Design and Development of an Educational SCARA Platform "EduSCARA", with E-waste Sorting Example Application

Alex Clark

Student number: 219325190

BEng Robotics Engineering with Placement

BEng Final Project Report

Academic year: 2024/25

Supervisor: Dr. Tareq Assaf

Assessor: Dr. Fang Duan

Number of words in Abstract: 248

Number of words in Main body: 11787

Ethical approval for the project

I certify that I have received ethical approval for this project, and have read and understood the Final year project handbook for the Department of Electronic and Electrical Engineering.

Abstract

This report presents the development of **EduSCARA**, an affordable, industrially relevant Selective Compliance Assembly Robot Arm (SCARA) development platform. Designed to lower the barrier to entry in industrial robotics, the EduSCARA platform offers a complete, hands-on development environment that serves as an accessible stepping stone by closely replicating the architecture and challenges of industrial systems without the prohibitive cost or complexity. The EduSCARA platform is composed of four key subsystems:

- A compact, 3D-printed SCARA manipulator that mirrors industrial SCARA mechanics, designed to be low-cost, easily replaceable and highly customisable, enabling students to explore kinematics and structural dynamics, as well as experiment with hardware modifications.
- An open-source STM32-based SCARA motion controller providing a transparent, flexible platform for investigating low-level control and embedded programming.
- A Python API for high-level application development that enables seamless integration with computer vision, AI frameworks and custom user interfaces, while using a language familiar to most robotics engineering students.
- A Unity-based simulation environment accelerating development and debugging by allowing students to prototype and test motion sequences virtually before deploying them to the physical robot, helping to bridge the simulation-reality divide.

In contrast to traditional industrial robots, often costing tens of thousands of pounds, EduSCARA empowers learners to experiment, debug, and innovate for less than £160. Its modular and transparent design prioritises realism, offering students a hands-on experience that mirrors professional workflows. More than just a small robot, EduSCARA serves as a gateway into industrial robotics, making high-impact, professional-grade learning truly accessible.

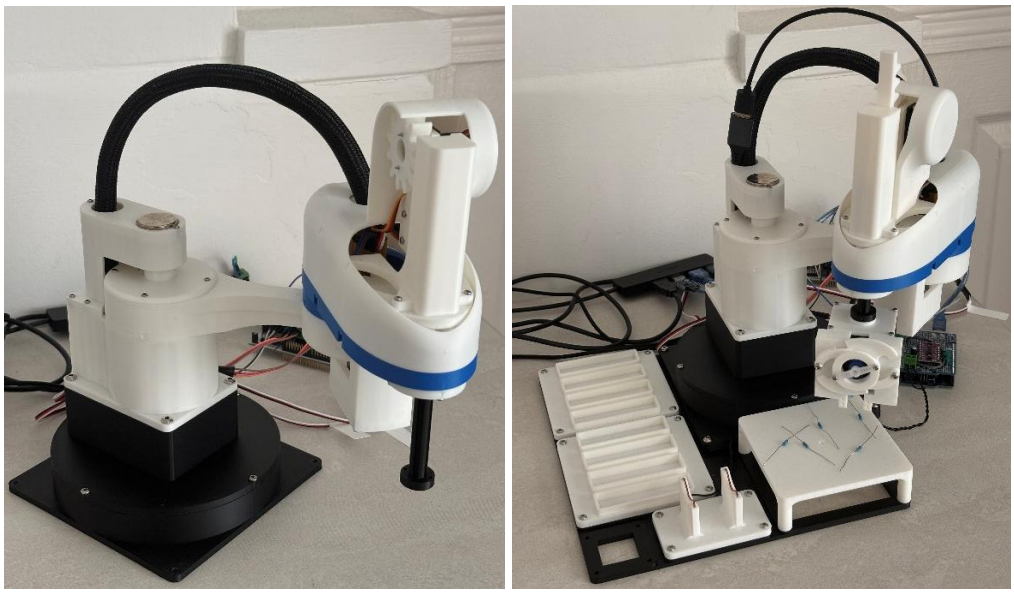


Figure 1: EduSCARA Manipulator (left) and E-waste Sorting Example Application (right)

Table of Contents

Table of Acronyms

Acronym	Definition
ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machine
AXIS	A degree of freedom that a robot can move or rotate around
CAD	Computer-Aided Design
CAN	Controller Area Network
CPU	Central Processing Unit
DC	Direct Current
GPIO	General-Purpose Input/Output
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
PC	Personal Computer
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative (a control loop feedback mechanism)
PLA	Polylactic Acid (a biodegradable thermoplastic used in 3D printing)
PLC	Programmable Logic Controller
PWM	Pulse Width Modulation
QTY	Quantity
ROS	Robot Operating System
RPS	Robot Programming System
RTOS	Real-Time Operating System
SCARA	Selective Compliance Articulated Robot Arm
SRAM	Static Random Access Memory
UART2	Universal Asynchronous Receiver-Transmitter (2^{nd} channel)
USB	Universal Serial Bus
YOLO	‘You Only Look Once’ (a real-time object detection algorithm)

1 Introduction and context

1.1 Motivation and Problem Statement

Practical robotics education bridges the gap between theoretical knowledge and real-world application, effectively enhancing student motivation and deepening their understanding of autonomous systems [1]. It exposes students to the non-idealities, system limitations and debugging challenges that are fundamental to developing successful robotic systems.

Despite the growing importance of robotics education, available teaching methods present significant barriers. Industrial robots are prohibitively expensive [2] and are not suitable for beginners, and while educational robots [3] [4] exist, they often lack the functionality and development workflows found in real industrial systems despite carrying a high cost. Furthermore, simulation-only approaches fall short in accurately conveying the challenges of robotics application development in the real-world.

This project proposes an educational SCARA (Selective Compliance Assembly Robot Arm) development platform, **EduSCARA**, designed to be both affordable and industrially relevant. EduSCARA provides a realistic and accessible environment for students to experiment, prototype, and develop real-world applications using a platform that closely mirrors industrial systems. This initiative aligns with the framework for robotic roles in education proposed by Xu and Ouyang [5], specifically targeting '*Learning through robotics*' as illustrated in their conceptual framework:

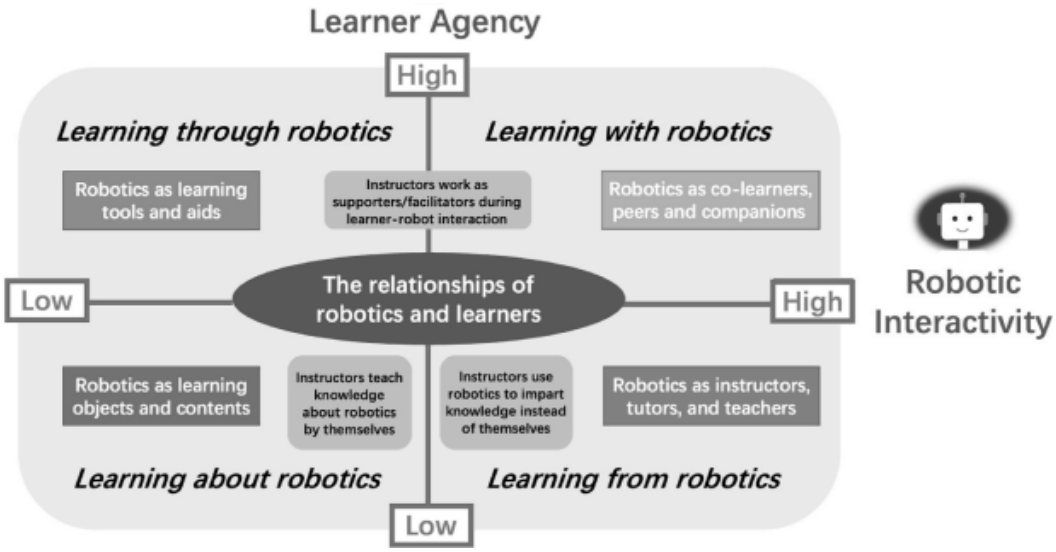


Figure 2: A conceptual framework of the learner-robot relationships [5]

This approach positions educational robotics as tools or assistants, enabling students to actively engage in the educational process by utilising robots to achieve specific objectives. The platform aims to provide a low-risk, hands-on approach to learning, making it ideal for educational settings, allowing students to gain meaningful experience and enhance key learning outcomes, including problem-solving, computational thinking, and self-efficacy [6], without the high costs and risks associated with industrial robotics, and making the transition into real-world robotics engineering roles less daunting and more accessible.

1.2 Project Objectives

The primary objective of this project is to develop a low-cost SCARA application development platform that enables students to practice and experiment with robotics application development. A SCARA configuration was chosen because of its industrial relevance – widely used in pick-and-place, assembly, inspection and packaging applications [7] – and offers a good balance between mechanical simplicity, speed, precision and cost [8]. To maximise its educational value, the SCARA must closely replicate the form factor and functionality of a real industrial system.

Figure 3: Yaskawa industrial SCARA [9]

The forward and inverse kinematics are straightforward, the system can be made compact, and the positional errors are primarily concentrated in two joints, allowing for use of lower-cost actuators without severely impacting usability.

The platform includes:

- Table-top sized closed-loop SCARA Manipulator using 3D printable components.
- SCARA Motion Controller with open-source firmware written in bare-metal C.
- Python API to communicate with the SCARA Motion Controller.
- Simulation environment in Unity to support the complete development workflow.

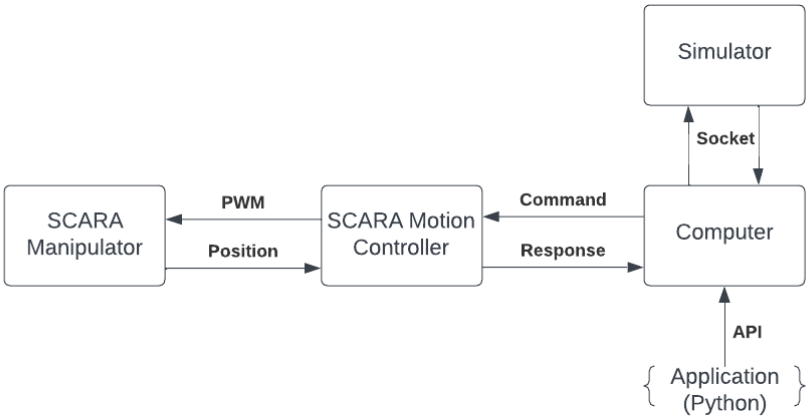


Figure 4: High-level block diagram of SCARA development platform for education

The key aim is to allow students to experience real-world challenges associated with robotics, such as mechanical backlash, sensor noise, delay, joint friction, control system tuning, sensor

calibration and real-world electrical and electronic engineering, which are often unrealistic or absent in purely simulated environments such as CoppeliaSim and Gazebo [10].

1.3 Existing Solutions and Their Limitations

Available options for practical robot education are industrial robot systems and educational robot platforms. Industrial SCARA robots from manufacturers such as ABB and KUKA offer the highest levels of fidelity and realism. However, they come at a significant cost, typically ranging from $25000 to $100000 per unit. Additional expenses for testing, installation, and integration can add a further 10–30% to the total system cost [11]. Moreover, their use requires specialised training, dedicated safety systems, and appropriate infrastructure.

Figure 5:KUKA KR SCARA [12]

This makes them impractical for hands-on education at scale, particularly for early-stage learners who face a higher risk of injury or damaging equipment. Moreover, the high stakes discourage experimentation and iterative, trial-and-error learning, which are essential for developing practical robotics skills.

A well-known example of a SCARA that can be classified as educational is the example from “How To Mechatronics” [13], with many other examples following a similar design approach:

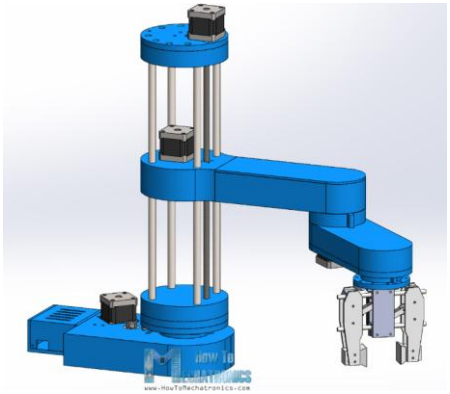


Figure 6: How To Mechatronics Arduino based SCARA [13]

This design does not meet the project criteria, as it does not closely replicate the form factor and functionality of a real industrial SCARA – most notably due to its structural integrity and the lack of real-time position feedback on critical joints. Among the reviewed literature, the closest example to meeting the manipulator criteria is the "ROBCO" SCARA [14] educational robot; however, it also operates as a fundamentally open-loop system. As a result, students are unable to engage with critical aspects of industrial robotics such as control system tuning.

Figure 7: EDUCATIONAL ROBOT - "ROBCO" SCARA [14]

These criticisms are to the manipulator designs in isolation, whereas the proposed platform aims to provide not only the manipulator but also the supporting systems necessary to make robotics application development realistic and educational. The development platform as a whole, rather than just the robotic arm, serves as the complete solution.

1.4 Novelty and Key Contributions

This platform is designed to implement a closed-loop SCARA system that closely mirrors industrial architectures. Design choices during the development of this system were made to satisfy this criteria. The tables below, presents a comparison of key design choices between the proposed subsystems and their industrial-grade equivalents (typical specifications obtained from the Trio Motion Technology product range [15]).

SCARA Manipulator	
EduSCARA Platform feature	**Industrial-grade equivalent feature (typical)**
DC hobbyist grade servo motors as actuators	Industrial AC servo motors as actuators + AC drives
Potentiometers as position feedback sensors	Absolute encoders as position feedback sensor
Rack and pinion mechanism for Z Axis	Ball screw mechanism for Z Axis
3D printed PLA links and shell	Metal alloy or composite links / plastic shell
Maximum reach 225mm	Maximum reach >=400mm

Table 1: EduSCARA vs Industrial Equivalent : SCARA Manipulator

Figure 8: EduSCARA Manipulator (left) and Trio SCARA (right)

The manipulator design replicates the core mechanical structure of an industrial SCARA using accessible components, such as low-cost DC servos, simple sensors and 3D printable components. While the performance is clearly reduced compared to industrial arms, the kinematics and control structure remain representative. This makes EduSCARA an ideal testbed for learning control theory, motion planning, and real-time feedback systems in a realistic but accessible format.

Robot Motion Controller	
EduSCARA Platform feature	**Industrial-grade equivalent feature (typical)**
STM32 development board with custom shield	Bespoke PCB
Open source firmware (bare-metal)	Closed source firmware (bare-metal/RTOS)
PWM signal direct to servo	EtherCAT/PROFIBUS/CAN/… etc. to drives
Not programmable (feature to be developed)	Programmable

Table 2: EduSCARA vs Industrial Equivalent : Robot Motion Controller

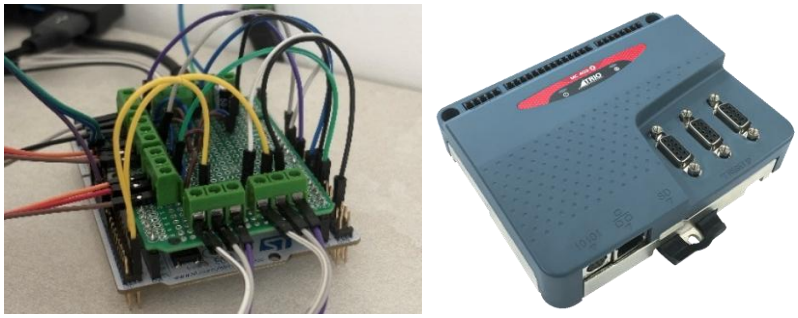


Figure 9: EduSCARA Motion Controller (left) and Trio MC403-X Motion Controller (right)

Unlike industrial motion controllers that are often proprietary, EduSCARA uses an STM32 microcontroller with open source firmware. This enables full access to the *SCARA Motion Coordinator Stack*, supporting educational engagement with embedded programming and motion control. The modular design of the hardware and firmware simplifies future enhancements to the subsystems.

Robot Motion Controller API	
EduSCARA Platform feature	**Industrial-grade equivalent feature (typical)**
Python API (more languages to be developed)	Common languages: Python, C++, .NET, etc. API

Table 3: EduSCARA vs Industrial Equivalent : Robot Motion Controller API

The Python-based API simplifies integration with external software such as computer vision systems and machine learning frameworks. This choice lowers the barrier to experimentation and development, as Python is often the first language learned by young engineers, while still maintaining industrial relevance.

Robot Simulator	
EduSCARA Platform feature	**Industrial-grade equivalent feature (typical)**
Unity based – written in C# – open source	Bespoke software – closed source

Table 4: EduSCARA vs Industrial Equivalent : Robot Simulator

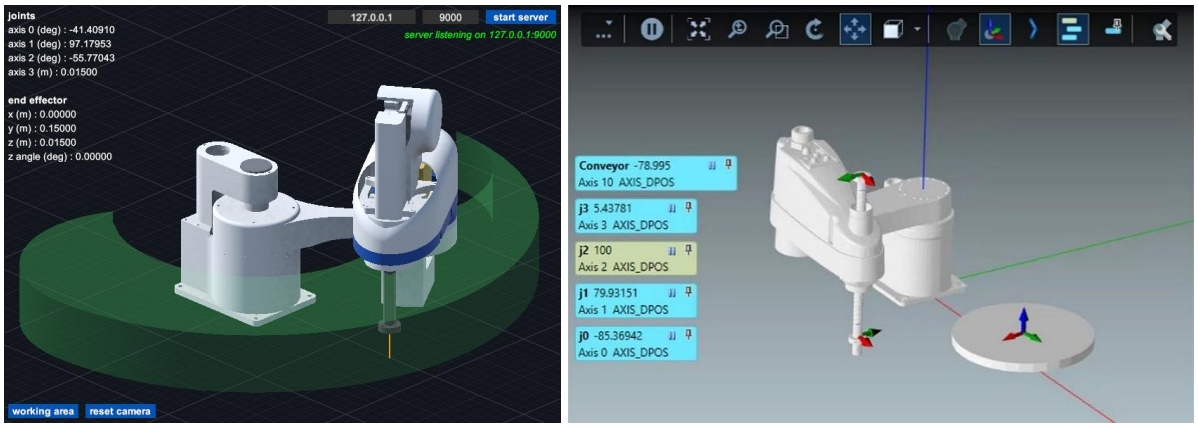


Figure 10: EduSCARA Simulator (left) and Trio Simulator (right)

The simulation environment provides an immersive platform for prototyping and testing movement commands and sequences, allowing users to avoid deploying applications blindly onto the robot. This also helps students to understand the differences between simulation and reality by highlighting the effects of real-world non-idealities. The simulator is made using the Unity engine, allowing students to import their own 3D models of application-specific components and attaching custom end effectors. Additionally, it visualises the robot's working area and allows users to interact with and explore the machine in a virtual space.

1.5 EduSCARA E-waste Sorting Application Exercise for Teaching

This example application presents an autonomous resistor-sorting application built using the EduSCARA platform, designed not only to solve a practical automation task but also to serve as a comprehensive exercise for students.

It provides students with full-stack robotics development [16] experience, covering mechanical design, simulation, firmware, software and system integration to solve a real-world problem. By developing this application, learners gain practical experience in CAD and iterative prototyping, firmware development, Python API design and machine learning model training using real datasets. The use of simulation for motion planning enables safe experimentation before deployment, while the comparison between simulated and real-world performance teaches students to account for mechanical tolerances and non-idealities.

This exercise highlights the educational value of integrating hardware, software and AI in a real-world context, making abstract concepts more tangible and equipping students with the essential skills required in modern robotics engineering.

1.6 Report Structure

This report begins by introducing the motivation behind the project and outlining its objectives, with a particular focus on its role in practical robotics education. It then reviews key technical concepts such as motion profiling, backlash reduction, and object detection that inform design decisions throughout development.

The core of the report presents the iterative design and implementation of the EduSCARA platform, covering both hardware and software development of its subsystems, as well as a walk-through of the E-waste Sorting Application Exercise. The design tools used throughout the methods and results section are all free to use:

- Autodesk Fusion for CAD modelling
- STM32Cube IDE and Arduino IDE for embedded systems programming in C
- Visual Studio Code for API and application development in Python
- Unity Editor to develop the simulator, and Visual Studio for C# emulator development

Finally, the project’s results are critically evaluated, with limitations acknowledged and clear directions proposed for future improvement. These sections provide a complete view of the platform’s development and its potential impact in educational settings.

2 Relevant literature and past work

2.1 Double-S velocity profile

Velocity profiling in the context of a point-to-point move is the process of controlling the velocity of robotic joints over time to ensure smooth and efficient motion. Rather than abruptly starting and stopping at target positions, velocity profiling gradually changes the motion [17], achieving faster settling times for accurate positioning while reducing mechanical stress and vibrations.

A Trapezoidal velocity profile is one of the most commonly used motion profiles, consisting of three distinct phases: constant acceleration, constant velocity, and constant deceleration [18]. However, this profile exhibits instantaneous changes in acceleration, resulting in infinite jerk at phase transitions. This leads to increased mechanical stress and undesired vibration at these transition points [19].

To address these limitations, the Double-S velocity profile provides smooth transitions between velocity phases by gradually changing acceleration. This results in a continuous, linear-piecewise acceleration profile characterised by seven velocity phases with constant jerk [19]. A graphical comparison between the Double-S and Trapezoidal velocity profiles is shown in Figure 11.

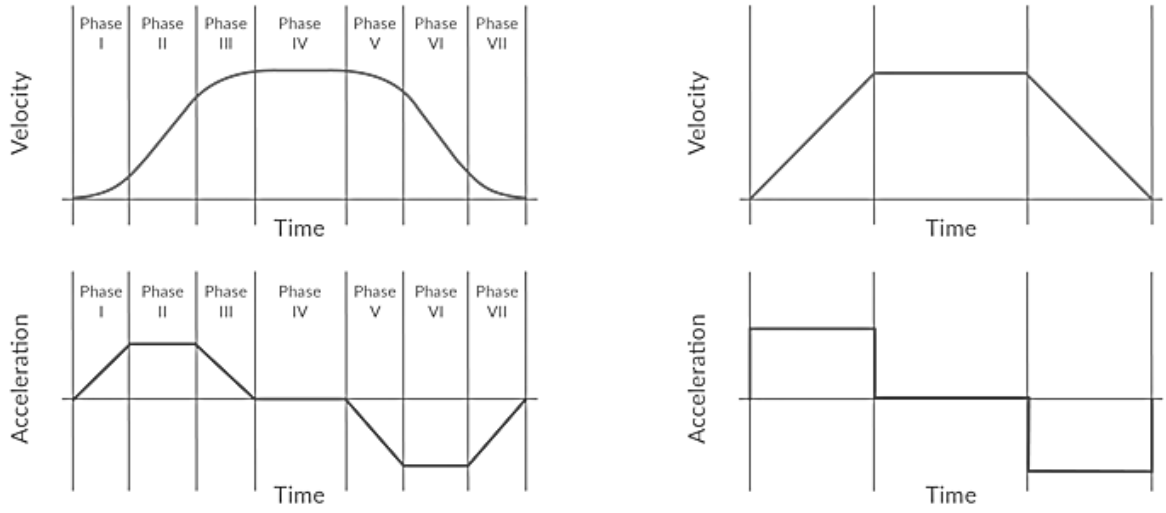


Figure 11: Double-S and Trapezoidal velocity profile [18]

By distributing jerk over time, the Double-S velocity profile effectively minimises mechanical stress, reduces vibration, and improves overall motion stability compared to the Trapezoidal velocity profile, making it a more suitable option for robotic applications.

2.1.1 Symmetric Double-S trajectory with assigned duration of different phases

A desired characteristic for the robot used in this project is high-speed operation. The most efficient way for the end-effector to reach a desired position is for all the robotic joints to reach their respective target positions simultaneously. This approach is commonly expressed as “MoveJ” or “MoveJoint” in industrial robot programming tools such as ABB RobotStudio [20].

A general approach for planning a Double-S trajectory with a given time length T and displacement h, is to specify the durations of the acceleration and deceleration phases T_a and T_d, and constant jerk phases T_j. This consists of defining the values for maximum velocity v_{max}, acceleration a_{max}, and jerk j_{max} reached as a function of the desired T, T_a, T_d and T_j.

In particular, for a symmetric case where $v_{min} = -v_{max}$, $a_{min} = -a_{max}$ and $j_{min} = -j_{max}$, where the initial and final velocities are both assumed to be zero (therefore $T_d = T_a$), and where v_{max} and a_{max} are reached…

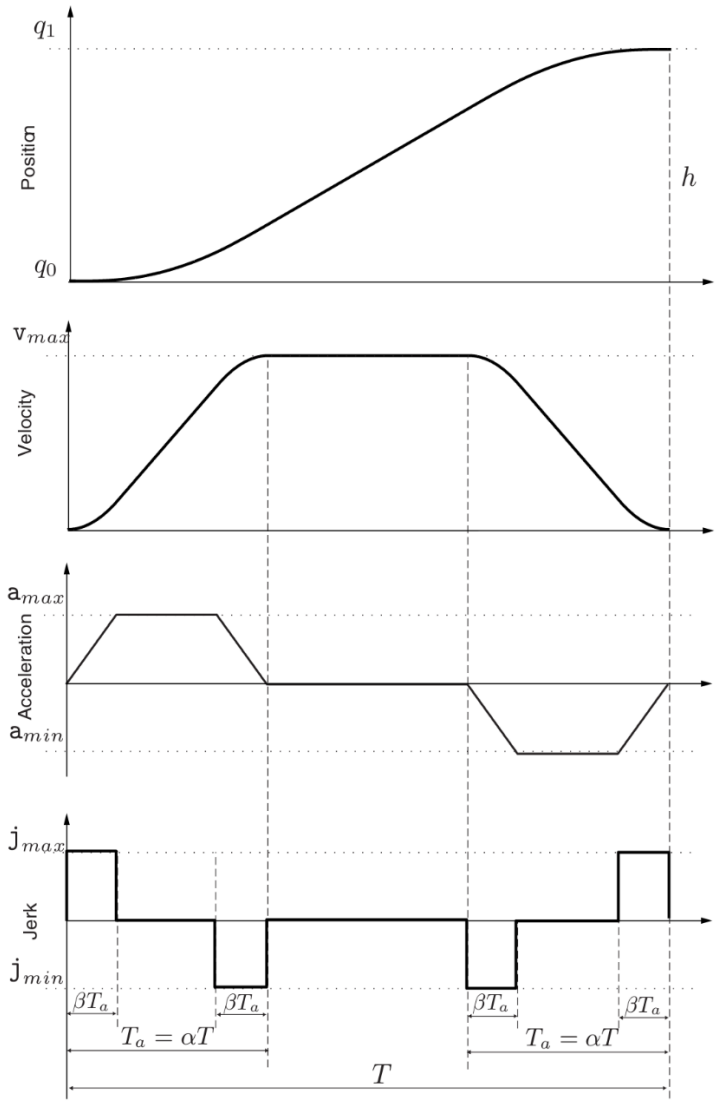


Figure 12: Symmetric Double-S displacement, velocity, acceleration and jerk profiles [19]

… the expressions for the total duration, acceleration phase durations and jerk segment durations are:

$$T = \frac{h}{v_{max}} + T_a \tag{2.1}$$

$$T_a = \frac{v_{max}}{a_{max}} + T_j \tag{2.2}$$

$$T_j = \frac{a_{max}}{j_{max}} \tag{2.3}$$

From this, it is possible to deduce the corresponding values of v_{max}, a_{max} and j_{max}:

$$v_{max} = \frac{h}{T - T_a} \tag{2.4}$$

$$a_{max} = \frac{h}{(T - T_a)(T_a - T_j)} \tag{2.5}$$

$$j_{max} = \frac{h}{(T - T_a)(T_a - T_j)T_j} \tag{2.6}$$

Assuming the acceleration period is a fraction of the entire trajectory duration:

$$T_a = \alpha T, \qquad 0 < \alpha \leq 1/2 \tag{2.7}$$

and similarly, the time length of the constant jerk phase is a fraction of the acceleration period:

$$T_j = \beta T_a, \qquad 0 < \beta \leq 1/2 \tag{2.8}$$

then the values for the maximum speed, acceleration, and jerk of the double S trajectory $q(t)$ are obtained as:

$$v_{max} = \frac{h}{(1 - \alpha)T} \tag{2.9}$$

$$a_{max} = \frac{h}{\alpha(1 - \alpha)(1 - \beta)T^2} \tag{2.10}$$

$$j_{max} = \frac{h}{\alpha^2\beta(1 - \alpha)(1 - \beta)T^3} \tag{2.11}$$

2.1.2 Computation of a Double-S trajectory

Once the time lengths T_a, T_j and values for v_{max}, a_{max} and j_{max} are defined, the Double-S trajectory can be computed using the following equations for the phases defined in Figure 13:

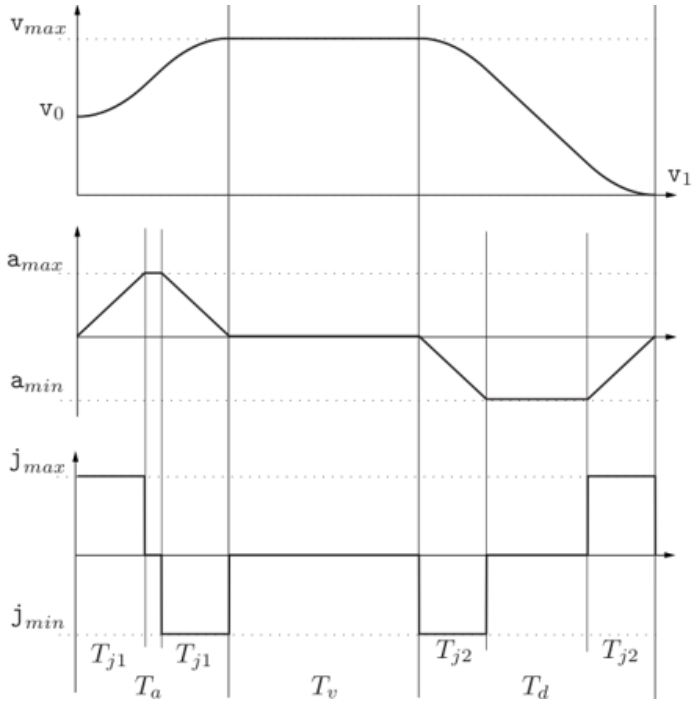


Figure 13: Seven phases of a Double-S velocity profile [19]

Acceleration phases

1) $t\epsilon[0, T_{j1}]$ **(for values of t between 0 and T_{j1})*

$$q(t) = q_0 + v_0 t + j_{max}\frac{t^3}{6} \tag{2.12}$$
$$q'(t) = v_0 + j_{max}\frac{t^2}{2}$$
$$q''(t) = j_{max}t$$
$$q'''(t) = j_{max}$$

2) $t\epsilon[T_{j1}, T_a - T_{j1}]$

$$q(t) = q_0 + v_0 t + \frac{a_{max}}{6}\left(3t^2 - 3T_{j1}t + T_{j1}^2\right) \tag{2.13}$$
$$q'(t) = v_0 + a_{max}(t - \frac{T_{j1}}{2})$$
$$q''(t) = j_{max}T_{j1} = a_{max}$$
$$q'''(t) = 0$$

3) $t\epsilon[T_a - T_{j1}, T_a]$

$$q(t) = q_0 + (v_{max} + v_0)\frac{T_a}{2} - v_{max}(T_a - t) - j_{min}\frac{(T_a - t)^3}{6} \tag{2.14}$$
$$q'(t) = v_{max} + j_{min}\frac{(T_a - t)^2}{2}$$
$$q''(t) = -j_{min}(T_a - t)$$
$$q'''(t) = j_{min} = -j_{max}$$

Constant velocity phase

4) $t\epsilon[T_a, T - T_d]$

$$q(t) = q_0 + (v_{max} + v_0)\frac{T_a}{2} + v_{max}(t - T_a) \tag{2.15}$$
$$q'(t) = v_{max}$$
$$q''(t) = 0$$
$$q'''(t) = 0$$

Deceleration phases

5) $t\epsilon[T - T_d, T - T_d + T_{j2}]$

$$q(t) = q_1 - (v_{max} + v_1)\frac{T_d}{2} + v_{max}(t - T + T_d) - j_{max}\frac{(t - T + T_d)^3}{6} \tag{2.16}$$
$$q'(t) = v_{max} - j_{max}\frac{(t - T + T_d)^2}{2}$$
$$q''(t) = -j_{max}(t - T + T_d)$$
$$q'''(t) = j_{min} = -j_{max}$$

6) $t\epsilon[T - T_d + T_{j2}, T - T_{j2}]$

$$q(t) = q_1 - (v_{max} + v_1)\frac{T_d}{2} + v_{max}(t - T + T_d) + \frac{a_{min}}{6}\left(3(t - T + T_d)^2 - 3T_{j2}(t - T + T_d) + T_{j2}^2\right)$$
$$q'(t) = v_{max} + a_{min}\left(t - T + T_d - \frac{T_{j2}}{2}\right) \tag{2.17}$$
$$q''(t) = -j_{max}T_{j2} = a_{min}$$
$$q'''(t) = 0$$

7) $t\epsilon[T - T_{j2}, T]$

$$q(t) = q_1 - v_1(T - t) - j_{max}\frac{(T-t)^3}{6} \tag{2.18}$$

$$q'(t) = v_1 + j_{max}\frac{(T-t)^2}{2}$$
$$q''(t) = -j_{max}(T - t)$$
$$q'''(t) = j_{max}$$

2.2 Gear backlash in robotics applications

Backlash (also called slop, play, or free-play) refers to the amount of movement between mating gears or components, occurring when the driving members is not directly connected to the load [21]. Backlash most commonly occurs due to the distance between the teeth of engaging gears in the case of bidirectional movement in a system:

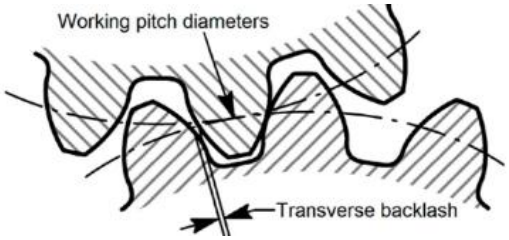


Figure 14: Gear backlash [22]

Minimising backlash is critical in robotic systems due to its impact on precision, efficiency, noise, vibrations, wear and overall motion control [23]. Manufacturing of gears is inherently imperfect, with small deviations occurring production that accumulate into larger deviations in a system with multiple gear stages. To ensure smooth operation, small clearances between the gears is necessary to prevent excessive friction, allow space for lubrication and accommodate thermal expansion [24].

Backlash is a contributing factor to 'lost motion', referring to the amount of movement lost due to the clearance of components, or the angular displacement in both directions where a load applied at the output results in a measured position that deviates from the desired position demanded by the input [25]:

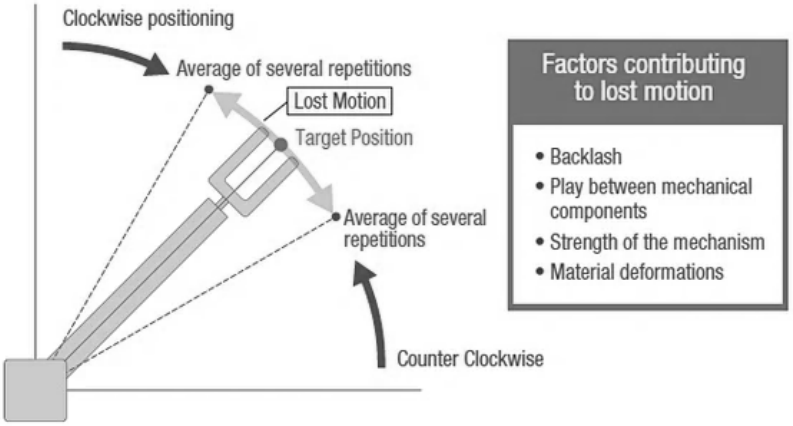


Figure 15: Lost motion and contributing factors [25]

2.2.1 Methods of reducing backlash

The effects of backlash and lost motion can be reduced through the use of specialised components such as harmonic gears, or design modifications such as preloading, which applies a force to the system to remove any clearance between components. Another approach involves incorporating position sensors within a control system to monitor motor position accuracy and apply necessary adjustments [21]. This project will explore preloading and position sensor methods to minimise lost motion in the robotic arm joints. Specialised components will not be considered due to their higher cost.

2.3 'You Only Look Once' (YOLO) real-time object detection algorithm

YOLO or You Only Look Once is an open-source real-time object detection algorithm that predicts object classes and bounding boxes in an image. The YOLO detection system resizes the input image, runs a single convolutional neural network on the image and thresholds the resulting detections by the model's confidence [26]:

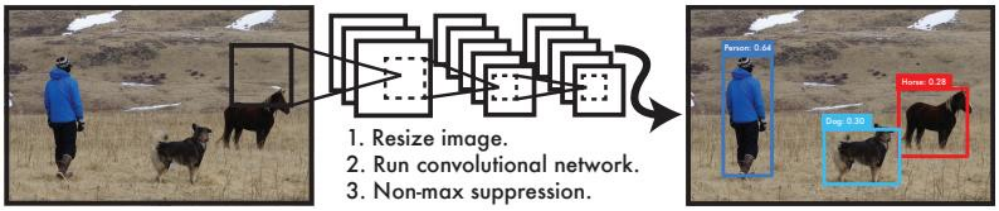


Figure 16: The YOLO Detection System [26]

The system models detection as a regression problem by dividing the image into an $S \times S$ grid and for each cell predicts B bounding boxes, the confidence for those boxes and C class probabilities. These predictions are encoded as a $S \times S \times (B * 5 + C)$ tensor [26]:

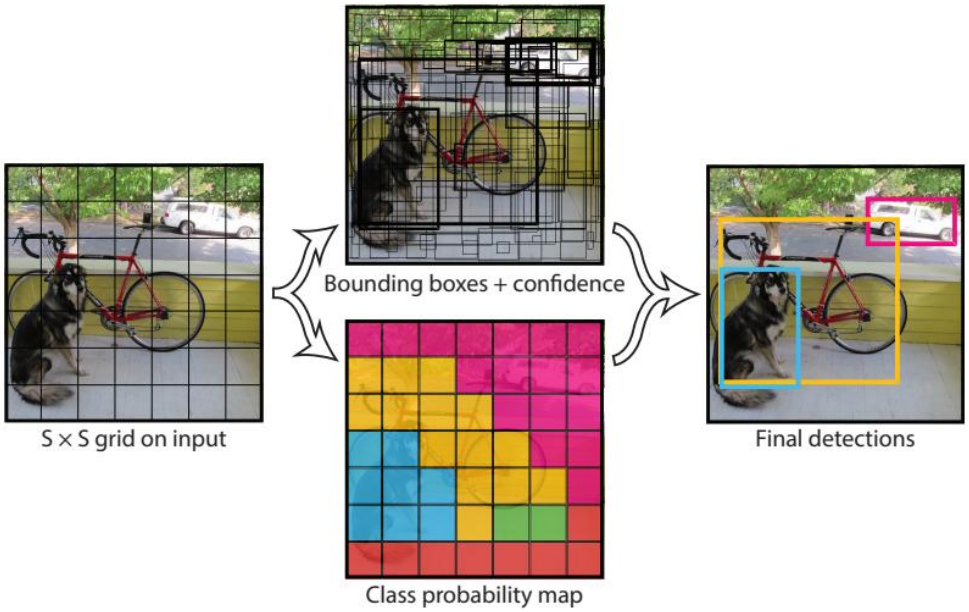


Figure 17: The YOLO Detection Model [26]

YOLO outperforms other object detection algorithms due to its speed, high detection accuracy, strong generalisation [27], and open-source nature, with YOLOv11 being the latest integration as of writing this report.

2.4 Existing Resistor Sorting Solutions

Before presenting the resistor sorting solution developed for the EduSCARA platform, which serves as an educational exercise for students and a potential aid in small R&D labs, it is important to establish originality of the problem.

A thorough literature review was conducted to determine whether similar solutions already exist and to assess whether this application offers a meaningful educational and technical challenge. Using Google Scholar, relevant prior work was identified through keywords such as '*autonomous'*, '*resistor'*, '*electrical components'*, '*robot'*, '*sorting'*, '*SCARA', etc*. The most applicable implementations found are summarised and critically analysed in the following sections, with attention given to their strengths, limitations, and relevance to this project.

2.4.1 Senior Design Project – Resistor Sorter – Trinity University (2013) [28]

The objective of this project was to develop a laboratory assistant tool capable of "straightening resistor leads, measuring their resistance, and sorting them based on similar values":

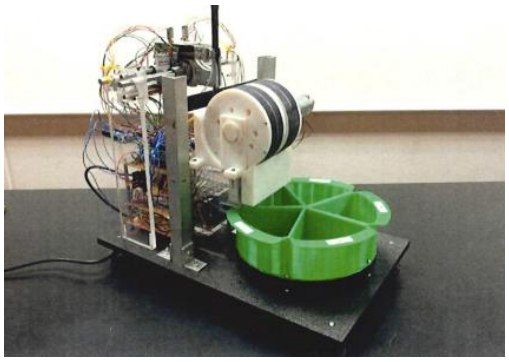


Figure 18: General view of integrated device - Resistor Sorter [28]

While the system was unable to autonomously sort the resistors as intended, a key strength of the design was the effective resistor lead straightening mechanism:

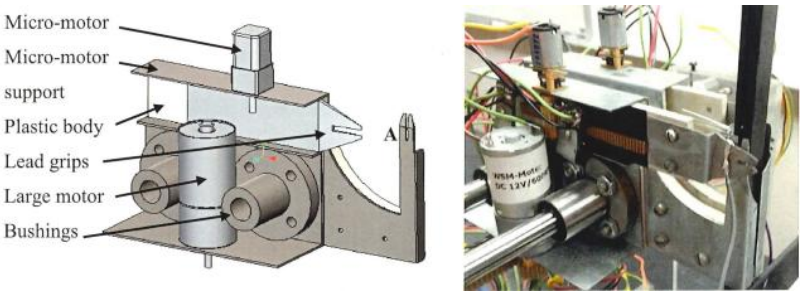


Figure 19: Primary Straightener module - Resistor Sorter [28]

Incorporating a similar mechanism into the EduSCARA resistor sorting application could enhance its functionality, provided it can be implemented cost-effectively.

2.4.2 Robotic Sorting of Mechanical and Electrical Parts: An Autonomous Vision-Based Approach in a Practical Case Study – University of Tehran (2024) [29]

This project uses an industrial delta robot to sort different electrical components into trays, using OpenCV and YOLOv3:

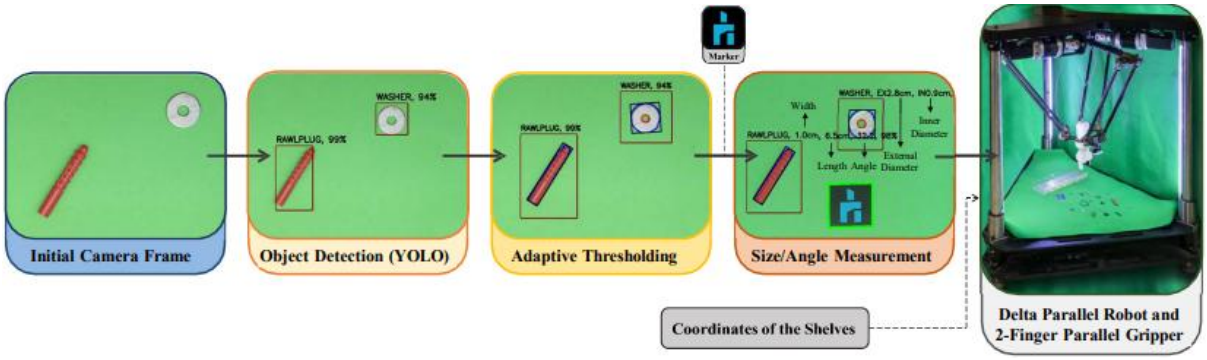


Figure 20: Overall steps in sorting parts into different shelves [29]

Based off of the reviewed literature, this project achieves the highest success rate in sorting electrical components. However, its large rig makes it unsuitable for lab environments where a compact, table-top solution is preferred. Additionally, components are sorted by type and not by specific value. The system is also very costly, utilising industrial-grade equipment such as PLCs, servos, and drives.

2.4.3 Summary and Insights on Resistor Sorting Solutions

While there are several other attempts to address similar objectives, such as the one cited in [30], the two examples discussed above show the most promising results. However, it is evident that the resistor sorting problem remains unsolved in an effective and affordable way. This makes it a suitable challenge for a small SCARA robot offered by the EduSCARA platform. By incorporating the following features, this project can address the limitations of previous implementations:

- Utilising the latest YOLO algorithm for reliable detection of resistor position and orientation.
- Designing a custom end effector with an integrated camera, eliminating the need for an external camera rig.
- Developing a digital ohmmeter to eliminate the reliance on colour band detection.
- Delivering a cost-effective solution.

3 Methods and results

3.1 The EduSCARA Motion Controller

The EduSCARA Motion Controller serves as a low-level interface for the EduSCARA manipulator. Its primary function is to act as a servo coordinator, receiving commands via serial and converting them into motion instructions to control servos.

The system is developed on a NUCLEO-F446RE development board featuring an STM32 microcontroller. The code is written in bare-metal C using the STM32 Hardware Abstraction Layer (HAL).

A Real-Time Operating System (RTOS) such as FreeRTOS or Mbed was intentionally not used in order to maintain simplicity and save memory, whilst maximising real-time control performance of the system. No third party libraries were used, in order to reduce potential sources of error and maintain full understanding and control over the system's operation.

3.1.1 The SCARA Motion Coordinator Stack

The SCARA Motion Coordinator stack diagram below represents the software architecture implemented on the SCARA motion controller:

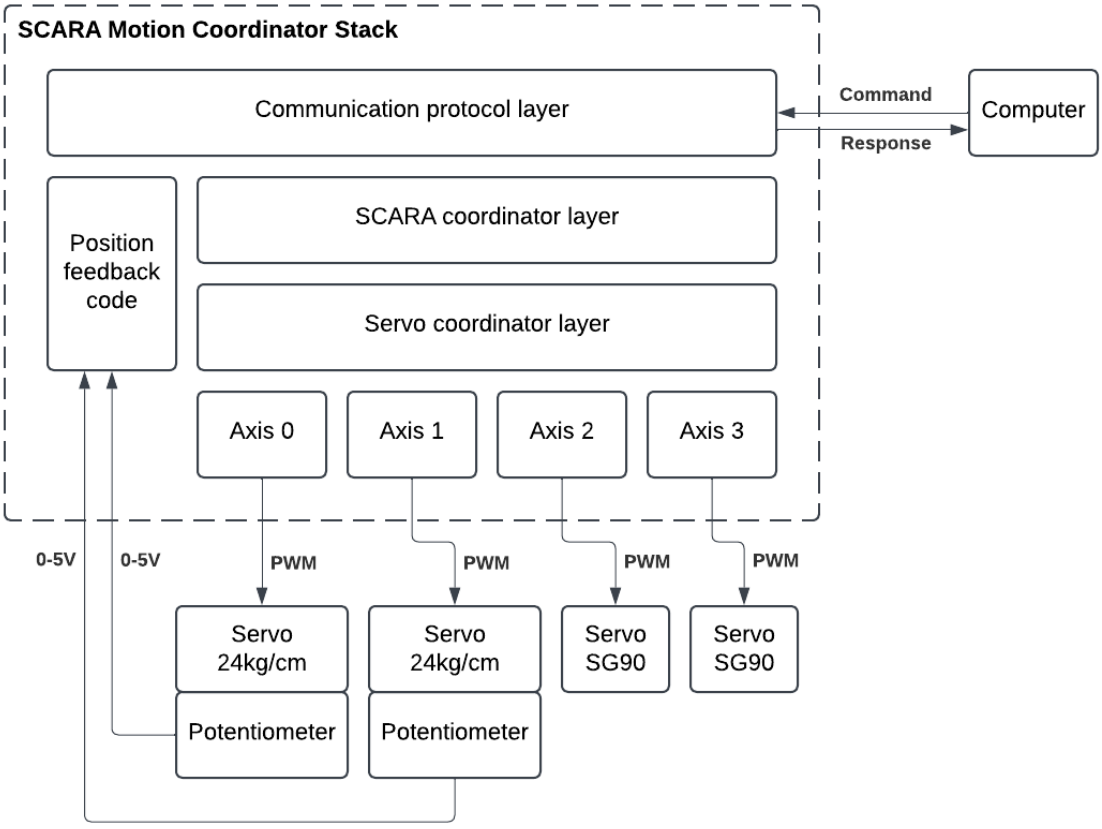


Figure 21: The SCARA Motion Coordinator Stack

The stack consists of the following layers:

- **Communication Protocol Layer** which handles the commands sent over serial.

- **SCARA Coordinator Layer** containing forward and inverse kinematics for the manipulator.
- **Servo Coordinator Layer** which implements velocity profiling and coordinated motion commands.
- **Servo Instances (Axes 0–3)** representing the individual servo motors, providing low-level control to set their positions.
- **Position feedback code** that converts raw ADC readings from the potentiometers to angles in degrees. This code is referenced throughout the stack.

The following sections provide a detailed explanation of the functionality of each layer.

3.1.1.1 Communication Protocol Layer

The Communication Protocol Layer handles commands transmitted over the UART2 serial interface (USB port), using a custom 48-byte (12-word) communication protocol. Commands are sent to the controller via a bespoke Python API library, ***scara_motion_controller_api.py***, which is responsible for encoding the commands and transmitting them over serial. The list of available Python API commands and their corresponding protocol are described below:

scara_motion_controller_api.py	
COMMAND	**DESCRIPTION**
SCARA_INITIALISE	Initialise SCARA with specified parameters
PYTHON API COMMAND: **SCARA_INITIALISE(link_1, link_2, z_min, z_max, settling_time, P_0, I_0, D_0, P_1, I_1, D_1)** `float` **link_1**: link length 1 in m `float` **link_2**: link length 2 in m `float` **z_min**: z co-ordinate of end effector at full extension `float` **z_max**: z co-ordinate of end effector at full retraction `float` **settling_time**: settling time after each joint move in seconds `float` **P_0, I_0, D_0**: P, I and D gains for axis 0 `float` **P_1, I_1, D_1**: P, I and D gains for axis 1 PROTOCOL [48 bytes]: **(|4 bytes|4 bytes|…)* **|command|link_1|link_2|z_min|z_max|settling_time|P_0 |I_0 |D_0 |P_1 |I_1 |D_1 |** **|1 |aaaa |bbbb |cccc |dddd |eeee |ffff|gggg|hhhh|iiii|jjjj|kkkk|**	
SCARA_AUTO_CALIBRATE	Auto angle calibration of Axis 0 and 1
PYTHON API COMMAND: **SCARA_AUTO_CALIBRATE()** PROTOCOL [48 bytes]: **|command|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|** **|2 |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|**	
SCARA_MOVE_JOINT	Move a joint to specified angle
PYTHON API COMMAND: **SCARA_MOVE_JOINT(axis, angle, time)**	

`int` **axis**: axis (0, 1, 2, 3)
`float` **angle**: angle
`float` **time**: time taken to reach angle

PROTOCOL [48 bytes]:

|command|axis|angle|time|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|3 |aaaa|bbbb |cccc|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

SCARA_MOVE_JOINTS	Move all joints to specified angles

PYTHON API COMMAND:

SCARA_MOVE_JOINTS(angle_0, angle_1, angle_2, angle_3, time)

`float` **angle_0**: desired angle of axis 0
`float` **angle_1**: desired angle of axis 1
`float` **angle_2**: desired angle of axis 2
`float` **angle_3**: desired angle of axis 3
`float` **time**: time taken for axes to reach desired angles

PROTOCOL [48 bytes]:

|command|angle_0|angle_1|angle_2|angle_3|time|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|4 |aaaa |bbbb |cccc |dddd |eeee|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

SCARA_MOVE_COORD	Move end effector to specified co-ordinate

PYTHON API COMMAND:

SCARA_MOVE_COORD(x, y, z_angle, z, time)

`float` **x**: desired x co-ordinate of end effector in m
`float` **y**: desired y co-ordinate of end effector in m
`float` **z_angle**: desired z angle of end effector in degrees
`float` **z**: desired z co-ordinate of end effector in m
`float` **time**: time taken for end effector to reach desired co-ordinates

PROTOCOL [48 bytes]:

|command|x |y |z_angle|z |time|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|5 |aaaa|bbbb|cccc |dddd|eeee|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

SCARA_READ_AXIS	Returns the angle of specified axis

PYTHON API COMMAND:

SCARA_READ_AXIS(axis)

`int` **axis**: axis (0, 1, 2, 3)

PROTOCOL [48 bytes]:

|command|axis|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|6 |aaaa|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

PROTOCOL (RETURN) [48 bytes]:

|command|angle|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|6 |aaaa |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

SCARA_READ_COORD	Returns the coordinates (x, y, z_angle, z) of the end effector

PYTHON API COMMAND:

SCARA_READ_COORD()

PROTOCOL [48 bytes]:

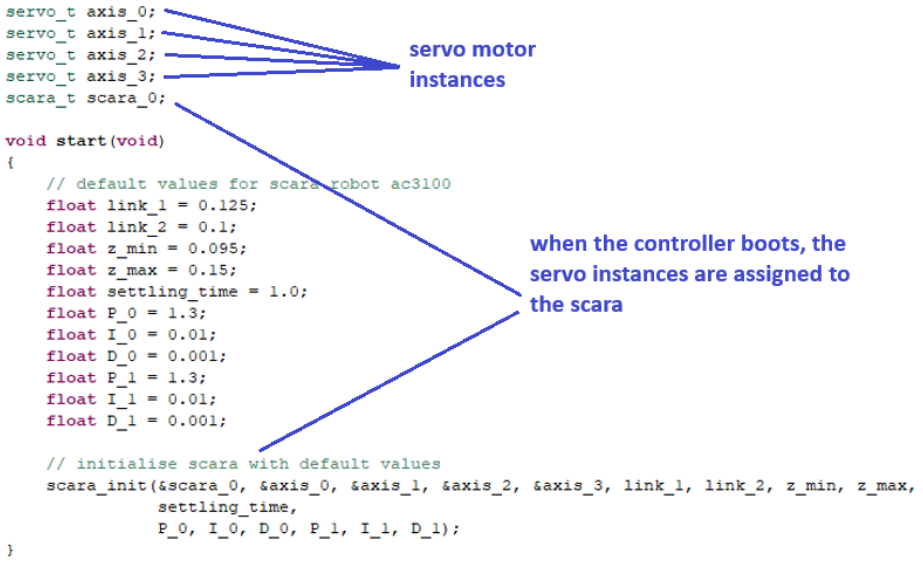
```
|command|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|7      |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
```

PROTOCOL (RETURN) [48 bytes]:

```
|command|x    |y    |z_angle|z    |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
|7      |aaaa|bbbb|cccc   |dddd|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
```

Table 5: scara_motion_controller_api.py commands list with protocol

When the motion controller boots, the SCARA robot instance is initialised with a default configuration:



The Communication Protocol Layer then waits for incoming commands using a blocking receive call. Once a complete command (48 bytes) is received, the *command_id* (first word) is inspected, and a switch-case statement is entered to determine the method for processing the received data.

As an example, the **SCARA_INITIALISE** handler (*command_id*=1) is shown below, where each word in the data stream is copied into floating point value and used in the *scara_reinit* command call. Upon a successful command call, the Communication Protocol Layer transmits a *success_flag* back to the sender, signalling that it is ready to receive the next command.

```
void update(void)
{
    int32_t success_flag = 1; // success flag

    uint8_t rxdata[48];  // 48 bytes total
    if (HAL_UART_Receive(&huart3, rxdata, 48, HAL_MAX_DELAY) == HAL_OK)
    {
        memcpy(&command, &rxdata, sizeof(command_t));

        switch (command.command_id)
        {
        case 1:         // SCARA_INITIALISE
            // |0001   |aaaa  |bbbb  |cccc |dddd |eeee         |ffff|gggg|hhhh|iiii|jjjj|kkkk|
            // |command|link_1|link_2|z_min|z_max|settling_time|P_0 |I_0 |D_0 |P_1 |I_1 |D_1 |

            float link_1; memcpy(&link_1, &command.values[0], sizeof(float));
            float link_2; memcpy(&link_2, &command.values[1], sizeof(float));
            float z_min; memcpy(&z_min, &command.values[2], sizeof(float));
            float z_max; memcpy(&z_max, &command.values[3], sizeof(float));
            float settling_time; memcpy(&settling_time, &command.values[4], sizeof(float));
            float P_0; memcpy(&P_0, &command.values[5], sizeof(float));
            float I_0; memcpy(&I_0, &command.values[6], sizeof(float));
            float D_0; memcpy(&D_0, &command.values[7], sizeof(float));
            float P_1; memcpy(&P_1, &command.values[8], sizeof(float));
            float I_1; memcpy(&I_1, &command.values[9], sizeof(float));
            float D_1; memcpy(&D_1, &command.values[10], sizeof(float));

            scara_reinit(&scara_0,
                         link_1, link_2, z_min, z_max,
                         settling_time,
                         P_0, I_0, D_0, P_1, I_1, D_1);

            HAL_UART_Transmit(&huart3, (uint8_t*)&success_flag, sizeof(int32_t), HAL_MAX_DELAY);

            break;
```

always waiting for next command

datastream handled according to command_id

scara_init is called on boot, if user calls SCARA_INITIALISE, then *scara_reinit* is called, as pointers do not need to be reassigned

3.1.1.2 SCARA Coordinator Layer

The SCARA Coordinator Layer implements the forward and inverse kinematics for the manipulator configured for elbow-up (left-handed) mode:

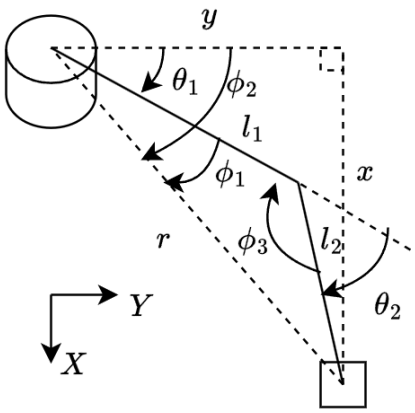


Figure 22: SCARA with "Elbow up" configuration in world frame

The forward kinematics, used to determine the current position of the end effector to send back to the user via the **SCARA_READ_COORD** command, can be calculated using the equations below. Feedback is only available for X and Y positions, as the potentiometers are attached to Axis 0 and 1 only:

$$x = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \qquad y = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

The forward kinematics implementation in the SCARA Coordinator Layer is shown below:

```
void calc_forward_kinematics(scara_t *_scara, float *_x, float *_y, float *_angle_z, float *_z,
                                              float _theta_1, float _theta_2, float _theta_3, float _theta_4)
{
    float l1 = _scara->link_1;
    float l2 = _scara->link_2;

    // convert angles from degrees to radians
    float t1 = _theta_1 * (PI / 180.0);
    float t2 = _theta_2 * (PI / 180.0);

    // compute X and Y position
    *_x = l1 * sin(t1) + l2 * sin(t1 + t2);
    *_y = l1 * cos(t1) + l2 * cos(t1 + t2);

    // compute total rotation angle around Z
    *_angle_z = _theta_1 + _theta_2 + _theta_3;

    // convert theta_4 (the servo angle) back to Z height
    *_z = _scara->z_min + ((_theta_4 - _scara->joint_3->min_angle) *
          (_scara->z_max - _scara->z_min) /
          (_scara->joint_3->max_angle - _scara->joint_3->min_angle));
}
```

scara link lengths configured by user

The inverse kinematics, used to calculate the joint angles required for the given link lengths to achieve the desired end effector X and Y positions used by the **SCARA_MOVE_COORD** command, can be solved using the inverse cosine rule:

$$r = \sqrt{x^2 + y^2}$$

$$\phi_1 = \cos^{-1}\left(\frac{l_2^2 - r^2 - l_1^2}{-2rl_1}\right) \qquad \phi_2 = \tan^{-1}\left(\frac{x}{y}\right) \qquad \phi_3 = \cos^{-1}\left(\frac{r^2 - l_1^2 - l_2^2}{-2l_1l_2}\right)$$

$$\boldsymbol{\theta_1 = \phi_2 - \phi_1} \qquad \boldsymbol{\theta_2 = 180 - \phi_3}$$

The inverse kinematics to achieve the desired Z coordinate and rotation can be determined by observing the Z axis mechanism designed for the EduSCARA manipulator:

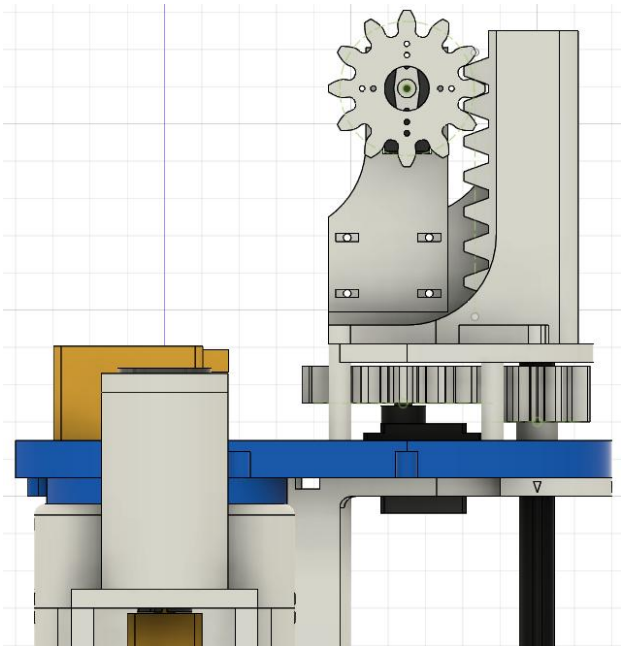


Figure 23: Small SCARA Z axis mechanism

The desired Z angle θ_Z in world frame is achieved by rotating Axis 3 to angle θ_3:

$$\boldsymbol{\theta_3 = \theta_Z - (\theta_1 + \theta_2)}$$

Axis 4 converts rotational motion into linear motion through a rack and pinion mechanism. The required angle of Axis 4 (θ_4) to achieve the desired Z coordinate is calculated by mapping the servo's rotational range to the linear travel range of the Z axis. The limits, *z_min* and *z_max* are user-defined as they depend on the end-effector attached to the Z axis. The Z coordinate is therefore determined by:

$$\boldsymbol{\theta_4 = minangle_4 + \left((z - z\_min) \times \frac{maxangle_4 - minangle_4}{z\_max - z\_min} \right)}$$

The inverse kinematics implementation in the SCARA Coordinator Layer is shown below:

```
void calc_inverse_kinematics(scara_t *_scara, float *_theta_1, float *_theta_2, float *_theta_3, float *_theta_4,
                                              float _x, float _y, float _angle_z, float _z)
{
    float l1 = _scara->link_1;
    float l2 = _scara->link_2;
    float x = _x;
    float y = _y;

    float r = sqrt((x * x) + (y * y));

    float phi1 = acos(((l2 * l2) - (r * r) - (l1 * l1)) / (-2 * r * l1));
    float phi2 = atan2(x, y);
    float phi3 = acos(((r * r) - (l1 * l1) - (l2 * l2)) / (-2 * l1 * l2));
    // convert z displacement to servo angle in degrees
    float phi4 = _scara->joint_3->min_angle + ((_z - _scara->z_min)
                 * (_scara->joint_3->max_angle - _scara->joint_3->min_angle) / (_scara->z_max - _scara->z_min));

    *_theta_1 = (phi2 - phi1) * (180.0 / PI);   // convert to degrees
    *_theta_2 = (PI - phi3) * (180.0 / PI);     // convert to degrees
    *_theta_3 = _angle_z - (*_theta_1 + *_theta_2);
    *_theta_4 = phi4;
}
```

3.1.1.3 Servo Coordinator Layer

The Servo Coordinator Layer implements velocity profiling described in **Section 2.1.2** and coordinated motion commands. This layer contains the underlying logic for:

- **SCARA_MOVE_JOINT**: Move joint to specified angle in T seconds
- **SCARA_MOVE_JOINTS**: Move all joints to specified angles in T seconds

**SCARA_MOVE_COORD* *is a SCARA_MOVE_JOINTS call using inverse kinematics to determine the angle input parameters to achieve a desired coordinate in world frame:*

```
void scara_move_coord(scara_t *_scara, float _x, float _y, float _z_angle, float _z, float _T)
{
    float theta_1, theta_2, theta_3, theta_4;
    calc_inverse_kinematics(_scara, &theta_1, &theta_2, &theta_3, &theta_4, _x, _y, _z_angle, _z);
    move_js_scara(_scara->servo_controller_scara, theta_1, theta_2, theta_3, theta_4, _T);
}
```

The **SCARA_MOVE_JOINT** and **SCARA_MOVE_JOINTS** commands contain the same underlying logic, and includes the following features which improve accuracy:

- Double-S trajectory
- Real-time PID control (available for Axis 0 and 1)
- Backlash correction (available for Axis 0 and 1)

The operation of each axis during a move is shown in Figure 24:

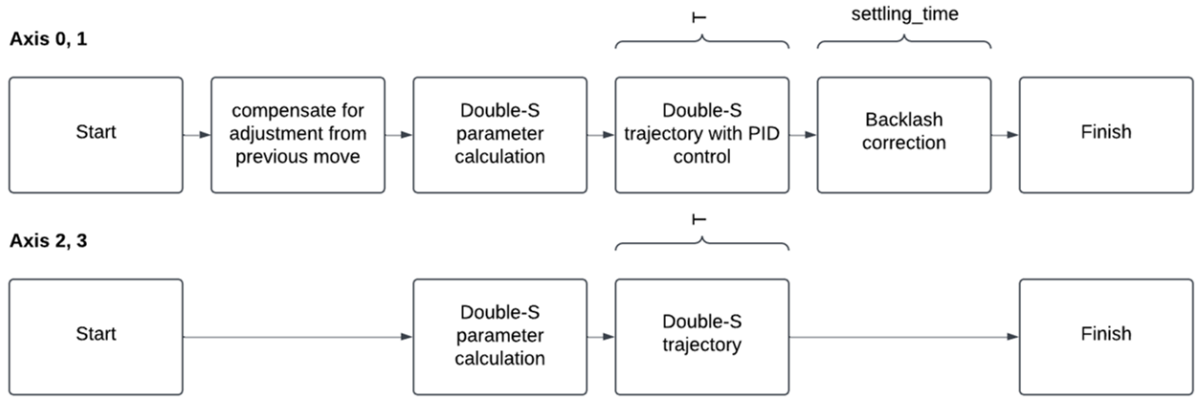


Figure 24: Operation block diagram of each axis during a move

The following sections provide a detailed explanation of the implementation of each feature.

3.1.1.3.1 Double-S trajectory implementation

The servo position is controlled by adjusting the PWM signal sent to it. The motor's speed is not controlled directly, instead, its position is specified at discrete time intervals. To achieve the desired motion profile, the displacement equations for each phase of the Symmetric Double-S trajectory outlined in **2.1.2** will be used:

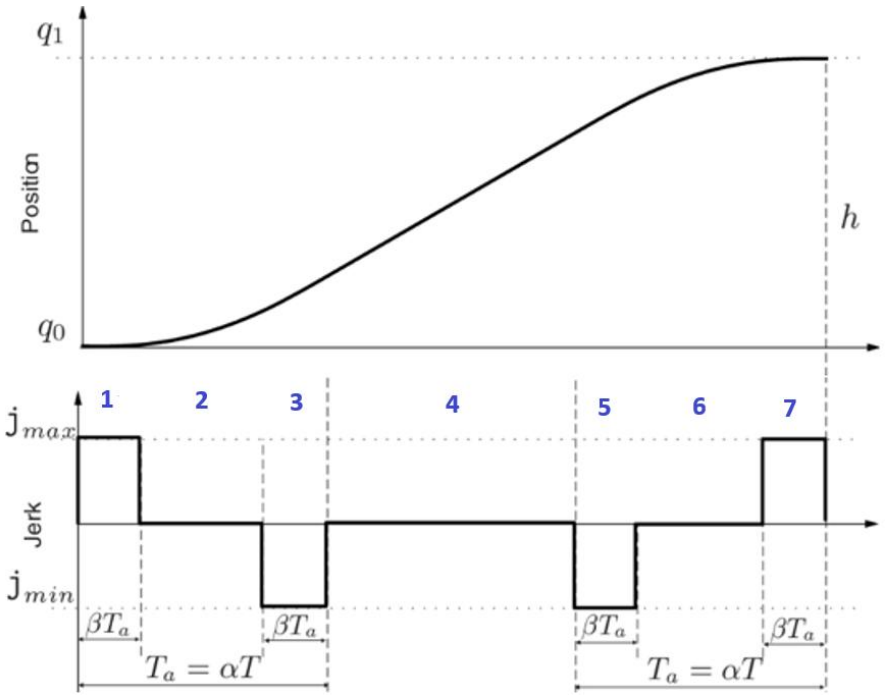


Figure 25: Double-S trajectory with 7 phases [19]

To achieve the desired angular displacement within a specified time while maintaining the Symmetric Double-S profile, the following constants must be determined:

- T_a: acceleration and deceleration durations
- T_j: jerk phase durations

- v_{max}: target velocity during constant velocity phase
- a_{max}: target acceleration during constant acceleration phases
- j_{max}: target jerk during constant jerk phases

alpha and *beta* are adjustable parameters, *alpha* defines the duration of the acceleration and deceleration phases as a proportion of the total profile time, while *beta* specifies the duration of the jerk phase as a proportion of the acceleration/deceleration phases. By default, $\alpha = 0.3$ and $\beta = 0.3$. Figure 26 demonstrates the effects of changing of α and β:

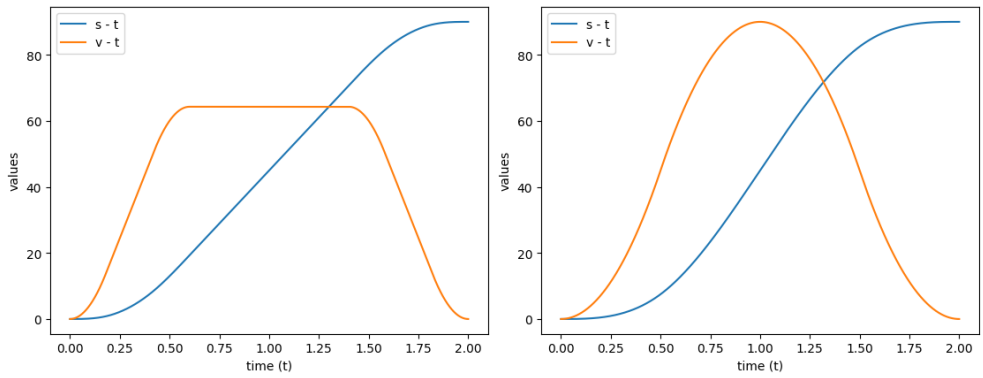


Figure 26: Python simulation of Symmetric Double-S profile (displacement: s, velocity: v) for 0 to 90 degree move in 2 seconds, with $\alpha = 0.3, \beta = 0.3$ (left) and $\alpha = 0.5, \beta = 0.5$ (right)

This is implemented in the Servo Coordinator Layer using the *calc_double_s_parameters* function, and called before starting every move:

```
void calc_double_s_parameters(float T, float h, float alpha, float beta,
                              float *T_a, float *T_j,
                              float *v_max, float *a_max, float *j_max)
{
    // accel / decel duration    0 < alpha < 1/2
    *T_a = alpha * T;
    // jerk duration             0 < beta < 1/2
    *T_j = beta * (*T_a);
    // calculate maximum velocity
    *v_max = h / ((1 - alpha) * T);
    // calculate maximum acceleration
    *a_max = h / (alpha * (1 - alpha) * (1 - beta) * T * T);
    // calculate maximum jerk
    *j_max = h / (alpha * alpha * beta * (1 - alpha) * (1 - beta) * T * T * T)
}
```

Using these constants, equations **(2.12-2.18)** can be implemented in the *calc_double_s_value* function to find the desired displacement for specified time. This function is called every *SAMPLING_TIME* milliseconds:

```
// calculate samples
int samples = (int)(T * 1000.0 / SAMPLING_TIME);

for (int i = 0; i < samples; i++)
{
    double t = i * SAMPLING_TIME / 1000.0; // Convert sampling time to seconds

    // desired position using double S-curve profile
    float dpos = cpos + calc_double_s_value(T, h, T_a, T_j, v_max, a_max, j_max, t);
```

The *calc_double_s_value* function takes a time elapsed *t* and returns desired position *dpos*:

```
float calc_double_s_value(float T, float h, float T_a, float T_j,
                          float v_max, float a_max, float j_max,
                          float t)
{                                                    equations...
    // acceleration phase
    if (t >= 0 && t <= T_j)                                  (2.12)
    {
        return j_max * (pow(t, 3) / 6.0);
    }
    else if (t > T_j && t <= T_a - T_j)                      (2.13)
    {
        return (a_max / 6.0) * ((3 * pow(t, 2)) - (3 * T_j * t) + pow(T_j, 2));
    }
    else if (t > T_a - T_j && t <= T_a)                      (2.14)
    {
        return (v_max * (T_a / 2.0)) - (v_max * (T_a - t)) - (-j_max * (pow(T_a - t, 3) / 6.0));
    }
    // constant velocity phase
    else if (t > T_a && t <= T - T_a)                        (2.15)
    {
        return v_max * (T_a / 2.0) + (v_max * (t - T_a));
    }
    // deceleration phase
    else if (t > T - T_a && t <= T - T_a + T_j)              (2.16)
    {
        return h - (v_max * (T_a / 2.0)) + v_max * (t - T + T_a) - (j_max * pow(t - T + T_a, 3) / 6.0);
    }
    else if (t > T - T_a + T_j && t <= T - T_j)              (2.17)
    {
        return h - (v_max * (T_a / 2.0)) + v_max * (t - T + T_a)
               - ((a_max / 6.0) * (3 * pow(t - T + T_a, 2) - 3 * T_j * (t - T + T_a) + pow(T_j, 2)));
    }
    else if (t > T - T_j && t <= T)                          (2.18)
    {
        return h - (j_max * (pow(T - t, 3) / 6.0));
    }
    else
    {
        return 999;          if t not in range 0 < t < T, return obvious error value for debugging
    }
}
```

3.1.1.3.2 Real-time PID control implementation (Axis 0 and 1)

PID (Proportional-Integral-Derivative) control is widely used in robotics for precise and stable motion control. It continuously calculates an error value between a desired setpoint and the current state and then corrects it using proportional, integral, and derivative terms [31]. This allows robots to handle disturbances and uncertainties in real-time [32].

The EduSCARA manipulator has position feedback sensors on Axis 0 and 1. Figure 27 illustrates how PID control is implemented on these Axes:

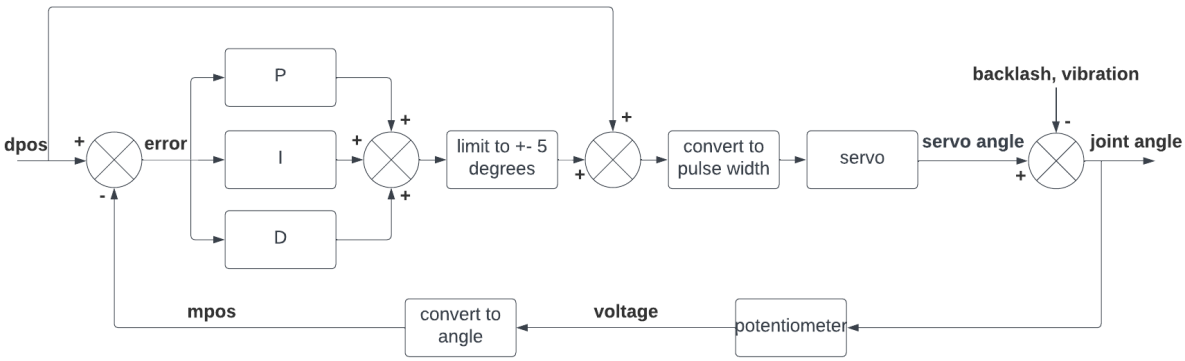


Figure 27: PID control system during joint move command

P, I and D gains are adjustable parameters for Axis 0 and 1, and can be changed at any time by the user, using the **SCARA_INITIALISE** API call. The PID correction described in Figure 27 is implemented after calculating the next desired position:

```
// ----- control loop -----
if (_axis == 0 || _axis == 1)
{
    // error calculation
    float pot_read = potentiometers_read_angle(_axis);
    float error = dpos - pot_read;

    // PID terms
    float P_term_move = error * P;

    // integral term (with anti-windup)
    error_sum += error * (SAMPLING_TIME / 1000.0);
    if (error_sum > max_integral)
        error_sum = max_integral;
    else if (error_sum < -max_integral)
        error_sum = -max_integral;

    float I_term_move = error_sum * I;

    // derivative term
    float D_term_move = ((error - last_error) / (SAMPLING_TIME / 1000.0)) * D;
    last_error = error;

    // PID output
    float pid_output = P_term_move + I_term_move + D_term_move;

    // limit PID output to +-5 degrees
    float adjusted_error;
    if (pid_output > 5)
    {
        adjusted_error = 5;
    }
    else if (pid_output < -5)
    {
        adjusted_error = -5;
    }
    else
    {
        adjusted_error = pid_output;
    }

    // set adjusted position with PID correction
    servo_set_angle(axis, dpos + adjusted_error);
}
// ------------------------
```

3.1.1.3.3 Backlash correction implementation (Axis 0 and 1)

To address lost motion caused by backlash and its detrimental impact on joint angle accuracy, as discussed in **Section 2.2**, a 'settling period' is introduced after each movement. During this time, the control system described below attempts to correct this error:

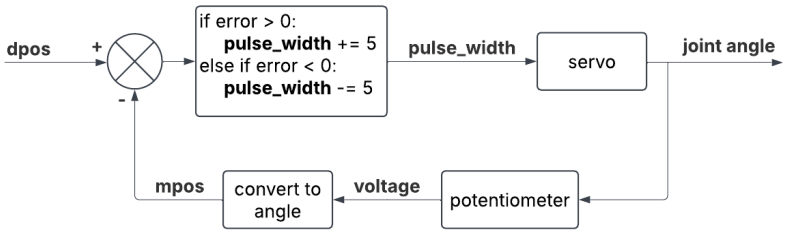


Figure 28: 'settling period' lost motion correction system

The control system illustrated in Figure 28 adjusts the pulse width of the PWM signal sent to the motor by ±5 microseconds. As the selected servo motor has a 'deadband' of 4 microseconds, any changes within this range do not result in motor movement due to the motor's built-in tolerance to small signal variations.

This deadband is a key reason behind the control system's design, as through testing, it was observed that a standard PID loop does not generate sufficient torque quickly enough to correct small-angle movements, as these fall within the motor's non-responsive range. The control system is applied for *SETTLING_TIME_SAMPLES* following the 'Double-S trajectory with PID control' phase. It is considered satisfactory when the joint angle remains within ±0.5 degrees of the target, preventing excessive oscillations that could accumulate and degrade accuracy:

```
// settling time (seconds) to reduce backlash
int settling_samples = (int)(_servo_controller_scara->settling_time * 1000.0 / SAMPLING_TIME_SETTLING);

for (int i = 0; i < settling_samples; i++)
{
    // ----- control loop -----
    if (_axis == 0 || _axis == 1)
    {
        float pot_read = potentiometers_read_angle(_axis);

        if (tpos - pot_read < -0.5) {
            pulse_width_correction = 5;
        } else if (tpos - pot_read > 0.5) {
            pulse_width_correction = -5;
        } else {
            pulse_width_correction = 0;
        }

        float current_pulse_width = servo_get_pulse_width_us(axis);
        float adjusted_pulse_width = current_pulse_width + pulse_width_correction;

        servo_set_pulse_width_us(axis, adjusted_pulse_width);
    }

    HAL_Delay(SAMPLING_TIME_SETTLING);
}
```

Following this adjustment, a discrepancy arises between the stored current position *cpos* of the axes in memory and the actual joint position. As a result, at the start of the next move the servo 'jumps' to the *cpos* value stored in memory:

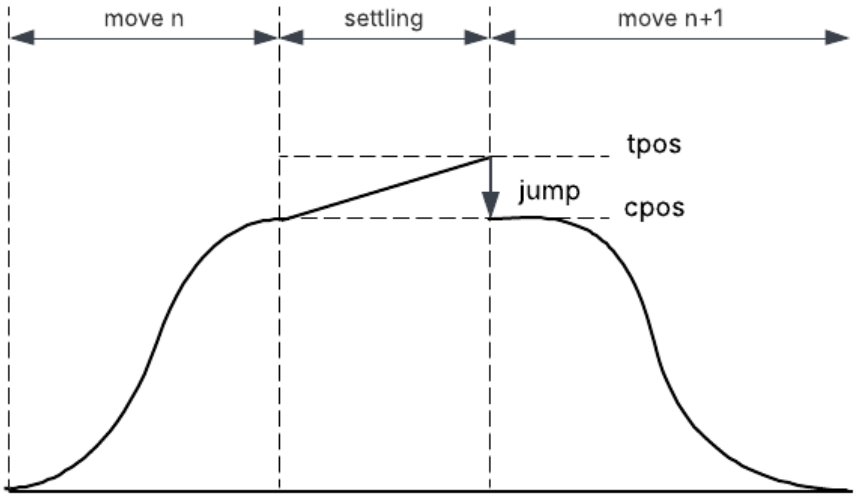


Figure 29: Subsequent moves without adjustment for compensation from previous move

Therefore, *cpos* must be updated to reflect the actual joint angle by reading the potentiometer before and after the settling phase and storing the difference in *last_angle_adjustment*:

```
// set last angle adjustment from settling
if (_axis == 0 || _axis == 1)
{
    axis->last_angle_adjustment = angle_before_settling - potentiometers_read_angle(_axis);
}
```

Then, at the start of the next move, *cpos* is adjusted by half the measured discrepancy (a 0.5 factor determined through trial and error) to compensate for the non-linear relationship between PWM pulse width, and the actual servo joint angle caused by backlash. Any remaining discrepancy is handled by biasing the initial 'jump' in the direction of the upcoming movement, making it less noticeable:

```
// compensate for adjustment from previous move
float cpos = axis->cpos;
if (_axis == 0 || _axis == 1)
{
    float cpos_adjustment;
    if (axis->last_angle_adjustment < 0)
    {
        cpos_adjustment = -(axis->last_angle_adjustment) * 0.5;
    }
    else
    {
        cpos_adjustment = axis->last_angle_adjustment * 0.5;
    }

    if (_angle - cpos > cpos_adjustment)
    {
        cpos += cpos_adjustment;
    }
    else if (_angle - cpos < cpos_adjustment)
    {
        cpos -= cpos_adjustment;
    }
}
```

The internal feedback in the servo does not know whether its undershot or overshot, so a consistent method to reduce the 'jump' is to 'jump' half the offset in the direction of the next move

This approach results in chained moves that closely approximates the desired behaviour:

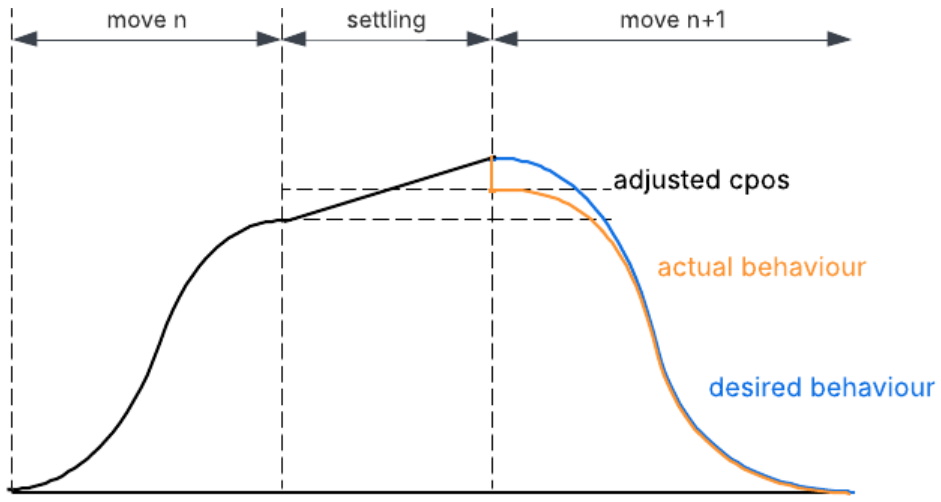


Figure 30: Subsequent moves with adjustment for compensation from previous move

3.1.1.4 Servo Instances (Axes 0–3)

Each Servo Instance in the stack is responsible for converting a desired angle into a corresponding PWM pulse width. However, this relationship is not perfectly linear due to backlash and manufacturing tolerances. As a result, servo calibration is necessary to achieve higher positional accuracy. Axis 0 and 1 are equipped with position feedback sensors, allowing for automatic calibration using the **SCARA_AUTO_CALIBRATE** function. In contrast, Axis 3 and 4 lack feedback sensors and therefore rely purely on manual calibration.

Figure 31: Servo angles with corresponding PWM pulse widths [33]

3.1.1.4.1 Automatic Servo Calibration

Before **SCARA_AUTO_CALIBRATE** is called for the first time, the servos are initialized with default values representing their manually calibrated state:

```
servo_init(_axis_0, 2500, 500, 1500, -90, 90, 0, 0, 0);
servo_init(_axis_1, 2500, 500, 1500, -90, 30, 90, 0, 1);
servo_init(_axis_2, 500, 2500, 1550, -180, 180, 0, 0, 2);
servo_init(_axis_3, 500, 2300, 1500, 90, -60, 0, 0, 3);
```

Using the *servo_init* function:

```
void servo_init(servo_t *_servo, float _min_pulse_width, float _max_pulse_width, float _zero_pulse_width,
                float  min angle, float  max angle, float  offset angle, float  cpos, int  channel);
```

Cases where the *min_pulse_width* is greater than the *max_pulse_width* indicate that the servo is mounted upside down or reversed relative to the joint frame.

The pseudocode for the auto-calibration sequence for Axis 0 and 1 is as follows:

```
PSEUDOCODE - AUTO-CALIBRATION PER AXIS
>> set the pulse width to zero_pulse_width

>> sweep towards max_pulse_width by increasing pulse width by 5 each time, while
sweeping check potentiometer angle

>> when potentiometer angle = max_angle, set calib_max_pulse_width to pulse width

>> sweep towards min_pulse_width by decreasing pulse width by 5 each time, while
sweeping check potentiometer angle

>> when potentiometer angle = min_angle, set calib_min_pulse_width to pulse width
```

```
>> sweep towards max_pulse_width by increasing pulse width by 5 each time, while
sweeping check potentiometer angle

>> when potentiometer angle = 0, set calib_zero_pulse_width to pulse width

If the servo is mounted in reverse, follow the inverse logic.
```

Once calibrated, *calib_max_pulse_width*, *calib_min_pulse_width* and *calib_zero_pulse_width* are used in-place of *max_pulse_width*, *min_pulse_width* and *zero_pulse_width*. These values are referenced whenever *servo_set_angle* is called:

```
void servo_set_angle(servo_t *_servo, float _angle)
{
    // set the current position parameter
    _servo->cpos = _angle;

    float pulse_width_us;
    _angle = _angle - _servo->offset_angle;
    if (_angle > 0)
    {
        pulse_width_us = _servo->calib_zero_pulse_width + ((_angle - 0)
                       * (_servo->calib_max_pulse_width - _servo->calib_zero_pulse_width)
                       / (_servo->max_angle - 0));
    }
    else if (_angle < 0)
    {
        pulse_width_us = _servo->calib_zero_pulse_width + ((_angle - 0)
                       * (_servo->calib_zero_pulse_width - _servo->calib_min_pulse_width)
                       / (0 - _servo->min_angle));
    }
    else
    {
        pulse_width_us = _servo->calib_zero_pulse_width;
    }

    // Convert pulse width (us) to timer counts for 333Hz PWM
    // timer counts = (pulse width in s x 90000000) / (270 * 1000000)
    uint16_t pulse_width = (uint16_t)((pulse_width_us * 9) / 27);
    __HAL_TIM_SET_COMPARE(_servo->pwm_timer, _servo->pwm_timer_channel, pulse_width);
}
```

the *servo_set_angle* assumes linearity between pulse width and angle:

the calibrated zero position pulse width is used, in case the servo is not centered at 1500us.

3.1.1.5 Position feedback code

The position feedback code is straightforward: the potentiometer's wiper is connected to an analog input pin, which is used to calculate a joint angle between 0 and 180 degrees. The accuracy of the joint angles are limited by the precision of the position sensors; therefore, the potentiometers must be calibrated manually. Calibration involves aligning the markers on the manipulator at known reference positions 0, 90, and -90 degrees for Axis 0, and 0 and 90 degrees for Axis 1, and then recording the corresponding raw analog values.

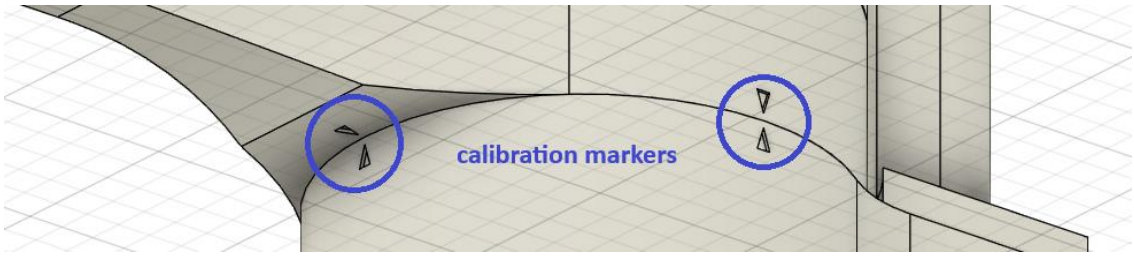


Figure 32: Calibration markers on EduSCARA Manipulator - Axis 0

These values are used in *potentiometers_init*, called during **SCARA_INITIALISE**:

```
void potentiometers_init()
{
    HAL_ADC_Start_DMA(&hadc1, pot_raw_value, POTS);

    // default values
    pots[0].min_angle = -90.0;
    pots[0].max_angle = 90.0;
    pots[0].min_raw_value = 700.0;
    pots[0].max_raw_value = 3300.0;

    pots[1].min_angle = 0.0;
    pots[1].max_angle = 90.0;
    pots[1].min_raw_value = 720.0;
    pots[1].max_raw_value = 2090.0;
}
```

These currently cannot be modified via API call, user must adjust these in firmware

max_angle is not the maximum angle of the joint, but rather the angle at the positive calibration point

These calibration values are used to scale the raw analog readings into joint angles:

```
float potentiometers_read_raw_value(int _channel)
{
    // range 0-4095
    return pot_raw_value[_channel];
}

float potentiometers_read_angle(int _channel)
{
    float angle = pots[_channel].min_angle
                + ((pot_raw_value[_channel] - pots[_channel].min_raw_value)
                * (pots[_channel].max_angle - pots[_channel].min_angle))
                / (pots[_channel].max_raw_value - pots[_channel].min_raw_value);
    return angle;
}
```

3.1.2 The EduSCARA Python API

scara_motion_controller_api.py provides a communication interface between a PC and the EduSCARA Motion Controller via serial. It uses the *pyserial* library to open a connection with the STM32, and sends encoded commands listed and described in Table 5.

The function call parameters are encoded into a byte stream using the struct module in little-endian format to match the STM32 memory architecture [34]. The *send_command* method writes this packet over serial and waits for a 4-byte success flag from the microcontroller, raising an exception if the communication fails. This operation allows each command to be complete on the microcontroller side before allowing the next command to be sent.

```
def send_command(self, command):
    print("sending command ...")
    self.api.write(command)

    success_flag = self.api.read(4)

    if len(success_flag) != 4:
        raise TimeoutError("system failure")

    flag, = struct.unpack('<i', success_flag)

    if flag != 1:
        raise ValueError(f"unexpected flag received: {flag}")

    print("... command successful")
```

send the command...

blocking wait for success flag

next command can be sent

The EduSCARA Python API includes both write and read commands. An example of a write command is **SCARA_MOVE_COORD**, which follows the standard communication pattern described above, sending a packed byte stream and waiting for a success flag:

```
def SCARA_MOVE_COORD(self, x, y, z_angle, z, time):

    # |0005   |aaaa|bbbb|cccc   |dddd|eeee|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
    # |command|x   |y   |z_angle|z   |time|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

    command = struct.pack('<iffffffiiiii', 5, x, y, z_angle, z, time, 0, 0, 0, 0, 0, 0)
    self.send_command(command)
```

Read commands such as **SCARA_READ_COORD** require the microcontroller to respond with multiple bytes of data. In these cases, the Python function must unpack the received byte stream and return multiple values:

```
def SCARA_READ_COORD(self):
    # |0007   |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
    # |command|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|

    command = struct.pack('<iiiiiiiiiiii', 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
    self.send_command(command)

    # |0007   |aaaa|bbbb|cccc   |dddd|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
    # |command|x   |y   |z_angle|z   |xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
    # Read 4 floats = 16 bytes
    response = self.api.read(16)

    if len(response) != 16:
        raise TimeoutError("No response received")

    x, y, z_angle, z = struct.unpack('<ffff', response)
    return x, y, z_angle, z
```

Sending commands using the EduSCARA Python API is straightforward. First, place ***scara_motion_controller_api.py*** in the script directory and import the module at the top of the script. Then, create an instance of the scara_motion_controller class, specifying the serial port. Then, simply call any of the available commands. An example script given below:

```
import scara_motion_controller_api as smc_api
import serial.tools.list_ports
import time

ports = list(serial.tools.list_ports.comports())
if not ports:
    print("No USB device found.")
else:
    port_list = [f"{port.device} - {port.description}" for port in ports]
    print(f"Available Ports:\n" + "\n".join(port_list))
    scara_port = input("\nEnter the COM port for EduSCARA (e.g., COM3):")
    scara = smc_api.scara_motion_controller(port=scara_port)

    link_1, link_2 = 0.124, 0.096
    z_min, z_max = 0.095, 0.15
    settling_time = 1.0
    P_0, I_0, D_0 = 1.3, 0.01, 0.001
    P_1, I_1, D_1 = 1.3, 0.01, 0.001

    scara.SCARA_INITIALISE(link_1, link_2, z_min, z_max,
                           settling_time,
                           P_0, I_0, D_0, P_1, I_1, D_1)
    time.sleep(1)
    scara.SCARA_AUTO_CALIBRATE()
    scara.SCARA_MOVE_COORD(0.13, 0.13, 45, 0.11, 1)
```

display all available COM ports...

connect to motion controller via port

(re)initialise the scara with new parameters

run auto calibration sequence

move to specified coordinate and angle

3.1.3 The EduSCARA Motion Controller Hardware

The SCARA Motion Coordinator Hardware includes the following components:

- 1x **NUCLEO-F446RE**: A cost-effective development board featuring an ARM Cortex-M4 CPU (180 MHz) with 128kB SRAM and 512kB Flash. This platform was selected for its high GPIO count within a compact form factor [35], as well as for its ease of configurability and programmability using the STM32CubeIDE. Its compatibility with Arduino shields enables the use of widely available prototyping boards, making it easier to maintain a compact hardware design.
- 1x **Arduino Prototyping Shield**: For prototyping purposes, this is used to mount wire terminals, enabling straightforward and clearly organised wiring that resembles the terminals commonly found in industrial motion controllers.
- 1x **100uF Capacitor**: Filter on the servo power supply rails.
- 1x **5V 3A Power supply**: Power supply to the servo motors.

The resulting wiring schematic and the EduSCARA Motion Controller are shown below:

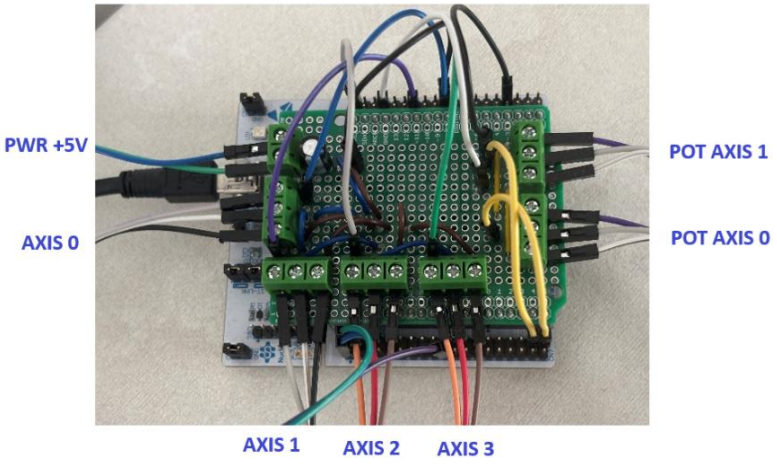


Figure 33: EduSCARA Motion Controller

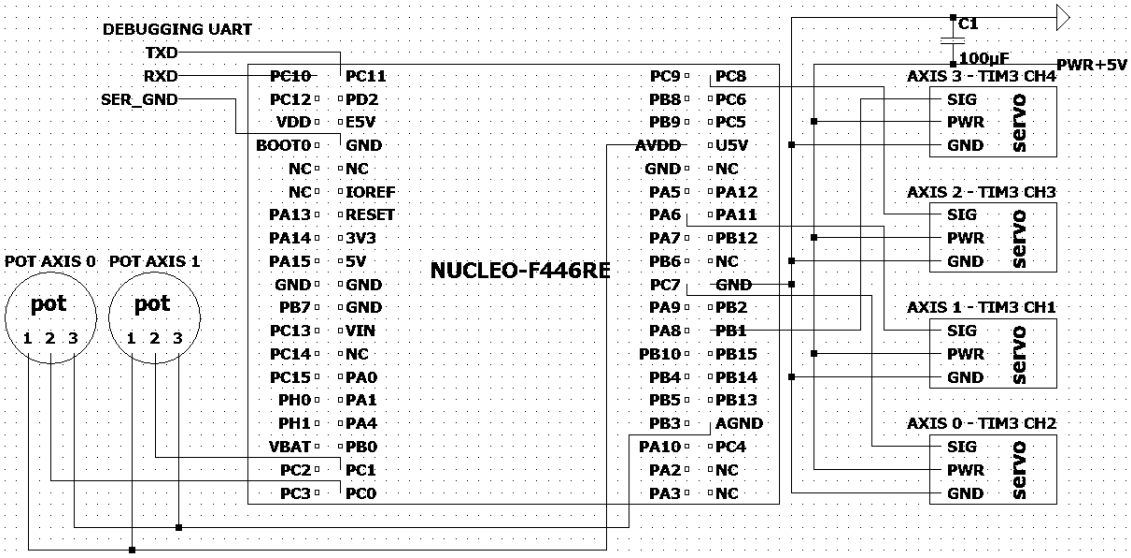


Figure 34: EduSCARA Motion Controller wiring schematic

3.2 The EduSCARA Manipulator

The EduSCARA Manipulator is designed to be a scaled-down yet functional equivalent of an industrial SCARA. Components are carefully selected to achieve balance of performance, affordability and accessibility, making the system ideal for education and prototyping.

To maximise adaptability and ease of manufacture, all structural components are 3D printable. Only screws, motors and sensors require separate sourcing. This approach enables users to customise their own manipulator, including changing the link lengths by simply modifying the *link_1* and *link_2* parameters for their manipulator in software using the **SCARA_INITIALISE** API call.

The following sections detail the evolution of the manipulator design through multiple iterations while highlighting key improvements leading to the final prototype. Afterwards, the performance of the system is assessed through experimental tuning of PID parameters and settling times, a recommended configuration is determined from these results that can be benchmarked against comparable platforms.

3.2.1 Manipulator Design Iteration 1

To drive the axes of a small-scale SCARA manipulator, the two most common and accessible actuator choices are servo motors and stepper motors, both of which are popular in small-scale robotic systems, with each offering distinct advantages and trade-offs [36]. The table below summarises the key differences relevant to this application:

Servo motors vs Stepper motors for small-scale SCARA manipulator		
Category	**Servo motor**	**Stepper motor**
Power supply	5-8V	12-24V
Power consumption	Low current draw	Moderate current draw
Precision	Moderate precision	High precision (open-loop)
Torque	Constant at all speeds, no slip due to internal closed-loop mechanism	High at low speeds, Low at high speeds, prone to slip
Ease of integration	Very easy, one signal line	Harder, interface with stepper drive
Wiring	Less wires	More wires
Temperature	Low temperature when working	Very high temperature when working
Size	Smaller + lighter	Larger + heavier
Cost	Moderate (for high torque servos)	Moderate (+ stepper drive)
Range of motion	180 degrees	Continuous rotation

Table 6: Servo motors vs Stepper motors comparison for small-scale SCARA manipulator

For the EduSCARA manipulator, servo motors were selected as the preferred actuators due to several advantages that align well with the project's educational and structural goals. Some key factors include their ease of use, compact size, low weight, compatibility with low-voltage DC power supplies and notably, their lower operating temperatures. The temperature is critical as stepper motors often reach 70, 80 or even 90°C during operation due to constant current draw [37], which poses a serious risk when using common 3D printing materials such as PLA, which often warp or deform around 60-65°C [38]. This makes it unsuitable for these structural parts unless additional cooling mechanisms or non-printed heat-resistant components are introduced. This would conflict with one of the project's core goals: that all manipulator structural components must be fully 3D printable and easily modifiable by users.

Furthermore, servo motors include internal end-stops, offering a level of safety, and preventing the robot from over-rotating and damaging itself, a common risk with stepper motors due to their continuous rotation capability and lack of inherent position limits unless external sensors are used. Although servo motors typically have a limited range of motion (up to 180 degrees), this is sufficient for the EduSCARA's workspace requirements, provided the robot operates in either an elbow-up or elbow-down configuration exclusively.

However, servo motors are not without drawbacks, as their internal closed-loop feedback relies on built-in potentiometers [39] whose accuracy is compromised by mechanical backlash, particularly in high-torque servos with complex gear trains. Despite this, the benefits of servos outweighed the drawbacks for the purposes of this application.

The servos chosen for Axis 0 and 1 are the AGFRC B53DHS, selected for their high torque output of 24 kgcm and programmability, enabling tuning of internal parameters such as travel range, neutral position, damping factor, and sensitivity:

Figure 35: AGFRC B53DHS servo [40]

The first prototype was developed solely to observe the behaviour of the servos when configured in a SCARA layout:

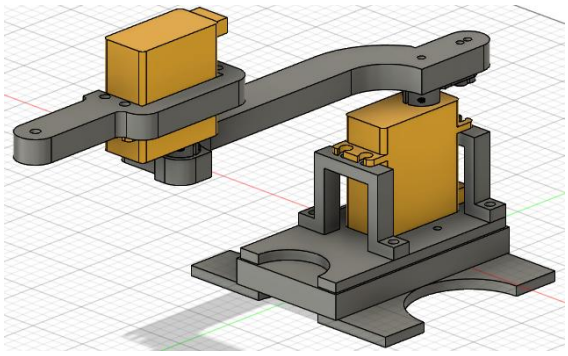


Figure 36: Prototype 1 (CAD)

It was observed that the weight of the arm caused it to slump forward, as the shaft alone was responsible for supporting the entire load. Additionally, the arm experienced excessive vibrations when stopping due to mechanical backlash, making it unusable even with a Double-S trajectory applied.

3.2.2 Manipulator Design Iteration 2

To address the vibration issues, the design requires the addition of mechanical damping and structural reinforcement or counterweights to reduce the excessive load on the shaft.

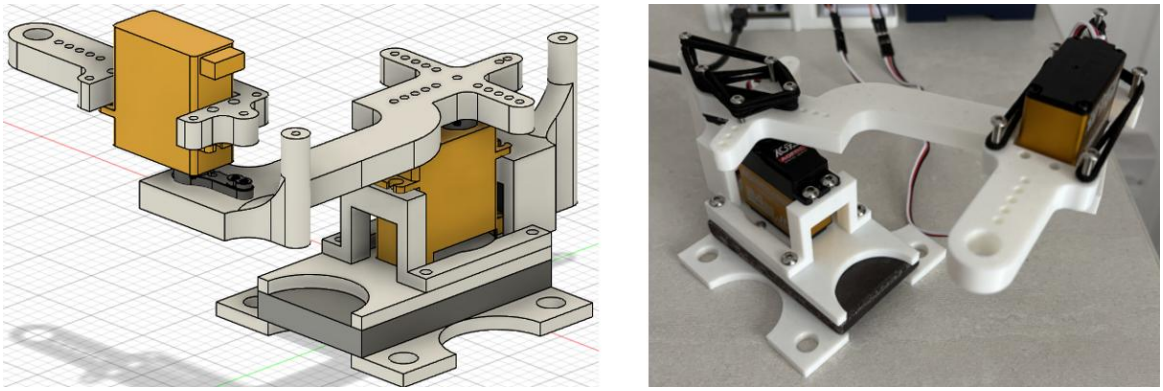


Figure 37: Prototype 2 (CAD) (left) and Prototype 2 (right)

This design incorporates elastic bands to apply tension on the gears while also serving as a counterweight. Although this approach effectively reduces vibration through mechanical damping, the tension varies with joint angle, leading to inconsistent joint behaviour. While the prototype confirms that damping improves stability, elastic bands are suboptimal in terms of both strength and consistency, particularly considering that future iterations will need to support the additional weight of Axes 3 and 4. As a result, the next prototype must explore a more robust and consistent method of mechanical damping, along with improvements in structural integrity.

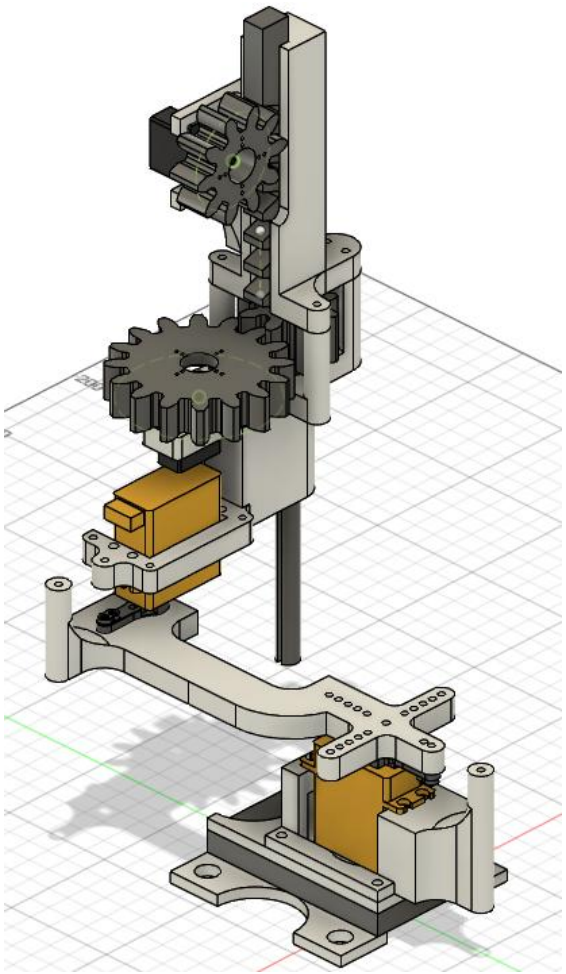


Figure 38: Prototype 2 (CAD) with Axis 3 and 4 mounted – not viable

3.2.3 Manipulator Design Iteration 3

The third iteration attempts a different strategy for mechanical damping using friction at the joints to reduce vibration. To ensure consistent friction and therefore damping throughout the full range of motion, a constant uniform force must be applied around each the joint. This is achieved by enclosing each motor within a circular housing and compressing the links against the joints. This method provides more predictable and consistent resistance compared to elastic bands used in the previous design.

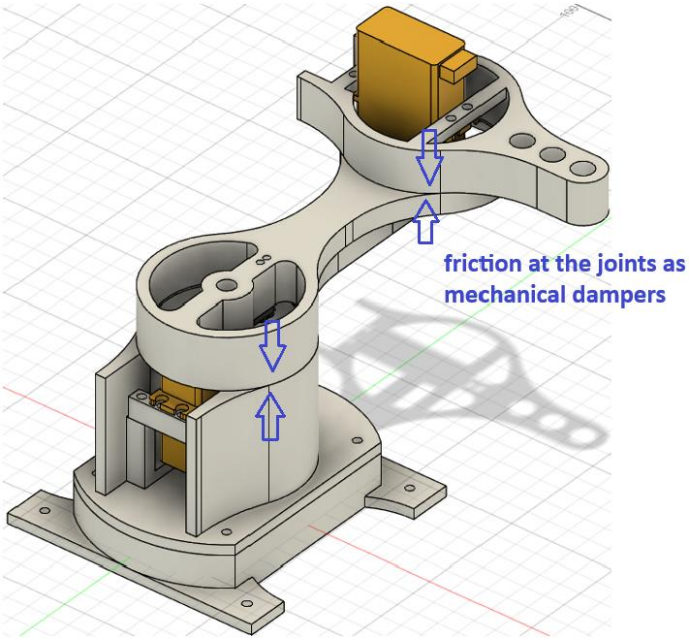


Figure 39: Prototype 3 (CAD)

The behaviour and structural integrity of the joints in this design is sufficient, making this suitable enough to proceed with mounting Axes 3 and 4 onto the structure.

3.2.4 Manipulator Design Iteration 4

The servo motors selected for Axis 3 and 4 are AGFRC B11DLS, metal-geared equivalents of the SG90 servo which offer a torque rating of 2.8 kgcm and improved durability for lightweight end-effector tasks:

Figure 40: AGFRC B11DLS servo [41]

Axis 3 and 4 control the rotation and vertical movement of the end-effector respectively. Axis 3 features a 1:2 gear ratio, enabling the end effector to achieve a full 360 degree rotation, while Axis 4 uses a rack and pinion mechanism that provides up to 50 mm of Z-axis displacement. The core mechanism excluding the housing alongside Prototype 4 is shown in Figure 41:

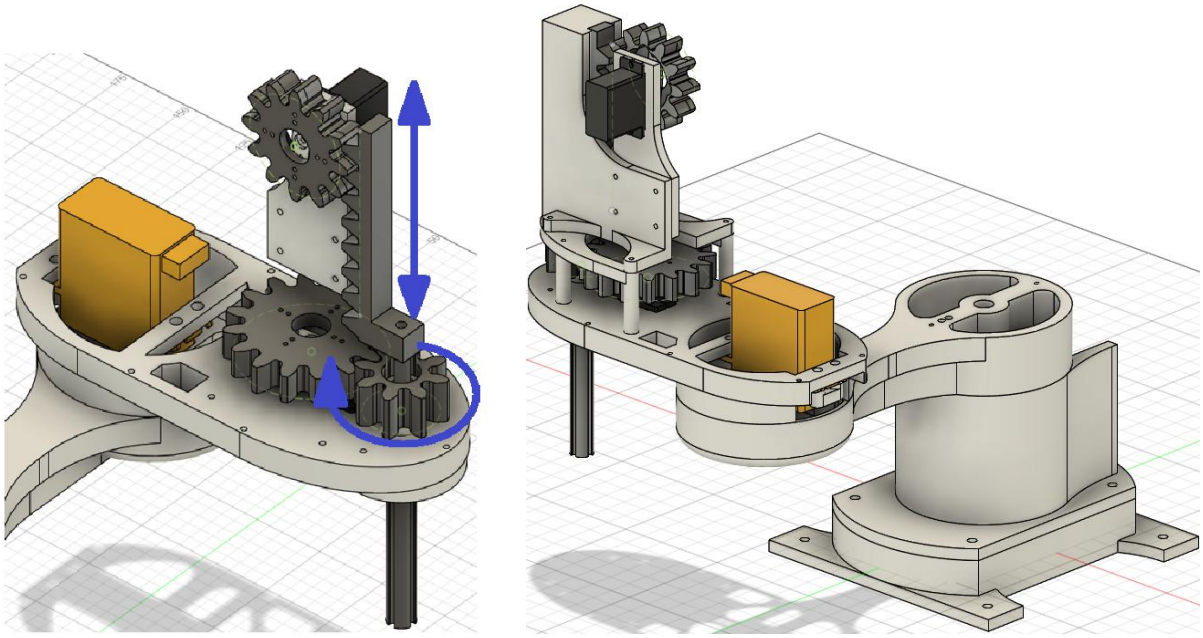


Figure 41: Z axis mechanism (no housing) (left), and Prototype 4 (CAD) (right)

With no external feedback sensors, Prototype 4's accuracy was visually assessed using **SCARA_MOVE_COORD** commands. When commanded to trace a rectangle, the SCARA's path showed deviations, revealing the limitations of relying solely on internal servo feedback:

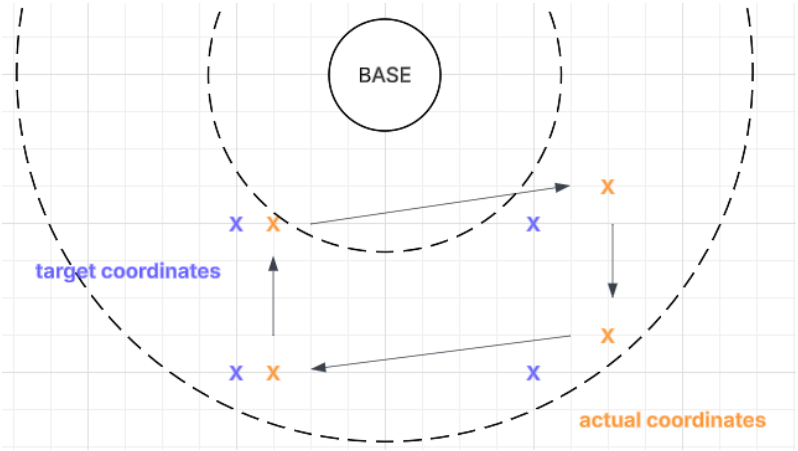


Figure 42: target vs observed coordinates through visual inspection – Prototype 4

3.2.5 Manipulator Design Iteration 5

Based on this observation, it is clear that relying solely on servo motor internal feedback is insufficient for accurate joint positioning. To address this, external position sensors were attached to directly measure joint angles in real-time. Since the servos operate within a 180 degree range, a single-turn potentiometer is suitable without risking mechanical damage. The chosen sensor is the RV24YN20F B502 5kΩ single-turn carbon film potentiometer, which strikes a good balance between precision, noise and power consumption [42]:

Figure 43: RV24YN20F B502 [43]

The potentiometer mounted to Axis 0 is housed in enclosures that features a dedicated hole for routing wires from the second joint for clean cable management:

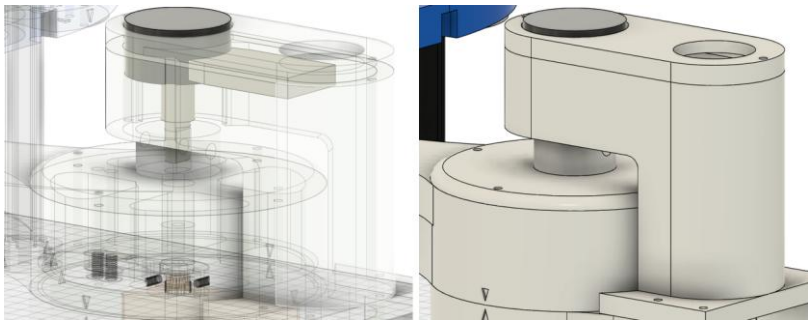


Figure 44: Potentiometer installed at Axis 0 on Prototype 5

The potentiometer mounted to Axis 1 is offset by 90 degrees to match the servo's orientation to accommodate the full range of motion in the elbow-up configuration:

Figure 45: Potentiometer installed at Axis 1 on Prototype 5

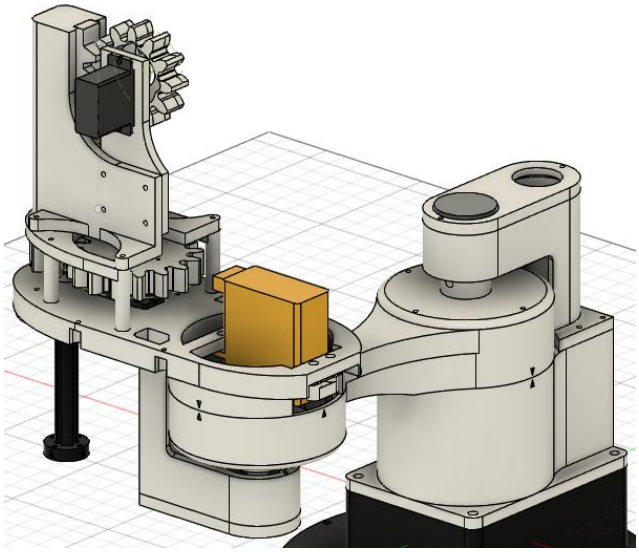


Figure 46: Prototype 5 (CAD)

3.2.6 Manipulator Design Iteration 6 – Final design

The core functionality of the SCARA is achieved in Prototype 5. The final design builds on this, incorporating key refinements. The added features of Prototype 6 are highlighted below:

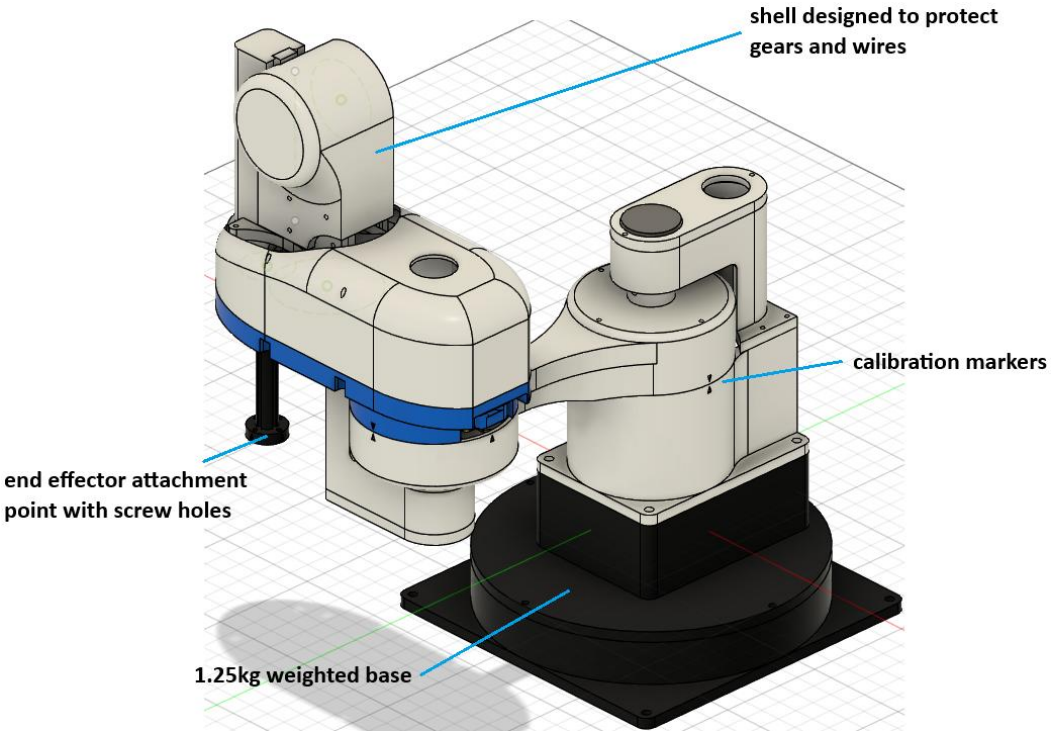


Figure 47: Prototype 6 - Final design (CAD)

Figure 48: Prototype 6 - Final design

3.2.7 Performance Analysis of the EduSCARA Manipulator

With the manipulator design complete, it must undergo a series of tests to evaluate the accuracy of the EduSCARA when operated using the EduSCARA Motion Controller and API. These tests aim to analyse the effect of varying key control parameters on end-effector accuracy: an essential metric for assessing the system's ability to perform "industrial tasks" at a small scale. The following variables will be examined:

- PID parameter tuning
- Settling time adjustment

Using these results, optimised settings for PID and settling time can be identified. These will then serve as a benchmark for comparing the performance of EduSCARA against similar systems. However, since most educational robotic arms operate in open-loop configurations, there is a lack of publicly available and reliable performance data. As a result, comparisons will be limited to the actuator specifications used by competitors assuming ideal integration within those systems.

3.2.7.1 PID tuning effect on performance

The goal of this test is to evaluate how PID control impacts positioning accuracy without the settling period. The P, I, and D gains are set to zero to create an open-loop configuration for Axis 0 and 1. Using the **SCARA_MOVE_JOINT** command, a sequence of moves is executed to span the full range of motion. The test sequence includes varying movement distances, directions and speeds to thoroughly assess performance across different motion scenarios.

The P, I, and D parameters are then tuned to improve accuracy. These values are not optimal, as the goal is to demonstrate the impact of PID tuning rather than to achieve maximum performance, since the system is intended to be user-customizable, fine-tuning is left to the user. Test data is obtained efficiently and automatically using **SCARA_READ_ANGLE** in the following script:

```
settling_time = 0
P_0 = 1.3, I_0 = 0.01, D_0 = 0.001
P_1 = 1.3, I_1 = 0.01, D_1 = 0.001

scara.SCARA_INITIALISE(link_1, link_2, z_min, z_max,
                       settling_time,
                       P_0, I_0, D_0, P_1, I_1, D_1)
scara.SCARA_MOVE_JOINT(2, 0.0, 1.0)
scara.SCARA_MOVE_JOINT(3, -60, 1.0)
scara.SCARA_MOVE_JOINT(0, 0.0, 1.0)
scara.SCARA_MOVE_JOINT(1, 90.0, 1.0)
scara.SCARA_AUTO_CALIBRATE()

axis = 0
move_time = 1
# list of target angles
if (axis == 0):
    target_angles = [0, 90, -90, -45, 45, -30, 30, 60, -60, 0, 1, 2, 4, 8, 16, 0, -1, -2, -4, -8, -16, 0]
elif (axis == 1):
    target_angles = [90, 120, 0, 120, 45, 60, 100, 120, 30, 90, 91, 92, 94, 98, 106, 90, 89, 88, 86, 82, 74, 90]

# open text file for writing
with open('robot_test_data.txt', 'w') as file:
    for angle in target_angles:
        scara.SCARA_MOVE_JOINT(axis, angle, move_time)
        time.sleep(1.0)
        measured_angle = scara.SCARA_READ_ANGLE(axis)
        print(measured_angle)

        file.write(f'{measured_angle}\n')
```

The graphs below compare the open-loop performance against a tuned PID configuration:

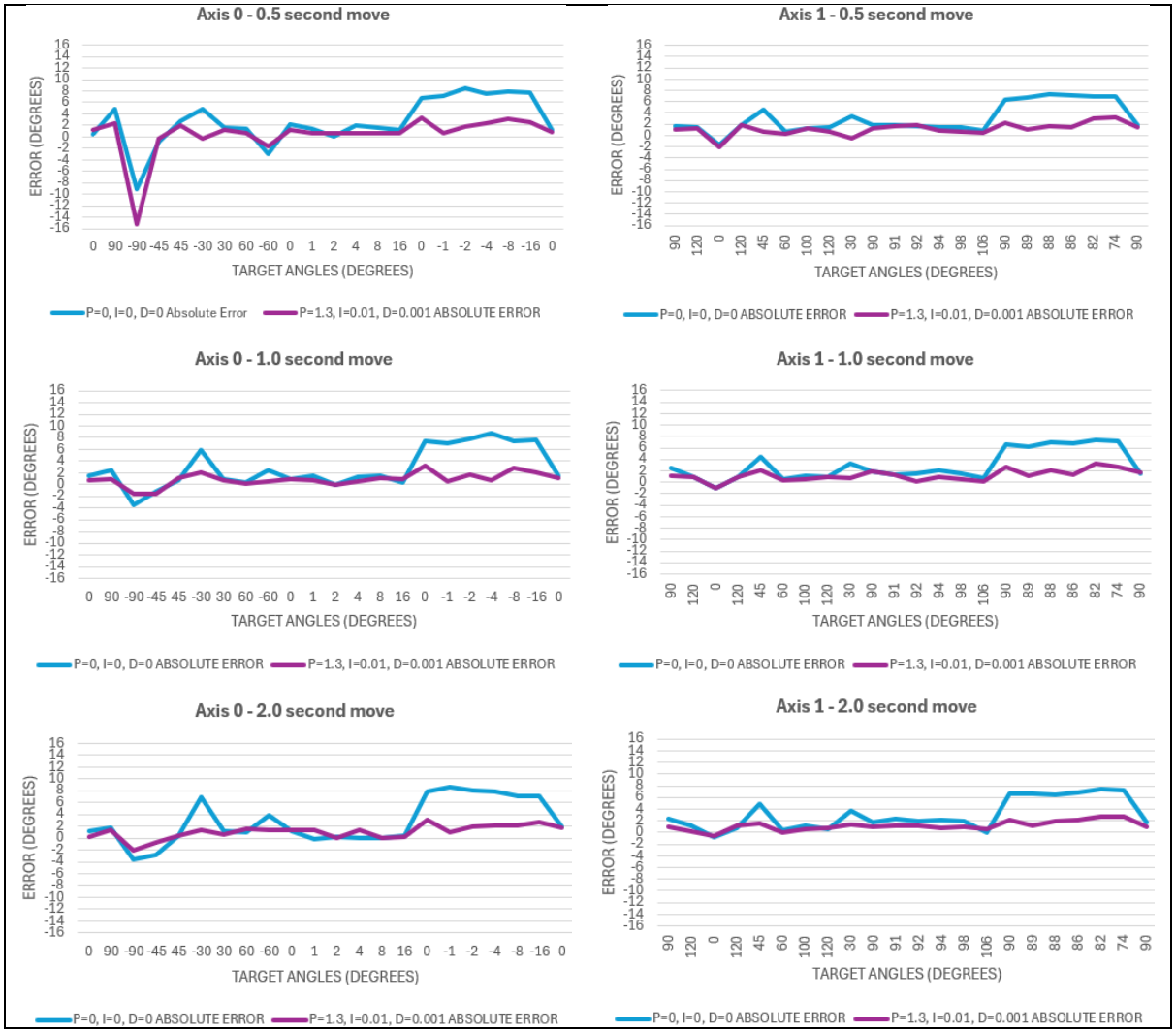


Figure 49: Open-loop vs PID tuned accuracy

Parameters P = 1.3, I = 0.01, and D = 0.001 were selected through trial and error, based on visually minimising vibration during arm movement. Figure 49 shows that the tuned PID significantly improves the consistency and reliability of reaching the desired joint angles, particularly for small movements, where the motor torque would otherwise be insufficient to overcome the friction at the joints.

3.2.7.2 Settling time effect on performance

While PID tuning improves accuracy, discrepancies remain, up to 4 degrees for small moves and as much as -16 degrees for the largest and fastest moves (-90 to +90 degrees swing in 0.5 seconds) as shown in Figure 49. This section evaluates the effectiveness of introducing a settling phase.

Using the PID values P = 1.3, I = 0.01, D = 0.001, automated tests are performed using the same script as in **Section 3.2.7.1** to measure the accuracy of Axis 0 and 1 for settling times of 0, 0.5, and 1 second, across move durations of 0.5, 1, and 2 seconds, following the same move sequences performed in the PID evaluation.

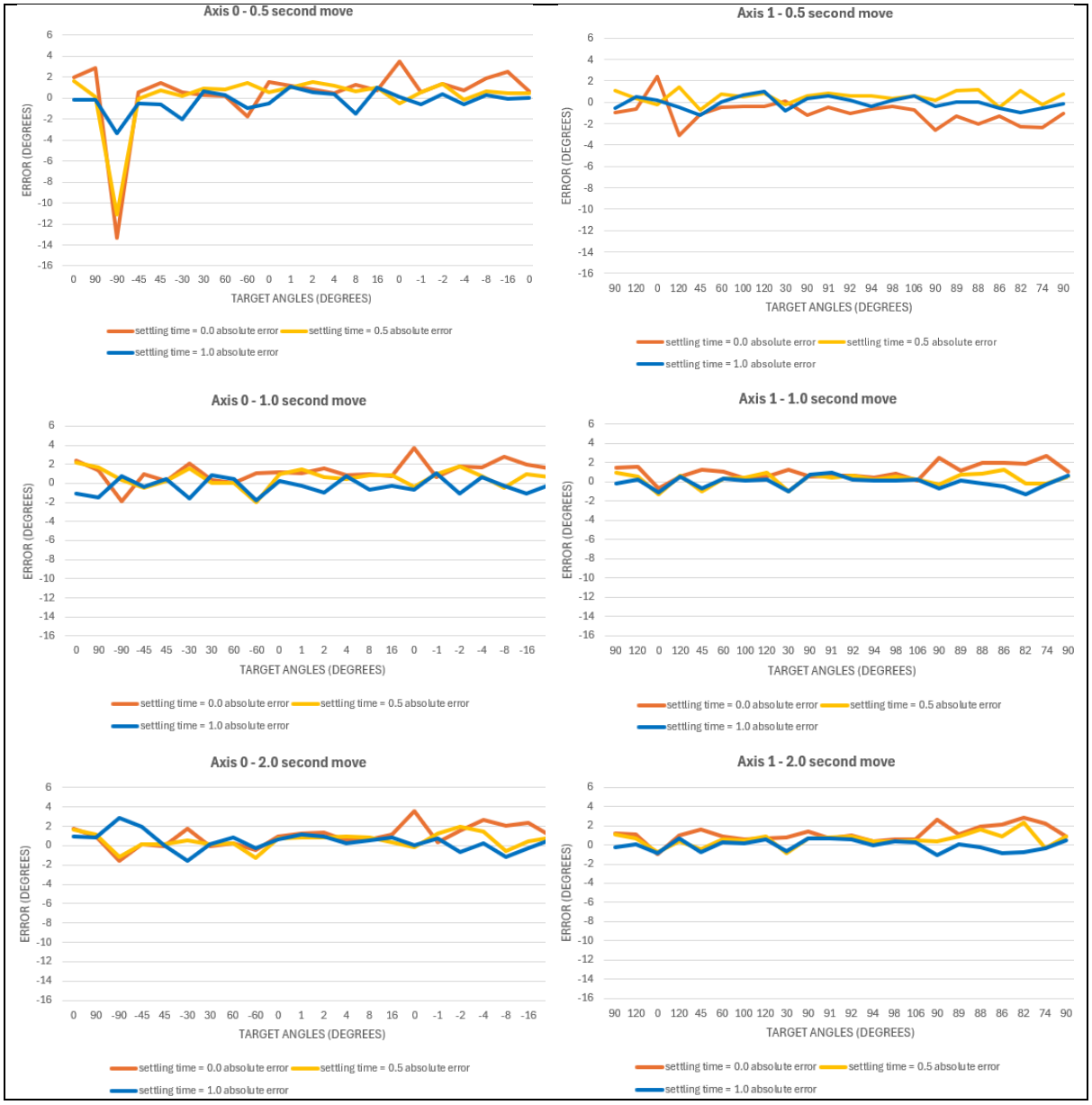


Figure 50: settling times of 0, 0.5, and 1 across move durations of 0.5, 1, and 2 seconds

The graphs show that a settling time of 1 second provides the most consistent accuracy out of the tested configurations, consistently maintaining error within 2 degrees. The biggest improvement is seen during the largest and fastest moves (−90 degrees to +90 degrees in 0.5 seconds), where the error drops from 13 without settling to just 4 degrees with a 1 second settling time. While a 0.5 second settling time does not significantly reduce error for the largest move, it achieves similar accuracy to the 1 second delay for less demanding moves.

3.2.7.3 Performance against other Educational Manipulators

As previously mentioned, there is a lack of reliable and publicly available performance data for educational SCARA. As a result, comparisons are based on actuator specifications from competing platforms under ideal conditions. Most competitors use stepper motors as actuators, so the EduSCARA manipulator will be compared against a theoretical 'ideal' SCARA robot with identical link lengths but equipped with stepper motors instead of servos. This ideal

system assumes no missed steps or slip. For this comparison, the widely used NEMA17 stepper motor is selected as a representative example, as it is commonly found in compact educational robots such as the NIRYO Ned2 [3]. A standard bipolar NEMA17 stepper motor [44] has a default step angle of 1.8 degrees. When paired with a high-performance stepper driver such as the TMC2208 [45] operating at 1/16 microstepping, it can achieve a theoretical resolution of 0.1125 degrees, resulting in negligible error when compared to the joint angle accuracy achieved by the EduSCARA manipulator. The following tests are automated using the script below, which utilises **SCARA_MOVE_COORD** and **SCARA_READ_COORD** commands. The measured coordinates are then plotted against the target trajectory, which also represents the performance of the theoretical SCARA system:

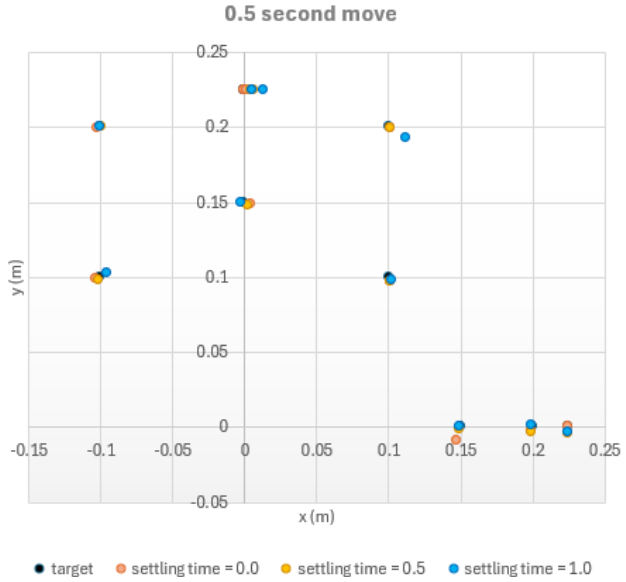
```
move_time = 1
settling_time = 1

scara.SCARA_INITIALISE(link_1, link_2, z_min, z_max,
                       settling_time,
                       P_0, I_0, D_0, P_1, I_1, D_1)
scara.SCARA_MOVE_JOINT(2, 0.0, 1.0)
scara.SCARA_MOVE_JOINT(3, -60, 1.0)
scara.SCARA_MOVE_JOINT(0, 0.0, 1.0)
scara.SCARA_MOVE_JOINT(1, 90.0, 1.0)
scara.SCARA_AUTO_CALIBRATE()

targets = [
    (0, 0.225),
    (0, 0.15),
    (-0.1, 0.1),
    (-0.1, 0.2),
    (0.1, 0.1),
    (0.1, 0.2),
    (0.2, 0),
    (0.15, 0),
    (0.225, 0),
    (0, 0.225)
]

with open('scara_move_coord_results.txt', 'w') as f:
    for target_x, target_y in targets:
        # move to the target
        scara.SCARA_MOVE_COORD(target_x, target_y, 0.0, 0.13, move_time)
        time.sleep(1.0)
        # read back actual x, y
        x, y, z_angle, z = scara.SCARA_READ_COORD()
        # Add the x, y to the line
        line = f"{x:.6f},{y:.6f}"

        f.write(line + "\n")
```



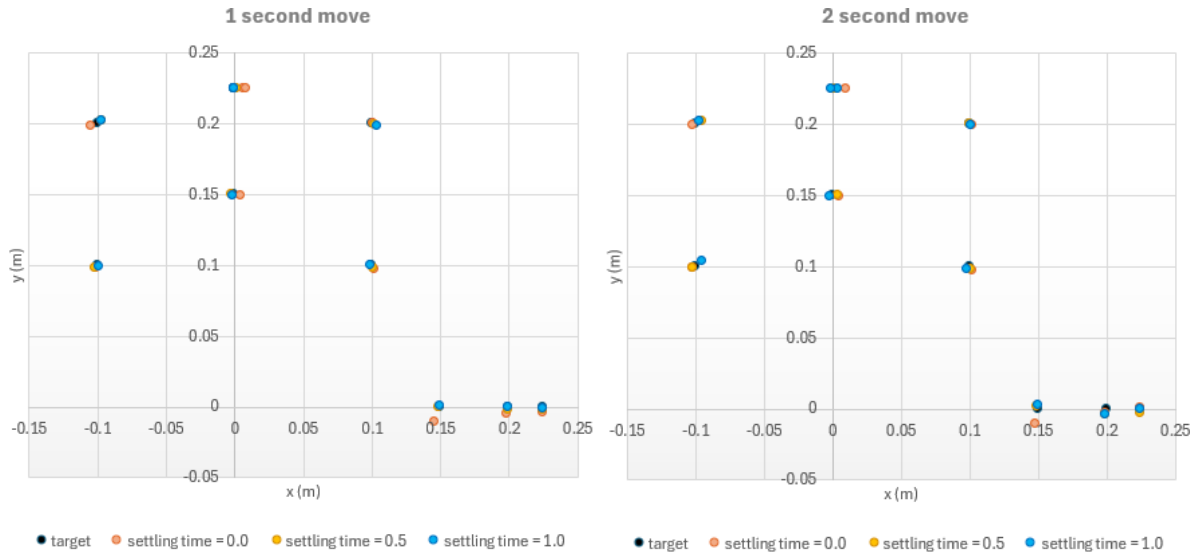


Figure 51: SCARA_MOVE_COORD performance against 'ideal' stepper SCARA (target)

As the theoretical stepper motor SCARA system has negligible positioning error compared to the EduSCARA, the 'target' coordinate is assumed to also represent the ideal performance of the stepper-based system. Despite implementing PID control and a settling phase the EduSCARA, due to mechanical backlash and the limitations of chosen servo motors, cannot match the precision of its stepper-driven counterpart.

This outcome is not surprising as the trade-offs between servo and stepper motors were discussed earlier. However, these tests provide a clearer understanding of the practical limitations of the EduSCARA manipulator in its current form, and offers valuable insight into its areas for future refinement.

3.3 The EduSCARA Simulator

The EduSCARA Simulator was developed to align with the typical robotics application development workflow outlined in: *"A Framework for Rapid Robotics Application Development for Citizen Developers"* [46]. This paper describes the common end-to-end processes involved in robotics development:

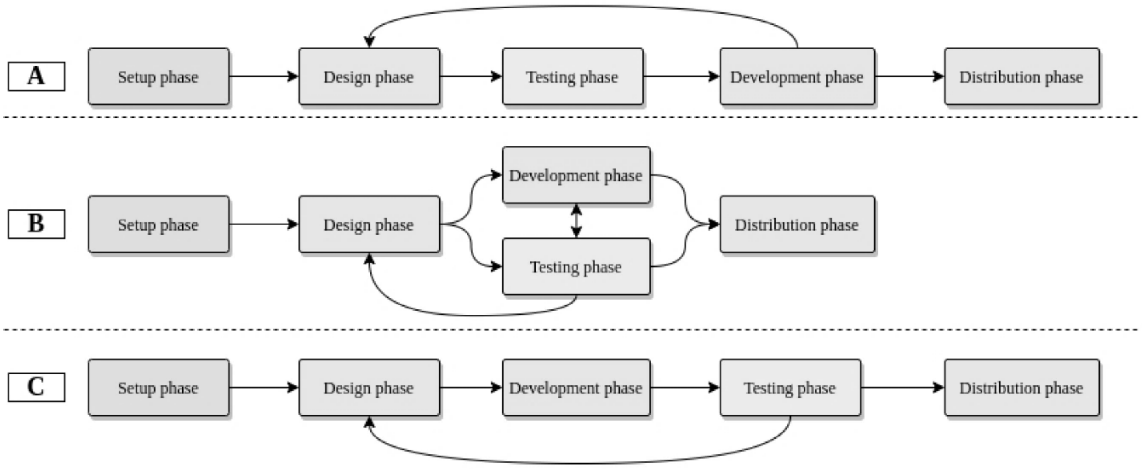


Figure 52: Common end-to-end development processes in robotics: Case A indicates a proactive method, where tests are performed before the actual implementation; Case B adopts the agile development model; Case C proposes initial development to precede testing [46].

Regardless of the development approach chosen, the EduSCARA Simulator is designed to reduce the burden of the *Setup Phase* and accelerate the *Design Phase* in the robotics development process. This is achieved by providing a Python API for the simulator that mirrors the structure and functionality of the API used for the physical motion controller, enabling seamless transition between simulation and real hardware, demonstrated in **Section 3.4.5**.

The EduSCARA Simulator features a built-in EduSCARA Motion Controller Emulator, eliminating the need for external hardware and enabling fully local development and testing. Communication between the simulator and the Python API is handled via sockets, with the simulator acting as a persistent server that automatically listens for incoming connections. If the API client disconnects, the simulator remains active and ready to reconnect, allowing for seamless development and testing. The diagram below illustrates the communication between the Simulator (server) and API application (client) interact through a socket-based connection:

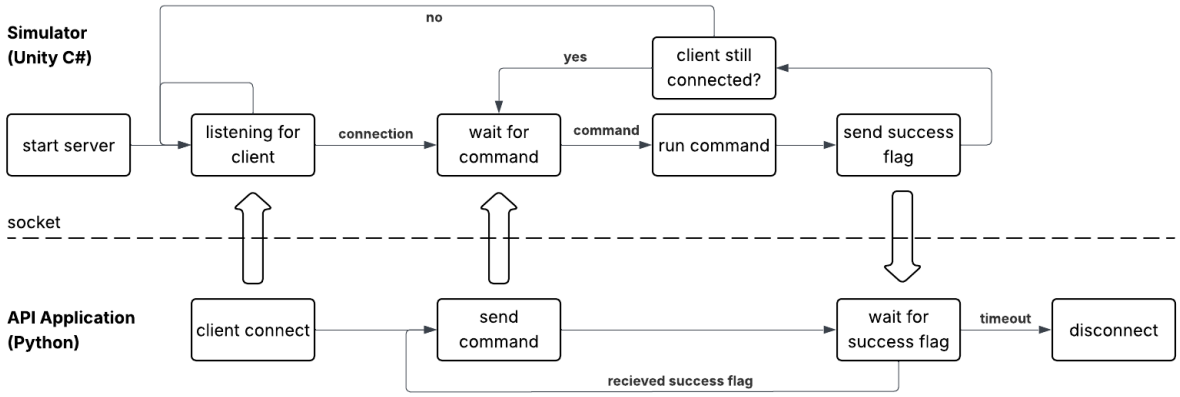


Figure 53: EduSCARA Simulator Operation

The EduSCARA Simulator includes several key features designed to make application development efficient and user-friendly to beginners. Real-time text displays show the end effector's position and rotation, as well as the joint angles, providing immediate feedback during testing. A visual representation of the robot's working area can be toggled, which helps users understand workspace constraints.

Intuitive camera controls using W (zoom in), A (left), S (zoom out), D (right), SPACE (up), and SHIFT (down) keys, hold RIGHT CLICK (to look around), mirror controls familiar from computer games, allowing users to freely navigate and closely inspect the robot during development. Additionally, the end effector includes a visual pointer directed toward the floor grid, providing clear feedback on its position relative to the workspace:

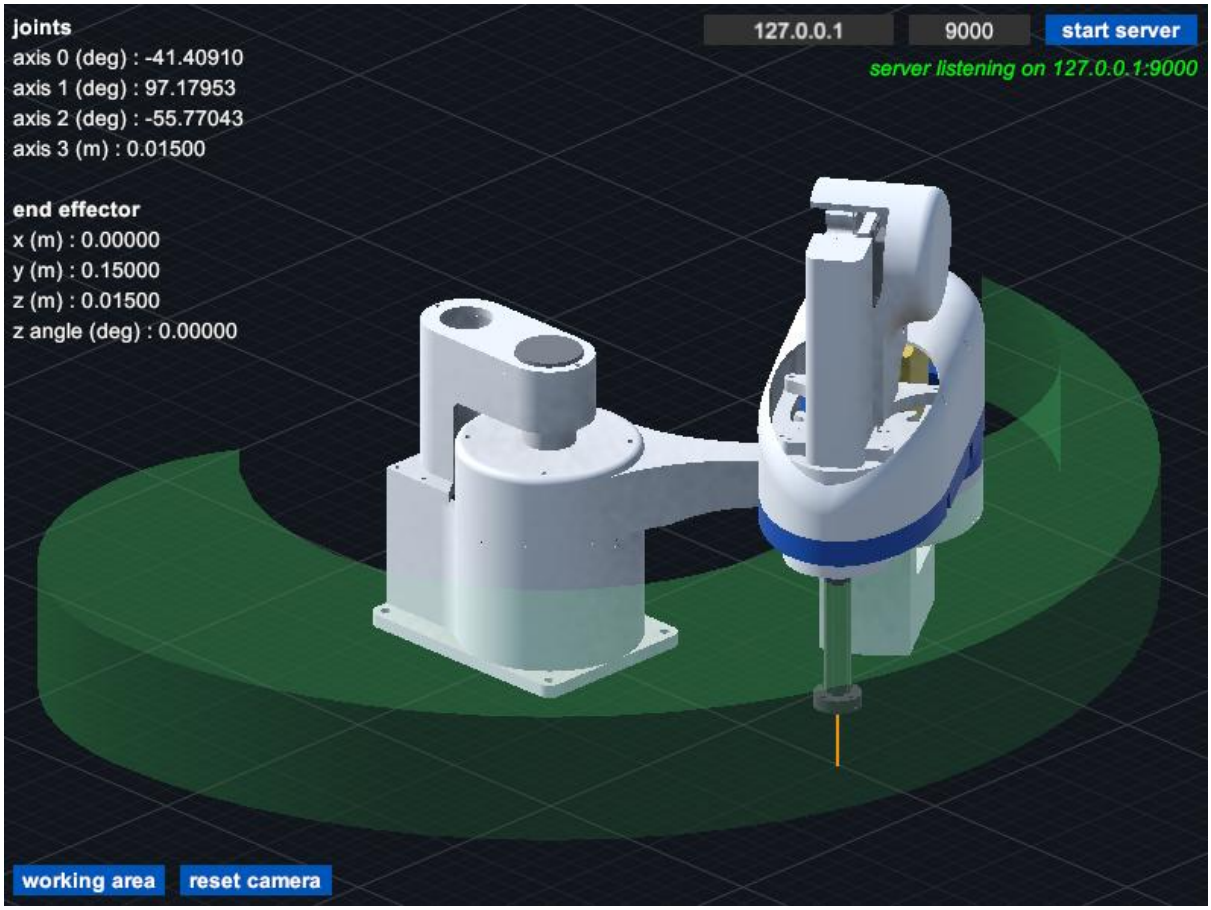


Figure 54: The EduSCARA Simulator

The EduSCARA Simulator Python API is stored in ***sim_scara_motion_controller_api.py***. Below is an example script that demonstrates how to use the API to move the Simulated SCARA through a sequence of specified coordinates:

```
import sim_scara_motion_controller_api as ssrc_api

sim_scara = ssrc_api.sim_scara_motion_controller(host='127.0.0.1', port=9000)

sim_scara.SCARA_MOVE_COORD(0.00, 0.225, 0, 0.02, 1)
sim_scara.SCARA_MOVE_COORD(0.00, 0.15, 0, 0.05, 1)
sim_scara.SCARA_MOVE_COORD(-0.10, 0.10, 0, 0.02, 1)
sim_scara.SCARA_MOVE_COORD(-0.10, 0.20, 0, 0.05, 1)
sim_scara.SCARA_MOVE_COORD(0.10, 0.10, 0, 0.02, 1)
```

3.4 E-waste Sorting Application Exercise for Teaching

While e-waste accounts for 70% of all toxic waste globally, only around 17% of e-waste is recycled [47]. E-waste generated from electronics facilities such as R&D labs, universities, and hobbyist workspaces contribute to this growing problem. Electrical components often become e-waste due to difficulty in identification, leading to the disposal of functional components [48]. While large-scale waste sorting robots exist, they are too expensive and impractical for use in smaller facilities [49]. A more compact, low power and cost-effective solution is required to enable efficient sorting and reuse of these components. This section demonstrates how the EduSCARA platform can address real-world problems by showcasing its effectiveness as both an educational tool and a functional robotic platform through the development of an **automated through-hole resistor-sorting system**.

This problem is not unique, as numerous academic projects, particularly from other university students, have attempted to solve the resistor sorting problem over the years. Several notable mechanisms are shown below. However, based on the reviewed literature, the existing solutions have not successfully achieved full automation of the resistor sorting process, nor have they implemented their systems in a low-cost, compact, lab-friendly form:

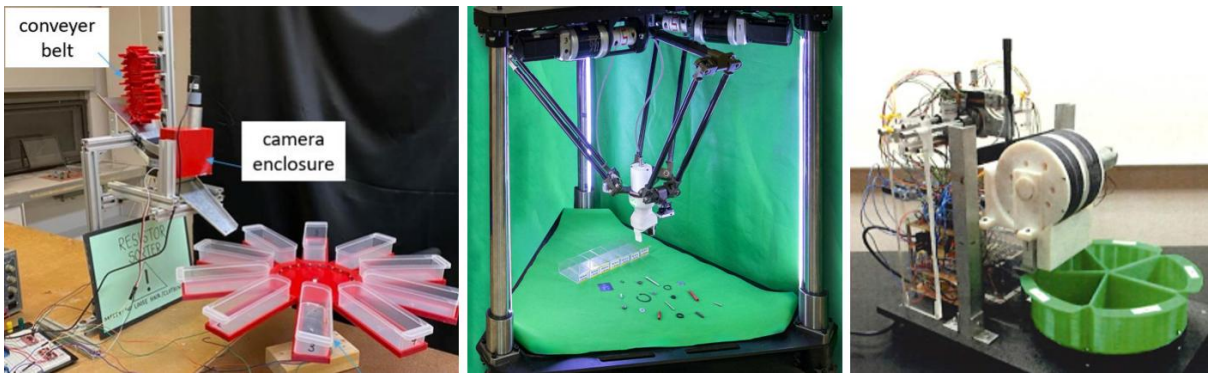


Figure 55: "ENPH 454 Final Report Resistor Sorter (2022)" (left) [30]
"Robotic Sorting of Mechanical and Electrical Parts" (2024) (middle) [29]
"Senior Design Project - Resistor Sorter" (2013) (right) [28]

This example application aims not only to solve a practical automation problem but also to demonstrate how the EduSCARA platform enables students to apply their skills effectively in a real-world context. The high-level system block diagram for this application is shown below:

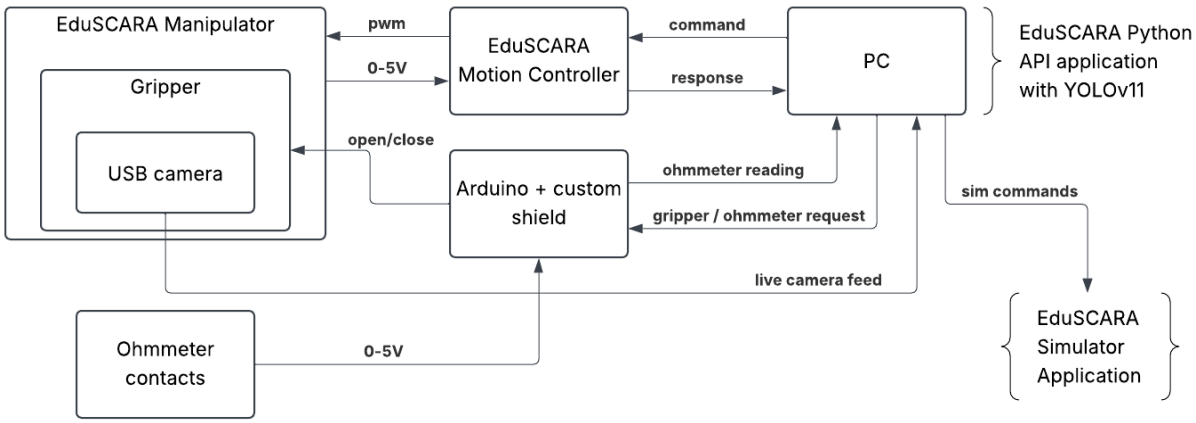


Figure 56: EduSCARA resistor sorting application with YOLOv11 object detection AI

The following sections provide a brief overview of the design and performance of each component within the system, as well as the overall functionality of the complete solution. Each section highlights the educational value of its respective subsystem and outlines the skills learners acquire through practical implementation.

3.4.1 The Gripper Mechanism and Camera Mount

The Gripper Mechanism was prioritised early in the design process, as without the ability to pick up the resistor, the rest of the system cannot function. Designing the gripper and camera mount offers learners hands-on experience with iterative mechanical design, applying CAD skills to a real-world system. It also exposes them to challenges like manufacturing tolerances and how to address them through refinement. The example below shows how successful design is achieved, though iterative prototyping using a single servo motor to actuate the gripper:

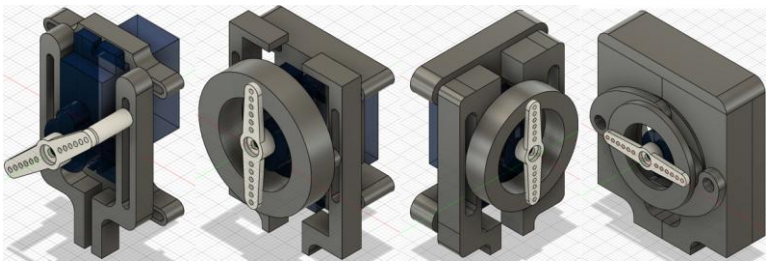


Figure 57: Servo gripper mechanism for resistor pick-and-place Prototype 1 (left) to 4 (right)

The final design example includes a camera housing mounted directly along the Z-axis, positioning the center of the lens beneath it. This allows the camera to rotate around the Z-axis without shifting laterally. The gripper is aligned to open and close directly beneath the camera, with enough clearance for the camera to see between the gripper jaws when open:

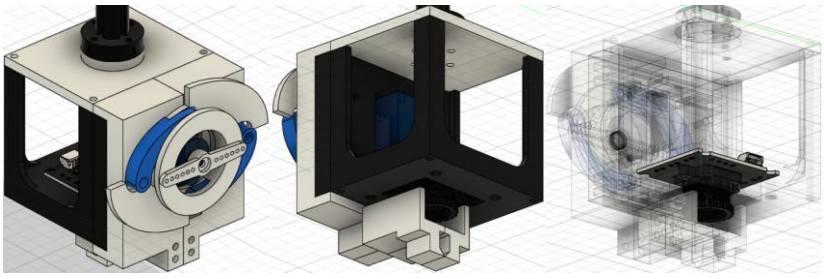
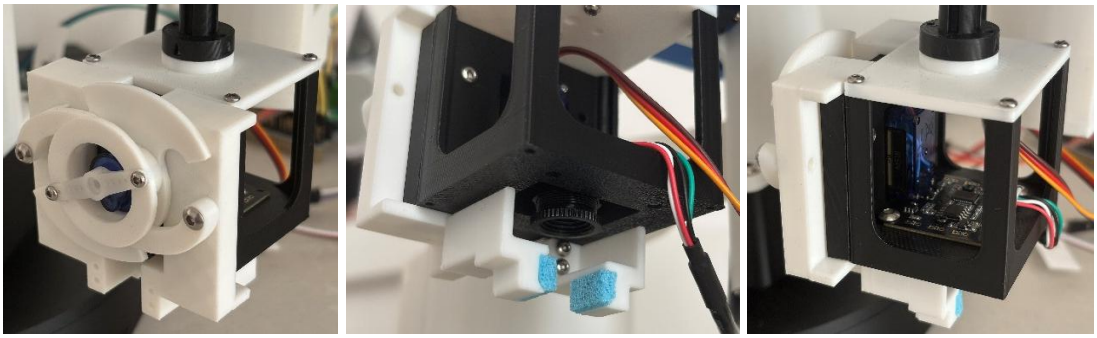


Figure 58: Servo Gripper Mechanism Prototype 5 + camera housing (CAD) – Final design



3.4.2 Arduino Gripper Controller + Ohmmeter

The Arduino in this system serves as an interface for both the gripper and the ohmmeter. To integrate with the EduSCARA Python application, students must develop a Python API for this, and use a communication protocol to exchange commands and data with the Arduino. But first, the hardware and firmware must be developed:

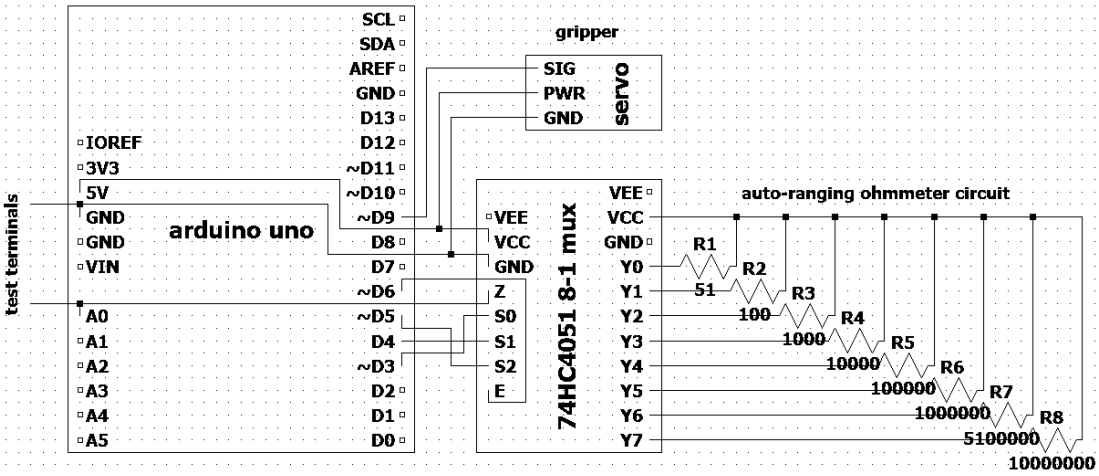


Figure 59: Arduino Gripper Controller + Ohmmeter schematic

The ohmmeter portion of the circuit is copied from "ArduinOhmmeter" by Programming Electronics Academy [50] – an auto-ranging ohmmeter using a potential divider configuration. It is straightforward and offers a valuable exercise where students interpret an existing schematic and write the corresponding firmware. The servo gripper is controlled via a PWM signal on pin 9. The firmware example for this subsystem shown below:

servo_gripper_ohmmeter.ino

```
#include "servo_gripper.h"
#include "ohmmeter.h"

const int SERVO_PIN = 9;
const int OPEN_PULSE = 2000;
const int CLOSE_PULSE = 1575;

servo_gripper gripper(SERVO_PIN, OPEN_PULSE, CLOSE_PULSE);

float resistance = -1.0;

void setup() {
    Serial.begin(9600);
    gripper.begin();
    initOhmmeter();
}

void loop() {
    if (Serial.available()) {
        String command = Serial.readStringUntil('\n');
        command.trim();

        if (command == "OPEN") {
            gripper.open();
            Serial.println("Gripper opened.");
        } else if (command == "CLOSE") {
            gripper.close();
            Serial.println("Gripper closed.");
        } else if (command == "IDLE") {
            gripper.idle();
            Serial.println("Gripper idle.");
        } else if (command == "TEST") {
            resistance = readResistance();
            Serial.println(resistance);
        } else {
            Serial.println("Unknown command.");
        }
    }
}
```

ohmmeter.h

```
#ifndef OHMMETER_H
#define OHMMETER_H

#include <Arduino.h>

#define NUM_REF_RESISTORS 8
#define NUM_SELECT_PINS   3
#define MAX_ANALOG_VALUE  1023
#define SWITCH_RESISTANCE 75

void initOhmmeter();
float readResistance();

#endif
```

servo_gripper.h

```
#ifndef SERVO_GRIPPER_H
#define SERVO_GRIPPER_H

#include <Arduino.h>
#include <Servo.h>

class servo_gripper {
public:
    servo_gripper(int servoPin, int openPulse, int closePulse);
    void open();
    void close();
    void idle();
    void begin();
private:
    Servo servo;
    int pin;
    int openPulseWidth;
    int closePulseWidth;
};
```

The example firmware listens for serial commands: “OPEN”, “CLOSE”, and “IDLE” for controlling the gripper, and “TEST” to trigger an ohmmeter reading, which is then transmitted back to the sender. The Python API script **servo_gripper_ohmmeter.py** handles communication with the device, providing a simple interface for sending commands and receiving responses:

```python
import serial
import time

class servo_gripper_ohmmeter:
    def __init__(self, port, baudrate=9600, timeout=None):
        self.api = serial.Serial(port, baudrate, timeout=timeout)
        time.sleep(2)
        pass

    def send_command(self, cmd: str):
        self.api.write((cmd + '\n').encode())
        response = self.api.readline().decode().strip()
        print(f"GRIPPER: {response}")
        return response

    def OPEN(self):
        self.send_command("OPEN")

    def CLOSE(self):
        self.send_command("CLOSE")

    def IDLE(self):
        self.send_command("IDLE")

    def TEST(self):
        self.send_command("TEST")
        return self.send command("TEST")
```

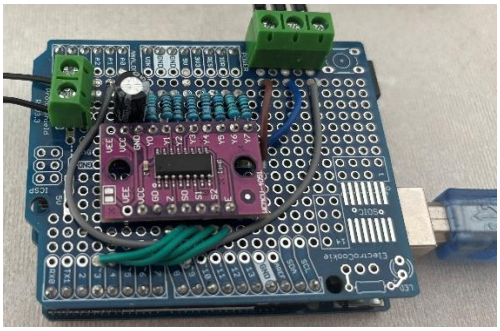


Figure 60: Arduino Gripper Controller + Ohmmeter

3.4.3 Performance Analysis of the Gripper Controller + Ohmmeter

The success of this application relies heavily on the gripper’s reliability and the ohmmeter’s accuracy. The gripper’s ability to hold resistors is mainly affected by wear and tear of the foam, not gripping strength, as resistors are very lightweight. The ohmmeter circuit’s accuracy was validated by comparing readings against a Fluke 15B+ multimeter:

Figure 61: Fluke 15B+ Digital Multimeter

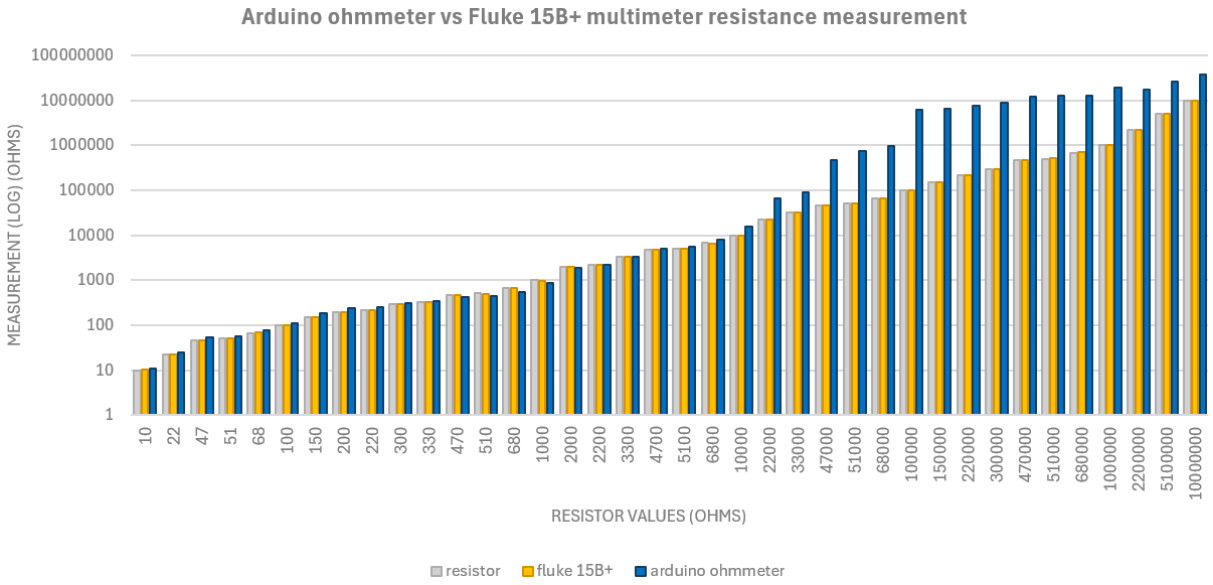


Figure 62: Arduino ohmmeter vs Fluke 15B+ resistance measurement comparison

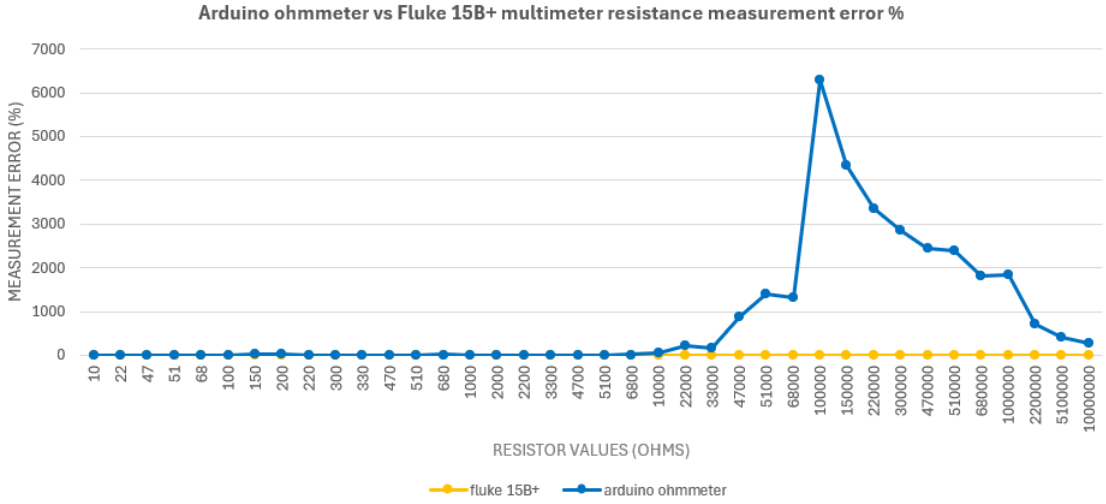


Figure 63: Arduino ohmmeter vs Fluke 15B+ resistance measurement error %

These results show that the Arduino ohmmeter becomes highly unreliable for resistance values above 10kΩ. Therefore, the application is limited to sorting resistors below this threshold.

3.4.4 YOLOv11 Resistor Detection Algorithm

The resistor detection algorithm is designed to identify any resistors within the camera's frame and determine their position and orientation relative to the camera. This information enables the Manipulator to adjust the end-effector until the resistor is centered in the frame, aligning it with the gripper for pickup. The detection uses the YOLO (You Only Look Once) algorithm (see **Section 2.3**) and the example is trained on the open-source "ECDetector Dataset" [51] (license: CC).

Students must train their own resistor detection model and develop a Python library to interface with it to get resistor coordinates and angles, gaining experience with dataset curation, real-world AI applications and state-of-the-art object detection frameworks.

The example dataset consists of 405 training images, 30 validation images, and 30 test images. The model was trained for 100 epochs, and the resulting weights are saved in a PyTorch file: ***resistor_detector_model.pt***. Figure 64 shows the training results, illustrating how the model improves over 100 epochs:

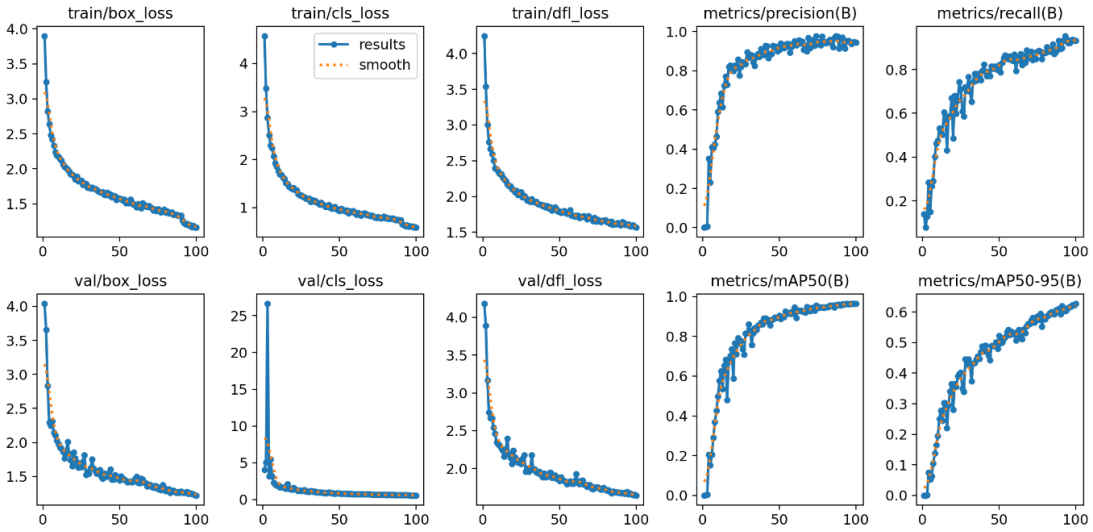


Figure 64: Training results: ECDetector Dataset – 100 epochs

To simplify application development using the resistor detection algorithm, the example resistor detector Python API was created with a find_resistors() function that returns the position and rotation of detected resistors:

```python
from ultralytics import YOLO
import cv2
import numpy as np
import math

class resistor_detector:
    def __init__(self, model_path="resistor_detector_model.pt", camera_index=0, ···

    def get_angle_and_draw_line(self, binary_img, x1, y1, x2, y2, frame, is_vertical): ···

    def find_resistors(self, draw=False): ···

    def release(self): ···
```

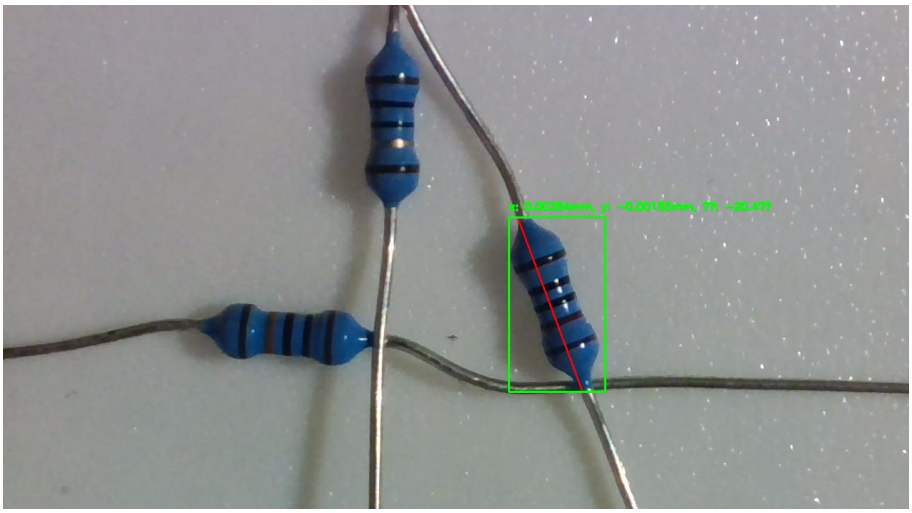


Figure 65: Resistor detector identifying 1 resistor at a time, returns position and angle

3.4.5 Simulating the Application using the EduSCARA Simulator

This section demonstrates how the EduSCARA Simulator is used to develop the move sequences for this application. The CAD model for the workspace and gripper are imported into the EduSCARA Simulator Unity project, and the gripper is attached to the Z axis:

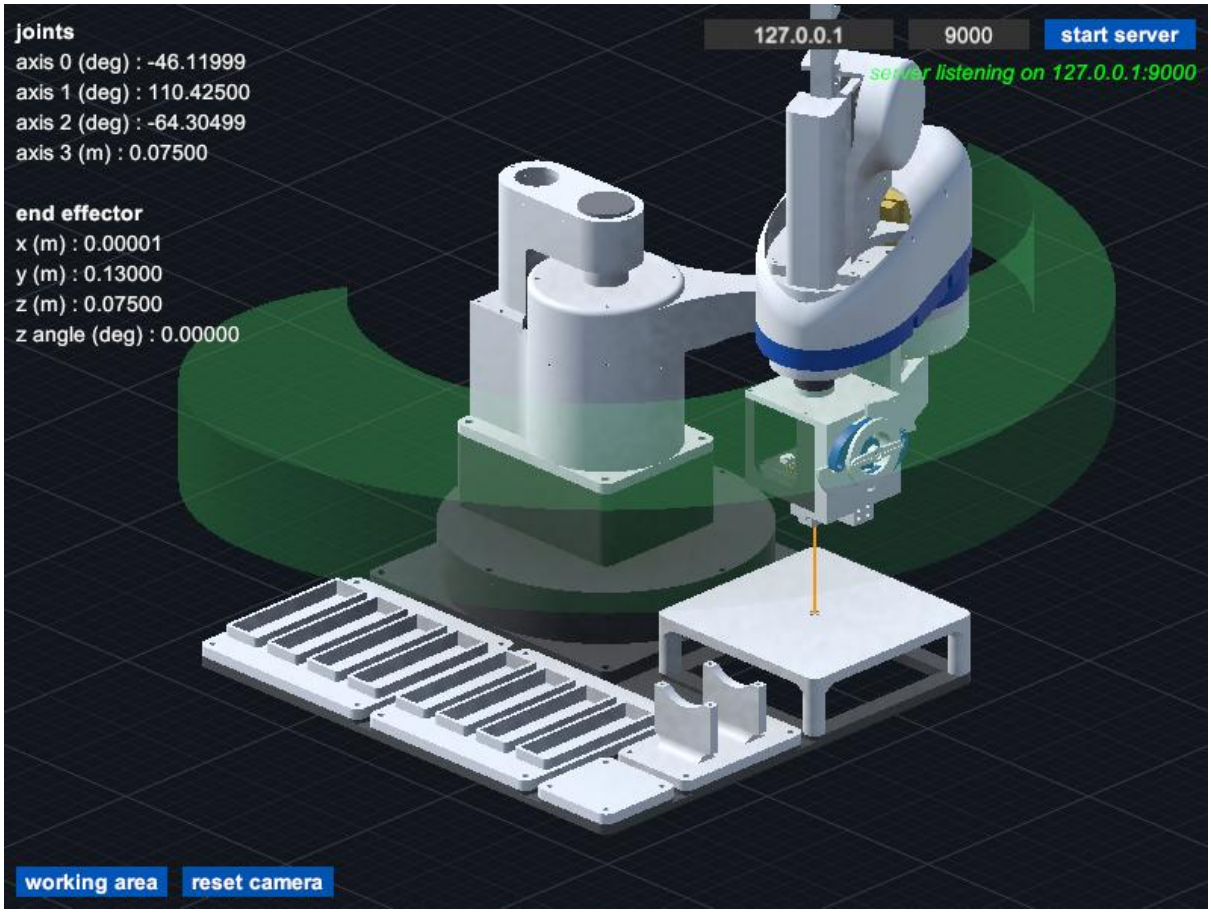
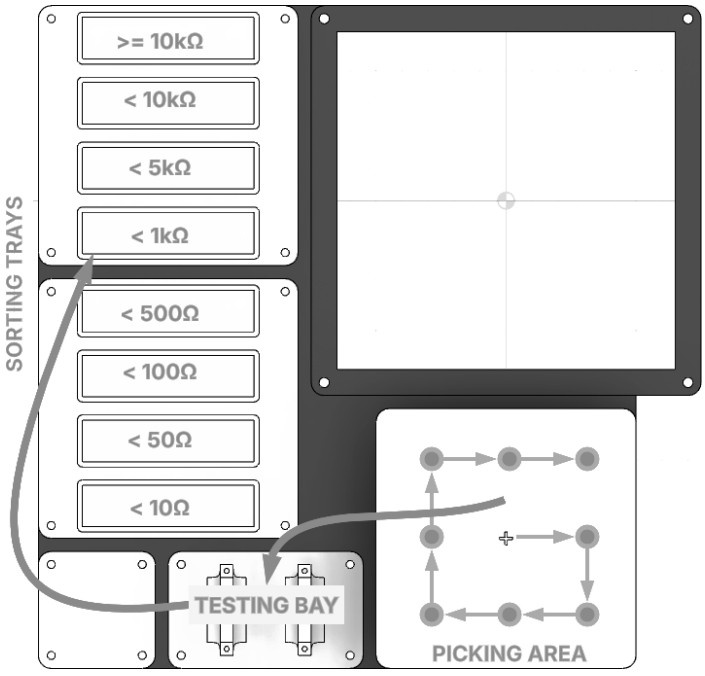


Figure 66: Resistor sorting application in the EduSCARA Simulator



Resistors are placed on the 'picking area' platform, and the gripper opens to allow the camera to see through. Due to the camera's limited field of view even when the Z-axis is fully retracted, the platform is scanned by moving through nine predefined positions. If a resistor is detected, it is picked it up and pushed down on to the metal contacts in the 'testing bay'. Based on the measured resistance, it is sorted into appropriate trays.

For this demonstration, values are sorted by range due to the limited space, and the ohmmeter only being reliable up to 10kΩ measurements.

The example application ***sim_resistor_application.py*** is successful in simulation:

```
import sim_scara_motion_controller_api as ssrc_api
import time
import random

scara = ssrc_api.sim_scara_motion_controller(host='127.0.0.1', port=9000)

tool_offset = 0.067
# picking tray scanning coordinates
starting_x = [0.000, -0.020, -0.020, 0.000, 0.020, 0.020, 0.020, 0.000, -0.020]
starting_y = [0.130,  0.130,  0.150, 0.150, 0.150, 0.130, 0.110, 0.110,  0.110]
starting_z = 0.022 + tool_offset
picking_z = -0.036 + tool_offset
# location and orientation of testing point [x, y, z_angle, z]
testing_z = -0.036 + tool_offset
testing_point_coord = [0.0925, 0.1575, 90.0, testing_z]
# location and orientation of sorting trays [x, y, z_angle, z]
sorting_y_10 = 0.1175
sorting_y_50 = 0.0925
sorting_y_100 = 0.0675
sorting_y_500 = 0.0425
sorting_y_1000 = 0.0125
sorting_y_5000 = -0.0125
sorting_y_10000 = -0.0425
sorting_y_other = -0.0675
sorting_y = [sorting_y_10, sorting_y_50, sorting_y_100, sorting_y_500,
             sorting_y_1000, sorting_y_5000, sorting_y_10000, sorting_y_other]
sorting_point_coord = [0.130, sorting_y[7], 90.0, picking_z] # default to 'other'
while True:
    for i in range(9):

        # ==================== PICKING ====================
        # move arm above picking tray
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, starting_z, 0.5)
        # open gripper
        # <- ->

        scara.SCARA_MOVE_COORD
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, starting_z, 0.5)
        # move arm down
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, picking_z, 1.0)
        time.sleep(0.5)
        # close gripper
        # -> <-
        time.sleep(0.5)
        # move arm up
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0, starting_z, 0.5)

        # ==================== TESTING ====================
        # move arm above tester
        scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], starting_z, 0.5)
        # move arm down
        scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], testing_point_coord[3], 0.5)
        # wait for resistor test result
        time.sleep(0.5)
        # read resistor value
        # move arm up
        scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], starting_z, 0.5)

        # ==================== SORTING ====================
        # simulate unknown resistor value
        n = random.randint(0, 7)
        print(f"random tray: {n}")
        # go to dispose tray
        scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], 0.0, starting_z, 0.5)
        # move arm down
        scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], sorting_point_coord[2], sorting_point_coord[3], 0.5)
        time.sleep(0.5)
        # open gripper
        # <- ->
        time.sleep(0.5)
        # move arm up
        scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], 0.0, starting_z, 0.5)
        # gripper idle
        # -| |-
        # move back to picking tray
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, starting_z, 0.5)

    again = int(input("again? "))
    if again == 0:
        break
```

3.4.6 Transitioning from Simulation to Real-World Application

The example application is now fully functional in simulation, enabling the transition to deployment on the physical system. The first step involves building the physical rig, followed by verifying that the end effector positions accurately align with the intended target locations. This step is critical, as 3D-printed components may deviate from the exact dimensions in the CAD models:

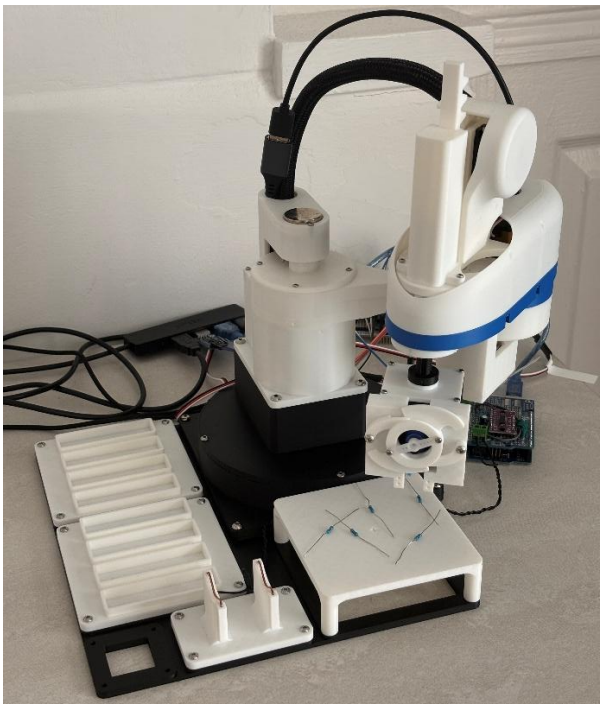


Figure 67: EduSCARA resistor sorting application rig

One way to verify this is by writing a script that continuously displays the live camera feed alongside the current end-effector position using the **SCARA_READ_COORD** command. Joints are then manually positioned to the desired locations with the aid of the camera feed aligned with the Z axis, and the corresponding coordinates are recorded on the diagram below:

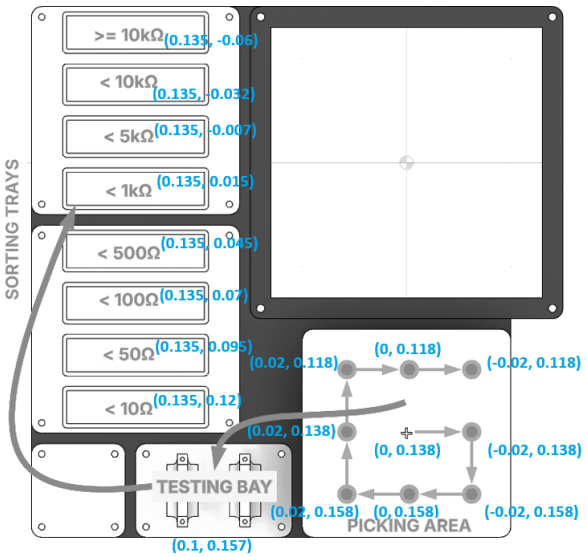


Figure 68: Real-world locations of target positions on the resistor sorter workspace

With aid from the simulator application ***sim_resistor_application.py***, which provides a general outline of the system operation, this script can be easily adapted resulting in the completed resistor sorting application described by the flow diagram:

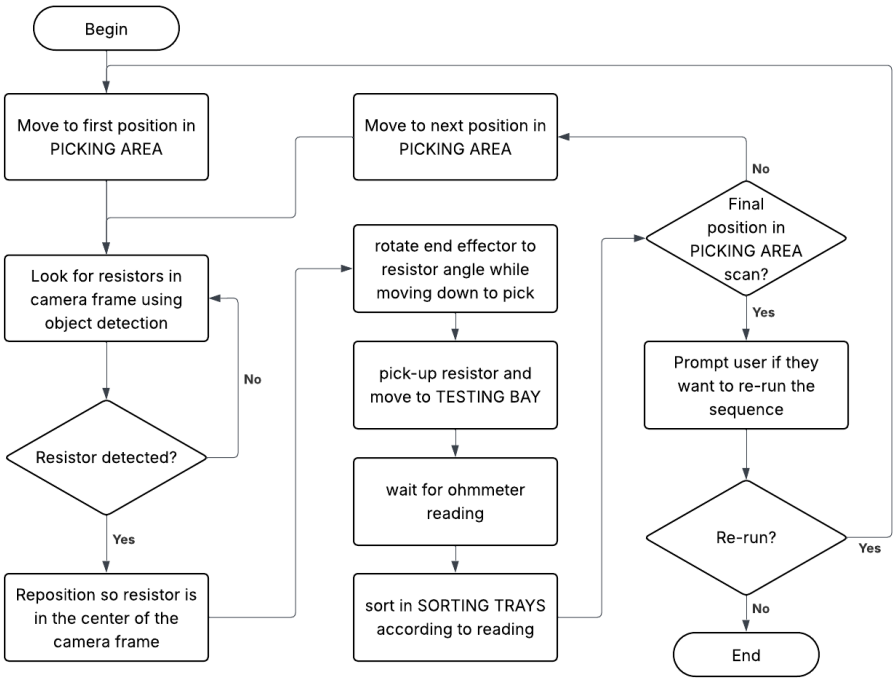


Figure 69: Resistor sorting application flow diagram

The completed, successful resistor sorting application: ***resistor_application.py*** is shown below:

```
import scara_motion_controller_api as smc_api
import servo_gripper_ohmmeter_api as sgo_api
import gripper_camera_api as gca_api

import serial.tools.list_ports
import time
import cv2

ports = list(serial.tools.list_ports.comports())
if not ports:
    print("No USB device found.")
else:
    port_list = [f"{port.device} - {port.description}" for port in ports]
    print(f"Available Ports:\n" + "\n".join(port_list))

    scara_port = input("\nEnter the COM port for SCARA (e.g., COM3):")
    scara = smc_api.scara_motion_controller(port=scara_port)

    gripper_port = input("\nEnter the COM port for Gripper (e.g., COM3):")
    gripper = sgo_api.servo_gripper_ohmmeter(port=gripper_port)

    resistor_detector = gca_api.resistor_detector(screen_width_m=0.035, screen_height_m=0.02)

    gripper.IDLE()

    link_1 = 0.125
    link_2 = 0.1
    z_min = 0.095
    z_max = 0.15
    settling_time = 0.5
    P_0 = 1.3
    I_0 = 0.01
    D_0 = 0.001
    P_1 = 1.3
    I_1 = 0.01
    D_1 = 0.001

    scara.SCARA_INITIALISE(link_1, link_2, z_min, z_max,
                           settling_time,
                           P_0, I_0, D_0, P_1, I_1, D_1)
    scara.SCARA_MOVE_JOINT(2, 0.0, 1.0)
    scara.SCARA_MOVE_JOINT(3, -60, 1.0)
    scara.SCARA_MOVE_JOINT(0, 0.0, 1.0)
    scara.SCARA_MOVE_JOINT(1, 90.0, 1.0)
    scara.SCARA_AUTO_CALIBRATE()
```

```
# picking tray scanning coordinates
starting_x = [0.000, -0.020, -0.020, 0.000, 0.020, 0.020, 0.020, 0.000, -0.020]
starting_y = [0.138,  0.138,  0.158, 0.158, 0.158, 0.138, 0.118, 0.118,  0.118]
starting_z = 0.15
picking_z = 0.107
# location and orientation of testing point [x, y, z_angle, z]
testing_z = 0.109
testing_point_coord = [0.092, 0.157, 90.0, testing_z]
# location and orientation of sorting trays [x, y, z_angle, z]
sorting_y_10 = 0.12
sorting_y_50 = 0.095
sorting_y_100 = 0.07
sorting_y_500 = 0.045
sorting_y_1000 = 0.015
sorting_y_5000 = -0.007
sorting_y_10000 = -0.032
sorting_y_other = -0.06
sorting_y = [sorting_y_10, sorting_y_50, sorting_y_100, sorting_y_500,
             sorting_y_1000, sorting_y_5000, sorting_y_10000, sorting_y_other]
sorting_point_coord = [0.135, sorting_y[7], 90.0, picking_z] # default to 'other'

while True:
    for i in range(9):
        # ==================== PICKING ====================
        # move arm above picking tray
        scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, starting_z, 0.5)
        # open gripper
        gripper.OPEN()
        # <- ->

        # --------------- LOOK FOR RESISTOR ---------------
        # repeatedly move to center the resistor
        max_attempts = 10
        attempts = 0
        current_x, current_y, _, _ = scara.SCARA_READ_COORD()

        while attempts < max_attempts:
            pos = resistor_detector.find_resistors(draw=True)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                cv2.destroyAllWindows()
                break

            if pos:
                dx, dy, zangle = pos
                distance = (dx**2 + dy**2)**0.5
                print(f"[attempt {attempts+1}] resistor offset: dx={dx:.5f} m, dy={dy:.5f} m, dist={distance:.5f} m")
                if distance < 0.003:
                    print("resistor is centered!")
                    break
                P = 0.5
                # move towards the resistor
                current_x, current_y, _, _ = scara.SCARA_READ_COORD()
                current_x += dx * P
                current_y += dy * P

                scara.SCARA_MOVE_COORD(current_x, current_y, 0.0, 0.15, 0.1)
                time.sleep(0.5)

            else:
                print("no resistor detected.")

            attempts += 1
        # -------------------------------------------------

        # if resistor detected
        if pos:
            # move arm down
            scara.SCARA_MOVE_COORD(current_x, current_y, zangle, picking_z, 0.5)
            # close gripper
            gripper.CLOSE()
            # -> <-
            time.sleep(0.5)
            # move arm up
            scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0, starting_z, 0.5)

            # ==================== TESTING ====================
            # move arm above tester
            scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], starting_z, 0.5)
            # move arm down
            scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], testing_point_coord[3], 0.5)
            # wait for resistor test result
            time.sleep(1.0)
            # read resistor value
            resistance = float(gripper.TEST())
            # move arm up
            scara.SCARA_MOVE_COORD(testing_point_coord[0], testing_point_coord[1], testing_point_coord[2], starting_z, 0.5)
            # ==================== SORTING ====================
            # sort based on resistor value
            if resistance >= 10000:
                n = 7
            elif resistance < 10000 and resistance >= 5000:
                n = 6
            elif resistance < 5000 and resistance >= 1000:
                n = 5
```

```
            elif resistance < 1000 and resistance >= 500:
                n = 4
            elif resistance < 500 and resistance >= 100:
                n = 3
            elif resistance < 100 and resistance >= 50:
                n = 2
            elif resistance < 50 and resistance >= 10:
                n = 1
            else:  # resistance < 10
                n = 0
            # go to dispose tray
            scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], 0.0, starting_z, 0.5)
            # move arm down
            scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], sorting_point_coord[2], sorting_point_coord[3], 0.5)
            time.sleep(0.5)
            # open gripper
            gripper.OPEN()
            # <- ->
            time.sleep(0.5)
            # close gripper
            gripper.CLOSE()
            # -> <-
            # move arm up
            scara.SCARA_MOVE_COORD(sorting_point_coord[0], sorting_y[n], 0.0, starting_z, 0.5)
            # gripper idle
            gripper.IDLE()
            # -| |-
            # move back to picking tray
            scara.SCARA_MOVE_COORD(starting_x[i], starting_y[i], 0.0, starting_z, 0.5)
        else:
            print("resistor not found at position")

    again = int(input("again? "))
    if again == 0:
        break
```

To conclude this section, this resistor sorting implementation successfully automates the entire process, with the only required human intervention being placing resistors into the picking area. Once a resistor is detected, the system completes testing and sorting within 12 seconds, allowing for a theoretical sorting rate of up to five resistors per minute. The system performs reliably under normal conditions, achieving a success rate of over 90%. Most failures occur when multiple resistors are stacked and picked simultaneously, or when a resistor is excessively bent, preventing proper contact with the testing bay and resulting in incorrect readings. Outside of these edge cases, the sorting process is consistent. The primary limiting factor in long-term performance is the wear and tear on the gripper's sponge contacts.

Nonetheless, under proper operating conditions, this example application demonstrates efficient autonomous resistor sorting, and highlights how the EduSCARA platform can be integrated with other systems to create a solution to real-world problems while providing strong educational value. Through completing this project, students apply and develop a wide range of practical engineering skills, including:

- CAD and prototyping for designing and refining functional components such as the gripper and camera mount.
- Embedded programming and API design to for communication between the Arduino subsystem and host PC.
- Machine learning skills including dataset curation and training a resistor detection model using a state-of-the-art object detection framework.
- Simulation-based development for safe path planning and motion sequence testing before real-world deployment.
- Understanding real-world non-idealities comparing simulated vs. actual performance, and adapting the application accordingly.
- Full-stack robotics development, covering mechanical design, simulation, firmware, software, and system integration to solve a real-world problem.

4 Discussion and Conclusions

4.1 Evaluation of Project Outcomes

The EduSCARA platform successfully meets its goal of delivering a functional, affordable, and industry-relevant tool for practical robotics education. It includes a working manipulator, an open-source STM32 motion controller, a Python API and a Unity-based simulator, forming a workflow that supports development in both virtual and real environments.

Unlike open-loop alternatives such as the "How to Mechatronics" [13] and "ROBCO" [14] SCARA, the EduSCARA platform features closed-loop control, enabling students to explore sensor calibration and PID tuning, which are skills vital in industrial robotics. The platform's low cost and open design make it accessible while maintaining strong educational value. The total cost to build a single EduSCARA unit is broken down as follows:

PART	DESCRIPTION	QTY	UNIT COST £	SUB TOTAL £
NUCLEO-F446RE	Microcontroller prototyping board	1	14.48	14.48
Arduino shield	Arduino Uno R3 prototyping shield	1	1.88	1.88
AGFRC B53BHS	24kgcm programmable servo	2	39.99	79.98
AGFRC B11DLS	9g metal gear micro servo	2	18.49	36.98
RV24YN20F B502	Single circle potentiometer, 5k	2	4.34	8.68
PLA filament	3D printing filament	<500g	13.99/kg	6.99
Other	Passive components, screw terminals			5.00
			TOTAL	**153.99**

Figure 70: Cost breakdown of one EduSCARA unit

This is demonstrated by the autonomous resistor sorting application example capable of sorting up to five resistors per minute, which leads students through the full development lifecycle using the EduSCARA platform. By completing this exercise, learners gain practical experience in CAD design, iterative prototyping, embedded firmware development, Python API integration and machine learning – providing a hands-on interdisciplinary learning experience that reflect real-world robotics engineering challenges.

4.2 Uncertainties and Limitations of Current work

Despite the successful delivery of a working prototype, several sources of uncertainty and design limitations remain. These are inherent in the choice of components, development constraints and educational trade-offs The selected servos, particularly the high-torque models suffer from significant backlash, resulting in lost motion. This can cause positional errors of up to 14 degrees for fast moves with high momentum even with closed-loop control:

Figure 71: Axis 0, 0.5 second moves, PID tuned, changing settling times (3.2.7.2)

Although backlash compensation using a settling period improves the end point accuracy, it introduces undesirable pauses between moves. While these trade-offs make the system usable for applications with lower precision demands, achieving higher accuracy would require selecting more precise motors.

The mechanical damping method at the manipulator joints uses friction to reduce vibration. while effective for damping, it negatively impacts repeatability, again, due to servo backlash. Depending on movement speed and momentum, joints may unpredictably overshoot or undershoot. Additionally, during small, slow moves, the servo may lack sufficient torque to overcome friction, resulting in significant error highlighted in Figure 72:

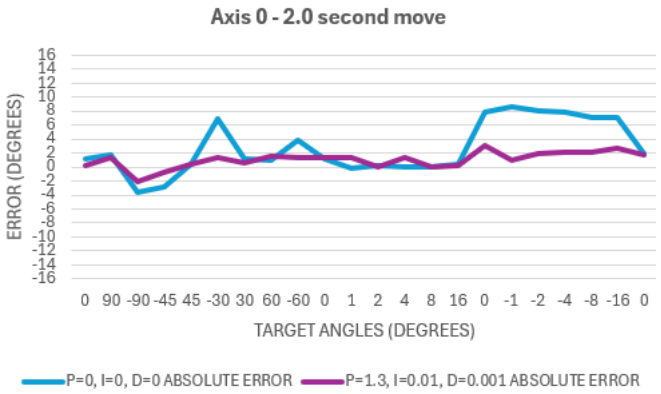


Figure 72: Axis 0, 2 second moves, Open-loop vs PID, no settling phase (3.2.7.1)

This issue stems from the combination of servo backlash and having to compensate for it using this damping method. Selecting different motors and implementing a suitable damping method to accommodate could significantly reduce lost motion.

The motion controller is currently not programmable, which limits its educational relevance to industrial counterparts. At present, all motion commands are issued via the API, with the controller functioning primarily as an interface rather than an autonomous computer. This diverges from typical industrial workflows, where motion controllers often handle program logic by themselves. Although this feature was deprioritised due to time constraints, enabling onboard programmability remains a critical enhancement for future iterations.

4.3 Recommendations for Future work

Future development of the EduSCARA platform presents multiple opportunities to enhance both its technical performance and educational value. Recommendations are therefore be grouped into these two key areas:

Enhancing Manipulator Control and Performance through improving accuracy, speed and robustness of the physical system:

- **Replace the servos with stepper motors:** Stepper motors offer significantly higher precision and eliminates backlash due to their gearless design. However, they must be used in closed-loop configurations to preserve the platform's educational value. Stepper motors require additional stepper drivers, increasing cost and wiring complexity. They also generate significant heat, posing a risk of melting PLA components, making

thermal insulation essential. Their added weight may require structural reinforcement, and control logic must also be updated in the SCARA Motion Coordinator Stack to support stepper operation.

- **Upgrade joint feedback sensors**: Potentiometers, while very straightforward to implement, introduces noise that limits accuracy and causes control instability. Incremental encoders provide cleaner signals due to their binary nature, but require homing switches and a homing routine on startup. Although absolute encoders offer the best performance, they are expensive. Using steppers with integrated encoders could be viable middle ground. However, the high precision open-loop performance of the stepper motors may obscure the effects of control algorithms, making it harder for learners to observe and understand system behaviour and therefore hurting its educational value, though it would enable the platform to support more advanced, high-accuracy applications.
- **Improve joint mechanics**: Friction-based damping was initially used to counteract servo backlash. If backlash-free stepper motors are used, replacing friction interfaces with ball bearings can enhance motion smoothness and reduce the required starting torque, making the manipulator more robust.

Increasing Educational Value to strengthen learning outcomes and industrial relevance:

- **Enable on-device programmability**: Allowing users to upload autonomous programs to the EduSCARA Motion Controller would align with industrial workflows.
- **Add I/O functionality**: Integrating digital, analog I/O ports or even PWM ports directly on the EduSCARA Motion Controller would reduce the wiring to the PC, and enable more realistic architectures
- **Expand API language support**: Supporting C++ and MATLAB would cater to diverse academic and research environments.
- **Integrate ROS compatibility**: Adding support for ROS would align the system with an industry-standard framework, enabling another method for users to develop modular, scalable robotic applications.
- **Develop a teach pendant**: A physical or virtual teach pendant allows users to jog the robot instead of having to manually move it to read coordinates, representative of industrial workflows.
- **Implement additional motion commands**: Commands such as MOVE_L (linear move) are essential for precise path planning, particularly in conveyor-based applications where the robot must stay synchronised with the conveyor to maintain a straight-line trajectory relative to moving objects.
- **Design a custom PCB**: Creating a PCB shield for the NUCLEO-32 board would reduce wiring complexity, improve reliability and allow for protective housing, improving both usability and presentation.
- **Create additional educational exercises**: Developing additional educational exercises will strengthen the platform's value as a teaching tool for industrial robotics, particularly through tasks involving other automation components such as conveyors and pneumatics, which can also be implemented in scaled versions using affordable components.

These improvements would make EduSCARA more capable, realistic, and useful for hands-on robotics learning. By refining the hardware, expanding software features and better reflecting

real industrial systems, the platform could support more advanced tasks while staying accessible. Continued development in these areas would help EduSCARA become a more complete tool for robotics education.

4.4 Project Review

This project began as a student-initiated effort aimed at sorting resistors as a contribution to solving the e-waste problem. However, after reviewing the literature, it is clear that although not fully addressed, this problem is not unique, and there are numerous papers that describe attempts to solve this problem over the years. Following feedback from my supervisor, Dr Tareq Assaf, it became clear that the robot's capabilities were being underutilised, and the project was re-scoped to focus on creating an effective teaching platform for practical robotics education.

This shift in direction, though ultimately beneficial, came near mid-project and required revisiting earlier decisions and designs. Had a thorough literature review been undertaken and the scope defined in consultation with the project supervisor and potential stakeholders (such as robotics educators) from the start, the project could have progressed more efficiently and achieved greater technical depth.

Most technical challenges were manageable, as the core engineering concepts involved are nothing new. The greater difficulty was in balancing budget and time constraints. A key takeaway was the importance of component selection; for example, the high-torque servos chosen introduced significant backlash, which critically affected motion precision. Due to lack of experience with these limitations, the performance gap between industrial-grade AC servos and hobbyist-level DC motors was significantly underestimated. Redesigning the platform with more suitable actuators wasn't feasible within the time and budget constraints, largely due to not seeking expert feedback early on.

This experience highlighted the value of thorough research and early expert consultation. Relying on incomplete information during component selection led to compromises that limited system performance. If the project were to be repeated, early stakeholder engagement and expert input should be prioritised to make more informed design decisions and better allocate time and budget.

5 Acknowledgements

I would like to thank my supervisor, Dr. Tareq Assaf, for helping shape this project into something with real impact and long-term value. I am also grateful to Dr. Uriel Martinez Hernandez for his insight into how the platform could be developed into a more effective educational tool. I hope this project can one day evolve into something that inspires future students to discover a passion for robotics just as I have.

GenAI acknowledgement statement: GenAI was used solely to help structure paragraphs and rephrase existing points during the writing of this report.

6 References

[1] H. Z. Z. Z. F. S. Y. R. Z. J. B. K. W. V. D. M. B. V. K. R. M. Johannes Betz, "Teaching Autonomous Systems Hands-On: Leveraging Modular Small-Scale Hardware in the Robotics Classroom," *IEEE TRANSACTIONS ON LEARNING TECHNOLOGIES,* 2022.

[2] Robot Store, "Robotic Cost Comparison: New vs. Refurbished Robots," [Online]. Available: https://www.robot-store.co.uk/robotic-costs. [Accessed 8 5 2025].

[3] NIRYO, "NIRYO 6-Axis robot," [Online]. Available: https://niryo.com/6-axis-robot/. [Accessed 6 5 2025].

[4] Dobot, "Magician E6," [Online]. Available: https://www.dobot-robots.com/products/education/magician-e6.html. [Accessed 8 5 2025].

[5] F. O. Weiqi Xu, "Robotic roles in education: A systematic review based on a proposed framework of the learner-robot relationships," *Educational Research Review,* vol. 47, pp. 1-4, 2025.

[6] K. G. L. D. A. A. A. Z. Z. a. S. A. C. S. Evripidou, "Educational Robotics: Platforms, Competitions and Expected Learning Outcomes," *IEEE Access,* vol. 8, pp. 219534-219562, 2020.

[7] FANUC, "SCARA Series," 2025. [Online]. Available: https://www.fanuc.eu/eu-en/scara-series. [Accessed 29 4 2025].

[8] EPSON, "SCARA robots," 2025. [Online]. Available: https://www.epson.co.uk/en_GB/products/robots/scara-robots/c/scararobots?srsltid=AfmBOoqAsKES3wGLUxJ1M5k7nS0fWD_l-mdCYvaj9l5hCwILKAaidlHe. [Accessed 29 4 2025].

[9] YASKAWA, "SG400," 2025. [Online]. Available: https://www.yaskawa.eu.com/robotics/robots/pick-place/productdetail/product/sg400_6637. [Accessed 29 4 2025].

[10] J. W. J. A. M. Andrew Farley, "How to pick a mobile robot simulator: A quantitative comparison with a focus on accuracy of motion," *Simulation modelling practice and theory,* vol. 120, pp. 14-15, 2022.

[11] STANDARD BOTS, "How much does a SCARA robot cost? SCARA robot price in 2025," *STANDARD BOTS,* 28 4 2025.

[12] KUKA, "KR SCARA," [Online]. Available: https://my.kuka.com/s/category/robots/scara/kr-scara/kr-scara/0ZG1i000000XalCGAS?language=en_US&tab=Products. [Accessed 29 4 2025].

[13] Dejan, “SCARA Robot | How To Build Your Own Arduino Based Robot,” How To Mechatronics, [Online]. Available: https://howtomechatronics.com/projects/scara-robot-how-to-build-your-own-arduino-based-robot/. [Accessed 29 4 2025].

[14] P. Kopacek, “EDUCATIONAL ROBOT - “ROBCO” SCARA,” ReseachGate, Vienna, 2010.

[15] Trio Motion Technology, “Products,” [Online]. Available: https://www.triomotion.com/public/products/products.php?tabno=0. [Accessed 29 4 2025].

[16] buildplus, “Robotics Full-stack Engineer in Japan: What You Need to Know,” 1 12 2023. [Online]. Available: https://buildplus.io/blog/robotics-fullstack-engineer-need-to-know#:~:text=Answer%3A%20A%20Full%2DStack%20Robotics,and%20handling%20server%2Dside%20logic.. [Accessed 7 5 2025].

[17] K. E. M. T. D. D. Rafal Szczepanski, “Optimal Path Planning Algorithm with Built-In Velocity Profiling for Collaborative Robot,” *Advances in Sensing, Control and Path Planning for Robotic Systems,* vol. 24, no. 16, p. 1, 2024.

[18] Chuck Lewin, “Mathematics of Motion Control Profiles,” *Performance Motion Devices,* vol. 1, no. 1, p. 1.

[19] C. M. Luigi Biagiotti, “Trajectory with Double S Velocity Profile,” in *Trajectory Planning for Automatic Machines and Robots*, Heidelberg, Springer-Verlag, 2008, p. 79.

[20] ABB Robotics, “Operating manual - RobotStudio,” ABB, 2025.

[21] IMSystems, “Eliminating Backlash in Mechatronic applications Without Compromises,” IMSystems, [Online]. Available: https://imsystems.nl/backlash/. [Accessed 31 March 2025].

[22] Wikipedia, “Backlash (engineering),” Wikipedia, 27 January 2024. [Online]. Available: https://en.wikipedia.org/wiki/Backlash_(engineering). [Accessed 31 March 2025].

[23] J. Lemire, “Gear Backlash in Robotics Applications,” *Gear Technology,* p. 1, 9 April 2024.

[24] S. N. G. S. A. T. Eliana Giovannitti, “A virtual sensor for backlash in robotic manipulators,” *Journal of Intelligent Manufacturing,* vol. 33, pp. 1921-1937, 2022.

[25] J. Tang, “Gear Basics: Backlash vs Lost Motion,” *Oriental motor,* p. 1, 7 January 2021.

[26] S. D. R. G. A. F. Joseph Redmon, “You Only Look Once: Unified, Real-Time Object Detection,” 2016. [Online]. Available: https://pjreddie.com/media/files/papers/yolo_1.pdf. [Accessed 31 March 2025].

[27] Z. Kelta, “YOLO Object Detection Explained,” Datacamp, 28 September 2024. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained. [Accessed 31 March 2025].

[28] H. G. ,. A. O. ,. R. P. ,. T. Y. Alvaro Garcia, “Senior Design Project - Resistor Sorter,” Trinity University, San Antonio, 2013.

[29] M. D. N. A. K. M. T. M. M. R. H. Y. Mohammad Yousefi, “Robotic Sorting of Mechanical and Electrical Parts: An Autonomous Vision-Based Approach,” IEEE Xplore, 2024.

[30] B. D. S. O. M. G. D. V. D. W. M. S. M. G. Claire Floras, “ENPH 454 Advanced Engineering Physics Design Project – Group 8 Final Report - Resistor Sorter,” Queen's University, Kingston, 2022.

[31] T. H. Karl J. Åström, in *Advanced PID Control*, Lund, ISA, 2006, pp. 1-11.

[32] O. K. Bruno Siciliano, in *Springer Handbook of Robotics*, Springer, 2016.

[33] [Online]. Available: https://www.amazon.co.uk/AGFRC-Digital-Torque-Programmable-Steering/dp/B08YYR66QJ/ref=sr_1_1_sspa?crid=3PS9HHXVUICX8&dib=eyJ2IjoiMSJ9.jLgIOpzsDo3Zjr-al9bnefPE7J0tlruhHFUhGZlBox90z3WD5N5PXoY5a4Jb9EHH86Ll8zfLqP6bLl0kC6H8kt69Orfq_1Z9jf5nD0RiFt7uLtYgcsrO2mI53.

[34] ST Microelectronics, “RM0390,” 3 2021. [Online]. Available: https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf. [Accessed 2 5 2025].

[35] ARM MBED, “NUCLEO-F446RE,” [Online]. Available: https://os.mbed.com/platforms/ST-Nucleo-F446RE/. [Accessed 1 5 2025].

[36] J. Worth, “Servo Motor vs. Stepper Motor: how to choose the right one,” *igus® Engineer's Toolbox,* p. 1, 20 7 2023.

[37] L. EITEL, “Three ways to reduce heat when operating a step motor,” *MOTION CONTROL TIPS,* 29 6 2018.

[38] T. Polygenis, “How PLA Temperature Resistance Compares to Other 3D Printing Materials,” *Wevolver,* 9 6 2023.

[39] Sparkfun electronics, “Servos explained,” [Online]. Available: https://www.sparkfun.com/servos. [Accessed 2 5 2025].

[40] “AGFRC 24KG Digital High Torque Servo Programmable Metal Gear Steering Servo,” [Online]. Available: https://www.amazon.co.uk/dp/B08YYR66QJ?ref_=ppx_hzsearch_conn_dt_b_fed_asin_title_3. [Accessed 2 5 2025].

[41] “AGFRC B11DLS,” [Online]. Available: https://www.aliexpress.com/item/1005002041892505.html?src=google&pdp_npi=4%40dis!GBP!10.59!10.59!!!!!%40!12000018521765211!ppc!!!&src=google&albch=shopping&acnt=615-992-9880&isdl=y&slnk=&plac=&mtctp=&albbt=Google_7_shopping&aff_platform=google&aff_short_k. [Accessed 2 5 2025].

[42] R. Elliott, “Beginners' Guide to Potentiometers,” *Elliott Sound Products,* 22 1 2002.

[43] “Single Circle Potentiometer, 5k potentiometer RV24YN20F B502,” [Online]. Available: https://www.amazon.co.uk/Potentiometer-RV24YN20F-Accuracy-Resistance-Installation/dp/B08FDV2N4W?pd_rd_w=nmWRt&content-id=amzn1.sym.6d8ed89c-516c-44b0-b8e3-62363d275168&pf_rd_p=6d8ed89c-516c-44b0-b8e3-62363d275168&pf_rd_r=DS263KZ8X5M1NWGME3KX&pd_rd_wg=Slfx. [Accessed 3 5 2025].

[44] STEPPER ONLINE, “Nema 17 Bipolar 1.8deg 26Ncm(36.82oz.in) 0.4A 42x42x34mm 4 Wires,” [Online]. Available: https://www.omc-stepperonline.com/nema-17-bipolar-1-8deg-26ncm-36-8oz-in-0-4a-12v-42x42x34mm-4-wires-17hs13-0404s1. [Accessed 6 5 2025].

[45] Fysetc, “TMC2208,” [Online]. Available: https://wiki.fysetc.com/docs/TMC2208. [Accessed 6 5 2025].

[46] T. E. Z. C. S. A. P. L. Panayiotou K, “A Framework for Rapid Robotic Application Development for Citizen Developers,” *MDPI,* vol. 1, no. 1, 2022.

[47] G. Matthews, “Business waste,” E-waste Facts and Statistics, 17 1 2025. [Online]. Available: https://www.businesswaste.co.uk/your-waste/weee-recycling/e-waste-facts-and-statistics/. [Accessed 3 5 2025].

[48] A. Joshi, “Advancing E-waste Recycling,” *Circular Innovation Lab,* 2022.

[49] Recycleye, “UK AI-based waste-sorting robot start-up wins $17m of funding,” *Drives & Controls,* 8 2 2023.

[50] Programming Electronics, “Build your own ohmmeter!,” [Online]. Available: https://www.programmingelectronics.com/ohmmeter/?utm_source=YouTube&utm_medium=YouTubeDescription&utm_campaign=Build+your+own+auto-ranging+Ohmmeter!&utm_id=LeadGen. [Accessed 4 5 2025].

[51] ElectronicComponentsDetector, “ECDetector Dataset,” Roboflow , 1 2024. [Online]. Available: https://universe.roboflow.com/electroniccomponentsdetector/ecdetector. [Accessed 4 5 2025].

[52] Ultralytics, “Ultralytics YOLO11,” 30 September 2024. [Online]. Available: https://docs.ultralytics.com/models/yolo11/#overview. [Accessed 31 March 2025].

[53] Trio Motion Technology, “Trio RPS,” [Online]. Available: https://www.triomotion.com/public/software/roboticsHome.php?tabno=0. [Accessed 28 4 2025].

[54] Trio Motion Technology, “Products,” [Online]. Available: https://www.triomotion.com/public/products/products.php?tabno=1. [Accessed 28 4 2025].

7 Appendix 1

Final costing of project:

PART	DESCRIPTION	QTY	UNIT COST £	SUB TOTAL £
NUCLEO-F446RE	Microcontroller prototyping board	1	14.48	14.48
Arduino shield	Arduino Uno R3 prototyping shield	2	1.88	1.88
AGFRC B53BHS	24kgcm programmable servo	2	39.99	79.98
AGFRC B11DLS	9g metal gear micro servo	2	18.49	36.98
RV24YN20F B502	Single circle potentiometer, 5k	2	4.34	8.68
PLA filament	3D printing filament	3kg	13.99/kg	41.97
Other	Passive components, screw terminals			10.00
Arduino Uno R3	Microcontroller prototyping board	1	24.50	24.50
OV9726	USB camera module	1	13.29	13.29
74HC4051	8-channel mux	1	1.67	1.67
Miuzei MS18	9g micro servo	1	3.00	3.00
			TOTAL	**236.43**