mm – 48 mm

The NEMA 17 stepper motor works by converting electrical pulses into precise mechanical steps. It consists of a stator with electromagnetic coils and a rotor with teeth.

When current flows through the coils in a specific sequence, it creates a rotating magnetic field that pulls the rotor into alignment.

Each pulse moves the rotor by a fixed angle—typically 1.8° per step—allowing accurate position control without feedback sensors. The direction is determined by the order of coil activation, while speed depends on pulse frequency. This makes it ideal for applications like 3D printers, CNC machines, and robotics.

3.3 Module Relay 5V:

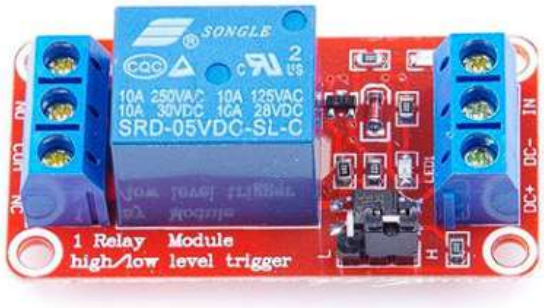

Figure 3.3 Module Relay 5V.

Table 3.3 Module Relay 5V sheet.

Parameter	**Specification**
Operating Voltage	5V DC
Trigger Voltage	< 1.5V (Low-level Trigger)
Relay Type	Single Pole Double Throw, Electromechanical
Isolation	Optocoupler (Photocoupler)
Max Switching Voltage	250VAC / 30VDC
Max Switching Current	10A
Control Voltage	3.3V – 5V
Input Pin Logic	Active LOW
Response Time	< 10ms
Indicator	Onboard LED for relay status
Terminals	Screw terminals: NO, NC, COM
Board size	50mm x 26mm

3.4 L298N Dual H-Bridge DC Motor Driver Module

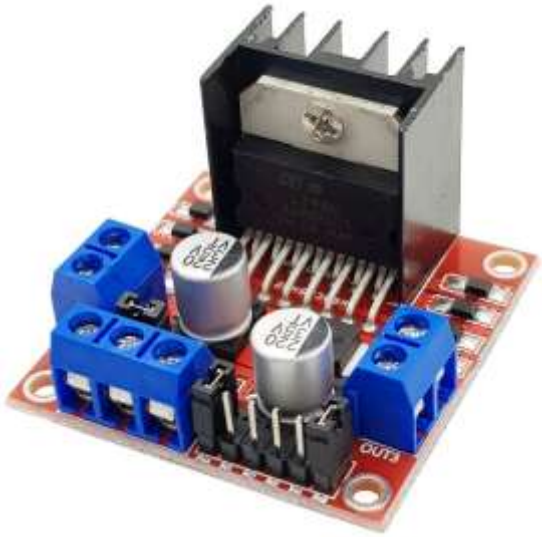

Figure 3.4 Module L298N.

Table 3.4 Module L298N specifications.

Parameter	**Specification**
Operating Voltage (Logic)	5V
Motor Supply Voltage	5V – 35V
Logic Supply Voltage	5V
Max Output Current per Channel	2A continuous
Total Output Power	25W
Number of Channels	2
Logic Input Current	0.6mA
Control Interface	TTL Logic (IN1, IN2, IN3, IN4, ENA, ENB)
Speed Control	PWM
Dimensions	43 x 43 x 27 mm

3.5 SMC Vacuum Suction Cup DP20:

Figure 3.5 SMC DP20 Vacuum Suction Cup.

SMC DP20 Vacuum Suction Cup is a compact and durable vacuum pad designed for handling flat or slightly curved surfaces. With a 20mm diameter, it is ideal for light to medium-duty applications such as pick-and-place systems. The cup is made of flexible material for secure gripping and stable suction. It's commonly used in automation, robotics, and packaging industries.

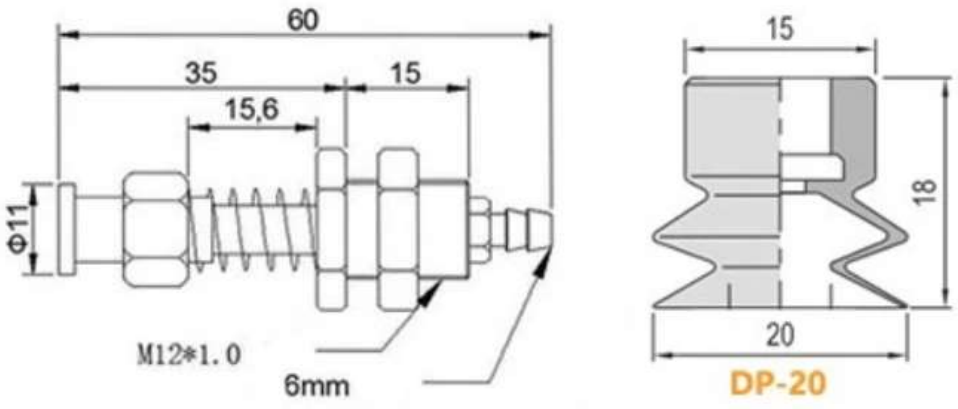

Figure 3.6 Vacuum Suction Cup dimension.

3.6 Limit Switch KW11-N:

Figure 3.7 Limit Switch KW11-N.

Limit Switch KW11-N is a compact, durable mechanical switch used to detect the position or presence of an object in automation systems. It features a roller lever actuator that provides reliable and precise switching action. The switch is commonly employed in industrial machinery for end-stop detection, safety interlocks, and control circuit feedback. It supports a wide range of voltages and currents, making it versatile for various applications.

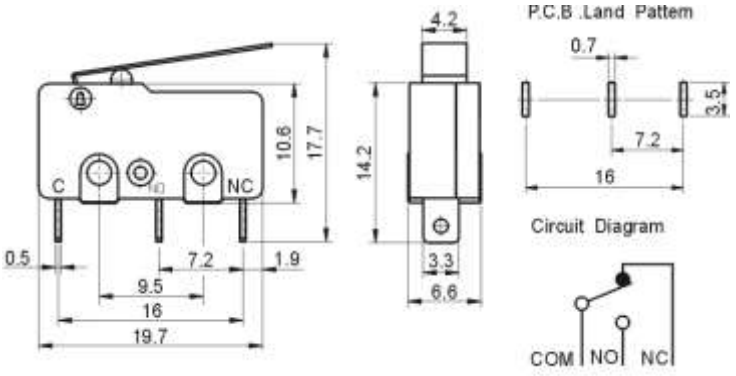

Figure 3.8 Limit Switch KW11-N dimensions.

3.7 Switching power supply (12V – 10A):

Figure 3.9 Switching power supply (12V – 10A).

- Input Voltage: 100V – 240V AC, 50/60 Hz
- Output Voltage: 12V DC
- Output Current: 10A
- Maximum Power: 120W
- Efficiency: Typically, above 85%
- Voltage Regulation: ±5%
- Ripple & Noise: ≤ 120mVp-p
- Protections: Overcurrent, Overvoltage, Short circuit
- Operating Temperature: -10°C to 50°C
- Operating Humidity: 20% – 90% RH (non-condensing)
- Dimensions: Approximately 190 x 90 x 50 mm (varies by model)
- Weight: Around 500g – 700g

3.8 SG90 Micro Servo Motor:

Figure 3.10 SG90 Micro Servo Motor.

Table 3.5 SG90 Servo specifications.

Parameter	**Specifications**
Weight	9 grams
Operating Voltage	4.8V – 6.0V
Stall Torque	1.8 kg.cm
Speed	0.1 s/60°
Rotation Angle	180°
Control Signal	PWM
PWM Pulse Range	500 – 2400 µs
Connector Type	3 – pin (Signal, VCC, GND)
Motor Type	Brushed DC
Dimensions	Approx 22.2 x 11.8 x 31 mm
Cable Length	25 cm

- The SG90 micro servo motor is designed for small-scale applications that require precise angular movement. It is especially popular in robotics, model airplanes, and Arduino projects.
- This servo motor is capable of rotating from 0 to 180 degrees, controlled by a PWM (Pulse Width Modulation) signal. The position of the motor is determined by the width of the input pulse.
- Inside the SG90 is a feedback mechanism using a potentiometer. This allows the motor to detect its current position and adjust accordingly, ensuring accurate and stable rotation.
- Despite its compact size, the SG90 can generate up to 1.8 kg·cm of torque, making it suitable for moving lightweight mechanical parts such as joints in robotic arms or camera gimbals.
- The motor has three connection wires: signal, VCC, and ground. It is compatible with most microcontrollers, including Arduino and Raspberry Pi, and can be powered with a 5V supply.
- Because of its low cost, easy control, and reliable performance, the SG90 is one of the most widely used servo motors in hobby electronics and educational projects.

3.9 Arduino Mega2560:

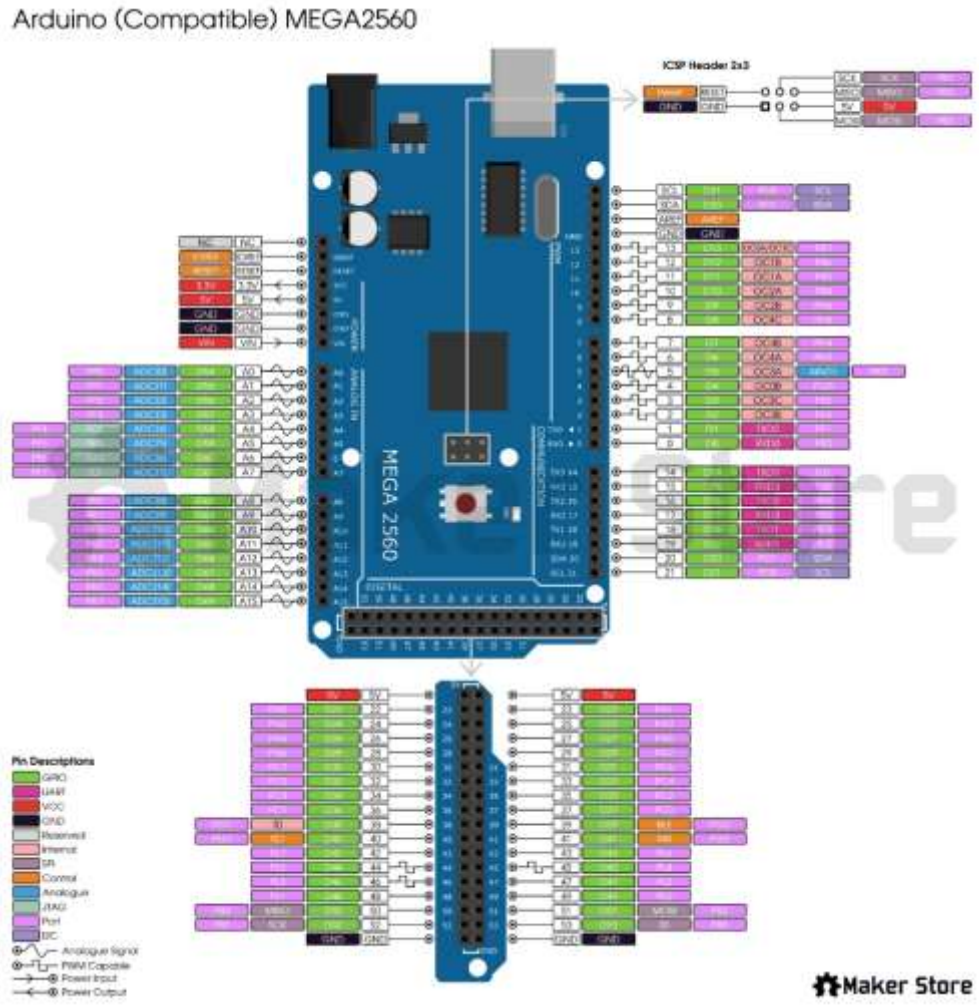

Figure 3.11 Arduino Mega2560.

Table 3.6 Arduino Mega2560 specification.

Parameter	**Specification**
Microcontroller	Atmega2560
Operating Voltage	5V
Input Voltage (recommended)	7V – 12V
Input Voltage (limits)	6V – 20V
Digital I/O Pins	54 (15 support PWM output)
PWM Channels	15
Analog Input Pins	16
DC Current per I/O Pin	20mA

DC Current for 3.3V Pin	50mA
Flash Memory	256 KB (8 KB used by bootloader)
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Communication Interfaces	UART, SPI, I2C
USB Connector	Type-B (USB printer port)
Power Jack	2.1 mm Barrel Jack
Dimensions	101.5 mm x 53.3 mm
Weight	37 g

- The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It is designed for projects that require many input/output connections, memory, and processing power.
- It provides 54 digital input/output pins, of which 15 can be used as PWM outputs. These are used to control devices like LEDs, motors, and other digital electronics.
- The board includes 16 analog input pins that can read sensor values and convert analog signals to digital data.
- It has multiple serial communication ports (UARTs), which allow it to communicate with other devices like computers, Bluetooth modules, or other microcontrollers.
- The PWM pins allow for analog-like control of actuators, such as adjusting motor speed or LED brightness.
- The SPI and I2C interfaces allow it to connect with various peripherals such as displays, memory cards, and sensors.
- The built-in USB interface allows for easy uploading of programs and serial communication with a PC.
- It also features onboard EEPROM, SRAM, and Flash memory, giving you space to store programs and runtime data.
- Thanks to its expandability and stability, the Arduino Mega 2560 is widely used in robotics, automation systems, 3D printing, and IoT applications.

3.10 385 Micro DC Water Pump Motor:

Figure 3.12 385 Micro DC Water Pump Motor.

Table 3.7 385 Micro DC Water Pump Motor specification.

Parameter	**Specification**
Motor Type	Brushed DC Motor (Model 385)
Operating Voltage	6V – 12V DC
Rated Voltage	12V DC
Current	0.5A – 1A
Power	5W – 10W
Weight	150g

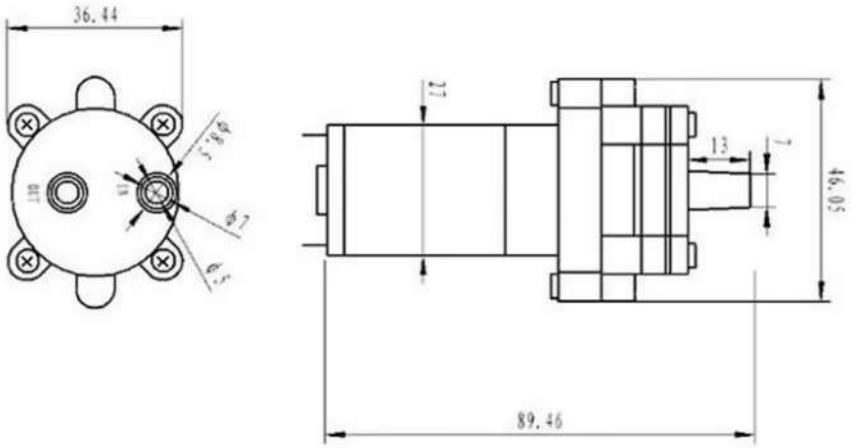

Figure 3.13 385 Micro DC Water Pump dimension.

3.11 Camera:

Figure 3.14 1080P HD Webcam.

In the automated PCB classification system, the final quality inspection stage plays a crucial role in ensuring that each circuit board meets the required standards in terms of component presence, placement accuracy, and overall dimensions. To achieve high precision and full automation, the system integrates a high-resolution industrial camera for image acquisition. This camera is designed for fast and reliable performance in harsh industrial environments. The captured image data is transmitted to a processing unit—such as an industrial computer or a microcontroller running machine vision software—for real-time analysis using advanced image processing and machine learning algorithms.

Main Functions of the Vision Camera in the Project:

- **Component verification:** Detects missing, misplaced, or incorrect electronic components on the circuit board.
- **Orientation and alignment check:** Ensures that all components are properly oriented and aligned according to design specifications.
- **Dimensional inspection:** Measures key dimensions of the PCB to verify compliance with manufacturing tolerances.
- **Automated classification and sorting:** Sends the analysis results to the robotic arm, which then classifies the board as correct, missing parts, or incorrect assembly based on machine learning predictions.

CHAPTER 4 MECHANICAL DESIGN AND MANUFACTURING

From the perspective of actuation and control, industrial robots are composed of mechanical structural modules that operate through various actuating mechanisms. Although robot structures can vary significantly depending on their specific applications, they typically consist of three main components: the manipulator (robotic arm), the power supply, and the control unit.

The manipulator includes multiple mechanical links and joints designed to allow relative motion between components, enabling complex movements and tasks. In our project, Chapter 2 focuses on the mechanical design of the entire system, including the robotic arm, conveyor belt, and other supporting components.

To create both 2D and 3D models of the robotic arm, the conveyor, and the system's structural elements, we utilize SolidWorks and AutoCAD. These software tools not only support precise and efficient mechanical design but also provide user-friendly features that streamline the design process.

4.1 Robotic arm design:

4.1.1 Analysis and selection of arm structure:

After researching various types of robotic arms, our team decided to design and fabricate a **4-degree-of-freedom (DOF) spherical coordinate robotic arm with an RRRR configuration**. This type of robot is widely used in industrial applications due to its several advantages:

- Compact footprint and space efficiency.
- Easy integration into workspaces.
- High dexterity and flexibility.
- Wide working range.
- Capable of handling relatively heavy payloads.

However, this configuration also presents certain limitations, such as:

- Lower structural stiffness in the vertical direction.
- Reduced positional accuracy at extended reach positions.

Therefore, the robotic arm must be designed with appropriate dimensions and lengths to ensure it can reach, grasp objects from the conveyor belt, and accurately place them into their designated compartments.

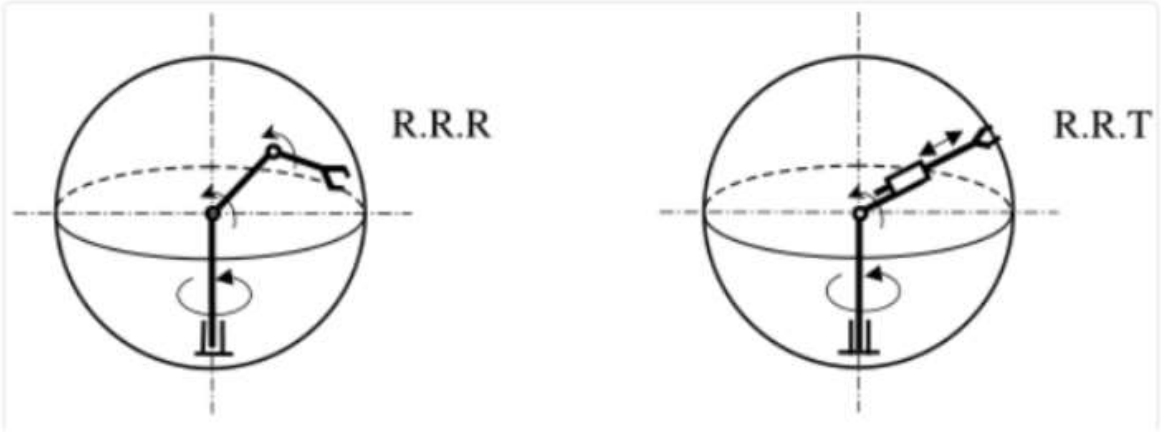

Figure 3.15 Spherical Coordinate Robot.

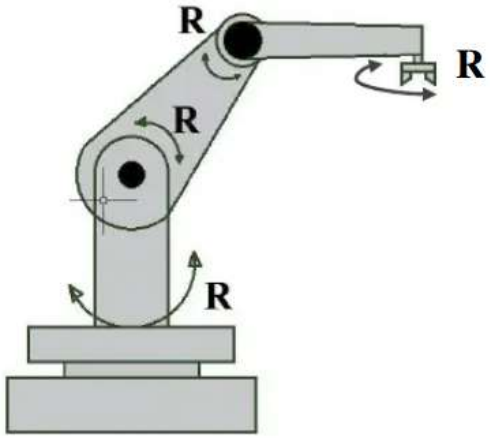

Figure 3.16 4-DOF robot mechanism.

The structure of the 4-degree-of-freedom robotic arm is as follows:

- A rotating base of the robot.
- A moving link that positions the suction head at the target location.
- A suction arm that picks up the object and places it into a container.

4.1.2 Dimensional Specifications of the Robotic Arm:

From the objectives presented in Chapter 1, the required dimensions and configuration of the 4-degree-of-freedom robotic arm have been defined to fulfill the project's operational demands. The arm is composed of several key subsystems:

- The mechanical structure of the arm comprises a series of links and joints that work together to generate fundamental movements. The wrist mechanism enhances maneuverability and precision, while the end-effector is designed to perform direct manipulation tasks on target objects.

- Motion in the robotic arm is achieved through an actuation system utilizing both stepper motors and servo motors, which drive the joints through appropriate transmission mechanisms.
- A sensor system is integrated to provide feedback and environmental awareness. Internal sensors monitor the positions and status of the robot's joints, while external sensors detect relevant information from the surroundings.

In this project, we opted to manufacture the structural parts of the robotic arm using 3D printing technology. This method was chosen for its affordability, ease of customization, and ability to produce components with good aesthetics and sufficient mechanical performance.

4.1.3 Kynematic Analysis:

4.1.3.1 Forward Kinematics:

- **Coordinate frames are defined as illustrated in the figure.**

The coordinate frame 0_0 is chosen as shown in the figure.

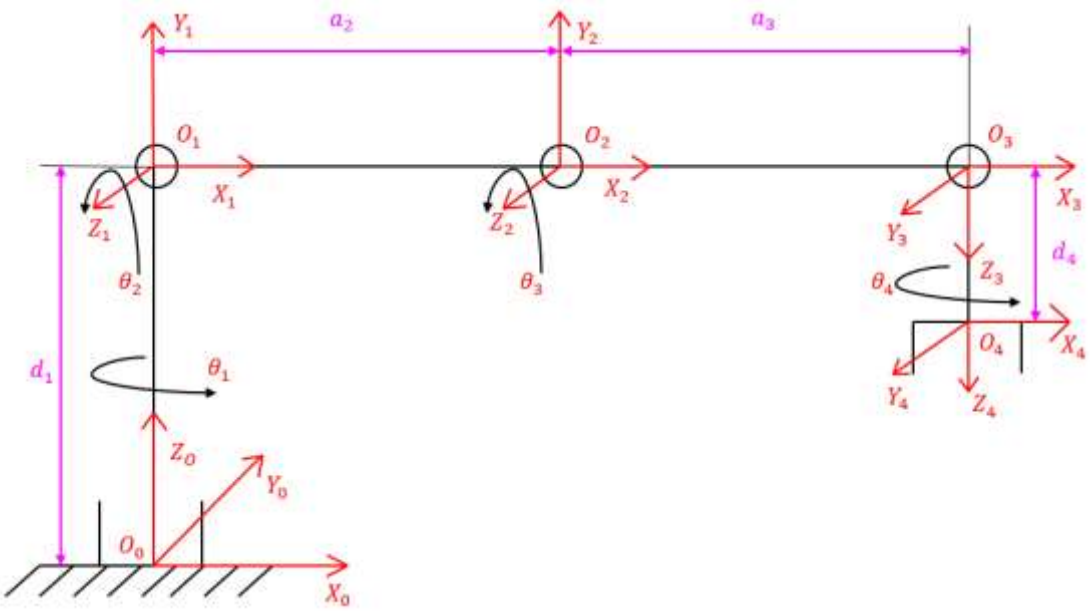

Figure 4.1 Establishment of Coordinate Frames and Parameters for the Robot.

- **Construct the Denavit–Hartenberg (D-H) Parameter Table**

Table 4.1 Denavit–Hartenberg Parameters for the 4-DOF Robotic Arm.

Link	θ_i	α_i	a_i	d_i
1	θ_1^*	$90°$	0	d_1
2	θ_2^*	0	a_2	0
3	θ_3^*	$90°$	a_3	0
4	θ_4^*	0	0	d_4

From the general form of the Denavit–Hartenberg transformation matrix:

$$A_i = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let $\sin\theta_i = S_i$, $\cos\theta_i = C_i$

The transformation matrices for each link A_i (i= 1÷ 4) are given as follows:

$$A_1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} C_3 & 0 & S_3 & a_3C_3 \\ S_3 & 0 & -C_3 & a_3S_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_4 = \begin{bmatrix} C_4 & -S_4 & 0 & 0 \\ S_4 & C_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using MATLAB software for computation and simplification.

$T_4=A_1.A_2.A_3.A_4$

$$T_4=\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Balancing the elements of the matrix T_4 leads to the following system of equations.

$n_x = C_1C_4C_{23} + S_1S_4$ $o_x = C_4S_1 - C_1S_4C_{23}$ $a_x = C_1S_{23}$

$p_x = C_1(a_3C_{23} + a_2C_2 + S_{23}d_4)$

$n_y = S_1C_4C_{23} - C_1S_4$ $o_y = -C_1C_4 - S_1S_4C_{23}$ $a_y = S_1S_{23}$

$p_y = S_1(a_3C_{23} + a_2C_2 + S_{23}d_4)$

$n_z = C_4S_{23}$ $o_z = -S_4S_{23}$ $a_z = -C_{23}$

$p_z = d_1 - d_4C_{23} + a_3S_{23} + a_2S_2$

This matrix contains two components: the position and orientation of the transformation matrix from frame 0 to frame 4. After simplification, the final position coordinates are obtained as follows:

$p_x = C_1(a_3C_{23} + a_2C_2) + S_1d_4$

$p_y = S_1(a_3C_{23} + a_2C_2) - C_1d_4$

$p_z = a_3S_{23} + a_2S_2 + d_1$

4.1.3.2 Inverse Kinematics:

In robotic systems, inverse kinematics plays a crucial role in determining the necessary joint configurations to achieve a desired end-effector position or orientation. The process involves solving mathematical equations derived from the kinematic structure of the system.

Inverse Kinematics using the Algebraic Method:

$$A_1^{-1} = \begin{bmatrix} c_1 & s_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ s_1 & -c_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_1^{-1} \cdot {}^0_4T = A_1^{-1} \cdot (A_1 \cdot A_2 \cdot A_3 \cdot A_4)$$

$$\begin{bmatrix} * & * & * & n_1 \\ * & * & * & n_2 \\ * & * & * & n_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} * & * & * & m_1 \\ * & * & * & m_2 \\ * & * & * & m_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} n_1 = p_xc_1 + p_ys_1 \\ n_2 = p_z - d_1 \\ n_3 = p_xs_1 - p_yc_1 \end{cases}$$

$$\begin{cases} m_1 = a_3c_{23} + a_2c_2 + d_4s_{23} \\ m_2 = a_3s_{23} + a_2s_2 - d_4c_{23} \\ m_3 = 0 \end{cases}$$

$\theta_{23} = \theta_2 + \theta_3 = 0$ Select initial constraints.

Given the target position of the robot, i.e., the coordinates x, y, z and the approach vector a_x, a_y, a_z (representing the direction in which the end-effector approaches the object projected onto the x, y, z axes), it is possible to determine the corresponding joint angles.

Calculation of Theta1:

$$n_3 = m_3 \leftrightarrow p_x\cos(\theta_1) + p_y\sin(\theta_1) = 0$$

$$\frac{\sin(\theta_1)}{\cos(\theta_1)} = \frac{y}{x}$$

$$\theta_1 = \text{atan2}(p_y, p_x)$$

Calculation of Theta3:

$$n_1 = m_1$$

$$n_2 = m_2$$

$$\begin{cases} p_x\cos\theta_1 + p_y\sin\theta_1 = a_3\cos(\theta_2 + \theta_3) + a_2\cos\theta_2 \\ p_z - d_1 = a_3\sin(\theta_2 + \theta_3) + a_2\sin\theta_2 \end{cases}$$

Let:

$$n_x = p_x\cos\theta_1 + p_y\sin\theta_1 - d_4\sin(\theta_2 + \theta_3)$$

$$n_y = p_z - d_1 + d_4\cos(\theta_2 + \theta_3)$$

Given the equations:

$$\begin{cases} n_x = a_3\cos(\theta_2 + \theta_3) + a_2\cos\theta_2 \\ n_y = a_3\sin(\theta_2 + \theta_3) + a_2\sin\theta_2 \end{cases}$$

$$n_x^2 + n_y^2 = (a_3\cos(\theta_2 + \theta_3) + a_2\cos\theta_2)^2 + (a_3\sin(\theta_2 + \theta_3) + a_2\sin\theta_2)^2$$

$$n_x^2 + n_y^2 = a_3^2 + a_2^2 + 2a_2a_3\cos(\theta_3)$$

$$\cos\theta_3 = \frac{n_x^2 + n_y^2 - a_2^2 - a_3^2}{2a_2a_3}$$

⇨ $\sin\theta_3 = \pm\sqrt{1 - c_3^2} = \pm\sqrt{1 - \left(\frac{n_x^2 + n_y^2 - a_2^2 - a_3^2}{2a_2a_3}\right)^2}$

– The negative sign (–) indicates the downward movement (descent) of the robotic arm

$$\theta_3 = \text{atan2}\left(\pm\sqrt{1 - c_3^2},\ c_3\right) = \text{atan2}\left(\pm\sqrt{1 - \left(\frac{n_x^2 + n_y^2 - a_2^2 - a_3^2}{2a_2a_3}\right)^2},\ \frac{n_x^2 + n_y^2 - a_2^2 - a_3^2}{2a_2a_3}\right)$$

Calculation of Theta2:

$$n_1 = m_1$$

$$n_2 = m_2$$

$$\begin{cases} p_x cos\theta_1 + p_y sin\theta_1 = a_3 cos(\theta_2 + \theta_3) + a_2 cos\theta_2 \\ p_z - d_1 = a_3 sin(\theta_2 + \theta_3) + a_2 sin\theta_2 \end{cases}$$

Let:

$$n_x = p_x\cos\theta_1 + p_y\sin\theta_1 - \mathrm{d}_4\sin(\theta_2 + \theta_3)$$

$$n_y = p_z - d_1 + \mathrm{d}_4\cos(\theta_2 + \theta_3)$$

Given the equations:

$$\begin{cases} n_x = a_3\cos(\theta_2 + \theta_3) + a_2\cos\theta_2 \\ n_y = a_3\sin(\theta_2 + \theta_3) + a_2\sin\theta_2 \end{cases}$$

$$\begin{cases} n_x = a_3(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3) + a_2\cos\theta_2 \\ n_y = a_3(\sin\theta_2\cos\theta_3 + \cos\theta_2\sin\theta_3) + a_2\sin\theta_2 \end{cases}$$

$$\begin{cases} n_x = \cos\theta_2(a_3\cos\theta_3 + a_2) - \sin\theta_2(a_3\sin\theta_3) \\ n_y = \sin\theta_2(a_3\cos\theta_3 + a_2) + \cos\theta_2(a_3\sin\theta_3) \end{cases}$$

$$\begin{cases} \cos\theta_2 = \dfrac{n_x + \sin\theta_2(a_3\sin\theta_3)}{(a_3\cos\theta_3 + a_2)} \\ \sin\theta_2 = \dfrac{n_y - \cos\theta_2(a_3\sin\theta_3)}{(a_3\cos\theta_3 + a_2)} \end{cases}$$

$$\theta_2 = atan2(\sin\theta_2,\ \cos\theta_2)$$

The formulation of kinematic equations represents a fundamental step in the analysis of robotic motion and the subsequent development of control algorithms.

When the joint variables are provided as time-dependent inputs, the position and orientation of the robot's end-effector at any given moment can be accurately derived through the forward kinematics equations.

The inverse kinematics problem involves determining the joint variable values corresponding to specified end-effector pose parameters. Solutions to these inverse kinematics equations form the basis for programming the robot's operational control.

4.1.4 Design of Robotic Arm Components:

4.1.4.1 Rotational mechanism of the end-effector:

For the joints of the robot, in order to ensure a compact design and precise operation of the robotic arm, we chose to use belt transmission systems for the rotational joints of the base, joint 1, and joint 2.

A gear ratio of 2:9 was selected. A GT20 pulley was attached to the motor, and the number of teeth on the driven gear is 90.

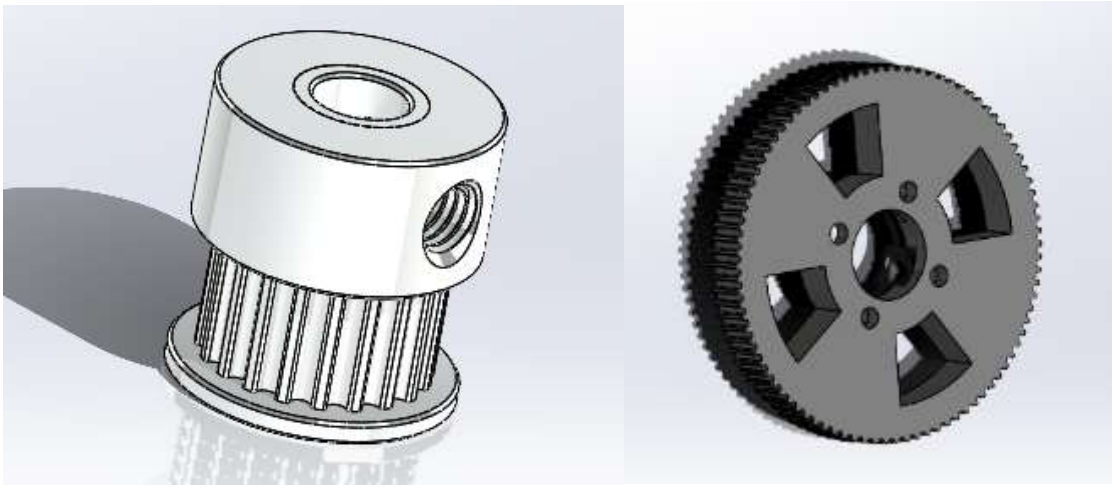

Figure 4.2 Pulley and driving sprocket.

4.1.4.2 Motor selection for the mechanism:

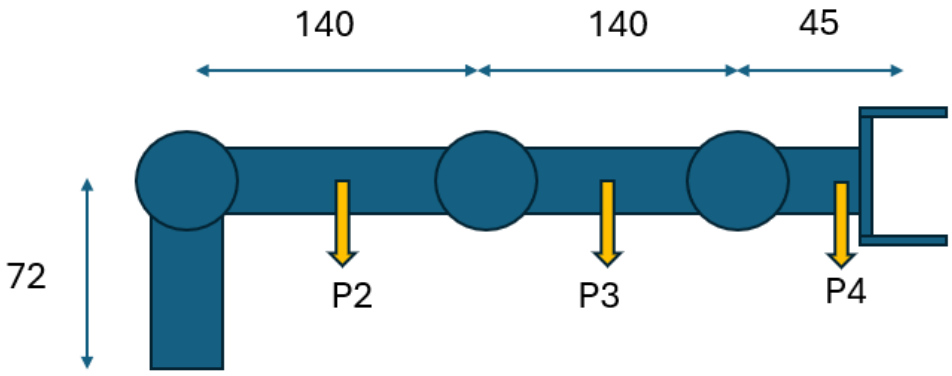

Figure 4.3 The weight acting on each link.

Link 1:

Mass of Link 2	400 g
Mass of the motor rotating Joint 2	350 g
Mass of the motor rotating Link 3	350 g

Link 2:

Mass of Link 2	300 g

Link 3:

Mass of Link 2	250 g

End-effector (Vacuum):

Mass of Link 4	100 g
Mass of the motor at Link 4	150 g

Assumptions:

– All links are homogeneous.

– Gravitational force is uniformly distributed along each link.

– Gravitational acceleration: g = 9.8 m/s²

The weight of each link is calculated as follows:

Link 1:

$$p_1 = m_1 \times g = \frac{(350 + 350 + 400) \times 9.8}{1000} = \frac{1100 \times 9.8}{1000} = 10.78\text{N}$$

Link 2:

$$p_2 = m_2 \times g = \frac{300 \times 9.8}{1000} = 2.94\text{N}$$

Link 3:

$$p_3 = m_3 \times g = \frac{250 \times 9.8}{1000} = 2.45\text{N}$$

End-effector (Vacuum):

$$p_4 = m_4 \times g = \frac{(150 + 100) \times 9.8}{1000} = \frac{250 \times 9.8}{1000} = 2.45\text{N}$$

Analysis of the Acting Characteristics at Joint 1:

- At joint 1, the gravitational force acts along the motor's axial direction, thus it does not generate any torque.
- The motor only needs to produce sufficient force to overcome the static friction at the joint's rotational axis.
- To reduce friction, thrust bearings are used, helping to lower the friction coefficient to approximately:

$$\mu = 0.005$$

The static friction force at joint 1 is calculated as:

$$F = \mu \times m \times g = 0.005 \times 10.78 = 0.054\ \text{N}$$

Assuming a safety factor of 2, the minimum force required to rotate joint 1 is:

$$F = 2 \times 0.054 = 0.104\ \text{N}$$

Given that the distance from the rotational axis to the motor axis is 40 mm, the minimum torque required by the motor to rotate joint 1 is:

$$M_1 = F_{min} \times d = 0.104 \times 0.04 = 0.00416\ \text{N·m}$$

Since joint 1 uses a gear transmission with a gear ratio of 4.5, the required torque output of the motor is:

$$M_{motor} = M_1 / i = 0.00416 / 4.5 \approx 0.000924\ \text{N·m}$$

⇒ The selected motor is a NEMA 17 stepper motor (Model: 17HS4401).

Joint 2 bears the entire load of the subsequent links, so the torque at this joint is calculated as the sum of the torques generated by the weights of links 2, 3, and 4.

The formula for torque is as follows:

$$M_2 = (P_2 \times d_2) / 2 + P_3 \times (d_2 + d_3 / 2) + P_4 \times (d_2 + d_3 + d_4 / 2)$$

Substituting values:

$$M_2 = (2.94 \times 7 + 2.45 \times 21 + 2.45 \times 30.25) / 100 = 1.46\ \text{N·m}$$

Applying a safety factor of 2:

$$M_{2,\text{safe}} = 1.46 \times 2 = 2.92\ \text{N·m}$$

Using a gear transmission with a ratio of 1:5, the required motor torque is:

$$M_{motor} = 2.92 / 5 = 0.584\ \text{N} \cdot \text{m}$$

⇒ Selected motor: NEMA 17 – Model 42BYGH47

Motor Selection for Joint 3:

The torque at Joint 3 is calculated based on the contributions from the weights of Link 3 and Link 4, using the following formula:

$$M_3 = (P_3 \times d_3) / 2 + P_4 \times (d_3 + d_4 / 2)$$

Substituting values:

$$M_3 = (2.45 \times 7 + 2.45 \times 16.25) / 100 = 0.57\ \text{N} \cdot \text{m}$$

Applying a safety factor of 2:

$$M_{3,\text{safe}} = 0.57 \times 2 = 1.14\ \text{N} \cdot \text{m}$$

Using a gear transmission with a ratio of 1:5, the required motor torque is:

$$M_{motor} = 1.14 / 5 = 0.228\,N \cdot m$$

⇒ Selected motor: NEMA 17 – Model 42HS33.

Figure 4.4 Nema17 Stepper Motor.

Motor Selection for Jooint 4:

The torque at Joint 4 is calculated using the following formula:

$$M_4 = (P_4 \times d_4) / 2$$

Substituting the values:

$$M_4 = (2.45 \times 2.25) / 100 = 0.055\,N \cdot m$$

Applying a safety factor of 2:

$$M_{4,safe} = 0.055 \times 2 = 0.11\,N \cdot m$$

⇒ Selected motor: SG90 (rated torque: 0.2 N·m)

Figure 4.5 SG90 Micro Servo Motor.

Based on the torque calculations and the selected motors — including NEMA17 (42BYGH47 and 42HS33) and SG90 — with a minimum safety factor of 2, the robotic arm is capable of lifting a maximum payload of 400 grams at the end-effector without motor overloading or performance instability.

4.1.5 3D Design of the 4-DOF Robotic Arm:

Our team utilized SolidWorks software to develop the 3D design of both the robotic arm and the overall system. Individual components were modeled using the Part environment, followed by assembly of the complete structure using the Assembly module. This process also enabled us to detect and resolve any mechanical interferences between components, ensuring proper fit and ease of assembly during physical implementation.

Based on the dimensional drawings and 3D models of the robotic arm, the components are exported as STL files to initiate the 3D printing process.

Figure 4.6 4-DOF Robotic Arm.

The following is a 3D illustration of the links in the model:

Base Link: The base serves as a fixed support for the robotic arm. It is rigidly mounted to the working surface and does not participate in the arm's motion. Its primary function is to provide a stable and secure foundation, ensuring structural integrity during operation.

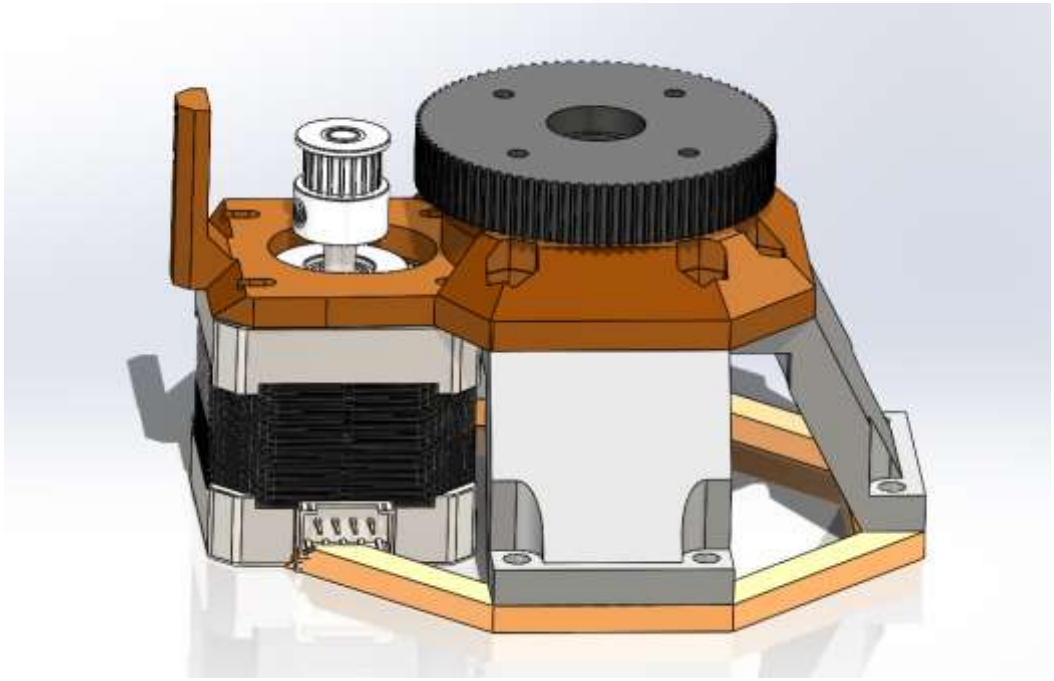

Figure 4.7 Base Link.

Link 1: Link 1 is responsible for the horizontal rotation of the entire arm around the vertical (Z) axis. This joint enables the arm to sweep left or right within its working envelope. It plays a critical role in determining the orientation of the arm relative to the workspace and is typically actuated by a stepper or servo motor.

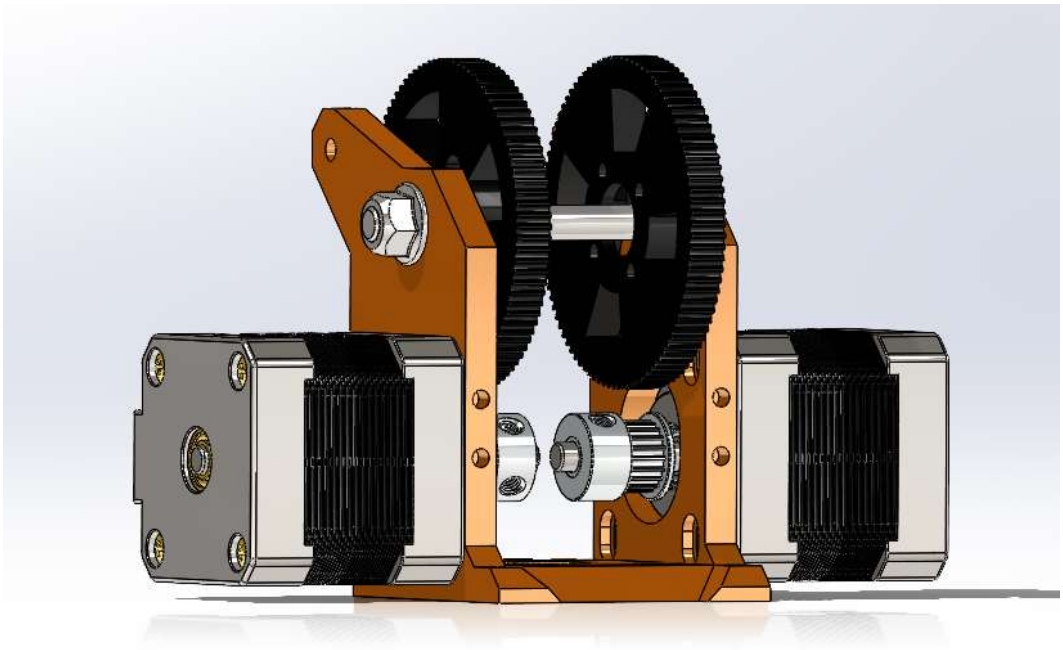

Figure 4.8 Link 1.

Link 2: Link 2 allows the arm to move vertically by rotating in the sagittal plane. It raises or lowers the arm to adjust the height of the end-effector. This degree of freedom

is essential for approaching objects at different elevations and contributes to the reachability of the arm.

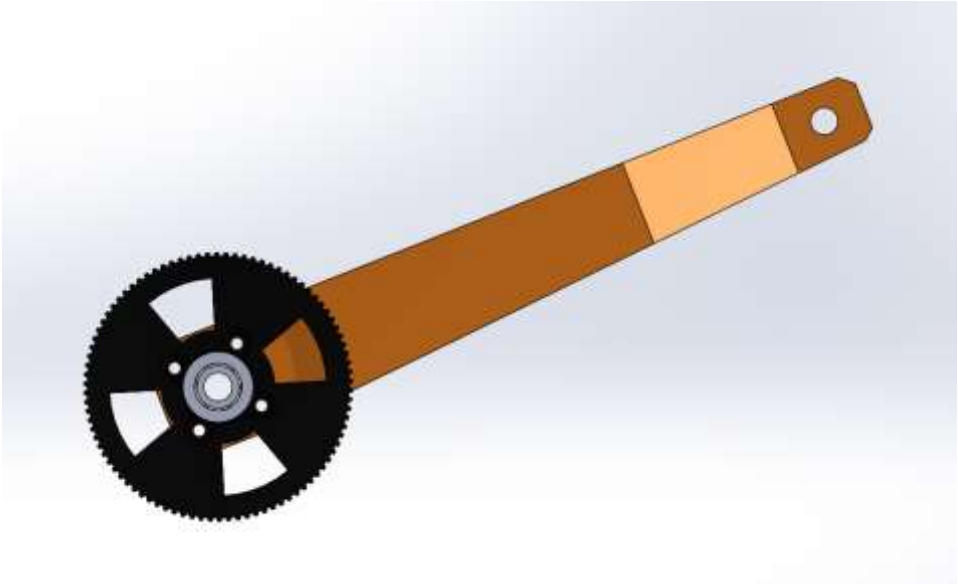

Figure 4.9 Link 2.

Link 3: Working in tandem with Link 2, Link 3 provides forward and backward movement of the end-effector. It adjusts the radial position of the suction head, allowing the robot to reach targets at various distances. This joint is vital for precise positioning and is actuated to extend the robot's range within the workspace.

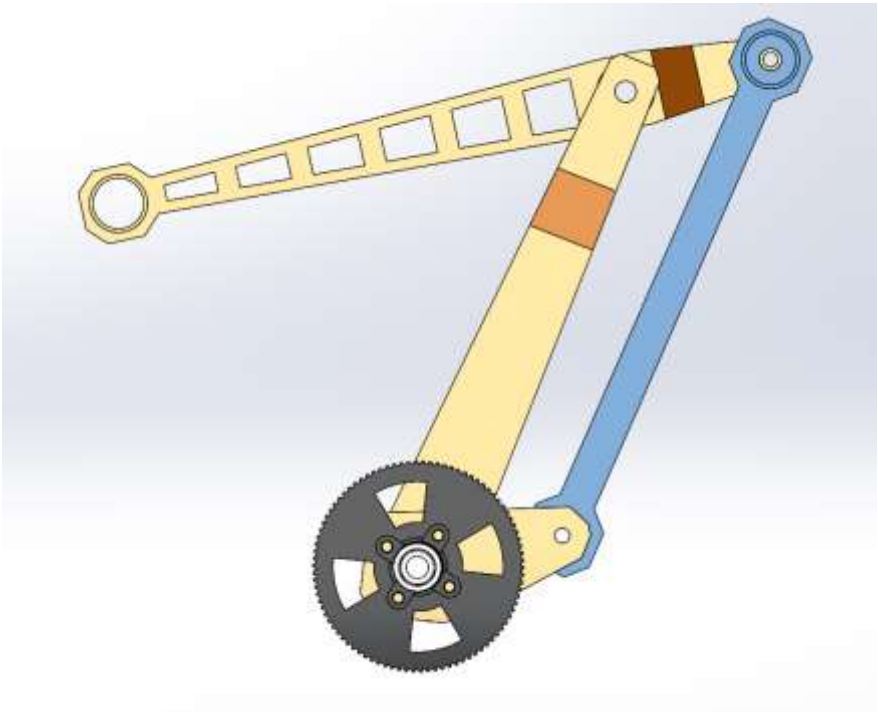

Figure 4.10 Link 3.

Link 4: The fourth link is dedicated to the rotation of the vacuum suction head. This joint allows the end-effector to align properly with objects of varying orientations. Such rotational capability enhances the flexibility and adaptability of the robot, particularly in object sorting and manipulation tasks.

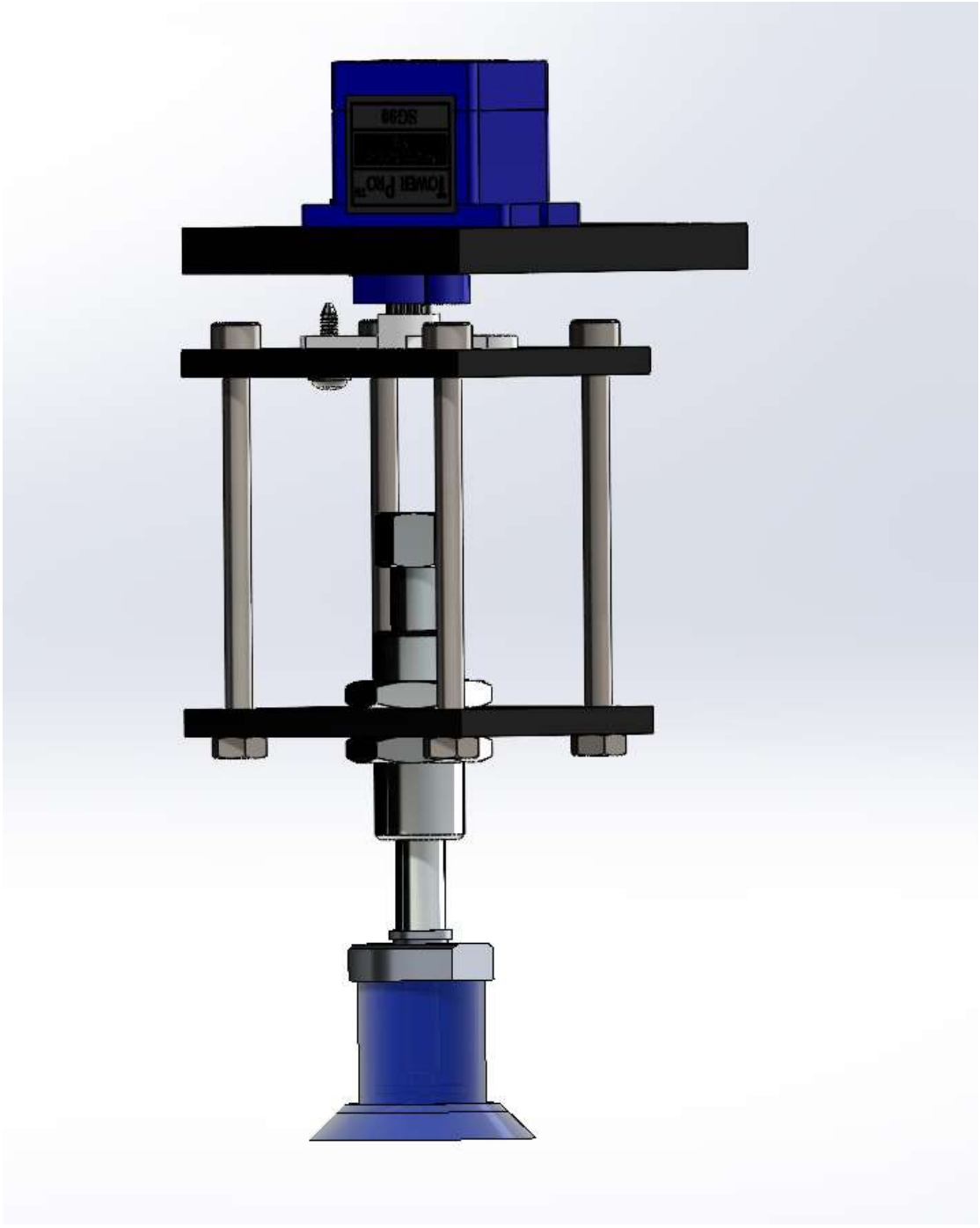

Figure 4.11 Link 4.

4.1.6 Workspace of the 4-DOF Robotic Arm

To evaluate the operational capabilities of the 4-DOF robotic arm, we conducted a workspace simulation based on forward kinematics. The simulation was implemented in MATLAB using joint angle ranges corresponding to the actual mechanical constraints of the system. The joint angles were sampled in the following ranges:

- Joint 1 (base rotation): −150° to +150°
- Joint 2: 80° to 120°
- Joint 3: 80° to 130°

From these ranges, a 3D visualization of the reachable workspace was generated, along with the XY plane projection. These visualizations help confirm that the arm has sufficient reach to perform pick-and-place tasks across the conveyor area and target bins. The XY projection also demonstrates the effective horizontal coverage of the robot's workspace.

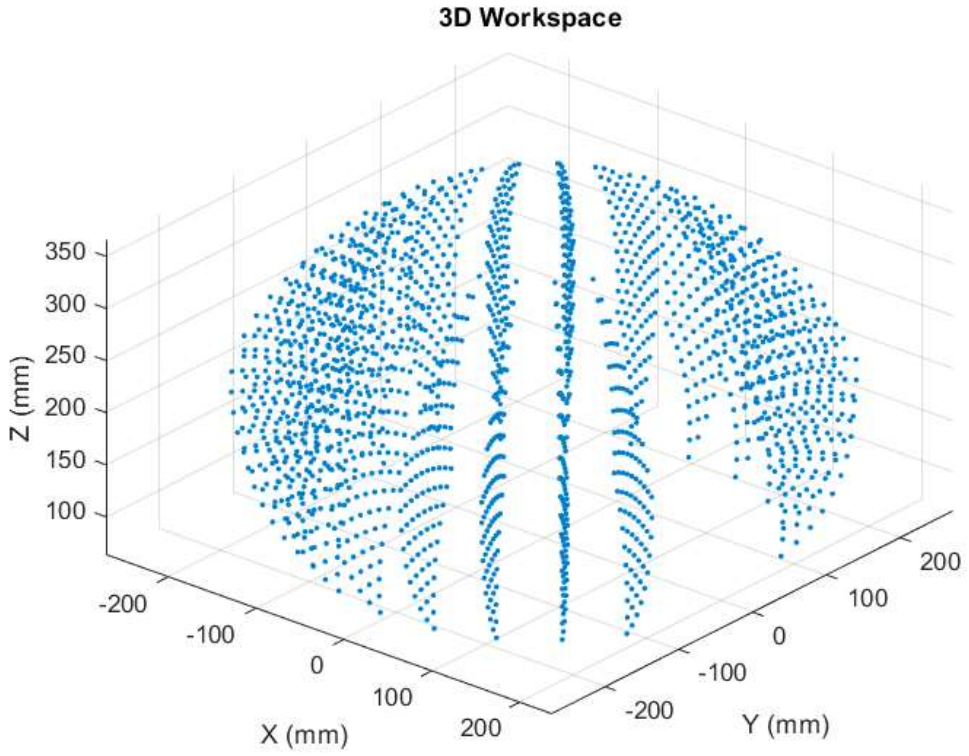

Figure 4.12 3D Workspace Visualization of the 4-DOF Robotic Arm.

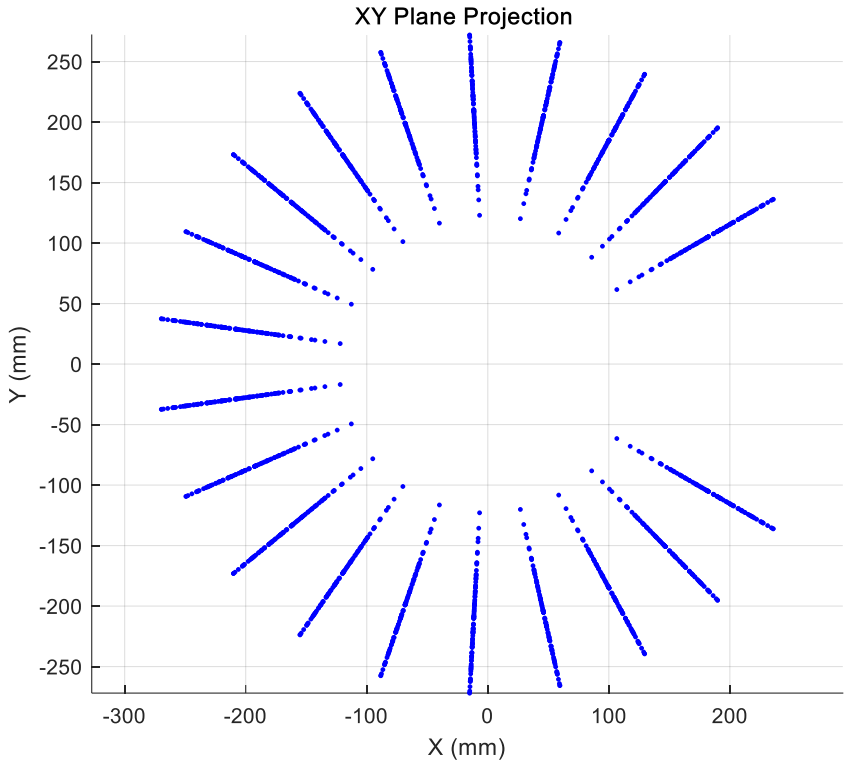

Figure 4.13 XY Plane Projection of the 4-DOF Robotic Arm Workspace.

4.2 Conveyor Belt Design

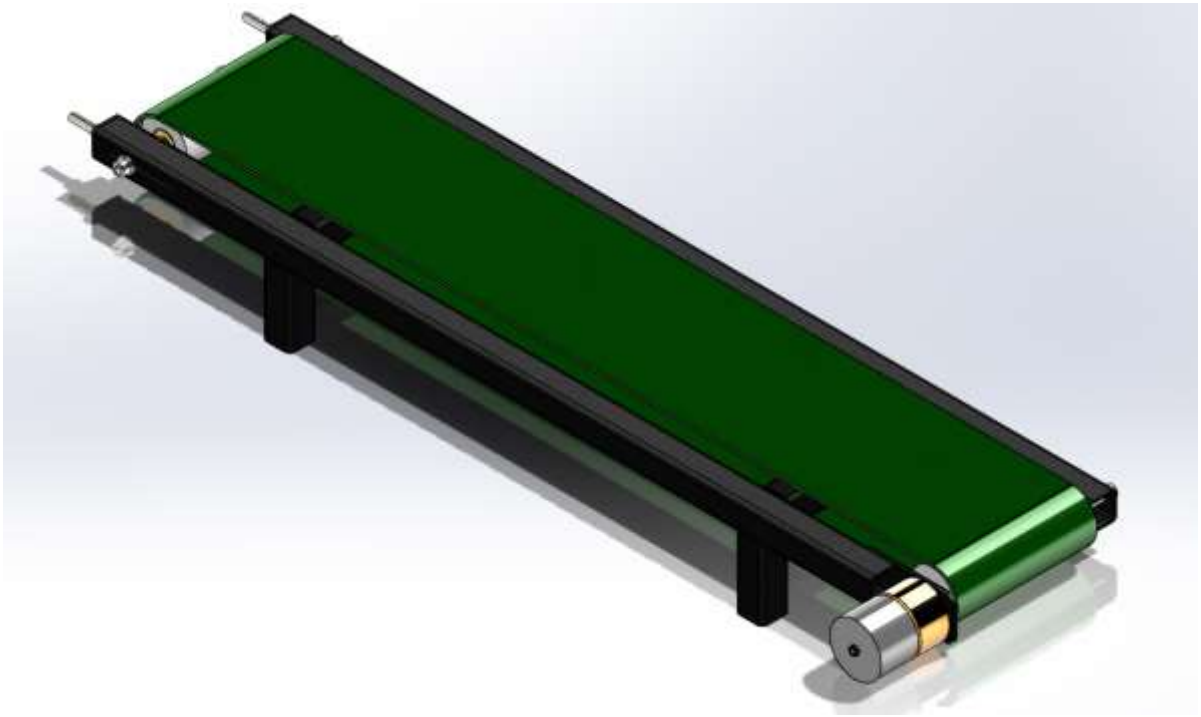

Figure 4.14 Conveyor.

Initial technical specifications:

- Time time luck transfer product products: t1= 3 second

- Belt speed: v = 1/3 = 0.33 m/s = 33.3 m/min
- Friction coefficient of belt with roller: μ = 0.6
- Belt length: L = 600 mm = 0.6 m
- Belt width: W = 100 mm = 0.1 m
- Belt thickness: h = 3 mm = 0.003 m
- Diameter of roller drum: D = 60 mm = 0.6 * 10−3m
- Belt weight: mb = 0.6 * 0.2 * 0.003 = 0.6 * 10−3 $m3$
- Rubber specific volume: D = 1360 kg/$m3$
- Total weight of belt: $m1$ = 0.6 * 10−3 * 1360 = 0.816 kg = 816 g
- Total volume of goods on the belt: $m1$ = 200 g * 2 = 400 g
- Total conveyor belt mass: m = 400 + 816 = 1216 g = 1,216 kg
- Conveyor belt traction force:

F$k \geq Fms = \mu * m * g$ * k = 0.6 * 1,216 * 9.8 * 2 = 14,3 N

- In this case:

M: Mass of the object (kg)

g: Gravitational acceleration (m/s^2)

μ: Coefficient of friction

k: Safety factor

Hence, N = F * v = 14,3 * 0.33 = 4,719 (W)

- Overall system efficiency:

$$\eta = \eta x * \eta ol\ 2 = 0.93 * 0.992 = 0.9115$$

Ηx = 0.93 - chain drive efficiency

Ηol = 0.99 - roller pair efficiency

- Required motor power: Nct = N η = 4,719 0.9915 = 4,76 (W)
- Shaft speed of the tangential axis: nlv = 60000$*v$ $\pi * D$
- Preliminary gear ratio calculation
- Choose gear ratio of gearbox: ih = 1

4.3 Overall System Model

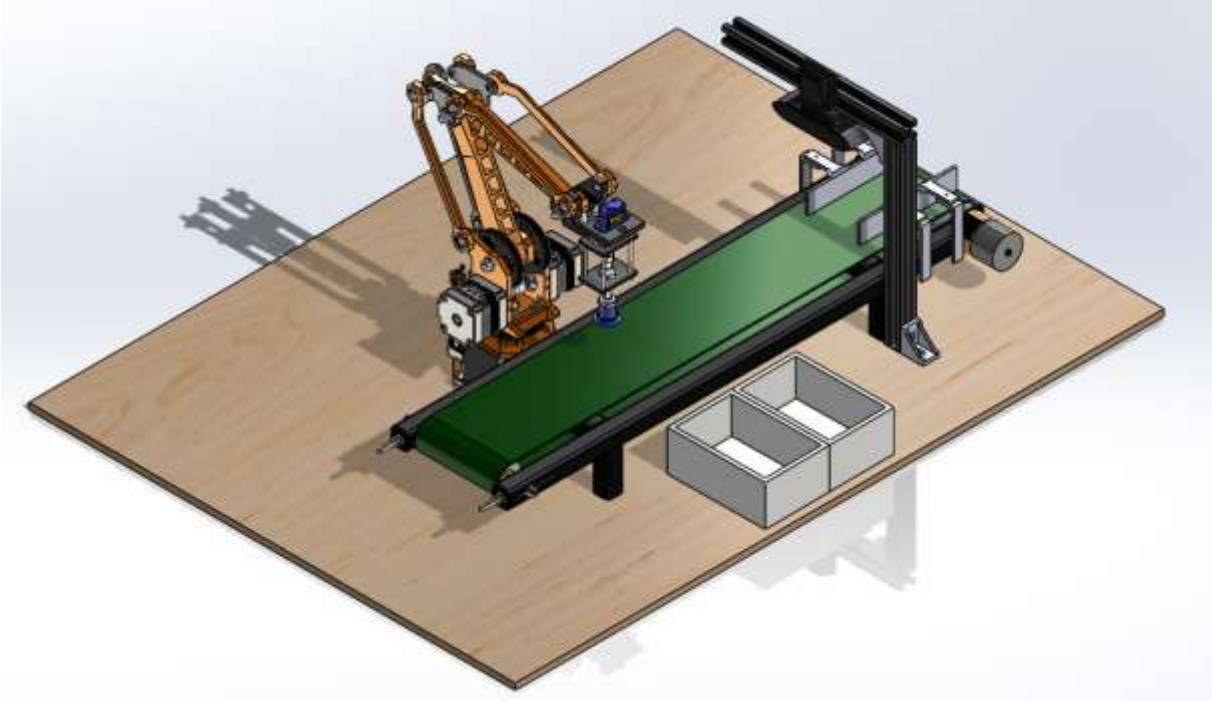

Figure 4.15 The model after design completion.

CHAPTER 5 CONTROL SYSTEM DESIGN

5.1 Block Diagram:

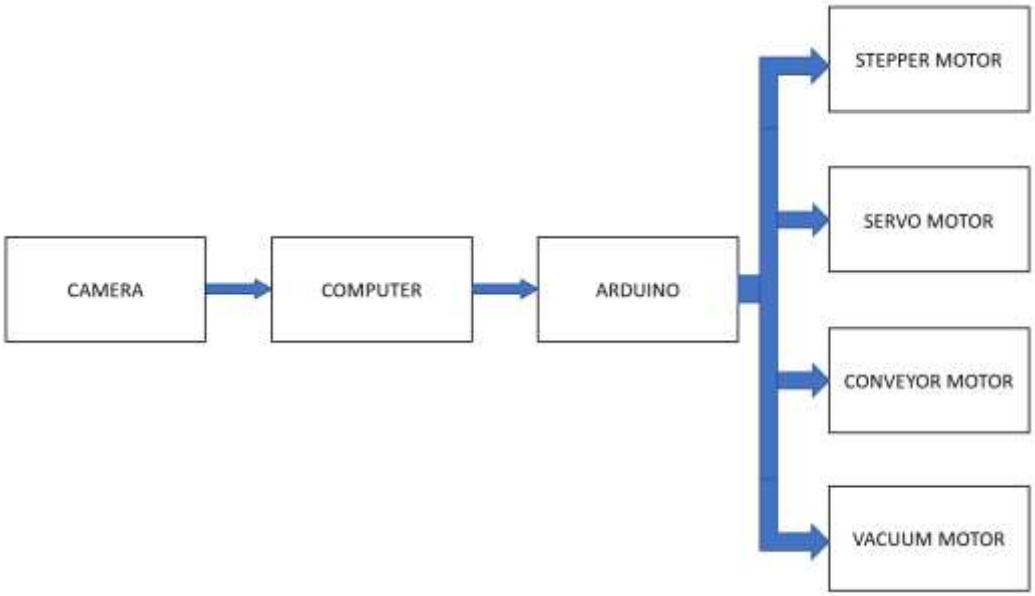

Figure 5.1 System block diagram.

The camera captures images of the target object for processing.

The computer (CPU) receives data from the controller and image data from the camera to perform image processing for identifying the type of electrical circuit. Simultaneously, it also receives signals and programs from the operator.

The controller handles the transmission and reception of control data between the computer and various sensors, and sends control signals to the motor drivers to adjust motor rotation angles and generate control pulses for the servo motors. It also processes feedback signals from limit switches and push buttons to execute the predefined control functions.

After receiving information about the type of circuit from the computer via UART communication and signals from the limit switches, the central controller manages the stepper motors and vacuum gripper to pick and place the circuits into corresponding bins.

Upon receiving signals from the sensors, the central controller halts the conveyor motor and resumes its operation after the robotic arm completes a full pick-and-place cycle. Simultaneously, the servo motor is activated when the controller detects sensor signals and the robotic arm finishes placing the item.

5.2 Algorithmic flow diagram:

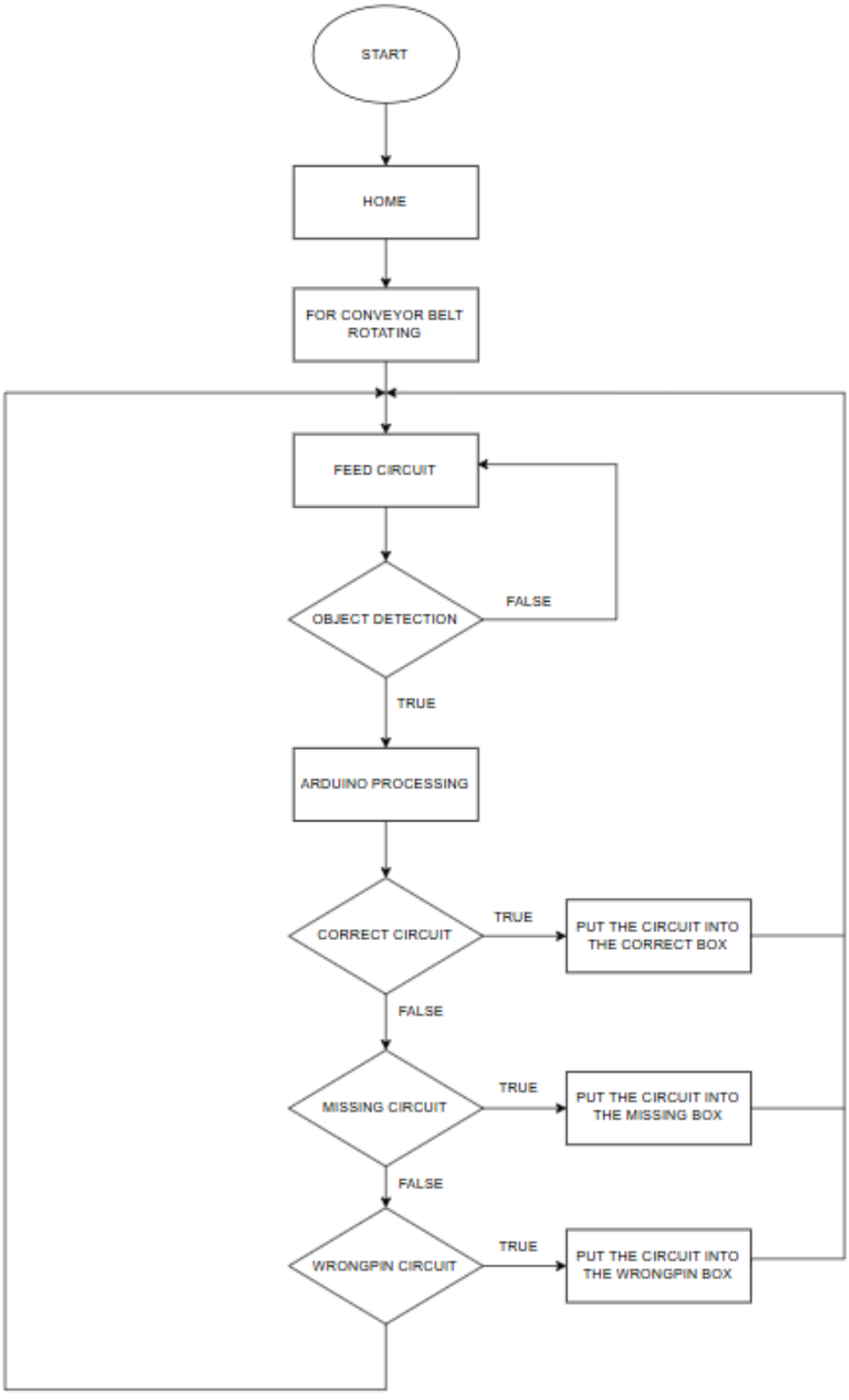

Figure 5.2 System algorithm flowchart.

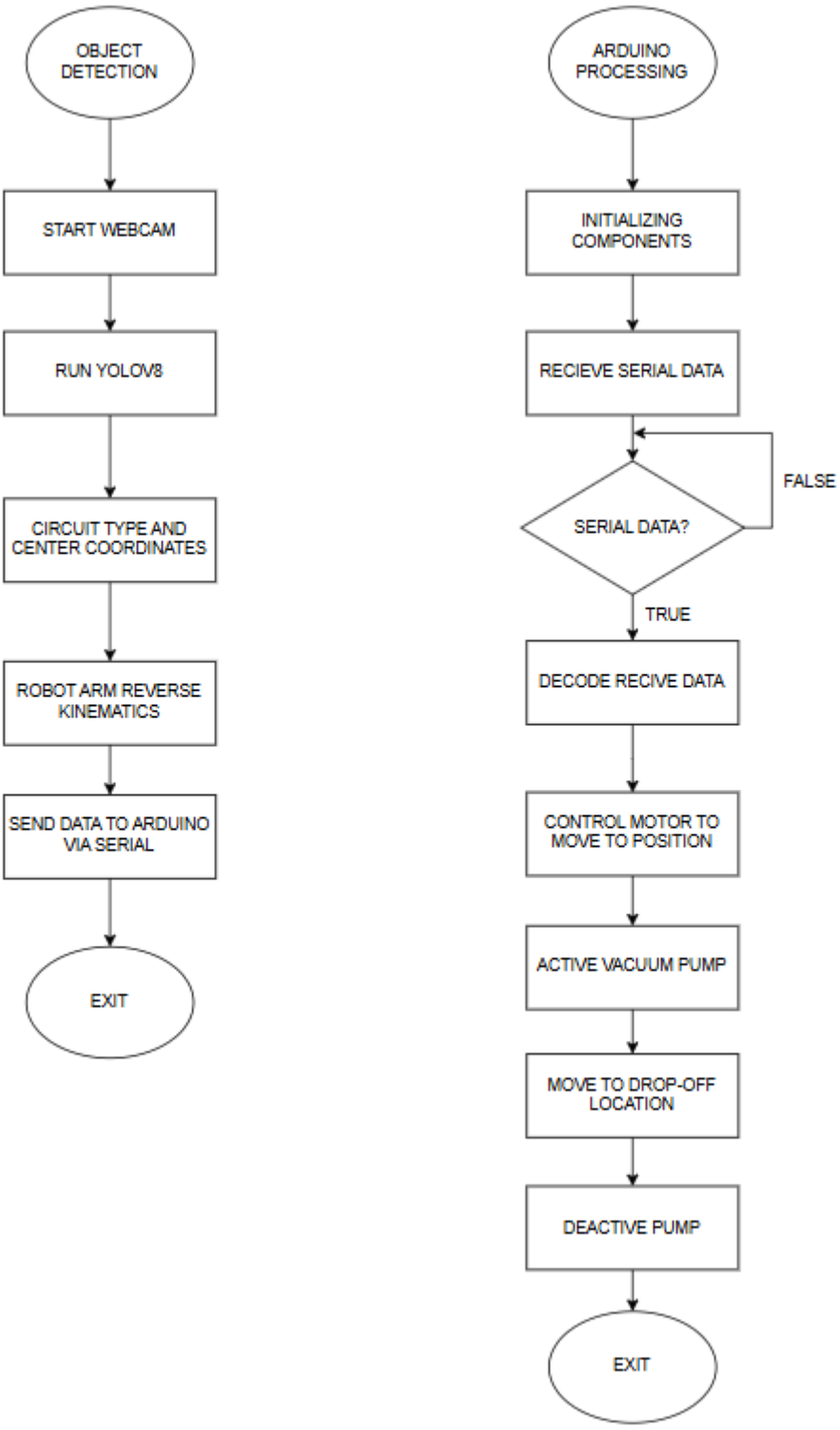

Figure 5.3 Microcontroller Algorithm Flowchart.

5.3 Control circuit schematic diagram:

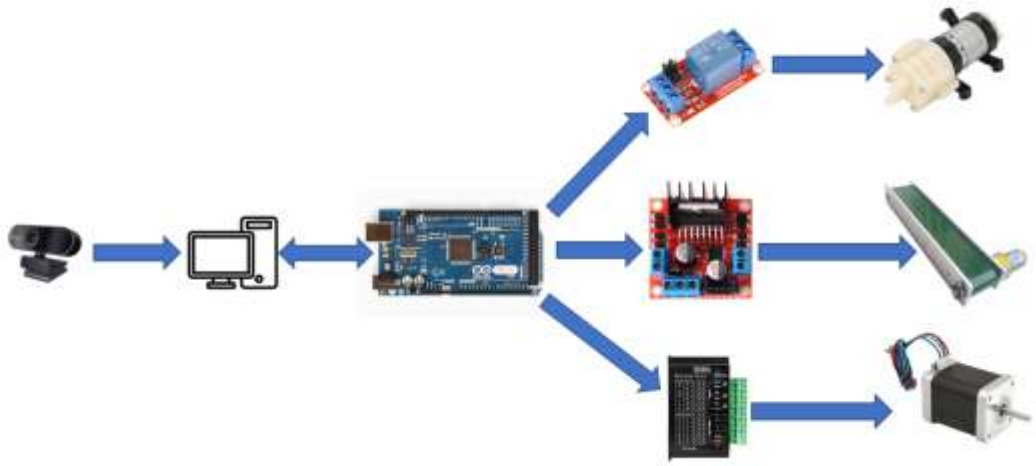

Figure 5.4 Overall System Diagram.

5.4 Monitoring interface:

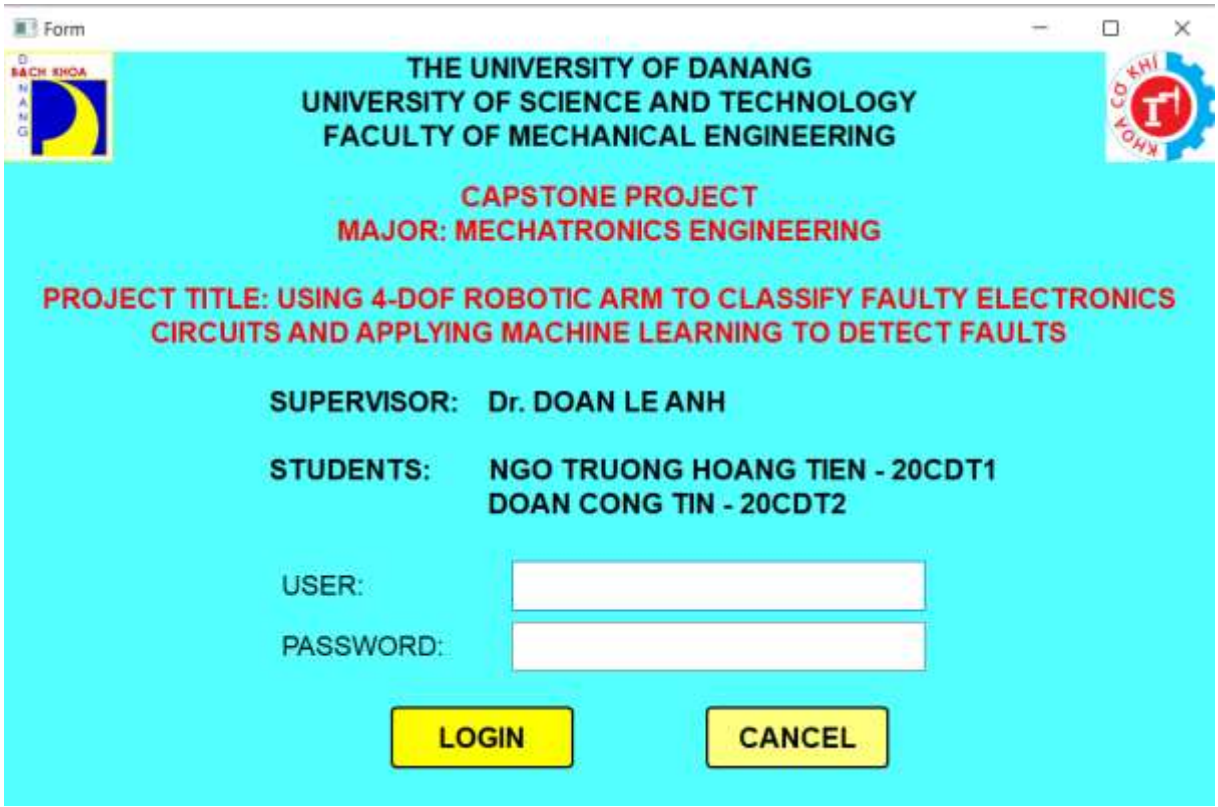

Figure 5.5 Login Interface.

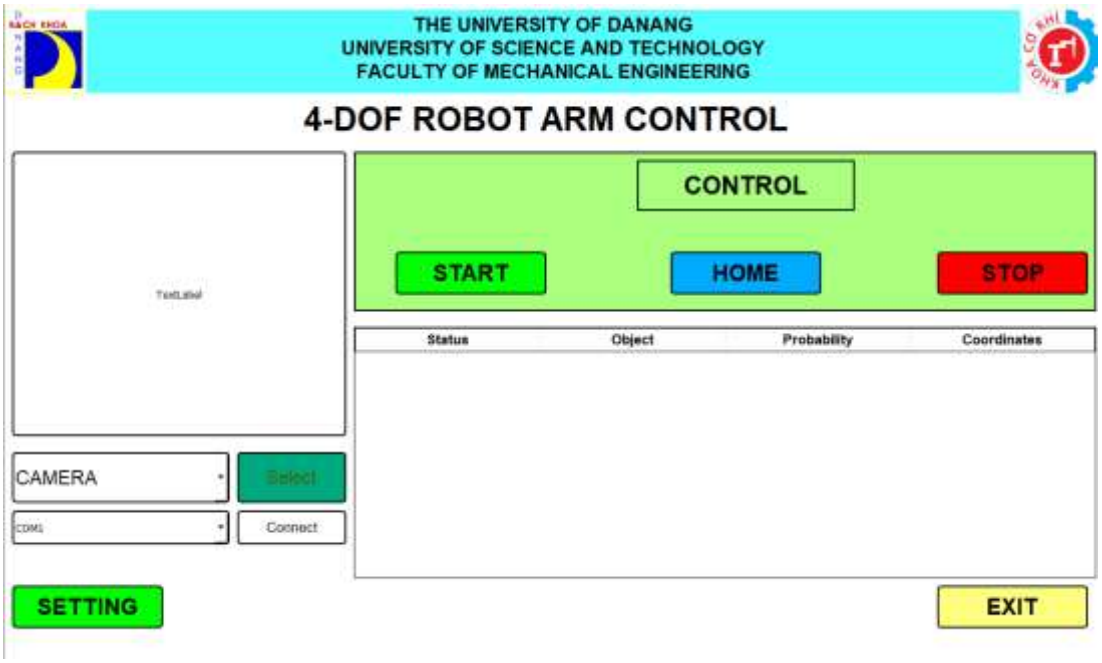

Figure 5.6 Control Panel.

5.5 Software:

5.5.1 Arduino control programming software:

Use Arduino IDE software for programming.

Figure 5.7 Arduino IDE Interface.

Arduino IDE (Integrated Development Environment) is an open-source platform used for writing, compiling, and uploading code to Arduino microcontrollers. It supports C/C++ programming and provides a simple, user-friendly interface for developing embedded systems. The IDE includes a serial monitor for real-time

debugging and communication, making it ideal for prototyping robotics, automation, and IoT applications.

5.5.2 Computer Interface design software:

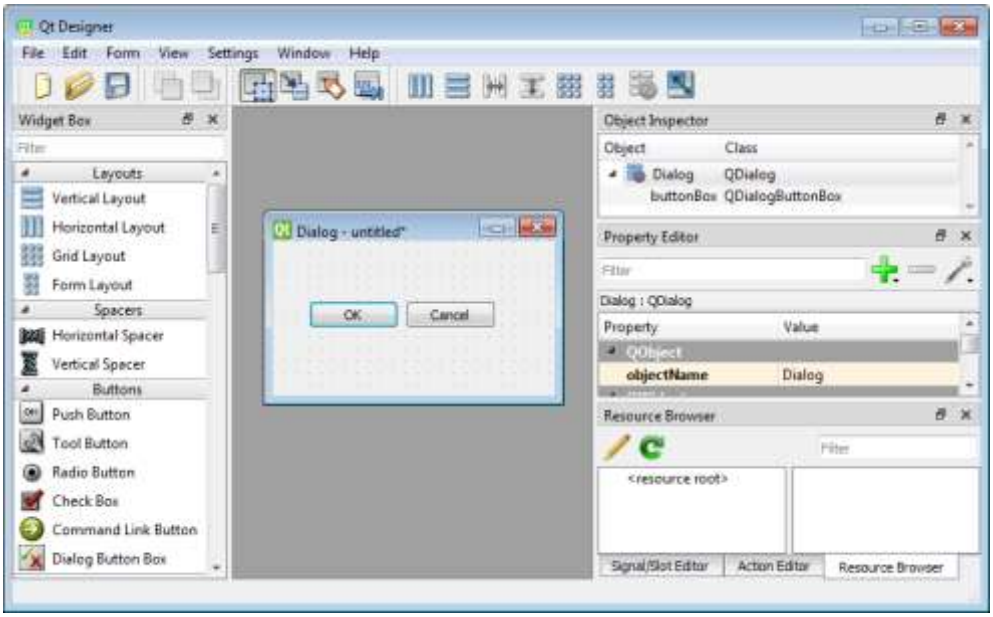

Figure 5.8 QtDesigner Interface.

Qt Designer is a graphical user interface (GUI) design tool that is part of the Qt development framework. It allows developers to design and customize application interfaces visually by dragging and dropping widgets, without writing code manually. The generated .ui files can be integrated into Python or C++ projects, making it a powerful and efficient tool for rapid GUI development.

5.5.3 Implementing GUI Logic and Integration:

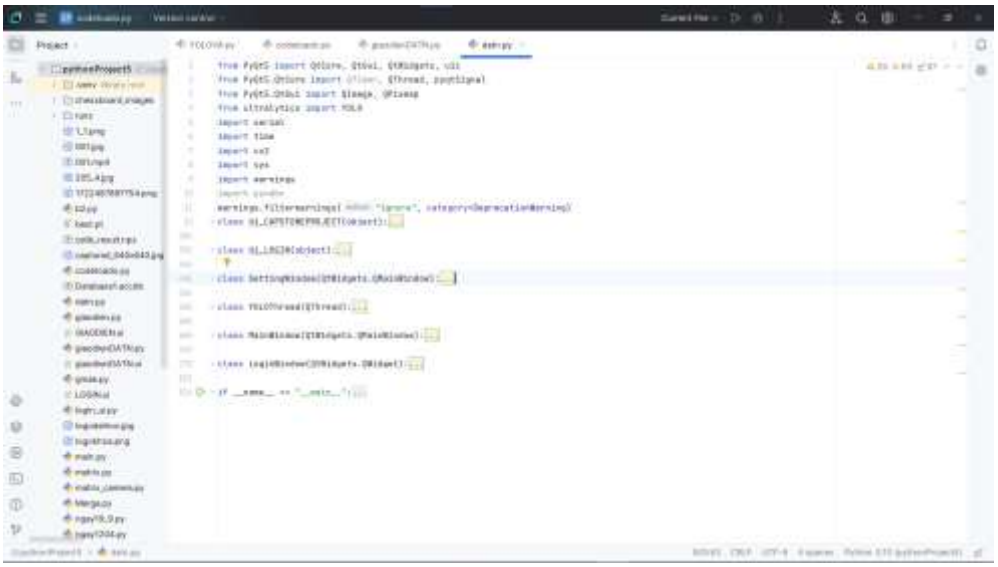

Figure 5.9 Pycharm Interface.

PyCharm is a powerful integrated development environment (IDE) specifically designed for Python programming. It offers advanced features such as code auto-completion, real-time error detection, version control integration, and a built-in terminal. PyCharm greatly enhances development efficiency, especially in large-scale or multi-file projects. In GUI-based applications, it allows seamless integration with tools like Qt Designer, making it easier to implement and manage interface logic.

5.6 Machine Learning Techniques and the Deep Learning Model YOLOv8:

5.6.1 Machine Learning:

Machine learning (ML) is a subfield of artificial intelligence that focuses on the development of algorithms and statistical models enabling computers to perform tasks without being explicitly programmed. By analyzing large datasets, machine learning systems are capable of identifying patterns, making predictions, and improving performance over time. ML techniques are broadly categorized into supervised learning, unsupervised learning, and reinforcement learning, each suited for different types of data and objectives. These techniques have been successfully applied in various domains, including image recognition, natural language processing, robotics, and medical diagnostics, demonstrating their potential to solve complex real-world problems efficiently and autonomously.

Main Branches of Machine Learning:

- Supervised Learning**:** Learning from labeled data. Commonly used for classification and regression tasks.
- Unsupervised Learning**:** Learning from unlabeled data. Applied in clustering and pattern detection.
- Semi-supervised Learning**:** Combines a small amount of labeled data with a large amount of unlabeled data.
- Reinforcement Learning**:** Learns through interaction with the environment by receiving rewards or penalties.
- Deep Learning**:** A subset of machine learning that employs deep neural networks to handle complex data representations.

5.6.2 Deep Learning Model Yolov8:

5.6.2.1 Introduce:

YOLO (You Only Look Once) is one of the leading algorithms for object detection tasks in images and videos. It stands out for its fast and accurate image processing capabilities, enabling real-time object recognition and classification.

Key advantages of YOLO:

- Speed**:** YOLO detects multiple objects in a single forward pass, rather than processing each image region separately, which significantly reduces computational time.
- Accuracy**:** It employs advanced neural networks to predict both the position and class of objects with high precision.
- Flexibility**:** YOLO can be customized to fit various tasks and requirements, ranging from simple object detection to the classification of complex categories.

Limitations of YOLO:

- Accuracy trade-off**:** Compared to two-stage detection methods, YOLO may exhibit slightly lower accuracy, especially when detecting small or obscure objects.
- Generalization**:** YOLO can struggle to detect new or unseen objects that were not included in the training data.

To overcome these limitations, researchers have continuously improved the algorithm, leading to the development of YOLOv8.

YOLOv8 offers several enhancements over previous versions:

- Speed**:** Achieves processing speeds of up to 400 FPS on GPU, making it more than twice as fast as YOLOv5.
- Accuracy**:** Reaches a mean Average Precision (mAP) of 64% on the COCO dataset, outperforming YOLOv5 by 4%.
- Versatility**:** Supports various image input types, including static images, videos, webcam feeds, and live streams.
- Ease of use: Provides a simple and user-friendly API, enabling straightforward integration into different applications.

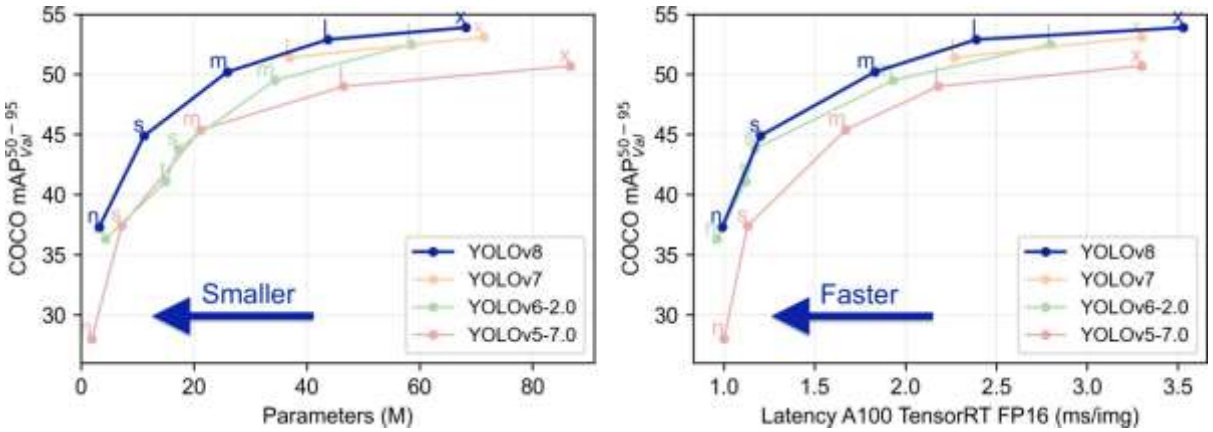

Figure 5.10 Comparison of YOLOv8 with Previous YOLO Versions.

This image compares the performance of different YOLO models based on two key aspects: model size (number of parameters) and speed (latency). Detailed observations of the charts are as follows:

Left Chart:

- Y-axis: Performance measured by mean Average Precision (mAP), a widely used metric for evaluating object detection accuracy. Higher mAP values indicate better model accuracy.
- X-axis: Model size measured by the number of parameters (in millions). Smaller models with fewer parameters are generally more compact and easier to deploy.
- Results: YOLOv8 achieves the highest mAP while maintaining the smallest model size, demonstrating an excellent balance between performance and size.
- Comparison with previous versions: YOLOv7, YOLOv6-2.0, and YOLOv5-7.0 all exhibit lower mAP values than YOLOv8. YOLOv7 has a larger model size than YOLOv8, whereas YOLOv6-2.0 and YOLOv5-7.0 are smaller but less accurate.

Right Chart:

- Y-axis: Performance still measured by mAP.
- X-axis: Latency measured in milliseconds (ms) per image. Lower latency indicates faster processing speed.
- Results: YOLOv8 continues to deliver the highest mAP with the lowest latency, indicating that it is not only accurate but also very fast.

- Comparison with previous versions: Other models have higher latency compared to YOLOv8. YOLOv7 has the highest latency, while YOLOv6-2.0 and YOLOv5-7.0 have lower latency but still exceed that of YOLOv8.

Conclusion:

- YOLOv8 is a highly efficient object detection model, achieving the highest mAP while maintaining a small size and fast processing speed.
- The charts illustrate significant advancements of YOLOv8 over previous versions.
- YOLOv8 is an excellent choice for applications requiring both high accuracy and real-time processing capabilities.

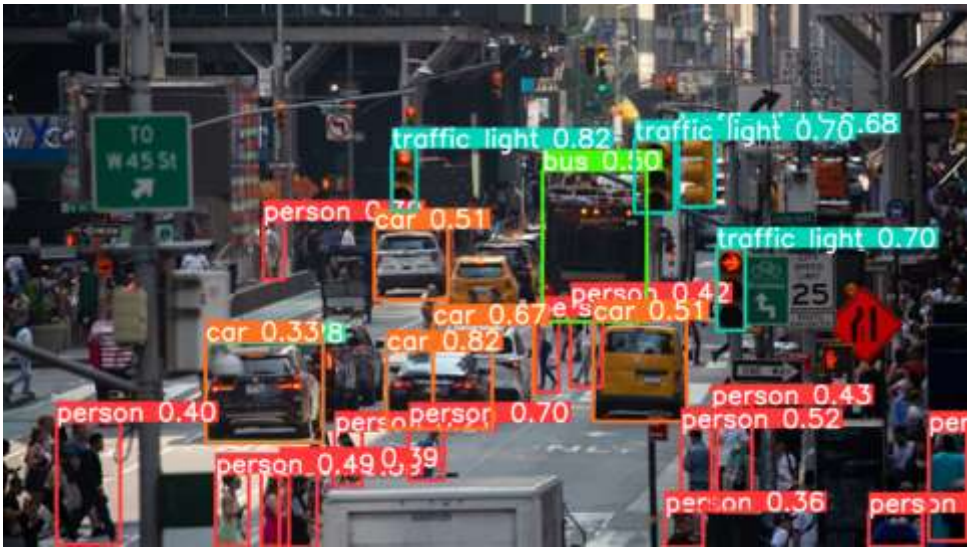

Figure 5.11 Object detection with YOLOv8.

5.6.2.2 Steps to Train the Model Using YOLOv8

a. Data Preparation

- Image Collection: Gather a dataset of images containing the objects you want the model to detect. In this project, we collected images using two methods. First, we searched for component images on the Internet. Simultaneously, we used a smartphone to capture images of the components to create the image dataset.
- Image Annotation: Label each object in the images with the corresponding class. For annotating images used in training, we utilized the Label Studio application, which facilitates efficient image labeling.

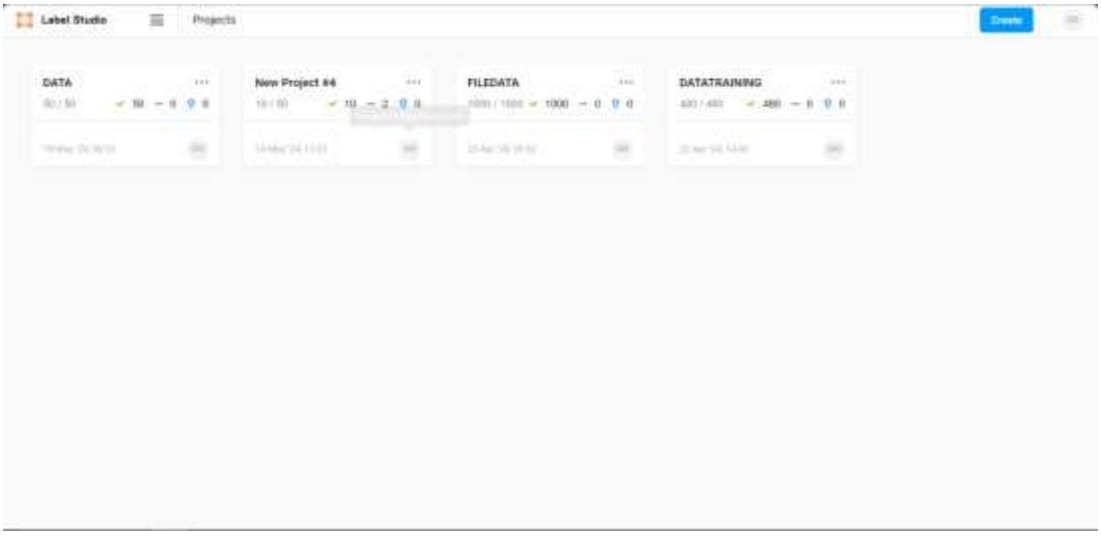

Figure 5.12 Label Studio Interface.

Table 5.1 List of Errors and Corresponding Labels.

Status	Error (Label)	Detailed Description
1	Correct	No error, complete and correct component types
2	Missing	Missing 1 terminal block + 2 terminal blocks
3		Missing 1 capacitor + 2 capacitors
4		Missing power resistor
5		Missing B688 transistor
6		Missing 7805 voltage regulator IC

Figure 5.13 Annotation of objects to be processed in the images.

- Create dataset file: Label Studio provides a built-in method to export annotated images into a dataset file for the subsequent training steps.

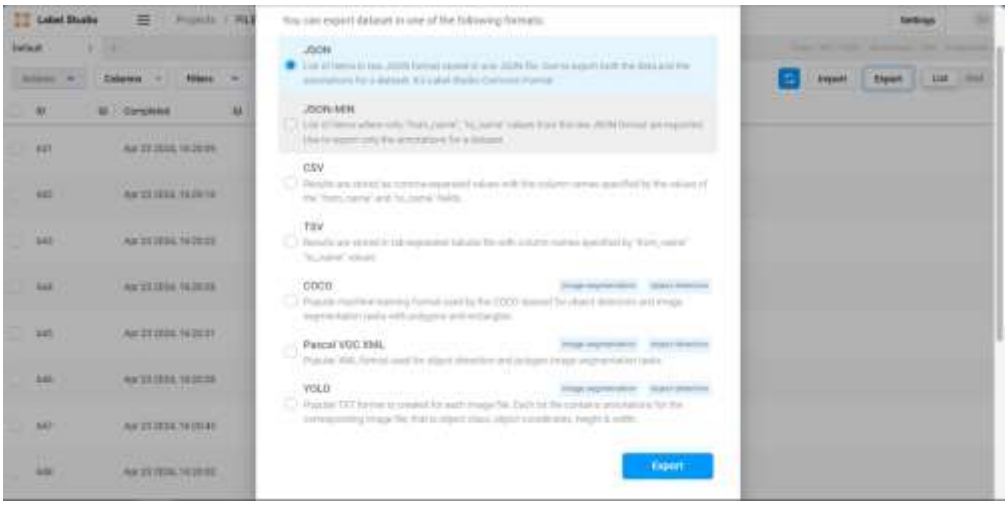

Figure 5.14 Export the dataset.

c. Setting up the environment and training the model:

To carry out model training, we utilized Google Colaboratory, a platform provided by Google for this purpose. Google Colaboratory, commonly known as Google Colab, is a free online platform that allows users to write and share Python code. It was developed by Google Research to support research and development in the fields of machine learning and data science.

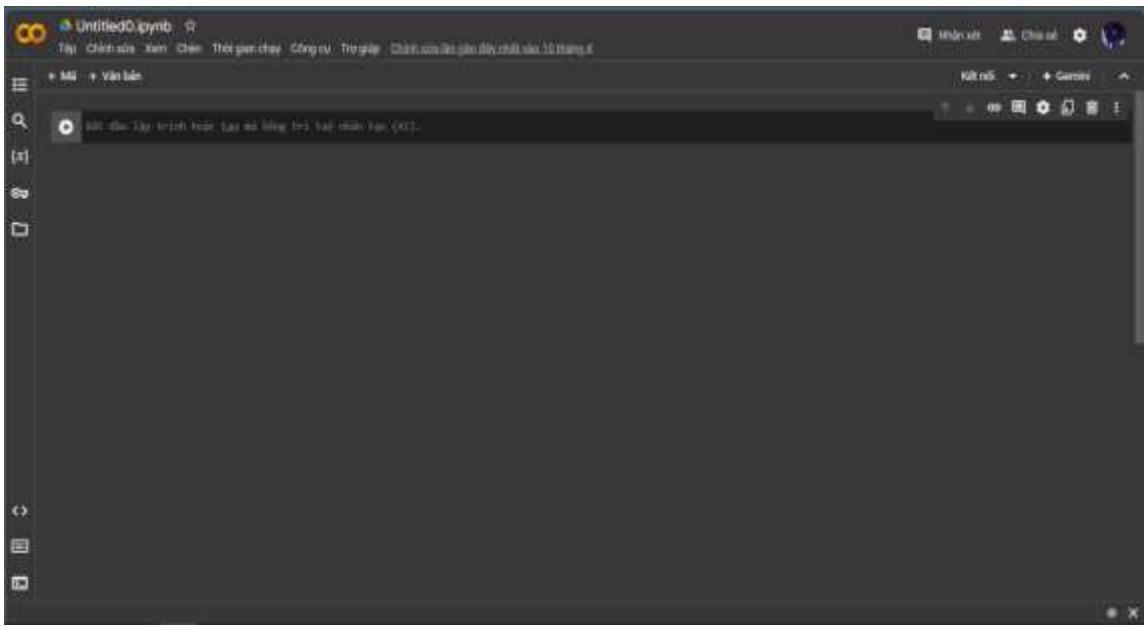

Figure 5.15 Google Colaboratory Interface.

After setting up the necessary environment for training, the next step is to install the required libraries to proceed with the training process.

Upon successful training on Google Colaboratory, we obtained files containing the training results.

5.6.2.3 Training Results:

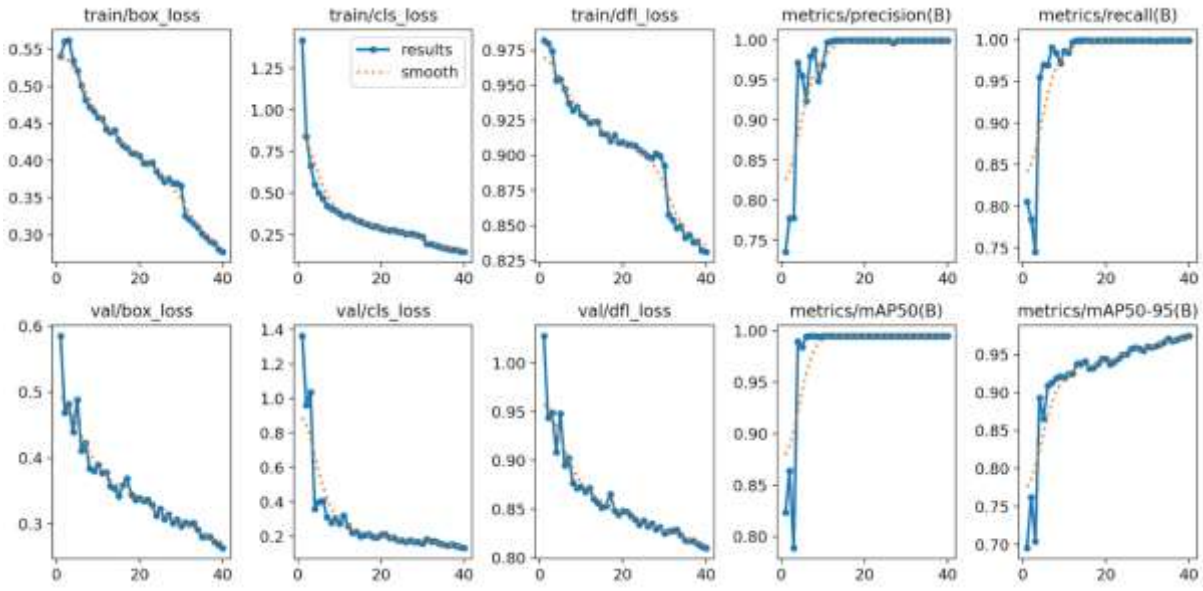

Figure 5.16 Model performance.

Metrics in the Charts:

- Line Chart Metrics:
    - *Train/box_loss*: Bounding box loss on the training dataset.

- *Train/cls_loss*: Classification loss on the training dataset.
- *Train/dfl_loss*: Overall loss on the training dataset.
- *Metrics/precision(B)*: Precision according to class B.
- *Metrics/recall(B)*: Recall according to class B.

– Bar Chart Metrics:

- *Val/box_loss*: Bounding box loss on the validation dataset.
- *Val/cls_loss*: Classification loss on the validation dataset.
- *Val/dfl_loss*: Overall loss on the validation dataset.
- *Metrics/mAP50(B)*: Mean Average Precision (mAP) at Intersection over Union (IoU) threshold of 50% for class B.
- *Metrics/mAP50-95(B)*: Mean Average Precision (mAP) averaged over IoU thresholds from 50% to 95% for class B.

Observations on the Charts:

– Accuracy:

- *Metrics/precision(B)*: The average precision for class B ranges from 0.9408 to 0.9983. This indicates a strong ability of the model to accurately identify objects within images.
- *Metrics/recall(B)*: The average recall for class B ranges from 0.7078 to 0.9983, demonstrating the model's effectiveness in detecting most objects in images.
- *Metrics/mAP50(B)*: The mean average precision at IoU 50% for class B ranges from 0.9172 to 0.9932, indicating excellent object detection accuracy at this threshold.
- *Metrics/mAP50-95(B)*: The mean average precision averaged across IoU thresholds from 50% to 95% ranges from 0.6150 to 0.7323, reflecting strong detection accuracy across multiple IoU levels.

– Loss:

- *Train/box_loss*: Bounding box loss on the training dataset varies between 1.3456 and 0.7666, indicating good localization accuracy in predicting object positions.

- *Train/cls_loss*: Classification loss on the training dataset ranges from 1.9146 to 0.6686, showing effective classification capability.
- *Train/dfl_loss*: Overall training loss ranges from 1.2553 to 1.1233, reflecting consistent performance in both localization and classification.
- *Val/box_loss*: Bounding box loss on the validation dataset ranges from 1.1876 to 0.6306, confirming high positional prediction accuracy on unseen data.
- *Val/cls_loss*: Classification loss on the validation dataset varies between 1.3843 and 0.6143, indicating reliable classification performance during validation.
- *Val/dfl_loss*: Overall validation loss is very low, ranging from 0.00048877 to 0.0012471, demonstrating excellent prediction accuracy in both localization and classification on the validation set.

– Performance:

- Overall, the YOLOv8 model achieves strong performance on both training and validation datasets.
- High precision, recall, and mAP metrics indicate the model's strong capability to accurately identify and detect objects.
- Low loss values demonstrate the model's effectiveness in accurately predicting object locations and classifications.

Table 5.2 Summary of YOLOv8 Training Results.

Metric	Epoch 1	Epoch 20	Epoch 40	Commented
mAP50	0.82358	0.995	**0.995**	Achieve near absolute accuracy
mAP50-95	0.69561	0.94461	**0.97445**	The model is stable even with strict criteria (IoU 0.5–0.95).
Precision (B)	0.73559	0.99975	**0.99996**	There are almost no false positives.

Metric	Epoch 1	Epoch 20	Epoch 40	Commented
Recall (B)	0.80587	0.99951	**0.99951**	Detects almost any object (extremely low False Negative).
Train/Box Loss	0.54043	0.40653	**0.27716**	The model learns the bounding box location well.
Val/Box Loss	0.58527	0.33881	**0.26362**	No overfitting, stable validation performance.
Train/Cls Loss	1.41985	0.28801	**0.14678**	Class classification ability is significantly improved.
Val/Cls Loss	1.36189	0.21002	**0.13380**	High confidence in classification on validation set.

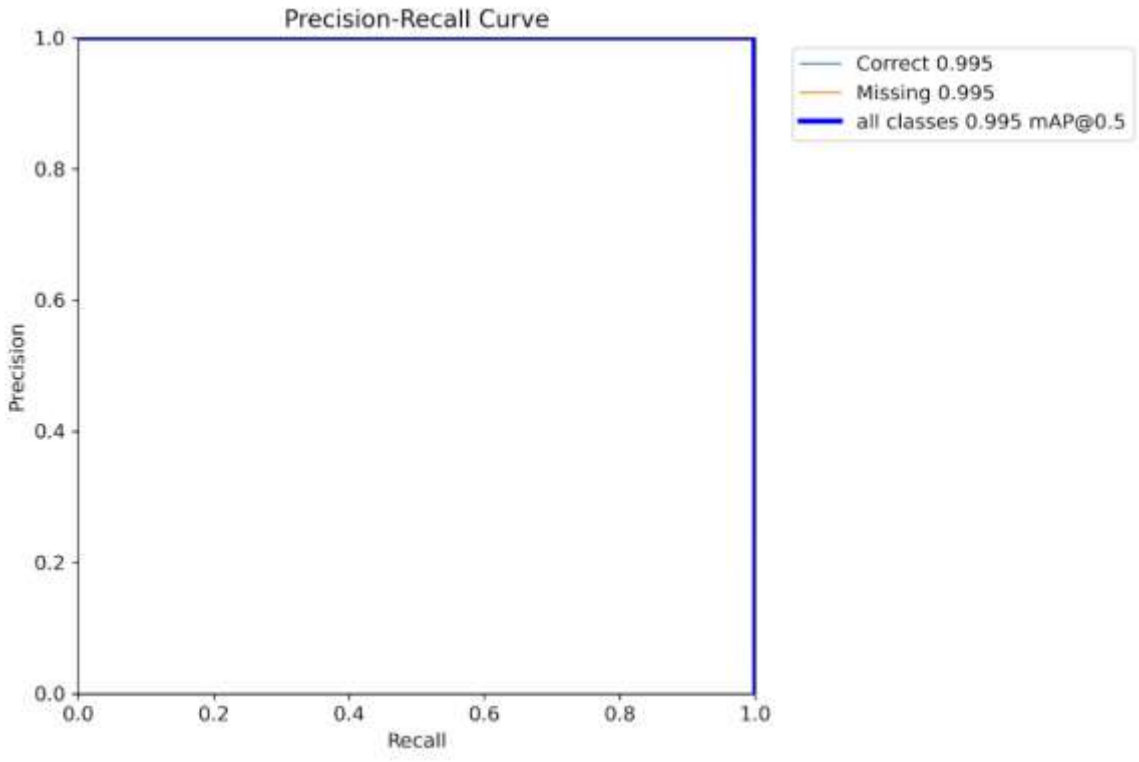

Figure 5.17 Precision-Recall Curve Diagram.

- **Y-axis:** Precision is the proportion of true positive predictions among all positive predictions.
- **X-axis:** Recall is the proportion of true positive predictions among all actual positive instances.

The graph shows that the model achieves very high Precision and Recall, nearly reaching 1.0 for all classes (Correct and Missing) as well as overall. This indicates that the model classifies effectively, accurately distinguishing between classes and missing very few true positive cases.

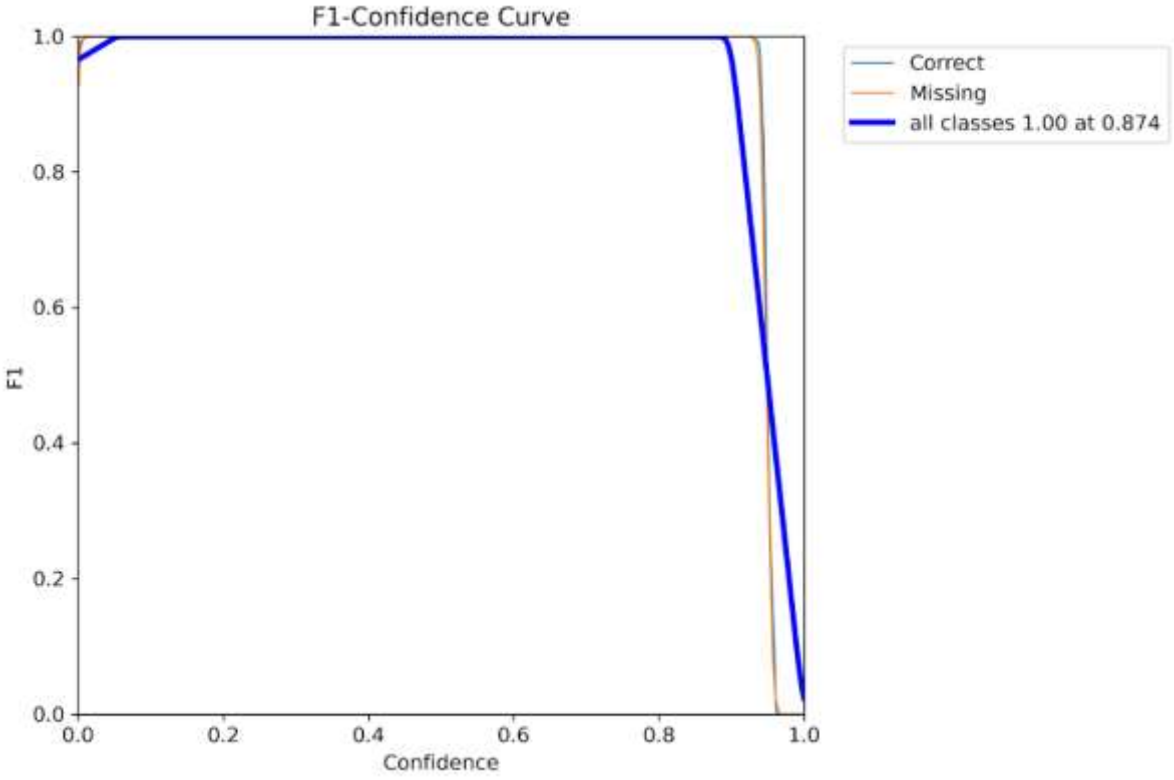

Figure 5.18 F1-Confidence Curve Diagram.

- Y-axis: F1 Score, a composite performance metric that combines Precision and Recall. The higher the F1 Score, the more effective the model.
- X-axis: Confidence represents the model's certainty in its predictions. Higher confidence indicates greater trust in the model's output.

The graph shows:

- High performance: The model's F1 scores are very high—nearly reaching 1.0 for all classes (Correct, Missing) and overall. This suggests the model classifies

effectively, accurately distinguishing between classes and missing few true positives.

- Strong confidence handling: The F1 score remains high until the confidence level becomes quite low, indicating that the model is capable of making accurate predictions even when its confidence is not at the maximum.

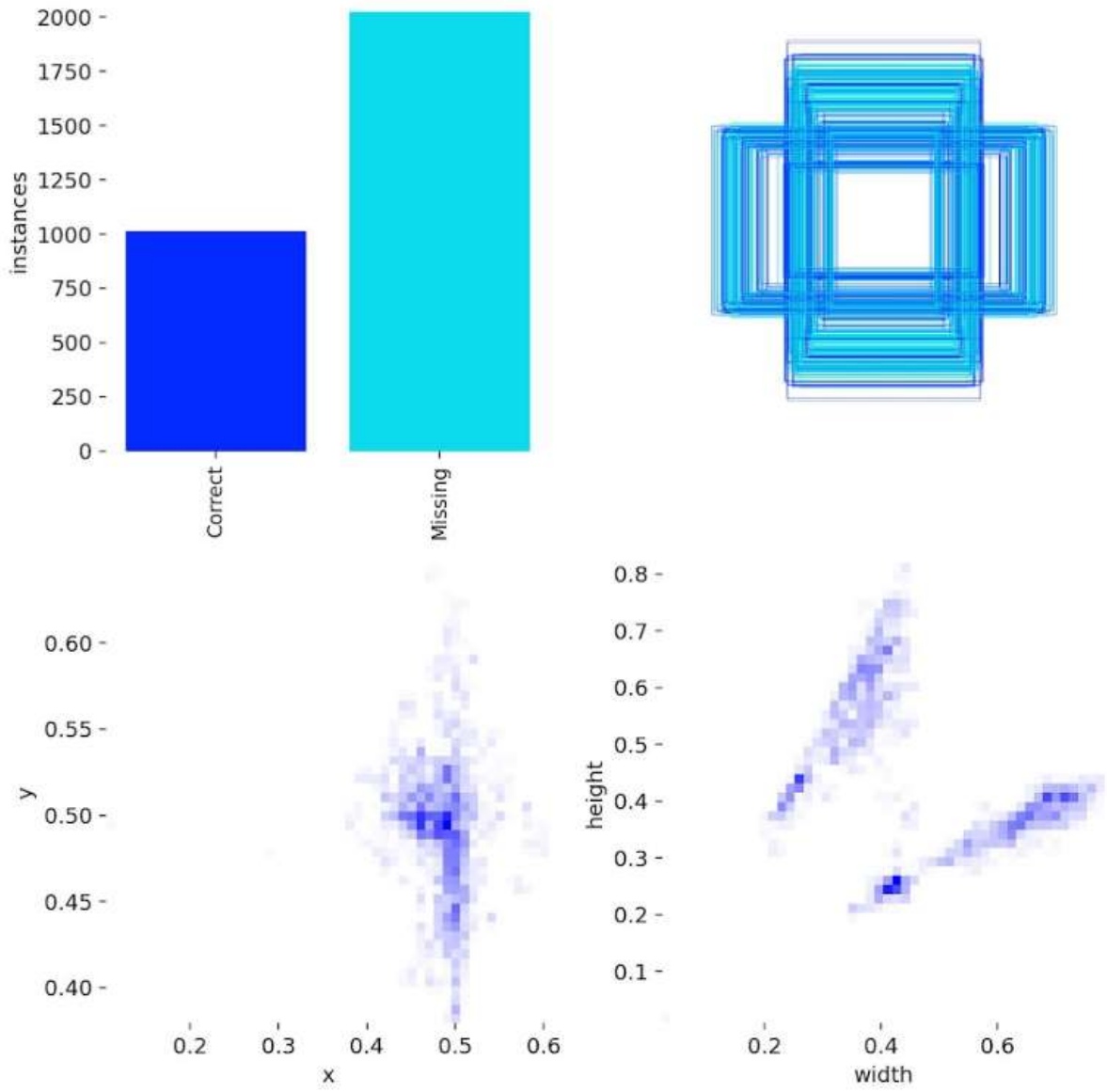

Figure 5.19 Diagram of the Distribution and Size of Detected Bounding Boxes.

It consists of three components:

- Bar chart: Displays the number of detected bounding boxes for each class (Correct, Missing).

- Scatter plot: Shows the spatial distribution of bounding boxes along the x and y axes, with color indicating density.
- Overlay diagram**:** Illustrates the detected bounding boxes with their respective sizes and positions.

Explanation:

- Bar chart: Indicates that the Capacitor class has the highest number of bounding boxes, followed by Missing, while Correct has the fewest. This may suggest that the training dataset contains more examples of Correct compared to Missing.
- Scatter plot: Demonstrates that bounding boxes are distributed fairly evenly along the x and y axes, but tend to cluster in specific regions. For instance, there is a noticeable concentration of bounding boxes in the lower part of the images.
- Overlay diagram: Shows bounding boxes of varying sizes and positions. It can be observed that bounding boxes for Capacitors and Resistors are relatively large, whereas those for Relays are smaller in size.

  Actual Results Detected by the Model After Training:

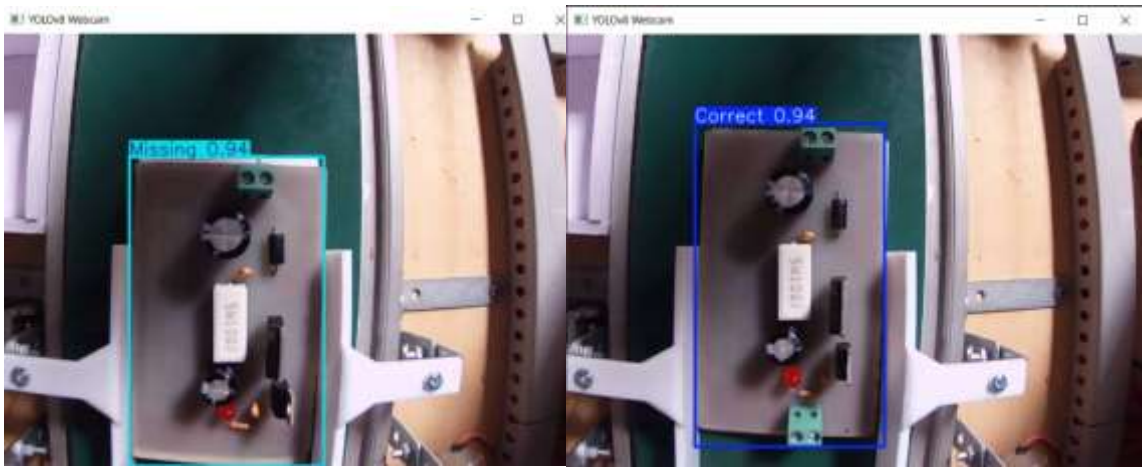

Figure 5.20 Real-World Image Detection.

However, in order for the robot model to successfully detect and pick up objects, after fully training the model and downloading the trained weights to our local machine, we made additional modifications to the code. These modifications allow the robot to determine the coordinates and orientation of the objects and transmit this information to the group's robotic system.

CHAPTER 6 CONCLUSION AND FUTURE WORKS.

6.1 Results:

- The system closely meets the objectives and requirements set out at the beginning of the report.
- Successfully designed and fabricated a 4-degree-of-freedom robotic arm model.
- Achieved accurate object classification and successful picking of items on the conveyor belt.

6.2 Operational Efficiency:

- **Production speed**: The system consistently achieves a processing rate of 3-4 circuits per minute.
- **Standards compliance rate**: The classification accuracy exceeds 90%, meeting the design requirements.
- **Continuous operation**: The system has been tested for long-term operation, running smoothly for extended periods without interruption or the need for immediate maintenance.
- **Flexibility**: The machine has successfully handled various types of circuits, demonstrating its flexibility and adaptability.

6.3 System Performance Evaluation:

6.3.1 System Complexity and Operability

The robotic system was developed with an emphasis on ease of use, modularity, and efficient operation. The key performance indicators related to system complexity and operability are as follows:

- **Operator Training Time:** New users were able to learn and operate the robotic arm efficiently within 5 minutes, which aligns well with the design objective of creating a user-friendly interface and control logic.
- **User Feedback:** More than 85% of test users rated the system as easy to use and intuitive, surpassing the initial design goal of 80% positive usability feedback. This result confirms the system's operational accessibility even for non-expert users.

6.3.2 Classification Accuracy and Mechanical Precision

Accuracy in classification and movement precision are crucial to ensure reliable performance in circuit board handling and sorting. The system was evaluated based on the following criteria:

- **Positional Accuracy**: The robotic arm consistently maintained a movement deviation within ±1 mm during pick-and-place tasks, meeting the mechanical precision requirements for small electronic components.
- **Classification Accuracy:** Leveraging machine learning for object recognition, the system achieved over 90% accuracy in identifying and sorting circuits into correct categories (e.g., Correct, Missing, Wrong Pin), fulfilling the project's performance target.
- **Repeatability:** The arm demonstrated stable and repeatable motion paths during extended operation, indicating reliable mechanical performance and consistent classification behavior.

Overall, these results demonstrate that the 4-DOF robotic arm system is technically sound and practically feasible for real-world applications in small-scale automated circuit classification and assembly environments.

6.4 Future Development Directions:

While the current prototype has met the initial goals of functionality and accuracy, there remain several areas for future improvement to enhance system performance, scalability, and industrial applicability.

6.4.1 Mechanical Improvements

- Structural Optimization: Re-engineer specific mechanical components to reduce weight and improve modularity for easier upgrades, transport, and maintenance.
- Enhanced Durability: Use more durable materials and surface treatments for high-stress joints and rotating parts to extend the system's operating life.
- Noise Reduction: Fine-tune mechanical assemblies and apply damping techniques to reduce noise levels during high-speed operation.
- Precision Upgrades: Consider upgrading key joints to closed-loop stepper motors or servo motors for higher positioning accuracy, particularly during fast pick-and-place cycles.

6.4.2 Control System Enhancements

- Advanced User Interface: Integrate a touchscreen-based Human-Machine Interface (HMI) with graphical status updates and real-time diagnostics to improve usability.
- Adaptive Automation: Implement additional sensors (e.g., limit switches, encoders) and feedback loops to support adaptive error correction and enhance autonomy.
- Data Logging and IoT Connectivity: Enable real-time data logging and wireless connectivity (e.g., Wi-Fi, Bluetooth) for production monitoring, fault tracking, and integration into smart factory systems.
- Safety Features: Add emergency stop buttons, overload protection, and motion fault detection to ensure safe operation in industrial or educational environments.
- Vision System Expansion: Improve the machine learning model and vision system to handle more complex classification tasks, such as detecting soldering defects or identifying additional types of circuit errors.

By pursuing these development directions, the robotic arm system can evolve into a robust, intelligent, and scalable solution for industrial-level automated circuit inspection and classification.

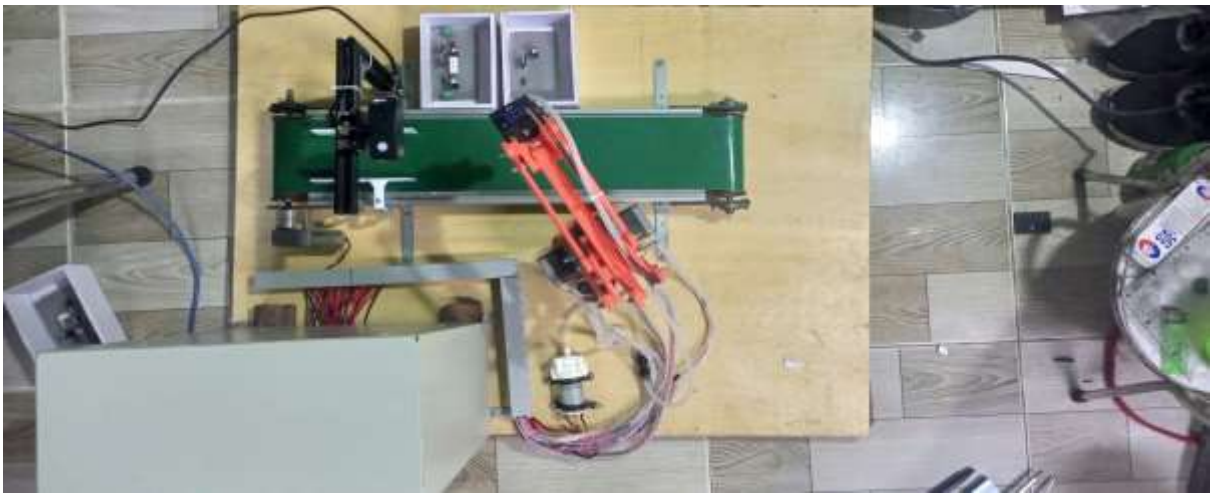

Figure 6.1 Physical Model.

REFERENCES

[1] Unknown author, *Sự phát triển của robot công nghiệp*, No publication date, https://phukiencongnghiep.com.vn/su-phat-trien-cua-robot-cong-nghiep/ [Accessed: May 15, 2025].

[2] Unknown author, *Ứng dụng robot trong ngành công nghiệp ô tô*, No publication date, https://intechvietnam.com/tin-tuc/tin-cong-nghe/ung-dung-robot-trong-nganh-cong-nghiep-o-to-494/ [Accessed: May 15, 2025].

[3] Unknown author, *SCARA LS10-B803S/RC90-B*, No publication date. https://robot.epson.eu/products/robot/scara-ls10-b803s-rc90-b [Accessed: May 15, 2025].

[4] Unknown author, *TB6600 Stepper Motor Driver Module – Pinout, Features, Datasheet, Working & Application*, No publication date. https://components101.com/modules/tb6600-stepper-motor-driver-module-pinout-features-datasheet-working-application-alternative [Accessed: May 10, 2025].

[5] Unknown author, *TB6600 Stepper Motor Driver Module*, No publication date. https://www.watelectronics.com/tb6600-stepper-motor-driver-module/ [Accessed: May 10, 2025].

[6] Pham Dang Phuoc, *Robot Công nghiệp*, 1st ed. Hanoi: Construction Publishing House, 2007.

[7] Nguyen Manh Tien, *Điều khiển Robot công nghiệp*, 1st ed. Hanoi: Science and Technology Publishing House, 2007.

[8] Ultralytics, *YOLOv8*, No publication date. https://docs.ultralytics.com/vi/models/yolov8/ [Accessed: May 10, 2025].

[9] Arduino, *Getting Started with Arduino IDE 2*, No publication date. https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/ [Accessed: May 20, 2025].

[10] Ultralytics, *Ultralytics GitHub Repository*, No publication date. https://github.com/ultralytics/ultralytics [Accessed: May 20, 2025].

[11] JetBrains, *PyCharm*, No publication date. https://www.jetbrains.com/pycharm/ [Accessed: May 22, 2025].

[12]Hongjin Wu, Ruoshan Lei, Yibing Peng, "PCBNet: A Lightweight Convolutional Neural Network for Defect Inspection in Surface Mount Technology," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, Art. no. 3518314, July 22, 2022, doi: 10.1109/TIM.2022.3193183.

[13]Si Shen, Juan Wang, and Fan Yang, "PCB Component Recognition based on Improved YOLOv8s for Lightweight Detection," *International Core Journal of Engineering*, vol. 11, no. 4, pp. 7–14, 2025. DOI: 10.6919/ICJE.202504_11(4).0002.

[14]Mitsubishi Electric, *BFP-A3616A Manual*, No publication date. https://www.mitsubishielectric.com/dl/fa/document/manual/school_text/bfp-a3616/bfp-a3616a.pdf [Accessed: Jun. 5, 2025].

[15]ABB, *IRB 1200 – Articulated robots*, No publication date. https://new.abb.com/products/robotics/robots/articulated-robots/irb-1200 [Accessed: June 5, 2025].

[16]Mì AI, *Thử nghịch YOLOv8 và train với dữ liệu cá nhân – Mì AI*, February 1, 2023. https://youtu.be/Mpsdby8zCTY?si=gyB6iV6sJDqwz-Br [Accessed: June 5, 2025].

APPENDIX

ARDUINO CODE:

```
#include <Servo.h>
#include <AccelStepper.h>
// ====== KHAI BÁO SERVO VÀ STEPPER ======
Servo servo4;
AccelStepper stepper1(AccelStepper::DRIVER, 2, 3);
AccelStepper stepper2(AccelStepper::DRIVER, 4, 5);
AccelStepper stepper3(AccelStepper::DRIVER, 6, 7);
// ====== ĐỊNH NGHĨA CHÂN KẾT NỐI ======
const int servo4Pin = 8;
const int relayPin = 24;
const int limitSwitch1 = 36, limitSwitch2 = 44, limitSwitch3 =
40;
const int ENA = 30;
const int IN1 = 10;
const int IN2 = 11;
// ====== THÔNG SỐ KỸ THUẬT ======
float maxSpeed = 2600.0;
float homeSpeed = 800.0;
float acceleration = 1200.0;
float homeAcceleration = 500.0;
const int stepsPerDegree = 40;
const int conveyorFixedSpeed = 200;
// ====== ĐỊNH NGHĨA VỊ TRÍ ======
long posHome[4] = {0, 0, 0, 0};
long posA[4] = {15, -66, 61, 0};
long posB[4] = {20, -20, 10, 0};
long posC[4] = {24, -104, 32, 108};
long posD[4] = {30, -30, 20, 0};
long posE[4] = {42, -88, 45, 88};
long posF[4] = {45, -40, 20, 0};
long posG[4] = {64, -80, 50, 75};
long posH[4] = {40, -50, 30, 0};
long posI[4] = {22, -50, 10, 0};
long posJ[4] = {64, -40, 20, 0};
long posK[4] = {9, -50, 45, 0};
// ====== BIẾN TRẠNG THÁI ======
long currentPos[4] = {0, 0, 0, 0};
bool pumpActive = false, isHomed = false;
bool conveyorRunning = false;
bool systemPaused = false;
// ====== BIẾN CHU KỲ BĂNG TẢI ======
bool conveyorCycleActive = false;
unsigned long cycleStartTime = 0;
```

```
int cyclePhase = 0;
const unsigned long CYCLE_DURATION = 2000;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(200);
  updateStepperSettings();
  pinMode(limitSwitch1, INPUT_PULLUP);
  pinMode(limitSwitch2, INPUT_PULLUP);
  pinMode(limitSwitch3, INPUT_PULLUP);
  pinMode(relayPin, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  servo4.attach(servo4Pin);
  servo4.write(0);
  digitalWrite(relayPin, LOW);
  startConveyor();
  printMenu();
}
void loop() {
  handleConveyorCycle();
  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');
    input.trim();
    if (input.length() == 0) {
      return;
    }
    // Phân loại command type
    if (input.indexOf(':') > 0) {
      handleConfigCommand(input);
    } else {
      handleNumericCommand(input);
    }
  }
}
// ====== XỬ LÝ CHU KỲ BĂNG TẢI ======
void handleConveyorCycle() {
  if (!conveyorCycleActive || systemPaused) return;
  unsigned long elapsed = millis() - cycleStartTime;
  static unsigned long lastDebug = 0;
  if (millis() - lastDebug > 1000) {
    lastDebug = millis();
  }
  switch (cyclePhase) {
    case 0:
      if (elapsed >= CYCLE_DURATION) {
```

```
        stopConveyor();
        cyclePhase = 1;
        cycleStartTime = millis(); // Reset timer cho phase
tiếp theo
      }
      break;
    case 1:
      if (elapsed >= CYCLE_DURATION) {
        startConveyor();
        cyclePhase = 2;
        conveyorCycleActive = false;
      }
      break;
    case 2:
      conveyorCycleActive = false;
      break;
  }
}
// ====== KHỞI TẠO CHU KỲ BĂNG TẢI ======
void startConveyorCycle() {
  if (systemPaused) {
    return;
  }
  conveyorCycleActive = true;
  cycleStartTime = millis();
  cyclePhase = 0;
  // Đảm bảo băng tải chạy trong phase 0
  startConveyor();
}
// ====== DỪNG CHU KỲ BĂNG TẢI ======
void stopConveyorCycle() {
  conveyorCycleActive = false;
  cyclePhase = 0;
}
void handleConfigCommand(String input) {
  if (input.startsWith("MAXSPEED:")) {
    setMaxSpeed(input.substring(9).toFloat());
  } else if (input.startsWith("HOMESPEED:")) {
    setHomeSpeed(input.substring(10).toFloat());
  } else if (input.startsWith("ACCEL:")) {
    setAcceleration(input.substring(6).toFloat());
  } else if (input.startsWith("HOMEACCEL:")) {
    setHomeAcceleration(input.substring(10).toFloat());
  } else {
    Serial.println("Unknown config command");
  }
```

```
}
void handleNumericCommand(String input) {
  int command = -1;
  if (input == "0") command = 0;
  else if (input == "1") command = 1;
  else if (input == "2") command = 2;
  else if (input == "3") command = 3;
  else if (input == "4") command = 4;
  else if (input == "5") command = 5;
  else if (input == "6") command = 6;
  else if (input == "100") command = 100;
  else if (input == "101") command = 101;
  else {
    Serial.println("Invalid command! Send number 0-6, 100-101
or config commands");
    return;
  }
  // Handle pause/resume first
  if (command == 100) {
    pauseSystem();
    return;
  }
  if (command == 101) {
    resumeSystem();
    return;
  }
  executeCommand(command);
  clearSerialBuffer();
}
// ====== CẤU HÌNH THÔNG SỐ ======
void setMaxSpeed(float newMaxSpeed) {
  if (newMaxSpeed <= 0 || newMaxSpeed > 5000) {
    Serial.println("Invalid maxSpeed! Range: 1-5000");
    return;
  }
  maxSpeed = newMaxSpeed;
  updateStepperSettings();
}
void setHomeSpeed(float newHomeSpeed) {
  if (newHomeSpeed <= 0 || newHomeSpeed > maxSpeed) {
    Serial.println("Invalid homeSpeed! Range: 1-maxSpeed");
    return;
  }
  homeSpeed = newHomeSpeed;
}
void setAcceleration(float newAcceleration) {
```

```
  if (newAcceleration <= 0 || newAcceleration > 5000) {
    Serial.println("Invalid acceleration! Range: 1-5000");
    return;
  }
  acceleration = newAcceleration;
  updateStepperSettings();
}
void setHomeAcceleration(float newHomeAcceleration) {
  if (newHomeAcceleration <= 0 || newHomeAcceleration > 5000) {
    Serial.println("Invalid homeAcceleration! Range: 1-5000");
    return;
  }
  homeAcceleration = newHomeAcceleration;
}
void updateStepperSettings() {
  stepper1.setMaxSpeed(maxSpeed);
  stepper1.setAcceleration(acceleration);
  stepper2.setMaxSpeed(maxSpeed);
  stepper2.setAcceleration(acceleration);
  stepper3.setMaxSpeed(maxSpeed);
  stepper3.setAcceleration(acceleration);
}
// ====== TẠM DỪNG/KHỞI ĐỘNG HỆ THỐNG ======
void pauseSystem() {
  stepper1.stop();
  stepper2.stop();
  stepper3.stop();
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  servo4.write(0);
  stopConveyor();
  stopConveyorCycle();
  systemPaused = true;
}
void resumeSystem() {
  startConveyor();
  stopConveyorCycle();
  systemPaused = false;
  isHomed = false;
}
// ====== ĐIỀU KHIỂN BĂNG TẢI ======
void startConveyor() {
  if (systemPaused) return;
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, conveyorFixedSpeed);
```

```
  conveyorRunning = true;
}
void stopConveyor() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);
  conveyorRunning = false;
}
void reverseConveyor() {
  if (systemPaused) return;
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, conveyorFixedSpeed);
  conveyorRunning = true;
}
// ====== XỬ LÝ LỆNH ======
void executeCommand(int command) {
  Serial.print("Executing: ");
  Serial.println(command);
  switch (command) {
    case 0:
      stopConveyorCycle();
      setHome();
      break;
    case 1:
      stopConveyorCycle();
      if (checkHomed()) moveToPositionA();
      break;
    case 2:
      if (checkHomed()) {
        startConveyorCycle();
        executeSequence1();
      }
      break;
    case 3:
      if (checkHomed()) {
        startConveyorCycle();
        executeSequence2();
      }
      break;
    case 4:
      if (checkHomed()) {
        startConveyorCycle();
        executeSequence3();
      }
      break;
```

```
    case 5:
      stopConveyorCycle();
      if (checkHomed()) returnHomePriority();
      break;
    case 6:
      stopConveyorCycle();
      if (checkHomed()) returnToPositionA();
      break;
  }
}
// ====== THIẾT LẬP GIA TỐC ======
void setHomingAcceleration() {
  stepper1.setAcceleration(homeAcceleration);
  stepper2.setAcceleration(homeAcceleration);
  stepper3.setAcceleration(homeAcceleration);
}
void setNormalAcceleration() {
  stepper1.setAcceleration(acceleration);
  stepper2.setAcceleration(acceleration);
  stepper3.setAcceleration(acceleration);
}
// ====== THIẾT LẬP HOME ======
void setHome() {
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  servo4.write(0);
  delay(1000);
  setHomingAcceleration();
  homeAllJoints();
  for (int i = 0; i < 4; i++) currentPos[i] = posHome[i];
  stepper1.setCurrentPosition(0);
  stepper2.setCurrentPosition(0);
  stepper3.setCurrentPosition(0);
  setNormalAcceleration();
  isHomed = true;
  if (!conveyorRunning) {
    startConveyor();
  }
}
void homeAllJoints() {
  stepper1.setMaxSpeed(homeSpeed);
  stepper2.setMaxSpeed(homeSpeed);
  stepper3.setMaxSpeed(homeSpeed);
  stepper1.setSpeed(-homeSpeed);
  stepper2.setSpeed(homeSpeed);
  stepper3.setSpeed(-homeSpeed);
```

```
  bool homeDone[3] = {false, false, false};
  unsigned long startTime = millis();
  while (!(homeDone[0] && homeDone[1] && homeDone[2]) &&
         (millis() - startTime < 30000)) {
    if (systemPaused) {
      stepper1.setSpeed(0);
      stepper2.setSpeed(0);
      stepper3.setSpeed(0);
      return;
    }
    if (!homeDone[0] && digitalRead(limitSwitch1) == HIGH) {
      stepper1.runSpeed();
    } else if (!homeDone[0]) {
      homeDone[0] = true;
      stepper1.setSpeed(0);
    }
    if (!homeDone[1] && digitalRead(limitSwitch2) == HIGH) {
      stepper2.runSpeed();
    } else if (!homeDone[1]) {
      homeDone[1] = true;
      stepper2.setSpeed(0);
    }
    if (!homeDone[2] && digitalRead(limitSwitch3) == HIGH) {
      stepper3.runSpeed();
    } else if (!homeDone[2]) {
      homeDone[2] = true;
      stepper3.setSpeed(0);
    }
    delay(1);
  }
  stepper1.move(30); stepper2.move(-30); stepper3.move(30);
  while (stepper1.isRunning() || stepper2.isRunning() ||
stepper3.isRunning()) {
    if (systemPaused) break;
    stepper1.run(); stepper2.run(); stepper3.run();
  }
  stepper1.setMaxSpeed(maxSpeed);
  stepper2.setMaxSpeed(maxSpeed);
  stepper3.setMaxSpeed(maxSpeed);
}
// ====== CHUYỂN ĐỘNG CƠ BẢN ======
bool checkHomed() {
  if (systemPaused) {
    Serial.println("SYSTEM PAUSED! Send command 101 to
resume.");
    return false;
```

```
  }
  if (!isHomed) {
    Serial.println("ERROR: Not homed! Run command 0 first!");
    return false;
  }
  return true;
}
void moveToPositionA() {
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  moveToPosition(posF);
}
void returnToPositionA() {
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  moveToPosition(posA);
}
void returnHome() {
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  setHomingAcceleration();
  moveToPosition(posHome);
  setNormalAcceleration();
}
void returnHomePriority() {
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  stepper1.stop();
  stepper2.stop();
  stepper3.stop();
  while (stepper1.isRunning() || stepper2.isRunning() ||
stepper3.isRunning()) {
    stepper1.run();
    stepper2.run();
    stepper3.run();
  }
  setHomingAcceleration();
  moveToPosition(posHome);
  setNormalAcceleration();
}
// ====== CÁC SEQUENCE HOẠT ĐỘNG ======
void executeSequence1() {
  digitalWrite(relayPin, HIGH);
  pumpActive = true;
  moveToPosition(posA);
  moveToPosition(posB);
```

```
  rotateServo4(108);
  delay(300);
  moveToPosition(posC);
  delay(100);
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  delay(1000);
  moveToPosition(posI);
  setHomingAcceleration();
  moveToPosition(posHome);
  setNormalAcceleration();
}
void executeSequence2() {
  digitalWrite(relayPin, HIGH);
  pumpActive = true;
  moveToPosition(posA);
  moveToPosition(posD);
  rotateServo4(88);
  delay(300);
  moveToPosition(posE);
  delay(100);
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  delay(1000);
  moveToPosition(posH);
  setHomingAcceleration();
  moveToPosition(posHome);
  setNormalAcceleration();
}
void executeSequence3() {
  digitalWrite(relayPin, HIGH);
  pumpActive = true;
  moveToPosition(posA);
  moveToPosition(posF);
  rotateServo4(75);
  delay(500);
  moveToPosition(posG);
  delay(100);
  digitalWrite(relayPin, LOW);
  pumpActive = false;
  delay(1000);
  moveToPosition(posJ);
  setHomingAcceleration();
  moveToPosition(posHome);
  setNormalAcceleration();
}
```

```
// ====== HÀM HỖ TRỢ ======
void moveToPosition(long targetPos[4]) {
  if (systemPaused) return;
  long targetSteps[3];
  for (int i = 0; i < 3; i++) {
    targetSteps[i] = targetPos[i] * stepsPerDegree;
  }
  stepper1.moveTo(targetSteps[0]);
  stepper2.moveTo(targetSteps[1]);
  stepper3.moveTo(targetSteps[2]);
  while (stepper1.isRunning() || stepper2.isRunning() ||
stepper3.isRunning()) {
    if (systemPaused) {
      stepper1.stop();
      stepper2.stop();
      stepper3.stop();
      break;
    }
    stepper1.run(); stepper2.run(); stepper3.run();
    handleConveyorCycle();
  }
  if (!systemPaused) {
    for (int i = 0; i < 3; i++) currentPos[i] = targetPos[i];

    if (targetPos[3] != currentPos[3]) {
      servo4.write(targetPos[3]);
      currentPos[3] = targetPos[3];
      delay(500);
    }
  }
}
void rotateServo4(int angle) {
  if (systemPaused) return;
  angle = constrain(angle, 0, 180);
  int currentAngle = servo4.read();
  if (currentAngle < angle) {
    for (int pos = currentAngle; pos <= angle; pos++) {
      if (systemPaused) break;
      servo4.write(pos);
      delay(15);
      handleConveyorCycle();
    }
  } else {
    for (int pos = currentAngle; pos >= angle; pos--) {
      if (systemPaused) break;
      servo4.write(pos);
```

```
      delay(15);
      handleConveyorCycle();
    }
  }
  if (!systemPaused) {
    currentPos[3] = angle;
  }
}
void clearSerialBuffer() {
  while(Serial.available() > 0) {
    Serial.read();
    delay(1);
  }
}
```

PYTHON CODE:

```
from PyQt5 import QtCore, QtGui, QtWidgets, uic
from PyQt5.QtCore import QTimer, QThread, pyqtSignal
from PyQt5.QtGui import QImage, QPixmap
from ultralytics import YOLO
import serial
import time
import cv2
import sys
import warnings
from openpyxl import Workbook
import datetime
warnings.filterwarnings("ignore",
category=DeprecationWarning)
class SettingWindow(QtWidgets.QMainWindow):
        def __init__(self, serial_port):
                super().__init__()
                try:
                        uic.loadUi('SETTING.ui', self)
                        self.setWindowTitle("SETTING")
                        self.serial = serial_port

self.ENTER.clicked.connect(self.send_settings_to_arduino)

self.setFocusPolicy(QtCore.Qt.StrongFocus)
                except Exception as e:
                        print(f"Error initializing Setting
Window: {e}")

QtWidgets.QMessageBox.critical(None, "Error", f"Unable to
initialize Setting Window: {e}")
        def send_settings_to_arduino(self):
                try:
```

```
                            accel =
self.ACCELERATION.text().strip()
                            home_speed =
self.HOMESPEED.text().strip()
                            max_speed =
self.ROBOTSPEED.text().strip()

                            if not accel or not home_speed or
not max_speed:

QtWidgets.QMessageBox.warning(self, "Warning", "Please
enter complete information.")
                                    return
                            # Kiểm tra giá trị số
                            try:
                                    float(accel)
                                    float(home_speed)
                                    float(max_speed)
                            except ValueError:

QtWidgets.QMessageBox.warning(self, "Warning", "Please
enter a valid numeric value.")
                                    return
                            # Kiểm tra serial còn mở
                            if self.serial and
self.serial.is_open:
                                    try:
                                            # Gửi dữ liệu với
delay nhỏ giữa các lệnh

self.serial.write(f"ACCEL:{accel}\n".encode())
                                            time.sleep(0.1)

self.serial.write(f"HOMESPEED:{home_speed}\n".encode())
                                            time.sleep(0.1)

self.serial.write(f"MAXSPEED:{max_speed}\n".encode())

                                            # Hiển thị thông
báo thành công

QtWidgets.QMessageBox.information(self, "Success","Data
sent successfully!")
                                            # Không đóng cửa
sổ, chỉ reset form nếu muốn
                                            #
self.reset_form()
                                    except
```

```
serial.SerialException as e:

QtWidgets.QMessageBox.critical(self, "Serial Error",
f"Error sending data: {str(e)}")
                                    except Exception as e:

QtWidgets.QMessageBox.critical(self, "Error", f"Unknown
error: {str(e)}")
                            else:

QtWidgets.QMessageBox.warning(self, "Serial Error","Serial
port is not open or disconnected.")
                    except Exception as e:
                            print(f"Error
send_settings_to_arduino: {e}")

QtWidgets.QMessageBox.critical(self, "Error", f"Data
processing error: {str(e)}")
        def reset_form(self):
                    try:
                            self.ACCELERATION.clear()
                            self.HOMESPEED.clear()
                            self.ROBOTSPEED.clear()
                    except Exception as e:
                            print(f"Error reset_form: {e}")
        def closeEvent(self, event):
                    try:
                            event.accept()
                    except Exception as e:
                            print(f"Error closeEvent: {e}")
                            event.accept()
        def keyPressEvent(self, event):
                    try:
                            # Ngăn Enter key đóng cửa sổ
                            if event.key() ==
QtCore.Qt.Key_Return or event.key() ==
QtCore.Qt.Key_Enter:

self.send_settings_to_arduino()
                                    event.accept()
                            else:

super().keyPressEvent(event)
                    except Exception as e:
                            print(f"Error keyPressEvent: {e}")
                            super().keyPressEvent(event)
class YOLOThread(QThread):
    new_frame = pyqtSignal(object)
```

```
    new_detection = pyqtSignal(object)
    def __init__(self, model_path="best.pt",
camera_index=0):
        super().__init__()
        self.model = YOLO(model_path)
        self.cap = cv2.VideoCapture(camera_index)
        self.running = False
        self.serial = None
        self.last_sent = None
        self.last_time = 0
        self.send_interval = 5.0
        self.count = 0
    def set_serial(self, serial_obj):
        self.serial = serial_obj
    def run(self):
        self.running = True
        # --- Gửi số 0 một lần sau 3 giây nếu Serial tồn
tại ---
        if self.serial:
            time.sleep(3)
            self.serial.write(b"0\n")
            print("Đã gửi số 0 lúc bắt đầu")
        class_to_value = {
            "Correct": 2,
            "Missing": 3,
            "WrongPin": 4
        }
        state = "IDLE"
        state_start_time = 0
        current_class = None
        while self.running:
            ret, frame = self.cap.read()
            if not ret:
                continue
            results = self.model.predict(source=frame,
stream=False, verbose=False)[0]
            current_time = time.time()
            detected = False
            for box in results.boxes:
                cls_id = int(box.cls[0])
                cls_name = results.names[cls_id]
                conf = float(box.conf[0])
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                cx = int((x1 + x2) / 2)
                cy = int((y1 + y2) / 2)
                if conf > 0.8 and cls_name in
class_to_value:
                    value = class_to_value[cls_name]
```

```
                    detected = True
                    # Vẽ khung
                    label = f"{cls_name} {conf:.2f}"
                    cv2.rectangle(frame, (x1, y1), (x2,
y2), (0, 255, 0), 2)
                    cv2.putText(frame, label, (x1, y1 -
10),
                                cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 255, 0), 2)
                    if state == "IDLE":
                        current_class = cls_name
                        state = "WAITING_1S"
                        state_start_time = current_time
                        print(f"Phát hiện {cls_name}, chờ
1s...")
                    elif state == "WAITING_1S":
                        # Thời gian chờ tùy loại vật
                        if current_class == "Correct":
                            wait_time = 2.7
                        else:
                            wait_time = 2.0
                        if (current_time -
state_start_time) >= wait_time:
                            if self.serial:

self.serial.write(f"{value}\n".encode())
                                print(f"Đã gửi: {value}
(class: {current_class})")
                            self.count += 1

self.new_detection.emit((self.count, current_class, conf,
(cx, cy)))
                            state = "WAITING_4S"
                            state_start_time =
current_time

                    elif state == "WAITING_4S":
                        if (current_time -
state_start_time) >= 4.0:
                            state = "IDLE"
                            print("Đã xong 4s, sẵn sàng
nhận vật tiếp theo.")

                    break  # Chỉ xử lý 1 vật mỗi frame

            # Nếu không phát hiện vật → reset nếu không
đang chờ xử lý
            if not detected and state != "WAITING_4S":
```

```
                state = "IDLE"
                current_class = None

            self.new_frame.emit(frame)

        self.cap.release()
    def stop(self):
        self.running = False
        self.quit()
        self.wait()
class MainWindow(QtWidgets.QMainWindow):
        def __init__(self):
                super().__init__()
                try:
                        uic.loadUi("giaodienDATN.ui",
self)
                        header =
self.tableWidget.horizontalHeader()

header.setSectionResizeMode(QtWidgets.QHeaderView.Stretch)
                        self.login_window = None
                        # Khởi tạo Serial
                        self.arduino = None
                        # Kết nối nút

self.SETTING.clicked.connect(self.open_setting)

self.ConnectSerial.clicked.connect(self.connect_serial)

self.Start.clicked.connect(self.start_detection)

self.GOHOME.clicked.connect(self.go_home)

self.EXIT.clicked.connect(self.return_to_login)

self.SELECT_CAMERA.clicked.connect(self.select_camera)

                        # Khởi tạo YOLO thread - sửa lỗi
khởi tạo
                        self.yolo_thread = None
                        self.autosave_timer = QTimer()

self.autosave_timer.timeout.connect(self.auto_save_table)
                        self.autosave_timer.start(5000)  #
10000 ms = 10 giây (bạn có thể chỉnh)

                except Exception as e:
                        print(f"Error initializing
```

```
MainWindow: {e}")

QtWidgets.QMessageBox.critical(None, "Error", f"Unable to
initialize MainWindow: {e}")
        def select_camera(self):
                try:
                        selected_text =
self.CAMERA.currentText()
                        # Gán chỉ số tương ứng
                        if selected_text == "CAMERA":
                                camera_index = 0
                        elif selected_text == "DROID CAM":
                                camera_index = 1
                        elif selected_text == "WEBCAM":
                                camera_index = 2
                        else:
                                camera_index = 0  # mặc
định
                        # Dừng thread cũ nếu đang chạy
                        if self.yolo_thread and
self.yolo_thread.isRunning():
                                self.yolo_thread.stop()
                                self.yolo_thread.wait()
                        # Tạo luồng YOLO mới với
camera_index đã chọn
                        self.yolo_thread =
YOLOThread(model_path="best.pt",
camera_index=camera_index)

self.yolo_thread.new_frame.connect(self.update_webcam)

self.yolo_thread.new_detection.connect(self.update_table)
                        self.yolo_thread.start()
                except Exception as e:
                        print(f"select_camera error: {e}")

QtWidgets.QMessageBox.warning(self, "Error", f"Unable to
select camera: {e}")
        def return_to_login(self):
                try:
                        # Dừng YOLO thread
                        if self.yolo_thread and
self.yolo_thread.isRunning():
                                self.yolo_thread.stop()
                                self.yolo_thread.wait()
                        # Đóng kết nối serial
                        if self.arduino and
self.arduino.is_open:
```

```
                                self.arduino.close()
                        if self.login_window:
                                self.hide()

self.login_window.reset_fields()
                                self.login_window.show()
                except Exception as e:
                        print(f"Error return_to_login:
{e}")
        def connect_serial(self):
                try:
                        port =
self.SerialPort.currentText()
                        if self.arduino and
self.arduino.is_open:
                                self.arduino.close()
                        self.arduino = serial.Serial(port,
9600, timeout=1)
                        time.sleep(2)
                        if self.yolo_thread:

self.yolo_thread.set_serial(self.arduino)
                        print(f"Connected {port}")

QtWidgets.QMessageBox.information(self, "Success",
f"Connected {port}")
                except Exception as e:
                        print(f"COM connection failed:
{e}")

QtWidgets.QMessageBox.warning(self, "Error", f"COM
connection failed: {e}")
        def start_detection(self):
                try:
                        if self.arduino and
self.arduino.is_open:
                                self.arduino.write(b"0\n")
                        if self.yolo_thread and not
self.yolo_thread.isRunning():
                                self.yolo_thread.start()
                        elif not self.yolo_thread:

QtWidgets.QMessageBox.warning(self, "Warning", "Please
select camera first!")
                except Exception as e:
                        print(f"Error start_detection:
{e}")
```

```
QtWidgets.QMessageBox.warning(self, "Error", f"Unable to
start detection: {e}")
        def go_home(self):
                try:
                        if self.arduino and
self.arduino.is_open:
                                self.arduino.write(b"1\n")
                except Exception as e:
                        print(f"Error go_home: {e}")
        def update_webcam(self, frame):
                try:
                        rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
                        h, w, ch = rgb.shape
                        bytes_per_line = ch * w
                        qt_img = QImage(rgb.data, w, h,
bytes_per_line, QImage.Format_RGB888)

self.ui.Webcam.setPixmap(QPixmap.fromImage(qt_img).scaled(
                                self.ui.Webcam.width(),
self.ui.Webcam.height(), QtCore.Qt.KeepAspectRatio))
                except Exception as e:
                        print(f"Error update_webcam: {e}")
        def update_table(self, data):
                try:
                        count, cls_name, conf, (cx, cy) =
data
                        row = self.tableWidget.rowCount()
                        self.tableWidget.insertRow(row)
                        self.tableWidget.setItem(row, 0,
QtWidgets.QTableWidgetItem(str(count)))
                        self.tableWidget.setItem(row, 1,
QtWidgets.QTableWidgetItem(cls_name))
                        self.tableWidget.setItem(row, 2,
QtWidgets.QTableWidgetItem(f"{conf * 100:.2f}%"))
                        self.tableWidget.setItem(row, 3,
QtWidgets.QTableWidgetItem(f"({cx}, {cy})"))
                except Exception as e:
                        print(f"Error update_table: {e}")
        def open_setting(self):
                try:
                        # Kiểm tra đã kết nối Serial chưa
(bảo vệ an toàn)
                        if self.arduino and
self.arduino.is_open:
                                self.setting_window =
SettingWindow(serial_port=self.arduino)
                                self.setting_window.show()
```

```
self.setting_window.raise_()

self.setting_window.activateWindow()
                        else:

QtWidgets.QMessageBox.warning(self, "Error", "Arduino not
connected (Serial not open).")
                except Exception as e:
                        print(f"Error open_setting: {e}")
        def closeEvent(self, event):
                try:
                        if self.yolo_thread and
self.yolo_thread.isRunning():
                                self.yolo_thread.stop()
                                self.yolo_thread.wait()
                        if self.arduino and
self.arduino.is_open:
                                self.arduino.close()
                        event.accept()
                except Exception as e:
                        print(f"Error closeEvent: {e}")
                        event.accept()
        def auto_save_table(self):
            try:
                save_table_to_excel(self.tableWidget,
filename="DATN.xlsx")
            except Exception as e:
                print(f"Error auto_save_table: {e}")
class LoginWindow(QtWidgets.QWidget):
        def __init__(self):
                super().__init__()
                try:
                        uic.loadUi("LOGIN.ui", self)
                        self.main_window = None
                        # Gán sự kiện nút

self.LOGIN_2.clicked.connect(self.handle_login)

self.CANCEL.clicked.connect(QtWidgets.qApp.quit)

self.USER.returnPressed.connect(self.handle_login)

self.PASSWORD.returnPressed.connect(self.handle_login)
                except Exception as e:
                        print(f"Error initializing
LoginWindow: {e}")
```

```
QtWidgets.QMessageBox.critical(None, "Error", f"Unable to
initialize LoginWindow: {e}")
        def reset_fields(self):
                try:
                        self.USER.clear()
                        self.PASSWORD.clear()
                except Exception as e:
                        print(f"Reset_fields error: {e}")
        def handle_login(self):
                try:
                        username = self.USER.text()
                        password = self.PASSWORD.text()
                        if username == "admin" and
password == "123":
                                self.main_window =
MainWindow()

self.main_window.login_window = self
                                self.main_window.show()
                                self.hide()
                                print("Login successful!")
                        else:

QtWidgets.QMessageBox.warning(self, "Login Failed", "Wrong
USER or PASSWORD.")
                except Exception as e:
                        print(f"Wrong handle_login: {e}")

QtWidgets.QMessageBox.critical(self, "Error", f"Login
error: {e}")
def save_table_to_excel(table, filename="DATN.xlsx"):
    from openpyxl import Workbook
    wb = Workbook()
    ws = wb.active
    # Ghi tiêu đề cột
    for col in range(table.columnCount()):
        header_item = table.horizontalHeaderItem(col)
        if header_item:
            ws.cell(row=1, column=col + 1,
value=header_item.text())
    # Ghi từng hàng
    for row in range(table.rowCount()):
        for col in range(table.columnCount()):
            item = table.item(row, col)
            if item:
                ws.cell(row=row + 2, column=col + 1,
value=item.text())
    wb.save(filename)
```

```
    print(f"Saved table to file: {filename}")
if __name__ == "__main__":
        try:
                app = QtWidgets.QApplication(sys.argv)
                login = LoginWindow()
                login.show()
                sys.exit(app.exec_())
        except Exception as e:
                print(f"Lỗi chạy ứng dụng: {e}")
```