

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Development of an Object Pick-and-Place System with a SCARA Robot

João Pedro Gonçalves dos Santos

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Paulo José Cerqueira Gomes da Costa

October 18, 2022

This dissertation was co-supervised by professor José Luís Sousa Magalhães Lima, INESC TEC.

Resumo

Hoje em dia, os manipuladores robóticos são uma constante em vários setores. A sua versatilidade permite-lhes executar um conjunto de tarefas com melhor desempenho e de outra forma impossível para a sua contraparte humana. Esta adaptabilidade pode ser facilmente alcançada através das diferentes configurações que o robô pode possuir, que podem exigir a implementação de diferentes técnicas.

Esta dissertação visa a desenvolver o software de controlo de um manipulador SCARA montado num porta-paletes elétrico autónomo desenvolvido no âmbito de um projeto do INESC TEC. O braço robótico é responsável pela execução de movimentos *pick-and-place* de vários objetos, utilizando uma garra de vácuo.

Foi desenvolvida uma sequência de controlo com várias funcionalidades adicionais que permitem uma maior interação utilizador-sistema. Estas interações são possíveis graças à interface gráfica desenvolvida que permite exercer comandos sobre o manipulador e observar e extrair dados dos movimentos realizados. Além disso, é também desenvolvido um sistema de segurança para garantir o funcionamento em segurança do braço robótico.

O manipulador está equipado com vários encoders. Vários métodos são aplicados para corrigir as suas leituras, tais como filtro de mediana e calibração, de modo a obter medições mais corretas.

A configuração do manipulador proposta nesta dissertação torna-o altamente susceptível a vibrações devido às correias de borracha que unem as articulações ao motor. Por conseguinte, é essencial exercer um controlo suave. Existem atualmente várias técnicas para gerar trajetórias e perfis de movimento. Nesta dissertação, são implementadas duas soluções distintas, uma para o controlo linear no espaço das articulações e outra no espaço cartesiano. Esta última é gerada utilizando uma *Cubic Hermite Spline*. A primeira pode gerar três tipos de perfis de movimento: trapezoidal, *S-curve* tradicional e *S-curve* canónica.

Além disso, são realizados vários testes para verificar a validade do sistema desenvolvido, juntamente com uma análise dos efeitos dos diferentes movimentos gerados sobre as vibrações do sistema.

Abstract

Nowadays, robotic manipulators are a feature in various fields. Their versatility allows them to execute an assortment of tasks with better performance and otherwise impossible for their human counterpart. This adaptability can be easily achieved by the different configurations the robot can possess, which may require the implementation of other techniques.

This dissertation aims to develop the control software of a SCARA manipulator fitted onto an autonomous electric pallet truck previously developed in an INESC TEC project. The robotic arm is tasked to perform pick-and-place movements of a diversity of objects using a vacuum gripper.

A control sequence was developed with several additional features that allow for better user-system interaction. These interactions are made possible by the graphics interfaces that allow for the exertion of commands on the manipulator and observation and extraction of data from the carried out motions. Moreover, a simple safety system is also applied to guarantee the safe functioning of the robotic arm.

The manipulator is fitted with various encoders. Several methods are applied to obtain more correct measurements, such as mean filters and calibration.

The configuration of this dissertation manipulator makes it highly susceptible to vibrations due to the rubber belts connecting the joints to the motor. Therefore, the execution of a smooth control is essential. There are currently various techniques to generate trajectories and motion profiles. This dissertation implements two solutions, one for the linear control in the joint space and another in the Cartesian space. The latter is generated using a Cubic Hermite Spline. The former can generate three types of motion profiles: trapezoidal, traditional S-Curve and canonical S-Curve.

Furthermore, several tests are performed to verify the validity of the developed system, along with an analysis of the effects of the different generated motions on the system vibrations.

Acknowledgements

To my family, who always supported me throughout the years and never once made me question my decisions. To my father, Artur Santos, who awakened my engineering curiosity from a young age, and to my sister, Joana Santos, and my mother, Maria Arminda, who helped me achieve my dream.

To my partner, Sofia Maia, who supported me during this task and made it worth every second, thank you with all my heart.

To my supervisors, Paulo Costa and José Lima, thank you for all the aid and knowledge. I will be forever grateful for making me grow professionally.

To the members of the CRIS laboratory, thank you for the constant help provided. Special mention to André Baltazar and Alexandre Carvalho, who went out of their way to present me with all the assistance needed.

Thank you to my friends Eduardo Parracho and Carolina Marques, who assisted in the manipulator testing.

Lastly, to my dear friends who I made throughout the years, thank you all for making this 5-year journey worthwhile. I will always treasure these memories.

João Santos

“The way to succeed is to double your failure rate.”

Thomas J. Watson

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation and Problem Definition	2
1.3	Objectives	2
1.4	Document Structure	3
2	State of Art	5
2.1	Manipulator Composition	5
2.1.1	Stepper Motor	5
2.1.2	Rotary Encoder	7
2.2	Kinematics	7
2.2.1	Jacobian	9
2.3	Manipulator Control	9
2.3.1	Position Control	9
2.3.2	Trajectory Planning	12
2.4	Grippers	14
2.5	Safety	14
2.5.1	Post-collision	15
2.5.2	Pre-collision	15
3	Manipulator	19
3.1	Manipulator configuration	19
3.1.1	Kinematics	21
3.1.2	Jacobian	24
3.1.3	Workspace	25
3.2	Hardware	26
3.2.1	Stepper Motors	27
3.2.2	Drivers	27
3.2.3	Position Perception	28
3.2.4	Gripper	29
3.2.5	Controllers	29
3.3	Software	29
4	High-level Software	31
4.1	User Application	31
4.2	Raspberry Application	33
4.2.1	GUI	33
4.2.2	Features	34

5	Low-level Software	37
5.1	Arduino Application	37
5.1.1	State Control	37
5.1.2	Control	38
5.1.3	Trajectory Generation	39
5.1.4	Measurements	43
5.1.5	Additional Features	45
5.1.6	Safety	46
6	Results	49
6.1	Tests	49
6.1.1	Regular Operation	49
6.1.2	Impossible Movements	54
6.1.3	Motion Problem	56
6.1.4	Picking Action Timeout	58
6.1.5	Pause	58
6.1.6	Emergency Stop	61
6.2	Oscillation Analysis	62
6.2.1	MoveJ	62
6.2.2	MoveL	68
6.2.3	Different Movements Comparison	70
6.2.4	Stop Events	73
7	Conclusion and Future Work	79
7.1	Conclusion	79
7.2	Future Work	80
A	Manipulator Circuit Schematic	81
B	Arduino State Machine	83
	References	85

List of Figures

2.1	Schematic of a generic PID controller.	10
2.2	Schematic of a generic fuzzy controller for stepper control.	11
2.3	Three different types of motion profiles. Images adapted from ²	12
2.4	Different grippers.	14
2.5	Contact motion. Source: [1].	15
2.6	Proximity based safety measures.	16
2.7	Contact motion. Source: [2].	17
2.8	Kinect based safety measures.	17
3.1	Schematic of a SCARA manipulator with 3 DOF.	19
3.2	Manipulator used.	20
3.3	Joint movement range of the manipulator.	20
3.4	Schematic of the manipulator configuration.	21
3.5	Manipulator's top 2D view.	23
3.6	Manipulator workspace.	26
3.7	Hardware block diagram.	26
3.8	Vacuum gripper.	29
3.9	Flowchart of the developed control software.	30
4.1	User graphic interface.	31
4.2	Raspberry graphic interface.	33
4.3	Joint selection flowchart.	35
5.1	Arduino control flowchart.	38
5.2	Hermite spline basis functions. Source:	40
5.3	Joint 1 motor motion.	43
5.4	Tendency line for joint encoder offset.	44
5.5	Error before and after offset compensation.	45
6.1	Movement sequence performed.	50
6.2	Trajectory navigated during the sequence of <i>MoveJ</i>	50
6.3	Velocity and angle variation of the first motor on the different picking motions of the movement sequence.	51
6.4	Velocity of the fourth joint in <i>Move I</i>	52
6.5	Trajectory navigated during the sequence of <i>MoveL</i>	53
6.6	Velocity and angle variation of the first motor on the different picking motions of the movement sequence.	53
6.7	Fields verification warnings.	55
6.8	Out of bounds warning.	55

6.9	<i>MoveJ</i> with manual orientation warnings.	55
6.10	<i>MoveJ</i> with automatic orientation warning.	56
6.11	<i>MoveL</i> warning.	56
6.12	Collision obstacle.	56
6.13	End-effector path.	57
6.14	Joint velocity throughout the motion problem sequence.	57
6.15	Motion problem detection.	58
6.16	Warnings for picking timeout.	58
6.17	Pause state in GUI.	59
6.18	Charts the moment after the manipulator is paused.	59
6.19	Charts the moment after the manipulator is resume and reaches the destination.	59
6.20	Warning and trajectory after pausing while holding cargo.	60
6.21	Warning and trajectory after new placing coordinates.	60
6.22	Interface after entering the emergency state.	61
6.23	Trajectory and velocity of the manipulator after an emergency stop.	62
6.24	Velocity, acceleration, jerk and position profile of trapezoidal move type.	63
6.25	Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a trapezoidal motion profile.	63
6.26	Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a trapezoidal motion profile.	64
6.27	Velocity, acceleration, jerk and position profile of traditional S-curve move type.	65
6.28	Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a traditional S-curve motion profile.	65
6.29	Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a traditional S-curve motion profile.	66
6.30	Velocity, acceleration, jerk and position profile of canonical S-curve move type.	67
6.31	Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a canonical S-curve motion profile.	67
6.32	Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a canonical S-curve motion profile.	68
6.33	Velocity, acceleration, jerk and position profile of <i>MoveL</i> move type.	69
6.34	Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a <i>MoveL</i> move type.	69
6.35	Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a <i>MoveL</i> move type.	70
6.36	The first oscillation in error of the first joint of the trapezoidal motion profile with a 6 kg load.	71
6.37	Velocity and acceleration for pausing midway through a <i>MoveJ</i> movement with a canonical S-curve motion profile.	73
6.38	Variation in the encoder and stepper angle and their error during a pause while carrying a 0.5 kg load with a canonical S-curve motion profile.	74
6.39	Variation in the encoder and stepper angle and their error during a pause while carrying a 6 kg load with a canonical S-curve motion profile.	74
6.40	Velocity and acceleration for an emergency stop midway through a <i>MoveJ</i> movement with a canonical S-curve motion profile.	75
6.41	Variation in the encoder and stepper angle and their error during an emergency stop while carrying a 0.5 kg load with a canonical S-curve motion profile.	76

6.42	Variation in the encoder and stepper angle and their error during an emergency stop while carrying a 6 kg load with a canonical S-curve motion profile.	76
A.1	Schematic of the manipulator electric circuit.	82
B.1	Main state machine of the Arduino control algorithm.	83
B.2	State machines for additional features.	84

List of Tables

2.1	Variable and parameter definition. Source: [3].	6
3.1	D-H parameters.	22
3.2	Specifications of implemented motors ¹²	27
3.3	Motor drivers' configuration.	28
3.4	Theoretical maximum motor and joint velocity.	28
5.1	Motion Equation Parameters.	42
5.2	Motion Equations Parameters Definition.	42
6.1	End-effector position at ending of movement.	54
6.2	Motion Velocity Results.	71
6.3	Motion Performance Results.	72

Symbols and Abbreviations

AGVs	Autonomous Guided Vehicles
DOF	Degrees of Freedom
GUI	Graphical User Interface
IDE	Integrated Development Environment
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
PID	Proportional Integral Derivative
SCARA	Selective Compliance Assembly Robot Arm or Selective Compliance Articulated Robot Arm
SSI	Synchronous Serial Interface
UDP	User Datagram Protocol

Chapter 1

Introduction

This chapter provides the reader with the objectives of the dissertation in addition to the context in which was developed, alongside the document structure.

1.1 Context

Featured in a variety of fields, from medical to industrial, robotics is a constant in our everyday life. Since the introduction of automated systems, these sectors have increased exponentially in the last decades. These systems are mainly composed of AGVs and robotic manipulators. Nowadays, robotic arms can emulate the movement of humans with greater precision, speed and strength, allowing the execution of tasks otherwise impossible to their human counterparts. Despite the far better performance, these systems may sometimes be required to operate alongside workers. In these collaborative environments, the automated system is expected to continue to perform efficiently whilst not jeopardising the humans in the vicinity.

Behind every automated system, whether AGVs or manipulators, the movement is enabled through the generation of trajectories, which dictates how the system should behave in specific time frames, in order to move from the current to the desired point. This control is crucial and can greatly vary in complexity in accordance with specific variables, such as obstacle avoidance and motion smoothness.

Manipulators, an essential element in today's society, have been extensively researched and developed so as to present a system able to cope with highly demanding assignments. These robotic arms are composed of rigid structures (links) connecting the joints, which enable different movements according to their topology, providing motions such as linear movement with prismatic joints or rotational with revolut. Different combinations of joints and link lengths make it possible to attain various configurations for the manipulator. Therefore, selecting the appropriate design while considering the system requirements is imperative. Besides the actuators, additional sensing equipment may be fitted, enabling the manipulator to perceive its position, obstacles or objects of interest.

A sector where automated systems are a vital component of the process is logistics. With the constant growth of the industry, it is imperative that suppliers can rapidly dispatch the products to their customers. A commonly used method is the deployment of manipulators to perform pick-and-place tasks. These can be fixed or mobile, allowing the transportation of large volumes or individual objects. These autonomous systems must perform fast, precise operations to cope with the sector's fast pace.

1.2 Motivation and Problem Definition

Even though there are many manipulators available that can perform pick-and-place tasks, these systems tend to be costly and possess a high degree of complexity that may be redundant for the expected tasks, for example, deploying a six DOF manipulator for simple operations where part of the DOFs turn out redundant. Therefore, a more affordable and straightforward solution is favoured.

This dissertation provides an alternative manipulator capable of performing pick-and-place operations at a more reasonable price by employing an in-house (INESC TEC) built four DOF SCARA arm fitted onto an autonomous electric pallet truck. The manipulator is powered by stepper motors and possesses a vacuum gripper to handle the objects. As the manipulator was developed from the ground up, the control must be developed from the low-level (motor control) to the high-level (trajectory generation). The motion generated must be able to create smooth trajectories so as not to affect the manipulator, as its configuration (high elasticity of the joints) makes it highly susceptible to system vibrations.

1.3 Objectives

This dissertation aims to develop the control algorithm of a four DOF SCARA manipulator to perform pick-and-place operations. The algorithm developed must be able to operate smoothly throughout the entire duration of the motion whilst still maintaining precision and efficiency. The main objective can be categorised as follows:

- **Motor Control:** Algorithm capable of controlling the drivers powering the stepper motors.
- **Motion Control:** Algorithm used to generate the trajectory from the current to the desired position.
- **Pick-and-place sequence:** Algorithm to enable the execution of the pick-and-place task.
- **User Graphic Interface:** Interface employed for the user to interact with the manipulator.
- **Smooth Control:** The motions must be carried out while producing the fewer oscillations possible.

1.4 Document Structure

Besides the introduction, this document contains six more chapters:

- In chapter 2, is described the state-of-the-art and bibliographic review of similar projects, as well as the theoretical knowledge needed for a complete understanding of the dissertation.
- In chapter 3, a detailed description of the manipulator and its hardware is provided, in addition to the kinematic and dynamic study of the robotic arm.
- In chapter 4, the high-level software system is explained alongside the description of the different interfaces' features.
- In chapter 5, the algorithm implemented in the microcontroller is thoroughly described.
- In chapter 6, it is presented the results attained from the different executed tests and an analysis of the distinct motions in the system vibrations.
- In chapter 7, it is presented the conclusion drawn from this document, in addition to potential future work.

Chapter 2

State of Art

The purpose of this chapter is to present the reader with the technical knowledge needed for a full understanding of the dissertation, with the background explanation of the manipulator's components in Section 2.1 and kinematics in Section 2.2. In addition to the state of the art of the different thematics addresses in the dissertation, including the manipulator control in Section 2.3, grippers in Section 2.4 and finally safety measures in Section 2.5.

2.1 Manipulator Composition

For the manipulator to empower the movement of each joint, it needs to use a set of motors. The motors already fitted on the manipulator are hybrid steppers with coupled absolute encoders.

2.1.1 Stepper Motor

Stepper motors are electric motors capable of providing rotational movement through the conversion of electrical pulses. The ability to divide a full rotation into steps allows the motor to be positioned with great precision. The three main types of stepper motors are ¹:

- *Permanent Magnet* - Motors that are constructed, as the name suggests, with permanent magnets that interact with the stator electromagnet to produce movement and torque. They can achieve high torques with less power consumption when compared to other stepper motors.
- *Variable Reluctance* - Made of plain iron with a gear shape (presence of "teeth" around the rotor) instead of using permanent magnets. The presence of these protrusions allows the motor to achieve very high accuracy (more degrees of angular resolution), however with less production of torque.

¹Automate. "Understanding Stepper Motor Types & Operating Modes for Motion Control." automate.org. <https://www.automate.org/blogs/understanding-stepper-motor-types-and-operating-modes-for-motion-control> (accessed Feb. 11, 2022).

- *Hybrid Synchronous* - Combining both features from the permanent magnet and variable reluctance stepper, the hybrid stepper is able to achieve both high angular resolution and torque.

Besides the motor type, stepper motors can also be divided regarding the number of phases they possess. Being cheaper and capable of handling most requirements, the common practice is to bear only two phases. While a higher number of phases enables the motor to achieve a greater degree of angular resolution and a reduction in vibrations, the cost of the equipment tends to be higher.

Stepper motors possess 3 basic operating modes²:

- *Full Step* - During the full step, as both phases of the stator are energised, it provides maximum torque but is limited to the resolution to the number of teeth on the rotor.
- *Half Step* - Half step switches between having one or two phases of the rotor energised, providing twice the angular resolution with a reduction of produced torque.
- *Micro Step* - During this mode, the stator phases can either be fully, partially or not energised. The micro-step allows the motor to possess a significantly higher accuracy.

The motor model can be represented by the Equations 2.1 and 2.2 presented by [3] and [4], using the parameters from the table 2.1.

Table 2.1: Variable and parameter definition. Source: [3].

Variable	Description	Unit
v_a, v_b	Phase Voltage	V
i_a, i_b	Phase Currents	A
ω	Mechanical Speed of Rotor Shaft	rad/s
θ	Electrical Position of Rotor	rad
Parameter	Description	Unit
R	Resistance	ω
L	Inductance	H
K_t	Torque Constant	N.m/A
J	Rotor Inertia	N.m.s ² /rad
D	Damping Coefficient	N.m.s/rad
T_L	Load Torque	N.m
$K_d j$	Amplitude of j th harmonic	N.m
ϕ	Phase Shift of j th harmonic	rad
N_r	Number of teeth	-

²Automate. "Understanding Stepper Motor Types & Operating Modes for Motion Control." automate.org. <https://www.automate.org/blogs/understanding-stepper-motor-types-and-operating-modes-for-motion-control> (accessed Feb. 11, 2022).

$$L \frac{di_a}{dt} = v_a - Ri_a + K_t \omega \sin \theta \quad (2.1)$$

$$L \frac{di_b}{dt} = v_b - Ri_b + K_t \omega \cos \theta \quad (2.2)$$

The torque produced by the motor can be separated into three different components [3]:

- Electrodynamic (T_E) created by the interaction of the stator current and permanent magnet rotor.
- Cogging torque (T_C) produced by the teeth present in the configuration of the motor.
- Reluctance effect (T_R) is normally very insignificant in stepper motors, so it can be ignored.

$$T_E = -K_t i_a \sin \theta + K_t i_b \cos \theta \quad (2.3)$$

$$T_C = - \sum_{j=1}^{\infty} K_{dj} \sin(j\theta + \phi_j) \quad (2.4)$$

Resulting in motion Equations 2.5 and 2.6.

$$J \frac{d\omega}{dt} = (-K_t i_a \sin \theta + K_t i_b \cos \theta) - \sum_{j=1}^{\infty} K_{dj} \sin(j\theta + \phi_j) - D\omega - T_L \quad (2.5)$$

$$\frac{d\theta}{dt} = N_r \omega \quad (2.6)$$

Stepper motors have an inherent resonance problem, resulting in a momentary reduction of torque at some speeds, which leads to noise, vibrations and, in the worst-case scenario, loss of steps. This effect occurs when the step frequency coincides with the motor rotor's natural resonant frequency. To reduce the effect of resonance, a common practice is to operate the motor in micro-step mode [5].

2.1.2 Rotary Encoder

Rotary encoders are devices capable of perceiving angular position or motion and converting it into an output signal either through the use of mechanical, optical or magnetic systems. The encoders can be absolute, with the ability to maintain position information even when the device is powered off, or incremental when it only detects changes in position.

2.2 Kinematics

Kinematics is the field of studies that describes the motion (relationship between position, velocity and acceleration) of objects. Kinematics is divided in forward and inverse kinematics. The forward

kinematics allows the determination of the end-effector position and orientation in regards to the robot's joint angles. In contrast, inverse kinematics does the reverse, calculating the joint's angle to place the end-effector in a particular position and orientation.

In forward kinematics, the solution can be obtained through geometrical analyses or via a sequence of equations that relates the two different frames, resulting in the homogeneous matrix A . This last method is the most commonly used [6].

$$A_i = \begin{bmatrix} R_i & t_i \\ 0_{3 \times 3} & 1 \end{bmatrix} \quad (2.7)$$

Where i corresponds to the joint number, R to the 3×3 rotation matrix that relates the rotation between the two frames, and t is 1×3 matrix that relates the translation relation between both frames.

The most common method used to solve forward kinematics is known as DH convention. Where by applying the DH parameters, we can calculate the matrix. The DH parameters are the following four:

- a_i - link length (distance between o_i and z_{i-1})
- α_i - link twist (angle between z_i and z_{i-1})
- d_i - link offset (distance between x_i and o_{i-1})
- θ_i - joint angle (angle between x_i and x_{i-1})

Applying the parameters in the A matrix results in Matrix 2.8.

$$A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \sin\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.8)$$

Manipulators can be perceived as a chain of transformation matrices with the relationship between consecutive links. This relation can be obtained by multiplying the chain of transformation matrices, resulting in the matrix T . This allows, for example, the extraction of the end-effector coordinates in relation to the base link.

$$T = A_1 A_2 A_3 \dots A_n \quad (2.9)$$

Inverse kinematics is reached either as a result of using the inverse of the transformation matrices or per geometrical analyses, reaching the angles of the joint in relation to the end-effector's position. This last method is the most commonly used as it is less computationally expensive.

2.2.1 Jacobian

The jacobian allows to relate the velocity of the end-effector (ξ) with the velocity of the joints($\dot{q}$), resulting in Equation 2.10 and Matrix 2.11.

$$\xi(t) = J \dot{q}(t) \quad (2.10)$$

$$J = \begin{bmatrix} \frac{\delta f_1}{x_1} & \dots & \frac{\delta f_1}{x_n} \\ \dots & \dots & \dots \\ \frac{\delta f_m}{x_1} & \dots & \frac{\delta f_m}{x_n} \end{bmatrix} \quad (2.11)$$

Where $q_i = \theta$ for rotational joints and $q_i = d$ for prismatic joints. Having m and n as the number of functions and the number of variables respectively.

The velocity of the multiple joints can be obtained through the inverse of the jacobian according to Equation 2.12.

$$\dot{q}(t) = J^{-1} \xi; \quad (2.12)$$

2.3 Manipulator Control

Manipulator control is an intricate process that requires careful consideration of the methods used to accomplish the objective. Seeing that each manipulator can present different configurations, from the number and type of joints, the motors and sensors used, to the controller(s) utilised to run the algorithm, the best methodology for implementation varies. Nevertheless, one recurring practice should always be taken into account, separating the control algorithm into a low-level, controlling the actuators and read sensors and motion generation and a high-level for supervision and assisting the low-level system. This method reduces the complexity by having two different loops that run independent of each other but are in constant communication.

2.3.1 Position Control

Stepper motors possess the advantage of being controlled in either open-loop or closed-loop. Even though open-loop is cheaper and easier to implement, it is highly susceptible to step-out due to variation of load torques and operations at high speeds, as stated by Le *et al.* [3].

With the presence of a feedback device, like encoders, it's possible to operate the motor in a closed-loop by comparing the desired reference signal to the one obtained via the feedback device, correcting the position on the fly. This operation mode is known as stepper servo. Being first introduced in 1968 by Fredrisken [7], closed-loop control of stepper motors has been a subject of study for many years, and there are several methods available to tackle the problem in question. From the classic PID to newer and more advanced techniques that resort to fuzzy control and neural networks.

PID

The PID algorithm is one of the most widely used closed-loop control methods. The implementation simplicity and small computation required to run the algorithm are some of its most robust features [8]. It works based on a signal error, calculated by the difference in the reference signal to the feedback signal. Applying a correction with the proportional (P), integral (I) and derivative (D) terms as can be observed in Figure 2.1.

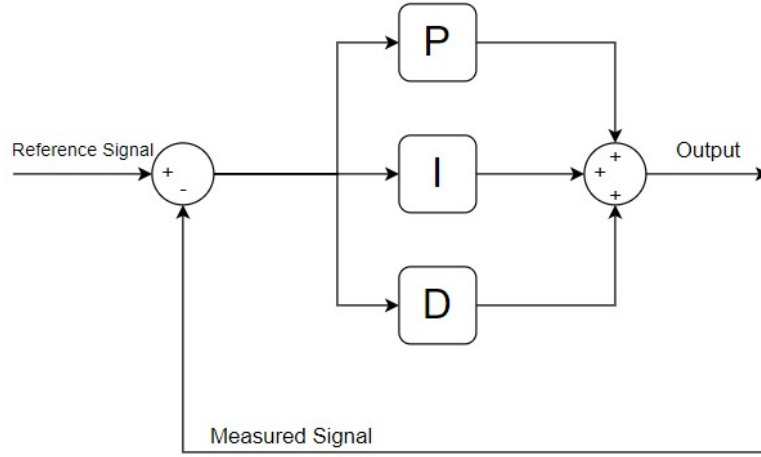


Figure 2.1: Schematic of a generic PID controller.

The output of the PID can be calculated in relation to the input with the in Equation 2.13 [8].

$$u(t) = K_p e(t) + \frac{1}{T_i} \int e(t) dt + T_d \frac{de}{dt} \quad (2.13)$$

For discrete time in Equation 2.14.

$$u(k) = K_p e(k) + K_I \sum_{j=0}^k e(j) + K_D [e(k) - e(k-1)] \quad (2.14)$$

Nevertheless, classic PID control algorithms do not possess a great response to system variations due to being a linear controller. To compensate for this loss, more recent algorithms have been developed by Elsodany *et al.* [9] and Ghafari *et al.* [10] that use Fuzzy logic to vary the values of the PID controller gains (K_p , K_I , K_D). Resulting in a faster response and smaller overshoot. Ho *et al.* [11] developed a Fuzzy Neural Network model for PID tuning achieving better results compared to other Fuzzy neural network performances and classic PID.

Another popular method is to use just the proportional and derivative component of the manipulator (PD), as a high value is usually given to the integrator component to reduce steady-state error and noise, leading to a worse transient performance and compromised stability [12]. Nevertheless, in the absence of this gain, the stability can be jeopardised when the manipulator dynamics

possess gravitational torque vector and friction [13]. To solve this problem, a model-based compensation can be implemented as seen in the PD+ tracking controller algorithm [14] with an outer compensation loop supplying a feedforward signal to the PD in the inner loop.

In 2014 Chevalier *et al.* [15] implemented PID-PI cascade controller to control motors on a mobile robot, where the PID took the input and the PID feedback, resulting in an output that is fed the information of the PI with the addition of disturbances and PI feedback. This algorithm provides a faster response by reducing disturbances. Nevertheless, this method is more complex and requires the tuning of two controllers and additional measurements for the inner loop.

Fuzzy Control

Fuzzy logic controllers allow the implementation of a control algorithm based on If-THEN rules. The fuzzy controller algorithm, Figure 2.2, consists of four steps. The fuzzification, where the membership functions are defined, can be triangular, trapezoidal and Gaussian. Then rules are determined based on IF-THEN conditions, followed by the inference mechanism where the rules' output is acquired through Mandami or Sugeno mechanisms. Finally, a defuzzification occurs where a control signal is achieved by employing defuzzification methods, such as centroid of area, maximum method, means of maximum or bisector of area [16].

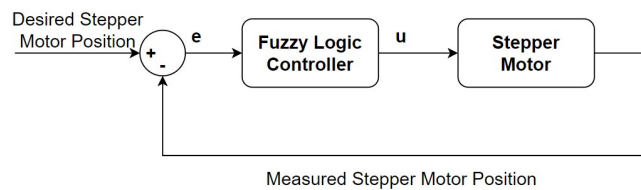


Figure 2.2: Schematic of a generic fuzzy controller for stepper control.

Salous [17] compares the fuzzy controller algorithm to a classic PID on stepper motors through Matlab simulation, reaching better results with the fuzzy logic controller as it was 40% faster to step response.

Motion profile

Controlled movements must be applied during the trajectory to ensure a smooth movement with a low jerk (rate of change of acceleration). Motion profile defines how the movement should be performed between points. There are three major motion profiles. The first one is triangular motion (Figure 2.3a), used when quick point-to-point moves are needed. It divides the motion into two moves, the first one with a constant acceleration of the motor and the second one with constant deceleration. This method provides high amounts of jerk, especially when the direction of acceleration is switched. The second method consists of a trapezoidal motion (Figure 2.3b). Where it's divided into three different phases. In the first one, an acceleration is applied, similar to the triangular motion. In the second phase, a constant velocity is maintained, and then deceleration

is carried out in the third phase. However, these solutions generate higher system vibrations and cause overshoot [18]. To minimise the effect of jerk, a method called S-curve can be implemented (Figure 2.3c). In this solution, in the first moment, an acceleration is gradually increased instead of accelerating immediately to the maximum value, then it is maintained constant before gradually reducing to switch to a constant velocity. After that, deceleration is gradually applied before maintaining it at a constant value, reducing this deceleration to finish the movement. This solution reduces system excitement [19], as the jerk generated is limited.

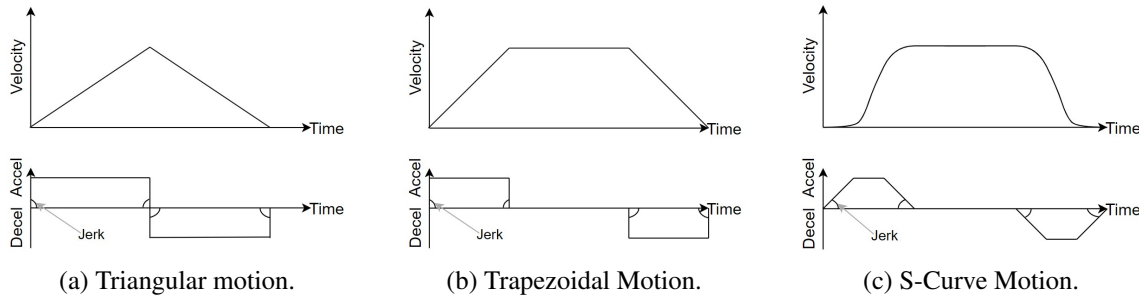


Figure 2.3: Three different types of motion profiles. Images adapted from².

Point-to-point s-curve profile generation can be achieved through a variety of methods. Piazzi *et al.*[20] proposed an approach based on cubic splines to obtain smooth and time-optimal motions. Higher-order splines have also been tested, reaching better results [19]. However, these methods tend to be computationally demanding. Macfarlane *et al.* [21] proposed a motion generation technique where a sine-wave is used to compute the way-points that are then connected by fifth-order polynomials, producing a smooth, jerk-bound trajectory. Besides time-optimal methods, minimum energy trajectory planning algorithms have been devised. In 2011 Huang *et al.* [22] presented a comparison between the energy consumed by an s-curve, a fifth-order, and a ninth-order motion profile. In 2004 Le *et al.* [23] devised a close control algorithm capable of implementing a velocity control using an S-curve motion profile. This method proved efficient in achieving smooth performance.

2.3.2 Trajectory Planning

The planning of trajectory can be classified in:

- Path planning - Description of the path taken from point A to B either in geometrical or mathematical terms.
- Trajectory planning - Besides the information provided by path planning, it adds the concept of time to the trajectory.
- Motion planning - In addition to the information from trajectory planning, it takes into consideration the kinematic and dynamics of the manipulator.

²Motion Control Tips. "What is a motion profile?" motioncontroltips.com. <https://www.motioncontroltips.com/what-is-a-motion-profile/> (accessed Feb. 17, 2022).

As the main focus of the dissertation is to develop a manipulator capable of performing a trajectory with the path already known *a priori*, only the concept of trajectory planning will be explored.

A trajectory can be generated either in Cartesian or joint space. In Cartesian space, points from the end-effector's path are considered, and a trajectory is calculated via a polynomial sequence. To plan the trajectory in the joint space, points from the end-effector path are extracted. Then, a kinematic inversion is applied to acquire the values of the robot joints from the position and orientation required of the end-effector at a certain point. With the robot joint angles determined, the trajectory is generated using interpolation sequences.

Trajectory in the joints space tends to be favoured as the control of the joints is made relative to their position rather than the end-effector position, making it easier to implement and avoiding the singularities and redundancy of kinematics. Even though motion control becomes more complex to implement[24].

Spline Functions

Splines are a method of interpolation that allows the determination of trajectories. Splines can be classified according to their properties in natural spline, Hermite spline, B-spline, etc.

In 1983 Lin *et al.* [25] implemented the cubic spline algorithm reaching a simple solution that minimises the manipulator's travelling time. By first creating cubic polynomial joint trajectories and then reducing the travelling time, taking into consideration the velocity, acceleration and jerk constraints. In 1990 Wang *et al.* [26] also developed a minimum time trajectory algorithm, this time through B-spline and flexible polyhedron search algorithm, considering the constraints. These solutions, being fourth-order, allow movements with continuous velocity and acceleration but a discontinuous jerk. By comparison, fifth-order splines allow for a continuous movement in both speeds, acceleration and jerk (but with jerk still being present at the end-point). In 2006 Gasparreto *et al.* [24] optimised the algorithm previously mentioned by implementing a fifth-order B-spline, achieving an efficient solution for a minimum time and minimum effort on the end effector. With the implementation of higher-order splines, the trajectory is guaranteed continuity of the directives, reducing the jerk, resulting in less wear of the robot and excitation of resonance [27] even though they tend to wander more between knots.

Simon *et al.* [27] described a trajectory planning algorithm base on the trigonometric splines. This method is computationally lighter as it results in fewer joint derivatives than algebraic splines, allowing to process each segment on the fly, meaning it can alter course mid-trajectory.

In 2004 Tian *et al.* [28] successfully implemented a simple Hermite cubic interpolation to construct the path of a two DOF manipulator capable of being adapted to an N DOF robotic arm. More recently, in 2018, Li *et al.* [29] implemented a quintic B-spline trajectory algorithm to a set of four DOF pick-and-place robots, similar to the one developed during the dissertation, achieving good results with smooth trajectory and very low vibrations of the end-effector.

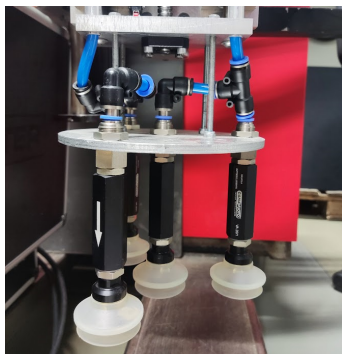
2.4 Grippers

The end-effector can be fitted with a variety of grippers. The most common grippers can be classified following their configuration as vacuum grippers, magnetic grippers or clamping grippers.

Vacuum grippers, Figure 2.4a, use the difference between the atmospheric pressure and the vacuum created to lift objects. These pressure changes can be achieved through vacuum pumps, ejectors, suction bellows or pneumatic cylinders. The main advantages are the fast grip-and-release function in which they can operate, and the ability to pick the objects in any position given the surface is regular enough to allow sealing of the vacuum cup. In 1995 Valanis *et al.* [30] devised a vacuum cup system to be fitted on AdeptOne and AdeptThree manipulators capable of handling fabrics.

Magnetic grippers, Figure 2.4b, being permanent-magnet or electro-magnet, are able to pick and release objects by magnetisation of the gripper and objects.

Clamping grippers, Figure 2.4c, apply pressure to the object's surface to pick them up. For irregular objects, soft grippers are usually used. Soft grippers typically possess a multitude of fingers allowing them to adjust to the surface of the object, keeping permanent contact with it. In contrast, hard grippers cannot adapt to the object surface, resulting in the risk of being subject to object rotation when handling them [31].



(a) Vacuum Gripper.



(b) Magnetic Gripper. Source:³.



(c) Clamping gripper. Source:⁴.

Figure 2.4: Different grippers.

2.5 Safety

In order to guarantee a jeopardy-free environment for the humans working in the vicinity of the robot, safety measures must be deployed. These methods can be classified as post-collision and pre-collision.

³Tulipe Tencologies[online].[viewed 07 August 2022].Available from:<https://www.tuliptechs.com/en/product/mg10-magnetic-gripper>

⁴BPLINFO, 2021. Commons Wikimedia[online].[viewed 07 August 2022].Available from: https://commons.wikimedia.org/wiki/File:Robotic_Claw_ROV_Gripper.png

2.5.1 Post-collision

Post-collision is when the impact detection is done as it occurs, and the system tries to reduce the impact.

Impact Sensing

The sensing of impact can be done by measuring the disturbances caused by the movement of the joints. In 2006, Luca *et al.* [1], proposed a collision detection system using the joint encoders to detect the impact, Figure 2.5. The impact sensing solution is not optimal as it only detects the impact as it is occurring, taking the risk of injuring someone or damaging something if the system cannot stop or reduce the inertia of the manipulator instantaneously in the moment of impact.

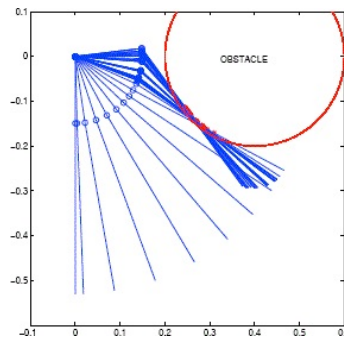


Figure 2.5: Contact motion.
Source: [1].

Impact Attenuation

A simple solution proposed by Heinzmann *et al.* [32] consists of reducing the impact force by limiting the torque produced by the manipulator's joints to operate in a safe region. This method proves inept as it requires the manipulator to move at slower speeds, reducing the process's efficiency.

2.5.2 Pre-collision

In pre-collision, the system is able to perceive dangerous situations reacting to them by either halting or changing the movement.

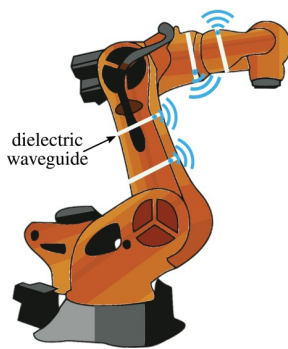
Proximity Sensors

The sensing before the impact with an object can be achieved by using proximity sensors. These proximity sensors can be:

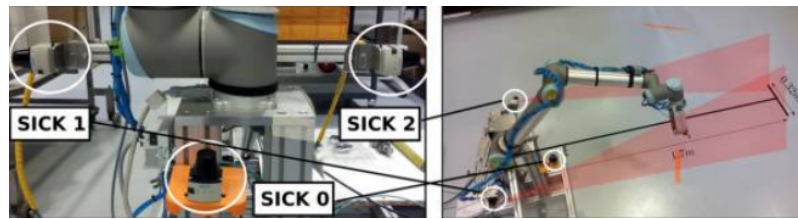
- Capacitive.

- Optic.
- Radar.
- Sonar.
- Mechanical switches.

In 2006 Geiger [33] used radar and flexible antennas to detect objects near the collaborative robots, Figure 2.6a. In 2018 Long *et al.* [34] developed a system based on lasers capable of perceiving objects that entered the robot's working area covered by the sensors, Figure 2.6b. Both solutions proved effective with sensors that require very little computational power to run and are easy to implement.



(a) Radar. Source: [33].



(b) Laser. Source: [34].

Figure 2.6: Proximity based safety measures.

Vision

Human tracking has been possible for a long time. Cameras allow the system to perceive the human presence and even acquire depth information (with stereo setup or additional sensors). These methods used to require an exceeding amount of computational power to execute the algorithms in real-time. Still, more efficient algorithms have been developed with the advancement in technology. One example is YOLO⁵, a real-time object detection algorithm capable of detecting and identifying a large variety of objects and even in newer versions is able to receive a data-set of images to learn and expand its database of objects that can recognise. Recently (2020), Gradolewsk *et al.* [2] implemented a safety system based on YOLO v3, Figure 2.7.

These systems are usually very efficient, especially nowadays, with the evolution of vision algorithms, as they tend to cover a wider area than the sensors. The drawback is that they usually require a high processing controller to run and do not present the proper certification needed to operate in an industrial environment.

⁵Joseph Redmon. "darknet". GitHub. <https://github.com/pjreddie> (accessed Feb 13, 2022)

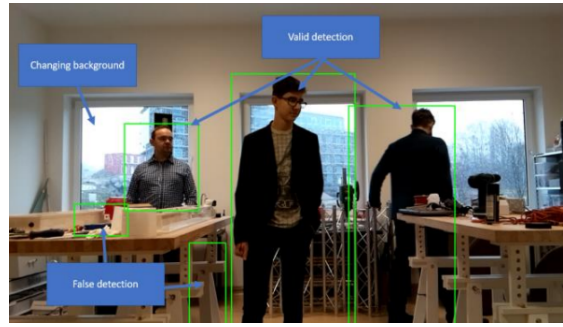
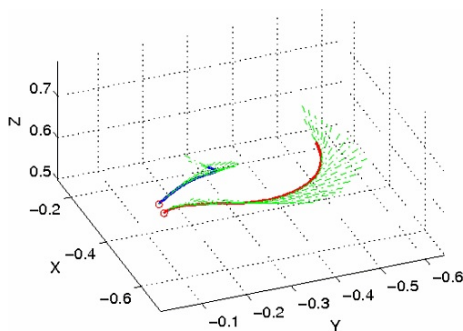


Figure 2.7: Contact motion. Source: [2].

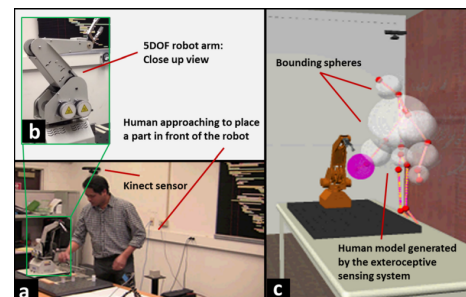
Depth Sensors

Depth sensors are highly used, being able to be utilised standalone or allied with vision software. One example of the first method is LiDARs. With the recent reduction of its price, their use in the robotic industry has increased significantly. LiDAR is a device capable of emitting a laser and measuring the time it takes to return the reflected light. Making them powerful components as they can perceive depth and create point clouds. A device that incorporates vision and depth-sensing is Kinect. Kinetic sensors contain an RGB camera and laser. The laser projects a pattern onto the scene, allowing the vision system to calculate the depth and surface information of the different objects. While the camera also enables the detection of the type of objects to be recognised.

In 2012 Flacco *et al.* [35] used a Microsoft Kinect sensor in a 7 DOF manipulator to determine the distance of objects to the robotic arm by creating repulsive vectors used in the execution of motion tasks, Figure 2.8a. In 2014 Morato *et al.* [36] developed a safety system based on multiple Kinects, Figure 2.8b. The solution consisted of detecting humans creating bounding spheres around their limbs that, if intersected by the manipulator, would lead to a halt in movement. This last solution, due to the creation of boundary boxes around human limbs, instead of repulsive camps of near objects like the first method, increases the algorithm's complexity. Nevertheless, both solutions are effective in maintaining safety.



(a) Repulsive vectors . Source: [33].



(b) Bounding Spheres. Source: [34].

Figure 2.8: Kinect based safety measures.

Chapter 3

Manipulator

This chapter aims to introduce the robotic manipulator used in this dissertation, with the analysis of its configuration in Section 3.1, followed by the description of the hardware in Section 3.2. As well as an overall view of the software architecture in Section 3.3.

3.1 Manipulator configuration

A traditional SCARA configuration is typically composed of three joints connecting the links, as seen in the example Figure 3.1, allowing it to possess three DOF. The first joint mimics the movement of the human shoulder, enabling it to rotate the arm in relation to the base. The second joint resembles the human elbow with a greater degree of rotation. The third joint, being prismatic, allows the upwards and downward movement of the manipulator's end effector.

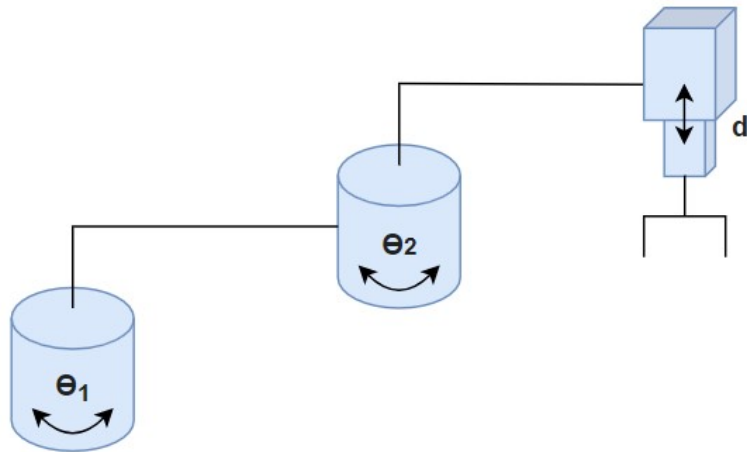
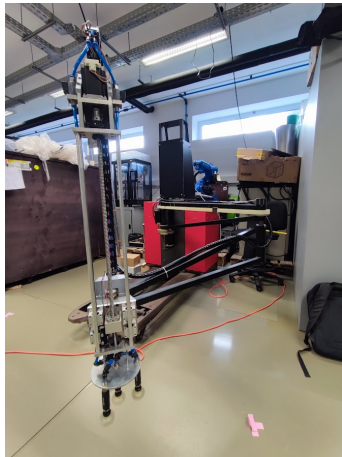
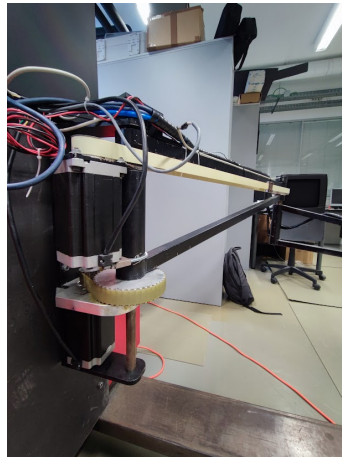


Figure 3.1: Schematic of a SCARA manipulator with 3 DOF.

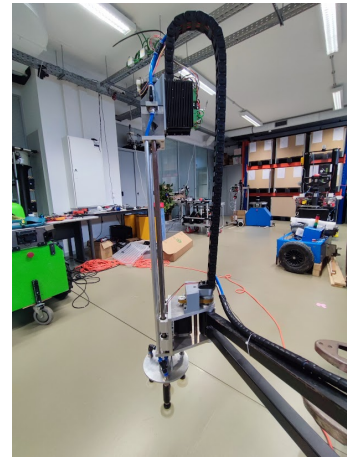
The robotic arm developed in this dissertation was constructed with a SCARA configuration with an additional rotational joint fitted before the prismatic joint, allowing it to possess four DOF, as seen in Figure 3.2.



(a) Manipulator.



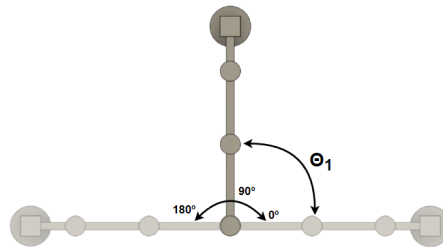
(b) Manipulator Belts.



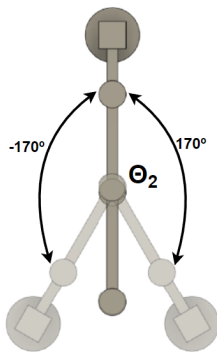
(c) End Effector.

Figure 3.2: Manipulator used.

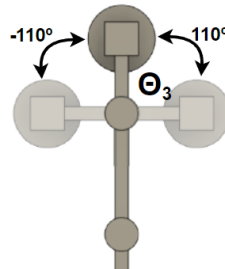
The manipulator movement is powered by stepper motors, which connect to the links via rubber belts as shown in Figure 3.2b. This simple configuration results in high susceptibility of the system to vibrations caused by the effect of jerk in the elasticity of the belts. Furthermore, each rotational link is equipped with a reduction gear where the belt is connected, resulting in more torque produced on the link and step precision. In addition, the last join is fitted with a lead screw which allows the rotational motion produced by the stepper motor to be transferred into linear movement on the Z axis as seen in Figure 3.2c.



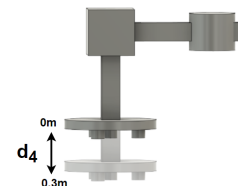
(a) Joint 1.



(b) Joint 2.



(c) Joint 3.



(d) Joint 4.

Figure 3.3: Joint movement range of the manipulator.

The configuration presented results in a distinct limitation for each joint. Each limit was chosen with a few degrees of margin to the actual real limitation in order to avoid potential self-collisions that could damage the system. As seen in Figure 3.3, the first joint (Θ_1) can rotate 180° . Whilst the second joint (Θ_2) with a rotational power of 340° is connected to the first one via a 1045 mm link (l_1) and to the third joint through a 935 mm link (l_2). The third joint can rotate 220° and is connected to the second joint through a 140mm link. The last joint being prismatic, is able to raise and lower the end effector by 300mm. The limit in the Z axis is set to allow it to travel until it reaches the floor, but further displacement is possible.

3.1.1 Kinematics

As presented in Section 2.2, kinematics can be divided into forward and inverse kinematics. The forward kinematics is used to perceive the position and orientation of the end-effector through the known parameters of the joints and the inverse to obtain the parameter values that the joints should possess to move to a particular pose (position and orientation).

Forward Kinematics

The solution for the forward kinematics can be obtained through a chain of transformations, from the manipulator base to the end-effector. By considering the joint axis relations presented in Figure 3.4 it is possible to extract the D-H parameters and present them in Table 3.1.

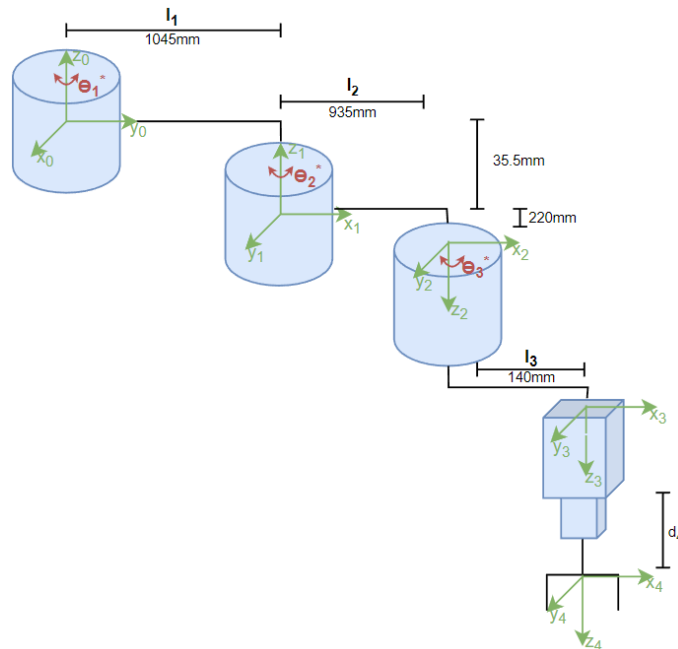


Figure 3.4: Schematic of the manipulator configuration.

With the D-H parameters is possible to formulate the transformation matrix A_i between consecutive links as displayed in Equations 3.1 through 3.3).

Table 3.1: D-H parameters.

Link i	a_i (m)	α_i (rad)	d_i (m)	θ_i (rad)
1	1.045	0	-0.0355	Θ_1^*
2	0.935	π	0.220	Θ_2^*
3	0.140	0	0	Θ_3^*
4	0	0	d_4^*	0

$$A_1^0 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 1.045\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & 1.045\sin\theta_1 \\ 0 & 0 & 1 & -0.0355 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

$$A_2^1 = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 & 0.935\cos\theta_2 \\ \sin\theta_2 & -\cos\theta_2 & 0 & 0.935\sin\theta_2 \\ 0 & 0 & 1 & -0.22 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

$$A_3^2 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0.14\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0.14\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$A_4^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

The T matrix can therefore be achieved using Equation 3.5, resulting in the T Matrix in 3.6.

$$T = A_4^0 = A_1^0 A_2^1 A_3^2 A_4^3 \quad (3.5)$$

$$T = \begin{bmatrix} s_3s_{12} + c_3c_{12} & -s_3c_{12} + c_3s_{12} & 0 & 0.14c_{123} + 0.935c_{12} + 1.045c_1 \\ -s_3c_{12} + c_3s_{12} & -s_3s_{12} + c_3c_{12} & 0 & 0.14s_{123} + 0.935s_{12} + 1.045s_1 \\ 0 & 0 & -1 & -0.2555 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

Where s_i and c_i corresponds to the sines and cosines of θ_i , respectively. While $c_{123} = \cos(\theta_1 + \theta_2 + \theta_3)$ and $s_{123} = \sin(\theta_1 + \theta_2 + \theta_3)$

From the previous matrix is possible to extract the position of the end-effector P and orientation Θ :

$$X = 0.14 * \cos(\Theta_1 + \Theta_2 + \Theta_3) + 0.935 * \cos(\Theta_1 + \Theta_2) + 1.045 * \cos(\Theta_1) \quad (3.7)$$

$$Y = 0.14 * \sin(\Theta_1 + \Theta_2 + \Theta_3) + 0.935 * \sin(\Theta_1 + \Theta_2) + 1.045 * \sin(\Theta_1) \quad (3.8)$$

$$Z = -0.255 - d_4 \quad (3.9)$$

$$\Theta = \Theta_1 + \Theta_2 + \Theta_3 \quad (3.10)$$

Inverse Kinematics

The inverse kinematic is used to attain the joint angles needed to position the end-effector in a specific position. In this problem's context, the latter's positioning can be achieved by decoupling the control of the last prismatic joint. This method reduces complexity by considering it as a three rotational links planar manipulator as seen in Figure 3.5.

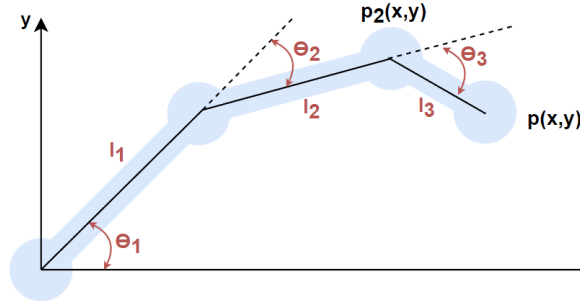


Figure 3.5: Manipulator's top 2D view.

Given the desired position for the end-effector in the X and Y axis with Θ orientation, it is possible to find the position coordinates of the second link extremity P_{2x} , P_{2y} in Equations 3.11 and 3.12.

$$P_{2x} = X - l_3 * \cos(\Theta) \quad (3.11)$$

$$P_{2y} = Y - l_3 * \sin(\Theta) \quad (3.12)$$

With P_x and P_y , it is then possible to find Θ_1 , Θ_2 and Θ_3 , the angle of the first, second and third joint, respectively in Equations 3.13, 3.14 and 3.15

$$\Theta_1 = \arctan2\left(\frac{P2_y}{P2_x}\right) - \arctan2\left(\frac{l_2 * \sin(\Theta_2)}{l_1 + l_2 * \cos(\Theta_2)}\right) \quad (3.13)$$

$$\Theta_2 = \arccos\left(\frac{P2_x^2 + P2_y^2 - l_1^2 - l_2^2}{2 * l_1 * l_2}\right) \quad (3.14)$$

$$\Theta_3 = \Theta - \Theta_2 - \Theta_3 \quad (3.15)$$

Where $l_1 = 1.045$ m, $l_2 = 0.935$ m and $l_3 = 0.14$ m, and represent the length of the first, second and third link, respectively.

3.1.2 Jacobian

The jacobian relates the joint velocities to the Cartesian velocity of the end-effector.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = J \begin{bmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ \dot{\Theta}_3 \end{bmatrix} \quad (3.16)$$

Where $\dot{X}$ and $\dot{Y}$ represent the velocity in the X, Y axis of the end-effector, $\dot{\Theta}$ the velocity in the end-effector angle and $\dot{\Theta}_1$, $\dot{\Theta}_2$ and $\dot{\Theta}_3$ the joint velocity of the first, second and third joint respectively.

In the context of this manipulator the jacobian matrix (J) can be determined by deriving the Equations 3.8, 3.9 and 3.10 in respect to each joint.

$$J = \begin{bmatrix} \frac{\delta(0.14c_{123}+0.935c_{12}+1.045c_1)}{\delta\Theta_1} & \frac{\delta(0.14c_{123}+0.935c_{12}+1.045c_1)}{\delta\Theta_2} & \frac{\delta(0.14c_{123}+0.935c_{12}+1.045c_1)}{\delta\Theta_3} \\ \frac{\delta(0.14s_{123}+0.935s_{12}+1.045s_1)}{\delta\Theta_1} & \frac{\delta(0.14s_{123}+0.935s_{12}+1.045s_1)}{\delta\Theta_2} & \frac{\delta(0.14s_{123}+0.935s_{12}+1.045s_1)}{\delta\Theta_3} \\ \frac{\delta(\Theta_1+\Theta_2+\Theta_3)}{\delta\Theta_1} & \frac{\delta(\Theta_1+\Theta_2+\Theta_3)}{\delta\Theta_2} & \frac{\delta(\Theta_1+\Theta_2+\Theta_3)}{\delta\Theta_3} \end{bmatrix} \quad (3.17)$$

Resulting in the Matrix 3.18.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} -0.14s_{123} - 0.935s_{12} - 1.045s_1 & -0.14s_{123} - 0.935s_{12} & -0.14s_{123} \\ 0.14c_{123} + 0.935c_{12} + 1.045c_1 & 0.14c_{123} + 0.935c_{12} & 0.14c_{123} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ \dot{\Theta}_3 \end{bmatrix} \quad (3.18)$$

Inverse Jacobian

The values of joint velocities according to the Cartesian velocity of the end-effector can be achieved using the inverse of the jacobian.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ \dot{\Theta}_3 \end{bmatrix} \quad (3.19)$$

Inverting the matrix J results in the Equation 3.20.

$$\begin{bmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ \dot{\Theta}_3 \end{bmatrix} = \begin{bmatrix} \frac{c_{12}}{l_1 s_2} & \frac{s_{12}}{l_1 s_2} & \frac{l_3 s_3}{l_1 s_2} \\ -\frac{l_2 c_{12} + l_1 c_1}{l_1 l_2 s_2} & -\frac{l_2 s_{12} + l_1 s_1}{l_1 l_2 s_2} & -\frac{l_3 (l_1 s_{23} + l_2 s_3)}{l_1 l_2 s_2} \\ \frac{c_1}{l_2 s_2} & \frac{s_1}{l_2 s_2} & \frac{l_3 s_{23} + l_2 s_2}{l_2 s_2} \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} \quad (3.20)$$

Singularities

During the control of a robot, some combination of joint positions might change the number of DOFs, resulting in the inability of the manipulator to perform accordingly and even causing deadlocks in the control algorithm [37]. These singularities are present in the Jacobian matrix and are determined using the determinant of the Jacobian matrix. The points of singularity are the ones which result in a determinant equal to zero.

$$Det(J) = 0 \quad (3.21)$$

$$l_1 l_2 \sin(\Theta_1 + \Theta_2) \cos(\Theta_1) - l_1 l_2 \cos(\Theta_1 + \Theta_2) \sin(\Theta_1) = 0 \quad (3.22)$$

The inverse jacobian singularities are determined as the determinant of the inverse jacobian matrix. The determinant of a matrix's inverse results in the determinant's inverse.

$$Det(J^{-1}) = \frac{1}{Det(J)} \quad (3.23)$$

$$\frac{1}{l_1 l_2 \sin(\Theta_1 + \Theta_2) \cos(\Theta_1) - l_1 l_2 \cos(\Theta_1 + \Theta_2) \sin(\Theta_1)} = 0 \quad (3.24)$$

Therefore, singularities occur when $\Theta_2 = 0$, meaning the second link is aligned with the first link.

3.1.3 Workspace

The workspace is defined as the area reachable by the manipulator. The configuration of the SCARA used in this series results in the working area presented in Figure 3.6. To plot the working area, a Matlab script was developed, where the joints' values varied within the joint's range. For each combination, the forward kinematics Equations 3.8 through 3.10 were applied in order

to extract the corresponding point coordinates so that the plot of all generated points was made possible.

It is worth mentioning that despite the ability of the manipulator to reach every point presented in Figure 3.6, specific values of X, Y, and Z coordinates and Θ orientation of the end-effector combined can fail to achieve a solution to reach the specified point, creating gaps in the workspace.

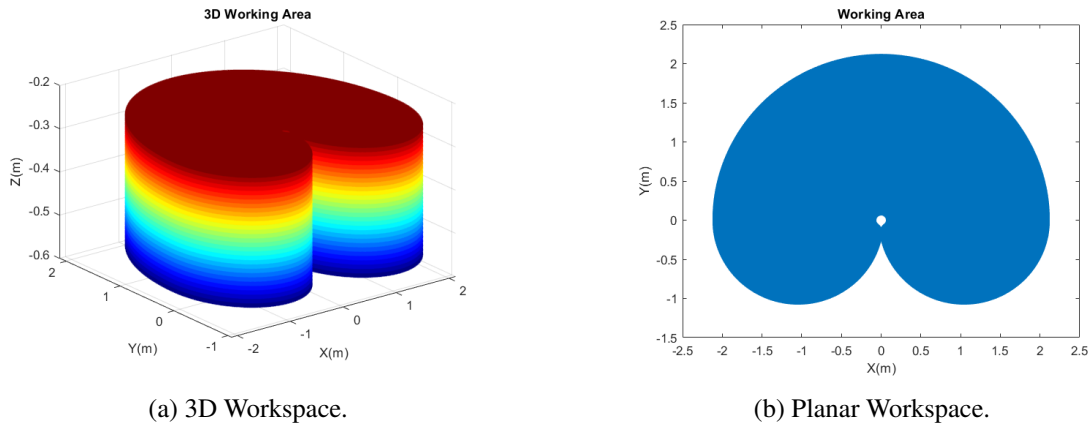


Figure 3.6: Manipulator workspace.

3.2 Hardware

The manipulator is composed of a variety of components that enable it to move and perceive the manipulator's position in its surroundings as seen in Figure 3.7. The schematic of all the components integrated into the robotic manipulator can be seen in Figure A.1.

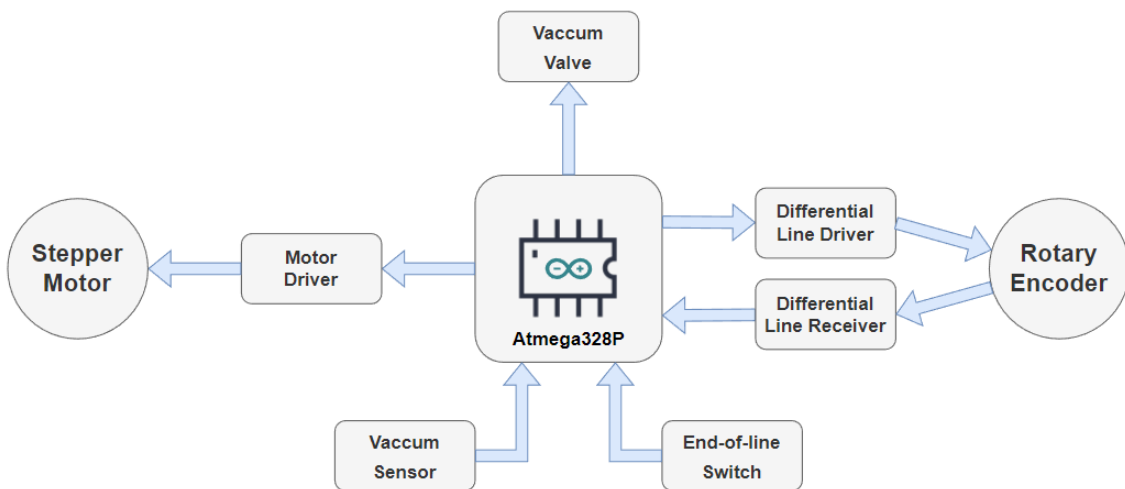


Figure 3.7: Hardware block diagram.

3.2.1 Stepper Motors

The manipulator is equipped with four bipolar hybrid stepper motors, the first two being 34HS59-5004S (NEMA 34), while the last two are smaller 103H7823-5740. The specifications of both motors can be seen in Table 3.2.

Table 3.2: Specifications of implemented motors ¹².

Motor	34HS59-5004S	103H7823-5740
Basic Step Angle (°)	1.8	1.8
Holding Torque (N.m)	13	2.7
Rated Current (A/phase)	5	2
Rotor Inertia ($kg.m^2$)	0.36×10^{-3}	0.84×10^{-4}

Due to the lengthy dimensions of the first two links, the first joints need to be able to produce high amounts of torque to cope with the weight of the end-effector structure and objects carried by it. Lower torque was needed for the last two motors because the links were shorter in length. In addition, motors with lower torque tend to be smaller and lighter, reducing the end-effector structure's weight. Hence the choice of stepper motors used.

3.2.2 Drivers

All stepper motors in the manipulator setup use the same model of driver, a DM556T driver³. The driver possesses two pairs of outputs, which allows the connection of the bipolar stepper motor, and three pairs of inputs, alongside the VDC and GND. These input ports are:

- PUL - Pulses generated by the microcontroller that make the motor take steps.
- DIR - To choose the direction of rotation.
- EN - To enable/disable the driver.

In addition, it is fitted with a DIP switch whose disposition allows for the driver's configuration: the microstep resolution, dynamic and standstill current configuration. The configuration implemented for each driver can be seen in Table 3.3.

Each step in the first three motors was divided into 32 microsteps allowing for a more precise motor positioning. Regarding the last motor, the microstep resolution is set to 4 as the screw lead ratio (step to displacement) is extremely low and therefore there is no need to divide the step into further microsteps.

The driver is able to receive a maximum input frequency of pulses of 200 kHz, resulting in a theoretical maximum velocity of the stepper motors of 1875 rpm and 15000 rpm for the first three

¹OYOSTEPER, "34HS59-5004S Full Datasheet Nema 34 CNC High Torque Stepper Motor 13Nm (1841oz.in) 5A 86x86x150mm - Oyostepper.com".

²SANYODENKI. s.d.. "2-Phase Stepping Motor SANMOTION F2 103H7823-5740". Accessed on 04/04/2022. <https://products.sanyodenki.com/en/sanmotion/stepping/f2/103H7823-5740/>.

³STEPPERONLINE, "User Manual DM556T 2-Phase Digital Stepper Drive"

Table 3.3: Motor drivers' configuration.

Motor	M1	M2	M3	M4
Microstep	32	32	32	4
Step/Revolution	6400	6400	6400	800
Peak Dynamic Current [A]	5.6	5.6	1.8	1.8
Standstill Current	FULL	FULL	FULL	FULL

motors and the last motor, respectively. However, this value is physically limited by the motor's properties. The maximum velocity for the NEMA 34 used was not found, though typically the values tend to range from 1000 rpm to 2000 rpm. The maximum velocity for the last two motors is defined by the manufacturer as 2200 rpm.

Nevertheless, the Arduino sends pulses in frequencies of 20 kHz to the drivers, resulting in a limitation of the maximum velocity of 187.5 rpm for the first three motors and 1500 rpm for the last one. With these values is possible to find the maximum joint velocity of each individually joint taking into account the reduction gears and lead screw as presented in Table 3.4.

Table 3.4: Theoretical maximum motor and joint velocity.

Joint	1	2	3	4
Max Motor Velocity (steps/s)	20000	20000	20000	20000
Max Motor Velocity (rpm)	187.5	187.5	187.5	1500
Gearing Ratio	3.75	2	2	200000 ⁴
Max Rotational Velocity (rpm)	50	93.75	93.75	0.0075
Max Joint Velocity	5,24(rad/s)	9,82(rad/s)	9.82(rad/s)	0.1(m/s)

3.2.3 Position Perception

For the system to be capable of perceiving the position of each individual joint, two different solutions are used for each type of joint. The rotational joints are fitted with RM44SC rotary encoders and the prismatic joint with an end-of-line sensor.

The RM44SC is a 13-bit SSI⁵ absolute magnetic rotary encoder with 8192 counts per revolution with a $\pm 0.5^\circ$ accuracy. In the communication between the encoder and the microcontroller, a differential line driver is applied to split the *clock* signal generated by the controller into *clock+* and *clock-*. In addition, a dual differential line receiver is implemented to convert the *data+*, and *data-* signals produced by the encoder into a single *data* that is then read by the microcontroller.

In the prismatic joint, a end of line sensor was fitted. The equipment is an SS-5GL2-FD Omron microswitch placed in such a way as to detect when the prismatic joint has reached the maximum possible height.

⁴One revolution results in a displacement of 5mm in the Z axis.

⁵Synchronous Serial Interface

3.2.4 Gripper

The system developed possesses a vacuum gripper composed of four suction cups, as shown in Figure 3.8. An air compressor motor is implemented to create the vacuum needed to lift the cargo. Whilst the motor is working, the airflow is controlled through an air valve. When an object is being gripped, a pressure sensor detects the change in vacuum pressure.

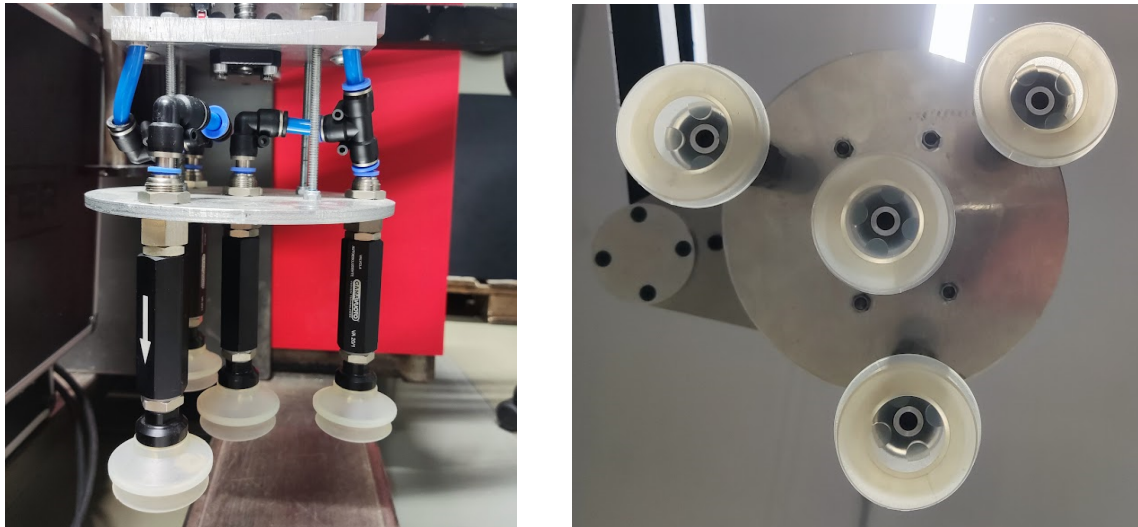


Figure 3.8: Vacuum gripper.

Using suction cups as grippers introduces some limitations on the handled object characteristics. For the gripper configuration to function properly, the load surface must be smooth, and the contact surface must be completely parallel to the gripper to produce perfect contact with the suction cups. The load weight must be no more than 6 kg so as not to damage the suction cups, whose material used, despite enabling them to adapt to the load surface also results in fragility on their part. Inability to comply with the requirements will most likely result in a load picking failure.

3.2.5 Controllers

To control the system, a microcontroller and a microcomputer are implemented. The microcontroller of choice is an Arduino Uno, which is based on the ATmega328p board, presenting a variety of digital and analogue pins that can be configured as inputs or outputs. The Arduino is in charge of the low-level control by managing all actuators and receiving information from the sensors deployed. While the microcomputer used, a raspberry PI 3 Model B+ is responsible for the supervision and assistance to the low-level control. Both controllers are in continuous communication through serial port communication in order to cooperate.

3.3 Software

The overall architecture of the system implemented can be observed in Figure 3.9.

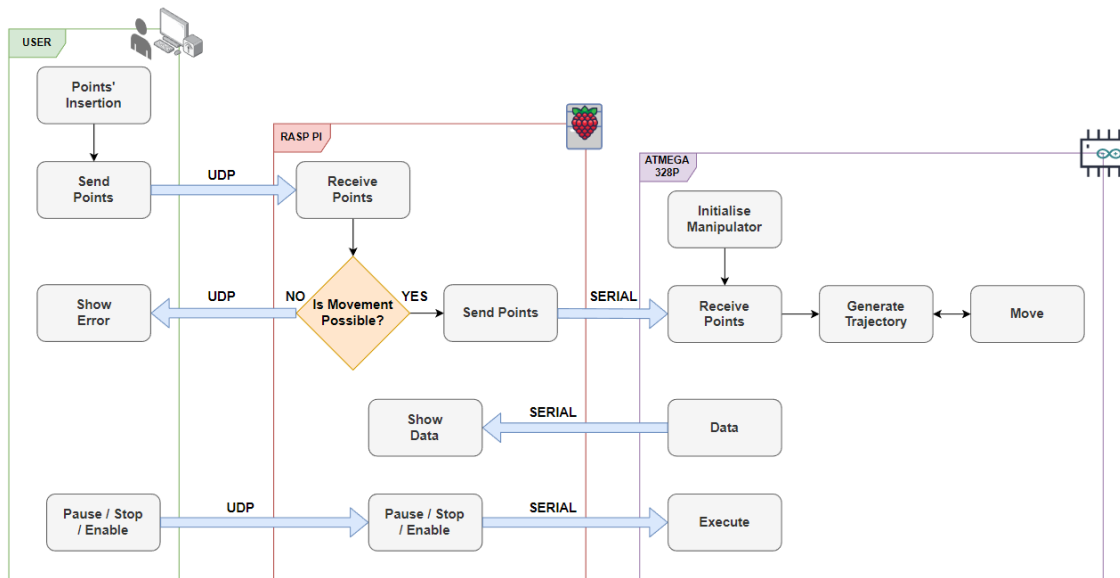


Figure 3.9: Flowchart of the developed control software.

The user application allows the consumer to interact with the system by defining the pick-and-place coordinates to be sent to the raspberry via UDP communication protocol. In the raspberry, the point coordinates are received and verified for possible problems in reaching the specified values, and then, if no issue was triggered, the values are sent to the Arduino controlling the manipulator through a serial communication protocol. Upon receiving the values, the Arduino generates the manipulator's trajectory and moves the stepper motor accordingly. Furthermore, the raspberry constantly receives various control information from the Arduino, from sensor values to internal variables. Moreover, the user and raspberry application can pause or stop the manipulator midway through the movement and enable/disable the drivers when stopped or while waiting for new points.

The system is divided into high and low-level applications. The high-level consists of user and raspberry applications, which were both developed using Lazarus IDE. Lazarus is a powerful tool that allows the creation of GUIs easily, using Pascal as the programming language. The Arduino, a low-level application, was accomplished in C++ using Visual Studio Code with the PlatformIO extension.

Chapter 4

High-level Software

This chapter displays the methodology used to develop the control software implemented in high-level applications, such as the user, in Section 4.1, and the raspberry pi application, in Section 4.2. The graphical interfaces and features of these applications are presented in the chapter.

4.1 User Application

The user application, seen in Figure 4.1, is primarily focused on user-manipulator interaction by sending the coordinates of the pick-and-place points to the manipulator and displaying information about the state of the manipulator movement.

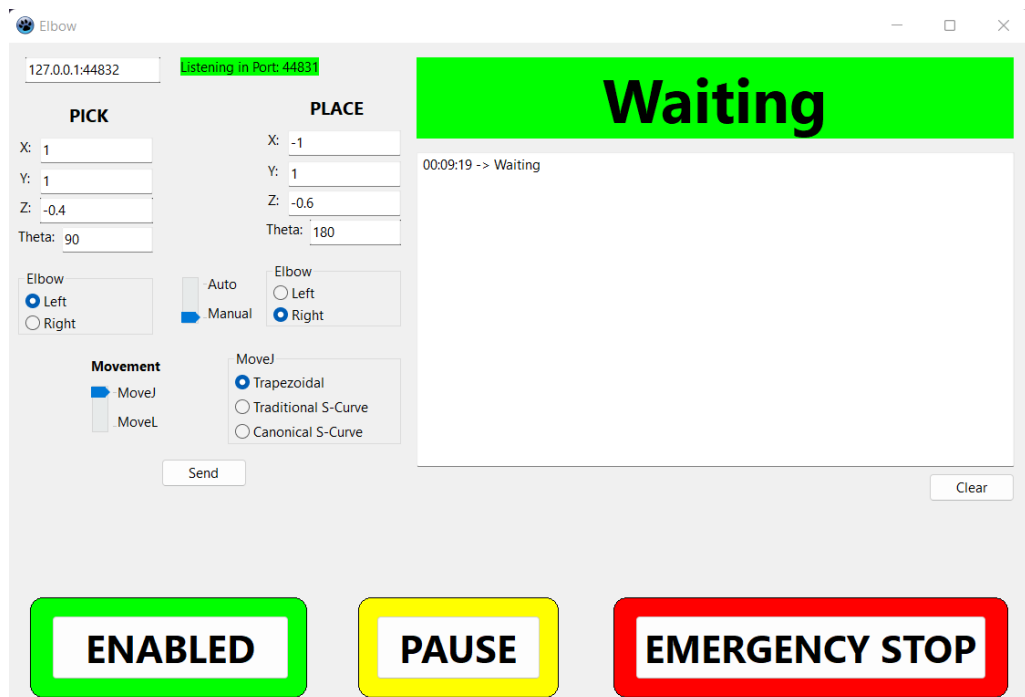


Figure 4.1: User graphic interface.

In the left top corner, the consumer can fill the blank boxes with the desired coordinates values of the end-effector position for the pick-and-place task in meters for X, Y and Z, and degrees for Θ . Furthermore, the user can choose between an automatic or manual selection of the "elbow" orientation of the manipulator. In addition, a choice of movement is also provided:

- **MoveJ** - This movement type provides a linear motion in the joint space. Moreover, the user can select between a trapezoidal, traditional or canonical S-Curve motion profile.
- **MoveL** - This movement type implements a linear motion in the Cartesian space.

The interface is also fitted with a label (top right corner) which displays the system's current state. Fluctuating between¹:

- **Disconnected** - When the user application is not connected to the Arduino.
- **Initialising** - When the manipulator is in initialisation.
- **Waiting** - When the manipulator is waiting for new points to perform another motion.
- **Operating** - When the manipulator is in motion.
- **Paused** - When the manipulator is in a pause state. The pause can be activated at any motion stage, enabling the user to resume or cancel the movement while in this state.
- **Stopped** - When the manipulator is stopped.
- **Cancelled With Load** - When the manipulator movement is cancelled whilst holding cargo. The user must supply the new coordinates to deliver the gripped load.
- **Stopped With Load** - When the manipulator movement is stopped whilst holding cargo. The user must manually remove the load to return from the stopping state by pressing the button to release the object.
- **Motion Problem** - When the difference between encoder and stepper motor angle of any rotational joint is superior to a threshold, meaning that the manipulator collided with something or was unable to move the load. The manipulator enters a pause state, allowing for the resume of the manipulator motion or cancellation of the same.
- **Pick Timed Out** - When the manipulator cannot pick the load within five seconds, counting when the lower movement has been completed. The manipulator stops and the user must press resume to continue.

All user interactions with the manipulator and the warning returned from the raspberry application are recorded in a log on the right side of the interface. Furthermore, the user can pause the manipulator movement, which gradually slows it down until it stops, enable/disable the stepper

¹More detailed description is supplied in chapter [5.1](#)

motors drivers, which allow the manual positioning of the manipulator, and press emergency stop, that instantaneously stops the manipulator. The triggering of various events, such as "Pick Timed Out" and "Motion Problem", result in the appearance of a pop-up window with the instruction for the user on how to proceed, as well as working as a way of forcing the acknowledgement of the current event.

4.2 Raspberry Application

The application implemented in raspberry serves as a bridge between the user application and the Arduino, receiving the commands from the user application, verifying if the points are reachable and then sending them to the Arduino. It also possesses a GUI showing a variety of information and charts.

4.2.1 GUI

The interface of the raspberry application, as seen in Figure 4.2, displays several functionalities. One example is a table showing the value of different values received from the Arduino, such as velocity and position from the other motors.

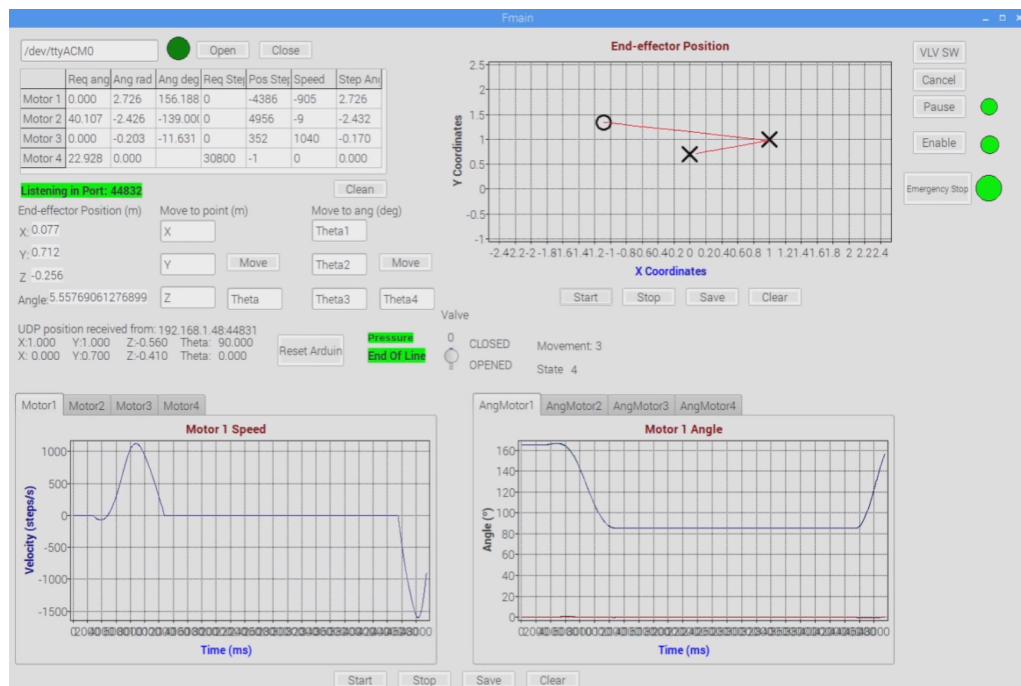


Figure 4.2: Raspberry graphic interface.

Furthermore, three distinct charts are visible, one set for motor velocity in steps/s, and another for the motor's position in degrees, both with respect to time. Finally, the last one illustrates the end-effector coordinates on the X and Y axis. In addition, a possibility is presented to move the

end-effector directly to a specific angle or position for testing purposes. It also displayed the current end-effector position and the information received via UDP. Lastly, a pause/enable/emergency stop control can be executed, similar to the user application.

4.2.2 Features

The raspberry application also functions as a communication bridge by connecting the user application to the Arduino, controlling the manipulator. Furthermore, it possesses features that allow the validation of the pick-and-place points to be sent to the robotic arm.

Communication

The communication between the user and the raspberry application is done via User Datagram Protocol (UDP). The choice of communication protocol relied upon the ability to quickly implement the UDP server and client in Lazarus using the *lnet* package. Moreover, the UDP protocol possesses a more efficient and faster communication than the TCP (Transmission Control Protocol), which is significantly slower due to the need to "acknowledge" the received packages [38]. Even though data can be lost via a UDP connection, package loss should not occur because the connection is expected to be accomplished within the local network.

The Arduino Uno is a board that can only communicate with external devices via serial communication. In order to implement the said protocol, an Arduino library "channels"² was used, where a frame is sent, consisting of an identification character followed by eight hexadecimal nibbles, with the values to be transmitted. Transmissions are sent in 50 milliseconds of intervals.

Pick-and-place Point Validation

The validation of the pick-and-place points is a crucial task of the raspberry application, as it guarantees that the point is reachable and does not overstep the joint limitations, as failure in detecting these problems could result in hardware wear and damage. The first verification performed in both user and raspberry applications is to verify if the intended end-effector coordinates are within the maximum reachable distance of the manipulator. If the point is within the robotic arm reach, another verification is executed where it is checked if the point is reachable with the specified combination of angle and coordinates received. Because the movement *MoveJ* is generated in the joint space and *MoveL* in the Cartesian space, the methods for validation of the pick-and-place points differ.

In the *MoveJ*, the user can opt to automatically or manually set the orientation of the "elbow", the second joint. If the manual mode was selected, the raspberry application finds the inverse kinematics solution and adjusts the Θ_2 angle to correspond to the selected "elbow" orientation. If the solution is not possible, the system warns the user application. In automatic mode, the process is slightly more complex, as a series of calculations and verification is needed to find the best

²P33a. "channels". GitHub. <https://github.com/P33a/MeArmControlVarSpeed/blob/master/channels.cpp> (accessed Ago 21, 2022)

solution, which can be seen in Figure 4.3. Where the Equations 3.13 through 3.15 are applied and the value of Θ_2 is fluctuated between its positive and negative value in order to find a solution.

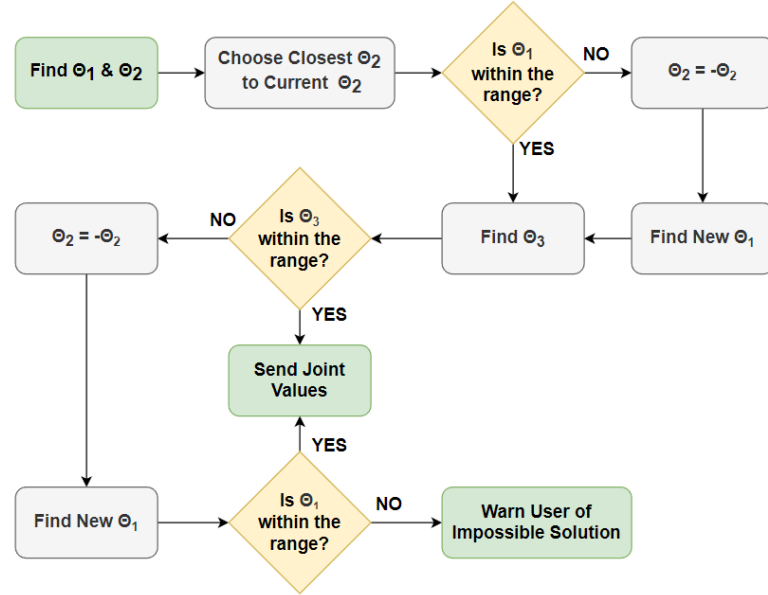


Figure 4.3: Joint selection flowchart.

Since *MoveL* needs to deal with the inverse of the Jacobian, it verifies if the end-effector would cross a singularity point during the motion. The singularity of the manipulator exists when Θ_2 equals zero, in layman's terms, when the second link is aligned with the first one. By assuming this alignment, it is possible to verify that this singularity would arise if the line connecting the end-effector starting and end point intersects the circumference with a radius of 1.98m, the combined length of the first and second link. Another validation executed for the *MoveL* algorithm consists of verifying if the manipulator exceeds any of the joints' limitations during the movement. The algorithm developed computes the inverse kinematics of the manipulator in ten equally spaced points of the line segment connecting the origin and destination coordinates, essentially estimating the manipulator's position at those different points to inspect if the solution is possible at each one.

Chapter 5

Low-level Software

This chapter displays the methodology used to develop the control software implemented in the low-level application, with a detailed description of the procedures and the techniques used to produce measurements and generate trajectories.

5.1 Arduino Application

The developed Arduino application functions as the manipulator controller. During execution, it exchanges information in intervals of 50 milliseconds with the raspberry application. This cycle value was chosen as during development it was visible that the raspberry wasn't able to compute all information with cycles lower than 50 milliseconds. Through serial communication are received the type of motion required, as well as the pick-and-place point coordinates for a *MoveL* motion and joint angle for a *MoveJ*. In the case of *MoveJ*, the type of motion profile is also acquired (Trapezoidal, Traditional S-Curve and Canonical S-Curve). These points are added to a list by order of reception. Moreover, commands are also received to either pause or stop the motion, disable/enable the motors' drivers and manual control the air compressor's valve, which produces the vacuum needed for gripping the cargo.

5.1.1 State Control

The controller is required to perform a series of pick-and-place motions. Therefore, a state control must be implemented with the ability to execute the various movements taking into consideration the desired motion type, duration and possible commands received from the user. The sequence of processes that compose the state machine can be observed in the flowchart of Figure 5.1 and with more detail in the state diagrams of Figure B.

Upon system boot, an initialisation is performed, where the prismatic link is raised to its maximum height, stopping when the end-of-line sensor is activated and setting the terminated position as its reference. When the initialisation process is concluded, the manipulator moves into a waiting state, where it remains if the list of points is empty. Otherwise, the controller extracts the first element and verifies the motion type. In the case of an "L" type point, a *MoveL* is performed.

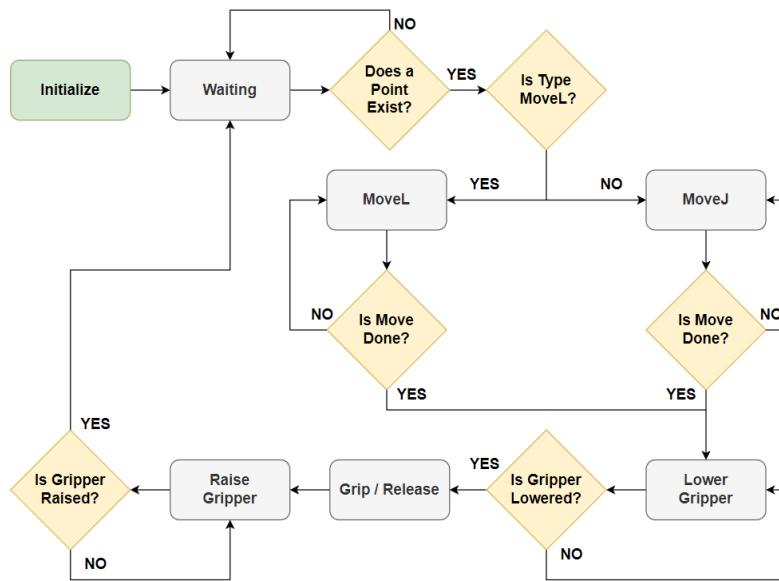


Figure 5.1: Arduino control flowchart.

On the other hand, if it is not an "L" type, the algorithm for *MoveJ* is executed instead. Both methods result in a trajectory generation in the planar plane formed by the first three links. The fourth joint is activated when the manipulator is positioned above the desired point, descending to the required height. After the motion completion, the gripper is triggered by opening the vacuum valve, which promptly grasp the load and raise the end-effector, returning to the waiting state upon completion, repeating the process for the next point on the list, if available. However, during the next move, the manipulator releases the gripped load rather than grasping it.

5.1.2 Control

Motor control can be achieved in either open-loop or closed-loop. With the presence of feedback sensors (encoders) in the system, it would be expected that the method of control chosen would be closed-loop. However, the approach implemented was based on open-loop control, justified by the use of stepper motors and other external factors.

Stepper motors are usually used as open-loop devices. Due to their working principles of increment of steps per pulse, it is possible to track the motor's position at any given time and position it with a high degree of accuracy without any feedback equipment. The only drawback of this type of control is the possibility of suffering a loss of steps during motion, which renders the tracking inaccurate. This phenomenon can be caused by a high acceleration of the motor or insufficient torque produced to move the load. In the first case, motion profiles are implemented to gradually change the acceleration throughout the motion to minimise the probability of losing steps during this phase, avoiding step loss. In the case of lack of torque, the only solution would be to switch the drivers/stepper for a higher torque combination or fit a higher reducing gearing

ratio. In both cases, developing a closed-loop control would prove redundant and unable to solve any of the stepper motor difficulties.

Furthermore, the manipulator configuration results in vibrations throughout the movement, which causes oscillations in the encoders' measurements, resulting in a subsequent oscillation in error (difference between the encoder and desired value). This instability in error, alongside the finding previously mentioned, renders the closed-loop approach ineffective.

Therefore, the stepper motors' angle is used throughout the developed control algorithm as a reference for all motions. These values are obtained through an internal counter that increments/decrements with every pulse generated from the microcontroller, depending on the direction intended. Nevertheless, the algorithm still benefits from the feedback devices implemented. Both encoders and end-of-line are used to initialise the stepper motor angles. Moreover, after every encoder measurement, the stepper motor angle value is updated unless the system is currently performing a *MoveJ* or *MoveL*, for the previously mentioned reasons. Through frequent updates in the motor's angle values, the system can detect a collection of phenomena. Among these, the loss of steps or a disturbance of the manipulator position caused by external agents. The perception of these disruptive events enables the user to interact with the system in order to overcome them, either during the motion or through a correction in the next movement.

Stepper Control

The stepper motor control is achieved by implementing a library, developed for the control of stepper motors of a CNC machine, adapted to the current manipulator configuration. The algorithm executes an interrupt every 50 microseconds to control the motors by selecting the correct direction to output to each driver and generate the impulses for each motor according to the specified velocity asked for each one.

The velocity values achieved during the different trajectory generation are initially in rad/s (joint) and must be converted to step/s (motor), considering the gearing ratio of each joint and microstep of each motor.

5.1.3 Trajectory Generation

The manipulator movement is performed between two pre-defined points, the current position and the specified destination. The method used to devise the motors' behaviour during the motion can vary according to the system requirements. In this dissertation, the user can opt between two different trajectory generation approaches, a linear motion in the joint space (*MoveJ*) and linear motion in the Cartesian space (*MoveL*). Both methods operate with the same average speed between two points of 0.2 m/s, which means that, theoretically, the manipulator completes the movement within the same time frame regardless of the employed motion type. The points in *MoveJ* are received in joint angles (radians) and *MoveL* in coordinates, with X and Y in meters and Θ in rad. However, in both cases, Z is received in meters. For the height control of the

end-effector, the *MoveJ* method is always applied, independently of the motion type previously selected.

MoveL

The generation of linear motion in the Cartesian space can be achieved through a variety of approaches. As the algorithm was implemented in an Arduino, which possesses some hardware processing limitations, the method applied needed to be computationally light, with fewer complex expressions as possible for online calculations. Therefore despite possibly producing more oscillations, the method for point interpolation should not exceed a third order polynomial, as the derivatives needed for higher order polynomial would result in more intricate solutions. The decision relied on what method to use, limiting the interpolation to third-order polynomials. After a thorough review of possible approaches, the one implemented was the cubic Hermite spline, as it possesses the lower computational cost and resulted in the most elementary expression.

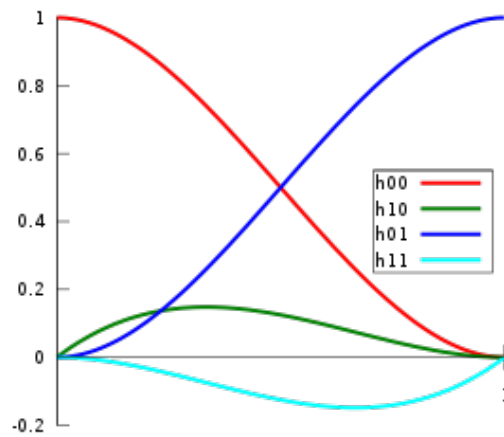


Figure 5.2: Hermite spline basis functions. Source: ¹.

Using the cubic Hermite spline, it is possible to describe the position $p(t)$ at any given time t in the generated trajectory between two points, as seen in Figure 5.2, using the Equation 5.1.

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1 \quad (5.1)$$

Where p_0 and p_1 represent the initial and final position, respectively, and m_0 and m_1 the starting and finishing tangent. In the expression it is assumed that t varies from $t = 0$ to $t = 1$.

As the stepper motor control is achieved by defining the rotational velocity, a derivative of the cubic Hermite spline is performed within a Δ_t assigned as the time required to perform the motion, computed with the distance between the two points and specified average speed. Thus resulting in

¹Figure from ElectroKid Berland, 2007, <https://commons.wikimedia.org/wiki/File:HermiteBasis.svg>

$\Delta_t = \frac{|P_f - P_i|}{V_{avg}}$, where P_f represents the final destination (X and Y coordinates) and P_i the initial. The derivative of the Equation 5.2 that results in the velocity equation can be seen in Equation 5.2.

$$v(t) = \left(\frac{6t^2}{\Delta_t^3} - \frac{6t}{\Delta_t^2}\right)p_0 + \left(\frac{3t^2}{\Delta_t} - \frac{4t}{\Delta_t} + \frac{1}{\Delta_t}\right)m_0 + \left(\frac{-6t^2}{\Delta_t^3} + \frac{6t}{\Delta_t^2}\right)p_1 + \left(\frac{3t^2}{\Delta_t^3} - \frac{2t}{\Delta_t^2}\right)m_1 \quad (5.2)$$

Setting m_1 and m_0 to zero allows the generation of a linear movement in the Cartesian plane between p_0 and p_1 . Resulting in the elementary Equation 5.3.

$$v(t) = \left(\frac{6t^2}{\Delta_t^3} - \frac{6t}{\Delta_t^2}\right)p_0 + \left(\frac{-6t^2}{\Delta_t^3} + \frac{6t}{\Delta_t^2}\right)p_1 \quad (5.3)$$

To propel the manipulator to follow the straight line defined by the current and destination point, a Hermite for each control variable must be computed. This means that throughout the motion, the Hermite velocity Equation 5.3 must be individually applied for X , Y and Θ . Achieving the values of velocity in the X axis (V_x), Y axis (V_y) and Θ (V_Θ).

As the control is done in the joint space, the velocities obtained in the Cartesian space must be converted to the joint space using the inverse Jacobian relation displayed in Equation 3.20. Resulting in the joint velocity in each joint fed to the stepper motors control.

MoveJ

The implemented *MoveJ* method employs the approach devised by Arevalo [39] where a motion profile is generated between the initial and final point using a pre-defined maximum velocity and maximum acceleration, where the velocity at time t , $v(t)$ are computed using the Equation 5.4. This method of profile generation was chosen on account of its capability of generating various types of motion profiles by simply changing a parameter value (γ), in addition to its straight-forward implementation. Furthermore, the method presents a computationally light approach by computing all parameters prior to the motion leaving just the calculus of joint velocity through the use of Equations 5.4. The expressions generate velocity values until midway through the motion, mirroring them afterwards, hence the recursive properties presented in the equations.

$$v(t) = \begin{cases} 0 & \forall t \leq 0 \\ \left(\frac{A_m}{2}\right) * \left(\frac{t^2}{\tau}\right) & \forall 0 < t < \tau \\ \left(\frac{A_m}{2}\right) * (2 * t - \tau) & \forall \tau < t \leq T_o \\ V_w - v(T_a - t) & \forall T_m < t \leq T_s \\ v(T_t - t) & \forall T_s < t \end{cases} \quad (5.4)$$

The *MoveJ* algorithm is generated for each joint individually using the current and desired motor position to generate the parameters presented in Table 5.2, whose equations are shown in Table 3.1.

For the first three joints in *MoveJ* motions, the current joint and destination angles are used to compute the value of Y_f . Furthermore, the total destination travel time (T_t) is calculated based on

Table 5.1: Motion Equation Parameters.

$Y_s = \frac{Y_f}{2}$	$Y_{aux} = \frac{1}{2} * (1 + \gamma) * (\frac{V_m^2}{A_m})$
$Y_a = \begin{cases} Y_s & \forall Y_s \leq Y_{aux} \\ Y_{aux} & \forall Y_s > Y_{aux} \end{cases}$	$V_w = \begin{cases} \sqrt{\frac{Y_s * A_m}{\frac{1}{2} * (1 + \gamma)}} & \forall Y_s \leq Y_{aux} \\ V_m & \forall Y_s > Y_{aux} \end{cases}$
$T_o = \frac{V_w}{A_m}$	$T_a = T_o * (1 + \gamma)$
$\tau = \gamma * T_o$	$T_m = \frac{T_a}{2}$
$T_k = 2 * \frac{Y_s - Y_a}{V_m}$	$T_s = T_a + \frac{T_k}{2}$
$V_m = \frac{2 * Y_f}{T_t + \sqrt{T_t^2 - 4 * (Y_f \frac{1 + \gamma}{A_m})}}$	

Table 5.2: Motion Equations Parameters Definition.

Symbol	Definition
V_m	Maximum allowed velocity
A_m	Maximum allowed acceleration
Y_f	Distance between current and intended joint angle
Y_s	Midway point of motion
Y_{aux}	Auxiliary variable to verify if switching acceleration point is before Y_s
Y_a	Acceleration switching point
V_w	Maximum velocity reached
T_o	First acceleration switching instance
T_m	Second acceleration switching instance
τ	Auxiliary first acceleration switching instance
T_a	Third acceleration switching instance
T_K	Duration of constant velocity
T_s	Half motion time duration
T_t	Motion time duration

the pre-defined average velocity in the Cartesian space and distance between the points P_f and P_i similarly to the method employed in *MoveL*. Moreover, the maximum acceleration value is set as a third of the Y_f . Considering the distance (Y_f), the motion duration (T_t) and the maximum acceleration previously mentioned is then possible to compute the maximum velocity value achieved during the movement. With all these figures, the motion profile is generated.

In the last joint, the used method to generate the profile differs in the definition of some parameter values. The motion, in this case, is not bound to a specific time frame, unlike the preceding method applied in the first three joints. In this approach, both maximum velocity and maximum acceleration are already pre-defined as 4000 *steps/s* and 2000 *steps/s²* respectively. After setting the basic parameters, the method to compute the remainder parameters and motion profile is the same as for the first three joints formerly mentioned.

As previously stated, the user can decide when selecting the *MoveJ* option what type of motion profile to implement, the choices available being a triangular, traditional S-Curve or canonical S-Curve motion. The parameter γ is set according to the selected type to generate the different profiles. γ defines how the acceleration behaves during the speeding up phase of the motion. With

a $\gamma = 0$, producing a phase with a constant acceleration, resulting in a trapezoidal S-Curve as seen in Figure 5.3a. By setting γ to 0.5, during the acceleration phase, the motor starts speeding up gradually, switching in the next stage to a constant acceleration until finally switching over to a gradual reduction in its value, resulting in a traditional S-Curve, as shown in Figure 5.3b. In $\gamma = 1$ the acceleration phase is divided into two stages, the first with an incremental acceleration and a second with a gradual deceleration, generating a canonical S-Curve, as displayed in 5.3c.

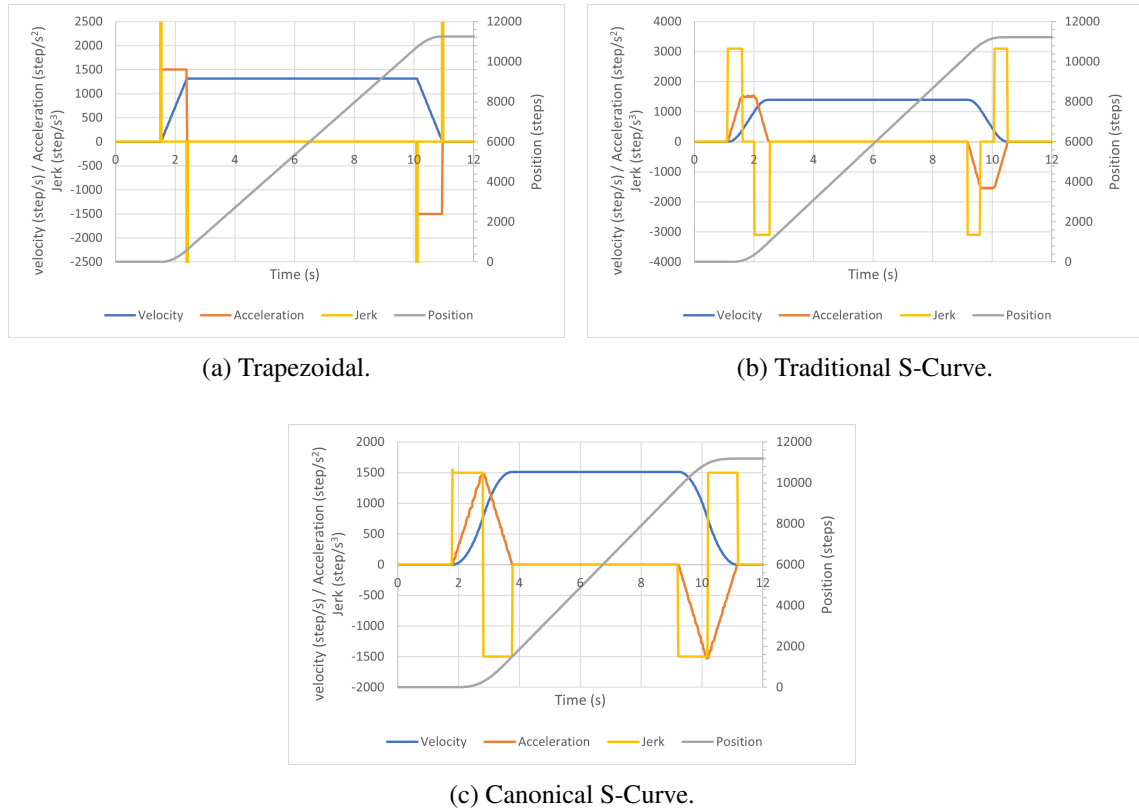


Figure 5.3: Joint 1 motor motion.

5.1.4 Measurements

The manipulator possesses a variety of sensors. Both end-line and vacuum pressure have a working principle of switches, closing when pressed or when certain vacuum pressure is reached. The detection of the state (opened/closed) of these sensors can be effortlessly achieved by reading the digital value that they generate on the pins of the microcontroller.

Meanwhile, the encoders require a more complex method for extracting the sensor readings, resulting in SSI communication between the encoder and microcontroller. To receive measurements, the microcontroller sends a clock signal. Each clock signal receives the value of a bit that constitutes the 13bit measured angle of the encoder until all bits are obtained.

Encoder Error Compensation

Magnetic encoders are sensitive devices that require a perfect alignment with the magnetic pin to produce an accurate reading. After several tests developed while analysing the oscillations, it was discovered that the last joint of the manipulator magnetic pin was miss-aligned as the readings returned were not linear with the motion and diverged significantly with the value yielded by the stepper angle near the region of 0° . Manually aligning the pin would require disassembling parts of the manipulator. Therefore, an offset was applied in the encoder readings to compensate for this phenomenon to resemble the actual value. To find this offset, the motor was slowly moved from one end of the joint limit to the other extremity while taking note of the error between the encoder and stepper angle. The points were then imported to a spreadsheet and plotted, and through the generation of a tendency line of all noted points, a sixth-order polynomial curve of the error was produced, as seen in Figure 5.4. After performing the described method in all joints associated with encoders (the first three joints), it was verified that the encoder in the first joint was correctly aligned, whilst the second and third joints could benefit from an error compensation.

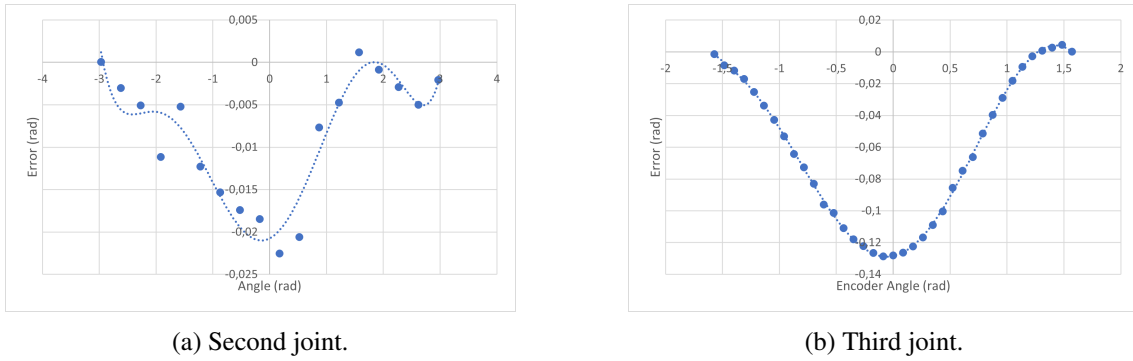


Figure 5.4: Tendency line for joint encoder offset.

The offset value to be applied in the second (ϵ_2) and third joint (ϵ_3) can be achieved by using Equations 5.5 and 5.6.

$$\begin{aligned} \epsilon_2 = & 2e^{-4}E_2^6 + 1e^{-5}E_2^5 - 2.5e^{-3}E_2^4 - 6e^{-4}E_2^3 \\ & + 11.8e^{-3}E_2^2 + 3.5e^{-3}E_2 - 20.7e^{-3} \end{aligned} \quad (5.5)$$

$$\begin{aligned} \epsilon_3 = & 4.9e^{-3}E_3^6 - 2.2e^{-3}E_3^5 - 44.7e^{-3}E_3^4 - 2e^{-4}E_3^3 \\ & + 1.324e^{-1}E_3^2 + 14.4e^{-3}E_3 - 1.284e^{-1} \end{aligned} \quad (5.6)$$

Where E_2 and E_3 are the raw values, read from the second and third joint encoders, respectively.

Implementing the compensation equations in the measurements returned by the encoders significantly improved the error, as seen in Figure 5.5, resulting in a 78.26% reduction in error in the

second joint and 68.98% in the third.

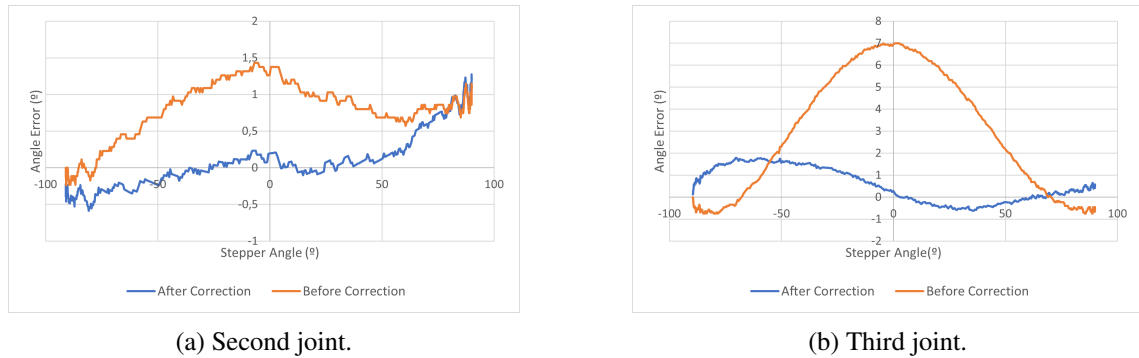


Figure 5.5: Error before and after offset compensation.

Encoder Measures Median Filter

During testing, it was visible the appearance of some outliers in the encoder values. To eliminate this phenomenon, a median filter was implemented to remove these incorrect measurements. The application of the described method was achieved by performing three consecutive measurements and sorting them using the "insert sort" algorithm described in Algorithm 1.

Algorithm 1 Insert Sort Algorithm. Source: Zeinolabedini [40].

```

for  $j \leftarrow 2$  to  $\text{length}[A]$  do
     $\text{key} \leftarrow A[j]$ 
    Insert  $A[j]$  into sorted sequence  $A[1..j-1]$ 
     $i \leftarrow j-1$ 
    while  $i > 0$  and  $A[i] > \text{key}$  do
         $A[i+1] \leftarrow A[i]$ 
         $i \leftarrow i-1$ 
    end while
     $A[i+1] \leftarrow \text{key}$ 
end for

```

After ordering the measurements in the array, the median value is extracted and used as the encoder reading. This method eliminated the occurrence of occasional wrong readings in its entirety.

5.1.5 Additional Features

The user can interact with the manipulator, as seen in Figure B.2, at any given time by pressing the PAUSE or EMERGENCY STOP button. In the case of EMERGENCY STOP activation, the manipulator instantaneously halts its movement, clearing all pending points for the subsequent motions and warning the user. When the EMERGENCY STOP button is unpressed, the system defaults to initialisation unless, when entering an emergency, the manipulator was holding cargo.

In this case, the user is warned of the event and is asked to manually switch off the manipulator and remove the cargo. Only then can the system return to initialisation.

Moreover, during motion, the user can select the pause button, which causes a gradual deceleration until stationary. Then, the user can opt to resume or cancel the movement. In the latter, the system checks if the manipulator is currently holding cargo, and if so, the user is instructed to select a destination to safely release the objects. If not, the system clears the current pick-and-place movement points and proceeds to the next ones if present.

The emergency stop option should only be used in dire situations, as the sudden change in the speed will result in difficulty in decelerating due to the moment carried by the manipulator and extremely high oscillation, increasing in magnitude with heavier carried loads. Entering the pause state is a more adequate alternative as the motor gradually decelerates, resulting in significantly fewer manipulator vibrations.

During any state where the manipulator is not performing any motion, the user can opt to enable/disable the drivers. This results in the ability to position the manipulator to the desired coordinates manually. This includes pause and emergency state.

During the gripping process, the manipulator might encounter a scenario where it cannot pick the intended object up. This event can be caused by an insufficient lowering of the gripper, irregularities on the load surface or incorrect positioning of the end-effector or load. A time-out is implemented to guarantee that the manipulator does not enter a deadlock. Where if, within a five seconds period, the vacuum pressure switch did not detect the grasping of the object, the manipulator warns the user and awaits for the event acknowledgement. After that, the system cancels the current movement and raises the gripper, returning to the waiting state.

5.1.6 Safety

The current manipulator setup only allows for the implementation of post-collision safety measures. Being only equipped with encoders, it is possible to provide a safety method capable of, upon impact, reducing further damage to both the manipulator and collided obstacle. The technique consists of actively verifying during motion (states three and four in Figure B.1) if, at any given point during those states, the difference between the angle measured by the encoder and the stepper motor angle is superior to a specified threshold, in which case the manipulator halts its movement instantly and warning the user. The user can then decide if he intends to proceed or cancel the current motion. The threshold value must be carefully chosen. It must not be too low to activate the safety mechanism during movement due to the system vibrations and not too high to continue to exert force on the collided object for an extended period, risking further damage to both the manipulator components and the obstacle. After considering the values of peak oscillation of the manipulator during motion, the value chosen was 5° , selected with a tolerance for higher vibrations that the heavier load might cause.

Further additional methods can be applied to increase safety and to comply with ISO 10218-1:2011². Through the implementation of additional certified measures that are able to perceive objects and humans that enter the manipulator workspace and alter the behaviour, such as cameras with depth information, radars, laser barriers and similar systems.

²ISO, ISO 10218-1:2011 Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots [online]. [viewed 06 July 2022]. Available from: <https://www.iso.org/standard/51330.html>

Chapter 6

Results

This chapter displays the tests performed to verify the correct functioning of the developed system in Section 6.1, as well as an analysis of the oscillations resulting from those movements in Section 6.2.

6.1 Tests

This section displays the results of the various tests performed to demonstrate the different functionalities of the manipulator, starting with a standard operation and proceeding to different scenarios the user might encounter, such as not reachable coordinates, problems met during the motion and in the gripping of the cargo. Lastly, the user interactions with the system during the movement are tested, specifically by pausing and pressing the emergency button.

6.1.1 Regular Operation

The manipulator control software was developed to perform a variety of pick-and-place tasks. Therefore, for the robotic arm to be able to execute the assigned duties, the user must specify both the picking and placing coordinates and the type of motion required. The choice of motion falls between *MoveJ* and *MoveL*.

To compare both movements, a sequence of tasks was pre-established. The course defined consists of three different pick-and-place motions as seen in Figure 6.1. In the first one, the manipulator starts with $X \approx -1.1$ m and $Y \approx 1.3$ m coordinates, with a $\Theta \approx 90^\circ$. Moving then to $X = 1$ m and $Y = 1$ m and $\Theta = 90^\circ$ where it lowers the gripper to $Z = -0.56$ m to grip a 500 g piece sheet of aluminium. After raising the gripper to its maximum height, the manipulator moves to $X = 0$ m and $Y = 0.7$ m with a $\Theta = 0^\circ$ lowering the end-effector to a $Z = -0.41$ m height to drop it off in the pallet truck's fork. Afterwards, the manipulator repeats the pick-and-place motion to move a 6 kg load to a different position and with a 90° rotation, following another motion to retrieve the first transferred cargo from the pallet truck to the original place of the second load with another 90° rotation.

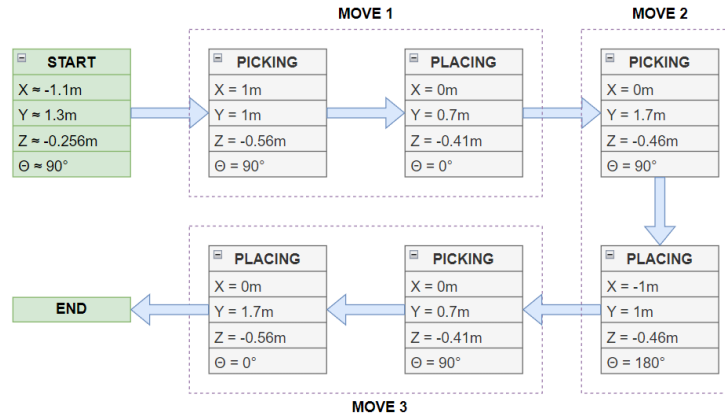
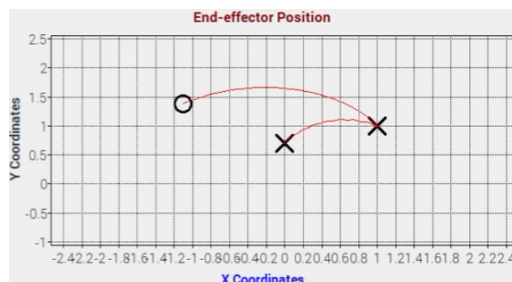


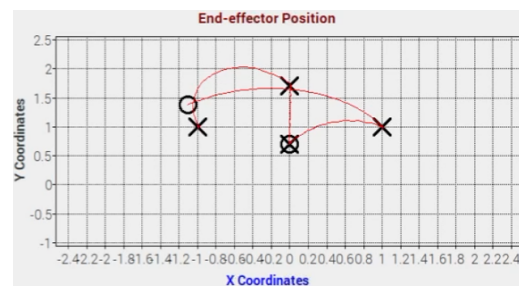
Figure 6.1: Movement sequence performed.

MoveJ

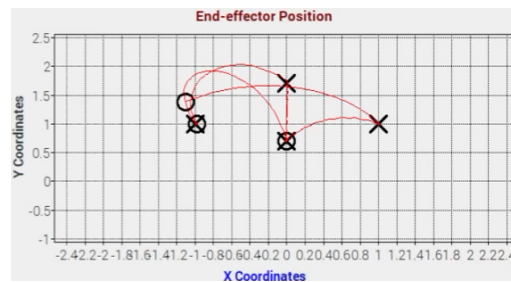
In *MoveJ*, the user can opt between different types of motion profiles (trapezoidal, traditional S-Curve and canonical S-Curve), in addition to the manipulator's elbow orientation. Therefore, to test all distinct options during the movement seen in Figure 6.1 *Move 1* was executed with a trapezoidal motion profile and an automatic choice for the second joint orientation, *Move 2* with a traditional S-Curve and manual selection in relation to the second joint, "elbow" to the left in the picking and right in the placing. The *Move 3* was accomplished with a canonical S-Curve and automatic orientation of the second joint.



(a) First Movement.



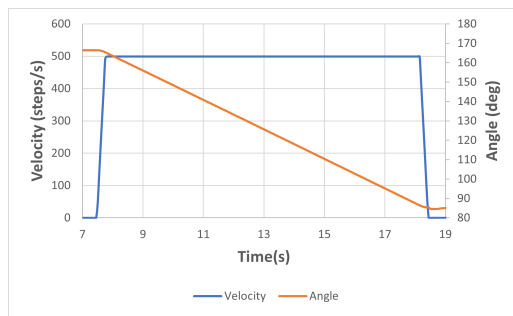
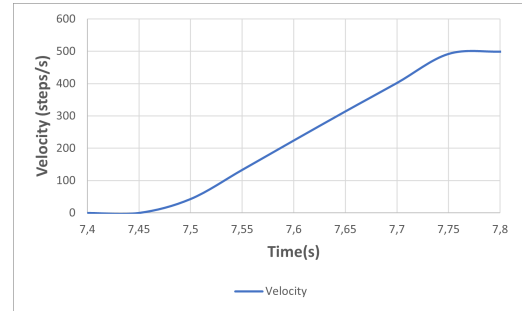
(b) Second Movement.



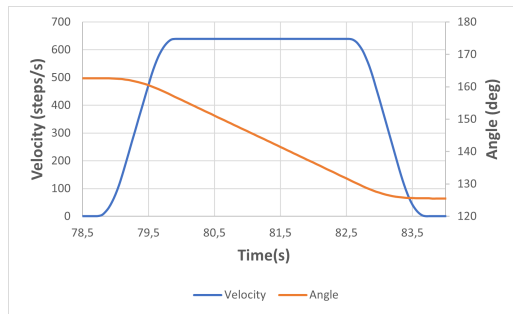
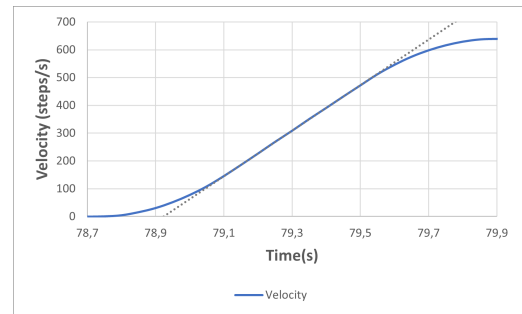
(c) Third Movement.

Figure 6.2: Trajectory navigated during the sequence of *MoveJ*.

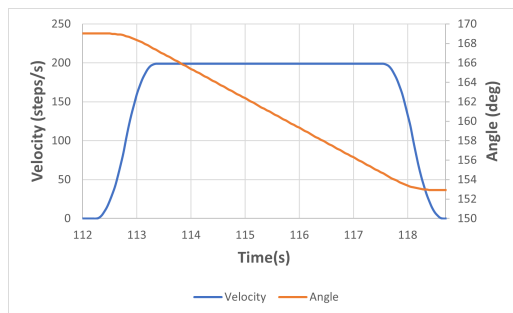
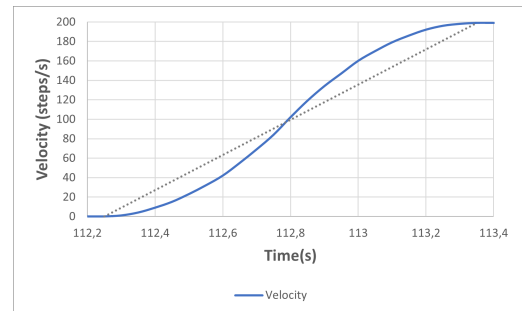
In Figure 6.2, the trajectory of the end-effector can be tracked throughout the different movements that composed the sequence defined for testing purposes (red line), where the circle (O) represents the end-effector position at the beginning of a new set movement and the cross (X), the coordinates of the pick-and-place positions. As expected, the route generated in the linear joint space does not translate to a linear course in the Cartesian space observed, with the link between points displaying an arch, except for the vertical movement performed from $Y = 0.7$ m to $Y = 1.7$ m, which exhibits a straight line.

(a) Trapezoidal motion profile (*Move 1*).

(b) Acceleration phase of the trapezoidal motion profile.

(c) Traditional S-Curve motion profile (*Move 2*).

(d) Acceleration phase of the traditional S-Curve motion profile.

(e) Canonical S-Curve motion profile (*Move 3*).

(f) Acceleration phase of the traditional S-Curve motion profile.

Figure 6.3: Velocity and angle variation of the first motor on the different picking motions of the movement sequence.

The sequence of movements previously described generates three different motion profiles for

each different movement. Therefore, to demonstrate the ability of the manipulator to generate these different profiles, the values of velocity and angle variation for the first motor is presented in Figure 6.3. In the presented charts, it's possible to identify on the left the motion profile achieved during the pick stage of the movement, as well as the linear motion of the angle represented by the orange line and on the right the velocity curve amplified in the acceleration zone. Moreover, in Figure 6.3d and 6.3f an additional dotted line is displayed to aid the understanding of the curve shape. It is visible that the three different motions were clearly performed, achieving the expected velocity shape in the acceleration stage for the motion profile.

Upon arrival at the coordinates in the X and Y axis selected by the user, the manipulator must lower the end-effector to grip the cargo and raise it. Independently of the type of motion chosen to move the manipulator to the point, the movement of the prismatic joint motor is generated using the *MoveJ* algorithm. Unlike the rotational joints, the motion is not a time constraint.

In Figure 6.4, the velocity of the prismatic motor for *Move 1* is displayed. It's possible to verify that the motor reaches high-velocity values as the relation of step to joint displacement is significantly low (1 step equals μm). As the prismatic joint is not highly affected by the jerk, the motor can be accelerated at a higher rate than the previous joints, as seen in Figure 6.4.

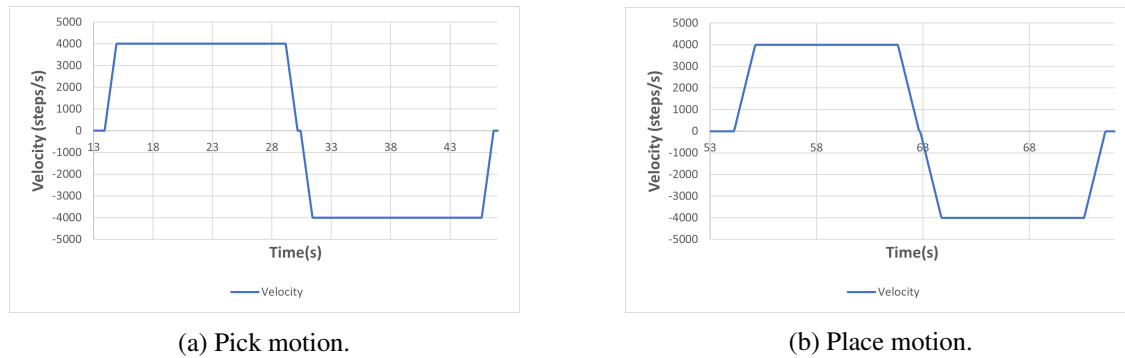
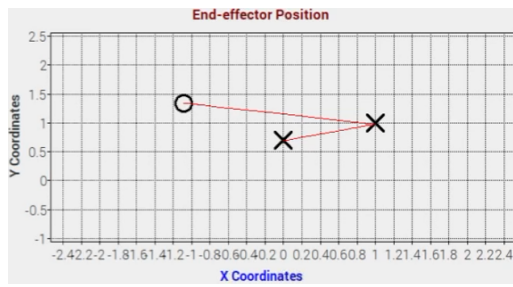


Figure 6.4: Velocity of the fourth joint in *Move 1*.

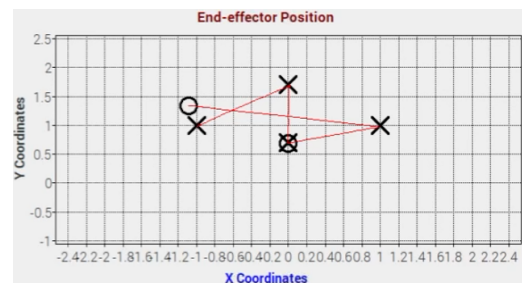
MoveL

Unlike the previous method, the *MoveL* movement does not require additional parameters. Therefore the sequence of movements implemented for testing can be achieved by entering the selected coordinates for the pick-an-place points into the user interface.

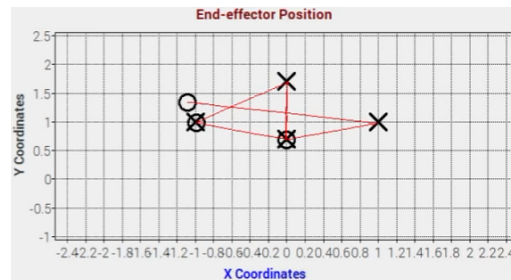
In Figure 6.5, the trajectories travelled by the end-effector are displayed. It is possible to observe that all movements performed by the manipulator resulted in a linear path in the Cartesian plane between points as intended. Furthermore, Figure 6.6 exhibits the charts of both rotational velocity, in blue, and angle, in orange, of the first motor in the different picking phases of the movement sequence. It's possible to verify that the velocity profile is significantly different from the ones found in *MoveJ* as a result of the necessity to generate a linear trajectory.



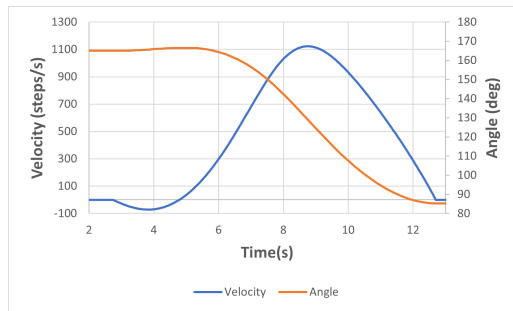
(a) First Movement.



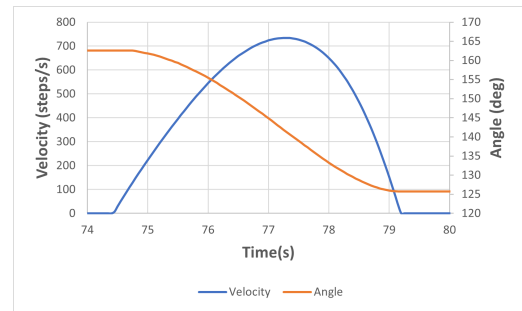
(b) Second Movement.



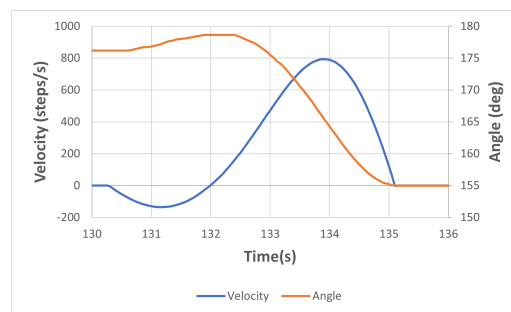
(c) Third Movement.

Figure 6.5: Trajectory navigated during the sequence of *MoveL*.

(a) First Movement.



(b) Second Movement.



(c) Third Movement.

Figure 6.6: Velocity and angle variation of the first motor on the different picking motions of the movement sequence.

Accuracy

Despite reaching the same coordinates, the different methods used to generate the diverse trajectory types result in various accuracies.

Table 6.1: End-effector position at ending of movement.

	Movement	Type	X (m)	Y (m)	Θ (°)	Duration (s)
MoveJ	Move 1 (TRAP)	PICK	1.001	0.995	92.13	11
		PLACE	-0.008	0.700	-1.43	5.25
	Move 2 (S-Curve)	PICK	0.007	1.697	89.55	5
		PLACE	-0.988	1.003	180.14	6.15
	Move 3 (CAN S-Curve)	PICK	-0.004	0.697	91.16	5
		PLACE	0.014	1.696	-0.17	4.8
MoveL	Move 1	PICK	0.993	0.972	91.85	10
		PLACE	-0.003	0.689	-1.43	4.95
	Move 2	PICK	0.01	1.69	89.38	4.8
		PLACE	-0.987	0.985	181.86	5.85
	Move 3	PICK	-0.016	0.702	88.52	4.9
		PLACE	0.011	1.687	-0.23	4.7

The values in Table 6.1 represent the values of the end-effector position at the end of each movement. It is possible to observe that both move types result in an accurate positioning with a minor deviation to the true planned value in most cases. Nevertheless, the average deviation of each position value was computed for both *MoveJ* and *MoveL*, resulting in $X = 7.67$ mm, $Y = 3.50$ mm and $\Theta = 0.913^\circ$ for *MoveJ* and $X = 10.0$ mm, $Y = 13.2$ mm and $\Theta = 1.25^\circ$ for *MoveL*. The examination of these values resulted in the conclusion that *MoveJ* provides slightly better precision in the positioning of the manipulator.

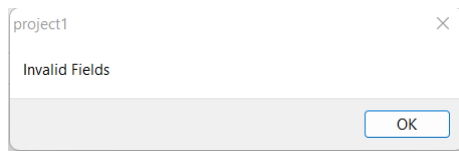
In regards to the duration of motion, theoretically, the same movement should be carried out in the same time frame, independently of the method used to generate it. However, that was not the case. *MoveJ* carried out the movement with a slightly longer duration. The noticeable difference in time in the first movement is due to the point where the system is initialised, which, having been manually positioned, was not the same in both *MoveJ* and *MoveL* and, therefore, should not be regarded.

6.1.2 Impossible Movements

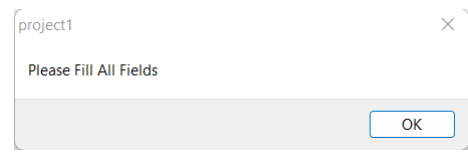
Due to the physical limitations of the manipulator, not all points entered by the manipulator will result in a solution for the manipulator. Therefore, an analysis of the user's inputs must be conducted before the actual movement.

Fields Verification

Foremost, the values introduced by the user are checked if they are actual numbers or if any field was left empty. If so, the system provides a warning as presented in Figure 6.7a and 6.7b.



(a) Invalid fields warning.



(b) Fill fields warning

Figure 6.7: Fields verification warnings.

Out of Bounds

The following verification consists of checking if the values introduced by the user reside within the manipulator's workspace. For example, introducing the values: $X = 1$ m, $Y = 2$ m and $\Theta = 90^\circ$ which does not belong to the workspace returns the following warning presented in Figure 6.8.

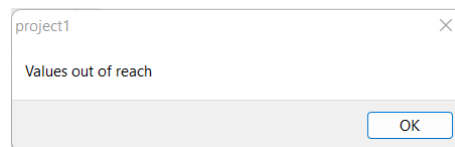
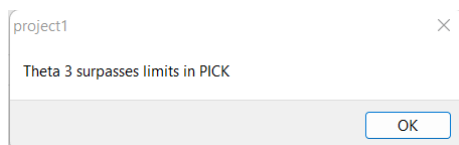


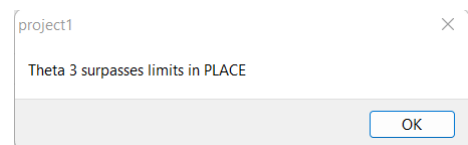
Figure 6.8: Out of bounds warning.

MoveJ

Certain combinations of values received might result in a solution outside the joint's physical limits, especially if the user chose a specific orientation for the manipulator "elbow". In case of an introduction of a point not reachable by the robotic arm with a specific second joint orientation, the user is warned of which joint surpasses the limit and if it was the pick or place point. For instance, choosing a left orientation and $X = 1$ m, $Y = 1$ m and $\Theta = 180^\circ$ for picking, the system returns the warning seen in Figure 6.9a and 6.9b for a right orientation of the second joint and $X = 0$ m, $Y = 1$ m and $\Theta = 0^\circ$.



(a) Picking warning.



(b) Placing warning.

Figure 6.9: *MoveJ* with manual orientation warnings.

For an automatic orientation of the manipulator "elbow", as the system tries to reach the best solution by taking advantage of the second joint ambiguity, the number of impossible solutions is significantly smaller but still present. An example is $X = 0$ m, $Y = 1.8$ m and $\Theta = 180^\circ$, resulting in the warning in Figure 6.10.

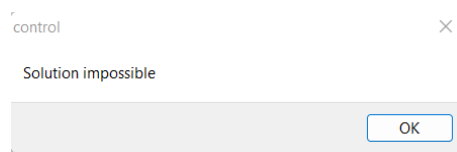


Figure 6.10: *MoveJ* with automatic orientation warning.

MoveL

In the *MoveL* algorithm, the inverse of the jacobian is performed to convert the Cartesian velocities into joint velocities. Therefore, singularities must be avoided. Moreover, during the motion, a joint might exceed its limit. These verifications are performed prior to the movement by testing if the linear movement requested passes near a singularity point or through a position where a joint is out of its physical limit. If so, the system gives the warning in Figure 6.11 with an indication of what movement is impossible.

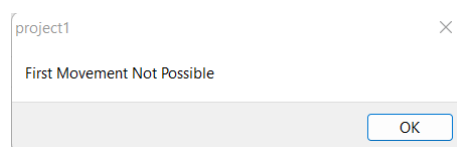


Figure 6.11: *MoveL* warning.

6.1.3 Motion Problem

During the movement to a specific point, the manipulator might encounter a situation where it is unable to move, which might be caused by a collision with an entity or a lack of torque to shift the joints to the desired position. The detection of any of these events is accomplished by regularly comparing the encoders' angle to the respective joint angle. Suppose the difference between them is superior to the threshold of 5° . In that case, the manipulator immediately halts its movement and warns the user, who can decide between resuming the motion or cancelling it after providing acknowledgement.



Figure 6.12: Collision obstacle.

To test this safety measure, the manipulator was started in $X = -1.14$ m, $Y = 1.38$ m and $\Theta = 180^\circ$ and sent to $X = 1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$ using the *MoveL* method. At $X = 0.15$ m and $Y = 1.5$ m an obstacle was placed to interrupt the movement as seen in Figure 6.12. Afterwards, the obstacle was removed and then selected the resume option to finish the motion, completing the path shown in Figure 6.13.

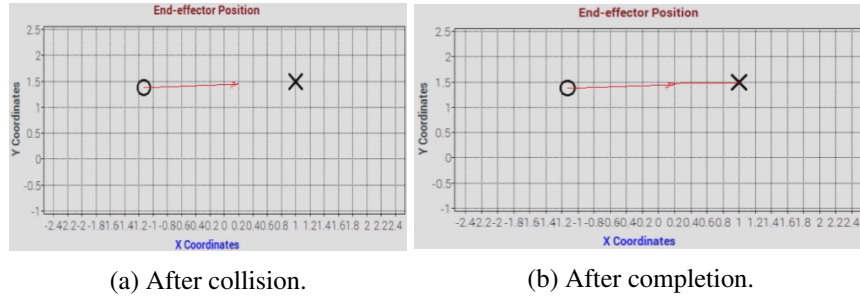


Figure 6.13: End-effector path.

During the movement, the first joint was the first to detect the crash after collision with the obstacle. As seen in Figure 6.15, when the "error", the difference between encoder and stepper motor angle, exceeds the 5° threshold, all motor velocities are instantly set to zero, halting the movement as seen in Figure 6.14. The red line represents the moment the collision was detected. The crash was approximately detected within 0.5 seconds after the first motor encountered significant resistance at the time moment of 16.7 seconds.

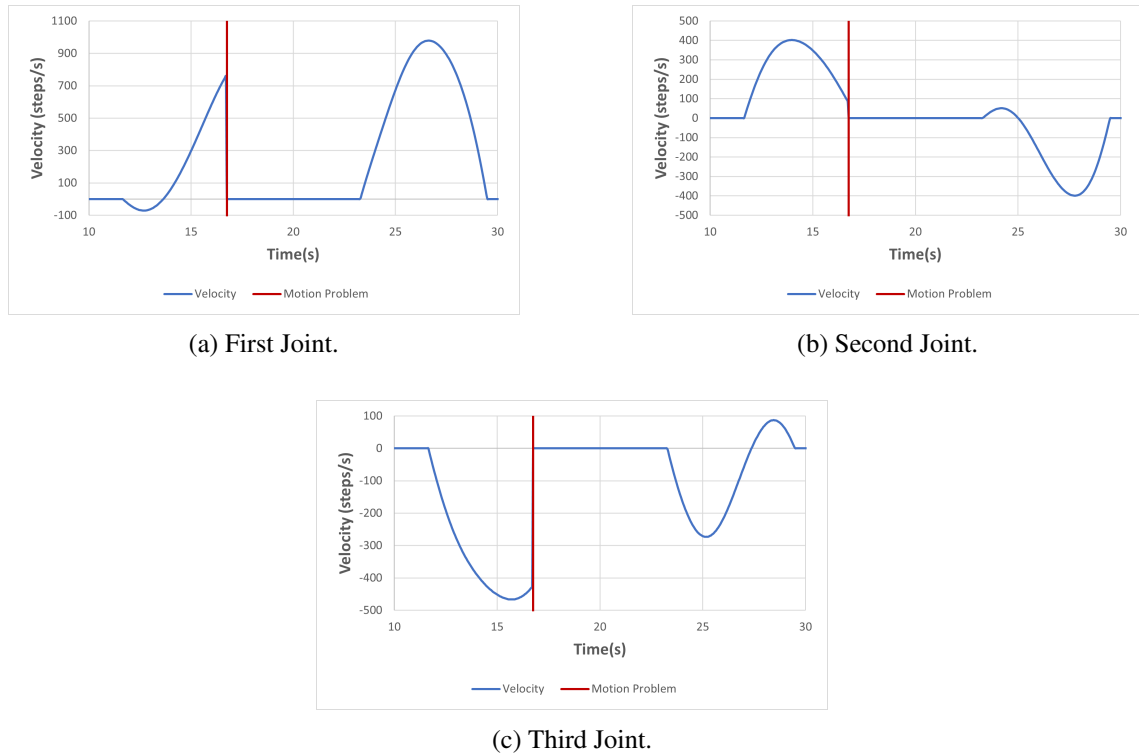


Figure 6.14: Joint velocity throughout the motion problem sequence.

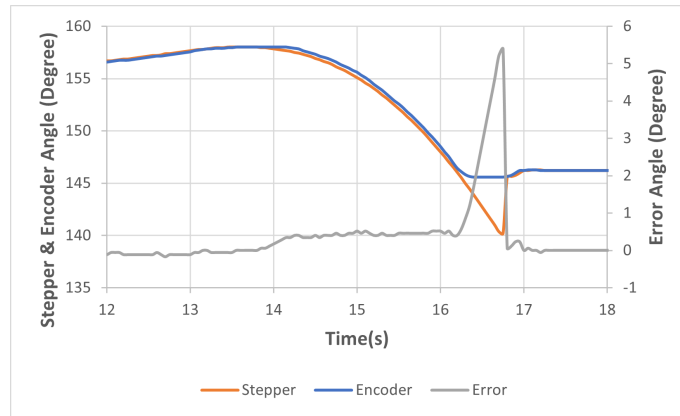


Figure 6.15: Motion problem detection.

6.1.4 Picking Action Timeout

The timeout sequence is executed in the gripping stage of the movement if, during a period of 5 seconds, the manipulator is unable to grasp the load. In that case, the system enters in a picking timeout state and warns the user of the event.

For testing purposes, the manipulator was tasked with picking up a non-existent object. Upon lowering of the end-effector and activation of the vacuum valve, it was confirmed that the manipulator wasn't able to grasp the load, entering a timeout state after 5 seconds, as seen in Figure 6.16b, whilst warning the user, Figure 6.16a.

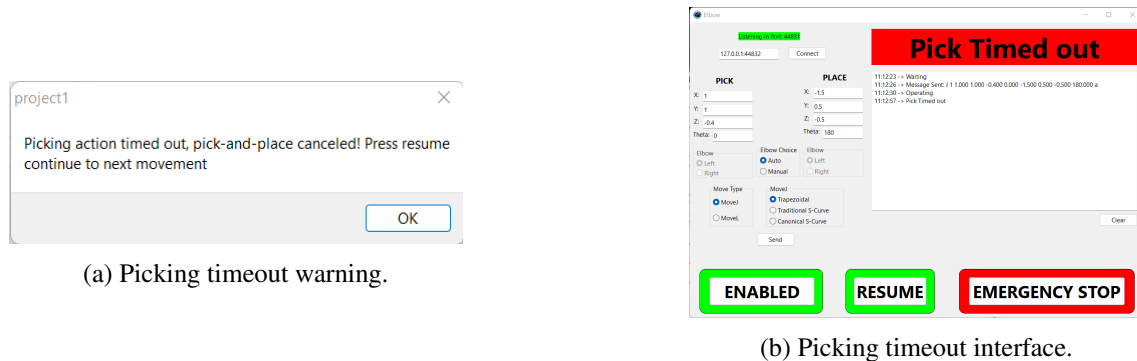


Figure 6.16: Warnings for picking timeout.

6.1.5 Pause

The pause button can be selected at any stage as long as the manipulator is operating. Upon entering this state, the manipulator is gradually decelerated until it stops. Reaching this state, the user can decide to resume the pick-and-place movement or to cancel it, as seen in Figure 6.17.

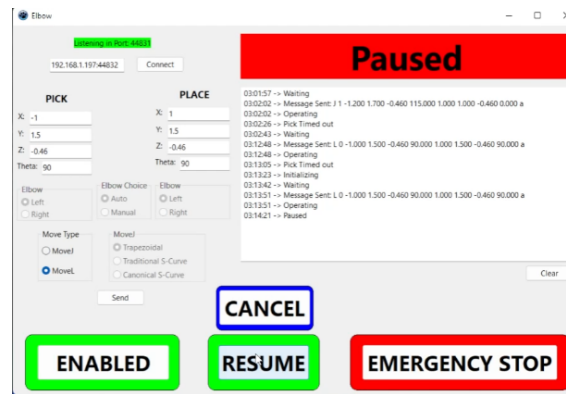
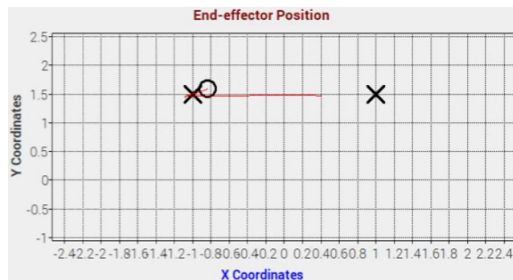


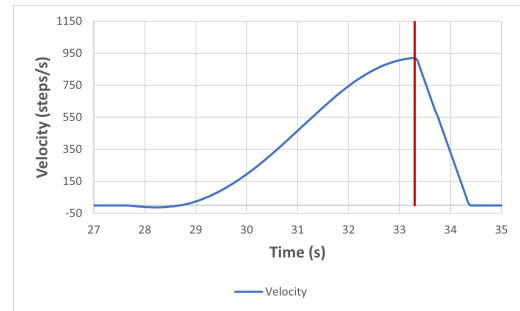
Figure 6.17: Pause state in GUI.

Pause and Resume

To test the resume feature, the manipulator was tasked with picking an object at $X = -1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$ and delivering it to $X = 1$ m, $Y = 1.5$ m, and $\Theta = 90^\circ$ being paused during the movement between the pick and place coordinates.

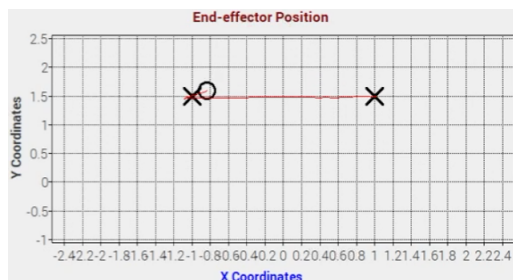


(a) Trajectory

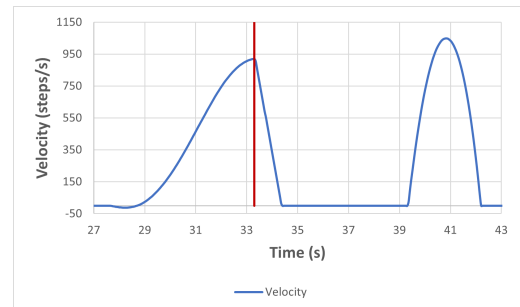


(b) Velocity of the first motor.

Figure 6.18: Charts the moment after the manipulator is paused.



(a) Trajectory.



(b) Velocity of the first motor.

Figure 6.19: Charts the moment after the manipulator is resume and reaches the destination.

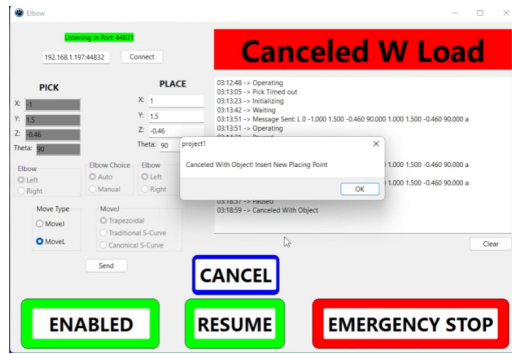
The movement from the picking point until the manipulator is paused can be seen in Figure 6.18a. After pressing the pause button, the manipulator slows down and stops at $X = 0.4$ m and

$Y = 0.46$ m. In Figure 6.18b, the velocity can be tracked through the movement until after a few moments of the system entering pause. The moment the pause button is pressed can be seen by the red line at 33.3 seconds, resulting in the gradual decrease in velocity observed subsequently.

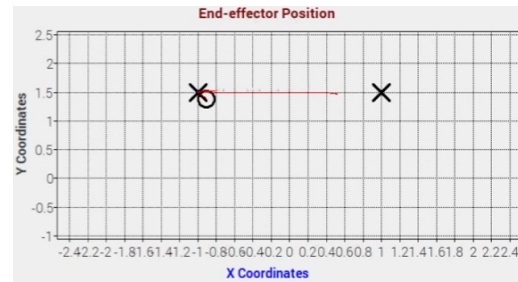
When the resume button is pressed the system generates a new trajectory for the completion of the movement, which was reached successfully, as seen in Figure 6.19a. The velocity profile for the remainder of the movement can be observed in Figure 6.19b.

Pause and Cancel

Upon pausing, the user might opt to cancel the current movement. At that moment, two different scenarios might occur. If the manipulator is currently gripping a cargo, the user is warned that he must define a new position to place the load. If not, the manipulator discards the pick-and-place points and moves to the next position if existent on the list.

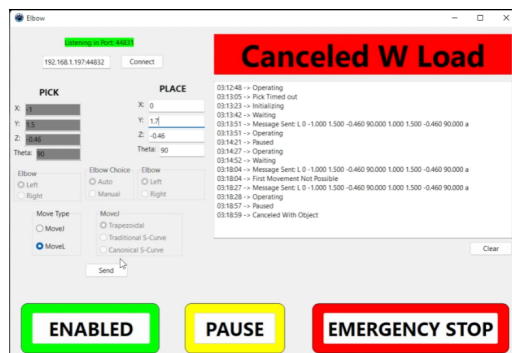


(a) Warning in GUI.

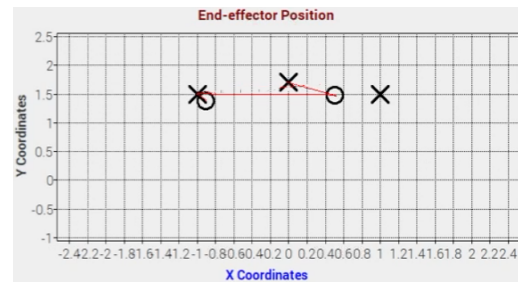


(b) Trajectory.

Figure 6.20: Warning and trajectory after pausing while holding cargo.



(a) New values insertion.



(b) Final Trajectory.

Figure 6.21: Warning and trajectory after new placing coordinates.

To test this feature, the manipulator was tasked with the same pick-and-place action previously tested (pick the cargo $X = -1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$ and deliver it to $X = 1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$) being stopped along the way while carrying an object, resulting in the warning presented

in Figure 6.20a. The pausing action caused the manipulator to halt its movements at $X = 0.5$ m and $Y = 1.48$ m in the trajectory in Figure 6.20b

After stopping, the user must input the new coordinates to place the cargo currently being held by the manipulator, as seen in Figure 6.21a. During the test, the new coordinates selected for the new placing point were $X = 0$ m, $Y = 1.7$ m and $\Theta = 90^\circ$, which resulted in the trajectory generated visible in Figure 6.21b, where it is visible the deviation in trajectory resulted from the cancellation and selection of a new drop-off place.

6.1.6 Emergency Stop

Although entering the emergency state is not recommended due to the strains exerted on the manipulator caused by the sudden change in velocity, in some cases, it might turn out to be necessary. Similar to the pause stage, two different scenarios might occur upon entering this state. If the manipulator is holding a cargo, the user is warned and requested to retrieve it in order to exit the emergency state manually. If not, the user might exit this state whenever he pleases.

To test the emergency stop, the same sequence of movements tested in the pause scenario was implemented (pick the cargo $X = -1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$ and deliver it to $X = 1$ m, $Y = 1.5$ m and $\Theta = 90^\circ$) while holding cargo. During the motion, the emergency stop button was pressed, resulting in the warning present in Figure 6.22a and the appearance of the button used to manually switch off the vacuum valve that allows for load release 6.22b. After the cargo was removed and the emergency state exited, the system defaulted back to initialising.

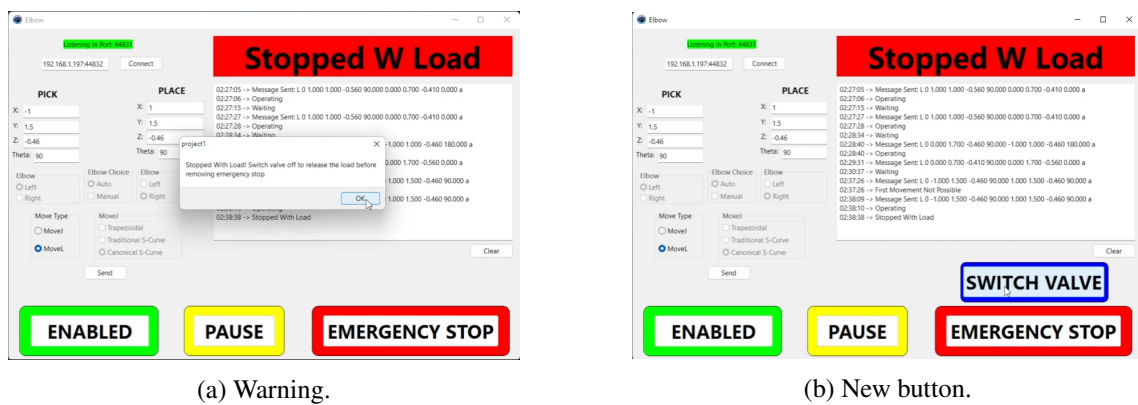


Figure 6.22: Interface after entering the emergency state.

The trajectory of the end-effector after halting its movement due to entering the emergency state can be seen in Figure 6.23a, where the manipulator's stopped location at coordinates $X = 0.16$ m and $Y = 1.47$ m is visible. Furthermore, Figure 6.23b displays the velocity of the first motor during the movement where it was stopped, with a red line representing the instance the emergency state was entered. It is clear that the velocity was successfully in immediately setting it to zero.

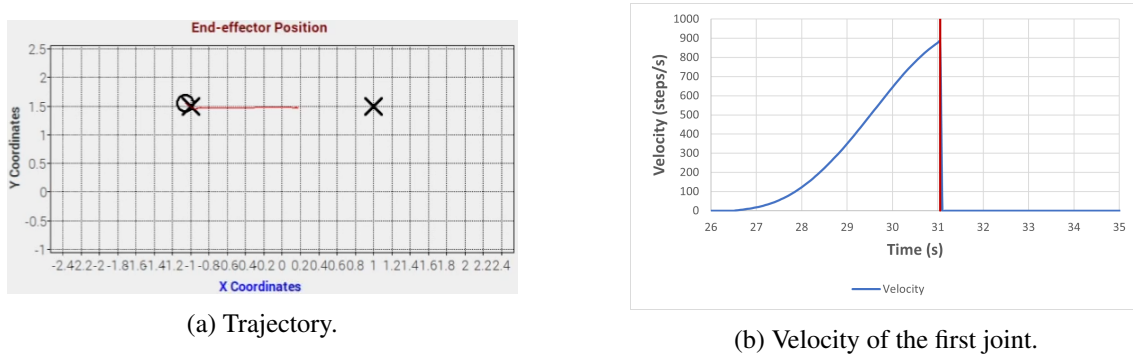


Figure 6.23: Trajectory and velocity of the manipulator after an emergency stop.

6.2 Oscillation Analysis

The current manipulator configuration causes the manipulator to be very susceptible to system vibrations caused by the jerk and variations in acceleration. This section analyses the impact of the different methods used to generate the motion profile and other events on the system's vibration.

The results were recorded using the same sequence of movements, picking an object at $X = -1$ m, $Y = 1.5$ m and $\Theta = 180^\circ$ and placing it at $X = 1$ m, $Y = 1.5$ m and $\Theta = 0^\circ$. It's during the movement between picking and placing that the oscillation analyses are performed. The tests were also executed with two different transported cargo, a 0.5 kg and a 6 kg load.

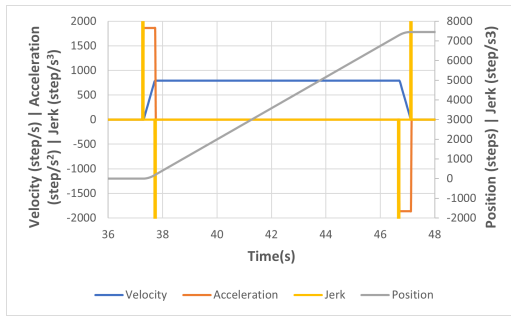
The raspberry application was executed in a computer running windows to allow for smaller serial communication intervals not possible with the computational power of the current raspberry implemented. Due to the superior hardware capability of the personal computer, the Arduino data was able to be received with 25 milliseconds of intermission, half of the interval achieved with the raspberry, leading to a more accurate representation of the state of the machine.

6.2.1 MoveJ

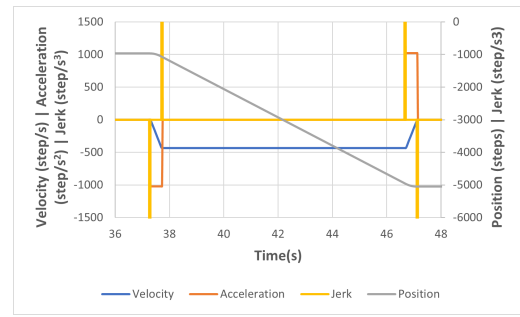
The different options of motion profile for *MoveJ* result in different accelerations profiles for each type, trapezoidal, traditional S-curve and canonical S-curve. Therefore, the oscillations produced by each motion profile cause various effects on the oscillation of each joint.

Trapezoidal

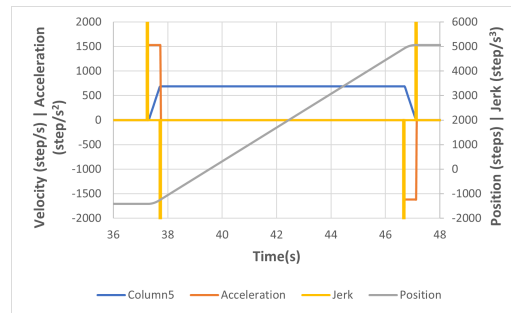
The trapezoidal motion profile generates three different phases, a constant acceleration in a first and third stage and a constant velocity in the second. This curve results in the motion profiles seen in Figure 6.24, where it is visible the velocity, acceleration and jerk of the motion, in addition to the variation in steps resulting from the movement. The sudden variation in acceleration produced by this type of motion profile generates an infinite jerk in the instances where the acceleration is altered, represented by the yellow line.



(a) First Motor.

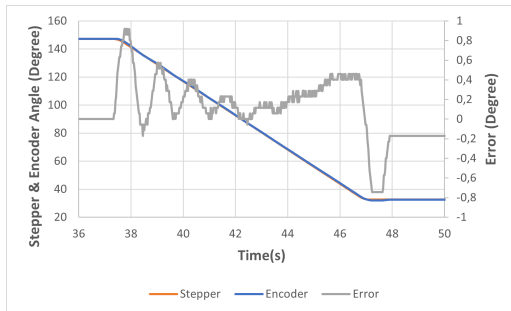


(b) Second Motor.

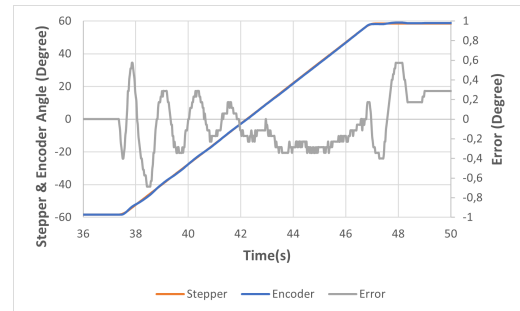


(c) Third Motor.

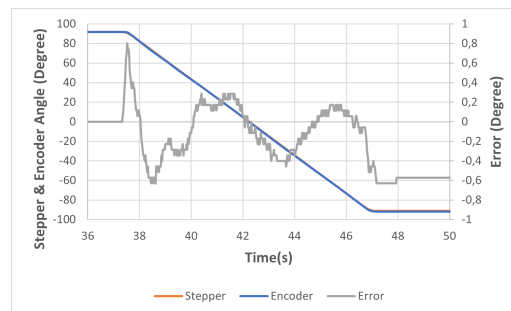
Figure 6.24: Velocity, acceleration, jerk and position profile of trapezoidal move type.



(a) First Motor.

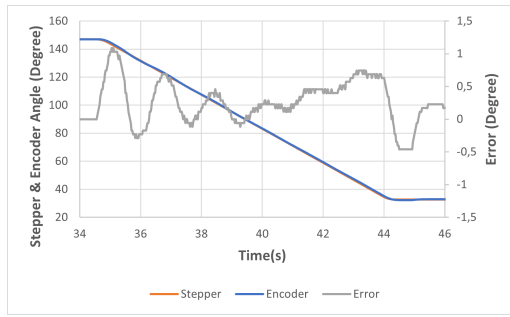


(b) Second Motor.

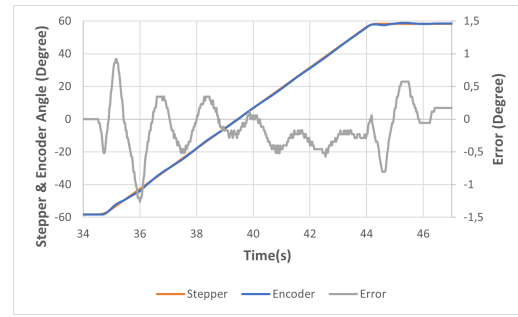


(c) Third Motor.

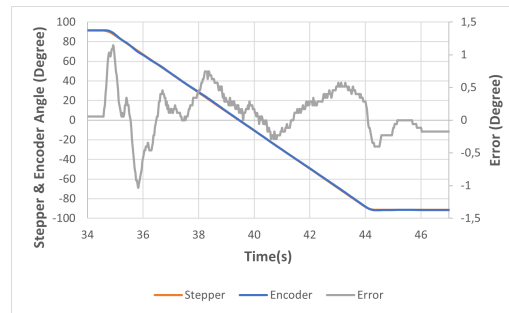
Figure 6.25: Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a trapezoidal motion profile.



(a) First Motor.



(b) Second Motor.



(c) Third Motor.

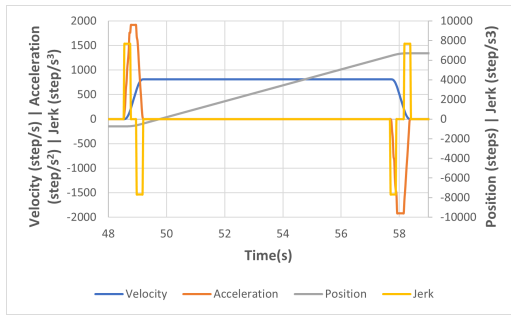
Figure 6.26: Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a trapezoidal motion profile.

The effect of the visible motion profiles on the variation of the encoder stepper and subsequently the error while carrying the 0.5 kg load can be observed in Figure 6.25 and for the 6 kg in Figure 6.26.

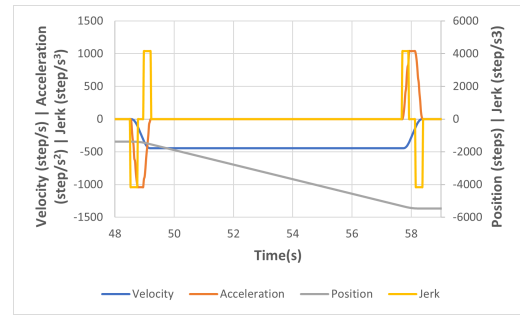
Traditional S-Curve

The traditional S-curve motion profile generates seven different phases. In the first stage, the acceleration is gradually accelerated and then maintained at a constant value, after which it is progressively decelerated until reaching the desired final velocity. The reduction in speed is mirrored in the acceleration phase. This results in the motion profiles seen in Figure 6.27, where it is visible the velocity, acceleration and jerk of the motion, in addition to the variation in steps resulting from the movement. This motion profile causes a limited jerk as an aftermath of the gradual changes in acceleration, represented by the yellow line.

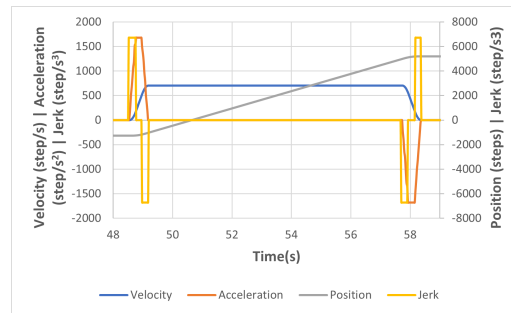
The effect of the visible motion profiles on the variation of the encoder angle stepper and subsequently the error while carrying the 0.5 kg load can be observed in Figure 6.28 and for the 6 kg in Figure 6.29.



(a) First Motor.

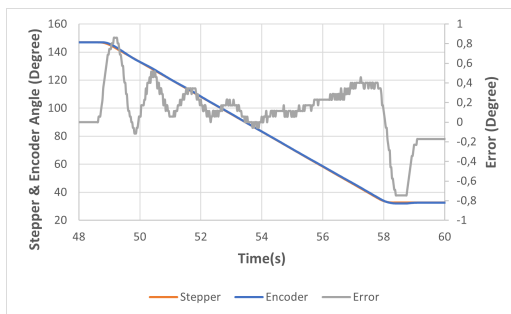


(b) Second Motor.

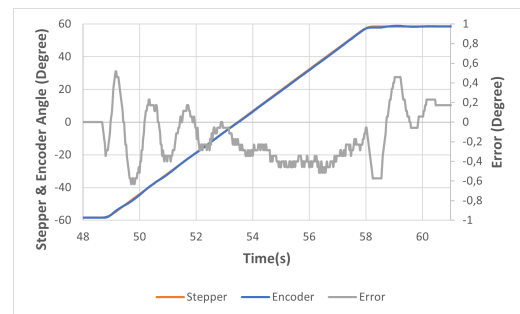


(c) Third Motor.

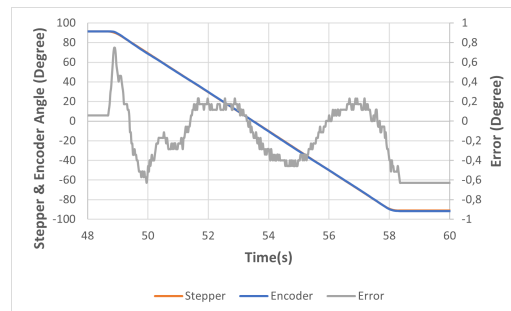
Figure 6.27: Velocity, acceleration, jerk and position profile of traditional S-curve move type.



(a) First Motor.

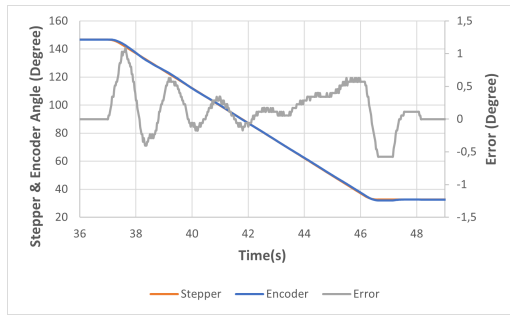


(b) Second Motor.

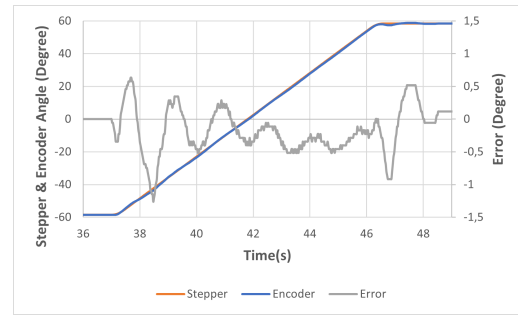


(c) Third Motor.

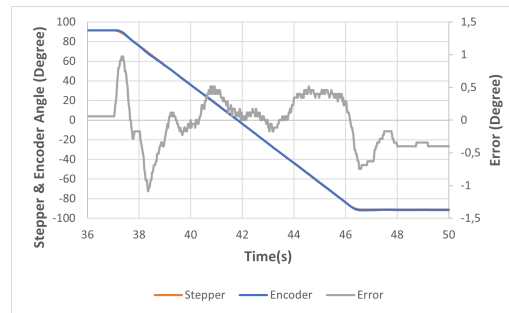
Figure 6.28: Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a traditional S-curve motion profile.



(a) First Motor.



(b) Second Motor.



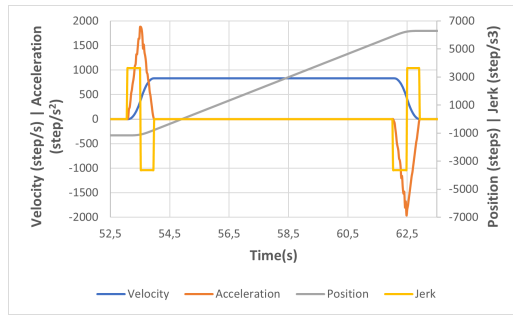
(c) Third Motor.

Figure 6.29: Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a traditional S-curve motion profile.

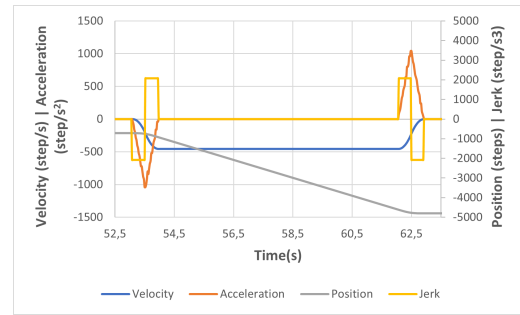
Canonical S-Curve

The traditional S-curve motion profile generates five different phases. In the first stage, the acceleration is gradually accelerated, switching to gradual deceleration until reaching the desired final velocity. This curve results in the motion profiles in Figure 6.30. Where is visible the velocity, acceleration and jerk of the motion, in addition to the variation in steps resulted from the movement. This motion type also returns a limited jerk, half of the traditional S-curve, due to using the same acceleration but having the gradual acceleration phase with double duration.

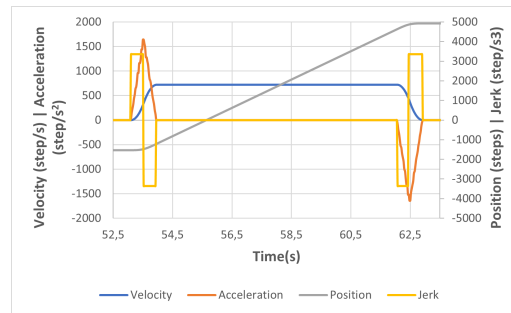
The effect of the visible motion profiles on the variation of the encoder angle stepper and subsequently the error while carrying the 0.5 kg load can be observed in Figure 6.31 and for the 6 kg in Figure 6.32.



(a) First Motor.

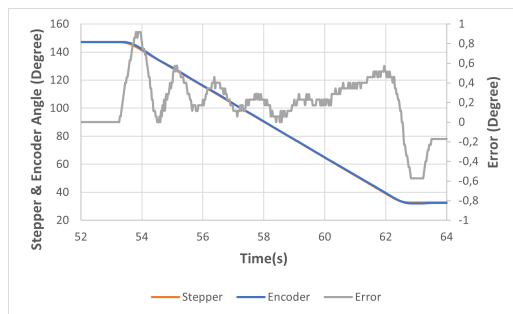


(b) Second Motor.

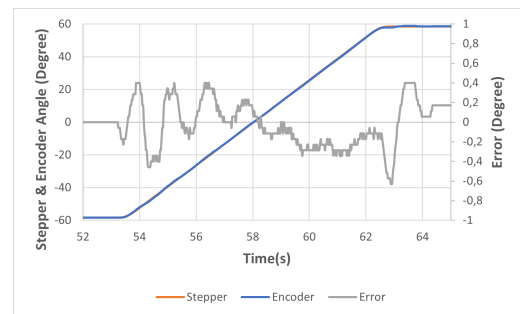


(c) Third Motor.

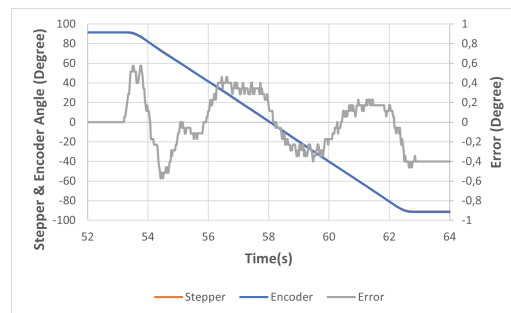
Figure 6.30: Velocity, acceleration, jerk and position profile of canonical S-curve move type.



(a) First Motor.

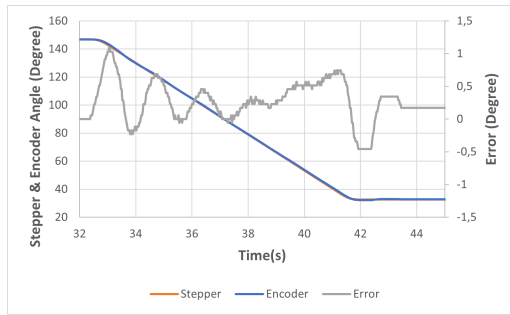


(b) Second Motor.

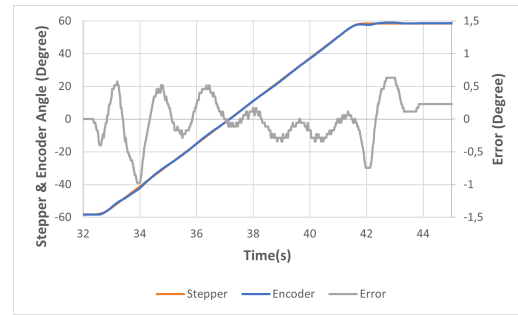


(c) Third Motor.

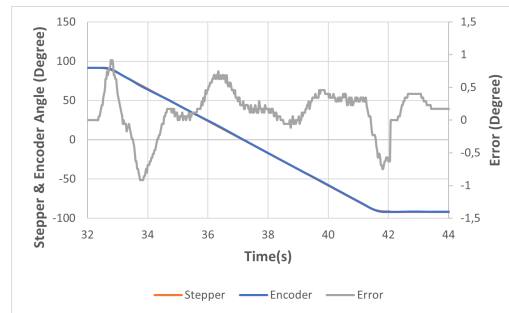
Figure 6.31: Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a canonical S-curve motion profile.



(a) First Motor.



(b) Second Motor.



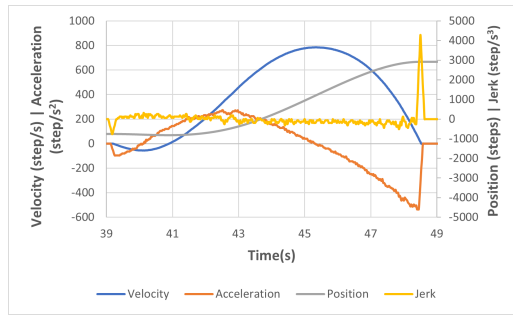
(c) Third Motor.

Figure 6.32: Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a canonical S-curve motion profile.

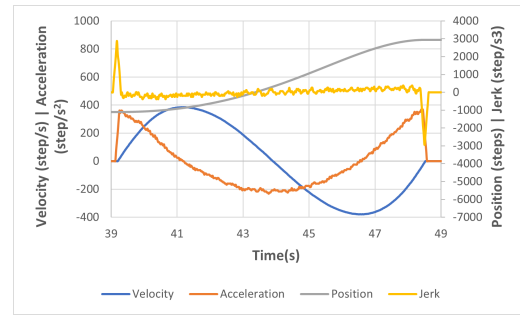
6.2.2 MoveL

Contrarily to the *MoveJ* where the profile is defined as having a particular shape, enabling the prediction of the motion. The *MoveL* generates the movement in order to follow a line connecting the origin and destination point. This property causes the velocity not to have a characteristic shape as the previously mentioned method. Figure 6.33 shows the motion profiles generated. The constant variation of the velocity during motion causes the acceleration and subsequently the jerk to possess an erratic behaviour. Nevertheless, the jerk produced, despite being present in the entirety of the movement, is far lesser than the one resulting from the *MoveJ* algorithm.

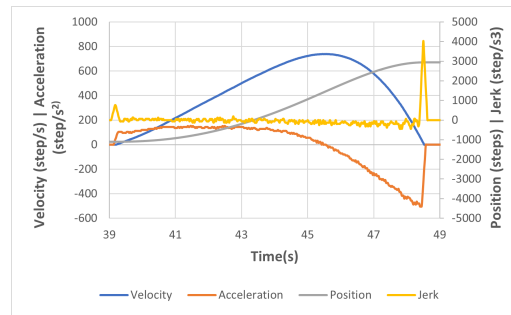
The effect of the visible motion profiles on the variation of the encoder angle stepper and subsequently the error while carrying the 0.5 kg load can be observed in Figure 6.34 and for the 6 kg in Figure 6.35.



(a) First Motor.

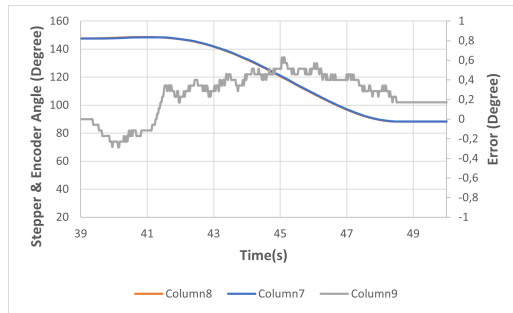


(b) Second Motor.

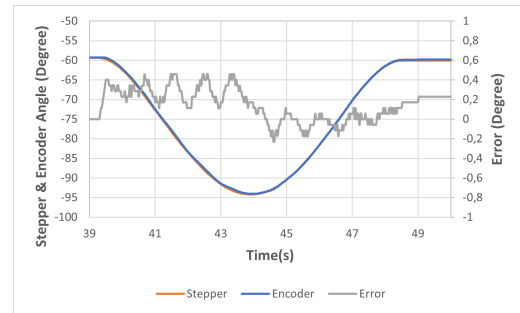


(c) Third Motor.

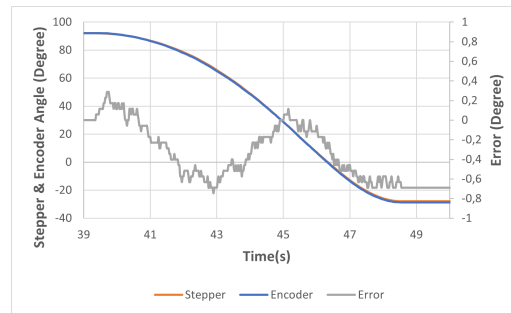
Figure 6.33: Velocity, acceleration, jerk and position profile of MoveL move type.



(a) First Motor.

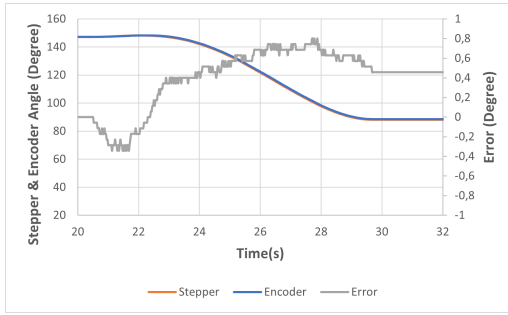


(b) Second Motor.

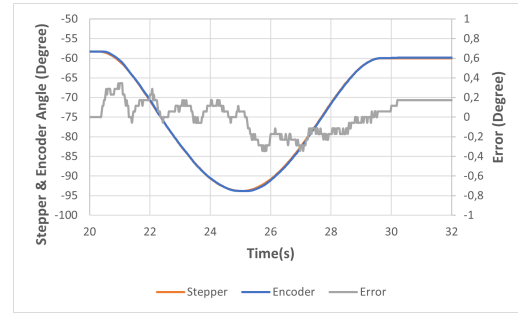


(c) Third Motor.

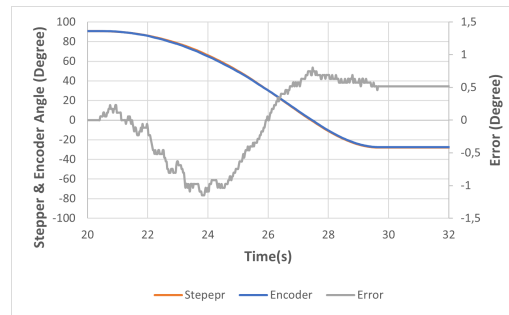
Figure 6.34: Variation in the encoder and stepper angle and their error while carrying a 0.5 kg load with a MoveL move type.



(a) First Motor.



(b) Second Motor.



(c) Third Motor.

Figure 6.35: Variation in the encoder and stepper angle and their error while carrying a 6 kg load with a MoveL move type.

6.2.3 Different Movements Comparison

A first analysis on the *MoveJ* method shows that both traditional and canonical S-curve result in oscillations with less magnitude than the trapezoidal profile. This result was to be expected as the latter produces an infinite jerk, causing more system vibrations. The canonical S-curve also proves to result in lower oscillations when compared to the traditional S-curve, resulting from the jerk generated being half of the latter method. The *MoveL* procedure generates better results than the *MoveJ* profiles, as the jerk, despite being always present and possessing an erratic value, is in the majority of the movement very close to zero, presenting only a spike in jerk at the start and the end of the movement.

A phenomenon visible in all movement types is the appearance of the first oscillation of high magnitude, as seen in the example of Figure 6.36. This is caused by the elastic properties of the rubber belt connecting the motor gear to their respective joint gear. Upon starting the motion, it creates a slack as a consequence of the resistance of the manipulator to start the movement. When the manipulator finally starts to travel, the previously generated slack creates further oscillation until the manipulator stabilises. When finishing the move, this phenomenon is also verified as the manipulator's deceleration causes the creation of yet another slack, this time generated by the momentum carried by the robotic arm. The magnitude of the first verified oscillation is directly proportional to the jerk produced at the beginning. It is also perceptible that the most affected joint was the second as anticipated, as it is the one which possesses the longest connecting belt.

The third joint was also surprisingly affected despite owning the smaller rubber belt. The high vibrations measured in this last joint most likely resulted from the material of this particular strap which was significantly thinner and pliable, basically possessing a more elastic property than the others.

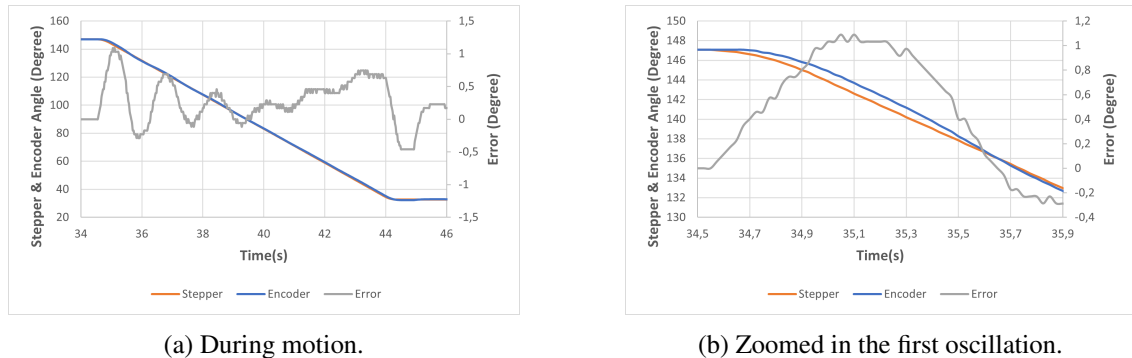


Figure 6.36: The first oscillation in error of the first joint of the trapezoidal motion profile with a 6 kg load.

Another variable in the performed tests was the change in carried load. As expected, the increase in load weight resulted in oscillations with higher magnitude and reduced frequency, independently of the applied move type. However, the higher the jerk generated, the more affected were the system vibrations.

Table 6.2: Motion Velocity Results.

	Joint	Max Velocity (<i>steps/s</i>)	Max Jerk ¹ (<i>steps/s³</i>)
Trap	1	792	∞
	2	-434	∞
	3	688	∞
S-Curv	1	811	7680
	2	-456	4160
	3	704	6720
Canon	1	832	3640
	2	-811	2080
	3	721	3360
MoveL	1	785	4288
	2	385	2944
	3	739	4032

¹ Absolute Values.

In order to complete the required movement within the same time frame, the different motion types reached different max velocities and distinct max jerks. Table 6.2 shows the values registered from each movement, where it is visible that within the *MoveJ* method, profiles that produced less jerk reached higher velocities. Furthermore, the jerk produced by the canonical S-curve is exactly half of the traditional S-curve, as both accelerations are limited to the same maximum value. However, in the canonical profile, the time duration of said acceleration is doubled, giving it

exactly half of the jerk generated, as achieved by Averalo [39]. It is worth mentioning that despite the higher value in jerk of *MoveL* compared to the canonical S-curve, the latter is maintained during a higher period of time, while the former possesses a spike of higher magnitude in a certain period.

To better display the difference in the performance of both methods, different relevant values were registered in Table 6.3. In this table is visible the maximum error achieved during motion, the settling time, the delay between the end of the movement and the error stabilisation. The results obtained support the previous statements made in the analyses of the performance of the different move types. It is visible that methods with higher jerk values resulted in a higher max error and consequently a greater settling time. With the best performance resulted from the *MoveL* method. The table also displays the values with different transported loads. As previously stated, the higher the weight of the cargo, the higher the vibrations in magnitude and the longer it takes the manipulator to stabilise at the ending of the movement. Affecting more the move types the produce higher jerks.

Table 6.3: Motion Performance Results.

		Joint	Max Error ¹ (°)	Settling Time (s)
0.5kg Added Weight	Trap	1	0.917	0.825
		2	0.688	2.100
		3	0.802	0.850
	S-Curv	1	0.802	0.800
		2	0.630	1.925
		3	0.745	0.025
	Canon	1	0.860	0.625
		2	0.630	1.475
		3	0.573	0.025
	MoveL	1	0.630	0.025
		2	0.459	0.575
		3	0.745	0.025
6kg Added Weight	Trap	1	1.089	1.950
		2	1.261	2.075
		3	1.146	1.650
	S-Curv	1	1.031	1.550
		2	1.261	1.950
		3	1.089	1.725
	Canon	1	1.031	1.500
		2	0.974	1.900
		3	0.917	1.675
	MoveL	1	0.802	0.050
		2	0.344	0.600
		3	1.146	0.025

¹ Absolute Values.

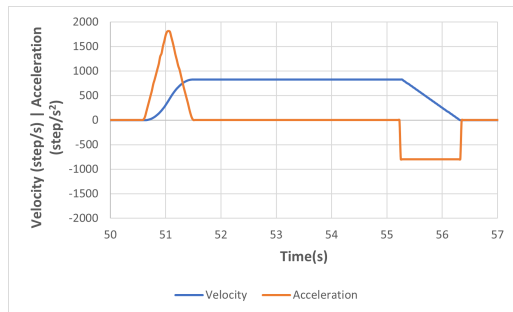
6.2.4 Stop Events

In addition to the oscillations produced during the normal functioning of the manipulator, other events might result in vibrations that are of interest to study. These events are interactions made by the user with the system during motion that cause the manipulator to alter its behaviour by pausing or performing an emergency stop. Both actions differ in the stopping methodology applied and therefore generate different results in system vibration.

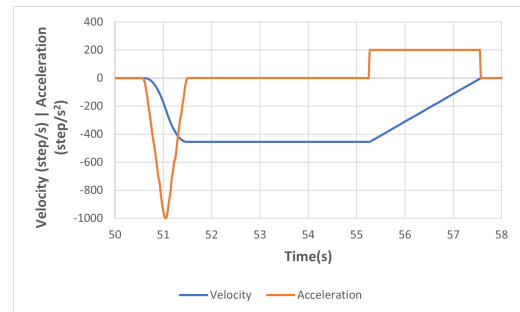
To test the different events' values, the motion was performed using the *MoveJ* method with a canonical S-Curve motion profile and moving from a left orientation of the elbow to a right orientation in order to achieve a greater disparity between the initial and desired joint value in each joint. The choice of using *MoveJ* relied on the fact that it reaches a continuous terminal velocity during motion and therefore allowed the use of the same parameters in the different tests performed.

Pause

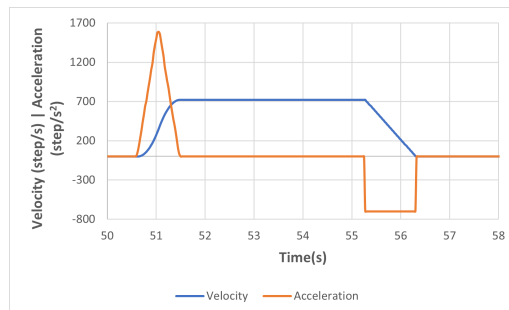
During the motion, in case of pressing the pause button, the manipulator gradually reduces its velocity, as seen in Figure 6.37. The result in the system vibration can be seen in Figure 6.38 for the 0.5 kg load and in Figure 6.39 for the 6 kg cargo, where the moment the pause started can be seen by the red line.



(a) First Motor.

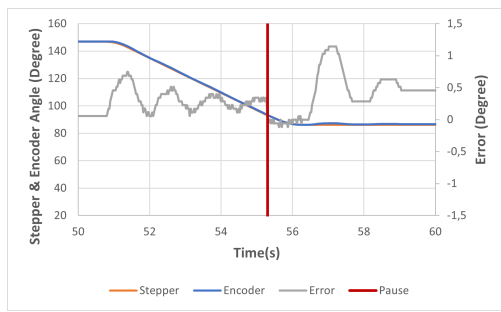


(b) Second Motor.

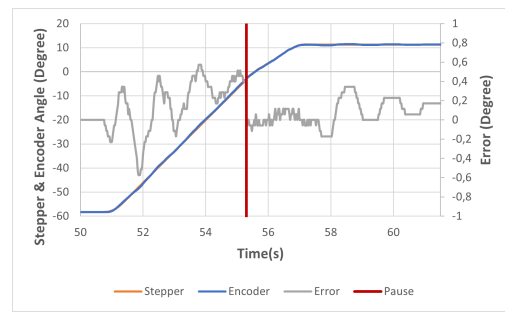


(c) Third Motor.

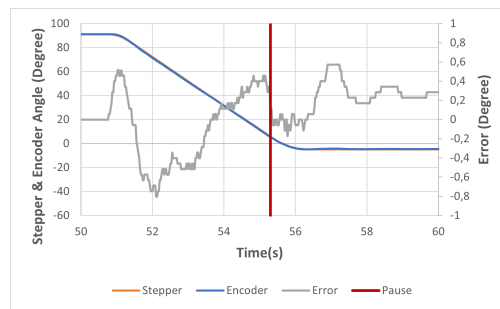
Figure 6.37: Velocity and acceleration for pausing midway through a *MoveJ* movement with a canonical S-curve motion profile.



(a) First Motor.

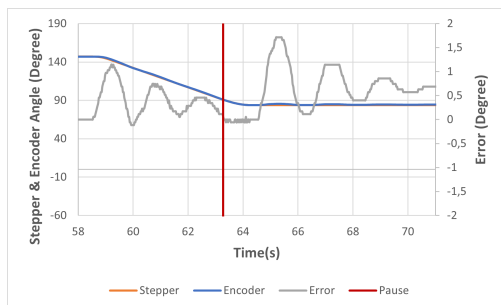


(b) Second Motor.

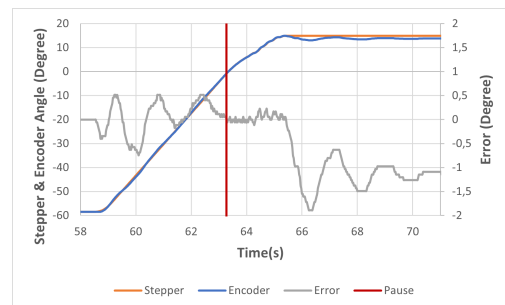


(c) Third Motor.

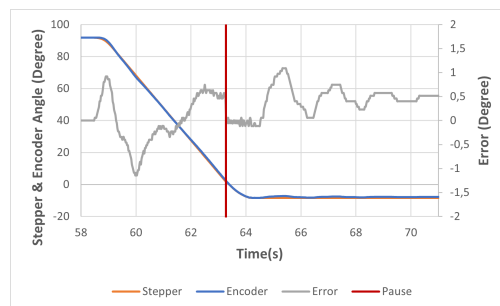
Figure 6.38: Variation in the encoder and stepper angle and their error during a pause while carrying a 0.5 kg load with a canonical S-curve motion profile.



(a) First Motor.



(b) Second Motor.

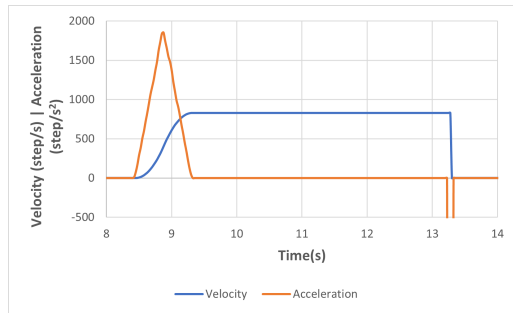


(c) Third Motor.

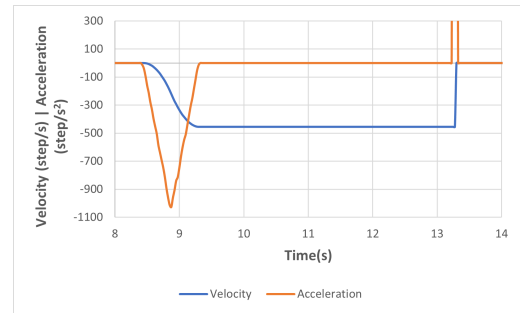
Figure 6.39: Variation in the encoder and stepper angle and their error during a pause while carrying a 6 kg load with a canonical S-curve motion profile.

Emergency Stop

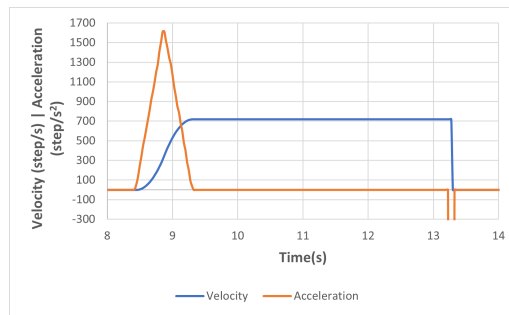
During the motion, in case of pressing the emergency stop button, the manipulator instantly reduces its velocity, as seen in Figure 6.40. The result in the system vibration can be seen in Figure 6.41 for the 0.5 kg load and in Figure 6.42 for the 6 kg cargo, where the moment the pause started can be seen by the red line.



(a) First Motor.

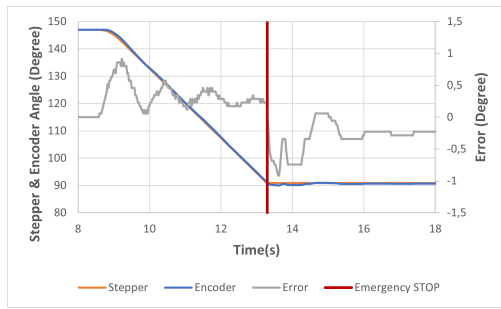


(b) Second Motor.

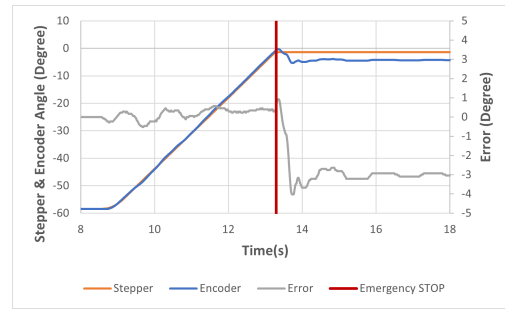


(c) Third Motor.

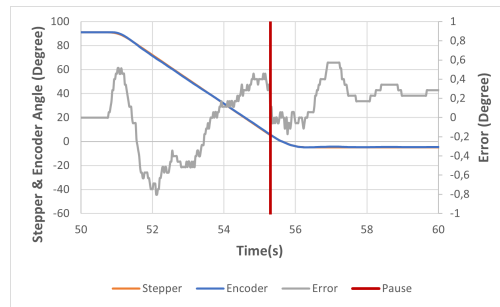
Figure 6.40: Velocity and acceleration for an emergency stop midway through a *MoveJ* movement with a canonical S-curve motion profile.



(a) First Motor.

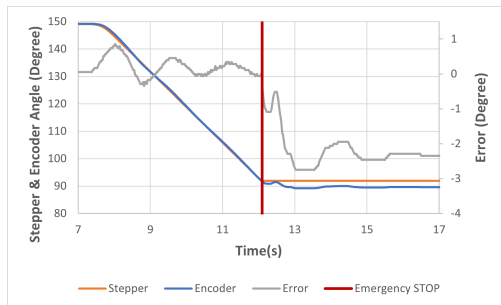


(b) Second Motor.

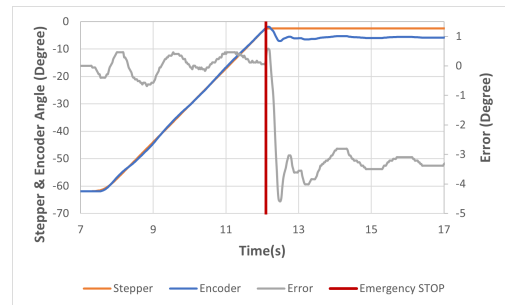


(c) Third Motor.

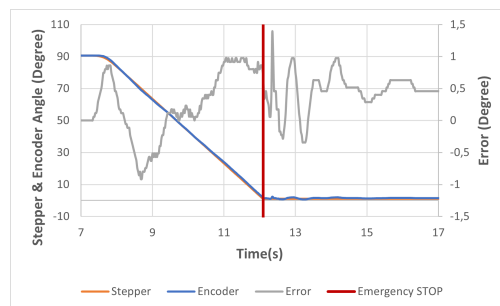
Figure 6.41: Variation in the encoder and stepper angle and their error during an emergency stop while carrying a 0.5 kg load with a canonical S-curve motion profile.



(a) First Motor.



(b) Second Motor.



(c) Third Motor.

Figure 6.42: Variation in the encoder and stepper angle and their error during an emergency stop while carrying a 6 kg load with a canonical S-curve motion profile.

Comparison

A first analysis shows that both pause and emergency stop result in highly different error curves. In the pause scenario, after it starts the deceleration, the system vibration is excited by the generation of jerk, resulting in high magnitude oscillations that significantly exceed the end of the motion. In the emergency stop, the sudden halt in the stepper motor results in another phenomenon, after the velocity of the stepper motor is set to zero, the joints continue to rotate forcefully due to the momentum carried by the manipulator. These displacements are visible in the charts and results, resulting, subsequently, in significantly high system vibrations.

Predictably, the increase in the transported load weight accentuates the phenomenons registered. When pausing, the magnitude of the vibrations increased and were reduced in frequency, in addition to a growth in the settling time. In the emergency stop, the added weight resulted in a larger displacement of the motor due to increases in resistance to stopping as well as the phenomena mentioned before.

Chapter 7

Conclusion and Future Work

This chapter provides a summary of all contents previously addressed in this document and possible future work that can be developed to improve the system.

7.1 Conclusion

The dissertation objective was to develop the control algorithm for the SCARA manipulator used throughout this project, which possessed, due to its configuration, exceptionally high susceptibility to vibrations. The objectives were successfully achieved using the devised state machine that enables the sequence of action to perform the pick-and-place tasks.

In a first instance, an analysis of the manipulator configuration was performed. Presenting the kinematics and dynamics analysis, both forward and inverse, needed to implement the control algorithms. In addition, a verification of the system hardware was accomplished to determine the best architecture and control methods. The control algorithm was achieved through the generation of motion profiles. Two different methods were used to generate them, the first one that enabled the linear control of the manipulator in the joint space and the second that allowed the control in the Cartesian space. The former approach is able to generate three different types of motion profiles, trapezoidal, traditional S-curve and canonical S-curve. In the latter, a Hermite cubic spline was used to devise to produce a straight line movement between two points. The control of the stepper motor that powered the manipulator was executed in open-loop as its configuration did not allow the use of closed-loop control.

Furthermore, two different GUIs were developed that allowed interaction with the system, a user interface to be run on a personal computer that enables the user to send the pick-and-place coordinates to the manipulator as well as the desired movement type. The application developed in raspberry was employed as the intermediary between the user GUI and the manipulator control software and executes verification of the validity of the points to be sent to it. The raspberry application also possesses a graphical interface capable of displaying various information and charts capable of storing the displayed information.

Additional features were also developed, enabling the user-system interaction by allowing the user to enable/disable the drivers, pause and perform an emergency stop during the manipulator operation. Moreover, a safety procedure was conceived to guarantee that the movement was halted upon a collision to prevent further damage.

Finally, tests were conducted to verify the designed control system's validity by testing each feature and each motion type, successfully passing all tests. In addition, a study of the effects of the different move types was conducted, concluding that lower jerk motion resulted in fewer oscillations. The best results were observed in the *MoveL* method and *MoveJ* while using the canonical S-curve. The overall results proved that the system is able to operate efficiently and perform the desired tasks smoothly.

7.2 Future Work

Further work can be implemented to improve the developed system.

The addition of extra sensors could be implemented to enable the system to perceive the surroundings to guarantee a safer control for the manipulator in a industrial environment.

Replacing the current raspberry PI 3 for the newer model, PI 4, would provide useful as certain performance limitations were encountered throughout the development of the system. Moreover, the increase in computational power would allow the use of more complex algorithms that, in addition to the previously mentioned sensors, could enable the manipulator to perceive the coordinates of both picking and placing autonomously.

A palletizer algorithm could also be implemented to increase the system's autonomy.

Lastly, the worn-out battery cells replacement in the manipulator could allow for the autonomous activation of the air compressor using the coupled inverter, reducing the stress caused by the continuous work verified during the development.

Appendix A

Manipulator Circuit Schematic

The following Figure [A.1](#) displays the circuit schematic of the manipulator, with all its components and connections. The links between the motors and drivers are visible and all hardware connected to the Arduino, such as the encoders, stepper drivers, valve, end-of-line microswitch and vacuum sensor. It also displays the two differential line receiver and the two differential line driver connecting the encoder to the Arduino.

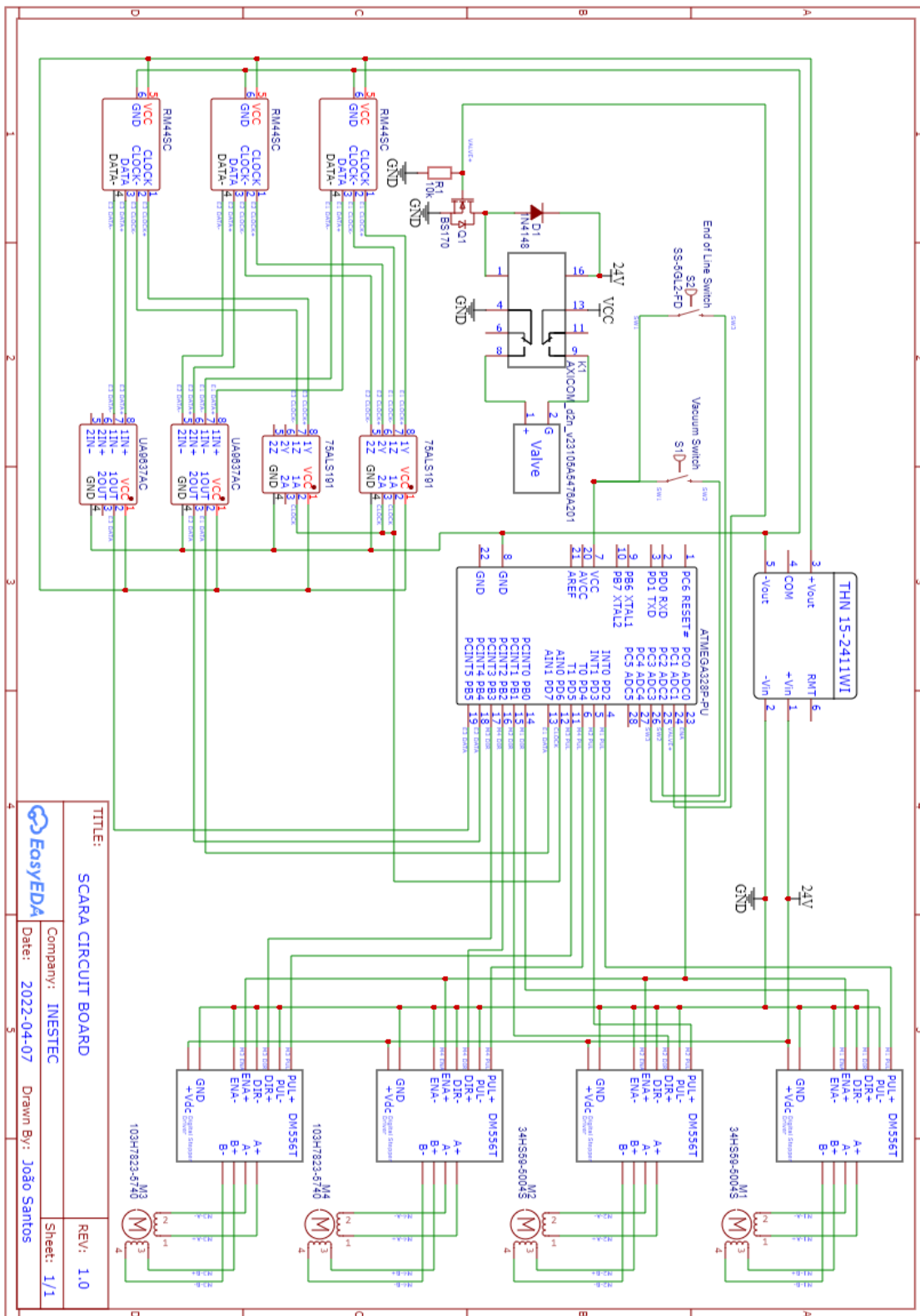


Figure A.1: Schematic of the manipulator electric circuit.

Appendix B

Arduino State Machine

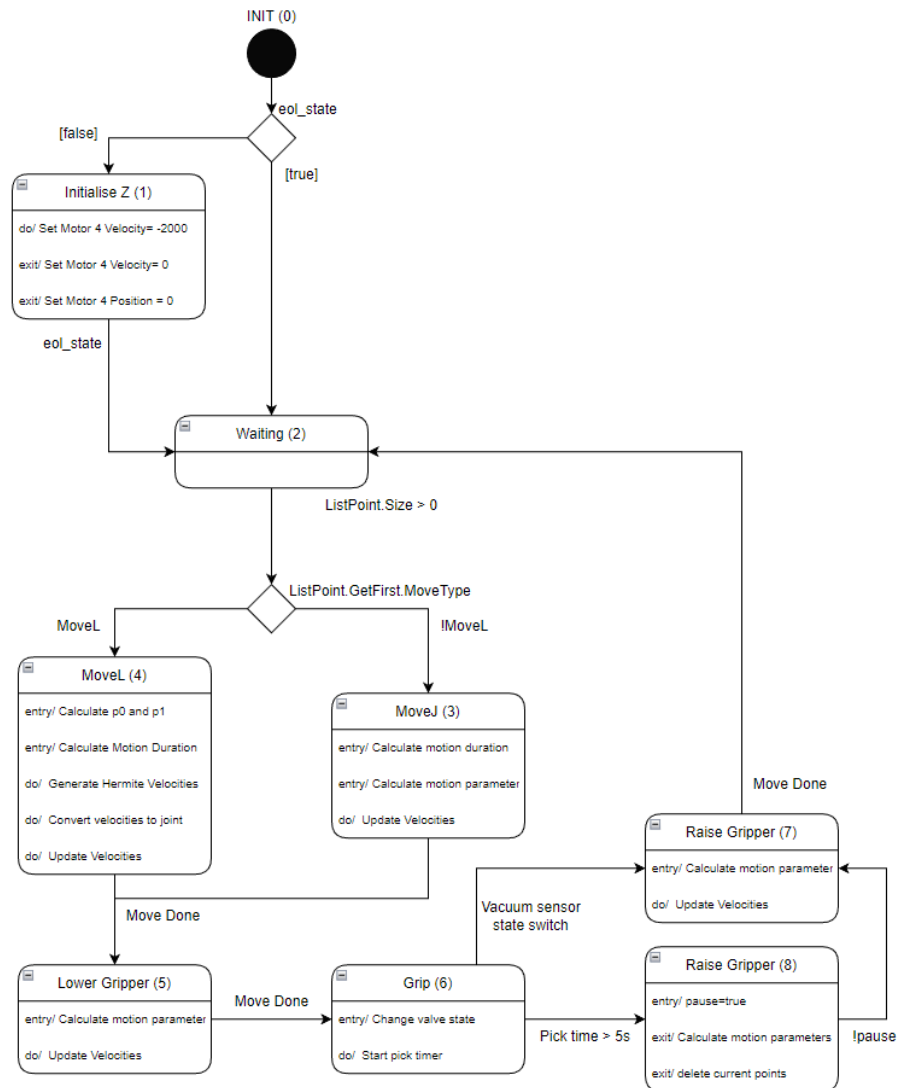
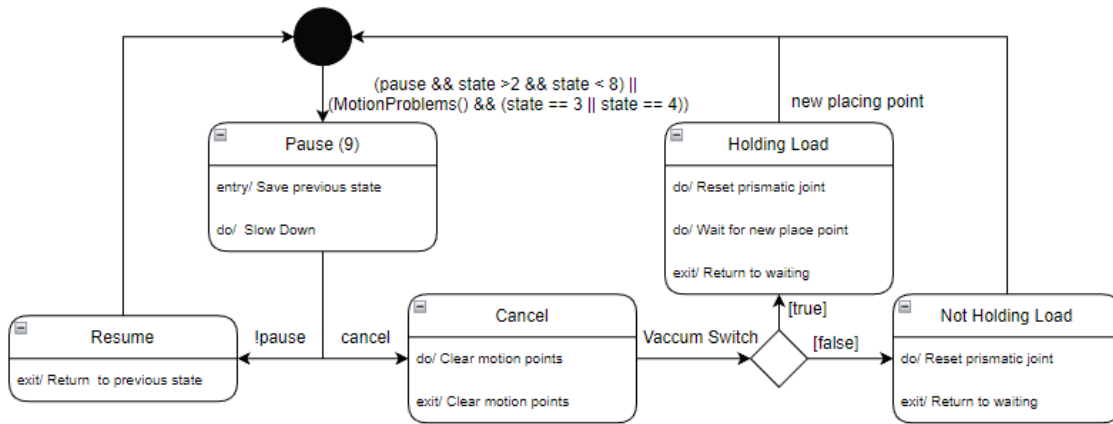
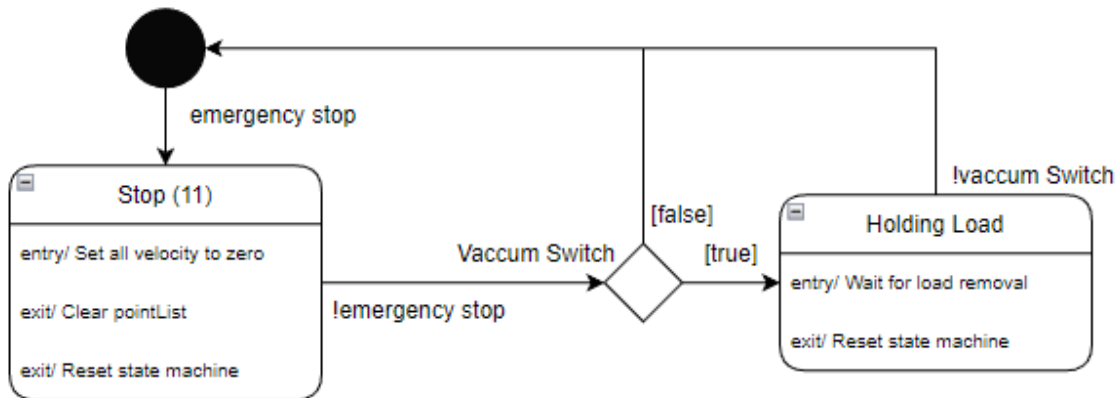


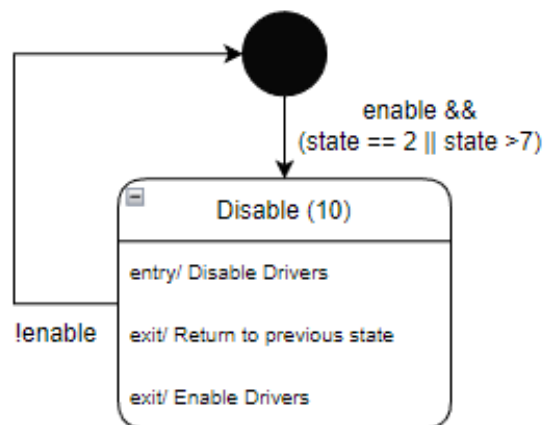
Figure B.1: Main state machine of the Arduino control algorithm.



(a) Pause.



(b) Emergency stop.



(c) Disable Drivers.

Figure B.2: State machines for additional features.

References

- [1] A. de Luca and R. Mattone. Sensorless robot collision detection and hybrid force/motion control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 999–1004, 2005. doi:10.1109/ROBOT.2005.1570247.
- [2] Dawid Gradolewski, Dawid Maslowski, Damian Dziak, Bartosz Jachimczyk, Siva Teja Mundlamuri, Chandran G. Prakash, and Wlodek J. Kulesza. A distributed computing real-time safety system of collaborative robot. *Elektronika ir Elektrotechnika*, 26(2):4–14, Apr. 2020. URL: <https://eejournal.ktu.lt/index.php/elt/article/view/25757>, doi:10.5755/j01.eie.26.2.25757.
- [3] Kien Minh Le, Hung Van Hoang, and Jae Wook Jeon. An advanced closed-loop control to improve the performance of hybrid stepper motors. *IEEE Transactions on Power Electronics*, 32(9):7244–7255, 2017. doi:10.1109/TPEL.2016.2623341.
- [4] Youngwoo Lee, Donghoon Shin, Wonhee Kim, and Chung Choo Chung. Proximate in-phase current estimator to reduce torque ripple in permanent-magnet stepping motor. *IEEE Transactions on Industrial Electronics*, 63(3):1707–1716, 2016. doi:10.1109/TIE.2015.2509949.
- [5] M. A. Vernezi, D. V. Nazarenko, and E. H. Abderrazzak. Vibration suppression of stepper motors by the electric method. In *2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pages 566–570, 2021. doi:10.1109/ICIEAM51226.2021.9446366.
- [6] Adelhard Beni Rehiara. *Kinematics of AdeptThree Robot Arm*. IntechOpen, Rijeka, 2011. URL: <https://doi.org/10.5772/17732>, doi:10.5772/17732.
- [7] T. Fredriksen. Applications of the closed-loop stepping motor. *IEEE Transactions on Automatic Control*, 13(5):464–474, 1968. doi:10.1109/TAC.1968.1098991.
- [8] Guan Fang Liu and Hua Wei Li. Design of stepper motor position control system based on dsp. In *Proceedings of the 2017 2nd International Conference on Machinery, Electronics and Control Simulation (MECS 2017)*. Atlantis Press, 2016/06. URL: <https://doi.org/10.2991/mecs-17.2017.38>, doi:<https://doi.org/10.2991/mecs-17.2017.38>.
- [9] Nehal M. Elsodany, Sohair F. Rezeka, and Noman A. Maharem. Adaptive pid control of a stepper motor driving a flexible rotor. *Alexandria Engineering Journal*, 50(2):127–136, 2011. URL: <https://www.sciencedirect.com/science/article/pii/S1110016811000329>, doi:<https://doi.org/10.1016/j.aej.2010.08.002>.

- [10] A.S. Ghafari and A. Alasty. Design and real-time experimental implementation of gain scheduling pid fuzzy controller for hybrid stepper motor in micro-step operation. In *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM '04.*, pages 421–426, 2004. doi:[10.1109/ICMECH.2004.1364476](https://doi.org/10.1109/ICMECH.2004.1364476).
- [11] S.-J. Ho, Li-Sun Shu, and Shinn-Ying Ho. Optimizing fuzzy neural networks for tuning pid controllers using an orthogonal simulated annealing algorithm osa. *IEEE Transactions on Fuzzy Systems*, 14(3):421–434, 2006. doi:[10.1109/TFUZZ.2006.876985](https://doi.org/10.1109/TFUZZ.2006.876985).
- [12] F.L. Lewis, D.M. Dawson, and C.T. Abdallah. *Robot Manipulator Control: Theory and Practice*. Automation and Control Engineering. CRC Press, 2003. URL: <https://books.google.pt/books?id=8002tUR1PP4C>.
- [13] Wen Yu and Jacob Rosen. Neural pid control of robot manipulators with application to an upper limb exoskeleton. *IEEE Transactions on Cybernetics*, 43(2):673–684, 2013. doi:[10.1109/TSMCB.2012.2214381](https://doi.org/10.1109/TSMCB.2012.2214381).
- [14] BRAD PADEN and RAVI PANJA. Globally asymptotically stable ‘pd+’ controller for robot manipulators. *International Journal of Control*, 47(6):1697–1712, 1988. doi:[10.1080/00207178808906130](https://doi.org/10.1080/00207178808906130).
- [15] Amélie Chevalier, Maarten Van Den Bossche, Cosmin Copot, and Robin De Keyser. Formation control strategies for emulation of field covering. In *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 520–525, 2014. doi:[10.1109/ICSTCC.2014.6982469](https://doi.org/10.1109/ICSTCC.2014.6982469).
- [16] S. N. Deepa S. N. Sivanandam, S. Sumathi. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. URL: https://doi.org/10.1007/978-3-540-35781-0_5, doi:[10.1007/978-3-540-35781-0_5](https://doi.org/10.1007/978-3-540-35781-0_5).
- [17] Wasif Abdlaziz Sadeh Salous. Comparison between pid and fuzzy logic control in the position of stepper motors. 2013.
- [18] P.H. Meckl and P.B. Arestides. Optimized s-curve motion profiles for minimum residual vibration. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, volume 5, pages 2627–2631 vol.5, 1998. doi:[10.1109/ACC.1998.688324](https://doi.org/10.1109/ACC.1998.688324).
- [19] Kim Doang Nguyen, I-Ming Chen, and Teck Chew Ng. Planning algorithms for s-curve trajectories. *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6, 2007.
- [20] A. Piazzzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *IEEE Transactions on Industrial Electronics*, 47(1):140–149, 2000. doi:[10.1109/41.824136](https://doi.org/10.1109/41.824136).
- [21] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1):42–52, 2003. doi:[10.1109/TRA.2002.807548](https://doi.org/10.1109/TRA.2002.807548).
- [22] Ming-Shyan Huang, Yi-Lung Hsu, and Rong-Fong Fung. Minimum-energy point-to-point trajectory planning for a motor-toggle servomechanism. *IEEE/ASME Transactions on Mechatronics*, 17(2):337–344, 2012. doi:[10.1109/TMECH.2010.2103366](https://doi.org/10.1109/TMECH.2010.2103366).

- [23] Ngoc Quy Le, Jung Uk Cho, and Jae Wook Jeon. Application of velocity profile generation and closed-loop control in step motor control system. In *2006 SICE-ICASE International Joint Conference*, pages 3593–3598, 2006. doi:[10.1109/SICE.2006.314747](https://doi.org/10.1109/SICE.2006.314747).
- [24] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. *Path Planning and Trajectory Planning Algorithms: A General Overview*, pages 3–27. Springer International Publishing, Cham, 2015. URL: https://doi.org/10.1007/978-3-319-14705-5_1, doi:[10.1007/978-3-319-14705-5_1](https://doi.org/10.1007/978-3-319-14705-5_1).
- [25] C. Lin, P. Chang, and J. Luh. Formulation and optimization of cubic polynomial joint trajectories for industrial robots. *IEEE Transactions on Automatic Control*, 28(12):1066–1074, 1983. doi:[10.1109/TAC.1983.1103181](https://doi.org/10.1109/TAC.1983.1103181).
- [26] C.-H. Wang and J.-G. Horng. Constrained minimum-time path planning for robot manipulators via virtual knots of the cubic b-spline functions. *IEEE Transactions on Automatic Control*, 35(5):573–577, 1990. doi:[10.1109/9.53526](https://doi.org/10.1109/9.53526).
- [27] DAN SIMON and CAN ISIK. A trigonometric trajectory generator for robotic arms. *International Journal of Control*, 57(3):505–517, 1993. doi:[10.1080/00207179308934404](https://doi.org/10.1080/00207179308934404).
- [28] Lianfang Tian and Curtis Collins. An effective robot trajectory planning method using a genetic algorithm. *Mechatronics*, 14(5):455–470, 2004. URL: <https://www.sciencedirect.com/science/article/pii/S0957415803000990>, doi: <https://doi.org/10.1016/j.mechatronics.2003.10.001>.
- [29] Yuhang Li, Tian Huang, and Derek G. Chetwynd. An approach for smooth trajectory planning of high-speed pick-and-place parallel robots using quintic b-splines. *Mechanism and Machine Theory*, 126:479–490, 2018. URL: <https://www.sciencedirect.com/science/article/pii/S0094114X17317810>, doi: <https://doi.org/10.1016/j.mechmachtheory.2018.04.026>.
- [30] R. Valanis, K.P. Valavanis, A. Steward, and M.J. Sonnier. A flat surface robotic gripper for handling limp material. *IEEE Robotics Automation Magazine*, 2(3):19–26, 1995. doi: [10.1109/100.414922](https://doi.org/10.1109/100.414922).
- [31] Ahmad Nuseirat, Georgios Stavroulakis, and and Panagiotopoulos. Hard and soft fingered robot grippers. the linear complementarity approach. *ZAMM Journal of applied mathematics and mechanics: Zeitschrift für angewandte Mathematik und Mechanik*, 71:257–265, 01 1991. doi:[10.1002/zamm.19910710718](https://doi.org/10.1002/zamm.19910710718).
- [32] Jochen Heinzmann and A. Zelinsky. The essential components of human-friendly robot systems. 01 2000.
- [33] Martin Geiger and Christian Waldschmidt. 160-ghz radar proximity sensor with distributed and flexible antennas for collaborative robots. *IEEE Access*, 7:14977–14984, 2019. doi: [10.1109/ACCESS.2019.2891909](https://doi.org/10.1109/ACCESS.2019.2891909).
- [34] P. Long, C. Chevallereau, D. Chablat, and A. Girin. An industrial security system for human-robot coexistence. *Industrial Robot*, 45(2):220–226, 2018. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85038033720&doi=10.1108%2fIR-09-2017-0165&partnerID=40&md5=a7593fcf3ed15a201fe699c35b3e0016>, doi:[10.1108/IR-09-2017-0165](https://doi.org/10.1108/IR-09-2017-0165).

- [35] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *2012 IEEE International Conference on Robotics and Automation*, pages 338–345, 2012. doi:10.1109/ICRA.2012.6225245.
- [36] C. Morato, K.N. Kaipa, B. Zhao, and S.K. Gupta. Toward safe human robot collaboration by using multiple kinects based real-time human tracking. *Journal of Computing and Information Science in Engineering*, 14(1), 2014. cited By 112. doi:10.1115/1.4025810.
- [37] Peter Donelan. *Kinematic Singularities of Robot Manipulators*. 04 2010. doi:10.5772/9548.
- [38] Irfaan Coonjah, Pierre Clarel Catherine, and K. M. S. Soyjaudah. Experimental performance comparison between tcp vs udp tunnel using openvpn. In *2015 International Conference on Computing, Communication and Security (ICCCS)*, pages 1–5, 2015. doi:10.1109/CCCS.2015.7374133.
- [39] Victor M. Arevalo. Sinusoidal velocity profiles for motion control. 2018. URL: https://nanopdf.com/download/sinusoidal-velocity-profiles-for-motion-control_pdf#modals.
- [40] Nasim Zeinolabedini. *Average-Case Analysis of Power Consumption in Embedded Systems*. PhD thesis, 08 2015.