

FACULTY OF ENGINEERING - UNIVERSITY OF PORTO

Gripper and Fourth Degree of Freedom for Educational SCARA Manipulator

Lucas Galvão Moreira de Freitas



Master's Degree in Electrical and Computer Engineering

Supervisor: Prof. Paulo Gomes Costa

Second Supervisor: Prof. José Luís Lima

October 31, 2025

Resumo

Nos últimos anos, a automação e a sua teoria dentro da robótica tornaram-se essenciais nos avanços tecnológicos e nos processos industriais, proporcionando optimizações em tarefas complexas em vários sectores com precisão. Entre os manipuladores robóticos, os manipuladores de braço robótico de montagem de conformidade seletiva (SCARA) destacam-se pela sua destreza, precisão e velocidade. Para melhorar a experiência dos estudantes num ambiente educativo, foi proposto um manipulador didático e acessível para fazer a ponte entre a teoria das aulas teóricas e as experiências práticas numa plataforma baseada em máquinas industriais reais.

Com base em investigações anteriores, este estudo apresenta um braço robótico com um novo grau de liberdade e uma garra para expandir as suas capacidades mecânicas e o seu impacto pedagógico, mantendo a sua segurança, baixo custo e modularidade. Os graus de liberdade (DOF) adicionais e os modelos de garras foram criados através de desenho assistido por computador (CAD) para permitir que a máquina efectue movimentos semelhantes às articulações do pulso e, consequentemente, tarefas de recolha e colocação.

Este documento ilustra os processos de conceção de um SCARA didático, desde o estudo dos princípios deste tipo de manipulador até à implementação de uma máquina com 4 DOF, e uma garra funcional. O trabalho engloba a conceção mecânica utilizando desenho assistido por computador e a seleção e integração de motores, codificadores, um microcontrolador e o desenvolvimento de um sistema de controlo baseado no FreeRTOS. A plataforma resultante pretende servir como uma ferramenta modular e replicável para o ensino da cinemática robótica, da integração mecatrónica e dos sistemas integrados em tempo real.

Este trabalho realça o valor educativo da integração de conceitos mecânicos, electrónicos e de software numa única máquina. A replicação do manipulador em maiores quantidades gerará uma melhor experiência para os iniciantes na área.

Abstract

In recent years, automation and its theory within robotics have become essential in technological advances and industrial processes, providing optimisations in complex tasks across several sectors with precision. Among robotic manipulators, selective compliance assembly robot arm (SCARA) manipulators stand out for their dexterity, accuracy and speed. To enhance the experience for students in an educational environment, a didactic and accessible manipulator was proposed to bridge theory from theoretical classes to practical experiences on an accurate platform based on real industrial machines.

Building on previous research, this study delivers a robotic arm with a new degree of freedom (DOF) and a gripper to expand its mechanical capabilities and educational impact while maintaining its safety, low-cost and modularity. The additional DOF and claw models were created using computer-aided design (CAD) to allow the machine to perform wrist-like articulation and pick and place tasks.

This document illustrates the processes of designing an educational SCARA, from studying the principles of this type of manipulator to implementing a working machine with 4 DOF and a functional gripper. The work encompasses mechanical design using CAD and the selection and integration of motors, encoders, a microcontroller, and a FreeRTOS-based control system development. The resulting platform aims to serve as a modular and replicable tool for teaching robotics kinematics, mechatronic integration and real-time embedded systems.

This work highlights the educational value of integrating mechanical, electronic, and software concepts into a single machine. Replicating the manipulator in larger quantities will generate a better experience for beginners in the area.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a more inclusive, equitable, and sustainable future. Comprising 17 interconnected goals, they address pressing global challenges including poverty, inequality, climate change, innovation, education, and environmental sustainability.

This project contributes directly to the following Sustainable Development Goals:

SDG 4 Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.3	The Development of a low-cost, open-source SCARA manipulator increases access to higher technical education and practical robotics training in underserved institutions.	Number of academic programs adopting the platform; student engagement rates in robotics and automation projects.
	4.4	Provides accessible, hands-on learning experiences in mechatronics, embedded systems, and automation, equipping learners with relevant industry skills.	Measured improvements in student proficiency in FreeRTOS, CAD modelling, and hardware interfacing
9	9.5	Promotes innovation through open-source hardware and firmware, encouraging research and student-led prototyping in low-cost automation.	Number of spin-off projects, capstone theses, or research papers utilising the platform.
	9.c	Reduces technological barriers in robotics education by leveraging affordable microcontrollers, 3D printing, and accessible software tools.	Decrease in total cost per unit; number of low-resource schools or labs implementing the system; hardware deployment statistics.

Acknowledgements

I want to express my profound gratitude to my supervisors Paulo Costa and José Lima for providing guidance, critical insights and support to shape my approach to the research and this thesis. Thanks for making me grow professionally as a person and as an engineer.

To my family, who always supported me throughout this journey. Thanks for showing and providing this path to make my life easier. To those who live far away, your encouragement crossed oceans and reminded me that distance cannot diminish love. And to the special member no longer with us, your memory has been a constant source of strength and inspiration.

To my laboratory partner and friend, Levy Mitsugui, who always provided valuable feedback and technical assistance when needed.

To my friends, a special thanks. Thank you for holding hands with me and crossing the finish line in this run as a one. Thanks for helping me in the most challenging moments and for the beautiful memories.

Lucas Freitas

“Alea iacta est”

Julius Caesar

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Problem Definition	2
1.3	Objectives	2
1.4	Thesis Structure	3
2	Literature Review	5
2.1	Robotics and Manipulators	5
2.2	Types of Manipulators	6
2.2.1	Articulated Manipulator (RRR)	6
2.2.2	Spherical Manipulator (RRP)	6
2.2.3	SCARA Manipulator	7
2.2.4	Cartesian Manipulator	8
2.2.5	Parallel Manipulator	8
2.3	Types of Joints	9
2.3.1	Revolute Joints	9
2.3.2	Prismatic Joints	9
2.4	Types of Links	10
2.5	Grippers	10
2.6	Embedded Control Systems	12
2.6.1	Role of Microcontrollers	12
2.6.2	Stepper Motor	12
2.6.3	Communication Protocols	13
2.6.4	Real-Time Operating Systems	14
2.7	Kinematics and Dynamics of SCARA Robots	15
2.8	Control Strategies for Robotic Manipulators	16
2.8.1	Open and Closed loop control	16
2.8.2	Proportional Integral Derivative control	17
2.9	Summary	18
3	Manipulator Architecture and Hardware	19
3.1	Mechanical Architecture	19
3.1.1	Kinematic diagram	19
3.1.2	Forward Kinematics	22
3.1.3	Inverse Kinematics	24
3.1.4	Jacobian Matrix	25
3.1.5	Workspace Analysis	27
3.2	Hardware Components	27

3.2.1	Motors and Drivers	27
3.2.2	Encoders and I2C Multiplexer	29
3.2.3	Load Cell and Amplifier	30
3.2.4	ESP32 Microcontroller and CNC Shield Integration	31
3.3	Summary	32
4	Software Architecture	34
4.1	System Architecture Overview	34
4.1.1	Real-Time Task Management with FreeRTOS	34
4.1.2	Task Architecture and Priority Management	34
4.1.3	Hierarchical System Design	35
4.1.4	Naming Relationship between Software and Hardware	36
4.2	One-Shot Functions	37
4.3	Sensor Data Acquisition Subsystem	38
4.3.1	Subsystem Architecture	38
4.3.2	Tasks	39
4.4	Motion Control Subsystem	42
4.4.1	Motor Control Task	42
4.4.2	PWM Control with MCPWM	42
4.5	Network Communication Subsystem	45
4.5.1	Server-Side Logic	45
4.5.2	Client-Side Logic	46
4.6	Kinematics Processing Subsystem	48
4.7	Summary and Conclusions	49
5	Results	51
5.1	Structural Improvements	51
5.1.1	Result Assembly	51
5.1.2	New Degree of Freedom and Gripper	51
5.1.3	Gripper	52
5.1.4	Encoder's Supports	53
5.2	Practical Tests	55
5.2.1	PWM Generation	56
5.2.2	Speed and Acceleration Analysis	56
5.3	Positional Testing	60
5.3.1	Elbow Up and Down on Inverse Kinematics	60
5.3.2	Repeatability Tests	63
5.4	Interactive Interface	64
5.4.1	System Status Panel	64
5.4.2	Quick Actions Panel	65
5.4.3	Operation Modes Panel	65
5.4.4	Real-Time System Data Panel	67
5.5	Cost Analysis	67
5.6	Alignment with UN Sustainable Development Goals	67
5.7	Summary of Results	68
6	Conclusion	70
	References	72

A	Schematic Diagram	76
B	Hardware Diagram	79
C	Encoder Task	80
D	Load Cell Task	81
E	Switch Task	82
F	Motor Task	83
G	Motor Callback Function	85

List of Figures

2.1	Difference between the motion of revolute and prismatic joints [7].	5
2.2	Articulated manipulator and its workspace.	6
2.3	Spherical manipulator and its workspace.	7
2.4	Industrial SCARA Yamaha YK-XR series [10].	8
2.5	Simulated 3D workspace.	8
2.6	SCARA manipulator and its workspace.	8
2.7	Designed cartesian manipulator from [11].	8
2.8	Simulated 3D workspace.	8
2.9	Cartesian manipulator and its workspace.	8
2.10	Prototype of parallel spherical manipulator for rehabilitation from [12].	9
2.11	Simulated 3D workspace.	9
2.12	Parallel manipulator and its workspace.	9
2.13	Deformation in different types of links [13].	10
2.14	Active gripper composed of 3 fingers by Robotiq [16–18].	11
2.15	Passive grippers operation.	11
2.16	ESP32 C3 DevKitC 02 [24].	12
2.17	Stepper motors used in this work.	13
2.18	Equivalent dynamic model of the stepper motor from [28].	13
2.19	The basic operation of I2C bus [30].	14
2.20	Open loop control system.	16
2.21	Closed loop control system.	17
3.1	Upgrade of the robot kinematic diagram.	20
3.2	SCARA manipulator with the considered axis labelled at each joint origin.	20
3.3	Top view of the manipulator.	21
3.4	Lateral view of the manipulator.	21
3.5	Reachable workspace of the SCARA robot in both 2D and 3D representations.	27
3.6	A4988 device. Adapted from [38].	28
3.7	Encoder and multiplexer used.	30
3.8	Load cell and its amplifier.	31
3.9	Compatible microcontroller with the CNC Shield V3.	31
3.10	CNC Shield V3 from [48].	32
3.11	3D Printer 42CH stepper motor driver shield expansion board from [49].	32
4.1	Hierarchical system block diagram.	36
4.2	Relation between drivers, motors and encoders.	37
4.3	Flow diagram of the encoder task.	39
4.4	Flow diagram of the loadcell task.	40

4.5	Flow diagram of the limit switch task.	41
4.6	Flow diagram of the motor control task.	43
4.7	Flow diagram of the motor callback function.	44
4.8	Flow diagram of the initialisation of the HTTP server.	46
4.9	Part one: Flow diagram with connection establishment from client to ESP32's server and the user's input sequence.	47
4.10	Part two: Flow diagram with sending the data input by the user and receiving the system data from the ESP32. If no values are sent, the interface uses the last valid user's input to receive new values from the server.	48
4.11	Flow diagram with kinematics engine.	49
5.1	Comparison between the designed claw and implementation.	52
5.2	Assembly of the new DOF.	52
5.3	Modular piece designed to carry the end-effector.	53
5.4	A second NEMA 11 is used to perform the open and close motions of the claw.	53
5.5	Open and closed forms of the claw.	54
5.6	Support of the encoder of the last joint.	54
5.7	Encoder holder demonstration.	55
5.8	Encoder holder and its magnet to measure the actuation of the arm.	55
5.9	Oscilloscope analysis of the PWM frequency.	56
5.10	Normal operation study using the encoder.	57
5.11	Decrease of 75% of acceleration study.	57
5.12	Decrease of 75% of speed study.	58
5.13	Elbow down study.	59
5.14	Elbow up study.	59
5.15	Python study of the inverse kinematics.	60
5.16	SCARA inverse kinematics study.	61
5.17	Desmos 3D simulation study	61
5.18	Python study two of the inverse kinematics.	62
5.19	SCARA inverse kinematics study two.	62
5.20	Desmos 3D platform with solutions to the inverse kinematics	63
5.21	Developed interface hosted by the microcontroller.	64
5.22	System status panel showing connection status and update information.	65
5.23	Quick actions panel.	65
5.24	Operation modes panels.	66
5.25	Real-time system data panel showing live sensor readings and calculated positions.	67
5.26	Fully assembled SCARA.	69
A.1	Schematic diagram of the experimental setup. Hardware connections.	77
A.2	Schematic diagram of the experimental setup. Shield connections.	78
B.1	Diagram with all components and communication protocols used in the project.	79

List of Tables

1.1	Educational SCARA Manipulator Project Requirements	3
3.1	Symbol Definitions	22
3.2	SCARA DH Parameters	22
3.3	Inverse Kinematics Solutions for SCARA Robot	25
3.4	Comparison of Stepper Motor Specifications	28
3.5	Key Characteristics of the A4988 Stepper Motor Driver [39]	28
3.6	Microstepping Resolution Truth Table for A4988 [39]	29
5.1	Positional Error Metrics with Inverse Kinematics	63
5.2	Machine components and their estimated costs.	68

List of Acronyms

ADC	Analog-to-Digital Converter
API	Application Programming Interface
CAD	Computer-Aided Design
DH	Denavit–Hartenberg
DOF	Degrees of Freedom
ESP-IDF	Espressif IoT Development Framework
FEUP	Faculty of Engineering of the University of Porto
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HMIs	Human-Machine Interfaces
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IP	Internet Protocol
MCPWM	Motor Control Pulse Width Modulation
PID	Proportional-Integral-Derivative
PLA	Polylactic Acid
PWM	Pulse Width Modulation
RTOS	Real-Time Operating System
SCARA	Selective Compliance Assembly Robot Arm
SCL	Serial Clock Line
SDA	Serial Data Line
SDGs	Sustainable Development Goals
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
WiFi	Wireless Fidelity

Chapter 1

Introduction

The dissertation proposal is based on the need for affordable robotic manipulators suitable for educational environments. Industrial robotic manipulators are prohibitively expensive, making acquiring and maintaining them in the quantities required difficult. Therefore, the objective is to develop a robotic manipulator with four degrees of freedom and a gripper, with performance characteristics close to those of industrial models but at a substantially lower cost and risk-free for robotics students.

The following chapter provides an overview of the objectives of the dissertation, focusing on the motivations and objectives while giving context of the environment in which the robot was developed. Furthermore, the document structure is defined.

1.1 Context and Motivation

In recent years, robotics has become increasingly central to technological advances and industrial processes. Robotics and automation are essential for optimising manufacturing operations, streamlining complex tasks, and improving precision across several sectors, from electronics assembly to medical applications [1, 2]. Among robotic systems, Selective Compliance Assembly Robot Arm or SCARA manipulators stand out for their flexibility, precision, and speed [3]. These properties distinguish operations like soldering, item placement, or even delicate operations in surgical environments from distinct systems [4].

This project builds on the dissertation authored by Carlos Costa in [5], where an affordable and safe 3 DOF (degrees of freedom) didactic SCARA manipulator was developed. Regardless of some mechanical limitations, since the manipulator is made primarily of PLA (Polylactic Acid), the goal of a valuable educational tool was achieved. The proposed improvements in this project aim to address constraints like developing a robust and modular software architecture, a new DOF and a gripper.

Industrial SCARA manipulators are expensive. When considering assembly, maintenance, safety concerns, and accessibility, there are no available options in the market. A low-cost, safe, and interactive manipulator prototype was developed to support the Faculty of Engineering at the

University of Porto (FEUP), providing the students with hands-on experiences. A well-designed robot allows for reproducing it on a larger scale, making a laboratory with multiple manipulators a reality for the robotic students.

1.2 Problem Definition

The problem addressed in this project is the lack of accessible robotic manipulators for educational institutions, particularly in robotics courses involving many students. Generally, many industrial robots have closed-source or proprietary software, limiting the learning experience.

Another problem is the lack of practical experience in electronics and embedded software, as many projects tend to work with software or hardware each time, but not both simultaneously. This work integrates mechanical design, electronics, and real-time software into a single project.

Therefore, there is a need for an affordable, modular and open-source SCARA platform suitable for teaching essential concepts for understanding automation and control systems. The challenge, therefore, is to add a new DOF and a gripper to an already designed manipulator while maintaining its educational use.

1.3 Objectives

As addressed in the conclusions section of the previous work [5], developing a Graphical User Interface (GUI) broadens the manipulator's utility. An interactive GUI allows the operator to know the state of the machine, including the readings from the sensors and the position of the joints in real-time. Another important aspect of this interface is to make the manipulator controllable. This acts like a manual mode where the student can experiment and input custom positions to the machine, making evident concepts of inverse and direct kinematics, showing the relevance of automation and Industry 4.0.

Another concern in the previous work was integrating an end-effector. This addition with a new DOF enables more complex manipulation tasks, including wrist rotation and object grasping, copying real-world industrial scenarios. In the end, a modular design is needed to easily replicate, repair or upgrade the gripper, making it suitable for its environment.

To establish the scope of the project, a set of technical and functional requirements was defined for the development of the manipulator. These requirements, addressed in [Table 1.1](#), encompass mechanical performance parameters, software functionalities, and economic considerations, ensuring they contain the required specifications inside an educational environment.

Table 1.1: Educational SCARA Manipulator Project Requirements

Requirement	Specification
Degrees of Freedom	4 DOF and Gripper: 3 rotational joints (θ_2^* , θ_3^* , θ_4^*), 1 prismatic joint (d_1^*), and a gripper mechanism for object manipulation
2D Workspace Area	$\geq 750.0 \text{ cm}^2$: Minimum reachable area in horizontal plane for educational demonstrations and object manipulation tasks
3D Workspace Volume	$\geq 5000.0 \text{ cm}^3$: Sufficient volumetric workspace to accommodate educational scenarios including pick-and-place operations
Positional Repeatability	$\leq 5.0 \text{ mm}$: Maximum positional deviation when returning to the same target coordinates, ensuring precision for manipulation tasks
User Interface	Interactive GUI with real-time monitoring: Web-based interface supporting both direct and inverse kinematics control with live sensor feedback and system status visualisation
Software Licensing	Open source implementation: Complete source code availability for educational modification and study
Total System Cost	$\leq 250 \text{ €}$: Maximum budget constraint including all mechanical components, electronics, sensors, and actuators to ensure accessibility for educational institutions

Therefore, this work aims to improve the manipulator prototype design with a fourth DOF and a gripper while documenting the mechanical, electrical and software design modifications. As an educational tool, the SCARA will be used to teach about electronics, real-time programming, control theory, and kinematics analysis, using practical experiences.

1.4 Thesis Structure

This document is structured in 6 chapters, explained as follows:

- **Chapter 1 – Introduction:** Provides the project’s motivation, objectives, and scope, along with an overview of the methodologies employed.

- **Chapter 2 – Literature Review:** Reviews relevant literature on robotic manipulator architectures, control systems, communication protocols, and existing SCARA implementations, highlighting the rationale for the design choices made in this work.
- **Chapter 3 – Manipulator Architecture and Hardware:** Details the manipulator's mechanical structure and electronic architecture, including component selection, kinematic modelling, and hardware integration.
- **Chapter 4 – Software Architecture:** Describes the control software developed using the ESP-IDF framework and FreeRTOS, covering task scheduling, motion control logic, and communication interfaces.
- **Chapter 5 – Results:** Presents the manipulator's experimental setup, testing procedures, and performance evaluation, including accuracy, repeatability, and overall mechanical design.
- **Chapter 6 – Conclusion:** Summarises the main contributions of the project and discusses potential directions for future improvements and research extensions.

Chapter 2

Literature Review

2.1 Robotics and Manipulators

Robotics is a young field where electrical and mechanical body components are combined with a computer brain [6]. It encompasses machine design, development, and application to perform tasks with precision, repeatability and efficiency. One of its robots, the manipulator, succeeded in evolving tasks like material transfer, injection moulding, and stamping. In general, they could be programmed to execute a sequence of movements [7].

The manipulators are machines composed of links connected by joints to form a kinematic chain. These joints are typically rotary (revolute) or linear (prismatic), shown in Figure 2.1. More types of joints are specified in Section 2.3. Ultimately, these robots come with the advantages of decreased labour costs, increased precision and productivity, increased flexibility and more humane working conditions as robots perform dull, repetitive or hazardous jobs [7].

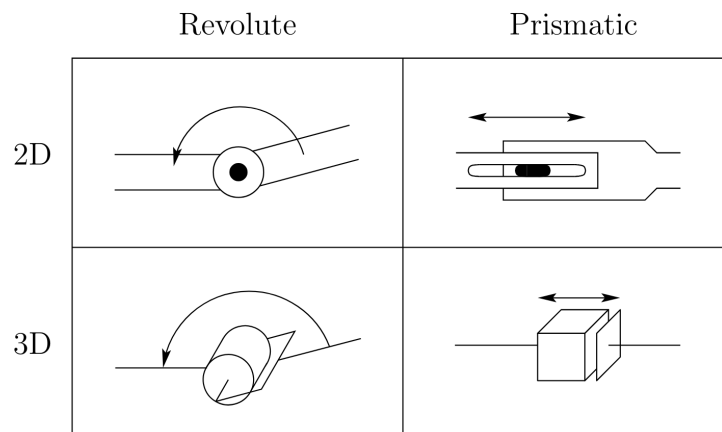


Figure 2.1: Difference between the motion of revolute and prismatic joints [7].

2.2 Types of Manipulators

These robots can be categorised according to their geometry, kinematics, and intended application. This subsection reveals common kinematic arrangements of manipulators. Matplotlib tools and Python were used to show the 3D workspace with the manipulator's structure to help visualise each robot's movement.

2.2.1 Articulated Manipulator (RRR)

Also called a revolute, or anthropomorphic manipulator, it provides a relatively large freedom of movement in a compact space [7]. Widely used for welding and painting due to its simplicity. The following Figure 2.2 shows an example of an articulated arm.

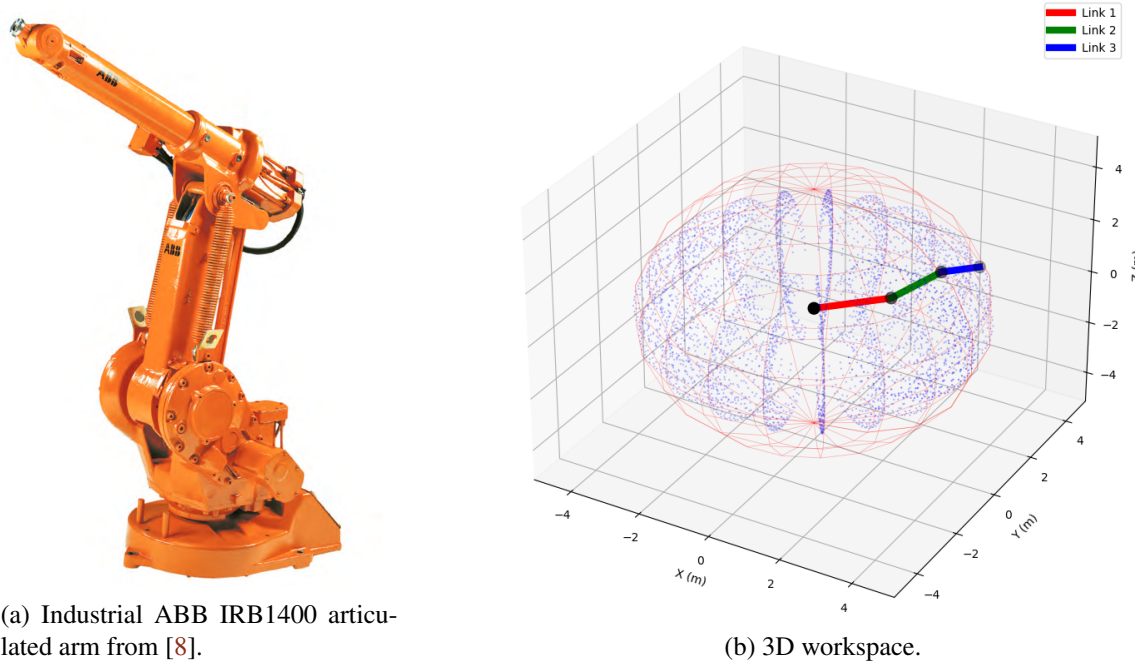


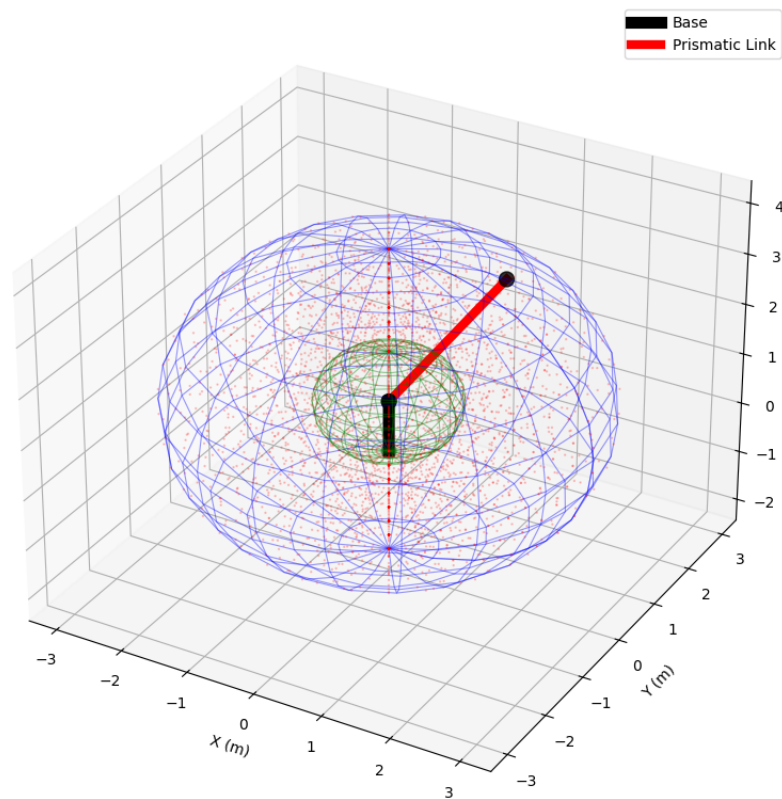
Figure 2.2: Articulated manipulator and its workspace.

2.2.2 Spherical Manipulator (RRP)

This manipulator can be obtained by replacing the last revolute joint of the previous manipulator configuration in Subsection 2.2.1 with a prismatic one, as shown in Figure 2.3a. As a side note, the term spherical is related to the spherical coordinates because the first three joint variables correspond to the spherical coordinates used to describe the end-effector's position relative to a reference frame with its origin located at the intersection of the three z-axes [7].



(a) Stanford arm with six degrees of freedom, designed at Stanford University by Victor Scheinman in 1969 [9].



(b) Simulated 3D workspace.

Figure 2.3: Spherical manipulator and its workspace.

2.2.3 SCARA Manipulator

SCARA Manipulators: The Selective Compliance Assembly Robot Arm consists of two parallel rotary joints and a vertical prismatic joint. Its selective compliance in the horizontal plane makes it ideal for high-speed pick-and-place operations, as shown in [Figure 2.6](#):

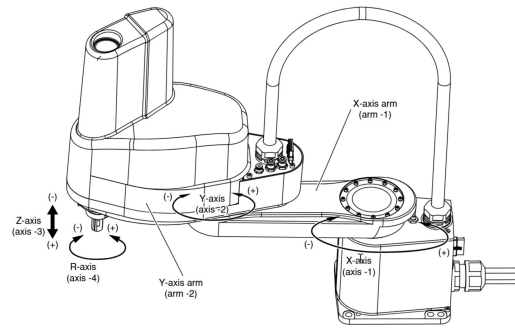


Figure 2.4: Industrial SCARA Yamaha YK-XR series [10]. Each joint can change the X, Y, and Z axes, achieving a variable range of motion.

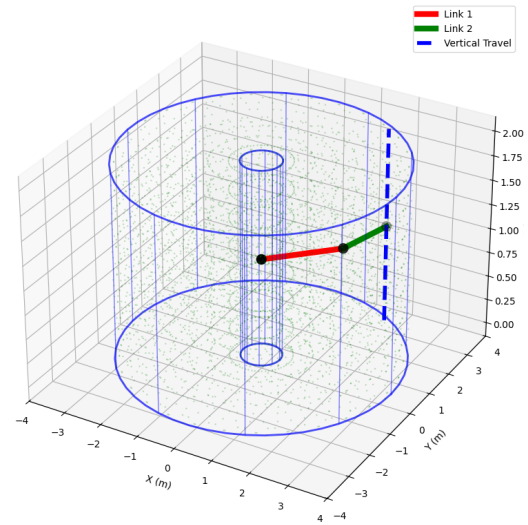


Figure 2.5: Simulated 3D workspace.

Figure 2.6: SCARA manipulator and its workspace.

2.2.4 Cartesian Manipulator

Cartesian Manipulators: Composed of three prismatic joints aligned along orthogonal axes, they provide linear motion in the X, Y, and Z directions. Their simple structure makes them precise and easy to control, but limits their workspace to a rectangular volume, as shown in Figure 2.9:

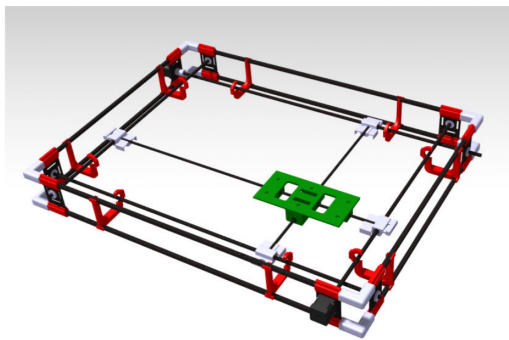


Figure 2.7: Designed cartesian manipulator from [11].

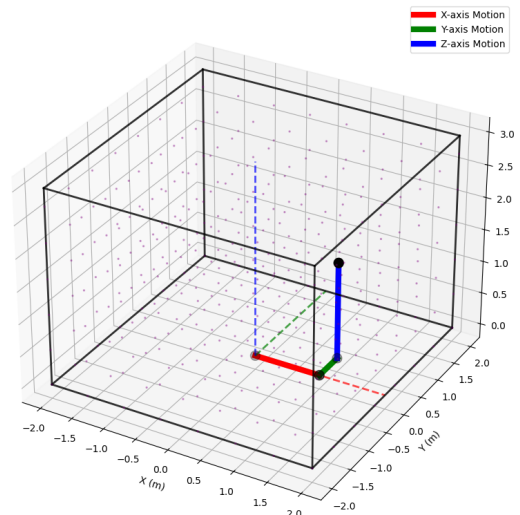


Figure 2.8: Simulated 3D workspace.

Figure 2.9: Cartesian manipulator and its workspace.

2.2.5 Parallel Manipulator

Parallel Manipulators: Consist of multiple kinematic chains (or legs) that simultaneously support and control the end-effector. This configuration offers a simple and intuitive workspace,

typically shaped as a rectangular volume, and allows for straightforward control and high positional accuracy. However, the workspace is limited by the physical travel of each prismatic joint.

Figure 2.12 shows an example of a parallel spherical manipulator:

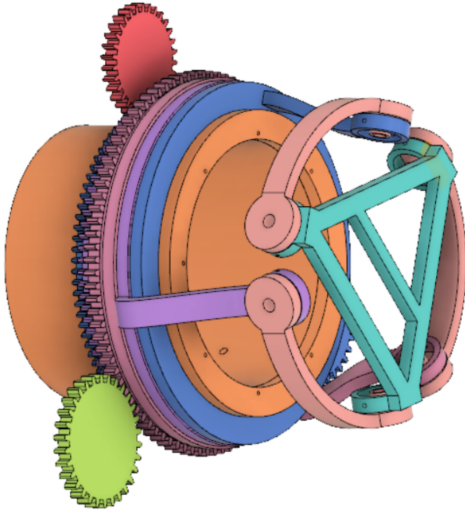


Figure 2.10: Prototype of parallel spherical manipulator for rehabilitation from [12].

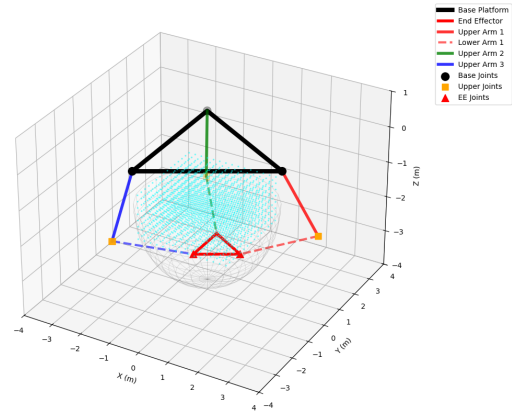


Figure 2.11: Simulated 3D workspace.

Figure 2.12: Parallel manipulator and its workspace.

2.3 Types of Joints

Robotic manipulators are made up of a series of joints. These joints are analogous to a shoulder, elbow, or wrist to connect various links resembling an upper arm or forearm to permit an end-effector to be moved to a particular status or direction [13]. Defining whether a joint is prismatic or revolute will be crucial when analysing the kinematics and dynamics of the SCARA robot, referred to in 2.7, as it directly affects the mathematical modelling used for the manipulator.

2.3.1 Revolute Joints

Revolute or rotational joints allow relative rotation between two connected links around a fixed axis, as shown in the Figure 2.1. These joints are often powered by servo or stepper motors, enabling the ability to move in an x-y axis [14].

2.3.2 Prismatic Joints

Prismatic joints allow linear motion along a single axis, shown in 2.1. Unlike revolute joints, prismatic joints are ideal for tasks requiring precise linear displacement, and are very commonly used for vertical positioning (Z-axis) during tasks like pick and place [14].

2.4 Types of Links

In robotics, the links refer to the structural components connecting joints in a robotic system (or joints to the end-effector). Defining the type of link is directly associated with its deformation when subjected to external loads. These types include: After identifying the joint types of the manipulator, the next step is to determine the types of links it contains. The types include:

- **Rigid Link** [13, 15]: Composed of a series of rigid links, connected by joints, that don't deform significantly under external load, shown in 2.13a.
- **Flexible Link**: Composed of links that deform under loads, causing deformation on both structures in 2.13b.
- **Rigid-Flexible Link**: Combination of rigid and flexible links in their structure as shown in 2.13c.

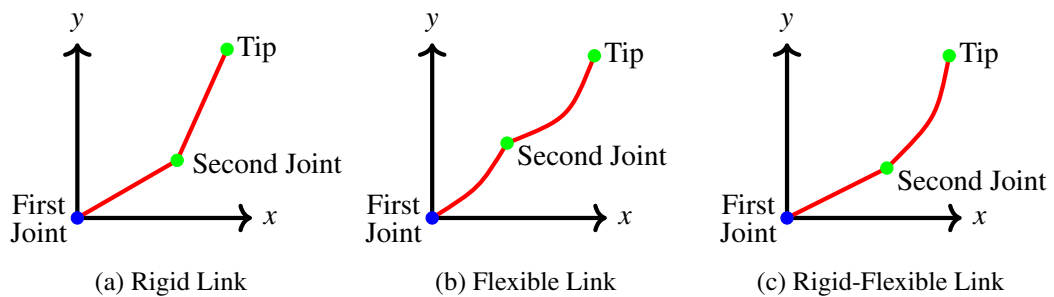


Figure 2.13: Deformation in different types of links [13].

Although rigid-link manipulators are easier to model, some applications might not accurately capture the system's dynamics. More realistic representations of real-world systems can be obtained with flexible and rigid-flexible link manipulators, but they demand more sophisticated modelling methods [13].

2.5 Grippers

As stated by Mahmud Huseynov:

"All industrial robots need a unique tool to perform tasks, known as an effector, mounted onto the 'end' of the robot; the end effector. [...] Nowadays, all commercial grippers are designed to be more universal to decrease robot costs." [16].

The end effector is one of the crucial areas in the design of such robots. Properties of the objects, such as size, weight, shape and surface texture, should be considered when designing the gripper for an industrial manipulator. The grippers can be defined as follows:

- **Active gripper:** Also known as anthropomorphic grippers or robot hands, active grippers are end-effectors designed to mimic the prehensile function of human fingers, allowing holding objects securely. This friction-based grasping approach typically relies on powered actuators such as servo motors, offering dynamic grip strength and finger positioning [16, 17]. One example is given in the 2.14.
- **Passive gripper:** This type of end-effector is mechanically simpler and cheaper than an active one, as shown in the 2.15a. They hold objects using the jamming effect or magnetorheological effect [16, 17].

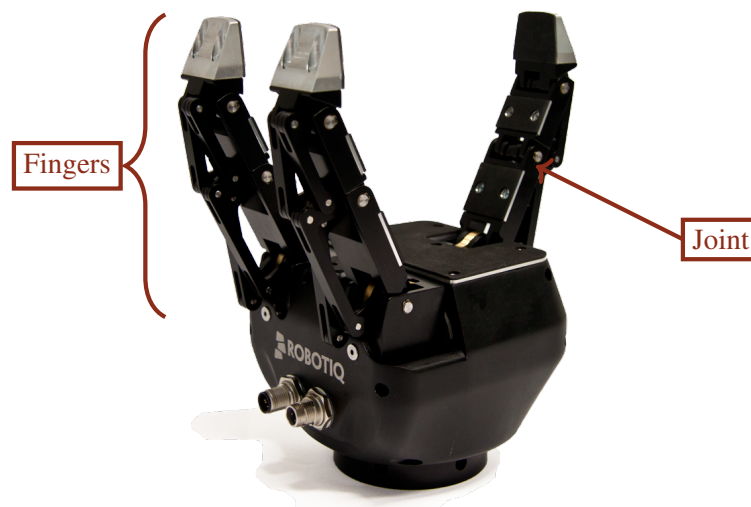
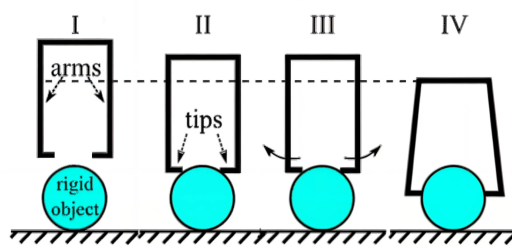


Figure 2.14: Active gripper composed of 3 fingers by Robotiq [16–18].



(a) Passive soft gripper allowing fingers to bend outwards when pushed against a rigid object. Adapted from [19].



(b) Elastic membrane holding object. Adapted from [16].

Figure 2.15: Passive grippers operation.

The articles [20] and [21] also explore hybrid gripper designs. These grippers combine characteristics of both rigid and active components.

2.6 Embedded Control Systems

2.6.1 Role of Microcontrollers

The microcontroller plays a central role in the control and operation of the SCARA robotic arm. This integrated circuit is designed to execute control tasks by processing inputs, running programmed instructions, and controlling the actuators of the manipulator [22].

Acting as the system's "brain", it coordinates joint movements and manages crucial functions. Some examples include generating signals such as pulse width modulation (PWM) for motor control, processing sensor data for collision avoidance, or using communication protocols such as Inter-Integrated Circuit (I2C) or Universal Asynchronous Receiver-Transmitter (UART) to receive and send data to the system.

The board used in this work contains an ESP32 chip, developed by **Espressif Systems**. This chip comes with Wireless Fidelity (WiFi) and Bluetooth capabilities. It is produced in prototype designs suitable for smart home devices, automation systems, wearables, audio applications, and cloud-based IoT solutions, making them ideal for industrial projects [23]. One example is the *ESP32 C3 DevKitC 02* in **Figure 2.16**.

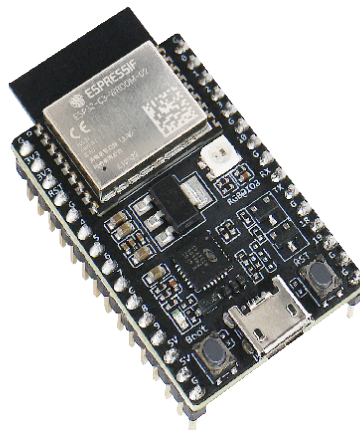


Figure 2.16: ESP32 C3 DevKitC 02 [24].

2.6.2 Stepper Motor

A stepper motor, called a pulse motor, can directly receive digital pulses from the computer to make the motor rotate through the corresponding angle. Stepping motor is widely used as an executive component when rapid start, stop, and precise positioning are required [25]. In other words, they convert electrical pulses into discrete mechanical movements.

Unlike other motors, like the DC or servo motors, stepper motors move in fixed angular increments, known as steps. The internal structure and winding configuration determine these steps. They are often used in CNC and 3D printing machines, since they do not require feedback sensors for basic positioning tasks.

Ultimately, even after considering servo and DC motors, the NEMA 11 and NEMA 17 from [Figure 2.17](#) stepper motors were chosen because of their durability and high torque at low speeds. In an educational environment, durability is appreciated because when giving the wrong input to the system, multiple motors could be damaged over time, making the system inoperative. Its circuit model can be found in [Figure 2.18](#).



(a) NEMA 17 from [26].



(b) NEMA 11 from [27].

Figure 2.17: Stepper motors used in this work. The number beside NEMA means the width of the motor's faceplate in tenths of an inch. 1.7 inches $\approx$ 43 mm and 1.1 inches $\approx$ 28 mm.

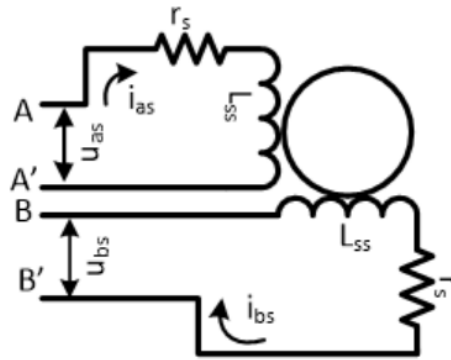


Figure 2.18: Equivalent dynamic model of the stepper motor from [28].

2.6.3 Communication Protocols

Integrating multiple electronic subsystems, such as motor drivers, sensors, and external computers for monitoring, requires reliable data exchange. In the industry, communication protocols define the interconnections between the communication entities, such as control systems and industrial devices. These protocols include signal transmission, error control, validation and security aspects [29]. Two protocols relevant to this work are the I2C and Transmission Control Protocol/Internet Protocol (TCP/IP).

I2C

The I2C or Inter-Integrated Circuit protocol is a synchronous, serial bus communication widely used in embedded systems and integrated circuits. It is based on the Serial Clock Line (SCL) and Serial Data Line (SDA) to support bi-directional communication between multiple interconnected devices in a master-slave structure [30].

This protocol allows multiple sensors, encoders, and expanders to share the same physical connection without additional wiring. In this project, the I2C is used to acquire the data from the system sensors while reducing the pin count and simplifying the circuit schematic. An example of this data transmission is shown in Figure 2.19.

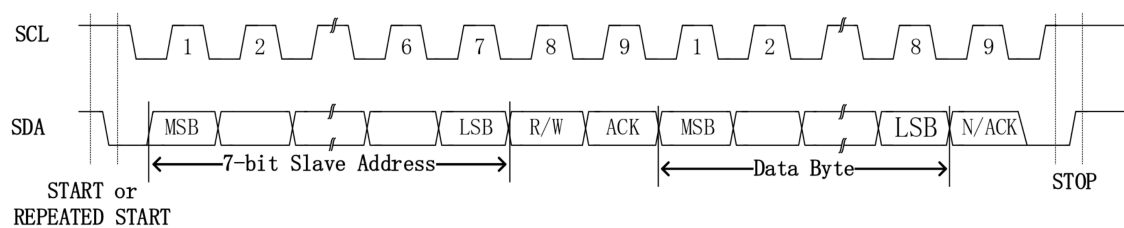


Figure 2.19: The basic operation of I2C bus [30].

TCP/IP

The Transmission Control Protocol/Internet Protocol or TCP/IP connects devices on the Internet and local networks. It defines how data is packetised, addressed, transmitted, routed and received, ensuring a reliable and ordered data delivery between networks.

In the context of robotic systems, TCP/IP is widely adopted for its flexibility and compatibility with existing network infrastructure. Although not optimised for hard real-time performance, typical TCP/IP-enabled communications exhibit response times of 100 milliseconds—adequate for supervisory control, system configuration, status monitoring, and data logging applications [29].

Moreover, TCP/IP can be used with embedded Hypertext Transfer Protocol (HTTP) servers hosted directly on microcontrollers. This allows the implementation of browser-accessible human-machine interfaces (HMIs), enabling intuitive remote control and monitoring of robotic manipulators through standard web technologies. Such interfaces significantly enhance user accessibility and facilitate remote diagnostics by removing the need for specialised client software.

2.6.4 Real-Time Operating Systems

A Real-Time Operating System (RTOS) is designed to execute tasks within strict timing constraints, ensuring that time-critical operations are completed deterministically. They are employed in applications where the outcome of a process directly affects ongoing system behaviour, and where delays or missed deadlines could lead to instability or unsafe operation. It is responsible

for timely task scheduling and managing hardware events that originate from outside the system, such as sensor readings or actuator triggers [31].

In this project, the FreeRTOS kernel—integrated into the ESP-IDF framework—serves as the RTOS environment for the ESP32 microcontroller. FreeRTOS provides preemptive multitasking, inter-task communication (queues, semaphores, and mutexes), time-slicing, and event-driven scheduling [32]. These features allow the control software to be structured into independent, prioritised tasks—for example, managing motor control, network communication, sensor feedback, and user interface interactions in parallel.

2.7 Kinematics and Dynamics of SCARA Robots

The kinematic analysis of a robotic manipulator involves understanding the relationship between the joint parameters and the resulting position and orientation of the end effector in space. This analysis is typically divided into two main components: forward kinematics and inverse kinematics.

Forward kinematics, also known as direct kinematics, involves computing the position and orientation of the end effector given a set of known joint variables and link parameters. This computation provides a deterministic mapping from joint to Cartesian space and is essential for trajectory planning and simulation [33].

Conversely, inverse kinematics determines the required joint parameters to achieve a specific end-effector pose (position and orientation) in Cartesian space. Solving the inverse kinematics problem involves manipulating nonlinear equations that relate the joint variables to the spatial configuration of the end effector. For manipulators such as SCARA robots, closed-form solutions are often possible due to their relatively simple structure and planar geometry [33].

To systematically study the geometry of how each link is connected in a manipulator, the Denavit-Hartenberg (DH) convention is used. The system is widely used to systematically define the spatial relationships between links and joints. This method standardises the assignment of coordinate frames and expresses each transformation between consecutive links as a product of four elementary transformations: two rotations and two translations. The homogeneous transformation matrix A_i for joint i is given by:

$$A_i = \text{Rot}_z(\theta_i) \cdot \text{Trans}_z(d_i) \cdot \text{Trans}_x(a_i) \cdot \text{Rot}_x(\alpha_i)$$

Which expands to:

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

In this matrix, the four parameters θ_i , d_i , a_i , and α_i define the geometric relationships between adjacent links:

- θ_i : the joint angle (rotation about the z-axis),
- d_i : the link offset (translation along the z-axis),
- a_i : the link length (translation along the x-axis),
- α_i : the link twist (rotation about the x-axis).

By chaining the individual transformation matrices A_1 to A_n , the full forward kinematics of an n -degree-of-freedom manipulator can be derived. This framework simplifies the computation of the end-effector pose and forms the foundation for Jacobian analysis and dynamic modelling in later stages of control system design [7].

2.8 Control Strategies for Robotic Manipulators

Robot manipulators encounter different challenges in achieving precise position tracking control, including load variations, external disturbances, uncertainties, friction, dynamic coupling, nonlinear behaviour, hysteresis effects, complex high-order dynamics, and underactuated characteristics.

Achieving precise position tracking control is challenging. Load variation, disturbances, friction, nonlinearity, hysteresis and other properties demand advanced control strategies to overcome these impediments [13]. In this section, the most relevant control strategies for the project are presented.

2.8.1 Open and Closed loop control

These two types of control are foundational strategies in robotic systems. In an open-loop control system 2.20, the commands are sent to the robot's actuators without feedback from the sensors. This method is easy to implement, but cannot adjust for disturbances or errors during operation (for example, when stepper motors skip a step).

Alternatively, the closed loop uses sensor feedback to continuously monitor the robot's state and adjust the commands according to the task, as seen in 2.21. This control system can correct deviations and compensate for external disturbances, making it essential for tasks requiring accuracy for precise positioning and handling variable loads.



Figure 2.20: Open loop control system.

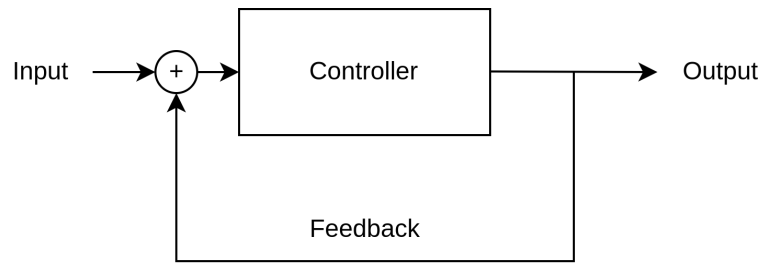


Figure 2.21: Closed loop control system.

2.8.2 Proportional Integral Derivative control

The Proportional Integral Derivative (PID) control is one of the most used closed-loop control strategies. Even with advances in digital technology, more than 90 industrial controllers are still implemented based around PID algorithms, particularly at the lowest levels [34]. This controller can be intuitive to design and tune, but when multiple are used or the system is complex and conflicting, high stability is hard to achieve [13, 34]. Designing this controller consists of completing the balance of speed, accuracy and stability by changing the following parameters:

- **Proportional:** Correction based on the magnitude of the error.
- **Integral:** Eliminating the steady-state error based on the accumulation of past errors.
- **Derivative:** Anticipation of future errors considering the rate of change.

The following expressions are the transfer function of the PID. Generally, they are written as 2.1, which is "parallel form" or 2.2 as the "ideal form" [34]:

$$G(s) = K_P + K_I \frac{1}{s} + K_D s \quad (2.1)$$

$$G(s) = K_P \left(1 + \frac{1}{T_I s} + T_D s \right) \quad (2.2)$$

- K_P : The proportional gain proportional to the error.
- $K_I \frac{1}{s}$: The integral term accounting for the accumulated error over time.
- $K_D s$: The derivative term predicts the error trend and improves the system stability.
- $T_I = \frac{K_P}{K_I}$: The integral time constant.
- $T_D = \frac{K_D}{K_P}$: The derivative time constant.

This method is very effective, but it requires careful parameter tuning. Poorly setting the parameters can lead to instability or bad performance, making a calibration step essential to achieve optimal control.

2.9 Summary

This chapter has provided an overview of relevant manipulator architectures, highlighting standard robotic arm configurations' structural and functional characteristics, including cartesian, articulated (revolute), SCARA, cylindrical, and parallel manipulators. Each configuration offers distinct advantages and limitations regarding workspace, payload capacity and repeatability.

This comparative analysis aims to select an appropriate joint configuration for a compact, low-cost, educational robotic platform focused on planar positioning tasks. In this context, the SCARA architecture emerged as a highly suitable candidate due to its inherent mechanical simplicity, ease of control, and efficiency in horizontal plane operations.

While articulated arms offer greater dexterity and a larger workspace volume, they also introduce increased kinematic complexity, higher inertia, and control challenges that are not always necessary for the target use case. SCARA manipulators, however, offer high speed, repeatability, and a rigid vertical axis, with a relatively simple kinematic complexity, making them the right choice for this project.

Chapter 3

Manipulator Architecture and Hardware

This chapter presents an overview of the physical structure, hardware components and mathematical modelling of the SCARA manipulator used in this work. The mechanical configuration is contextualised in [Section 3.1](#), where the manipulator's geometry and motion constraints are obtained from the links and joints of the manipulator. Then, a detailed description of the hardware architecture is made in [Section 3.2](#) while explaining the functionality and limits of each equipment used in this work.

3.1 Mechanical Architecture

The SCARA manipulator's mechanical architecture defines each DOF's physical configuration and provides the structure for actuation and sensorisation. The robot consists of a base frame, a vertical prismatic axis for height adjustment, two rotational joints in the horizontal plane, and a fourth rotational axis to orient the end-effector.

3.1.1 Kinematic diagram

The kinematic diagram of the manipulator is a simplified representation of the joint structure and motion relationships, without the context of physical dimensions. This is the first step for deriving both forward and inverse kinematic models to control the joints and position of the end-effector. The following [Figure 3.1](#) shows the kinematic diagram of the manipulator:

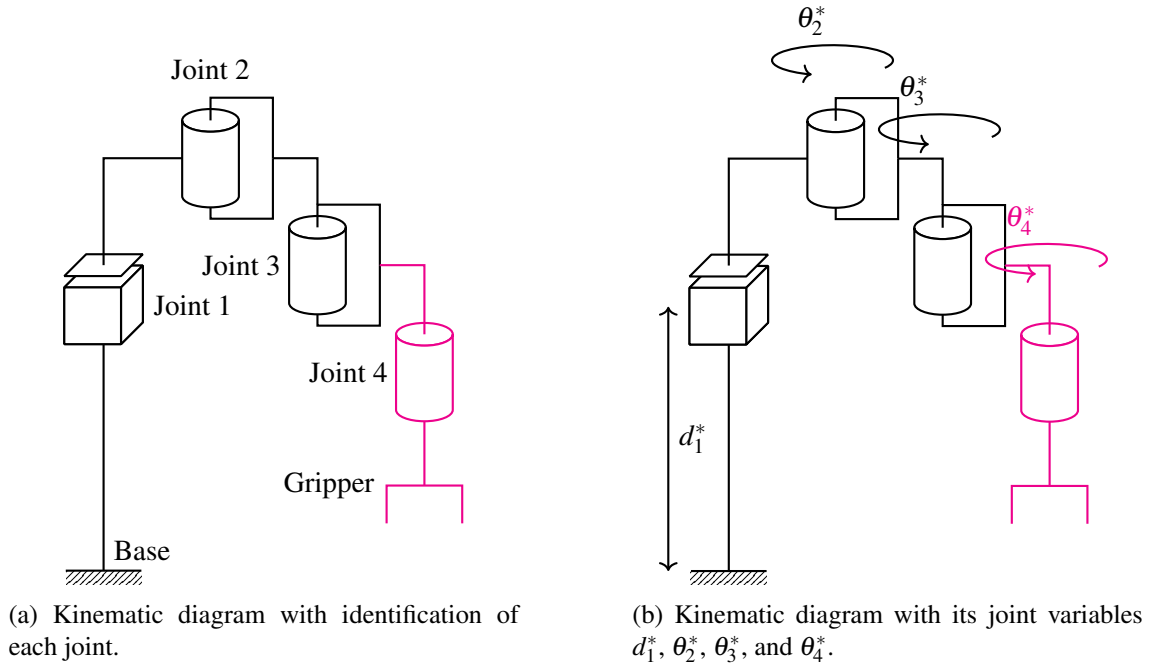


Figure 3.1: Upgrade of the robot kinematic diagram. Black frames represent the previous kinematic diagram [5], while magenta frames represent the fourth joint and claw added to the design.

As the manipulator is composed of 4 joints, it consists of 4 DOF: a vertical translational joint (d_1^*), two rotational joints in the horizontal plane (θ_2^* and θ_3^*), and a wrist rotation joint (θ_4^*) located at the end-effector. Following the DH convention, Figure 3.2 shows the attached coordinate systems for each joint:

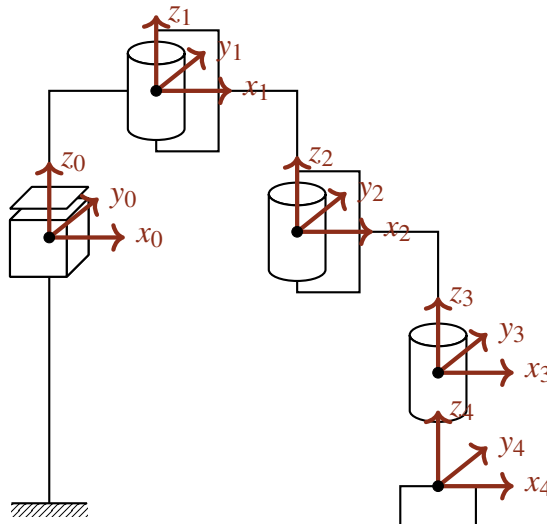


Figure 3.2: SCARA manipulator with the considered axis labelled at each joint origin.

Top View

The top view of the SCARA provides a clear visualisation of its planar geometry and arrangement of the rotational joints. This perspective, shown in Figure 3.3, is used in Subsection 3.1.3 to study the inverse kinematics of the manipulator.

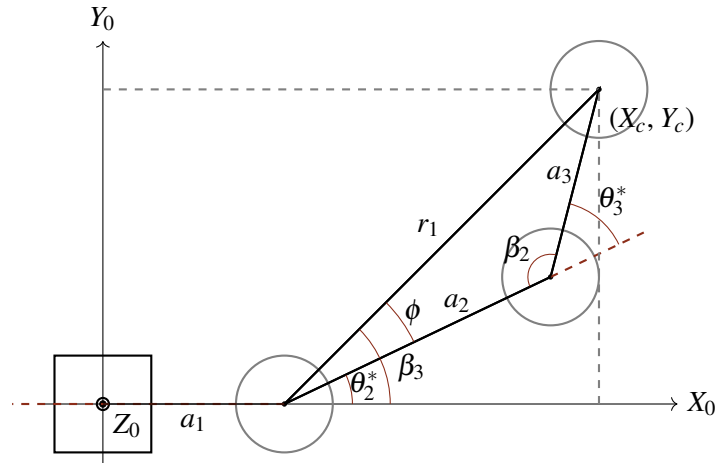


Figure 3.3: Top view of the manipulator. Circles represent revolute joints, and the square represents the prismatic joint.

Lateral View

The lateral view, however, is also used in the inverse kinematics study and is used to determine the value of d_1 as shown in Figure 3.4:

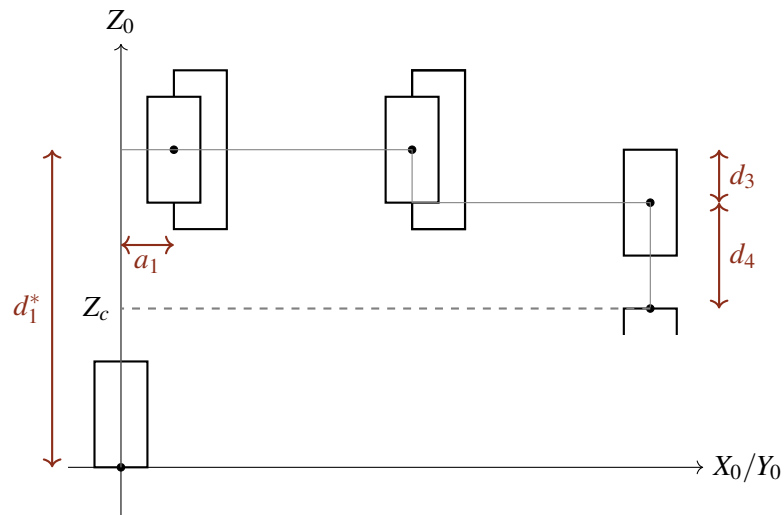


Figure 3.4: Lateral view of the manipulator.

3.1.2 Forward Kinematics

The direct kinematics involves obtaining the position of the end effector given the joint parameters [7]. With the standard DH convention, the homogeneous transformation matrices are used to represent the pose (position and orientation) [35] for each joint, and the final transformation matrix H_0^4 is obtained by sequential multiplication of the transformations defined by Equation 3.1. The Table 3.2 shows the established DH parameters. Finally, the direct kinematics are solved by multiplying the homogeneous transformation matrices from Equation 3.3 to Equation 3.6 to obtain H_0^4 as Equation 3.7.

Table 3.1: Symbol Definitions

Symbol	Unit	Definition
α_i	rad	Angle from z_{i-1} to z_i measured along x_{i-1}
a_i	m	Distance from z_{i-1} to z_i measured along x_{i-1}
d_i	m	Distance from x_{i-1} to x_i measured along z_i
θ_i	rad	Angle from x_{i-1} to x_i measured along z_i
H_j^i	–	Transformation matrix defining frame j relative to i
R_j^i	–	Rotation matrix from frame j to frame i
d_j^i	m	Displacement vector from frame j to frame i
c_i	–	$\cos(\theta_i)$
c_{i+j}	–	$\cos(\theta_i + \theta_j)$
s_{i+j}	–	$\sin(\theta_i + \theta_j)$
x	m	x component of end-effector position relative to base frame
y	m	y component of end-effector position relative to base frame
z	m	z component of end-effector position relative to base frame

Table 3.2: SCARA DH Parameters

Joint i	$a_{i-1}(\text{m})$	$\alpha_{i-1}(\text{rad})$	$d_i(\text{m})$	$\theta_i(\text{rad})$
1	$a_1 = 0.05$	0	d_1^*	0
2	$a_2 = 0.10$	0	0	θ_2^*
3	$a_3 = 0.10$	0	$d_3 = 0.06$	θ_3^*
4	0	0	$d_4 = 0.05$	θ_4^*

$$H_i^j = \begin{bmatrix} R_i^j & d_i^j \\ 0 & 1 \end{bmatrix} \quad (3.1)$$

$$H_0^4 = H_0^1 H_1^2 H_2^3 H_3^4, \quad (3.2)$$

$$H_0^1 = \begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$H_1^2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

$$H_2^3 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3 c_3 \\ s_3 & c_3 & 0 & a_3 s_3 \\ 0 & 0 & -1 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

$$H_3^4 = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & -d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

$$H_0^4 = \begin{bmatrix} c_4 c_{2+3} - s_4 s_{2+3} & -s_4 c_{2+3} - c_4 s_{2+3} & 0 & a_1 + a_2 c_2 + a_3 c_{2+3} \\ c_4 s_{2+3} + s_4 c_{2+3} & -s_4 s_{2+3} + c_4 c_{2+3} & 0 & a_2 s_2 + a_3 s_{2+3} \\ 0 & 0 & -1 & d_1^* - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

With Equation 3.7, the direct kinematics is obtained as shown in Equation 3.8.

$$d_0^4 = \begin{bmatrix} x_0^4 \\ y_0^4 \\ z_0^4 \end{bmatrix} = \begin{bmatrix} a_1 + a_2 c_2 + a_3 c_{2+3} \\ a_2 s_2 + a_3 s_{2+3} \\ d_1^* - d_3 - d_4 \end{bmatrix} \quad (3.8)$$

Finally, the missing $\theta_{gripper}$ is obtained as shown in Equation 3.9, from Figure 3.3 as follows:

$$\theta_{gripper} = \theta_2^* + \theta_3^* + \theta_4^* \quad (3.9)$$

To sum up, the forward kinematics equations provide a direct relationship between the joint variables and the position and orientation of the end-effector in the Cartesian space. This is essential for generating trajectories, planning motion and validating the system with accurate knowledge of the gripper's location. Besides this, understanding these relationships is fundamental before addressing the inverse kinematics. The following equations define the position of the end-effector:

$$x = 0.05 + 0.10\cos(\theta_2^*) + 0.10\cos(\theta_1^* + \theta_2^*) \quad (3.10)$$

$$y = 0.10\sin(\theta_1^*) + 0.10\sin(\theta_2^* + \theta_3^*) \quad (3.11)$$

$$z = d_1^* - 0.11 \quad (3.12)$$

$$\theta_{\text{gripper}} = \theta_2^* + \theta_3^* + \theta_4^* \quad (3.13)$$

3.1.3 Inverse Kinematics

A geometric approach is used to study the inverse kinematics of the robot. The position and orientation of the end-effector are determined by the joint variables [7]. Considering an end-effector located in a point P_c with coordinates $(X_c \ Y_c \ Z_c)$, and the top view from [Figure 3.3](#), the following equations can be defined to find both values of θ_3^* ($\theta_{3_{up}}^*$ and $\theta_{3_{down}}^*$) and θ_2^* ($\theta_{2_{up}}^*$ and $\theta_{2_{down}}^*$):

$$r_1 = \sqrt{(x_c - a_1)^2 + (y_c)^2} \quad (3.14)$$

$$\beta_2 = \arccos\left(\frac{r_1^2 - a_2^2 - a_3^2}{-2 \cdot a_2 \cdot a_3}\right) \quad (3.15)$$

$$\begin{aligned} \theta_{3_{up}}^* &= \pi - \beta_2 \\ \theta_{3_{down}}^* &= -\theta_{3_{up}}^* \end{aligned} \quad (3.16)$$

$$\beta_3 = \text{atan2}(x_c - a_1, y_c) \quad (3.17)$$

$$\phi_1 = \arccos\left(\frac{a_2^2 + r_1^2 - a_3^2}{2 \cdot a_2 \cdot r_1}\right) \quad (3.18)$$

$$\begin{aligned} \theta_{2_{up}}^* &= \beta_3 - \phi_1 \\ \theta_{2_{down}}^* &= \beta_3 - (-\phi_1) \end{aligned} \quad (3.19)$$

To conclude this study, two solutions were obtained for each end-effector's position. The first one corresponds to a solution where the elbow is up, defined with $\theta_{2_{up}}^*$ and $\theta_{3_{up}}^*$, and another with the elbow down, defined with $\theta_{2_{down}}^*$ and $\theta_{3_{down}}^*$. The value of d_1^* is known with [Equation 3.8](#) and θ_4^* from [Equation 3.10](#) as shown bellow:

$$d_1^* = z_c + d_3 + d_4 \quad (3.20)$$

$$\theta_4^* = \theta_{\text{gripper}} - \theta_2^* - \theta_3^* \quad (3.21)$$

To summarise and conclude this subsection, the following equations were obtained:

Table 3.3: Inverse Kinematics Solutions for SCARA Robot

Variable	Elbow-Up	Elbow-Down
d_1^*	$z_c + d_3 + d_4$	
θ_2^*	$\beta_3 - \phi_1$	$\beta_3 + \phi_1$
θ_3^*	$\pi - \beta_2$	$\beta_2 - \pi$
θ_4^*	$\theta_{\text{gripper}} - \theta_2^* - \theta_3^*$	

To interpret Table 3.3, d_1^* is the same in both elbow-up and elbow-down solutions. Then, each configuration has different θ_2^* and θ_3^* . Finally, θ_4^* depends on the previously obtained θ_2^* and θ_3^* .

3.1.4 Jacobian Matrix

Besides the position, the velocity of the manipulator is one of the most important quantities in the analysis and control of robot motion [7]. This velocity is obtained with the Jacobian matrix, which relates the robot's joints' velocities to the end-effector's linear and angular velocities.

The Jacobian is obtained by differentiating the forward kinematics equations presented in Subsection 3.1.2, concerning each joint parameter. The following equations show the steps to get the Jacobian Matrix:

Considering the vector of joint variables as:

$$q = \begin{bmatrix} d_1^* \\ \theta_2^* \\ \theta_3^* \\ \theta_4^* \end{bmatrix} \quad (3.22)$$

The Jacobian matrix $J(q)$ is the partial derivative of the end-effector pose concerning each joint variable:

$$\dot{x} = J(q) \dot{q} \quad (3.23)$$

Where $\dot{x}$ is the velocity vector of the end-effector:

$$\dot{x} = \begin{bmatrix} \dot{x}_0^4 \\ \dot{y}_0^4 \\ \dot{z}_0^4 \\ \dot{\theta}_{\text{gripper}} \end{bmatrix} \quad (3.24)$$

$$J(q) = \begin{bmatrix} \frac{\partial x_0^4}{\partial d_1^*} & \frac{\partial x_0^4}{\partial \theta_2^*} & \frac{\partial x_0^4}{\partial \theta_3^*} & \frac{\partial x_0^4}{\partial \theta_4^*} \\ \frac{\partial y_0^4}{\partial d_1^*} & \frac{\partial y_0^4}{\partial \theta_2^*} & \frac{\partial y_0^4}{\partial \theta_3^*} & \frac{\partial y_0^4}{\partial \theta_4^*} \\ \frac{\partial z_0^4}{\partial d_1^*} & \frac{\partial z_0^4}{\partial \theta_2^*} & \frac{\partial z_0^4}{\partial \theta_3^*} & \frac{\partial z_0^4}{\partial \theta_4^*} \\ \frac{\partial \theta_{gripper}}{\partial d_1^*} & \frac{\partial \theta_{gripper}}{\partial \theta_2^*} & \frac{\partial \theta_{gripper}}{\partial \theta_3^*} & \frac{\partial \theta_{gripper}}{\partial \theta_4^*} \end{bmatrix} \quad (3.25)$$

$$J(q) = \begin{bmatrix} 0 & -a_2 \sin(\theta_2^*) - a_3 \sin(\theta_2^* + \theta_3^*) & -a_3 \sin(\theta_2^* + \theta_3^*) & 0 \\ 0 & a_2 \cos(\theta_2^*) + a_3 \cos(\theta_2^* + \theta_3^*) & a_3 \cos(\theta_2^* + \theta_3^*) & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad (3.26)$$

This Jacobian enables mapping joint velocities to end-effector velocities and can also be used to perform inverse differential kinematics, singularity analysis, and control in task space.

$$\begin{bmatrix} \dot{x}_0^4 \\ \dot{y}_0^4 \\ \dot{z}_0^4 \\ \dot{\theta}_{gripper} \end{bmatrix} = \begin{bmatrix} 0 & -a_2 \sin(\theta_2^*) - a_3 \sin(\theta_2^* + \theta_3^*) & -a_3 \sin(\theta_2^* + \theta_3^*) & 0 \\ 0 & a_2 \cos(\theta_2^*) + a_3 \cos(\theta_2^* + \theta_3^*) & a_3 \cos(\theta_2^* + \theta_3^*) & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{d}_1^* \\ \dot{\theta}_2^* \\ \dot{\theta}_3^* \\ \dot{\theta}_4^* \end{bmatrix} \quad (3.27)$$

Singularities

Singularities represent configurations from which specific directions of motion may be unattainable. This includes end-effector velocities corresponding to unbounded joint velocities, unreachable points or multiple solutions for the inverse kinematics, and among other phenomena [7]. These singularities are determined using the determinant of the Jacobian matrix:

$$\det(J(q)) = 0 \quad (3.28)$$

$$\det(J(q)) = a_3 \sin(\theta_2^* + \theta_3^*) (a_2 \cos(\theta_2^*) + a_3 \cos(\theta_2^* + \theta_3^*)) \quad (3.29)$$

Singularities occur when:

$$\det(J(q)) = 0 \Rightarrow \begin{cases} \sin(\theta_2^* + \theta_3^*) = 0 \\ a_2 \cos(\theta_2^*) + a_3 \cos(\theta_2^* + \theta_3^*) = 0 \end{cases} \quad (3.30)$$

$$\Rightarrow \begin{cases} \sin(\theta_2^* + \theta_3^*) = 0 \Rightarrow \theta_2^* + \theta_3^* = k\pi, \quad k \in \mathbb{Z} \\ a_2 \cos(\theta_2^*) + a_3 \cos(\theta_2^* + \theta_3^*) = 0 \end{cases} \quad (3.31)$$

In the first case, when k is any integer, the second and third joints are either fully extended (or folded back on each other), causing the two links to align collinearly. The second case, however, is when the manipulator loses one degree of freedom in the horizontal plane, making it a workspace boundary.

3.1.5 Workspace Analysis

The workspace of the SCARA manipulator represents the set of all positions that the end-effector can reach, considering joint constraints and mechanical limitations. The following Figure 3.5a and Figure 3.5b include both planar and volumetric projections of the reachable area:

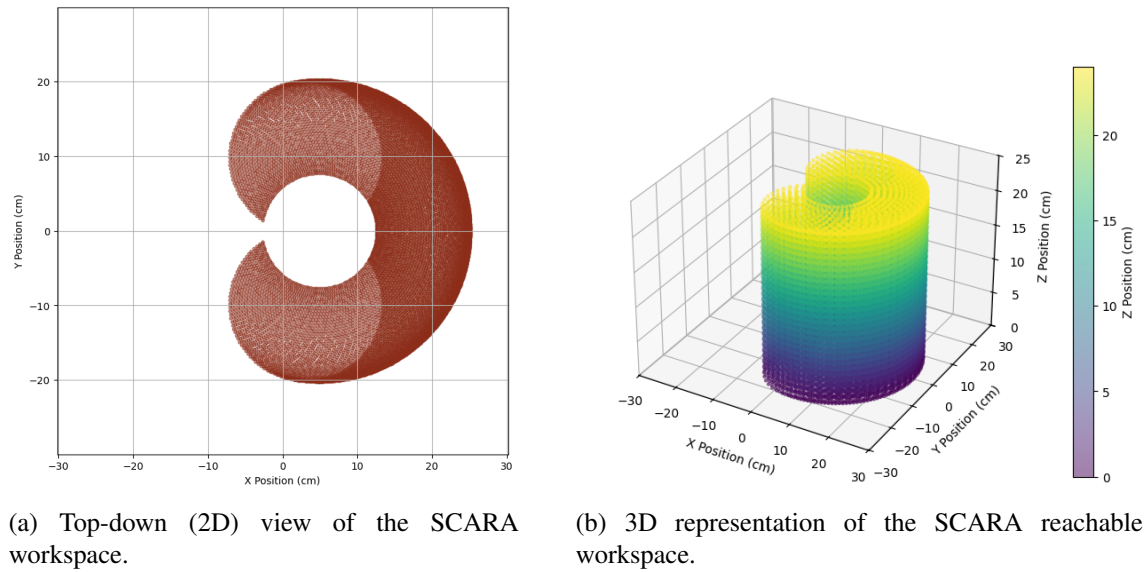


Figure 3.5: Reachable workspace of the SCARA robot in both 2D and 3D representations.

A grid-based approach created a point cloud representing the reachable space. Using this method, the estimated 2D workspace area (plane XY) is 900.0 cm^2 , and the estimated workspace volume is 7760.0 cm^3 .

3.2 Hardware Components

Implementing a 4 DOF manipulator with a gripper involves integrating several key components essential for the system's overall functionality. The following section analyses the hardware elements while examining their characteristics, dependencies and limitations.

3.2.1 Motors and Drivers

Two different motors were used in this work. Their specifications [36, 37] are in the Table 3.4 below:

In total, five motors are used. 3 NEMA 17 are used in joints 1, 2 and 3. Then, joint four and the claw have a NEMA 11. The NEMA 17 provides a higher torque output to overcome the

Table 3.4: Comparison of Stepper Motor Specifications

Feature	SY28STH32-0674A - NEMA 11	SM42HT38-1684A - NEMA 17
Frame Size (mm)	28 × 28	42 × 42
Body Length (mm)	32	38
Motor Type	2-phase hybrid stepper	2-phase hybrid stepper
Current / Phase (A)	0.67	1.68
Resistance / Phase (Ω)	5.6	1.65
Inductance / Phase (mH)	3.2	3.2
Holding Torque (oz-in)	6	50
Number of Leads	4	4
Shaft Type	Single shaft	Single shaft
Weight (kg)	0.12	0.28

cumulative inertia of the manipulator's arm segments and payload, if any. Then, at the robot's end, the lighter and smaller NEMA 11 were used to control the rotation of the end-effector and its opening and closing.

To control the motors, the A4988 stepper motor driver was used:

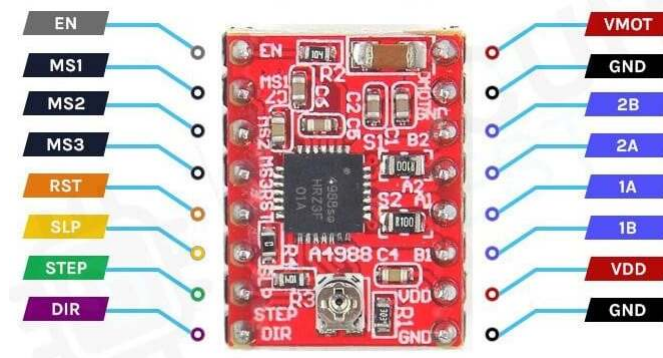


Figure 3.6: A4988 device. Adapted from [38].

Table 3.5: Key Characteristics of the A4988 Stepper Motor Driver [39]

Characteristic	Description
Driver Type	Complete microstepping driver with built-in translator
Step Modes Supported	Full-, half-, quarter-, eighth-, sixteenth-step
Voltage Range	Up to 35 V
Current Capacity	Up to ± 2 A per phase

One of these drivers controls all system motors, shown in Figure 3.6. Every motor besides the prismatic actuator was configured to run in the 1/16 step to obtain more precision. In other words, by following the Table 3.6, MS1, MS2 and MS3 were set to high by being connected to 5 V. Choosing this resolution step decreases the torque of each motor by giving the motor controlled

profiles of current to each phase. However, the motion of the movements is smoother and has less vibration.

Table 3.6: Microstepping Resolution Truth Table for A4988 [39]

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Before using the driver, a calibration is necessary to adjust the current used to power each motor. The steps are:

- Powering the board without connecting the motor.
- With a multimeter, measure the reference voltage, which controls the current limit.
- Calculate the desired V_{ref} , based on the motor's rated current with:

$$V_{\text{ref}} = I_{\text{max}} \times 0.8$$

- Use a small screwdriver to adjust the potentiometer until V_{ref} is reached.

After calibrating, a motor can be controlled by sending a signal to the STEP pin. A high-frequency signal like PWM makes the motor move fast, and a low-frequency signal makes the motor slow. A signal is sent to the DIR or direction pin to choose the direction of rotation. A high value makes the motor move in one direction, and a low value in the other.

3.2.2 Encoders and I2C Multiplexer

The position feedback system of the SCARA relies on AS5600 magnetic rotary encoders, which provide absolute angular position for each joint. This device, shown in Figure 3.7a, utilises Hall effect sensors to detect the position of a diametrically magnetised magnet, while having a 12-bit resolution, or 4096 positions per rotation (0.087° resolution). The AS5600's analogue and digital outputs are accessible through PWM and I2C interfaces.

To communicate with the microcontroller, the I2C interface is used. However, the implementation of all four encoders was not straightforward. All devices share the same fixed I2C address, making simultaneous communication impossible on a single bus. To address this conflict, an I2C multiplexer called TCA9548A is used.

The multiplexer enables communication with up to eight identical I2C devices sharing the same address. This is possible since the master sends an 8-bit message to the multiplexer indicating

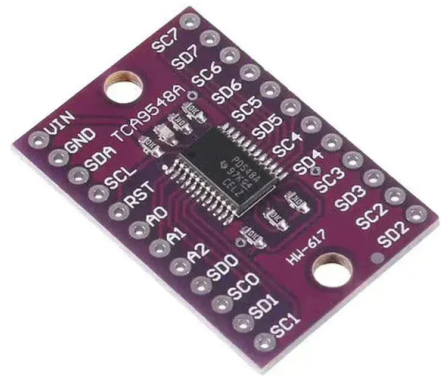
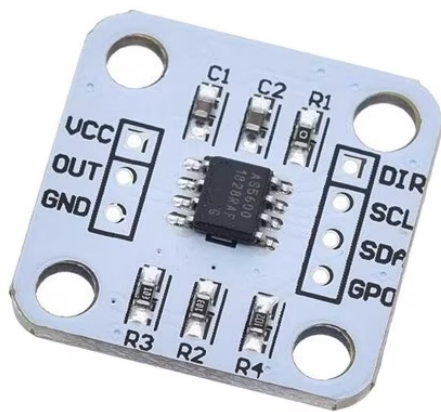


Figure 3.7: Encoder and multiplexer used.

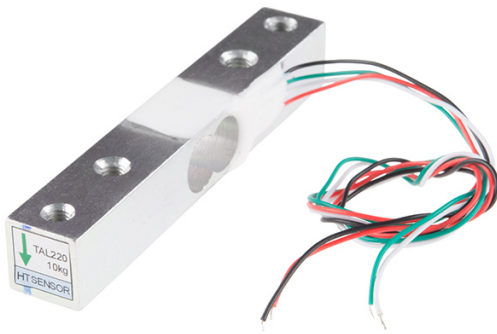
what channels will be active [42]. By setting one channel at a time, the system can acquire data from all the encoders despite having the same I2C address.

3.2.3 Load Cell and Amplifier

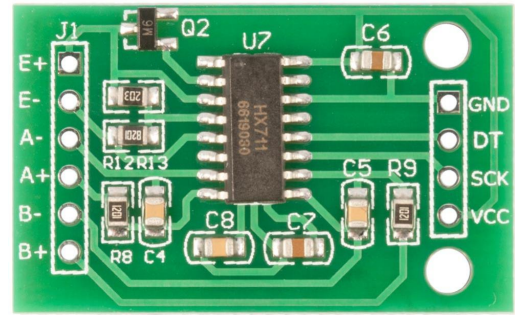
The system used two load cells to measure the loads of force applied in the end-effector. This device converts the mechanical force into a measurable electrical signal. This signal, however, is typically in the millivolt range and cannot be read by the microcontroller. For this reason, the Analogue Digital Converter (ADC) with integrated amplifier HX711 was used for each load cell.

The HX711 is a 24-bit precision ADC for weight scale and force measurement applications. It provides programmable gain options (32, 64, or 128) and supports two differential input channels [43], making it well-suited for low-voltage signals produced by load cells. The HX711 includes an internal low-noise amplifier and can interface directly with microcontrollers via a simple two-wire serial communication protocol (DATA and CLOCK lines).

The Figure 3.8a and Figure 3.8b show the load cell and its amplifier, respectively:



(a) Load cell from [44].

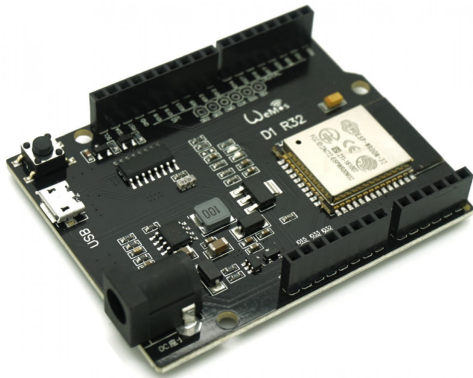


(b) HX711 amplifier [45].

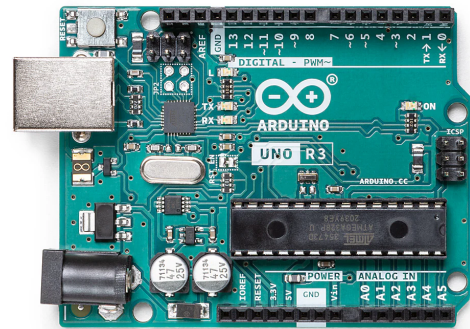
Figure 3.8: Load cell and its amplifier.

3.2.4 ESP32 Microcontroller and CNC Shield Integration

At the core of the SCARA manipulator control system is the ESP32 WeMos D1 R32 microcontroller, illustrated in Figure 3.9a. This microcontroller was chosen for its compatibility with the CNC Shield v3, as it shares the Arduino Uno form factor and pinout, shown in Figure 3.9b.



(a) ESP32 WeMos D1 microcontroller. Adapted from [46].



(b) Arduino Uno microcontroller from [47].

Figure 3.9: Compatible microcontroller with the CNC Shield V3.

Rather than relying on the Arduino IDE, the microcontroller firmware was developed using the official development environment for the ESP32 called Espressif IoT Development Framework (ESP-IDF). This framework provides low-level access to the ESP32's hardware features and supports real-time task scheduling via FreeRTOS.

Mounted on top of the microcontroller is the CNC Shield v3, an open-source interface board that facilitates using step motors, providing slots for four drivers. This component, shown in Figure 3.10, reduces the wiring used while providing accessibility to the board's pins below it.

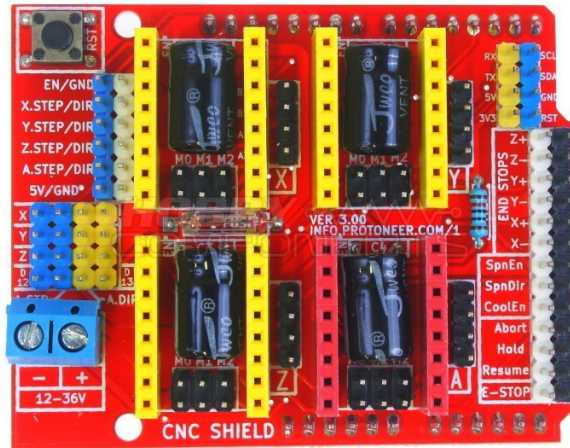


Figure 3.10: CNC Shield V3 from [48].

As five motors were used in this work, an additional shield was used to power the last driver, shown in Figure 3.11.

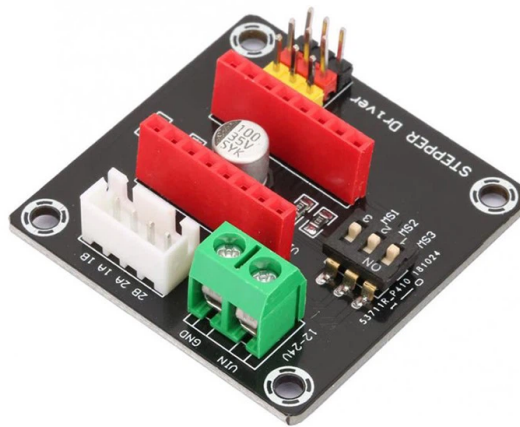


Figure 3.11: 3D Printer 42CH stepper motor driver shield expansion board from [49].

3.3 Summary

This chapter has comprehensively analysed the SCARA manipulator's mechanical architecture and hardware implementation, establishing the foundational framework for the educational robotics platform. The kinematic modelling demonstrates the robot's 4 DOF configuration, with mathematical formulations for forward kinematics relating the direct relationships between joint variables and end-effector positioning. In inverse kinematics, however, a geometric approach is used successfully, providing dual solutions for each target position through elbow-up and elbow-down

configurations. The derived Jacobian matrix enables velocity analysis and singularity identification, revealing critical workspace limitations when joints align collinearly or reach boundary conditions.

The hardware integration demonstrates a well-balanced approach to component selection and system design. The strategic use of NEMA 17 motors for high-torque applications in the primary joints, complemented by lighter NEMA 11 motors for end-effector control, optimises both performance and mechanical efficiency. Implementing AS5600 magnetic encoders with TCA9548A I2C multiplexing provides precise angular feedback across all revolute joints and claws. At the same time, integrating load cells with HX711 amplifiers enables force-sensing capabilities essential for educational demonstrations.

Chapter 4

Software Architecture

The software architecture of the manipulator represents a sophisticated integration of real-time control systems, communication protocols, and kinematic algorithms designed to achieve precise multi-axis coordination. It is built using the ESP-IDF framework and FreeRTOS real-time operating system.

This chapter examines the core software components, including signal generation mechanisms, real-time task management and network communication protocols, while explaining the implementation of concurrent tasks for motor control, sensor data acquisition and user interface communication.

4.1 System Architecture Overview

The manipulator's software architecture implements a layered, modular design that decomposes the complex robotic control problem into manageable and interconnected subsystems. This section explains the composition of the system, as well as the tasks used to control it.

4.1.1 Real-Time Task Management with FreeRTOS

The manipulator's software architecture leverages FreeRTOS to achieve deterministic, concurrent execution of multiple control tasks. It provides essential real-time capabilities, including task scheduling, inter-task communication, and resource synchronisation, critical for maintaining precise timing in robotic control applications.

4.1.2 Task Architecture and Priority Management

The system implements a hierarchical task structure where each hardware component operates within its dedicated task context. This modular approach ensures system reliability and maintainability while providing predictable real-time behaviour. To create the tasks `xTaskCreate()` function from "`freertos/task.h`" library is used. Its parameters and the pattern chosen to create all the tasks of the system are covered in the [Listing 4.1](#):

```

1 // xTaskCreate example with parameters explained
  xTaskCreate(
3     vTaskCode,           // task function
    "NAME",               // task name
5     STACK_SIZE,         // stack depth
    &ucParameterToPass,   // task input parameter
7     tskIDLE_PRIORITY,   // task priority
    &xHandle              // task handle
9 );
// Motor control tasks with priority 5
11 xTaskCreate(motor_control_task, "motor_ctrl_x", 4096, &motor_x, 5, NULL);
  xTaskCreate(motor_control_task, "motor_ctrl_y", 4096, &motor_y, 5, NULL);
13 xTaskCreate(motor_control_task, "motor_ctrl_z", 4096, &motor_z, 5, NULL);
  xTaskCreate(motor_control_task, "motor_ctrl_a", 4096, &motor_a, 5, NULL);
15 xTaskCreate(motor_control_task, "motor_ctrl_b", 4096, &motor_b, 5, NULL);

17 // Encoder reading tasks with priority 5
  xTaskCreate(encoder_task, "encoder0_task", 4096, &encoder_0, 5, NULL);
19 xTaskCreate(encoder_task, "encoder1_task", 4096, &encoder_1, 5, NULL);
  xTaskCreate(encoder_task, "encoder2_task", 4096, &encoder_2, 5, NULL);
21 xTaskCreate(encoder_task, "encoder3_task", 4096, &encoder_3, 5, NULL);

```

Listing 4.1: FreeRTOS task creation patters. Example is taken from [50].

A function defining its work is implemented to create the task. Like the examples in [Listing 4.1](#), the functions `motor_control_task()` and `encoder_task` are called. They also receive an input parameter as a struct with the data needed for the specific task.

Each task is allocated 4096 bytes of stack memory and assigned priority level 5, ensuring execution without starvation. This could lead to tasks not being executed or preempted when a higher-priority task interrupts a lower-priority task.

4.1.3 Hierarchical System Design

The SCARA manipulator software architecture implements a three-layer hierarchical design that promotes modularity and maintainability. This design was chosen to make each layer with specific responsibilities, allowing lower layers to handle hardware-specific operations and higher layers to handle application logic and system coordination. The following [Figure 4.1](#) shows the approach used in this work:

The three-layer architecture consists of: Layer 1 (ESP32 FreeRTOS Controller) managing tasks scheduling and resource allocation; Layer 2 (Subsystems) contains the four independent modules for network communication, motion control, kinematics processing and sensor data acquisition; Finally, layer 3 encompasses the hardware abstraction through I2C drivers, GPIO drivers, network protocol and MCPWM Application Programming Interface (API) to interact with the hardware components.

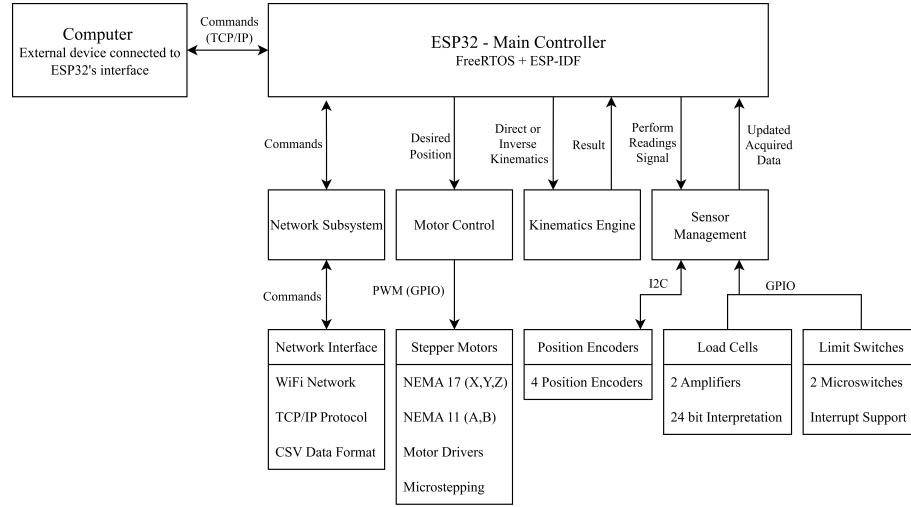


Figure 4.1: Hierarchical system block diagram.

4.1.4 Naming Relationship between Software and Hardware

The software architecture adheres to the naming convention between physical hardware components and their software representations. Naming each motor originates from the motor driver shield from [Subsection 3.2.4](#), where it defines the motors: X, Y, Z and A. Each corresponds to its dedicated motor driver with its associated step and direction General-Purpose Input/Output (GPIO) pins. Then, motor B was introduced when extending the system to have a fifth motor to control the claws.

The motor-to-joint mapping follows logical kinematic relationships: motor X controls the linear prismatic joint d_1^* along the vertical axis, motor Y manages the first rotational joint θ_2^* at the manipulator base, motor Z operates the second rotational joint θ_3^* at the elbow connection, motor A controls the wrist rotation θ_4^* , and motor B handles the gripper opening and closing mechanism.

Notably, motor X operates without encoder feedback as the linear joint d_1^* utilises step counting for position estimation, while the remaining four motors implement closed-loop control through their respective encoder. The relationship between the drivers, motors and encoders is shown in [Figure 4.2](#). A more detailed association between all the system hardware can be found in [Appendix A](#), with its components and protocols in [Appendix B](#).

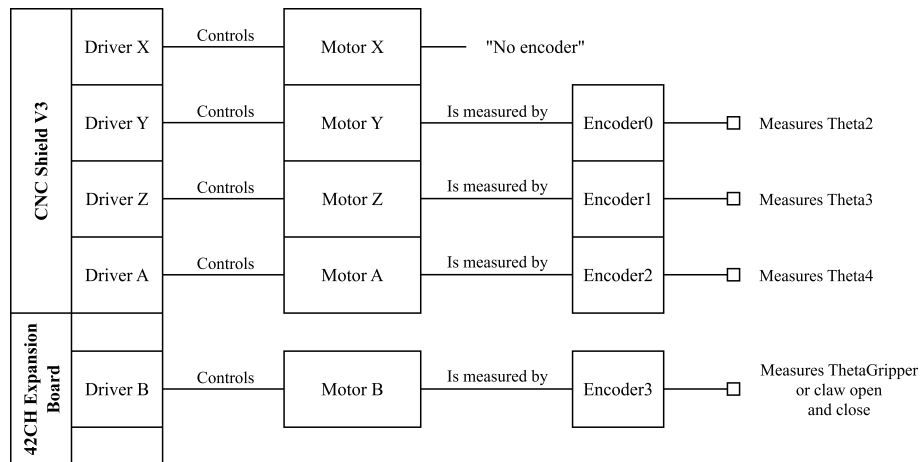


Figure 4.2: Relation between drivers, motors and encoders.

4.2 One-Shot Functions

Before discussing the subsystems and their periodic tasks, one-time functions present in the software are addressed. The function `init_scara()` shown in Listing 4.2 represents the collection of one-shot operations, meaning that these procedures are executed only once during system startup and are not repeated periodically. Its role is to prepare the essential subsystems of the SCARA manipulator before the FreeRTOS scheduler takes over. This includes configuring communication interfaces, I2C and GPIO pull-ups, initialising sensors such as encoders, load cells, and limit switches, and setting up the motor control subsystem. At the end of each function, the related tasks are initialised.

```

1 void init_scara() {
2     //initialise components
3     wifi_initialisation_task();           // network subsystem
4     motor_initialisation_task();          // motor control subsystem
5
6     encoder_initialisation_task();        // encoders sensors subsystem
7     hx711_initialisation_task();          // sensors loadcells subsystem
8     switch_initialisation_task();         // switches sensors subsystem
9
10    // calibration of loadcells + homing motors
11    calibration_initialization_func();
12
13    // run kinematics engine subsystem
14    xTaskCreate(kinematics_engine, "run kinematics engine", 4096, NULL, 5,
15                NULL);
16    return;
17 }
  
```

Listing 4.2: Initialisation of the components.

Besides initialising the components or starting tasks, the function `calibration_initialization_func()` has a vital role in calibrating the load cells and performing a homing procedure using the limit switch to establish a reference position and know its position even after restarting the microcontroller. Additionally, after the calibration procedures are completed, the kinematics engine starts running, allowing the position of the manipulator's joints to be determined.

4.3 Sensor Data Acquisition Subsystem

The sensor data acquisition subsystem of the manipulator is essential for providing real-time feedback of the external environment for closed-loop control and system monitoring. It manages multiple sensor types, ensuring reliable data collection and distribution across the system.

4.3.1 Subsystem Architecture

The sensor acquisition subsystem implements a multi-layered architecture handled by FreeRTOS. It consists of the following three primary sensor groups:

Position Sensors

As discussed in [Subsection 3.2.2](#), four magnetic encoders provide absolute angular position data for the joints θ_2^* , θ_3^* , θ_4^* , and $\theta_{gripper}^*$. These sensors communicate via the I2C protocol through the multiplexer, enabling access to all encoders on a single bus.

Force Feedback Sensors

Following [Subsection 3.2.3](#), two load cells provide force measurements in the gripper area to detect collisions with the environment. The 24-bit ADC components amplify the signal received from these load cells, enabling the detection of delicate bumps with objects.

Limit Detection Sensors

The limit switches are strategically placed at predefined reference positions on the manipulator. They serve as calibration points for the joints d_1^* , θ_2^* , and θ_3^* . The system establishes a known reference by slowly driving each joint until the corresponding switch is triggered, ensuring the manipulator can consistently recover the same end-effector position after restarting.

4.3.2 Tasks

Encoder Task Implementation

The encoder task provides continuous monitoring of rotational position using AS5600 magnetic encoders. The implementation addresses critical challenges in rotational measurement, including wraparound detection, accumulated step counting, and error handling for reliable long-term operation, shown in [Appendix C](#), with its diagram below in [Figure 4.3](#):

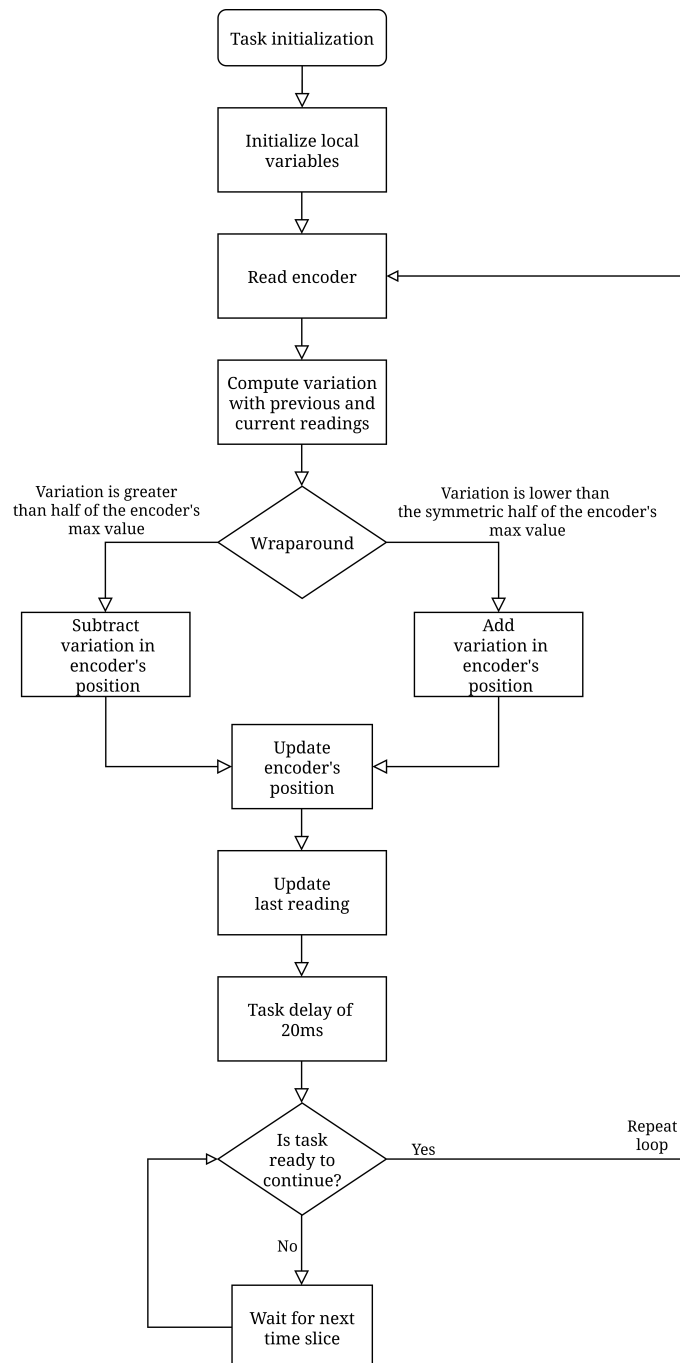


Figure 4.3: Flow diagram of the encoder task.

The encoder task operates at 20Hz (50ms intervals) to balance system responsiveness with computational efficiency. The wraparound detection algorithm ensures accurate position tracking across the 360° range by selecting the shortest rotational path between consecutive readings. This approach prevents accumulated errors that could occur when the encoder crosses the zero-point boundary.

Load Cell Data Acquisition

The HX711 load cell interface tasks demonstrate two approaches to analogue data acquisition: instantaneous readings and averaged measurements, shown in [Appendix C](#), with its diagram in [Figure 4.4](#):

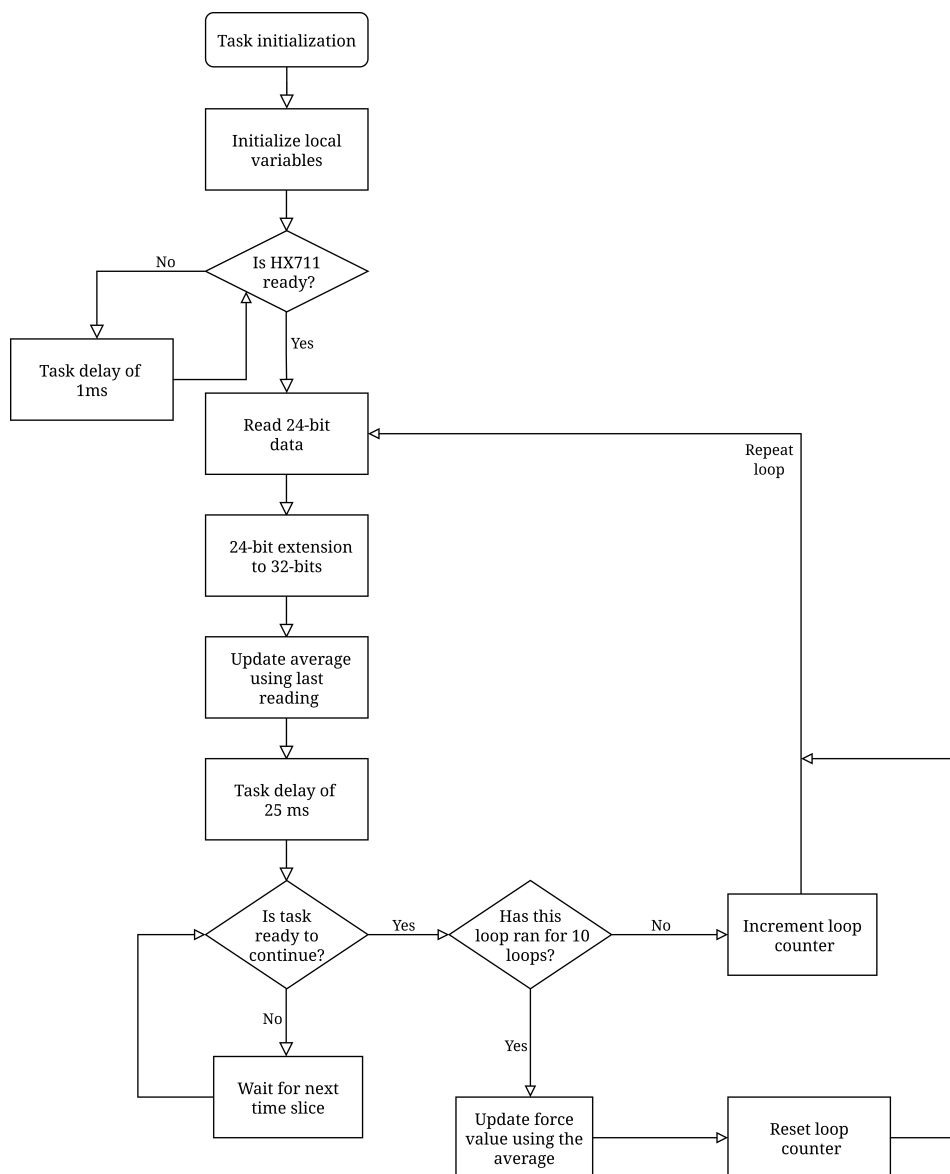


Figure 4.4: Flow diagram of the loadcell task.

The averaging implementation reduces noise in load cell measurements by collecting 10 samples and computing their arithmetic mean. This approach improves measurement stability while maintaining reasonable update rates for force monitoring applications.

Switch Monitoring Task

The switch monitoring task provides real-time status updates for limit switches in calibration and safety systems. Operating at 100Hz (10ms intervals), this task, shown in [Appendix E](#), ensures rapid detection of switch state changes. Its diagram is shown below in [Figure 4.5](#):

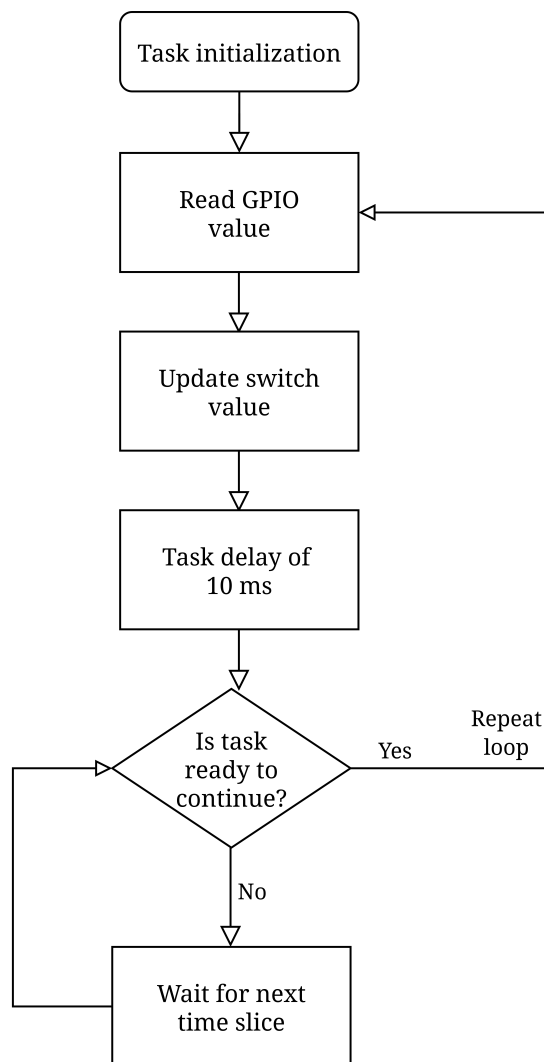


Figure 4.5: Flow diagram of the limit switch task.

The high-frequency monitoring ensures that switch state changes are detected within 10ms, providing adequate response time for safety systems and calibration procedures.

4.4 Motion Control Subsystem

The Motion Control Subsystem represents the actuator management layer of the manipulator, with precise movement through coordinated stepper motor control and real-time feedback. This subsystem integrates five independent motor control tasks running concurrently, each responsible for managing a specific degree of freedom. Each motor task implements its control strategy, including target position management, velocity profiling with configurable acceleration limits, and safety interlocks through limit switch integration. The subsystem maintains thread-safe communication with the sensor acquisition layer through mutex-protected shared memory structures, ensuring consistent data exchange for closed-loop position feedback.

4.4.1 Motor Control Task

The motor control task implements closed-loop position control using a discrete PID controller that maintains precise angular positioning for each joint. The controller operates with a 20 ms sampling period, providing responsive control while allowing sufficient time for encoder updates. This is shown in [Appendix F](#) with its flow diagram below in [Figure 4.6](#):

The PID controller contains the following features for robust operation:

- **Deadband Control:** A 2-step threshold prevents motor oscillation around the target position, reducing power consumption.
- **Integral Windup Protection:** Limits prevent the integral term from accumulating excessive values during significant or persistent errors.
- **Output Saturation:** Ensures motor frequency commands remain within safe operational limits.
- **Direction Logic:** Handles bidirectional motor control with support for inverted direction configurations.

4.4.2 PWM Control with MCPWM

The precise control of motors in the SCARA manipulator is achieved using the ESP32's MCPWM (Motor Control Pulse Width Modulation) peripheral. This hardware-based PWM generation offers better timing accuracy, lower CPU usage, and built-in motor control features like deadtime insertion and fault handling over basic GPIO.

PWM signals are generated via the `motor_create_pwm()` function inside the initialisation of the motors, which sets up the MCPWM's components: timer, operator, comparator, and generator. Configured with a 1 MHz resolution, the timer calculates the PWM period for the desired frequency. At the same time, the comparator ensures a 50% duty cycle, producing clean step signals for the stepper motors. Because signal generation is handled in hardware, the system maintains precise timing even under CPU load.

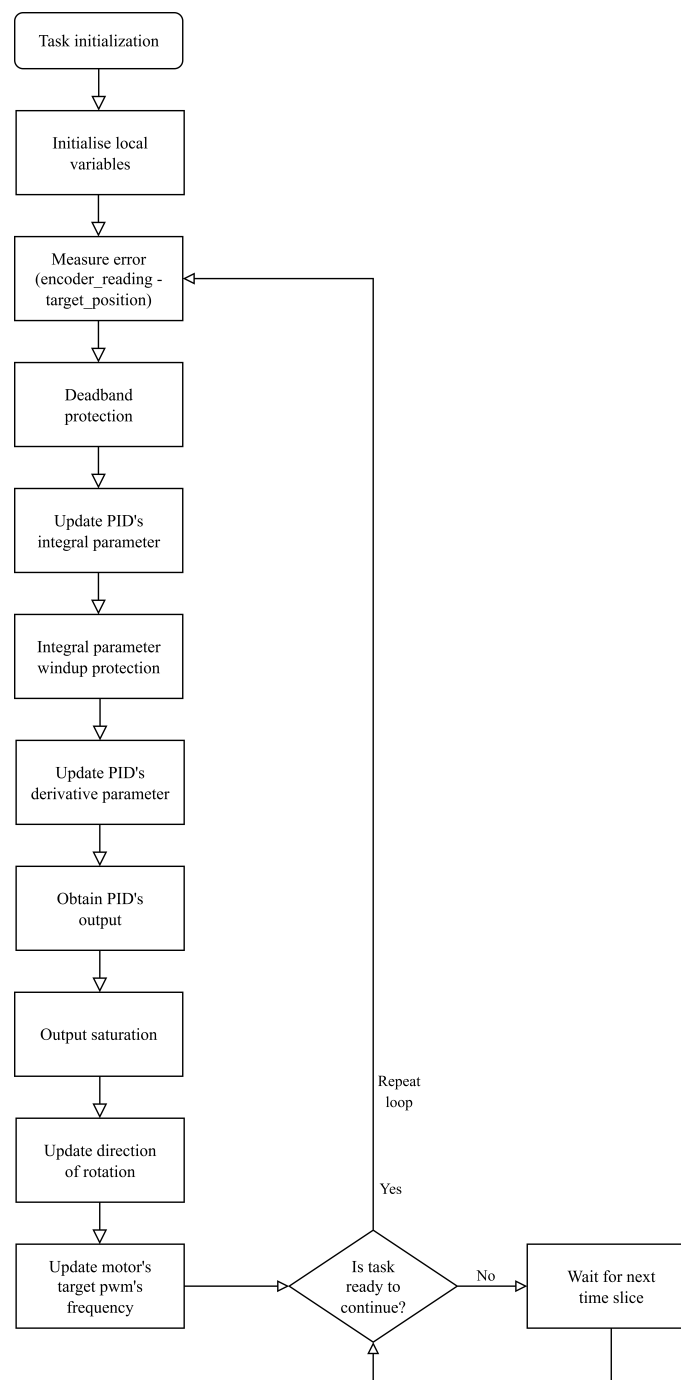


Figure 4.6: Flow diagram of the motor control task.

For smooth motion, dynamic frequency control is implemented using a callback function `motor_update_timer_cb()`. Its flow diagram is shown in Figure 4.7, and its function can be found in Appendix G. Its job is to gradually increase or decrease the PWM frequency based on set acceleration limits, preventing abrupt movements that could damage the motors. It also tracks the number of generated steps, stopping the motor automatically when the target is reached in the case of the prismatic actuator.

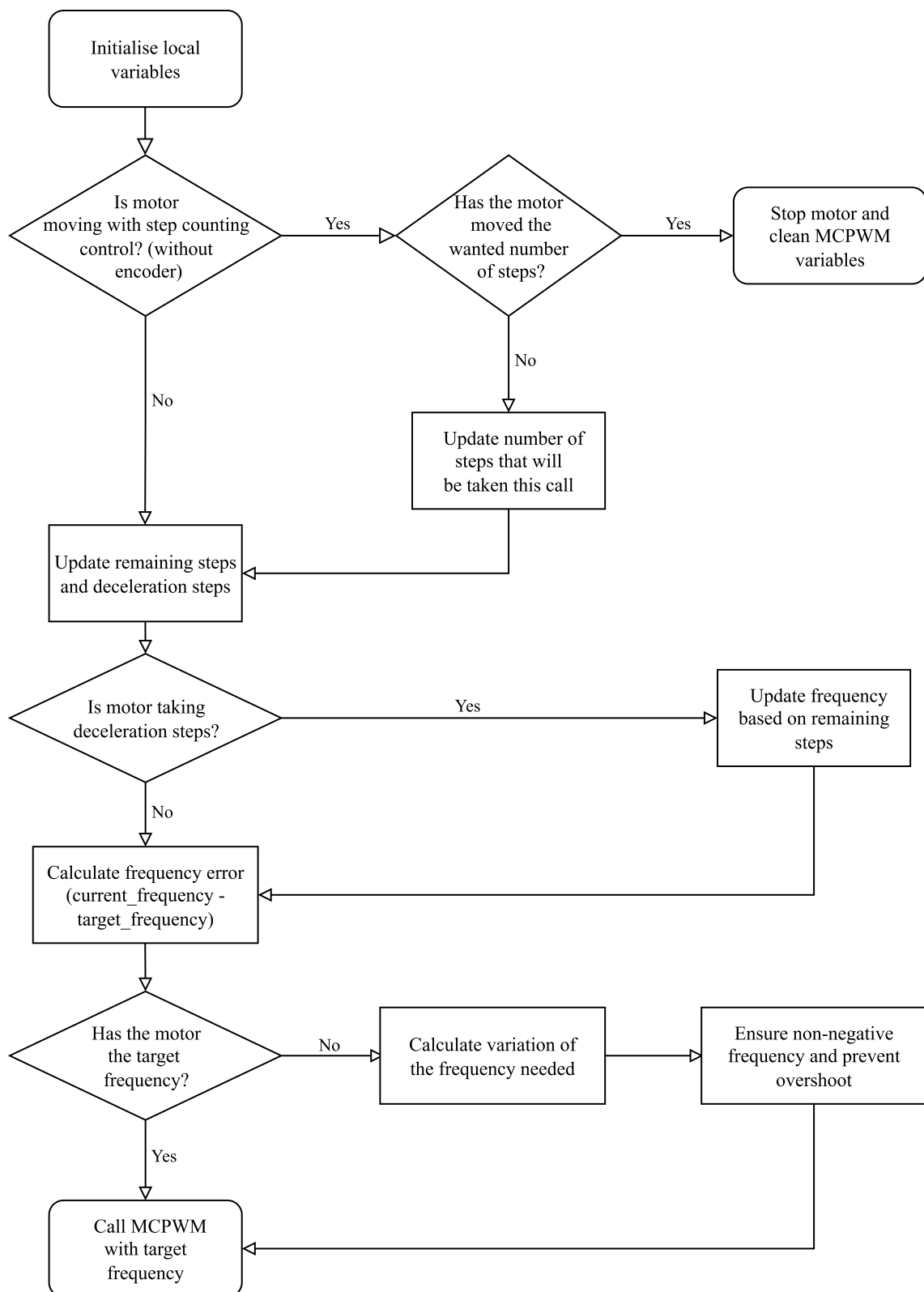


Figure 4.7: Flow diagram of the motor callback function.

4.5 Network Communication Subsystem

The Network communication subsystem establishes a bidirectional data exchange infrastructure that enables remote control and real-time monitoring of the SCARA through WiFi connectivity. It has a dual-protocol architecture combining an HTTP web server for user interface delivery and TCP socket communication for high-fidelity data streaming.

The subsystem operates through two primary FreeRTOS tasks: a TCP server task that handles incoming control commands and connection management, and a TCP sender task that provides continuous system status updates at 1-second intervals to connected clients.

In the end, this architecture supports both real-time control scenarios that require immediate response and monitor applications that demand continuous data visualisation, making it suitable for diverse educational scenarios ranging from basic positioning demonstrations to advanced control algorithm development.

4.5.1 Server-Side Logic

The server-side implementation provides a network service architecture that manages client connections, processes incoming commands, and delivers system status information through communication protocols.

The HTTP server component is initialised during system startup by establishing WiFi connectivity, configuring socket parameters including port binding (default port 3333) and connection queuing, and preparing the web interface delivery mechanism as illustrated in [Figure 4.8](#).

The server task operates in a continuous listening loop, accepting client connections with appropriate error handling and timeout configurations. After receiving client data, the server implements parsing mechanisms using formatted string scanning to extract eleven distinct parameters, including joint angles, Cartesian coordinates, and kinematic mode flags, with validation and error logging to ensure data integrity.

The sender task operates on a precise 1-second timer cycle, collecting current system state from all subsystems, including encoder positions, load cell readings, switch states, and computed kinematic values, then formatting this information into a standardised CSV string for transmission to connected clients.

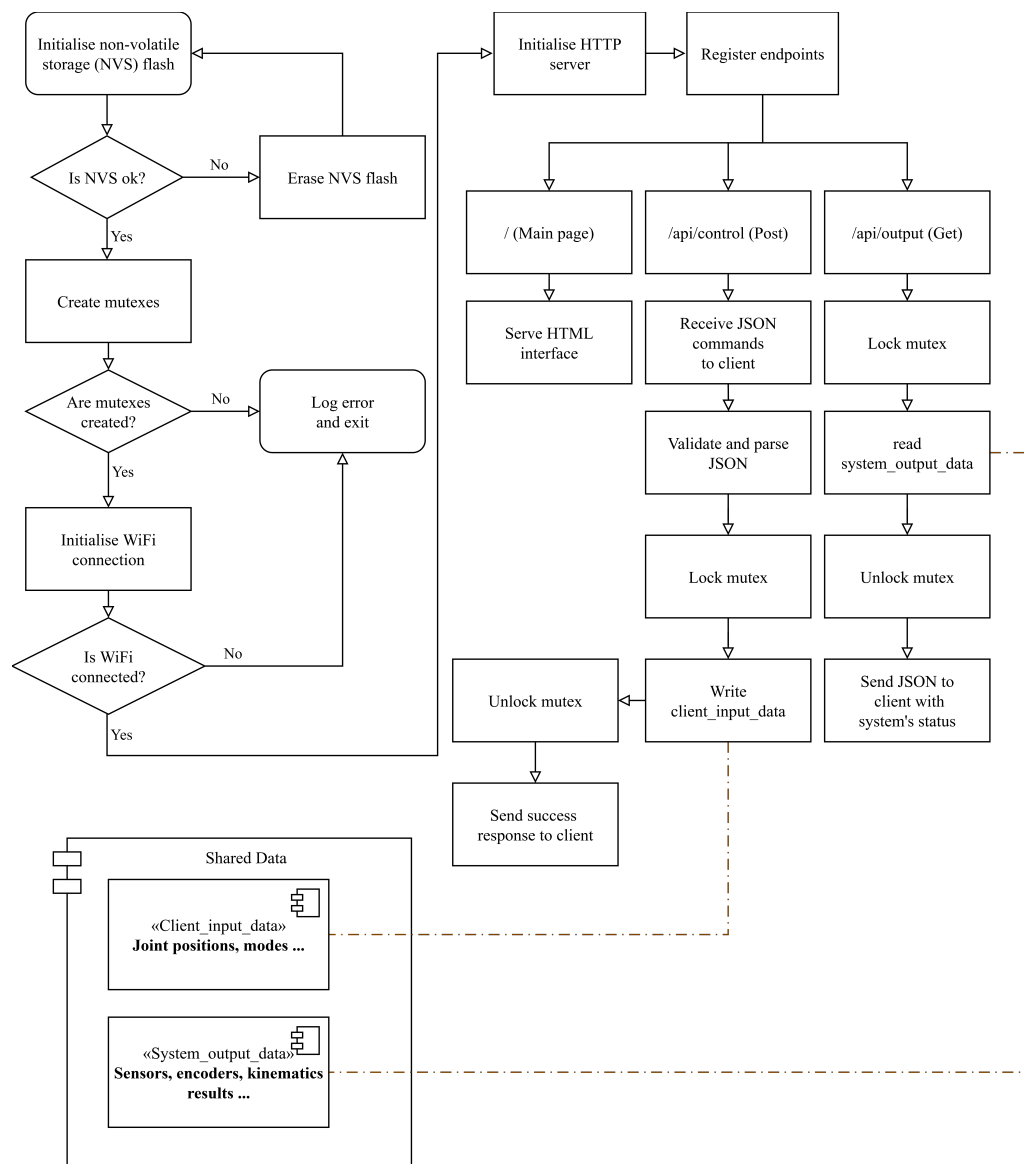


Figure 4.8: Flow diagram of the initialisation of the HTTP server.

4.5.2 Client-Side Logic

The client-side logic encompasses the user interaction from initial connection establishment through monitoring and control command transmission. It starts by establishing the ESP32's IP in a browser. Then, followed by an HTTP request to retrieve the user interface shown in [Section 5.4](#), the user is provided with direct and inverse kinematics control panels with appropriate input validation and limit checking. This is shown below in [Figure 4.9](#) and [Figure 4.10](#). The first figure shows the initial connection and interface request, followed by the second figure, where the client's input is processed and sent and the system's data monitor update.

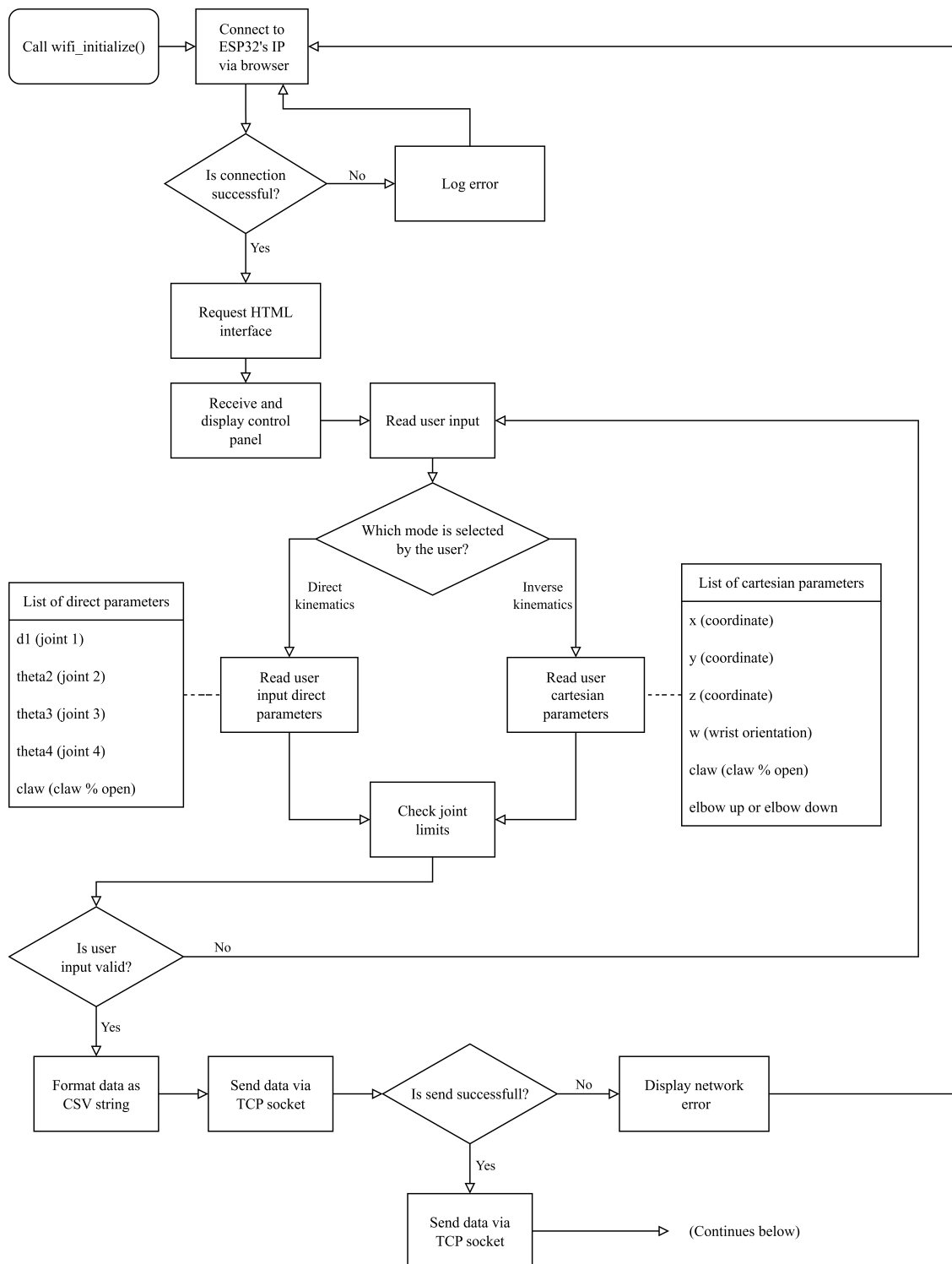


Figure 4.9: Part one: Flow diagram with connection establishment from client to ESP32's server and the user's input sequence.

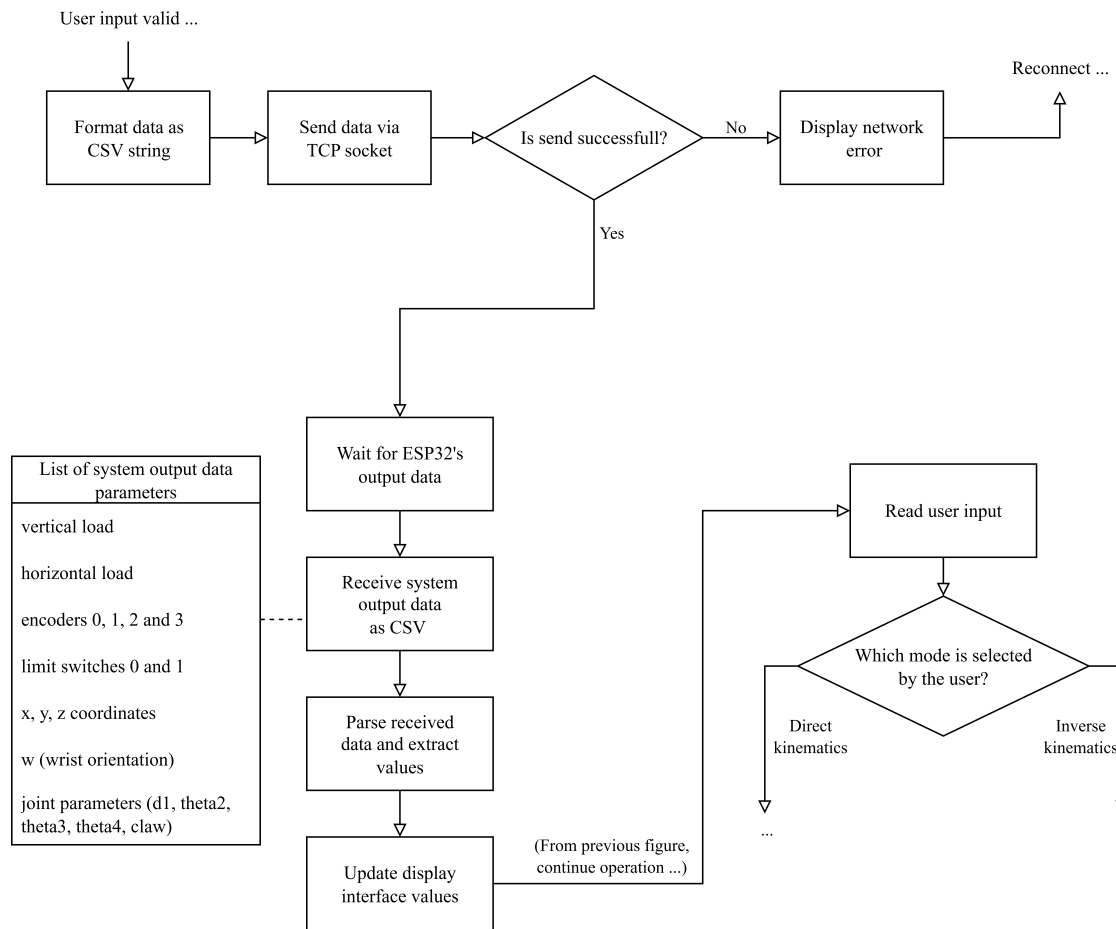


Figure 4.10: Part two: Flow diagram with sending the data input by the user and receiving the system data from the ESP32. If no values are sent, the interface uses the last valid user's input to receive new values from the server.

4.6 Kinematics Processing Subsystem

The kinematics processing subsystem serves as the mathematical core of the SCARA manipulator, implementing forward and inverse kinematic algorithms that enable transformation between joint space and Cartesian coordinate systems for end-effector positioning. It operates through a dedicated FreeRTOS task that processes kinematic computations using the established Denavit-Hartenberg parameters specific to the SCARA configuration.

The subsystem also integrates boundary checking mechanisms that validate joint limits and workspace constraints to prevent invalid commands and ensure safe operation within the manipulator's physical capabilities. Its flow diagram is shown below in [Figure 4.11](#):

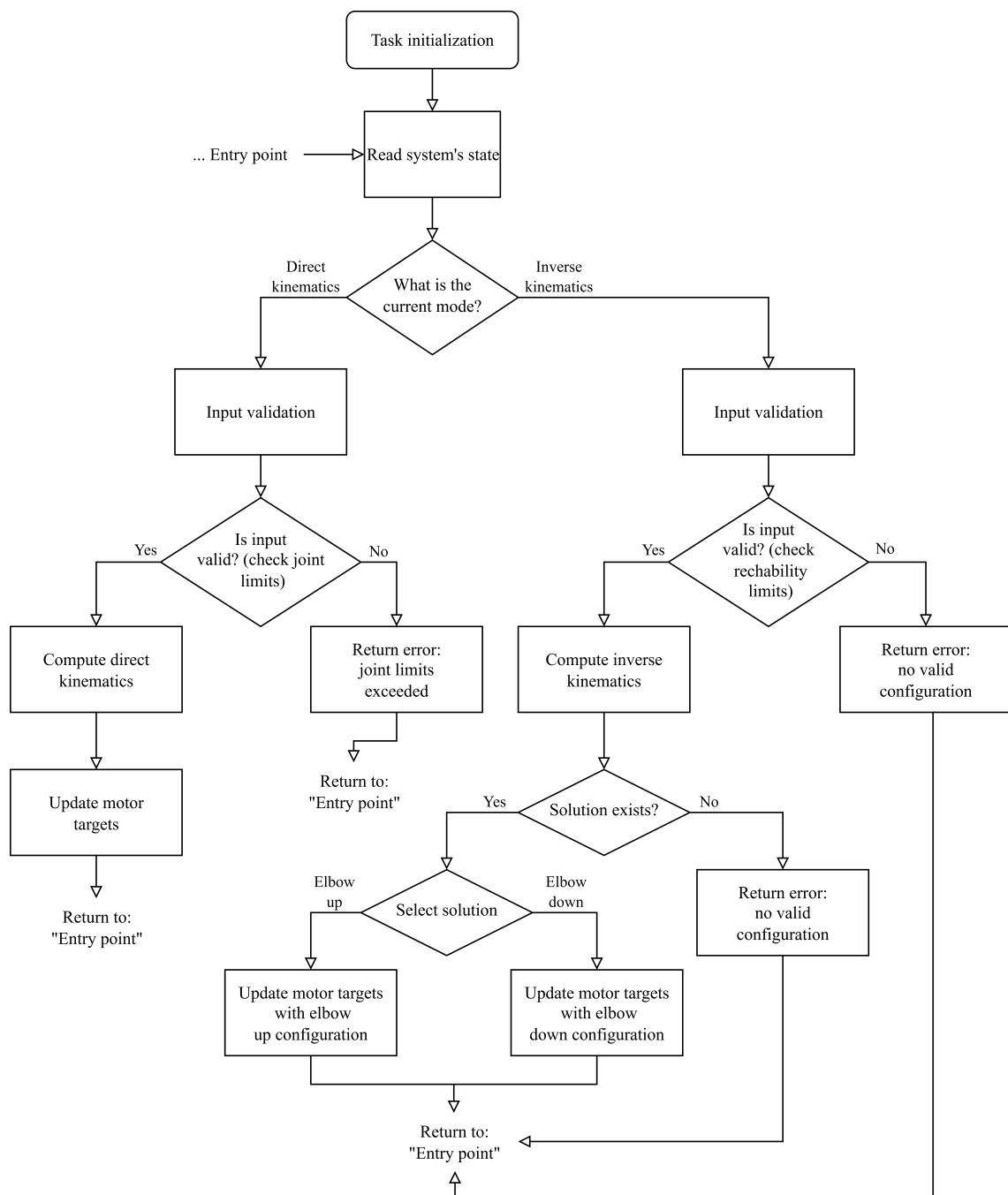


Figure 4.11: Flow diagram with kinematics engine.

4.7 Summary and Conclusions

This chapter has presented the software architecture for the manipulator, demonstrating how a hierarchical, multi-layered design enables effective integration of complex robotic systems within a real-time operating environment. The four-subsystem architecture comprises sensor data acquisition, motion control, network communication, and kinematics processing, providing clear sep-

aration of concerns while maintaining inter-subsystem communication through FreeRTOS task scheduling and mutex-protected shared memory structures.

Each subsystem operates independently yet cohesively, with the sensor acquisition layer continuously monitoring four magnetic encoders, two load cells, and limit switches; the motion control layer managing five stepper motors via MCPWM peripheral control; the network layer facilitating WiFi-based remote operation through dual HTTP/TCP protocols; and the kinematics processor executing real-time coordinate transformations using established SCARA geometric relationships.

Future enhancements to this architecture can be readily implemented through the established framework: additional degrees of freedom require only new motor and encoder instances following the existing patterns, supplementary sensors integrate seamlessly through the I2C multiplexer expansion, and advanced control algorithms can be incorporated within the motion control subsystem without affecting other system components.

Chapter 5

Results

The results achieved in this project highlight the functional improvements by extending a 3 DOF SCARA to a manipulator with an additional revolute joint and a gripper. This chapter presents and evaluates the integration and performance of the upgrades, including the analysis of the integration of the new DOF, gripper, practical tests and a conclusion with the resolution of the UN SDGs and the verification of the system's requirements.

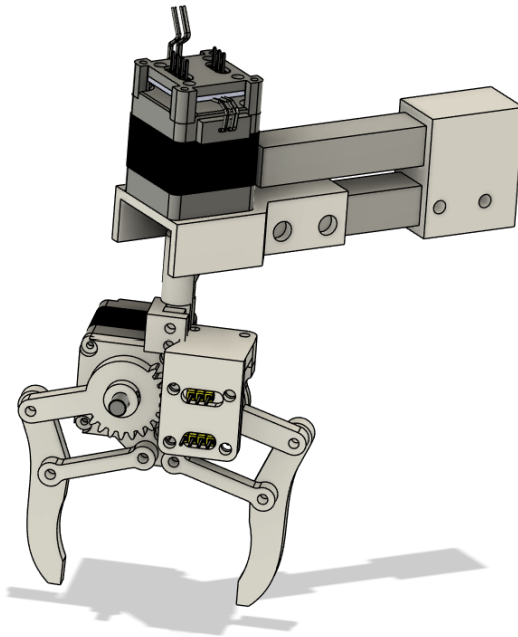
5.1 Structural Improvements

5.1.1 Result Assembly

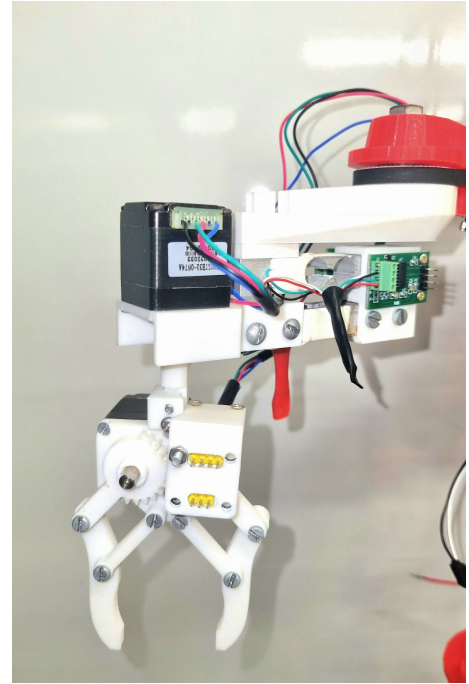
In this section, the steps to introduce a modular gripper are shown. Each piece printed and integrated into the SCARA is shown in the [Figure 5.1](#).

5.1.2 New Degree of Freedom and Gripper

Integrating a fourth DOF allows independent rotation of the end-effector around the workspace. This joint was implemented mechanically by extending the third link from the load cell with a NEMA 17, as shown in [Figure 5.2](#).

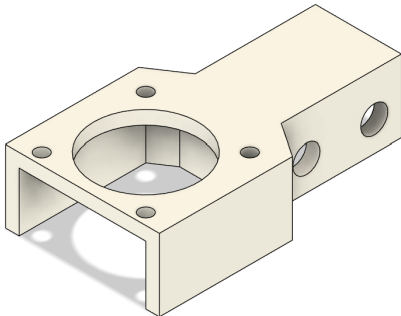


(a) Rendering of the fourth DOF and gripper.



(b) Final implementation of the design.

Figure 5.1: Comparison between the designed claw and implementation.



(a) Designed body to extend the third link.

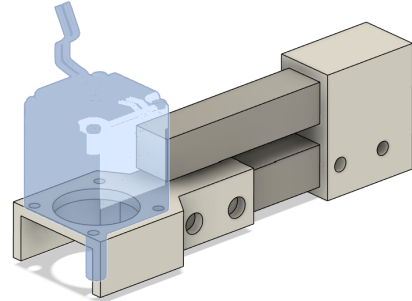
(b) Assembled body from [Figure 5.2a](#) attached in the load cell while holding the NEMA 17 highlighted in blue.

Figure 5.2: Assembly of the new DOF.

A small piece was designed to hold the gripper to the motor's rotor. This piece was created so that in the future, different grippers can be attached to the manipulator, allowing the robot to do other tasks like gripping using vacuum suction, soldering or inspection tasks with a camera.

5.1.3 Gripper

The gripper chosen for the didactic manipulator was an active gripper composed of two symmetrical fingers, actuated with the help of another stepper motor, as shown in [Figure 5.4](#), to enable open and close motions referenced in [Figure 5.5](#). The simplicity allows for observing, for example, the

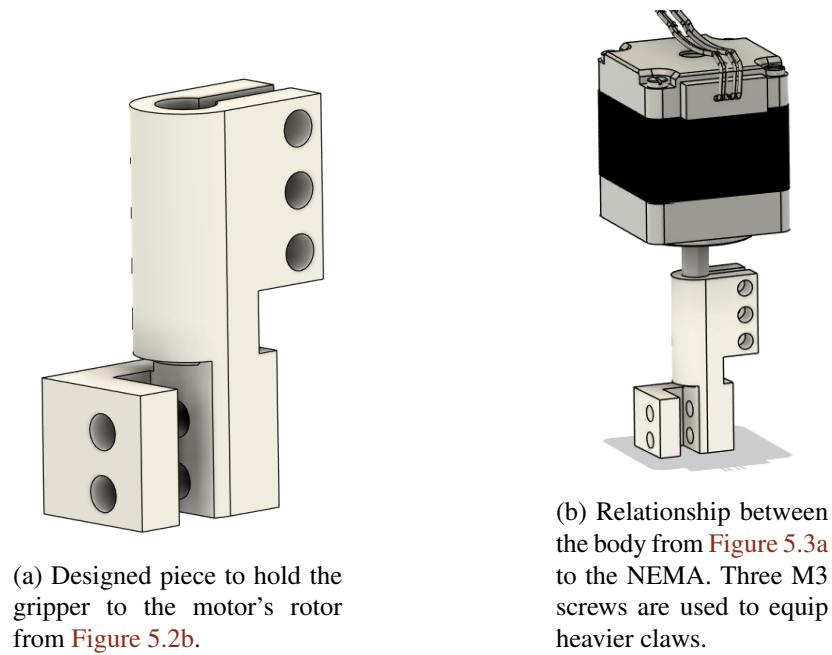


Figure 5.3: Modular piece designed to carry the end-effector.

constant alignment of the gripper in the inverse kinematic mode when going to different points, or to do simple angle adjustments in the workspace. Tasks such as inserting components or aligning objects are now possible with the rotational end-effector.

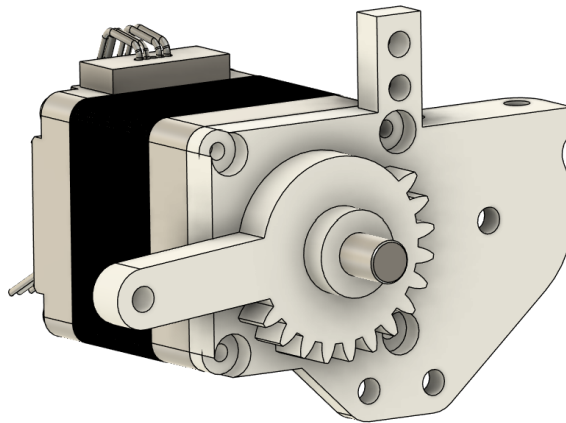


Figure 5.4: A second NEMA 11 is used to perform the open and close motions of the claw.

5.1.4 Encoder's Supports

Two different supports were also designed to hold encoders in place. While planning them, the distance between the encoder and its magnet should be from 0.5 mm to 3.0 mm as referenced in [51]. With this in mind, a 2.0 mm gap was chosen for the encoders. The following Figure 5.6 shows the integration of the encoder responsible for measuring the angle of the manipulator:

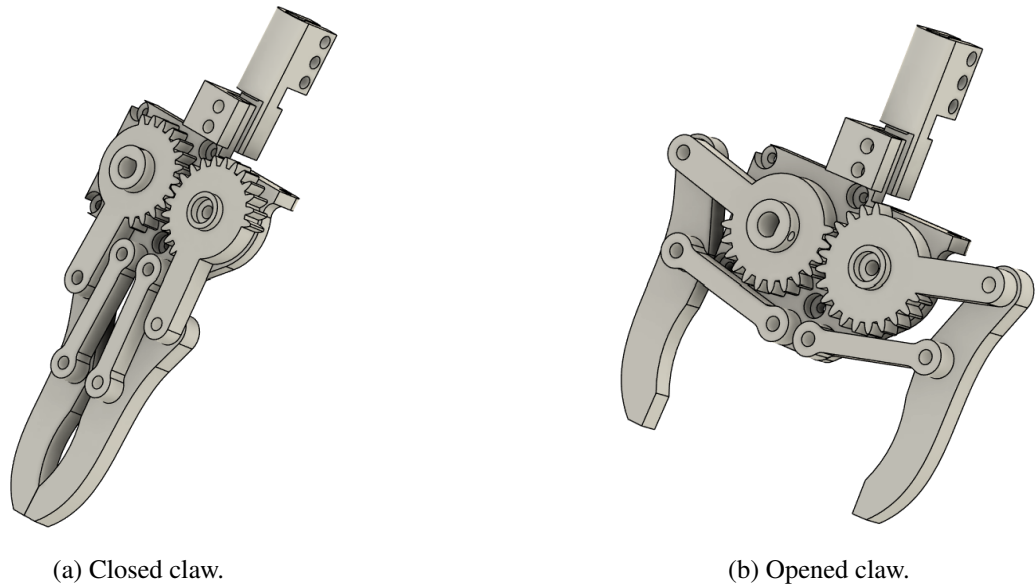


Figure 5.5: Open and closed forms of the claw. Both figures have the piece from [Figure 5.3](#) and are ready to be attached to the motor as shown in [Figure 5.3b](#).

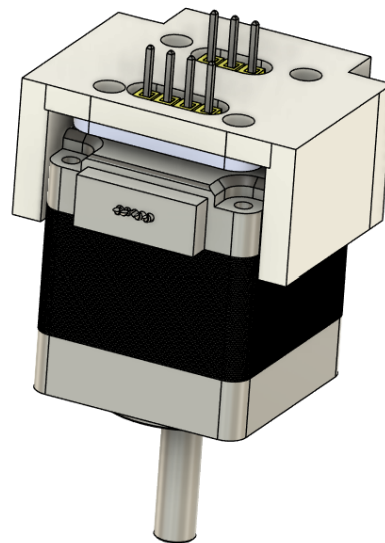


Figure 5.6: Support of the encoder of the last joint.

The second supporter is more advanced. To maintain the centre of mass closer to the projected line from the rotor of the previous motor, a solution at the back of the encoder was not optimal. Therefore, the workaround was to place an encoder in the front of the claw, as shown in [Figure 5.7](#).

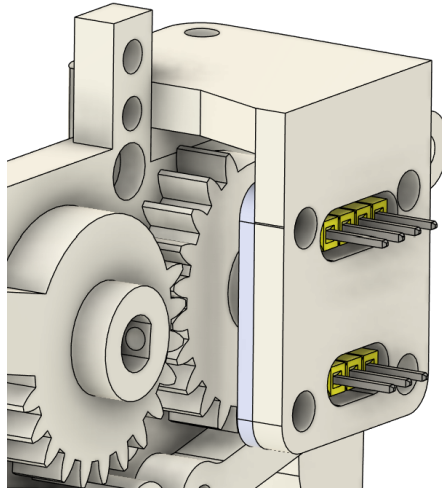
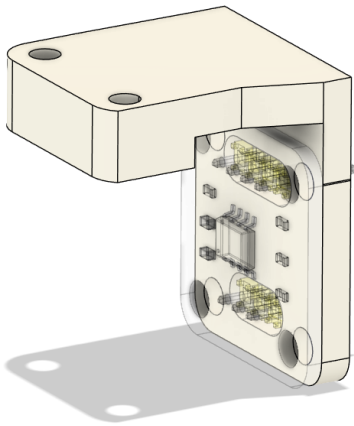
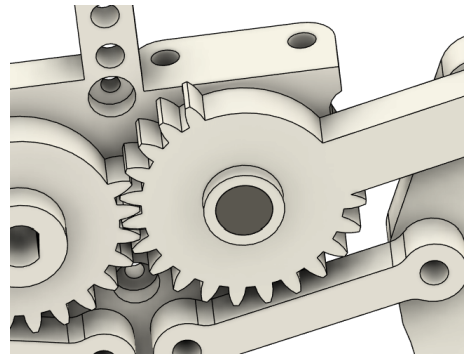


Figure 5.7: Encoder holder demonstration. Since the first actuating arm had the rotor from the NEMA 11, the second one was used to measure how open the gripper was.

To achieve this workaround, the second actuating arm was modified to hold a diametric magnet with a diameter of 6 mm and a height of 2 mm, as shown in Figure 5.8b. To support the encoder, the piece shown in Figure 5.8a was designed and attached to the body of the claw with two screws.



(a) Designed holder to guarantee a distance of 2 mm from the encoder to its magnet.



(b) Attached magnet in the actuating arm.

Figure 5.8: Encoder holder and its magnet to measure the actuation of the arm.

5.2 Practical Tests

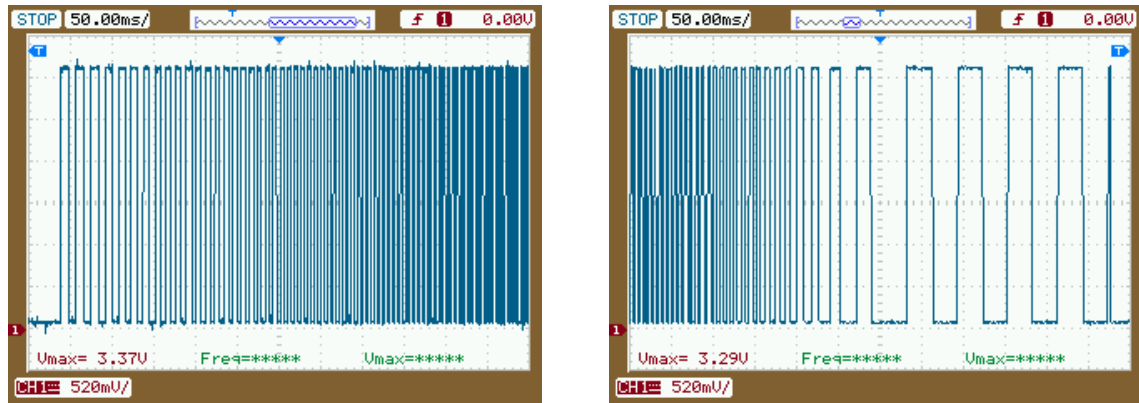
The tests covered in this section evaluate the functional performance of the manipulator under realistic operating conditions. The primary objective was to verify the correct operation of the implemented control strategies, the responsiveness of the mechanical components, and the accuracy of the motion control system.

These practical experiments are divided into two main areas: the generation of PWM control signals for the motors, and the evaluation of the manipulator's motion profile regarding speed and acceleration. The results are presented with supporting figures, oscilloscope captures, and numerical data for a detailed technical analysis.

5.2.1 PWM Generation

Before testing the acceleration and speed of each joint, the STEP signal generated by the MCPWM modules was evaluated. One of its features was the acceleration of the frequency of the PWM signal. It permits the variation of the speed of the motors by a limited amount over time, avoiding internally skipping steps and consequently losing its synchronisation and damaging it in the long term.

An oscilloscope was used with the "single" acquisition mode to capture these signals, allowing for the capture of the waveform. Then, the horizontal scale and position were adjusted to include the acceleration phases, which are the beginning and end of the signal. Finally, the vertical scale and position were adjusted to fit the entire waveform within the screen, increasing its resolution so that the PWM could be seen clearly. The following Figure 5.9 shows the generated signals in the acceleration phase and de-acceleration phase with different values of acceleration.



(a) PWM signal with increasing frequency (acceleration phase).

(b) PWM signal with decreasing frequency (deceleration phase).

Figure 5.9: Oscilloscope analysis of the PWM frequency.

Based on the analysis of Figure 5.9a and Figure 5.9b, the system demonstrates a consistent and predictable response to dynamic changes in PWM frequency during both acceleration and deceleration phases. Regions of higher signal density correspond to periods of increased PWM frequency, while areas with lower density indicate reduced frequencies.

5.2.2 Speed and Acceleration Analysis

This subsection studies the motors' response to other frequencies to further study the effects of the different PWM characteristics. This was achieved by observing the motor's response in two

different ways: The first was limiting the PWM's frequency to lower the maximum rotational speed. The second change was to use a different acceleration value, increasing or decreasing the rate at which the motor changes its rotational speed.

Single Motor

The direct kinematics and the manipulator's first revolute (θ_2^*) were used to study the motors' response to these effects. Encoder data was sampled at fixed intervals of 75 ms to track the motor's position over time. Then, using the interface, θ_2^* is set to -90° after starting in the 0° angle. After ending its movement, its value is changed to 90° .

The following three figures illustrate the motors' positional response under different conditions. The first **Figure 5.10** shows the system under nominal conditions, serving as a reference. In the second **Figure 5.11**, a 75% decrease in acceleration is applied. The third **Figure 5.12** suffered a 75% reduction in the maximum frequency.

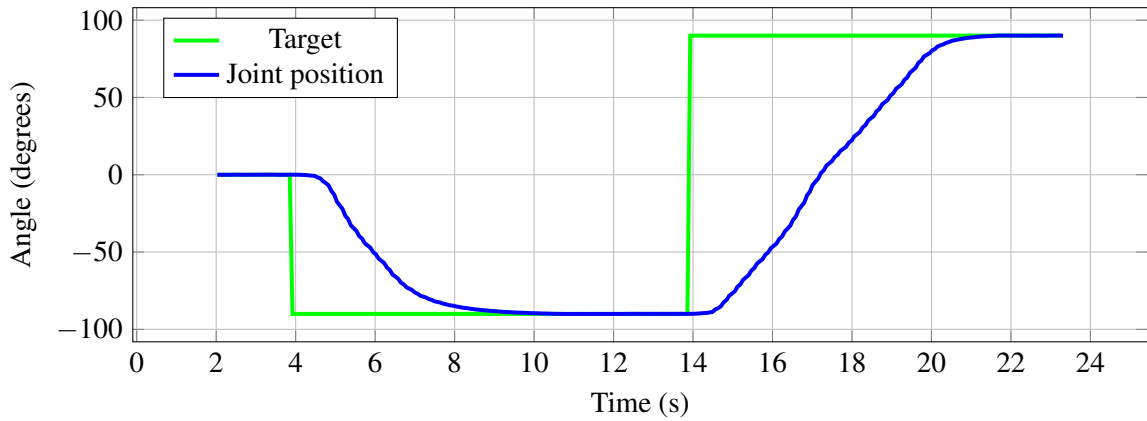


Figure 5.10: Normal operation study using the encoder.

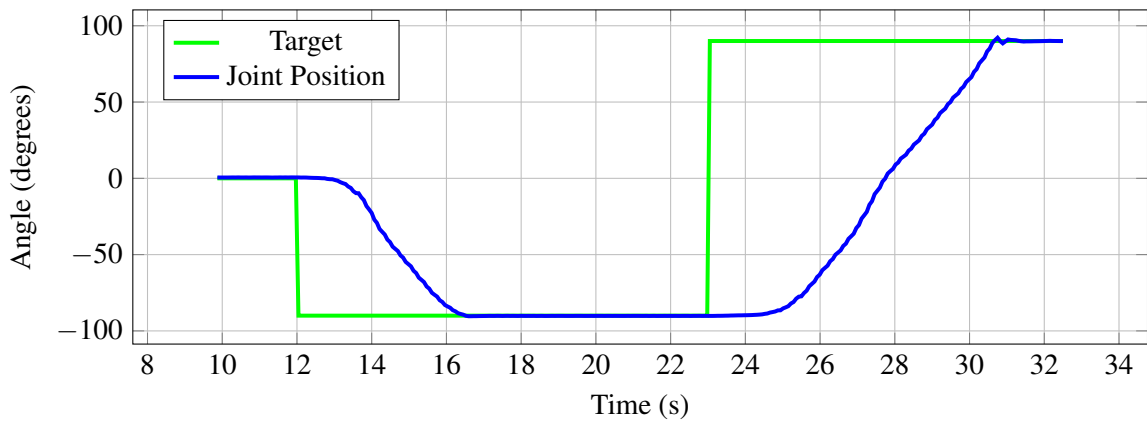


Figure 5.11: Decrease of 75% of acceleration study.

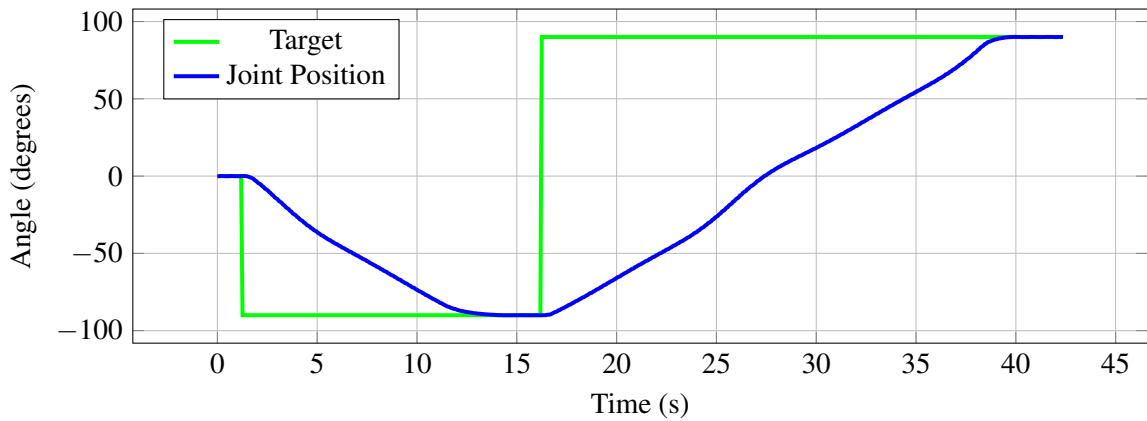


Figure 5.12: Decrease of 75% of speed study.

Overall, the responses of the motor in all cases were similar to each other. Starting with [Figure 5.11](#), the motor took longer to hit its maximum speed, from 2 to 2.5 seconds approximately (25% more time). Another relevant aspect is a slight overshoot after exceeding the target after 30 seconds. This has happened because changing the acceleration also changes the deceleration, causing the motor to skip steps. Besides that, it is also relevant to compare the slopes of both curves when the acceleration is null (slope is more or less constant). Analysing and comparing the slopes when they are constant reveals that both motors have the same maximum speed.

To conclude this analysis, [Figure 5.12](#) changes directly the value from the maximum rotational speed. This phenomenon is seen by analysing the slope when it is constant. Compared with the system without changes [Figure 5.10](#), it took about 22 seconds to move the arm from -90° to 90° instead of the 6.5 seconds (238% increase). Relative to [Figure 5.11](#), since the acceleration was not decreased, it does not present the overshoot after arriving at the target at 90° .

Multiple Motors

To evaluate the system's overall performance when all motors are moving, as in [Subsection 5.2.2](#), every encoder acquired the position of its motor every 75 ms. The following figures show the end-effector going from the initial position to the point with coordinates (0.15 0.15 0.15). After arriving at this point, it is set to reach the point (0.15 -0.15 0.15). Despite the differences between the figures, the end-effector is positioned at the same point. This happens because almost every point in the workspace can be achieved within two solutions using inverse kinematics as discussed in [Subsection 3.1.4](#). However, since the points chosen are not included in the singularities exceptions, [Figure 5.13](#) represents the solution with elbow down and [Figure 5.14](#) represents the solution with elbow up.

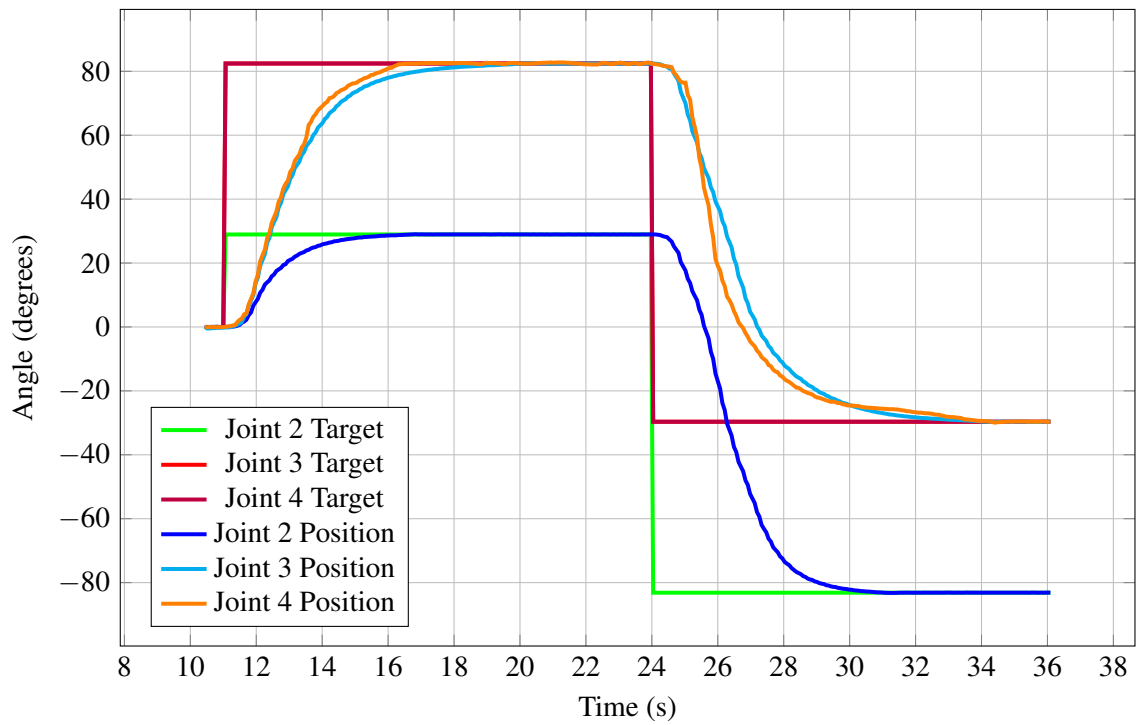


Figure 5.13: From point (0.15, 0.15, 0.15) to (0.15, -0.15, 0.15) with elbow down solution study.

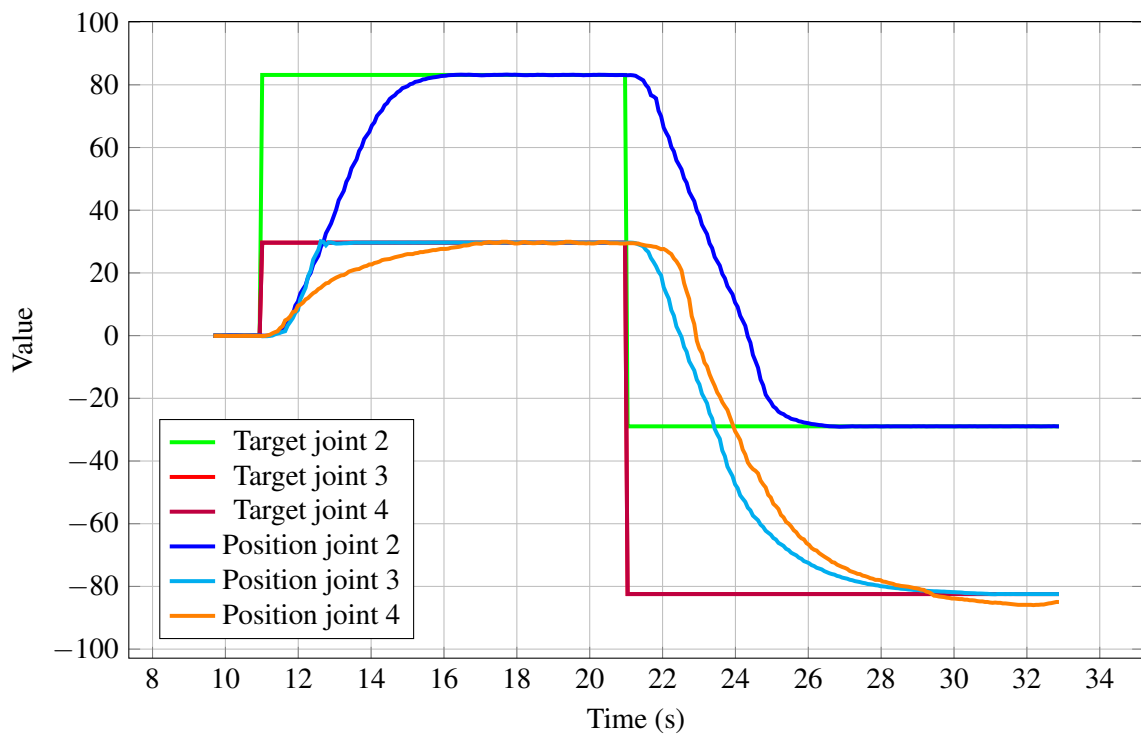


Figure 5.14: From point (0.15, 0.15, 0.15) to (0.15, -0.15, 0.15) with elbow up solution study.

Analysis of both figures shows that all joints can achieve their target position. However, observing [Figure 5.14](#) reveals that the fourth joint overshoots after the 30-second mark, suggesting

that its acceleration must be increased to avoid this phenomenon.

Conclusion About PWM's Control

From this study, it can be concluded that controlling motor position through PWM is a practical and reliable approach for embedded robotic systems. Adjusting its frequency can achieve precise control over speed and direction. The implementation using the ESP32's MCPWM demonstrated consistent performance under varying conditions.

Another essential advantage of this approach is its modularity. Since frequency range, pulse polarity, and dead-time, for example, can be independently defined for each output channel, the system can support different motor types or drive configurations without significant changes in the system's architecture.

5.3 Positional Testing

5.3.1 Elbow Up and Down on Inverse Kinematics

The Python and Desmos 3D platforms were used to test both inverse kinematics solutions. For each one, the kinematics model of the manipulator was implemented and studied. To compare the results, two points were considered: (0.15 0.15 0.15) and (0.15 -0.15 0.15). Then, both solutions of elbow up and elbow down were calculated. Figures 5.19a to 5.17b shows the end-effector with coordinates (0.15 0.15 0.15), and figures 5.18a to 5.20b shows the end-effector with coordinates (0.15 -0.15 0.15).

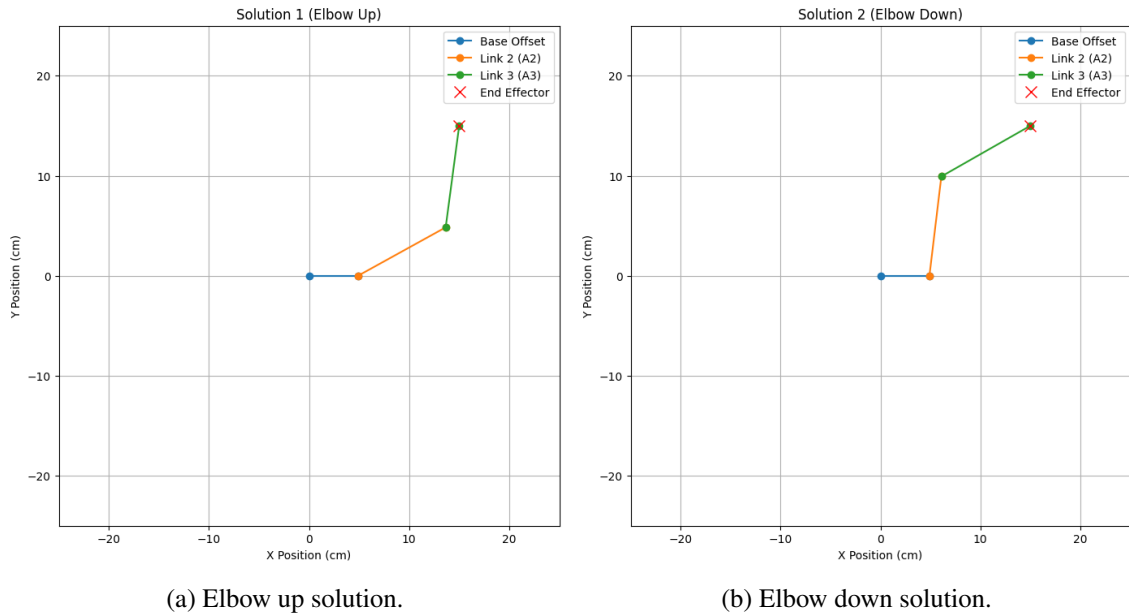


Figure 5.15: Python platform with solutions to the inverse kinematics at the point (0.15 0.15).

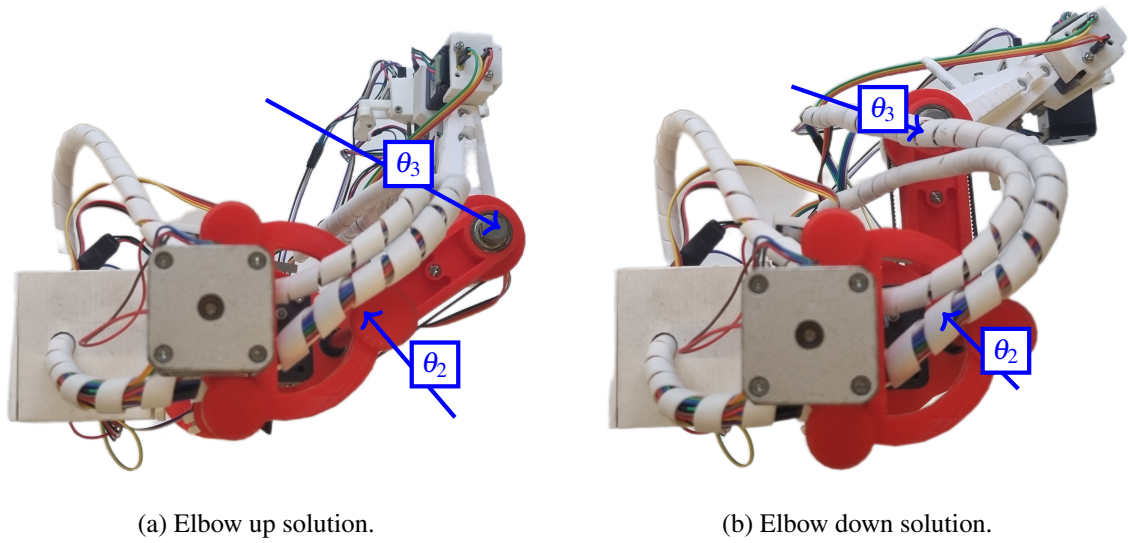


Figure 5.16: SCARA manipulator solutions to the inverse kinematics in the point (0.15 0.15 0.15).

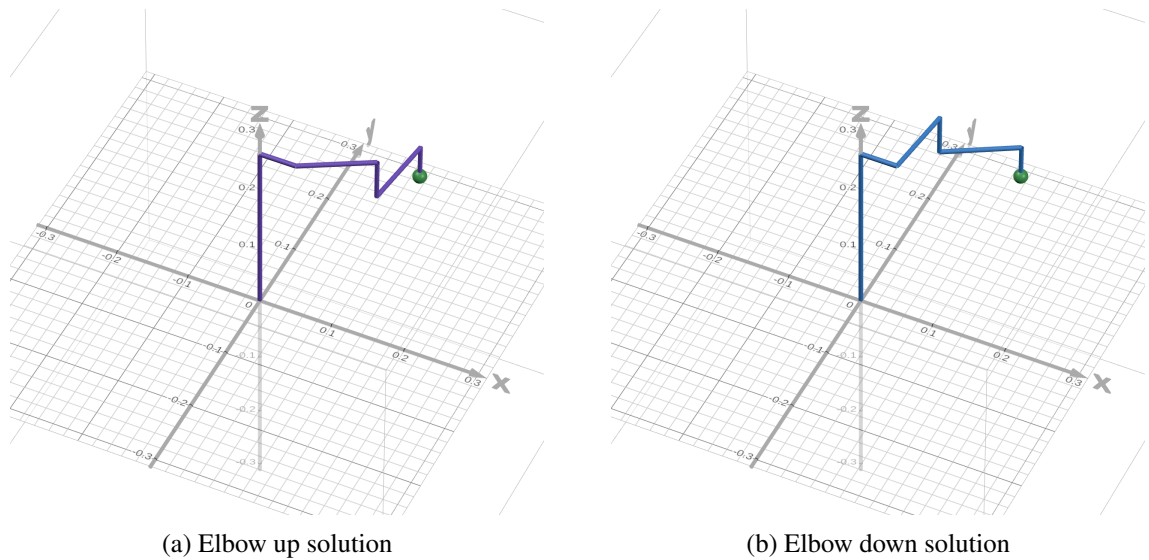


Figure 5.17: Desmos 3D platform with solutions to the inverse kinematics in the point (0.15 0.15 0.15). The purple and blue lines represent the links in Figure 5.17a and Figure 5.17b, respectively. The green point represents the position of the end-effector.

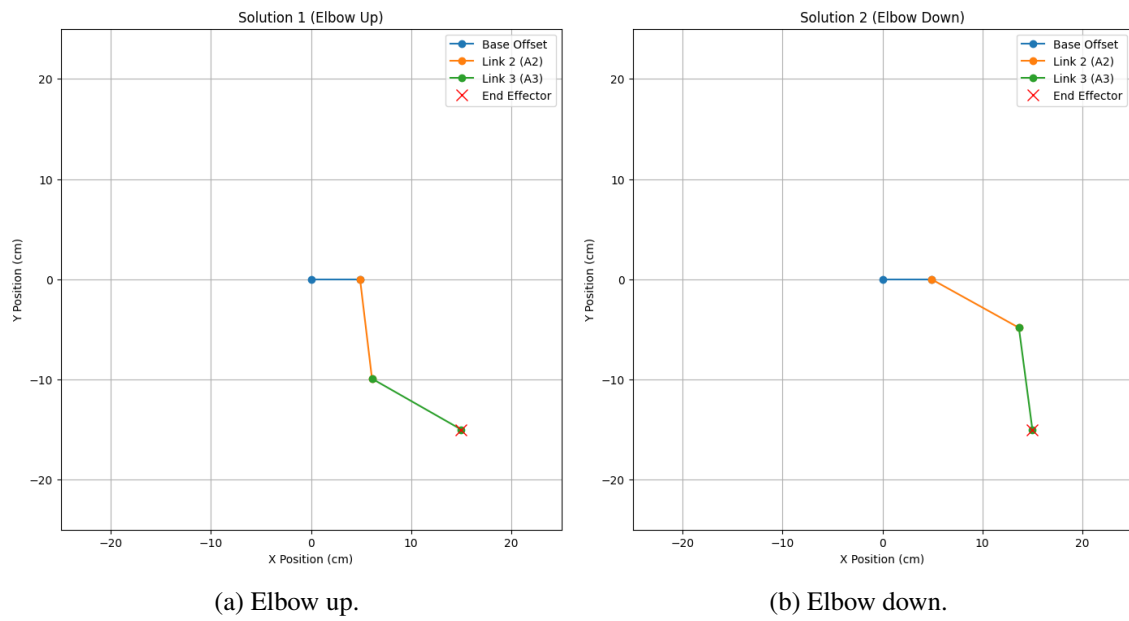


Figure 5.18: Python platform with solutions to the inverse kinematics in the point (0.15 -0.15).

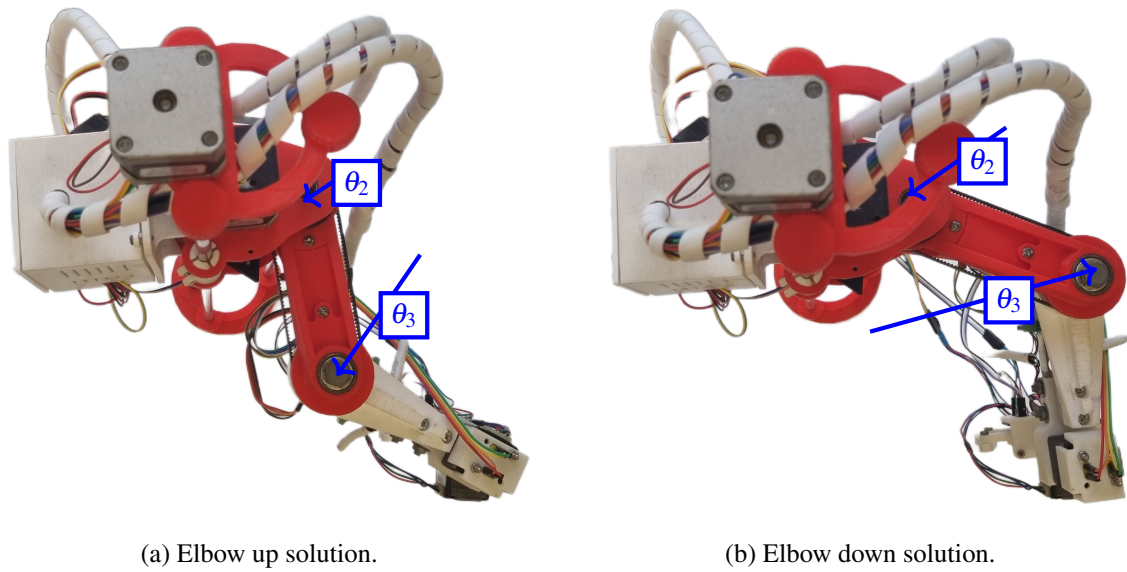


Figure 5.19: SCARA manipulator solutions to the inverse kinematics in the point (0.15 -0.15 0.15).

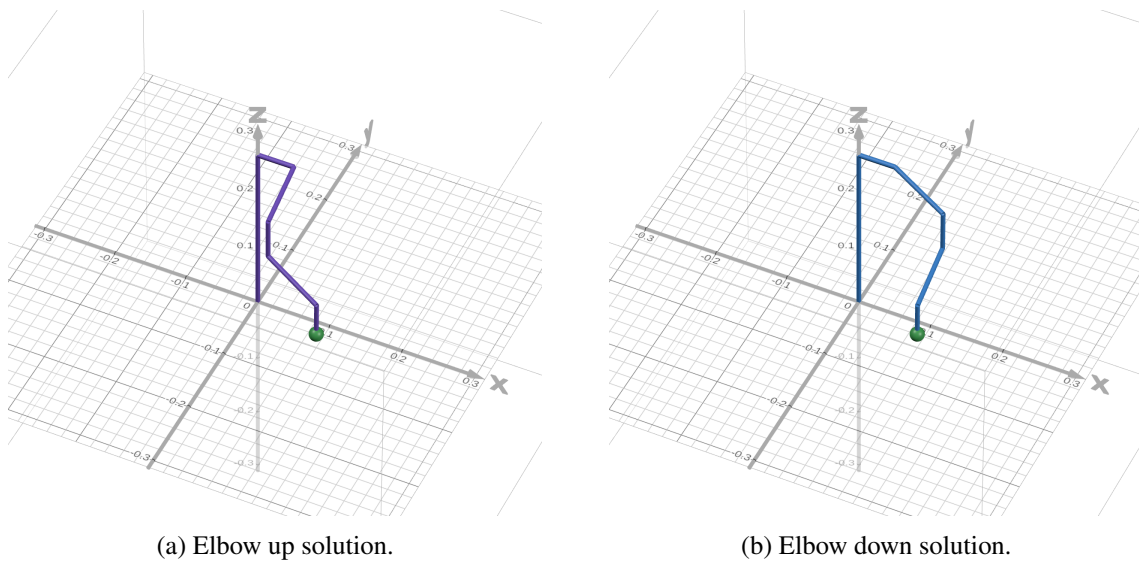


Figure 5.20: Desmos 3D platform with solutions to the inverse kinematics in the point (0.15 -0.15 0.15). As in Figure 5.17, the green point represents the end-effector's position. The purple and blue lines represent the links in Figure 5.20a and Figure 5.20b, respectively.

These tests allowed for visualisation of the forward and inverse kinematics, serving as a complementary learning aid for students before interacting with the physical manipulator.

5.3.2 Repeatability Tests

To evaluate the accuracy and consistency of the manipulator, a series of repeatability tests was performed using the inverse kinematics. A millimetric paper was used, and with the help of the interface, different values were tested each time.

For this test, 6 points inside the range of the millimetric paper were chosen. Then, starting from the default position with the joints θ_2^* and θ_3^* equal to 0° , the manipulator would be sent to the point to measure. After measuring its z coordinate with a ruler and x and y with the millimetric paper, the robot is sent to the default position, and the procedure can be repeated. For each point, a total of 5 measurements were made, resulting in the Table 5.1 below:

Table 5.1: Positional Error Metrics with Inverse Kinematics

Metric	Value (m)
Maximum Deviation	0.0033
Average Error	0.0014
Standard Deviation	0.0009
RMSE	0.0017

Ultimately, these tests show how reliably the end-effector can reach target positions over multiple cycles. Repeatability is a crucial performance metric in these manipulators, particularly when requiring precise assembly, component placement, or object manipulation.

5.4 Interactive Interface

This section presents the interactive interface developed to serve as a Supervisory Control and Data Acquisition (SCADA) system for the manipulator. This web application is hosted directly on the ESP32 to monitor the system state and control the machine in real time without external software or drivers.

The interface, shown in Figure 5.21, was implemented using HTML and is served via ESP32’s built-in HTTP server. The interface consists of modular panels, each with specific functions detailed in the following subsections.

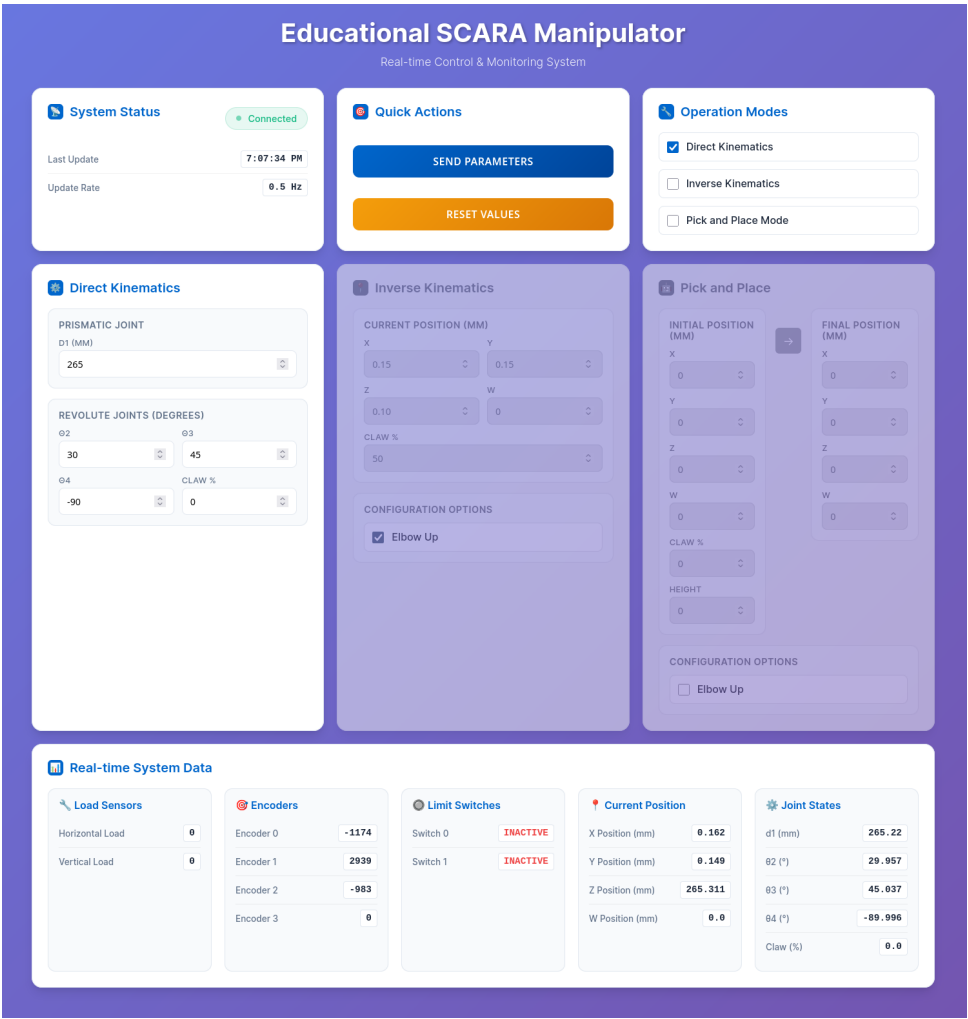


Figure 5.21: Developed interface hosted by the microcontroller.

5.4.1 System Status Panel

The system status panel, shown in Figure 5.22, displays the connection status between the interface and the ESP32 microcontroller. This panel includes the timestamp of the last data update and

the update frequency of the system data, providing users with information about the system's operational state and data freshness.

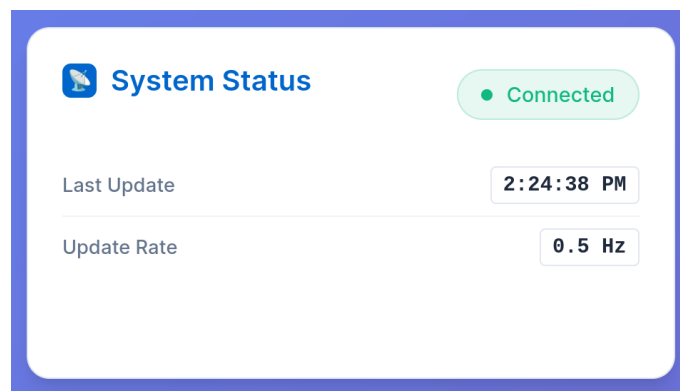


Figure 5.22: System status panel showing connection status and update information.

5.4.2 Quick Actions Panel

The quick actions panel, shown in [Figure 5.23](#), has two buttons for parameter management. The "send parameters" button transmits the currently configured values from any active operation mode (direct kinematics, inverse kinematics, or pick and place) to the microcontroller. The "reset values" button restores all parameter fields to their default values, allowing for a quick system reset without manual input clearing.

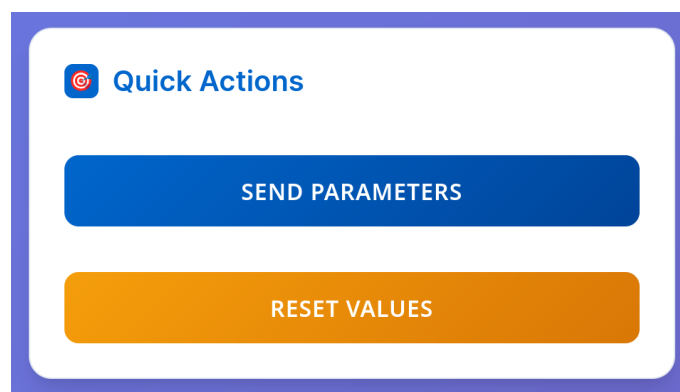


Figure 5.23: Quick actions panel.

5.4.3 Operation Modes Panel

The operation modes panel implements a mutually exclusive selection mechanism for choosing the control mode. When one operation mode is selected (for example, direct kinematics), the other modes become inaccessible, ensuring a clear operational state and preventing conflicting commands. This design, shown in [Figure 5.24a](#) to [Figure 5.24c](#), enforces a single active control paradigm at any given time.

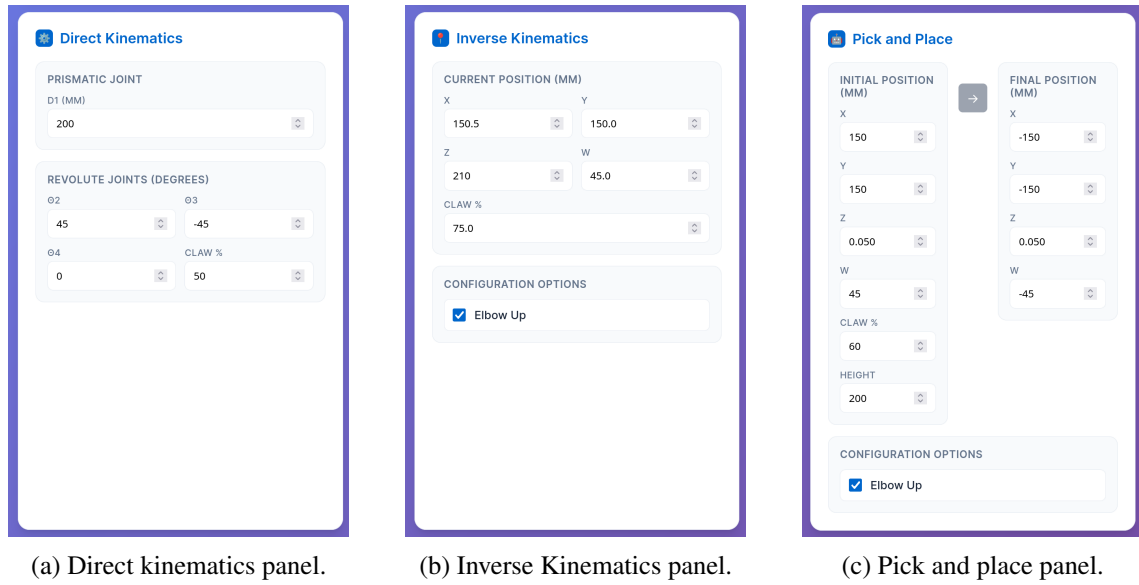


Figure 5.24: Operation modes panels.

Direct Kinematics Panel

The direct kinematics panel, shown in Figure 5.24a, enables joint-space control of the manipulator by allowing users to specify individual joint angles. This control mode provides direct manipulation of each degree of freedom, including the prismatic joint d_1^* and revolute joints (θ_2^* , θ_3^* , θ_4^*), along with gripper control (claw percentage).

Inverse Kinematics Panel

The inverse kinematics panel facilitates Cartesian-space control by allowing users to specify desired end-effector coordinates (x , y , z , w). This panel, shown in Figure 5.24b, includes an additional configuration selector that enables users to choose between elbow-up and elbow-down solutions. The gripper control is also integrated within this mode for complete end-effector management.

Pick and Place Panel

The last operation mode, called pick and place, provides an automated operation mode designed for object manipulation tasks. This panel, shown in Figure 5.24c, allows users to define two positions: an initial position (pick location) and a final position (place location), along with an intermediate height parameter for safe trajectory planning. The interface presents these parameters horizontally with a visual arrow indicating the motion sequence. While the interface is fully implemented, the underlying functionality represents a future enhancement for advanced manipulation tasks. This mode includes the same configuration options as inverse kinematics and demonstrates the system's extensibility for complex automated operations.

5.4.4 Real-Time System Data Panel

The real-time system data panel is a monitoring interface, displaying live sensor data from load cells, encoders, and limit switches. This read-only panel, shown in Figure 5.25, is continuously updated to show the end-effector position calculated through forward kinematics and the actual joint states obtained from encoder feedback. This real-time visualisation enables users to monitor system performance, verify command execution, and observe the manipulator's response to control inputs.

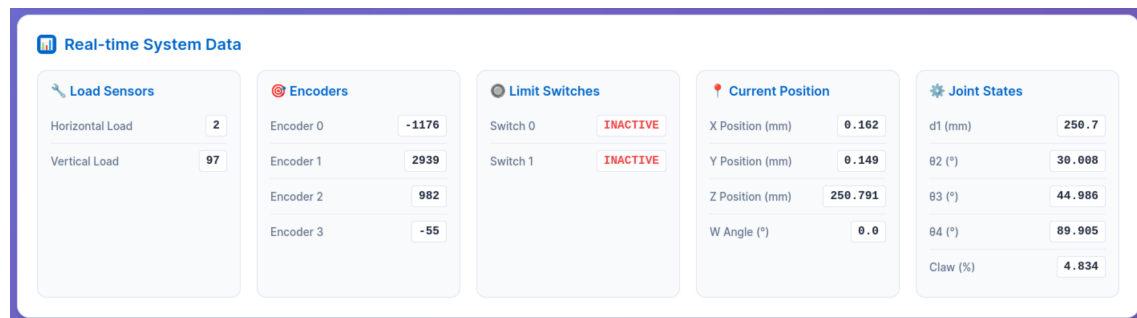


Figure 5.25: Real-time system data panel showing live sensor readings and calculated positions.

5.5 Cost Analysis

One of the key design constraints for the proposed SCARA was affordability. The Table 5.2 shows the components used in the project, including actuators, control electronics, sensors and structural elements.

The cost of 216.92 € shows it is significantly more affordable than most commercial robotic platforms. From an educational perspective, this cost allows assembling multiple units for parallel use within lab environments, fostering learning and experimentation.

5.6 Alignment with UN Sustainable Development Goals

The development of this educational SCARA manipulator directly supports the UN SDGs, with contributions to SDG 4 (quality education) and SDG 9 (industry, innovation and infrastructure). The following section shows how this project aligns with these global objectives.

Contribution to SDG 4: Quality Education

This project is aligned with SDG target 4.5 by providing a low-cost, open-source robotic platform which expands access to this type of technology to institutions with limited resources. Incorporating mechanical design, real-time control and electronics, the developed manipulator is a complete learning tool for students.

Table 5.2: Machine components and their estimated costs.

Component	Quantity	Unit Price (€)	Line Total (€)
Wemos R1 D32	1	0.7	0.7
CNC Shield V3	1	5.95	5.95
Motor Driver Expansion Shield	1	7.5	7.5
NEMA 17 Stepper Motor	3	16	48
NEMA 11 Stepper Motor	2	12	24
A4988 Driver	5	3.9	19.5
AS5600 Encoder	4	4.9	19.6
Microswitch Roller Lever	1	1.6	1.6
Microswitch Extended Lever	1	1.6	1.6
Load Cell + HX711	2	6.95	13.9
M8 Smooth Rod (1m)	3	4.29	12.87
M8 Threaded Rod (1m)	1	2.64	2.64
5mm to 8mm Coupler	1	3.2	3.2
GT2 Pulley 19T	2	2.33	4.66
Belt 320mm (160T)	1	2.29	2.29
GT2 Belt 160mm (80T)	2	1.95	3.9
M3 Screws Box	1	9.95	9.95
608-ZZ Bearing (8×22×7)	2	2.05	4.1
6002-ZZ Bearing (15×32×9)	3	2.5	7.5
LM8UU Bearing	4	1.3	5.2
M8 Screw + Nut	1	0.15	0.15
M8 Nut (extra)	2	0.03	0.06
Dupont Jumpers Set	1	8.05	8.05
Total Cost	-	-	216.92 €

Furthermore, the platform also supports the SDG target 4.4 by promoting skills in embedded systems programming and CAD modelling. Its modularity is ideal for laboratory and practical courses, preparing students for the modern industry.

Contribution to SDG 9: Industry, Innovation, and Infrastructure

This project addresses SDG target 9.5. This manipulator encourages a culture of innovation by encouraging experimentation and research using open-source hardware and software. Its affordability encourages students to explore automation and robotics.

Finally, under SDG target 9.c, using accessible technologies like ESP32 and 3D-printed structures lowers the barriers for implementing real solutions.

5.7 Summary of Results

The experimental results demonstrate the successful enhancement of the original 3 DOF SCARA manipulator by integrating a fourth DOF and a functional gripper. This upgrade expands the

robot's capabilities, enabling it to perform advanced tasks like object reorientation and alignment, which resemble real industrial manipulators. With the tests, the added DOF is stable and smoothly integrated into the FreeRTOS control architecture.

Using hardware PWM through the ESP32's MCPWM peripherals allowed for precise motor speed and acceleration control. Then, the kinematics were tested using Python with Matplotlib Tools and Desmos 3D to confirm the response of the system, and, finally, an interactive interface was shown, proving to be helpful even when testing the repeatability.

The system's modular design, both mechanically and in software, offers flexibility for future upgrades and educational experimentation. [Figure 5.26](#) shows the final assembled manipulator, including all 3D-printed components, integrated electronics and rotational end-effector. Overall, the results validate the platform's effectiveness in bridging theoretical knowledge with practical experiences in robotics.

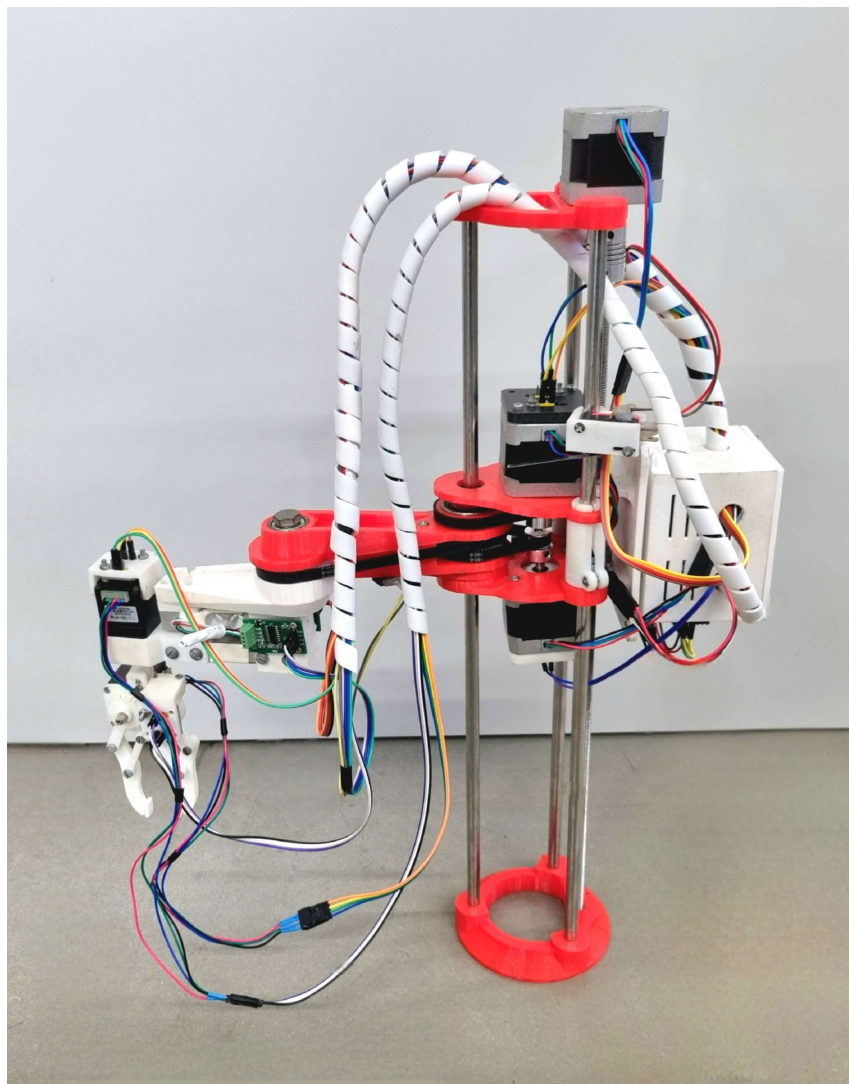


Figure 5.26: Fully assembled SCARA.

The complete source code can be accessed at the project's [GitHub repository](#).

Chapter 6

Conclusion

The dissertation presented the design, implementation, and evaluation of an SCARA manipulator while focusing on implementing a versatile gripper system and adding a degree of freedom for educational purposes. The project successfully combined mechanical, electrical and software aspects into a modular and suitable robot suitable for academic environments.

From a mechanical standpoint, the manipulator was designed for 3D printing, using lightweight and accessible materials and including the fourth degree of freedom. This significantly improved the manipulator's and task versatility, enabling operations that better reflect real-world industrial applications. The gripper adds object interaction while adding end-effector dynamics to the system, expanding the potential for pick-and-place and manipulation tasks.

Kinematic modelling, including forward and inverse calculations, was derived and validated. Jacobian analysis was performed to study singularities and motion behaviour, which offered insight into critical system limitations and safe workspace boundaries. The results from oscilloscope signal capture, encoder feedback, and position repeatability tests confirmed the platform's reproducibility in a lab environment.

The control system was built upon FreeRTOS and the ESP-IDF framework, allowing deterministic multitasking and structured task scheduling. The project explored and implemented advanced control through the MCPWM module of the ESP32, demonstrating how precise motor control and signal generation can be achieved using built-in hardware features. Communication protocols such as I2C and TCP/IP were used to acquire sensor data and provide connectivity, showcasing real-time feedback and external control capabilities.

Ultimately, the work provides a foundation for future development of educational robotic platforms, emphasising modularity, software-hardware co-design, and real-time system control. The SCARA manipulator developed is a robust and scalable platform for further research in robotics, motion planning, and automation, bridging the gap between theoretical instruction and practical experimentation.

Future Work

While the current system provides a functional and versatile 4-DOF SCARA manipulator, there are several directions in which this work can be extended and improved:

- **Additional Degrees of Freedom:** The modular software architecture enables straightforward expansion to higher DOF configurations by adding new motors and encoders instances following the established naming conventions and task patterns. Additional rotational joints could be incorporated at the wrist assembly to enhance workspace flexibility and dexterity, while maintaining the existing FreeRTOS task structure and I2C multiplexer.
- **Gripper Sensor Integration:** Incorporate tactile and proximity sensors within the gripper assembly to enable force-controlled grasping and object detection capabilities. These sensors would provide grip force feedback to facilitate automated object approach and grasp verification, extending the manipulator's capabilities toward more sophisticated manipulation tasks. Additionally, implementing a priority-based task scheduling system would allow security-critical operations to take precedence over routine tasks.
- **System Display Integration:** Integrate a compact display to present real-time system information, including the current IP address and active tasks. Given that the IP address may change depending on the network configuration, this addition would greatly facilitate user access to the server and improve overall system transparency.
- **Status LEDs:** Integrate LEDs to signal different system states such as idle, active, error, or calibration modes. This would provide users with visual feedback, enhancing system monitoring and usability.
- **Physical Control Buttons:** Add dedicated hardware buttons for restarting, stopping, or resetting the system. These controls would allow for safer manual overrides and simpler interaction during demonstrations or troubleshooting.
- **Improved Cable Management:** Design and implement a structured cable management system to organise power and signal wiring throughout the manipulator. This would reduce possible mechanical interference, ease maintenance, and enhance the appearance of the setup.

These improvements would enhance the system's usability, reliability, safety, and scalability. The platform would be better suited for extended experimentation, educational deployment, and potential industrial adaptation by addressing both functional and ergonomic aspects.

References

- [1] Patchara Opaspilai, Supachai Vongbunyong, and Arbtip Dheeravongkit. “Robotic System for Depalletization of Pharmaceutical Products”. In: *2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*. 2021, pp. 133–138. DOI: [10.1109/ICEAST52143.2021.9426302](https://doi.org/10.1109/ICEAST52143.2021.9426302).
- [2] Timotius Heries Noventino, Muhammad Ridho Rosa, and Azam Zamhuri Fuadi. “PID Control Design and Kinematic Modelling of 3-DoF Robot Manipulator”. In: *2022 International Conference on Electrical Engineering, Computer and Information Technology (ICEECIT)*. 2022, pp. 88–94. DOI: [10.1109/ICEECIT55908.2022.10030325](https://doi.org/10.1109/ICEECIT55908.2022.10030325).
- [3] Baohong Liu, Yongyi He, and Zhipeng Kuang. “Design and Analysis of Dual-arm SCARA Robot Based on Stereo Simulation and 3D Modeling”. In: *2018 IEEE International Conference on Information and Automation (ICIA)*. 2018, pp. 1233–1237. DOI: [10.1109/ICInfA.2018.8812353](https://doi.org/10.1109/ICInfA.2018.8812353).
- [4] R. Rascón, M.A. Reyna, and J. Torres. “Tracking and regulation control of a Scara robot with application to minimally invasive surgeries”. In: *2016 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE)*. 2016, pp. 1–6. DOI: [10.1109/GMEPE-PAHCE.2016.7504667](https://doi.org/10.1109/GMEPE-PAHCE.2016.7504667).
- [5] Carlos Rúben Silva da Costa. “Didactic SCARA Manipulator”. openAccess. Dissertação. University of Porto, Faculty of Engineering (FEUP), Oct. 2024. URL: <https://hdl.handle.net/10216/162838>.
- [6] Andreas Birk. “What is Robotics? An Interdisciplinary Field Is Getting Even More Diverse [Education]”. In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 94–95. DOI: [10.1109/MRA.2011.943235](https://doi.org/10.1109/MRA.2011.943235).
- [7] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. 1st. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 2006.
- [8] ABB Inc. *IRB 1400 Industrial Robot: Technical Data and Specifications*. PR10033EN_R3. Auburn Hills, MI, USA, 2005. URL: <https://www.abb.com/robotics>.
- [9] Stanford InfoLab. *Robot Museum: Display 1*. URL: <http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm> (visited on 02/10/2025).
- [10] LTD Yamaha motor CO. *Yamaha scara robot YK-XR series Installation Manual*. 1.10. 127 Toyooka, Chūō-ku, Hamamatsu, Shizuoka 433-8103, Japan: Robotics Business Unit Sales & Marketing Section FA Sales & Marketing Division, 2014, p. 47.
- [11] Nicolas Cortes et al. “Cartesian manipulator for infrastructure inspection and maintenance”. In: *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*. 2021, pp. 1–8. DOI: [10.1109/AIRPHARO52252.2021.9571058](https://doi.org/10.1109/AIRPHARO52252.2021.9571058).

- [12] Skvortsova Valeria and Popov Dmitry. “Design of the Parallel Spherical Manipulator for Wrist Rehabilitation”. In: *2019 3rd School on Dynamics of Complex Networks and their Application in Intellectual Robotics (DCNAIR)*. 2019, pp. 166–168. DOI: [10.1109/DCNAIR.2019.8875585](https://doi.org/10.1109/DCNAIR.2019.8875585).
- [13] Sherif I. Abdelmaksoud et al. “In-Depth Review of Advanced Control Strategies and Cutting-Edge Trends in Robot Manipulators: Analyzing the Latest Developments and Techniques”. In: *IEEE Access* 12 (2024), pp. 47672–47701. DOI: [10.1109/ACCESS.2024.3383782](https://doi.org/10.1109/ACCESS.2024.3383782).
- [14] Otavio Pellicano Moreira de Mello. *Desenvolvimento de um robô manipulador Scara*. Portuguese. Monografia (Bacharelado em Engenharia Mecatrônica). Brasília, Oct. 2016.
- [15] Mohammed H. Al-Mola and Sherif I. Abdelmaksoud. “Performance Improvement of a Space Robotic Manipulator Using Fuzzy-Based PID Controller”. In: *2023 International Telecommunications Conference (ITC-Egypt)*. 2023, pp. 524–529. DOI: [10.1109/ITC-Egypt58155.2023.10206133](https://doi.org/10.1109/ITC-Egypt58155.2023.10206133).
- [16] Mahmud Huseynov. *Integrated Design of Active And Passive Gripper For Robotic Pick And Place Applications*. PREPRINT (Version 1) available at Research Square. Sept. 2021. URL: <https://doi.org/10.21203/rs.3.rs-871636/v1>.
- [17] Mahmud Huseynov. *Integrated Design of Active And Passive Gripper For Robotic Pick And Place Applications*. Sept. 2021. DOI: [10.21203/rs.3.rs-871636/v1](https://doi.org/10.21203/rs.3.rs-871636/v1).
- [18] Robotiq. *Adaptive Grippers - Three-Finger Gripper*. 2024. URL: <https://robotiq.com/products/adaptive-grippers#Three-Finger-Gripper> (visited on 01/02/2024).
- [19] Josie Hughes et al. “Soft Manipulators and Grippers: A Review”. In: *Frontiers in Robotics and AI* 3 (Nov. 2016). DOI: [10.3389/frobt.2016.00069](https://doi.org/10.3389/frobt.2016.00069).
- [20] Josie Hughes et al. “Soft Manipulators and Grippers: A Review”. In: *Frontiers in Robotics and AI* 3 (2016). Ed. by Matteo Cianchetti. Edited by Matteo Cianchetti, Reviewed by Cagdas Denizel Onal, Tommaso Ranzani, Andrew T. Conn., p. 69. DOI: [10.3389/frobt.2016.00069](https://doi.org/10.3389/frobt.2016.00069). URL: <https://doi.org/10.3389/frobt.2016.00069>.
- [21] Mary E. Stokes et al. “The Claw: An Avian-Inspired, Large Scale, Hybrid Rigid-Continuum Gripper”. In: *Robotics* 13.3 (2024). ISSN: 2218-6581. DOI: [10.3390/robotics13030052](https://doi.org/10.3390/robotics13030052). URL: <https://www.mdpi.com/2218-6581/13/3/52>.
- [22] Balaji A, Nippun Kumaar AA, and Sudarshan TSB. “3 axis SCARA robot with universal gripper”. In: *2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*. 2015, pp. 1–5. DOI: [10.1109/ITACT.2015.7492687](https://doi.org/10.1109/ITACT.2015.7492687).
- [23] Marek Babiuch, Petr Foltýnek, and Pavel Smutný. “Using the ESP32 Microcontroller for Data Processing”. In: *2019 20th International Carpathian Control Conference (ICCC)*. 2019, pp. 1–6. DOI: [10.1109/CarpathianCC.2019.8765944](https://doi.org/10.1109/CarpathianCC.2019.8765944).
- [24] Espressif Systems. *ESP32-C3-DevKitC-02 User Guide*. 2023. URL: <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32c3/hw-reference/esp32c3/user-guide-devkitc-02.html> (visited on 12/21/2024).
- [25] Shilong Lv and Zenghui Liu. “Step Motor Control Based on PLC”. In: *2022 5th World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM)*. 2022, pp. 579–582. DOI: [10.1109/WCMEIM56910.2022.10021545](https://doi.org/10.1109/WCMEIM56910.2022.10021545).

- [26] RepRap Wiki contributors. *NEMA 17 Stepper motor*. URL: https://reprap.org/wiki/NEMA_17_Stepper_motor (visited on 04/25/2025).
- [27] Motion Control Products Ltd. *NEMA 11 Hybrid Stepper Motors*. URL: <https://motioncontrolproducts.com/electric-motors/stepper-motors/hybrid-stepper-motors/nema-11.html> (visited on 04/25/2025).
- [28] Abdulbasit Alhinqari and Alejeli Alhengari. "Simulation of Stepper Motor Motion and Control". In: *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*. 2021, pp. 573–578. DOI: [10.1109/MI-STA52233.2021.9464460](https://doi.org/10.1109/MI-STA52233.2021.9464460).
- [29] Muskan Shaikh, Pritesh Shah, and Ravi Sekhar. "Communication Protocols in Industry 4.0". In: *2023 International Conference on Sustainable Emerging Innovations in Engineering and Technology (ICSEIET)*. 2023, pp. 709–714. DOI: [10.1109/ICSEIET58677.2023.10303397](https://doi.org/10.1109/ICSEIET58677.2023.10303397).
- [30] Zijie Meng et al. "Design and Implementation of Ultra-Low Power I2C-Compatible Bus based on Asynchronous Circuit". In: *2023 6th International Conference on Electronics Technology (ICET)*. 2023, pp. 79–84. DOI: [10.1109/ICET58434.2023.10211915](https://doi.org/10.1109/ICET58434.2023.10211915).
- [31] B. Sharan Nripesh Reddy et al. "A Comprehensive Review on Functional Analysis of Real-Time Operating Systems". In: *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. 2023, pp. 1098–1102. DOI: [10.1109/ICIMIA60377.2023.10426594](https://doi.org/10.1109/ICIMIA60377.2023.10426594).
- [32] Wakako Nakano, Yukino Shinohara, and Nagisa Ishiura. "Full Hardware Implementation of FreeRTOS-Based Real-Time Systems". In: *TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON)*. 2021, pp. 435–440. DOI: [10.1109/TENCON54134.2021.9707328](https://doi.org/10.1109/TENCON54134.2021.9707328).
- [33] Yunhao Pan et al. "Forward and Inverse Kinematics Modeling and Simulation of Six-axis Joint Robot Arm Based on Exponential Product Method". In: *2020 IEEE 3rd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*. 2020, pp. 372–375. DOI: [10.1109/AUTEEE50969.2020.9315719](https://doi.org/10.1109/AUTEEE50969.2020.9315719).
- [34] Kiam Heong Ang, G. Chong, and Yun Li. "PID control system analysis, design, and technology". In: *IEEE Transactions on Control Systems Technology* 13.4 (2005), pp. 559–576. DOI: [10.1109/TCST.2005.847331](https://doi.org/10.1109/TCST.2005.847331).
- [35] Kevin M. Lynch and Frank C. Park. *3.3.1 Homogeneous Transformation Matrices*. <https://modernrobotics.northwestern.edu/nu-gm-book-resource/3-3-1-homogeneous-transformation-matrices/>. 2017. (Visited on 07/14/2025).
- [36] SOYO / Pololu. *SY28STH32-0674A Stepper Motor Datasheet*. Tech. rep. Pololu item #1205. Pololu. URL: <https://www.pololu.com/file/0J686/SY28STH32-0674A.pdf> (visited on 07/14/2025).
- [37] MyHobby-CNC. *1.8° NEMA17, Size 42mm High Torque Hybrid Stepping Motor: Technique Parameter Datasheet*. Tech. rep. NEMA17 Stepping Motor Series. MyHobby-CNC. URL: <https://shop.myhobby-cnc.de/media/pdf/af/ee/4f/nema17.pdf> (visited on 07/14/2025).
- [38] Circuit Digest. *Interface A4988 Stepper Motor Driver with Arduino*. 2019. URL: <https://circuitdigest.com/microcontroller-projects/interface-a4988-stepper-motor-driver-with-arduino> (visited on 07/14/2025).

- [39] Allegro MicroSystems. *A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection*. Tech. rep. Allegro MicroSystems. URL: https://www.pololu.com/file/0j450/a4988_dmos_microstepping_driver_with_translator.pdf (visited on 07/14/2025).
- [40] ESPHome. *AS5600 12-Bit Magnetic Position Sensor*. 2025. URL: <https://esphome.io/components/sensor/as5600.html> (visited on 07/15/2025).
- [41] 38-3D. *Using the TCA9548A with the Raspberry Pi*. 2025. URL: <https://38-3d.co.uk/blogs/blog/using-the-tca9548a-with-the-raspberry-pi> (visited on 07/15/2025).
- [42] Texas Instruments. *TCA9548A 8-Channel I2C Switch with Reset and Voltage Translation*. 2025. URL: <https://www.ti.com/product/TCA9548A> (visited on 07/15/2025).
- [43] AVIA Semiconductor Co., Ltd. *HX711 24-Bit Analog-to-Digital Converter for Weigh Scales*. Tech. rep. AVIA Semiconductor Co., Ltd. URL: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf (visited on 07/17/2025).
- [44] SparkFun Electronics. *Getting Started with Load Cells*. 2015. URL: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells/all> (visited on 07/17/2025).
- [45] Electrokit Sweden AB. *Load Cell Amplifier with 24-bit ADC Module — HX711*. 2024. URL: <https://www.electrokit.com/en/hx711-lastcellsforstarkare-monterad-pa-kort> (visited on 07/17/2025).
- [46] University of Auckland Design Tech. *D1 R32 – ESP32*. 2021. URL: <https://designtech.blogs.auckland.ac.nz/d1-r32-esp32/> (visited on 07/15/2025).
- [47] Arduino S.r.l. *Arduino Uno Rev3*. 2025. URL: <https://store.arduino.cc/products/arduino-uno-rev3> (visited on 07/15/2025).
- [48] Cirkuit Designer. *How to Use CNC Shield V3: Examples, Pinouts, and Specs*. 2025. URL: <https://docs.cirkitdesigner.com/component/2c4686b3-94a4-4df1-ae0-7a9f67fd5f2e/cnc-shield-v3> (visited on 07/15/2025).
- [49] LED-Diode. *Stepper Motor Driver Expansion Board DRV8825 / A4988*. URL: <https://www.xn--leddiode-cn3d.com/development-board/stepper-motor-driver-expansion-board-drv8825.html> (visited on 07/17/2025).
- [50] Espressif Systems. *FreeRTOS (IDF) API Reference — ESP-IDF Programming Guide*. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_idf.html. (Visited on 07/18/2025).
- [51] ams OSRAM Group. *AS5600: 12-Bit Programmable Contactless Potentiometer*. Version v1-06. Datasheet. June 2018. URL: <https://look.ams-osram.com/m/7059eac7531a86fd/original/AS5600-DS000365.pdf>.

Appendix A

Schematic Diagram

The **Figure A.1** and **Figure A.2** show the circuit schematic of the SCARA manipulator with its components and connections.

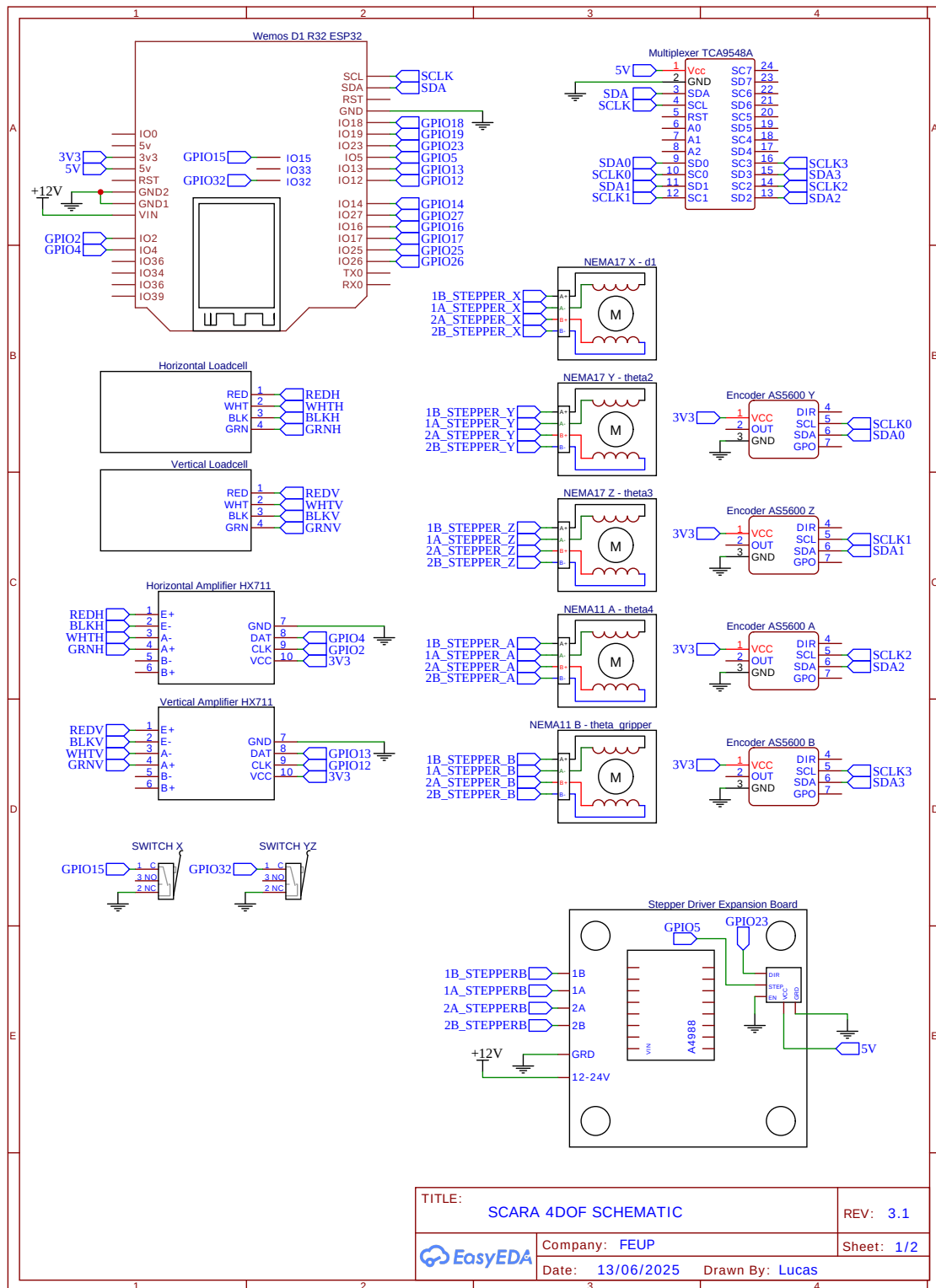


Figure A.1: Schematic diagram of the experimental setup. Hardware connections.

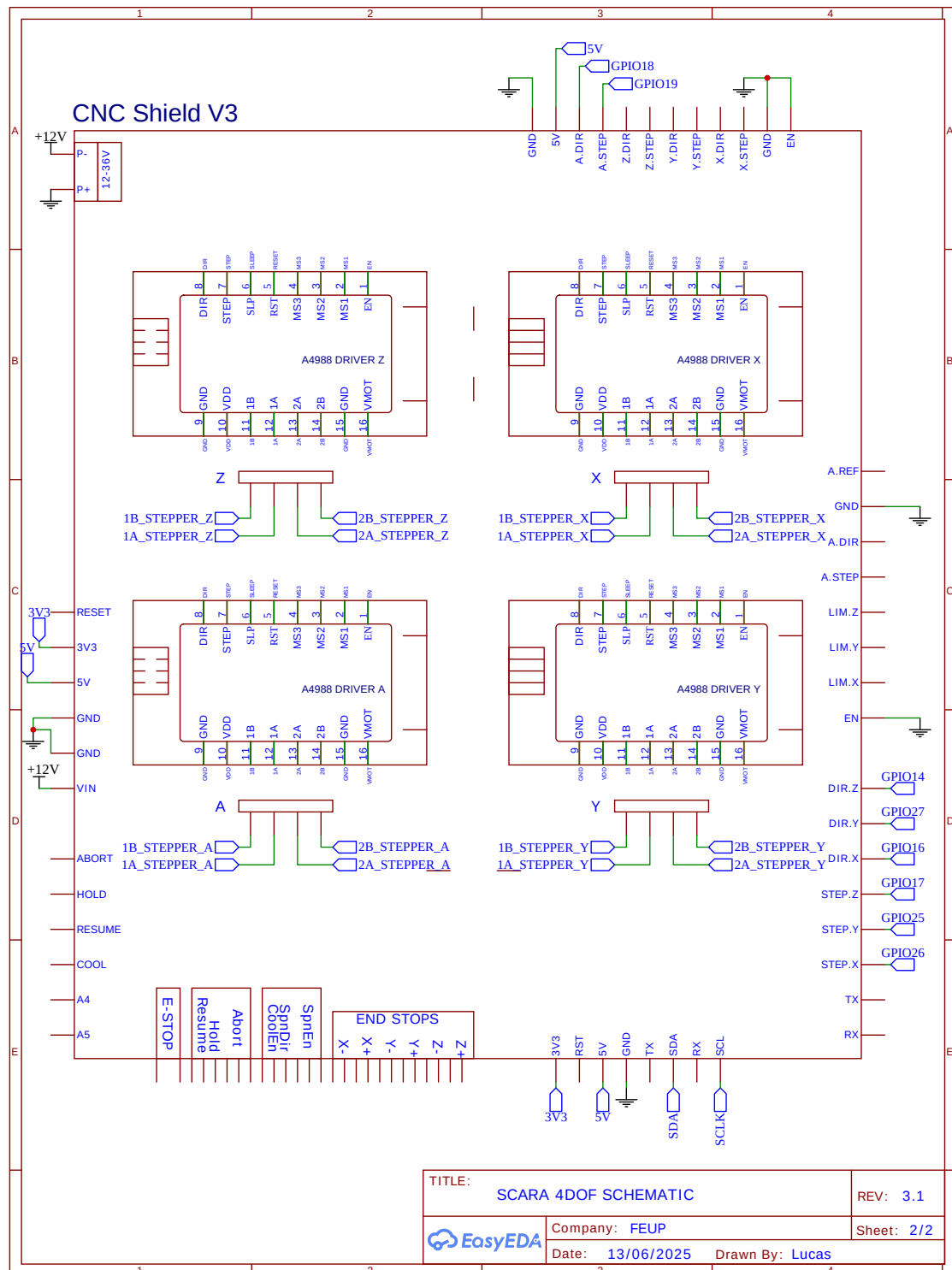


Figure A.2: Schematic diagram of the experimental setup. Shield connections.

Appendix B

Hardware Diagram

The **Figure B.1** shows the components and their communication protocols of the SCARA manipulator.

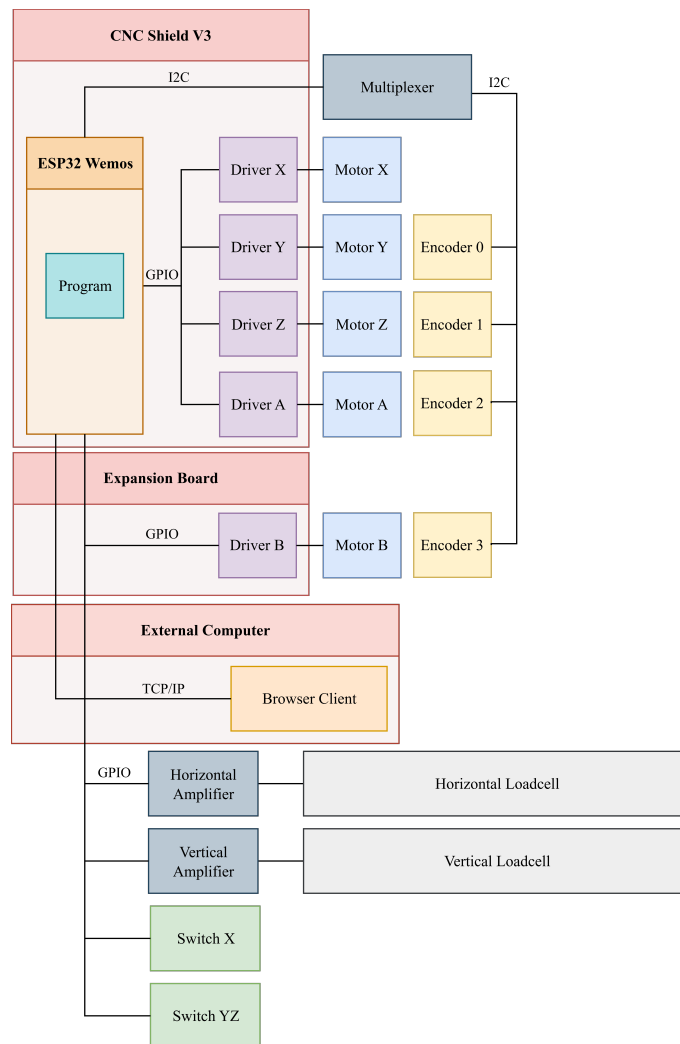


Figure B.1: Diagram with all components and communication protocols used in the project.

Appendix C

Encoder Task

The [Listing C.1](#) defines the function implemented in the `encoder_task()`:

```
1 void encoder_task(void *arg) {  
    encoder_t *encoder = (encoder_t *)arg;  
3    uint16_t max_encoder_val = encoder->encoder_resolution;  
    int16_t half_encoder_val = max_encoder_val / 2;  
5  
    uint16_t last_angle = encoder_read_angle(encoder);  
7    uint16_t current_angle;  
    int16_t delta;  
9  
    while (1) {  
11        current_angle = encoder_read_angle(encoder);  
13  
        // Calculate delta with wraparound handling  
        delta = (int16_t)current_angle - (int16_t)last_angle;  
15  
        // Handle wraparound (shortest path)  
17        if (delta > half_encoder_val) {  
            delta -= max_encoder_val;  
19        } else if (delta < -half_encoder_val) {  
            delta += max_encoder_val;  
21        }  
23  
        // Update accumulated steps  
        encoder->accumulated_steps += delta;  
25        encoder_update_absolute_angle(encoder);  
27  
        last_angle = current_angle;  
        vTaskDelay(pdMS_TO_TICKS(50));  
29    }  
}
```

Listing C.1: Encoder task with wraparound handling.

Appendix D

Load Cell Task

The Listing D.1 defines the function implemented in the `hx711_task()`:

```
void hx711_avg_task(void *arg) {
2   hx711_t *hx711 = (hx711_t *)arg;
   int64_t sum = 0;
4   int num_samples = 10;

6   while (1) {
       for (int s = 0; s < num_samples; s++) {
8           // Wait for HX711 ready signal
           while (gpio_get_level(hx711->data_pin) == 1) {
10              vTaskDelay(pdMS_TO_TICKS(1));
           }
12          uint32_t data = 0;
           for (int i = 0; i < 24; i++) {
14              gpio_set_level(hx711->clock_pin, 1);
              vTaskDelay(pdMS_TO_TICKS(1));
16              data = (data << 1) | gpio_get_level(hx711->data_pin);
              gpio_set_level(hx711->clock_pin, 0);
18              vTaskDelay(pdMS_TO_TICKS(1));
           }
20          // Sign-extend 24-bit to 32-bit
           if (data & 0x800000) {
22              data |= 0xFF000000;
           }
24          sum += (int32_t)data;
           vTaskDelay(pdMS_TO_TICKS(25));
26      }
       int32_t average = sum / num_samples;
28       ESP_LOGI("average", "Average HX711 reading: %ld", average);
   }
30 }
```

Listing D.1: Load cell averaging task.

Appendix E

Switch Task

The Listing E.1 defines the function implemented in the `switch_task()`:

```
void switch_task(void *arg) {  
2   switch_t *switch_n = (switch_t *)arg;  
   if (switch_n == NULL) {  
4       ESP_LOGE(TAG, "Parameter is null, aborting.");  
       vTaskDelete(NULL);  
6       return;  
   }  
8  
   while (1) {  
10      update_switch_value(switch_n);  
       vTaskDelay(pdMS_TO_TICKS(10));  
12  }  
}  
14  
void update_switch_val(switch_t *switch_n) {  
16   if (switch_n == NULL) {  
       ESP_LOGE(TAG, "Parameter is null, aborting.");  
18   return;  
   }  
20  
   // 1 means pressed and 0 means unpressed  
22   switch_n->is_pressed = !gpio_get_level(switch_n->gpio_pin);  
   return;  
24 }
```

Listing E.1: Switch monitoring task.

Appendix F

Motor Task

The Listing F defines the function implemented in the `motor_control_task()`:

```
void motor_control_task(void *arg) {
2   motor_t *motor = (motor_t *)arg;
   float error, output;
4   const float MOTOR_CONTROL_TASK_PERIOD_MS = 20;
   const float DEAD_BAND_THRESHOLD = 2.0f;
6   while (1) {
       error = motor->control_vars->encoder_target_pos -
8           motor->control_vars->ref_encoder->accumulated_steps;

10      float dt = MOTOR_CONTROL_TASK_PERIOD_MS / 1000.0f;

12      // Deadband: prevent oscillation around setpoint
      if (fabsf(error) < DEAD_BAND_THRESHOLD) {
14          motor->control_vars->pid->integral = 0.0f;
          motor_set_target_frequency(motor, 0.0f);
16          vTaskDelay(pdMS_TO_TICKS(MOTOR_CONTROL_TASK_PERIOD_MS));
          continue;
18      }
      // PID calculations
20      motor->control_vars->pid->integral += error * dt;

22      // Integral windup protection
      const float INTEGRAL_LIMIT = 8000.0f;
24      if (motor->control_vars->pid->integral > INTEGRAL_LIMIT)
          motor->control_vars->pid->integral = INTEGRAL_LIMIT;
26      else if (motor->control_vars->pid->integral < -INTEGRAL_LIMIT)
          motor->control_vars->pid->integral = -INTEGRAL_LIMIT;
28
      float derivative = (error - motor->control_vars->pid->prev_error) / dt;
30      motor->control_vars->pid->prev_error = error;
   }
}

[Continued on next page]
```

```
1  [Continuation from last page]
3      output = motor->control_vars->pid->Kp * error +
4              motor->control_vars->pid->Ki *
5              motor->control_vars->pid->integral +
6              motor->control_vars->pid->Kd * derivative;
7
8      // Output saturation
9      if (output > motor->pwm_vars->max_freq)
10         output = motor->pwm_vars->max_freq;
11     if (output < -motor->pwm_vars->max_freq)
12         output = -motor->pwm_vars->max_freq;
13
14     // Direction control and frequency application
15     bool reverse = output < 0;
16     gpio_set_level(motor->gpio_dir,
17                   motor->pwm_vars->dir_is_reversed ? !reverse : reverse);
18     motor_set_target_frequency(motor, fabsf(output));
19     vTaskDelay(pdMS_TO_TICKS(MOTOR_CONTROL_TASK_PERIOD_MS));
20 }
21 }
```

Listing F.1: PID Motor Control Implementation

Appendix G

Motor Callback Function

The Listing F defines the function implemented in the `motor_control_timer_cb()`:

```
1 void motor_update_timer_cb(void *arg) {  
    motor_t *motor = (motor_t *)arg;  
  
3  
    const float dt = UPDATE_INTERVAL_MS / 1000.0f;  
5    float current_freq = motor->pwm_vars->current_freq_hz;  
    float target_freq = motor->pwm_vars->target_freq_hz;  
7    float *velocity = &motor->pwm_vars->velocity_hz_per_s;  
    float max_accel = motor->pwm_vars->max_accel;  
9  
    // update step count and check target reached  
11   if (motor->pwm_vars->step_counting_enabled && current_freq > 0) {  
        // calculate and add steps taken this interval  
13        float steps_this_interval = current_freq * dt;  
        motor->pwm_vars->step_count += (uint32_t)steps_this_interval;  
15  
        // check if target steps reached  
17        if (motor->pwm_vars->step_count >= motor->pwm_vars->target_steps) {  
            motor_stop_and_cleanup(motor);  
19            return;  
        }  
21  
        // deceleration when approaching target  
23        uint32_t remaining_steps =  
        motor->pwm_vars->target_steps - motor->pwm_vars->step_count;  
25        uint32_t decel_steps =  
        (uint32_t)(current_freq * current_freq / (2 * max_accel));  
27  
    [Continued on next page]
```

```

[Continuation from last page]
2
    if (remaining_steps <= decel_steps) {
4        float ideal_freq = sqrt(2.0f * max_accel * remaining_steps);
        if (ideal_freq < target_freq) {
6            target_freq = ideal_freq;
            }
8    }
    }
10
    // calculate frequency error and check convergence
12    float freq_error = target_freq - current_freq;

14    if (fabs(freq_error) < 0.5f && fabs(*velocity) < 1.0f) {
        // Target reached - set exact frequency and stop timer
16        motor->pwm_vars->current_freq_hz = target_freq;
        motor_update_pwm_frequency(motor, target_freq);
18        esp_timer_stop(motor->timer_handle);
        return;
20    }

22    // apply acceleration control
    float desired_accel = (freq_error > 0) ? max_accel : -max_accel;
24    *velocity += desired_accel * dt;

26    // Limit maximum velocity to prevent overshoot
    float max_velocity = max_accel * 2.0f;
28    *velocity = fmax(-max_velocity, fmin(max_velocity, *velocity));

30    // calculate new frequency with overshoot prevention
    float new_freq = current_freq + (*velocity * dt);
32

    // Prevent overshooting target
34    if ((freq_error > 0 && new_freq > target_freq) ||
        (freq_error < 0 && new_freq < target_freq)) {
36        new_freq = target_freq;
        *velocity = 0;
38    }

40    // Ensure non-negative frequency
    new_freq = fmax(0.0f, new_freq);
42    motor->pwm_vars->current_freq_hz = new_freq;

44 [Continued on next page]

```

```
[Continuation from last page]
2
3     // update PWM output
4     if (new_freq > 0.0f) {
5         // Create PWM if transitioning from zero
6         if (motor->mcpwm_timer == NULL) {
7             motor_create_pwm(motor, new_freq);
8         } else {
9             motor_update_pwm_frequency(motor, new_freq);
10        }
11    } else {
12        // Stop and cleanup PWM when reaching zero
13        motor_stop_and_cleanup(motor);
14    }
15 }
```

Listing G.1: Motor update timer callback