HAW HAMBURG

BACHELORTHESIS
Thishan Warnakulasooriya

RTOS-Based Implementation of a Chip Validation System Featuring PID-Controlled Socket Insertion and Temperature

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Thishan Warnakulasooriya

RTOS-Based Implementation of a Chip Validation System Featuring PID-Controlled Socket Insertion and Temperature

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Ing Pawel Buczek
Second examiner: Mr. Holger Mahnke

Day of delivery: 10 September 2024

Thishan Warnakulasooriya

Title of the paper

RTOS-based implementation of a chip validation system featuring PID-controlled socket insertion and temperature

Keywords

RTOS, Control Systems, PID Control & Tuning, Parelle Processing, FSM, Automation, Integral Windup, ADC Spikes

Abstract

This document covers the requirement classification, design, implementation, development, and validation of control software for a chip validation system. The software is an RTOS that utilises PID control for the regulation of insertion force and heating of the IC.

Thishan Warnakulasooriya

Thema der Bachelorthesis

RTOS-basierte implementierung eines chip-validierungssystems mit PID-gesteuerter sockelinstallation und temperaturregelung

Stichworte

RTOS, Regelungssysteme, PID-Regelung & Abstimmung, Parallelverarbeitung, Endlicher Zustandsautomat, Automatisierung, Integralsättigung, ADC-Spitzen

Kurzzusammenfassung

Dieses Dokument behandelt die Anforderungsanalyse, das Design, die Implementierung, Entwicklung und Validierung der Steuerungssoftware für ein Chip-Validierungssystem. Die Software ist ein RTOS, das PID-Regelung zur Steuerung der Einpresskraft und Erwärmung des IC verwendet.

Table of Contents

1 Abbreviations

Abbreviation	Full Term
ADC	Analog to Digital Converter
ASMD	Algorithmic State Machine with Datapath
BL RFP	Business Line Radio Frequency Processing
FSM	Finite State Machine
IC	Integrated Circuit
ISR	Interrupt Service Routine
MCU	Micro Controller Unit
OS	Operating System
PCB	Printed Circuit Board
TX	Transmitter

2 List of Figures

3 List of Tables

4 Acknowledgement

I would like to express my sincere gratitude to my manager, Holger Mahnke, at NXP for his invaluable support and guidance throughout my project. His expertise and encouragement were crucial to the completion of this thesis.

I would also like to extend my heartfelt thanks to my supervisor, Professor Pawel Buczek, for his insightful advice and continuous support. His mentorship has been instrumental in shaping my research and achieving this milestone.

Additionally, I am deeply grateful to my colleague, Celestine Machuca, for her unwavering assistance and collaboration. Her contributions and support were invaluable.

Thank you all for your unwavering support and belief in my capabilities.

5 Introduction

5.1 Background

Under the leadership of my manager, Holger Mahnke, the Validation Department of the Business Line Radio Frequency Power (BL RFP) validates company-designed radar integrated circuits (ICs) to ensure compliance with their datasheet specifications. We then communicate the results of these validation tests to the design engineers. When necessary, adjustments are made to the IC design to ensure that the specified parameters are achieved.

The validation engineer manually subjects multiple ICs from various wafer areas to the validation test benches by selecting the IC, inserting it into the socket, and tightening the socket screw on the lid. An automated test script runs on a setup test bench with measurement and other heating and cold equipment for a tray of several dozen IC samples for a selected test case, requiring an engineer to be present on-site to swap the ICs.

The need for a lab handler to automate the IC swap procedure became evident when the lab had to manually test and configure 100 samples. Automating this process was crucial. This initial lab handler was supposed to fulfil the following requirements in Table 1 below.

Requirement	Description
Repeatability	0.1 mm repeatability of XY-axis placement (parallel to the setup board).
Maintenance	Maximum service and inspection of once per month.
Plug and play	Easy installation and immediate use require no complex setup or configuration, allowing it to function right out of the box with minimal technical expertise.
Accuracy	Precise placement and alignment of ICs on tray and socket is possible.
Scalability	The ability to handle varying quantities of ICs.
Compatibility	Support for various types and sizes of ICs is available.
Safety	The implementation of safety measures is necessary to prevent damage to ICs and equipment (ESD).
Speed	Efficient operation to minimize downtime between tests
User Interface	The intuitive controls and monitoring systems facilitate easy operation and troubleshooting.

Table 1: Previous Lab Handler Requirements

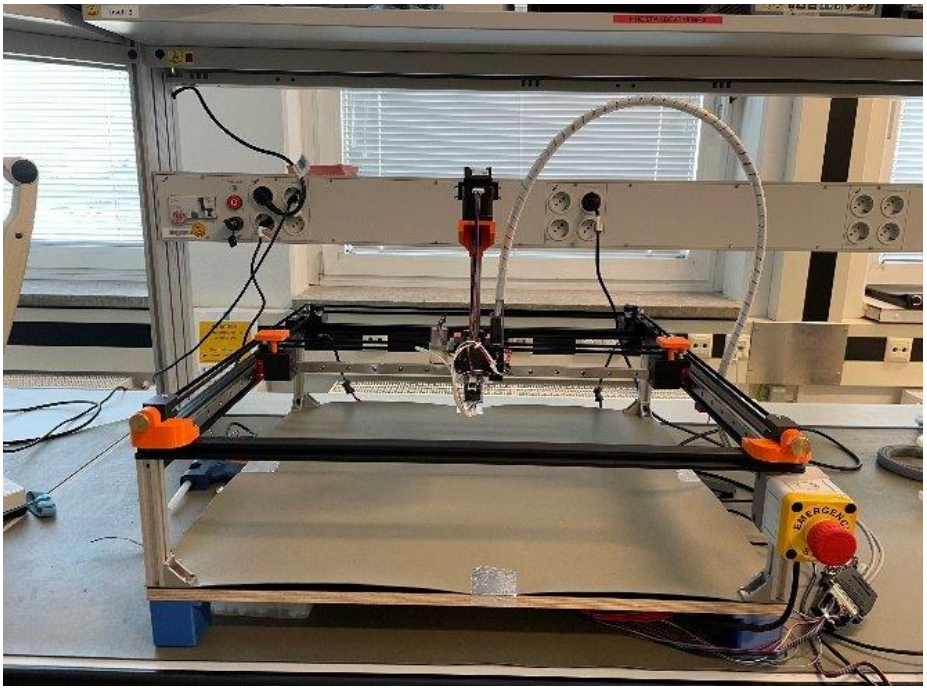

Figure 1 : CNC type Lab Handler "Bonita"

The initial two CNC-style lab handlers were developed and implemented by repurposing control units originally designed for 3D printers. After running various test setups, the validation team found the two new machines were contributing tremendously to their work by being able to run test scripts overnight, relieving engineers from manually handling samples. It was also discovered that the IC needs to be pushed with around 65N to 70N for an ideal contact with the test socket pogo pins.

While these units proved to be efficient and successful, the limited functions originally intended for 3D printers posed restrictions due to their complexity when attempting to set up new custom functions. Furthermore, their CNC-style design made them heavy and large, requiring substantial space, as can be observed in Figure 1 above. Due to their considerable size and weight, these machines lacked mobility, thereby restricting engineers to always setting up their test benches at fixed locations where the lab handlers were stationary. The need for a newer lab handler was immense, with the additional requirements stated in Table 2 below.

Requirement	Description
Mobile	It can effortlessly move from one test bench to another.
Customizable Firmware	The firmware could integrate additional features with the essential tasks.
Small Scale	Enabling it to be used in complicated test setups with limited clearing.
Scalable	It can easily be modified to be used with different IC housing sizes.

Table 2: Extended Lab Handler Requirements for the new iteration

The new iteration's concept involved purchasing a commercial Cobot and integrating it with an in-house solution to fulfil the basic functional requirements of the lab handler. To achieve this, we purchased a Cobot, the "*UFACTORY xArm 5 Lite*", which has 5 axis/joints and a maximum press and lift force of approximately 30 N [1]. This Cobot was then mounted also on an in-house designed table with the ability to be mobile and, when needed, stationary and stable by its heavy-duty wheels that are height adjustable, as shown in Figure 2 below.

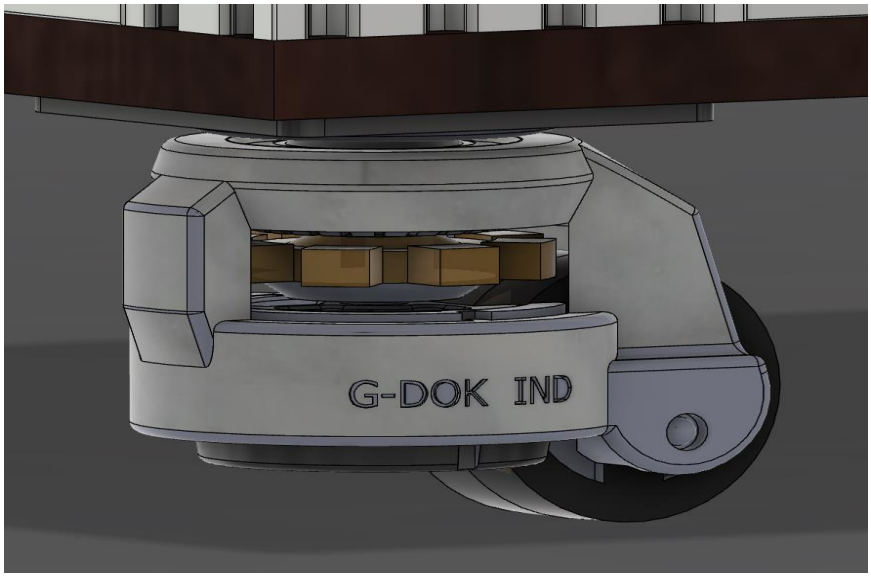

Figure 2 : Leveling Castor

The in-house solution must also be able to account for the controlled insertion force required for the project, as the Cobot's press/push force of 30N is insufficient for an ideal contact between the socket and the IC. The control system software of the in-house designed and built adapter, known as "Eva", is the primary focus and core subject of this thesis.

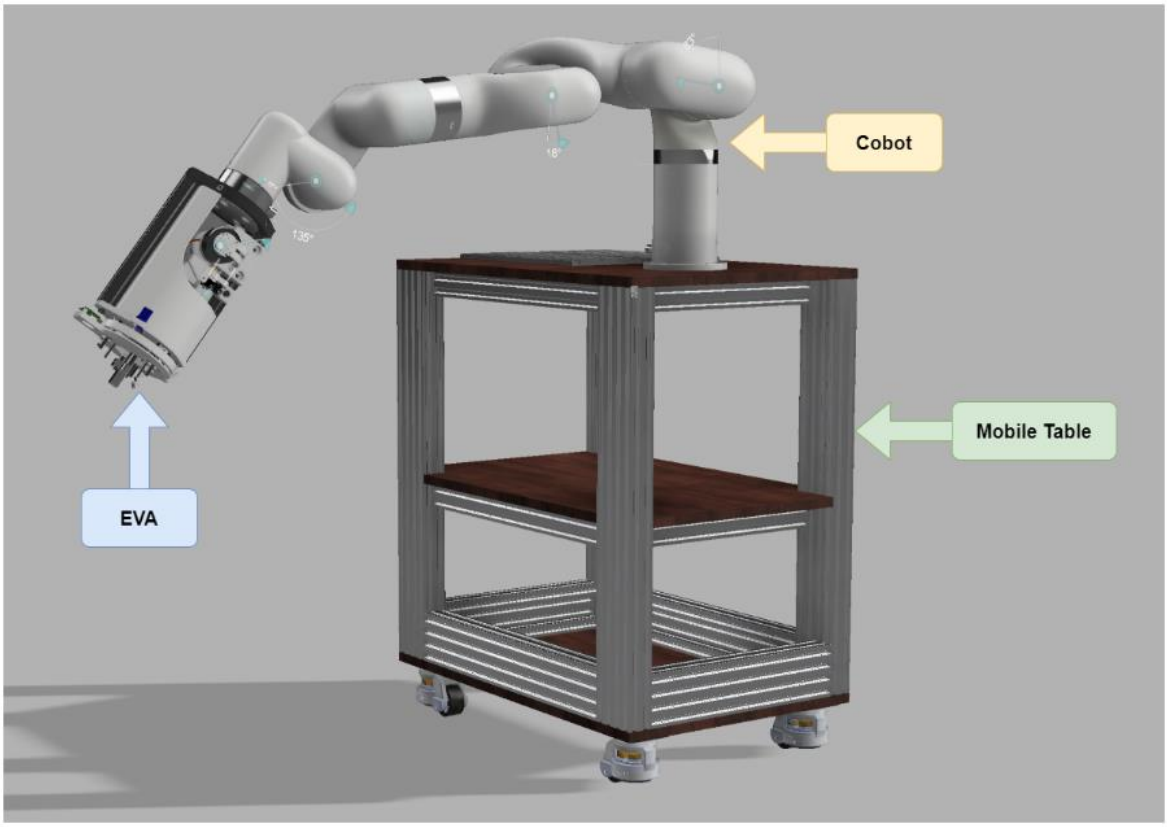

Figure 3 : Eva, connected with the Cobot and mounted on mobile table.

In Figure 3, Eva in combination with the Cobot and mounted on the mobile table setup is illustrated.

5.2 Objective

The primary objective of this thesis is to design and implement a software solution for the Lab Handler project. This involves conducting both theoretical and practical research studies to ensure the solution fulfils all the necessary requirements established during the requirements engineering phase, as outlined in the Requirements Chapter.

6 Requirements

6.1 Purpose of System

Despite the complexity and extensiveness of the system requirements, the fundamental functionality of the system can be simplified into the following basic functions, as shown in Table 3. They are the most important and basic system functional requirements.

Identifier	Function
REQ - 1	The system can **pick** up the IC from IC trays and sockets.
REQ – 2	The system can **place** the IC from IC trays and sockets.
REQ – 3	The system can **press** the IC into the socket for a maximum of up to 80N and maintain it.
REQ - 4	The system can **heat up** the IC and maintain it at a maximum temperature of 150 °C.

Table 3: Simplified System Functionality

This document consists of the design, implementation, optimization, and validation of the software solution for the required system. The software will be executed on a PCB designed in-house by Celestine Machucha and manufactured in China. The design of the physical infrastructure, including the selection of sensors, drivers, MCU, and other hardware components, was conducted in collaboration with Celestine Machucha through numerous iterations and prototyping. These aspects are not included within the scope of this document. In summary, the hardware with the smallest footprint and optimal specifications, tailored to the project requirements, was selected from the available list of devices provided by JLPCB [2]. Hardware available from JLPCB was chosen because they could ship the printed PCB boards with the components already soldered. This streamlined the manufacturing process, as the PCB was manufactured, and the hardware components were soldered by JLPCB.

The MCU ESP32-S3-MINI-1U was selected based on its compact size, affordable price, comprehensive documentation, strong community support, and well-implemented libraries from both the community and Espressif, along with its inclusion of a 240MHz dual-core processor.

6.2 Functional Requirements

6.2.1 Introduction to Functional Requirements

Functional requirements describe the specific functionalities that the system should be capable of performing [3]. They outline the essential tasks and processes that the system must execute to meet the needs of its users and stakeholders. They serve as the foundation for system design and development, ensuring that the final product aligns with the intended purpose of the system.

6.2.2 Functional Requirements of the System

The functional requirements of the system are stated and described in Table 4 below. It extends the simple requirements that were developed at the beginning of the project by modifying and extending the requirements of the previous Lab Handler CNC project.

Identifier	Requirement	Description
REQ – 5	Latency	• User interactive tasks are at least 120 Hz.
REQ – 6	Precise insertion force	• Minimum precision of 10N.
REQ – 7	Precise insertion force reading	• Minimum precision of 0.1N.
REQ – 8	Accurate insertion force reading	• Minimum accuracy of 1%.
REQ – 9	Precision in temperature reading	• 0.1 K (range 223.15 K –423.15 K).
REQ – 10	Precision in temperature step apply	• ±2 K.
REQ – 11	DUT pick & place confirmation	• Feedback of the air pressure change in the vacuum pump.
REQ – 12	Safety for the Engineer and the Validation board	• Thread / Deadlock safe. • Process infinite loop fault safety. • Emergency Stop. • PID safety.

Table 4 : Functional requirements of the control system for Eva.

6.3 Non-Functional Requirements

6.3.1 Introduction to Non-Functional Requirements

Non-functional requirements are criteria that can be used to evaluate the operation of a system, rather than specific functions and tasks. They define system attributes such as performance, security, usability, reliability, and scalability and are crucial to ensure that the system meets quality standards and performs efficiently under various conditions.

6.3.2 Non-Functional Requirements of the System

The Non-functional requirements of the system are stated and described in Table 5 below. The requirements were developed at the beginning of the project by modifying and extending the requirements of the previous Lab Handler CNC project.

Identifier	Requirement	Description
REQ – 13	Cost-efficient	• The cost should be as low as possible while being as high as necessary.
REQ – 14	Software portability	• SW can be used without major modifications across different MCUs.
REQ – 15	Optimized Power.	• Power usage for the whole system operation should be as low as possible while having the highest efficiency.
REQ - 16	System code comprehensibility	• The system code should be simple and easily readable even for an entry SW developer.

Table 5 :Non-Functional requirements of the control system for Eva.

7 Theory

This chapter dives deeply into the two primary areas of focus, RTOS and feedback-control systems, during the system's development. The system consists of several sensors that it needs to communicate with and drivers that need to be repeatedly called at a fixed frequency for smooth and efficient operation. It also consists of several control loops that are essential for correct and precise operation. The approaches taken to meet these requirements are discussed further in the RTOS and Feedback Control System chapters.

7.1 Embedded Systems

An embedded system is a combination of electronic components and software that is specifically designed to perform a specific function. An advanced embedded system typically includes a microcontroller that can be programmed to execute diverse functions, such as temperature sensing, battery level sensing, and retrieving acceleration data from an accelerometer. This system is applied in various applications including air-conditioning, remote-control devices, car entertainment systems, flight navigation systems, robotic automation in factories, MP3 players, smartphones, and smartwatches. Personal computers (PCs) running general-purpose operating systems like Windows, Linux, and Mac OS have the capability to perform a wide range of tasks and require significant resources in terms of processing power, graphics processing, and memory usage. On the other hand, embedded software is purposefully created for a particular application. An embedded operating system is specifically developed for microcontrollers with limited resources, particularly in terms of memory capacity, such as read-only memory (ROM) and random-access memory (RAM). A standard personal computer typically includes several gigabytes of random-access memory (RAM) and multiple terabytes of hard disk space. In contrast, the memory capacity of a microcontroller is significantly smaller in comparison to a PC [4].

For this purpose, an embedded OS is designed and developed resource sensitive and efficient. There are three main types of embedded OS solutions that are commonly used in industry, which is listed in Table 6 [4].

Embedded OS Solution
1. Super loop
2. Cooperative
3. Real time operating system (RTOS).

Table 6 : Most common embedded OS solutions.

7.1.1 Super Loop

Small embedded systems commonly employ a foreground and background pattern for their operating system design [5], [6]. As shown in Figure 4, the background area contains the tasks that are to be executed indefinitely. When an interrupt triggers the background tasks are interrupted and the software will switch to the interrupt service routine (ISR), which is conceptually part of the foreground area. The term for this process is pre-emption. Once the ISR is handled, it will resume execution from the exact point where it had previously paused in the background area. This is the basic operation of a super loop embedded operating system. The tasks in the background region are executed in a sequential manner. The subsequent task will be carried out only upon completion of the preceding task. Once the final task in the sequence is completed, it will cycle back to the initial task and begin again in a sequential manner [4].

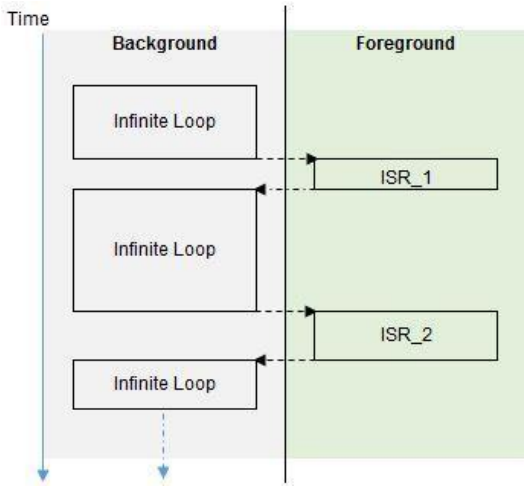

Figure 4 : Concept diagram of super-loop workflow [4].

7.1.2 Cooperative OS

The cooperative scheduler is commonly used in embedded systems. The basic concept is similar to the operation of the super loop scheduler, involving both foreground and background regions. In contrast to the sequential and cyclical execution of tasks in a super loop, the tasks in this case are organised into groups based on time slots. Their activity will be limited to the specific time slot when it is active. During the active time slot, the tasks within it will be executed in a sequential manner and will only be served once. Subsequently, they will remain inactive until their designated time slice becomes active once more. The cooperative scheduler is generally considered to be more structured and predictable compared to a super loop [7], [8]. As shown in Figure 5, the timer is programmed to accurately measure the passage of time. When the timer interrupt triggers, the background tasks will be interrupted, and the timer ISR routine in the foreground will be executed to record the time [9]. For example, one second had elapsed, two seconds had elapsed, and so forth. Based on the definition of the requirements, a flag can be activated within the ISR when the designated time is reached. As an example, the Timer 1 flag is activated when 1 second elapses, while the Timer 2 flag is activated when 2 seconds elapse. Subsequently, the program will resume execution from the precise location where it had previously been paused within the background region. Using this mechanism, the scheduler could selectively execute tasks that are assigned to a specific time region. Task 1 will only be executed when the Timer 1 flag is active, and Task 2 will only be executed when the Timer 2 flag is active. Contiki and TinyOS are operating systems that have implemented the cooperative mechanism [4].

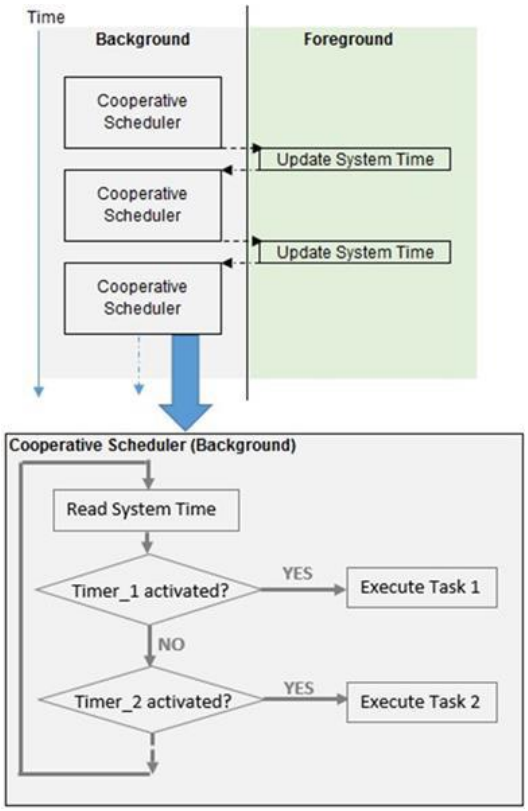

Figure 5 : Concept diagram of cooperative-loop workflow [4].

7.1.3 Real-time operating system (RTOS)

A real-time operating system (RTOS) is significantly more complex than a super loop and cooperative scheduler, and it has a unique deterministic capability in contrast to other operating systems.

The two categories of RTOS are hard RTOS and soft RTOS [10]. The hard real-time operating system (RTOS) consistently meets the specified deadline, whereas the soft real-time operating system (RTOS) is able to meet the deadline on the majority of occasions.

For time-sensitive real-time applications, especially in industries such as automotive and military, the use of a hard RTOS (Real-Time Operating System) is required. Failing to meet a deadline in these situations could result in fatal consequences. For example, the implementation of a car airbag: if the system needs to activate the airbag within 50 milliseconds after detecting a collision, the real-time operating system (RTOS) must guarantee that this time limit is consistently met.

Unlike simpler scheduling methods like super loops or cooperative schedulers, which can be developed internally relatively easily and quickly, the industry often relies on third-party real-time operating system (RTOS) solutions for more complex requirements. The kernel, or scheduler, is the core element of a real-time operating system (RTOS), with the primary responsibility of managing and supervising the execution of tasks within the system.

A commonly employed scheduling technique in real-time operating systems (RTOS) is round-robin with time slicing, which involves assigning time slots to tasks of equal priority, allowing them to run for a specified duration before being pre-empted to give way to other tasks. With the time slots being small and optimized enough to mimic parallel processing, especially for the human eye, and many real-world tasks, even though only one task is run on a single-core processor.

In Figure 6 an implementation of round-robin scheduling can be observed. All three tasks A, B, and C have the same priority and have an equal amount of time allocated (T1 = T2 = T3). In this case, task A starts first, then B, and finally C, and they require, respectively, three, two, and one time slots for complete execution. It is observed that task C is executed three times while task A completes the execution cycle in Figure 6.

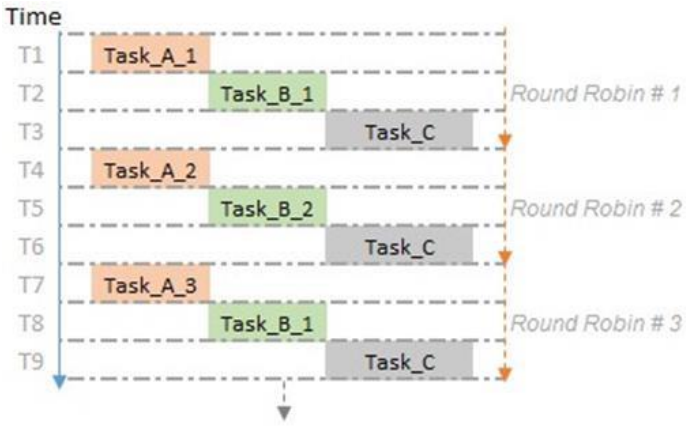

Figure 6 : Scheduled tasks in round robin pattern [4].

RTOS schedulers used in the industry are more complex than just having sliced time slots for tasks. Figure 7 shows the entire system of a commercial RTOS-µC/OS-III [11].In layman's terms, it is contained within a background and foreground design pattern. The ISR is situated in the foreground region, while the tasks are situated in the background. The round-robin time

slicing pattern is used to schedule the low-priority tasks, and as soon as an interrupt is triggered, the program will transition into the ISR, and the running task will be pre-empted. In the ISR, a higher priority task is made active, and this will be immediately detected by the scheduler at the completion of the ISR. It will serve this new higher priority task before reverting to the low priority task that was halted when the interrupt was triggered.

Figure 7 : RTOS workflow overview [4].

The aforementioned OS types cannot efficiently manage and operate the system due to its complexity, making RTOS the optimal choice. The following section will further discuss several possible RTOS solutions that best suit the system under implementation.

7.1.4 RTOS for ESP-32 S3

The PCB for the system consists of an ESP32-S3-MINI-1U microcontroller, which consists of two 32-bit Xtensa LX7 microprocessors. In Table 7, the system specifications of the MCU that were relevant when selecting the compatible MCU for this system are listed.

Category	Details
CPU and On-Chip Memory	• ESP32-S3 embedded, Xtensa® dual-core 32-bit LX7 microprocessor (with single precision FPU), up to 240 MHz • 384 KB ROM • 512 KB SRAM • 16 KB of SRAM in RTC • Up to 8 MB of quad SPI flash • 2 MB of PSRAM (ESP32-S3FH4R2 only)
Peripherals	• GPIO, SPI, LCD interface, camera interface, UART, I2C, I2S, remote control, pulse counter, LED PWM, full-speed USB 2.0 OTG, USB Serial/JTAG controller, MCPWM, SDIO host, GDMA, TWAI® controller (compatible with ISO 11898-1, i.e., CAN Specification 2.0), ADC, touch sensor, temperature sensor, timers, and watchdogs
Operating Conditions	• Operating voltage/power supply: 3.0 ~ 3.6 V • Operating ambient temperature: -40 ~ 85 °C
Tests	• HTOL/HTSL/uHAST/TCT/ESD

Table 7 : ESP32-S3-MINI-1U MCU Specifications [12].

For the selected MCU, the best possible RTOSs are Free-RTOS, Zephyr, and NuttX. These three systems are compared with each other in Table 8 based on the details obtained from their documentation [12], [13], [14].

Feature/RTOS	Free-RTOS	Zephyr	NuttX
Ease of Use	Easy to learn and implement	Moderate complexity, steep learning curve	Relatively simple, some learning needed
Community Support	Large, active community, extensive resources	Strong industry and community support	Smaller community, fewer resources
Modularity	Moderate customization, basic modularity	Moderate customization, basic modularity	Balanced modularity, moderate customization
POSIX Compliance	Minimal POSIX support	Partial POSIX support, some compatibility	Full POSIX compliance, easy porting
Memory Footprint	Lightweight, minimal resource usage	Moderate, depends on configuration	Low, designed for constrained devices
Advanced	Basic, sufficient for most	Rich feature set,	Moderate, suitable for

Features	applications	advanced networking	many use cases
Learning Curve	Low, straightforward	High, requires more effort	Medium, manageable moderate effort

Table 8 : Comparison of most used RTOS for ESP32-S3

Based on the comparison between the top possible RTOS's for the selected MCU, Free-RTOS was chosen to design and implement the system for the project mainly because of its ease of implementation and large community support. In the next section the basic structure of Free-RTOS and its features are discussed.

7.1.5 Free-RTOS

Free-RTOS is an open-source, lightweight Real-Time Operating System (RTOS) primarily written in the C programming language [7] which is specially designed for embedded systems and low-end IoT applications. Its performance in simplicity, scalability, and portability makes it an ideal choice for a wide range of applications [15] and has been ported over 27 different architectures, making it highly versatile and adaptable across various hardware platforms [16].

The architecture of Free RTOS is similar to other RTOSs, and tasks communicate through the kernel and drivers with hardware, as shown in Figure 8. Inter-task communication is done through the use of queues, where the task with the highest priority is granted access to the queue before others [7]. In Table 9, key features of Free-RTOS are summarized for an easy overview of the system [17].

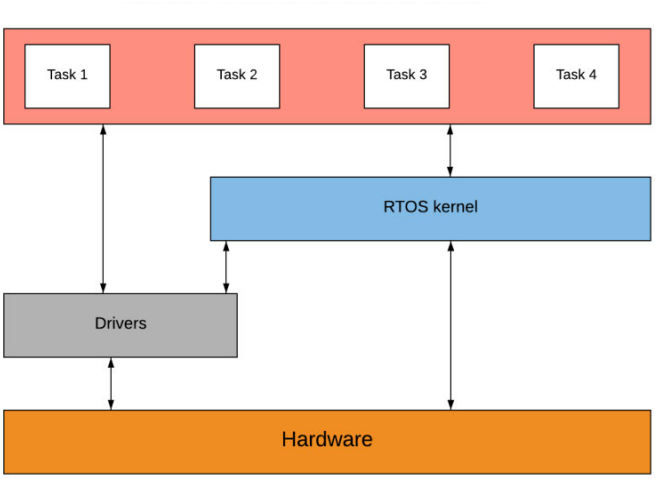

Figure 8 : Architecture Overview for VxWorks and FreeRTOS [18].

Feature	Description
Programming Model and API	User-friendly API that supports multiple threads, mutexes, semaphores, and timers. Lacks a hardware abstraction layer (HAL), which can increase debugging efforts in certain environments, like STM32Cube MCU firmware.
Scheduling	Configurable scheduler with options for fixed-priority pre-emptive or cooperative strategies. Utilizes Round-Robin (RR) scheduling for tasks with the same priority.
Memory Management	Supports dynamic memory allocation with a small memory footprint, making it suitable for resource-constrained environments.
Networking Protocols	Supports 6LoWPAN, CoAP, and Free-RTOS+TCP, a thread-safe TCP/IP stack for robust networking in embedded applications.
Simulation and Testing	Simulatable on Windows (Win32 simulator using Visual Studio 2015) and Linux (POSIX/Linux simulator using GCC and Eclipse), facilitating testing and debugging before hardware deployment.
Security	Uses WolfSSL, a lightweight TLS/SSL library for security, offering features like authentication, integrity, and confidentiality. Ideal for embedded systems due to its small footprint, 20 times smaller than OpenSSL.
Power Consumption	Includes features to reduce power consumption, such as an idle task hook and tickless idle mode, which stops periodic tick interrupts during idle periods, beneficial for battery-powered applications.

Table 9 : Free-RTOS features summarized

The use of Free-RTOS for the design of the system is more extensively discussed in the implementation chapter.

7.2 Control Systems

7.2.1 Introduction to Control Systems

Control systems are a key component in engineering fields that command, regulate, or manage the behaviour of devices or systems involved in some task or application by using a control loop. In Table 10, the main components of a control system are listed.

Component	Description
Sensor	Measures the output or state of the system and converts it into a signal that can be interpreted by the controller.
Controller	Processes the sensor's signal, compares it to the desired setpoint, and calculates the necessary control action to minimize the difference (error) between the desired and actual outputs.
Actuator	Executes the control action by adjusting the input to the system, such as opening a valve, increasing a motor speed, or altering the electrical current.
Plant (or Process)	The part of the system being controlled, which could be a mechanical device, an industrial process, or any system that requires regulation.

Table 10 : Components of a control system

7.2.2 Open-loop and Close-loop control systems

Based on their approach to controlling and utilising feedback, control systems can be categorised into two main types: open-loop control systems and closed-loop (feedback) control systems.

i. Open-Loop Control Systems

An open-loop control system is a control system that operates without using feedback from the output to influence or adjust the control inputs. In other words, the system's action is determined entirely by the initial input or a set of predefined instructions, without consideration of the outputs. An electric kettle that boils water for a set time is an example of an open-loop control system where it operates based on time, regardless of whether the water has reached the boiling point.

ii. Closed-Loop (Feedback) Control Systems

A closed-loop control system is a control system that continuously monitors the system output and uses this feedback to influence or adjust the control inputs. In other words, the system compares the measured output with the required setpoint and generates a corrective action that will bring the error between the system output and the desired setpoint to a minimum.

The thermostat-controlled heating system in a house act as a common example of a closed-loop system where it continuously monitors the ambient temperature of the room and compares it to the set and desired setpoint. If the ambient temperature of the room differs from the set point, the thermostat automatically controls the heating system in order to restore the temperature to the desired level. The mentioned feedback loop serves to maintain a consistent and appropriate temperature within the room, even in the face of changes in external circumstances. In Table 11 a comparison between open-loop and close-loop control is listed.

Aspect	**Open-Loop Control Systems**	**Closed-Loop (Feedback) Control Systems**
Control Action	Independent of system output; based on preset commands.	Dependent on system output; adjusted based on feedback.
Accuracy	Lower accuracy; cannot correct for disturbances.	Higher accuracy; can correct deviations and disturbances.
Complexity	Simpler and easier to design and implement.	More complex; requires sensors, feedback mechanisms, and advanced controllers.
Cost	Generally lower cost due to fewer components.	Higher cost due to the need for additional components like sensors.
Adaptability	Inflexible; cannot adapt to changes in system dynamics.	Highly adaptable; can adjust to changes and disturbances.
Applications	Suitable for systems where precision and adaptability are not critical.	Essential for systems requiring high precision and stability.

Table 11 : Comparison between open vs close loop control systems.

Since the system requires precise and stable output for controlling the force applied and heater temperature, closed-loop feedback is needed, and therefore, in the next section, more details about feedback control systems are listed.

7.2.3 Feedback Control Systems

The main objective of a feedback control system is to ensure that the output of a process or system follows a predetermined path, even when the system or process is influenced by disturbances. Control systems play a prominent role in today's technology, spanning a wide range of applications, from basic household appliances to complex industrial processes. Several reasons as to why feedback control systems are essential are listed in Table 12 [19]

Aspect	Description
Counteracting disturbances	External disturbances can significantly impact the output of a system. Feedback allows the system to automatically adjust its input to counteract these disturbances, ensuring stable operation.
Improving performance amid uncertainty	In cases where the system model is uncertain or imperfect, feedback helps correct discrepancies between the desired and actual outputs.
Stabilizing unstable systems:	Many industrial processes are inherently unstable in an open-loop configuration. Feedback is necessary to stabilize such systems, making their operation safe and reliable.

Table 12 : Importance of feedback control systems

There are two main types of feedback control systems: negative-feedback and positive-feedback systems. In a negative feedback control system, the output is subtracted from the setpoint, and the resulting error signal is used to adjust the input. Since this control type tends to be the best at stabilising the system for disruptions, it is the most common control type used in industries.

In a positive feedback control system, the addition of the output and the setpoint is considered when controlling, which often causes deviations and leads to system instability. Hence, this control type is less commonly used in industries. In Table 13, commonly used feedback controllers are compared with each other. In the next section, the PID controller will be

discussed because of its suitability to the project requirements, among other types of controllers.

Controller	Description	Advantages	Disadvantages	Typical Applications
On-Off (Bang-Bang) Controller	A simple controller that switches the output fully on or off based on whether the process variable is above or below the setpoint.	Simple, low-cost, and easy to implement.	Can cause oscillations and wear due to rapid switching; no fine control.	Thermostats, simple motor control, level control in tanks.
Proportional (P) Controller	Produces an output that is proportional to the current error. The control action is stronger when the error is larger.	Simple design, reduces steady-state error more effectively than on-off control.	Cannot eliminate steady-state error, may require manual tuning.	Flow control, pressure control, basic temperature regulation.
Proportional-Integral (PI) Controller	Combines proportional control with an integral component that accounts for the accumulation of past errors.	Eliminates steady-state error, relatively simple to design and implement.	Slower response to sudden changes compared to PD controllers, potential for overshoot.	Temperature control, speed control in motors, liquid level control.
Proportional-Derivative (PD) Controller	Combines proportional control with derivative action, which anticipates future errors by considering the rate of error change.	Improves system stability and response time, reduces overshoot.	Does not eliminate steady-state error, sensitive to noise in the system.	Motion control, robotics, systems requiring quick response.
Proportional-Integral-Derivative (PID) Controller	The most used controller that combines proportional, integral, and derivative actions to balance accuracy, stability, and response time.	Highly versatile, can be tuned to optimize performance for a wide range of applications.	Complexity in tuning the three parameters (P, I, D), potential for instability if not tuned correctly.	Industrial process control, motor drives, temperature regulation, flow control.

Fuzzy Logic Controller	A non-linear controller based on fuzzy set theory that handles imprecision and uncertainty by using a set of rules to determine control actions based on the error.	Robust to uncertainty and non-linearities, can be designed without a precise mathematical model.	Complex to design and tune, performance can be difficult to predict.	Consumer electronics (e.g., washing machines), automotive systems, complex process control.
Neural Network Control	Utilizes artificial neural networks to model and control systems, particularly useful for non-linear and complex systems where traditional methods fall short.	Capable of handling highly non-linear systems, can learn from data to improve performance.	Requires large datasets for training, computationally intensive, complex to design.	Robotics, autonomous systems, advanced manufacturing, complex non-linear processes.

Table 13 : Comparison of commonly used feedback control systems

7.2.4 PID Controller

The proportional-integral-Derivative (PID) controller is the most widely used control algorithm in feedback control systems, where the controller adjusts the control input based on three terms: the proportional term (P), the integral term (I), and the derivative term (D). In Table 14, these three terms, along with their descriptions, are listed.

Term	Description
Proportional Control (P)	The proportional term generates a control signal that is directly proportional to the error signal. It helps to reduce the error by applying a corrective action that is scaled according to the magnitude of the error.
Integral Control (I)	The integral term addresses the accumulation of past errors by integrating the error over time. This helps eliminate steady-state errors that may persist even after the proportional control has been applied.
Derivative Control (D)	The derivative term anticipates future errors by considering the rate of change of the error signal. It provides a damping effect, reducing the likelihood of overshoot and improving system stability.

Table 14 : Three main parts of a PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\ d\tau + K_d \frac{de(t)}{dt} \qquad (1)$$

The equation for a PID controller is given by equation (1), and its components are introduced in Table 15. These proportional, integral, and derivative gains should be tuned manually or using other optimisation algorithms depending on the specific application to make sure the PID controller will be efficient and effective for the application system. In Figure 9, the basic block diagram of a PID controller is shown.

Term	Description
u(t)	The control signal sent to the system (such as the voltage applied to a motor).
e(t)	e(t) is the error at time *t* which is the difference between the desired setpoint and the actual process variable.
K_p	The proportional gain, it Determines how aggressively the controller responds to the current error. Higher values can reduce the error faster but may cause instability.
K_i	The integral gain addresses accumulated errors over time, eliminating steady-state errors. Higher values improve long-term accuracy but can slow down the response.
K_d	The derivative gain predicts future errors and helps stabilise the system by counteracting rapid changes in the error. Higher values reduce overshoot but can make the system sensitive to noise.

Table 15 : Components of the PID control equation.

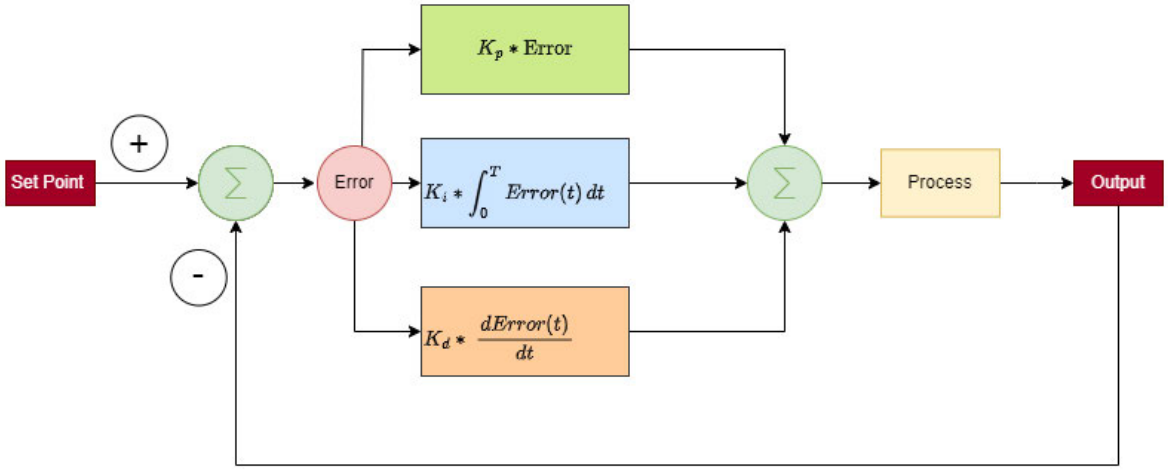

Figure 9 : Block diagram of a basic PID controller.

In the next section, two methods for tuning the PID parameters are introduced and discussed, which have been used to tune the PID controllers implemented in this project system.

7.2.5 PID Tuning

PID tuning is an important aspect when it comes to feedback control systems. Without proper tuning, the system would perform worse than a simple on-off controller and would be unstable. Proper tuning involves adjusting either manually or using algorithms, the proportional (K_p), derivative (K_d), and integral (K_i), gains to achieve the desired system response, which typically includes minimizing overshoot, settling time, and steady-state error while ensuring system stability. The terms overshoot, settling time, and steady state error are listed along with a description in Table 16 and is also illustrated in Figure 10.

Metric	Description	Significance
Overshoot	The extent to which the system's output exceeds the desired setpoint or final value after a disturbance.	High overshoot can indicate instability or excessive oscillation. Reducing overshoot leads to smoother response.
Settling Time	The time required for the system's output to settle within a specified percentage (e.g., 2% or 5%) of the final value.	Shorter settling times are preferred as they indicate a quicker return to stability after a disturbance.
Steady-State Error	The difference between the desired setpoint and the actual output after the system has stabilized.	Minimizing steady-state error ensures the output closely matches the desired value, indicating accurate control.

Table 16 : Overview of the terms overshoot, settling time and steady state error

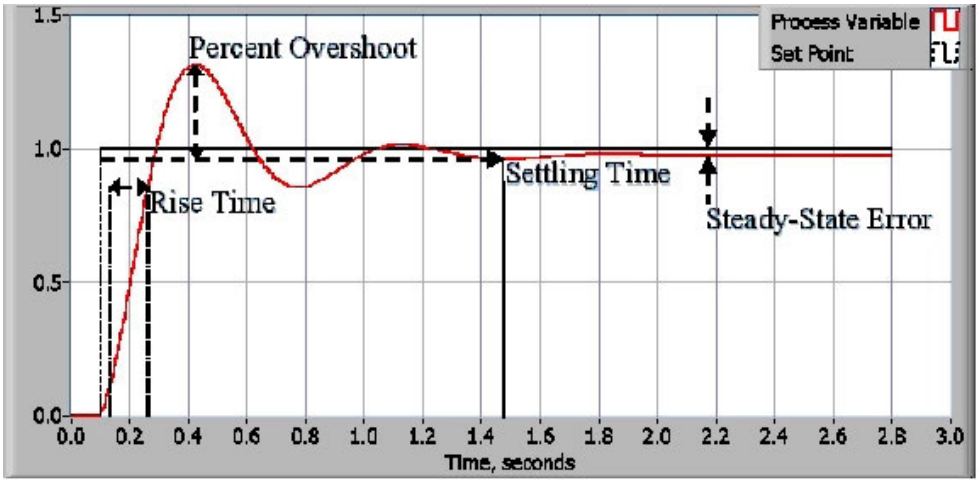

Figure 10 : Response of a typical PID closed loop system [20].

Depending on the control application, suitable tuning methods should be applied; otherwise, the control would end up making the system unstable or oscillate excessively. PID tuning could be divided into two main types of tuning, namely closed-loop and open-loop tuning.

In close-loop tuning, the PID controller parameters are adjusted while the control loop is closed, where the controller is actively controlling the process. Here, the feedback from the process output is used to adjust the control action continuously. Whereas in open-loop tuning, the parameters of the PID controller are adjusted while the control loop is open, the controller has no influence on the process during the tuning. Here, the process is manually driven, and the controller is tuned based on the open-loop response of the system. In summary, the system uses its frequency response in close-loop to tune the parameters, and its step response in open-loop [21].

In the next section, two methods, namely the Ziegler-Nichols Tuning Method and the Tyreus-Luyben Tuning Method, which are closed-loop tuning methods, are examined. These methods are more commonly used when detailed system dynamics, such as the system transfer function or time domain response, are not known or hard to achieve, which is the case in this project system.

I. Ziegler-Nichols Tuning Method

Ziegler–Nichols is a widely used approach for setting the parameters of P, PI, and PID controllers. Starting with the integral and differential gains being zeroed, this approach gradually increases the proportional gain until the system becomes unstable. The frequency

of oscillation is f_0(ultimate oscillation frequency), and the value of K_p at the point of instability is defined as K_{max} (ultimate gain). The method then reduces the proportional gain by a predetermined amount and sets the integral and differential gains as a function of f_0. The gains for P, I, and D are determined in accordance with Table 17 [22].In Figure 11, these tuning steps are illustrated in a flowchart for easier follow-through.

Controller Type	K_P	K_I	K_D
P controller	0.5 K_{MAX}	0	0
PI controller	0.45 K_{MAX}	1.2 f_0	0
PID controller	0.6 K_{MAX}	2.0 f_0	0.125/f_0

Table 17 : Gain calculation for P, PI & PID controllers according to Ziegler - Nichol's method.

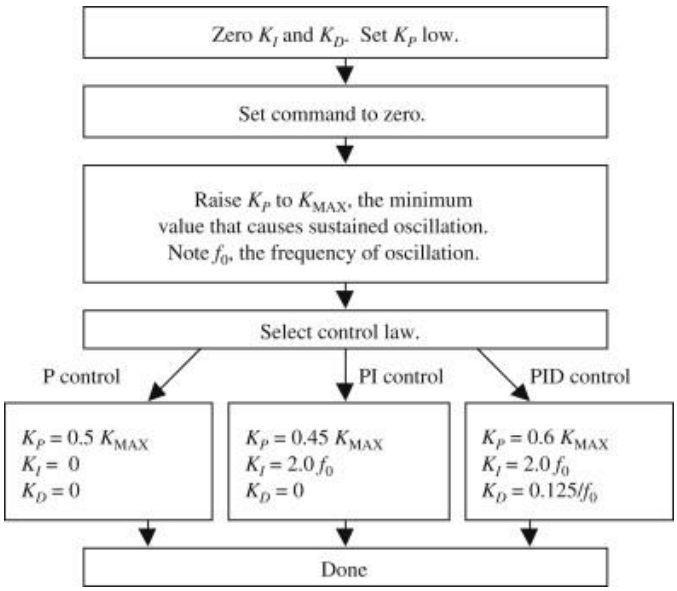

Figure 11 : Steps of tuning a controller according to Ziegler Nichols method [22].

II. Tyreus-Luyben Tuning Method

The Tyreus-Luyben tuning method, which was introduced in 1997 [23], [24], is based on ultimate gain and ultimate period, as in the Ziegler-Nichols method. However, the formulas for the controller parameters were modified to achieve more stability in the control loop compared to the Ziegler-Nichols method. This method also follows the same steps as Ziegler–Nichol's method to obtain the ultimate gain, K_{max} and the ultimate oscillation

frequency, f_0 but uses less aggressive pre-determined multiplicands for calculating the gains K_p K_i and K_d as listed in Table 18.

Controller Type	K_P	K_I	K_D
P controller	0.33 K_{MAX}	0	0
PI controller	0.31 K_{MAX}	$\frac{K_p}{2.2 \times f_0}$	0
PID controller	0.45 K_{MAX}	$\frac{K_p}{2.2 \times f_0}$	$\frac{K_p \times f_0}{6.3}$

Table 18 : Gain calculation for P, PI & PID controllers according to Tyreus-Luyben method

The objective of using these methods is to obtain PID parameters that result in the shortest possible settling time while minimising overshoot and ensuring system stability. By carefully tuning the PID controller, these methods aim to achieve an optimal balance between response speed and robustness, leading to enhanced overall system performance. In practice, these methods need to be further adjusted and fine-tuned manually by human operators according to the specific demands of the application.

In the implementation chapter the implementation, drawbacks and further improvements of using these methods are discussed.

7.2.6 PID Integral Windup

Integral windup happens in PID controllers when the integral term builds up too much because of error signals that last too long. This causes the controller output to reach its saturation points (100% or 0%). This occurs because the integral action continues to increase even when the system cannot respond, resulting in a stagnation of the output despite increasing input. This condition can cause significant control issues, as the system may continue to operate with a sustained error, unable to correct itself. Various methods have been developed to prevent integral windup and ensure the PID controller functions effectively without reaching these limits [25].

To mitigate the windup effect, some anti-windup techniques have been developed, including the clamping algorithm anti-windup technique and the back-calculation anti-windup technique.

I. Integral Anti-Windup Technique

Integral clamping is a technique used in PID controllers to prevent integral windup by restricting the accumulation of the integral term to predefined limits. When the controller's output reaches its saturation points (either maximum or minimum), the integral term is clamped to prevent further increase, avoiding excessive overshoot or long response times. Clamping ensures that the controller remains responsive and stable even in the face of long-term errors or system constraints by limiting the integral action. This technique aids in the maintenance of effective control while also avoiding the negative effects of integral windup.

II. Back-Calculation Anti-Windup Technique

The Back Calculation The anti-windup technique is a method for preventing integral windup in PID controllers that adjusts the integral term based on the difference between the controller's output and the actual actuator position when the output becomes saturated. When the controller output exceeds its maximum or minimum limits, the integrator receives the difference between the desired and saturated outputs. This feedback effectively reduces the accumulated integral value, preventing it from increasing or decreasing too much. This allows the controller to quickly recover from saturation and resume normal operation, ensuring stable and responsive control even in the presence of prolonged error signals. This technique is especially useful in systems where the actuator has physical limitations, as it helps maintain control performance while avoiding the negative effects of windup.

8 Implementation

In this chapter, the design and implementation of the software for this system are examined and listed. During the design stage, several prototypes have been tried to adhere to meet the most requirements that are specified in the requirements chapter, and different theories that are discussed in the theory chapter were also prototyped and improved with many iterations, resulting in a system that is efficient and meets most of the requirements specified. The generated Doxygen document for the system implementation can be found at "https://pierre-thishan.github.io/BachelorThesis_Warnakulasooriya/".

The embedded system that is designed for this project is fully developed and designed using the programming language C. It consists of six parallel processes with equal priority running on two cores, allocated with a low but sufficient task stack size to ensure safety. Parallel processing has been achieved by utilising the Free-RTOS library from the esp32 SDK.

The mechanical setup, known as Eva, primarily consists of two main operations: moving and heating. An eight-state finite state machine (FSM) manages the moving operations, maintaining the system's state in one of the defined states at all times. Similarly, a two-state FSM controls the heating operations. In addition, there are several other processes responsible for monitoring the system, reading data from sensors, controlling motor drivers, and performing other essential functions. In the next section, the overall system will be examined. Table 19 below has listed the names of these parallel processes and their operations, along with which core on the MCU it is running on. The MCU has two cores named "0" and "1".

Parallel Process Name	**Operations**	**Running Core**
Task1Code	• Executes PID motor velocity calculations. • Setting the motor speed. • Motor moving • Enabling and disabling the motor.	0
Task2Code	• Process and parsing of input commands from serial. • Execute the state machine.	1
Task3Code	• Reporting system status and other sensor data to the server API through serial.	1

Task4Code	• Responsible for regulating insertion force throughout the set time period by setting the state to keep pressing when the maintain force function is toggled. • Checking the MCU internal temperature.	1
Task5Code	• Setting up the inner and outer traverse bounds with the magnetic encoder. • Temperature sensor reading. • Heating PID loop execution. • Heat state machine execution.	0
Task6Code	• Set a zero position when setting the inner traverse bound. • Read the magnetic encoder angle. • Control motor speed and direction when a traverse position is given to move to. • Read the pressure sensor value.	0

Table 19 : Parallel processes and their tasks in the system.

8.1 Overall System Architecture

In this chapter, the overall system architecture is introduced, followed by an exploration of the configurations within the system boundary.

Figure 12 provides a high-level overview of the overall system architecture, showing the interaction between the various components involved in the system. A detailed list of the primary components and their respective functions within the system is listed in Table 20.

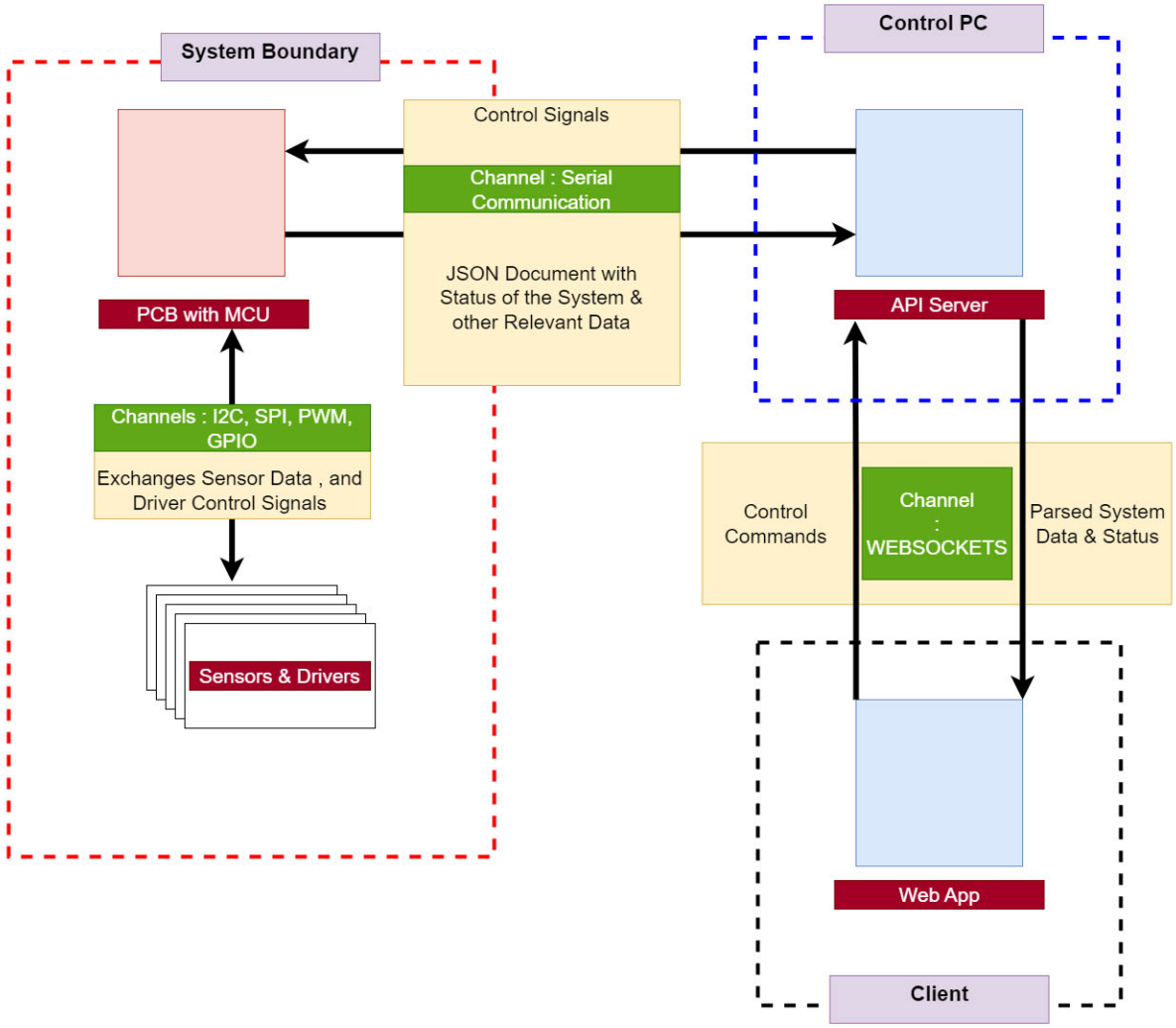

Figure 12 : High-level overview of system architecture.

Component	Description
System Boundary	The red dashed line marks the system boundary, containing all hardware and software components that are directly part of the embedded system. Inside this boundary, the PCB with the MCU (Microcontroller Unit) acts as the central processing unit for the system, coordinating the operations of connected sensors and drivers.
PCB with MCU	The PCB with MCU is the core component within the system boundary, managing communication with various sensors and drivers through different communication channels such as I2C, SPI, PWM, and GPIO. These channels facilitate the exchange of sensor data and the transmission of control signals to drivers, ensuring the embedded system operates correctly.
Sensors & Drivers	Connected to the PCB are multiple sensors and drivers, which are responsible for gathering data from the environment and executing actions based on commands received from the MCU. The system continuously processes data from these sensors, which is critical for maintaining control over the system's operations.
Control Signals & Serial Communication	The system exchanges Control Signals with an external Control PC. This communication is handled through a Serial Communication Channel, where data is transmitted in the form of a JSON document containing the status of the system and other relevant data.
Control PC	Outside the system boundary, we have the control PC housed within a blue dashed box. The control PC runs an API server that acts as an intermediary, receiving system status updates and sending back control commands. The API Server communicates with the embedded system using the serial communication channel and forwards the data to a web app via WebSockets.
Web App	The Web App, situated on the control PC, serves as the user interface where parsed system data and status information are presented to the user. It also allows the user to send control commands back to the system. The communication between the Web app and the API server is managed through a WebSocket channel, which enables real-time data exchange.
Client	Finally, the client interacts with the system through the web app. The client, who is typically a lab engineer, sends commands and receives system status updates, which allows for remote management and control of the embedded system.

Table 20 : Components of the system architecture and their interactions.

8.2 Mechanical Assembly of the System

This section describes the mechanical assembly of the system, with a focus on the setup known as “Eva”. This mechanical framework is the physical structure in which the control software system interacts, carrying out precise movements and operations.

The system’s carefully designed control algorithms and the integration of hardware and software within this assembly enable the smooth execution of tasks ranging from movement to temperature control. Figure 13 and Figure 14 illustrate the mechanical assembly of Eva and name the relevant components.

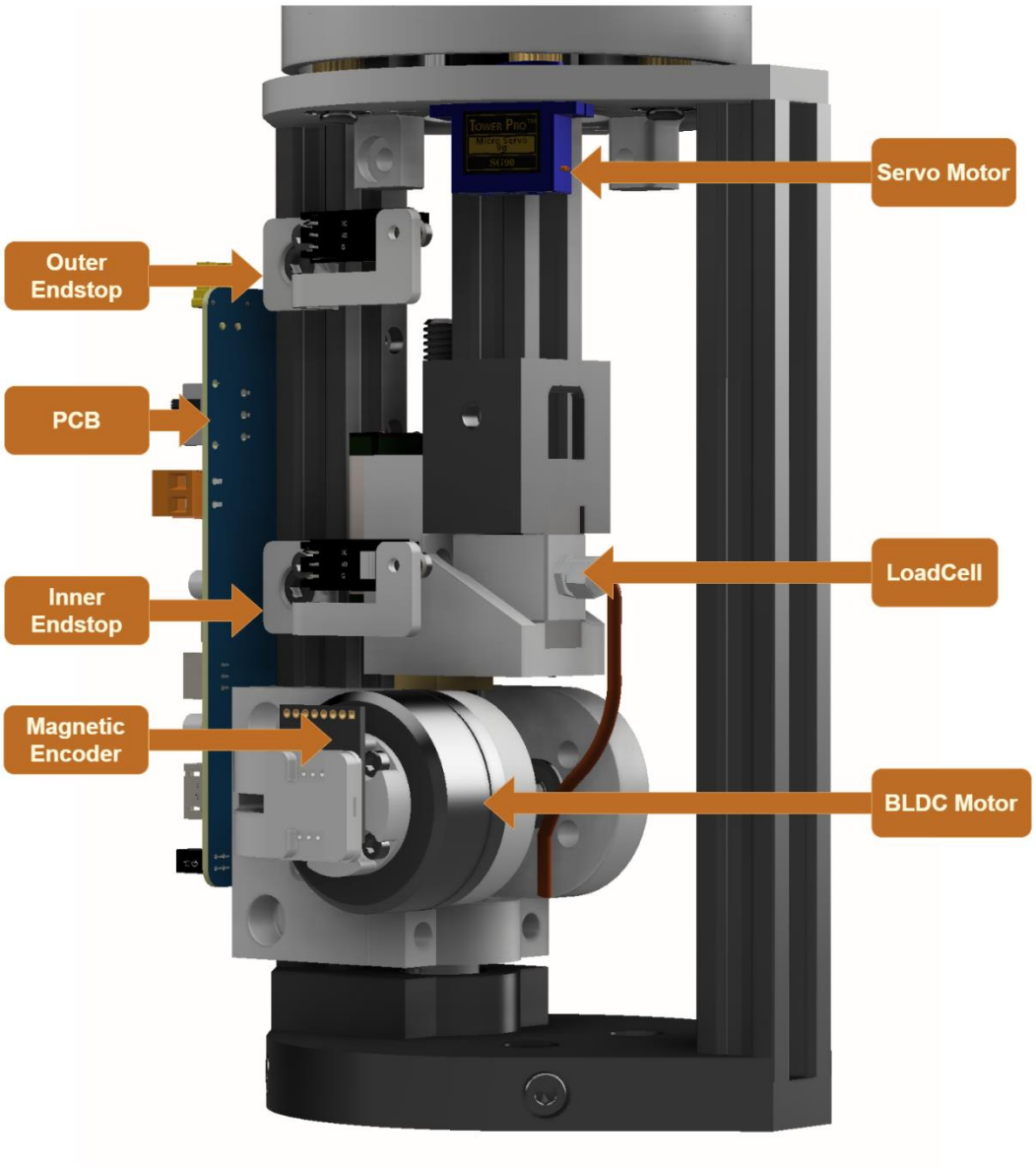

Figure 13 : Eva flipped front side view.

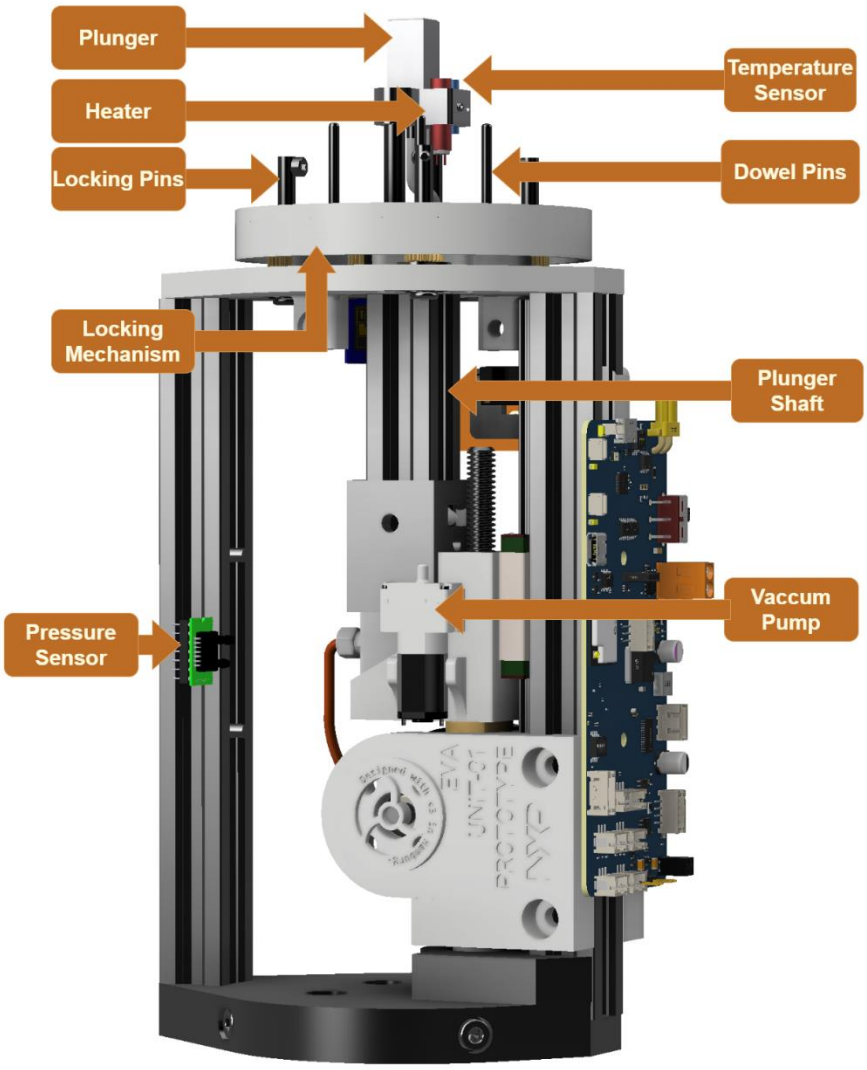

Figure 14 : Eva flipped back side view.

The ESP32-S3 MCU, which is on the PCB in Eva's mechanical assembly (Figure 13 and Figure 14), is at the centre of the system architecture shown in Figure 15. This MCU is the main processing unit and communicates with different sensors, actuators, and outside systems. Table 21 provides a detailed description of each component and its interaction with the entire system.

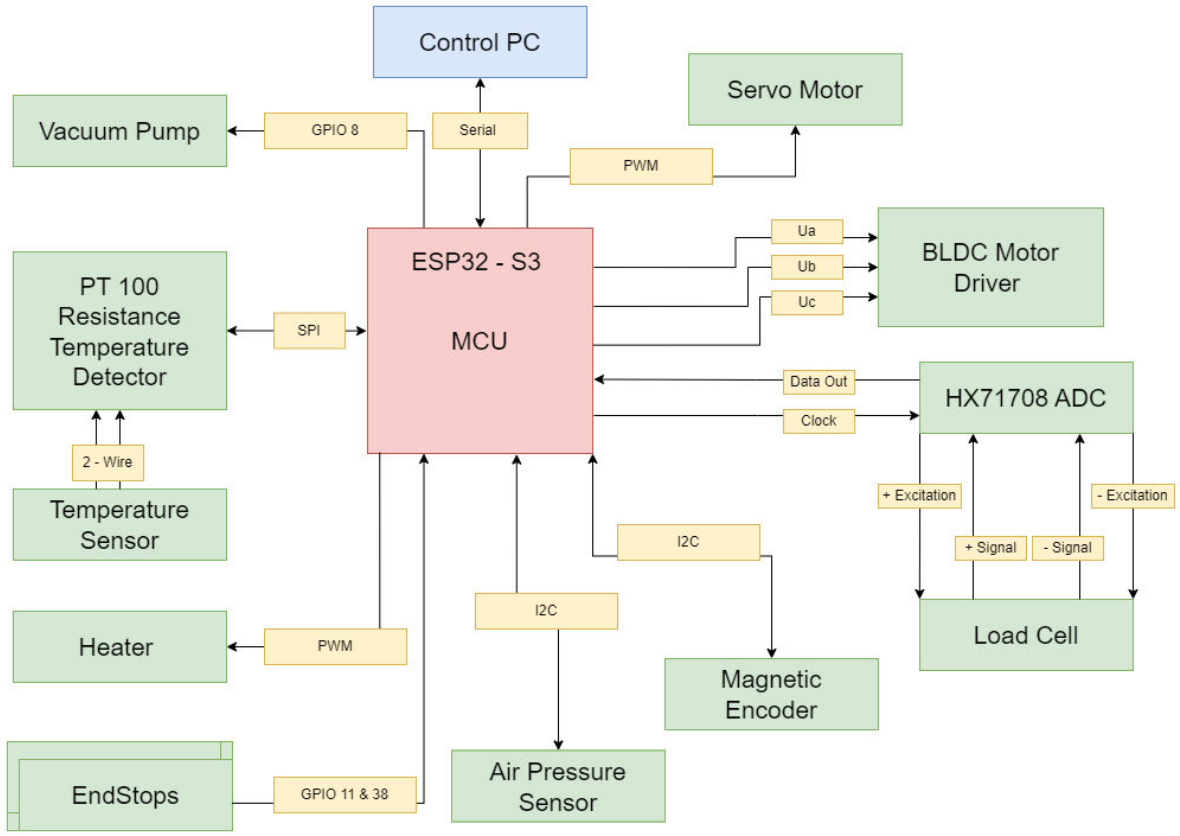

Figure 15 System block diagram.

Component	Description
ESP32-S3 MCU	The central processing unit is responsible for managing and coordinating all operations within the system, interfacing with various peripherals using GPIO, I2C, SPI, PWM, and serial communication protocols.
Control PC	Interacts with the ESP32-S3 MCU via serial connection, sending and receiving data such as commands and status updates.
Vacuum Pump	Controlled by the ESP32-S3 MCU through GPIO 8, it can be activated or deactivated based on operational requirements. Used to create suction to pick up ICs through the plunger.
Servo Motor	Controlled using a PWM signal from the ESP32-S3 MCU, used for the operation of the locking mechanism to lock Eva into the docking adapter with the test socket.
BLDC Motor Driver	Receives control signals via three-phase connections (Ua, Ub, and Uc) from the ESP32-S3 MCU, driving a brushless DC motor for high-efficiency applications.
BLDC Motor	The signals from the motor driver are controlled to generate the insertion force and traverse the plunger within the movement range.

HX71708 ADC and Load Cell	The ADC interfaces with the load cell, which is rated to measure weights up to 20 kg. The load cell measures the force applied to the socket for insertion force control, utilising MEMS strain gauge technology to measure the force accurately, and the ADC converts the analogue signals from the load cell into digital data for processing by the MCU.
Magnetic Encoder	Connected to the ESP32-S3 MCU via the I2C interface, providing precise position feedback and traversing the plunger to a set position.
Air Pressure Sensor	It measures the pressure from the outlet of the vacuum pump, is connected via I2C, sends data to the MCU for processing, and is used for IC pick-up confirmation.
PT100 Resistance Temperature Detector (RTD)	It measures temperature using a 2-wire connection interfaced with the MCU through SPI, providing accurate thermal data for process control.
Temperature Sensor	Connected to the PT100 RTD via a 2-wire method, assisting in monitoring temperature changes, especially in conjunction with the heater.
Heater	Controlled by the ESP32-S3 MCU through PWM, adjusting output based on temperature readings to maintain stable thermal conditions.
End Stops	Connected via GPIO 11 & 38, used to detect mechanical limits of movement, ensuring safety and precision in system operations. The optical endstop is a sensor that detects the presence or position of an object by using light, typically emitting a beam that is interrupted or reflected, signalling the object's position or limit of movement.

Table 21 : Components of Eva and their description.

8.3 State Diagrams

The movement of the plunger at any given time is managed by an eight-state finite state machine (FSM). In Table 22 and Table 23, the states are listed and described briefly, along with the corresponding state diagram figure. The system state transitions are executed within a parallel task called ‘Task2code’. The task first processes input commands received through the serial interface, parsing them to determine if a new state is specified, and if a new state is not set by the command, the system will determine the next state based on the current state and predefined conditions, allowing the appropriate state transition to occur.

State	Description	Figure Number
TRAVERSE	The plunger moves to a specified position, ensuring it is within tolerance. The system checks for any overload conditions or endstop triggers during this state.	Figure 22
HOMING	The plunger returns to a predefined home position, which is where the inner endstop is triggered by the extruded marker of the load cell holder.	Figure 18
STOP	The system halts all movement, disabling the motor and awaiting further commands.	Figure 17
IDLE	The system remains in a standby state, with no active movements, awaiting the next command or transition based on external inputs.	Figure 16
PRESS	The plunger applies a force to the target, continuing until a specified force or position is reached, while monitoring for overload and endstop conditions.	Figure 19
KEEP_PRESSING	The plunger maintains a continuous pressing force, often used to ensure sustained contact or pressure, until specific conditions are met.	Figure 23
PICKING	The plunger moves downward with the vacuum pump activated, enabling it to suction an IC. The operation is considered complete when feedback from the pressure sensor connected to the vacuum pump’s outlet confirms that the IC has been successfully grasped.	Figure 21
PLACING	The plunger moves to place an IC in a designated location (socket or tray), releasing it by turning off the vacuum pump with controlled placing force and returning to a safe position after the task is complete.	Figure 20

Table 22 : State descriptions of the FSM.

Below are the ASMD diagrams of the eight states mentioned above in Table 22 above.

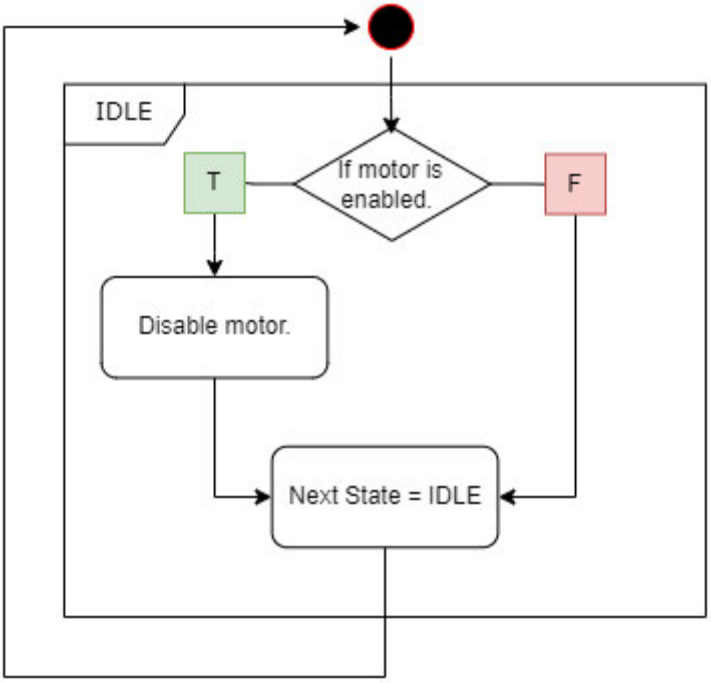

Figure 16 : ASMD of state IDLE

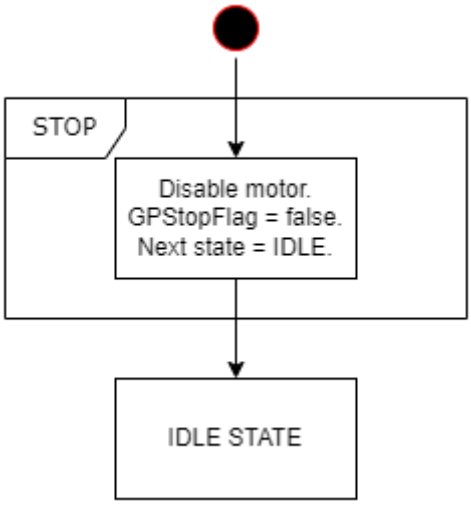

Figure 17 : ASMD of state STOP

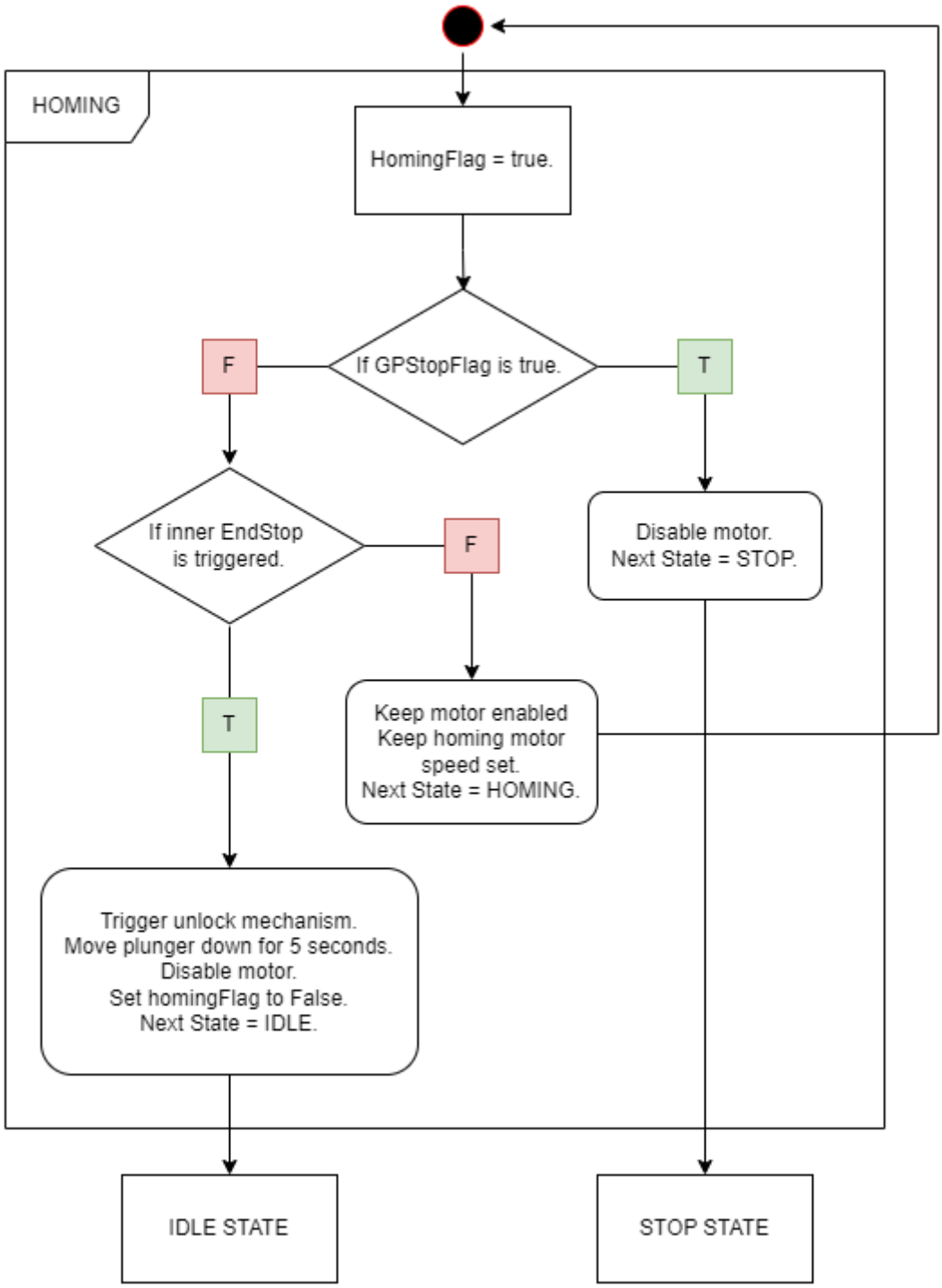

Figure 18 : ASMD of state HOMING

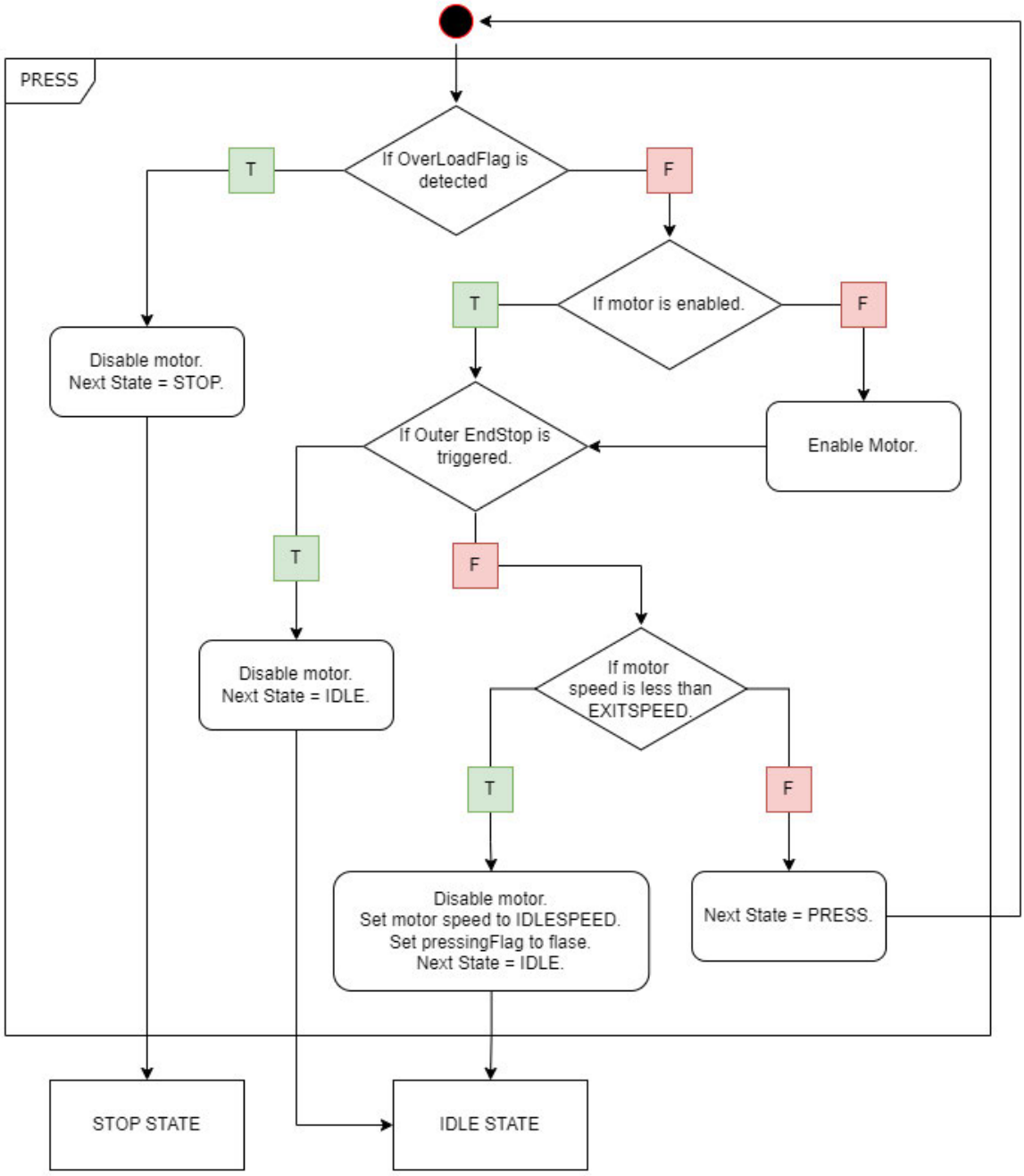

Figure 19 : ASMD of state PRESS

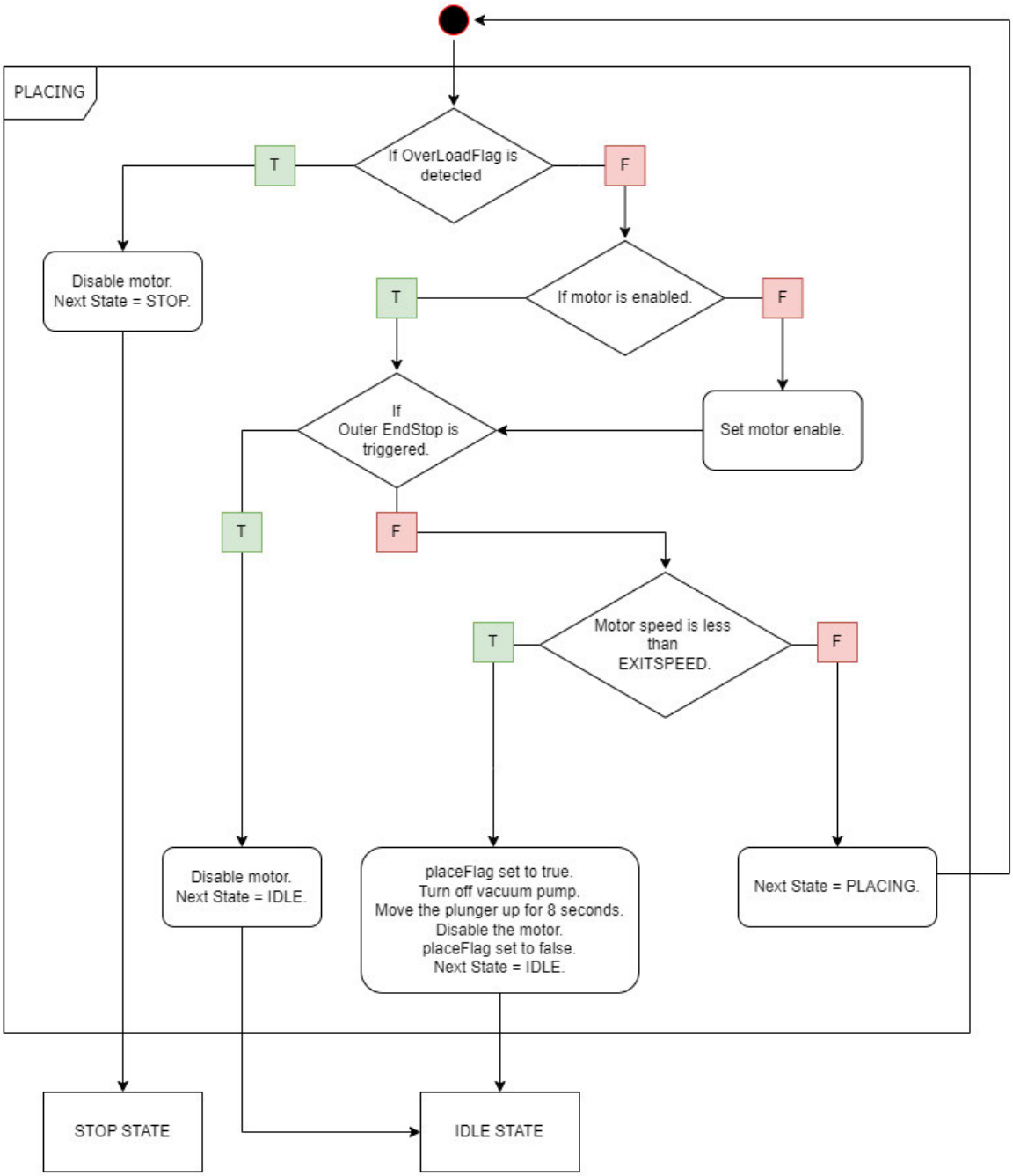

Figure 20 : ASMD of state PLACING

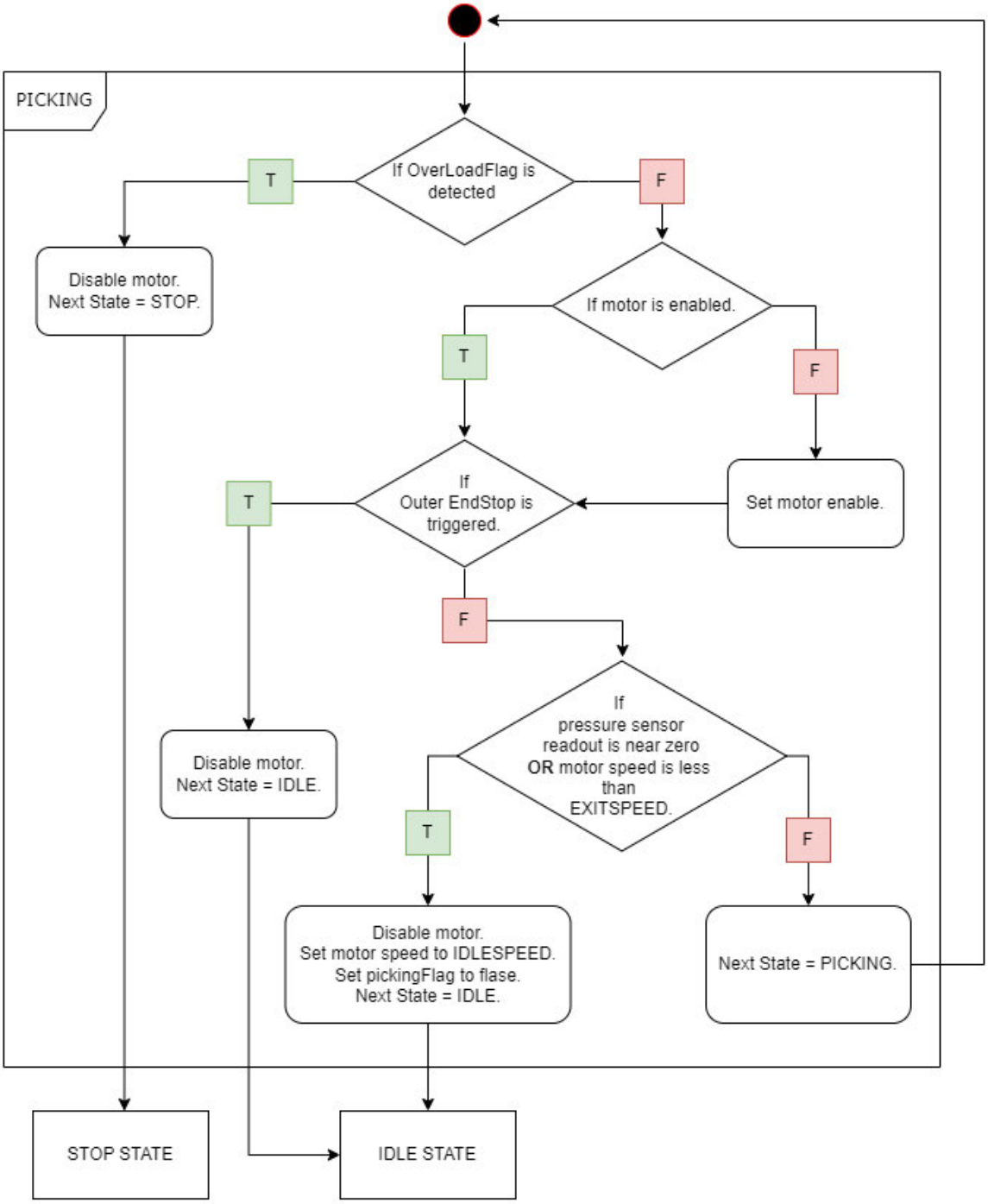

Figure 21 : ASMD of state PICKING

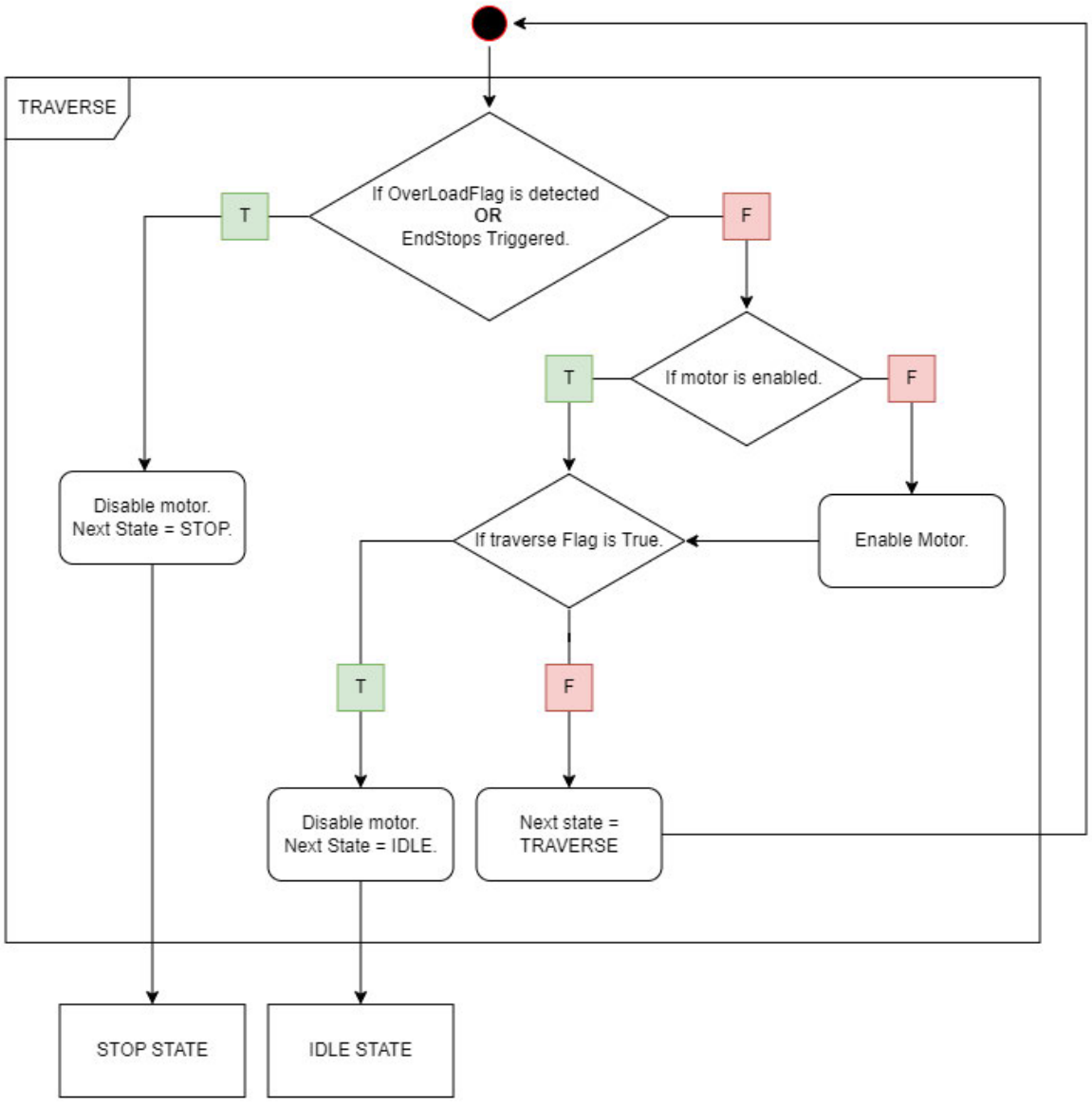

Figure 22 : ASMD of state TRAVERSE

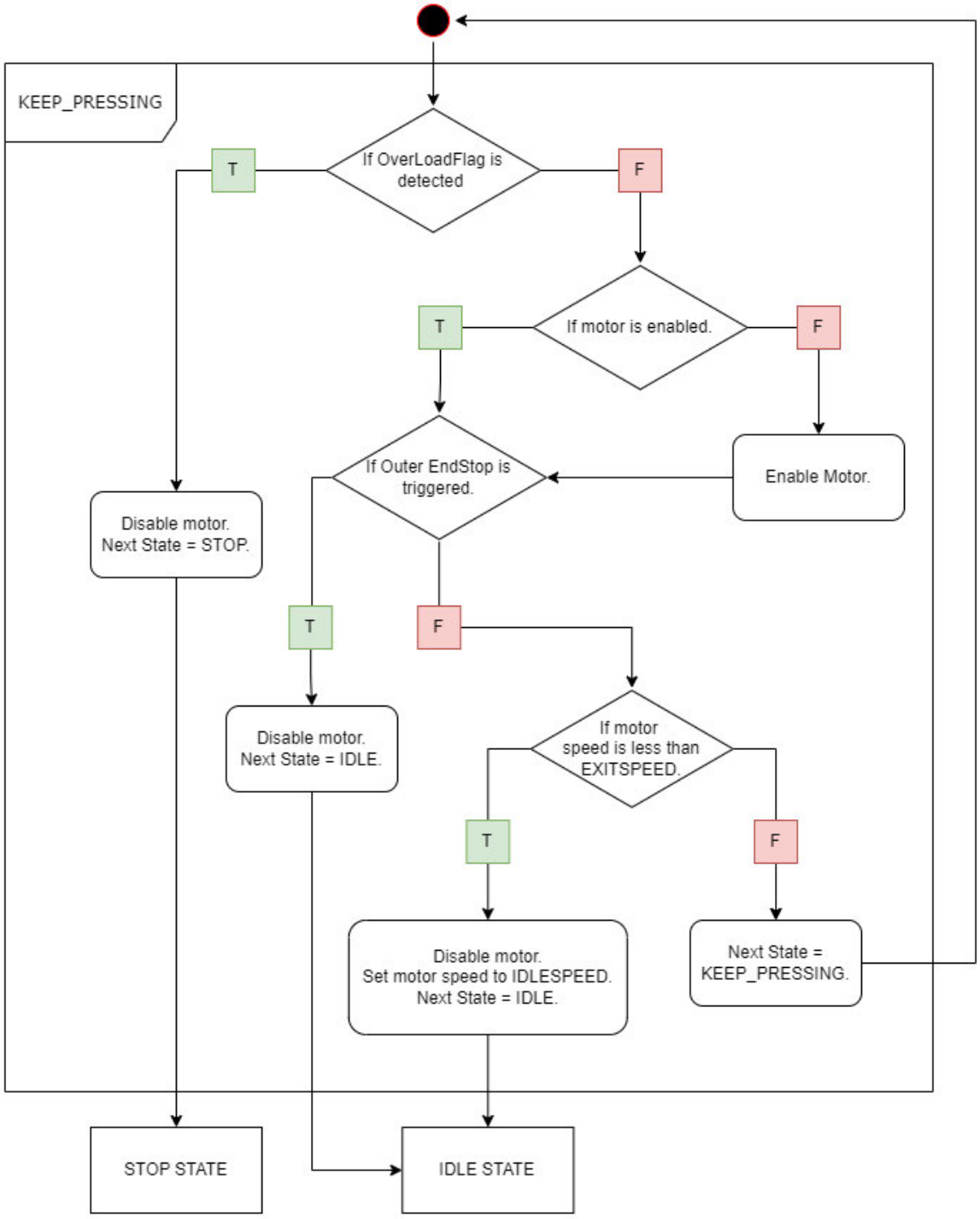

Figure 23 : ASMD of state KEEP_PRESSING

In Figure 24 a simplified version of the FSM is illustrated for convenience and for a quick general overview. As mentioned earlier, parsing and setting the next state based on an input command is not covered in Figure 24. The state will only transition to a state other than IDLE or STOP if the next state is explicitly set via an input command. For example, if the user enters the ‘press’ command, the input command parser will set the next state to PRESS, and the FSM will transition into the PRESS state.

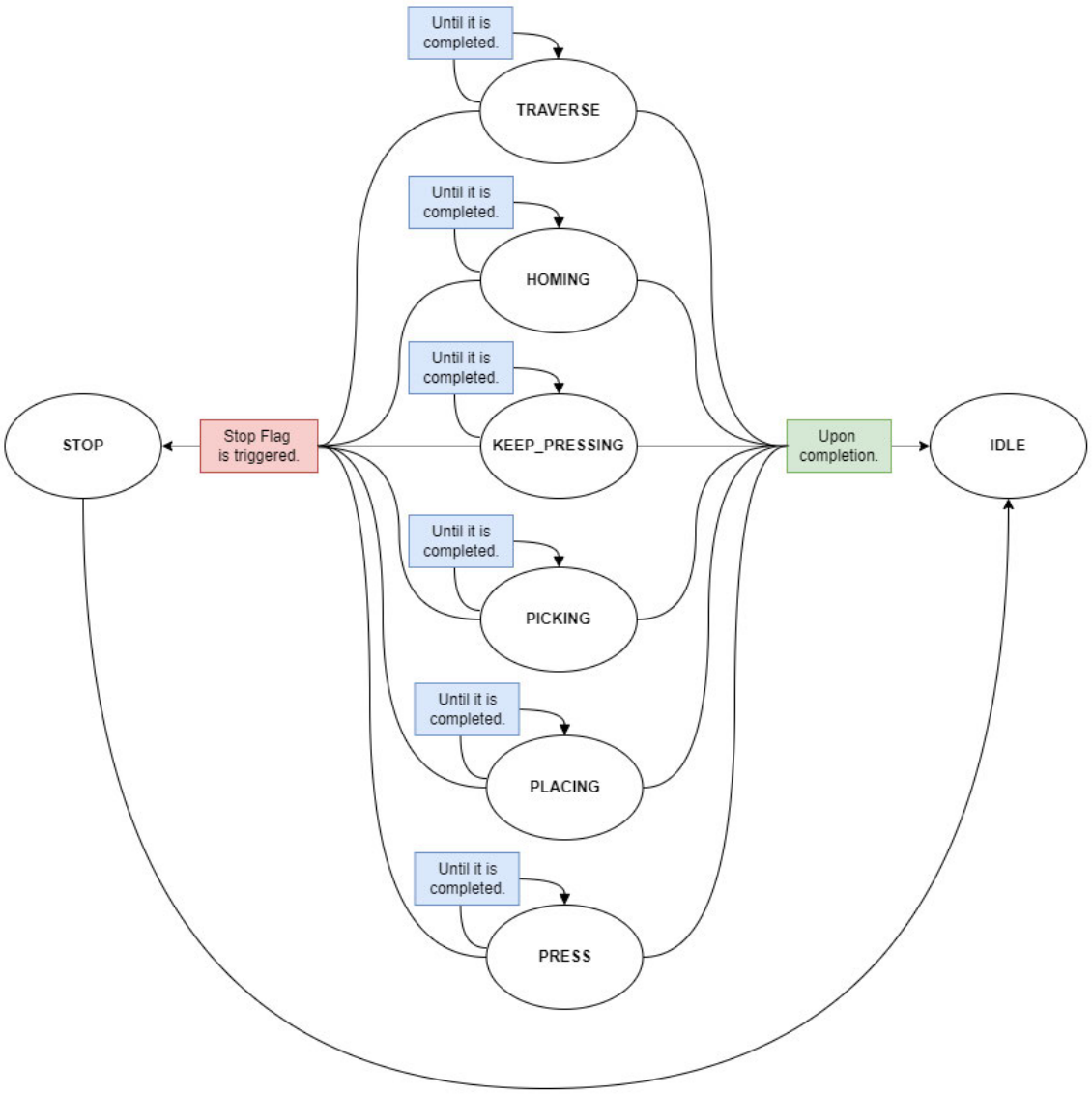

Figure 24 : Simplified FSM for overview.

8.4 PID Design

Utilising the theories and methods explored in the theory section, the PID control loops necessary for the system were developed and implemented to meet the specific requirements. This section will focus on the design and implementation of these control loops, detailing how each component contributes to achieving the desired system performance.

8.4.1 Motor Control PID

Here, the PID control loop created for the motor control, which controls the movement of the plunger, and the insertion force of the IC into the socket are examined.

In this system, when it comes to controlling the motor, the objective is to use motor motion to control the insertion force, which is read by a load cell. For the control of BLDC Motors, there exists an already well-implemented and reviewed library for controlling the motor driver called simpleton, which was utilised for the control of the motor in this system. The library offers two types of control modes, namely “closed-loop control” and “open-loop control”. Often, open-loop control is used for simple tasks where precision is not required, whereas closed-loop control is used for precise, sensitive operations.

Due to the system’s complexity and the required operation, the available closed-loop options, which are “torque control”, “velocity control” and “position control” would not be effective and would not be able to perform the required task. Therefore, the very well-implemented close control loops could not be used in this system, even though they consist of already well-tuned PID controllers. This restricts the system to the use of open-loop control for motor movement control. In this system, of the two available open-loop control methods, namely “velocity control’’ and “position control, the “velocity control’’ loop is utilised.

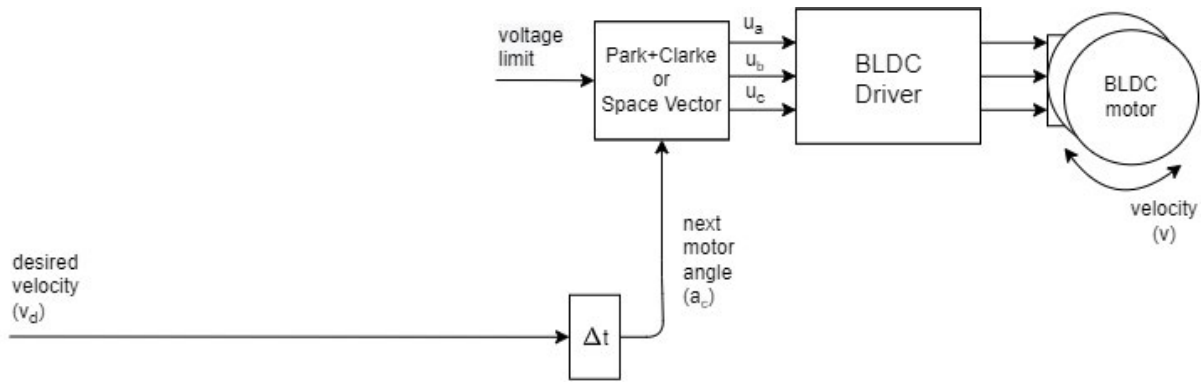

Figure 25 : Velocity open loop control block diagram [26].

In Figure 25, the block view of the velocity open-loop control system by simpleFOC library is illustrated. Using only the velocity control loop, the system would not be able to effectively control the insertion force. To address this challenge, a custom PID controller was implemented, utilising feedback from the load cell sensor to regulate the motor's velocity, which results in a closed-loop control system that adheres to the required precision and accuracy.

The block diagram in Figure 26 illustrates the components and flow of the PID controller, and in Table 23, a detailed description of each component's role is listed. In Figure 26, the dashed red line box shows the border of the open-loop control used by the simpleFOC library within the closed-loop control loop of the motor.

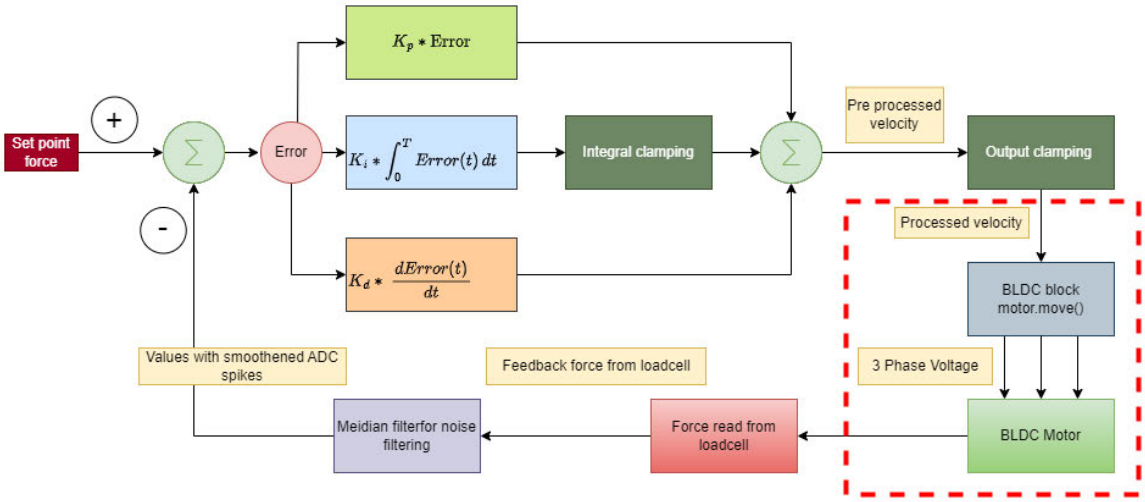

Figure 26 : Custom motor PID controller block diagram

.

Component	Description
Set Point Force	The target force value that the system aims to achieve serves as the reference input for the PID control loop.
Error Calculation	The difference between the set point force and the actual force measured by the load cell is used to determine the necessary adjustments.
Proportional Gain	The proportional term of the PID controller produces a correction proportional to the current error. This helps reduce the error quickly.
Integral Gain	The integral term accumulates the error over time, addressing any steady-state error to ensure the system reaches and maintains the desired force.
Derivative Gain	The derivative term predicts future error based on its rate of change, helping to dampen the response and reduce overshoot.
Integral Clamping	A mechanism that limits the integral term to prevent excessive buildup, which could lead to instability or overshoot in the system.
Output Clamping	A mechanism that restricts the total output of the PID controller, ensuring the motor operates within safe and effective limits.
Pre-Processed Velocity	The velocity output calculated by the PID controller before clamping represents the motor's required speed to achieve the set point force.
Processed Velocity	The final velocity command is sent to the motor after applying clamping, ensuring the velocity remains within operational limits.
BLDC Motor Driver	The function that applies the processed velocity to the Brushless DC (BLDC) motor controls its 3-phase voltage to achieve precise force application.
BLDC Motor	The Brushless DC (BLDC) motor applies the force as directed by the PID controller, using 3-phase voltage signals to control speed and torque.
Force read from load cell	The actual force is measured by the load cell, providing feedback to the PID controller for continuous adjustment of the motor's output.
Median Filter for Noise Filtering	A filter is applied to the load cell's output to remove noise or spikes, ensuring accurate and stable force measurement.
Values with smoothed ADC spikes	The filtered and smoothed force values used for error calculation prevent transient noise from destabilising the control loop.

Table 23 : Components of motor PID controller.

8.4.2 Temperature Control PID

The PID control loop for the temperature control, which controls the heating of the plunger to heat up ICs, is explored here. Figure 27 shows the block diagram of the PID controller's components and flow, and Table 24 contains a detailed description of each component's roles.

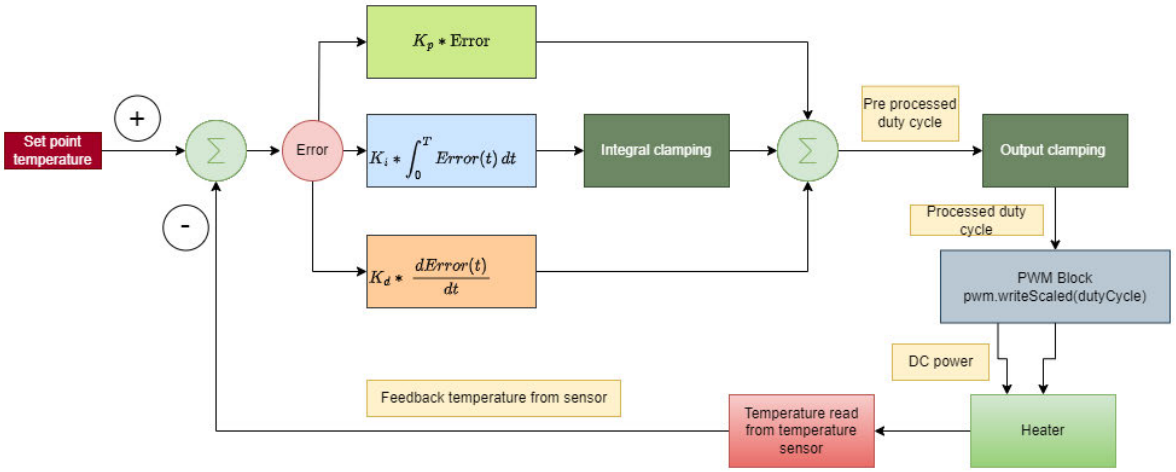

Figure 27 : Custom temperature PID controller block diagram.

Component	Description
Set Point Temperature	The desired temperature value that the system aims to maintain serves as the reference input for the PID control loop.
Error Calculation	The difference between the set point temperature and the actual temperature measured by the sensor. This error value is used to determine the necessary adjustments.
Proportional Gain (Kp * Error)	The proportional term of the PID controller, which adjusts the heater's power output in proportion to the current error, this helps quickly reduce the overall error.
Integral Gain (Ki * Integral of Error)	The integral term accumulates the error over time to eliminate steady-state error, ensuring the system reaches and maintains the desired temperature.
Derivative Gain (Kd * Derivative of Error)	The derivative term predicts future errors by evaluating the rate of change, helping to prevent overshoot and improve system stability.
Integral Clamping	A mechanism that limits the integral term to prevent excessive accumulation of error, which could otherwise lead to instability or overshoot.
Output Clamping	A mechanism that restricts the total output of the PID controller, ensuring the heater operates within safe and effective limits.

Pre-Processed Duty Cycle	The duty cycle output is calculated by the PID controller before clamping, representing the required power to achieve the set point temperature.
Processed Duty Cycle	The final duty cycle command sent to the PWM block after applying clamping, ensuring it remains within operational limits.
PWM Block	The function responsible for applying the processed duty cycle to the heater is modulating its power to maintain the desired temperature.
Heater	The device that generates heat based on the power delivered by the PWM signal, working to achieve the temperature set point specified by the PID controller.
Temperature Read from Sensor	The actual temperature measured by the sensor, providing feedback to the PID controller to continuously adjust the heater's output.
Feedback Temperature from Sensor	The filtered and accurate temperature feedback used for error calculation, ensuring the control loop responds correctly to changes in temperature.

Table 24 : Components of temperature PID controller.

8.5 System Flow

For a clearer understanding of the system's operation, this section includes the flow of events for an example test user case. The flow chart presented in Figure 28 shows the interactions between the various components and highlights the sequence of actions that occur within the system during this specific scenario. The example test case involves a lab engineer using the web app to send a command to insert the IC into the socket with approximately 68 N of force. This command is received by the control system as 'press 7000' after being parsed by the server API.

For easier comprehension, some additional information about Figure 28 is listed in Table 25.

Additional Details
1. The force value, such as 7000, is measured in grams.
2. The unit of motor speed is rad/s, which is angular velocity.
3. The SW engineer can alter the value of LOADMAX, which is 8000 grams, if necessary.
4. EXITSPEED is set for 0.3 rad/s, which is used to exit the state when the PID calculated velocity is less than this, resulting in an insertion force that is very close to the target.
5. Motor speed is set to IDLESPEED, which is 0.6 rad/s, two times higher than the state exit condition speed, for proper functionality of the states and system to avoid exiting the state upon entering when in the KEEP_PRESSING state.

Table 25 : Additional information for the test case illustrated in the flowchart in Figure 28.

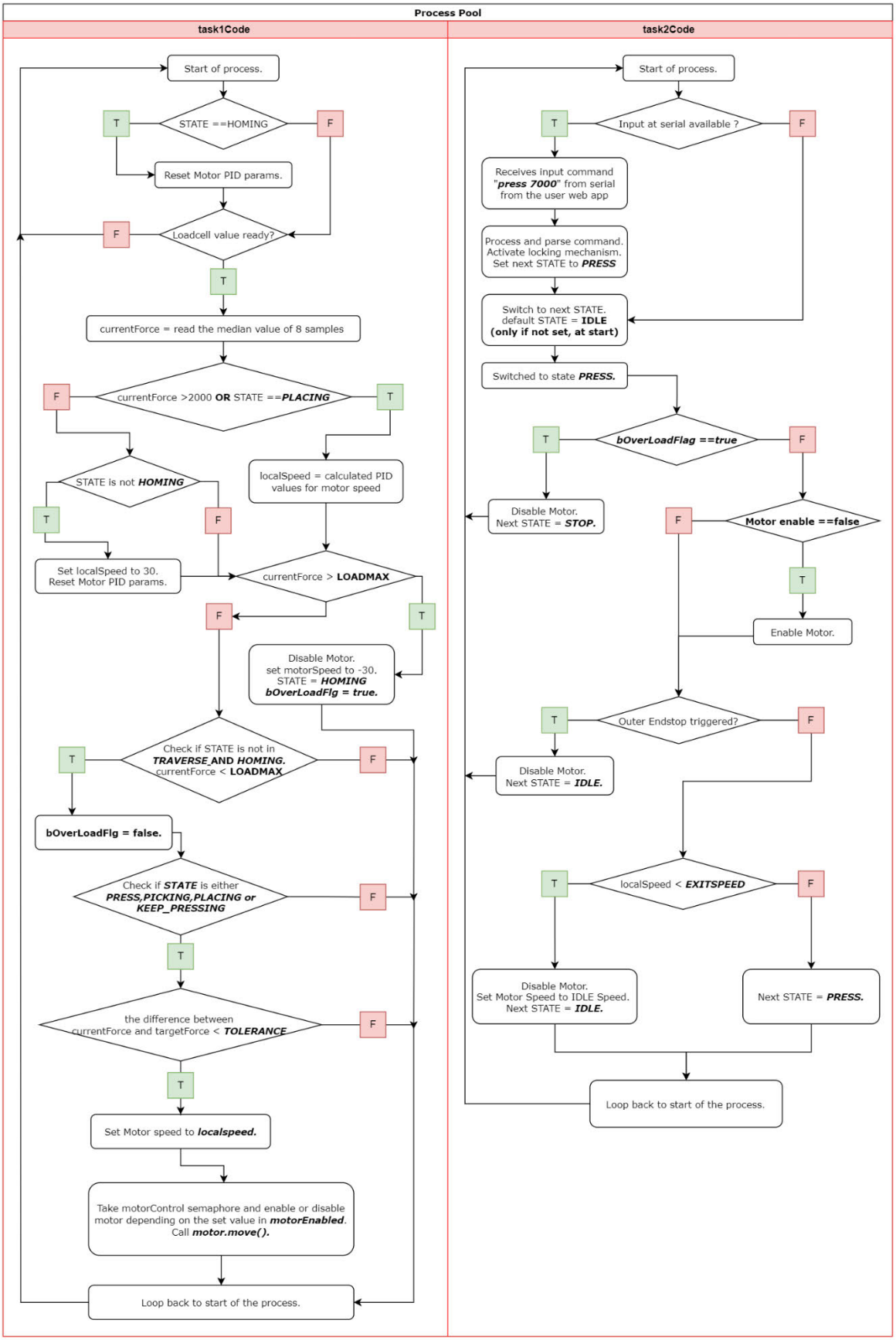

Figure 28: Flow chart of test case "press 7000".

8.6 Analysis of Challenges in Implementation

During the implementation of the system, several challenges were identified, and a few of them are listed and examined in this section.

8.6.1 ADC Spikes Analysis

During the implementation and testing phases of the system, sudden high spikes were occasionally sampled from the ADC, leading to system halts due to incorrect high force value readouts. In the lab environment, which consists of various tests and equipment for radar IC validation, ADC spikes could occur due to several factors. These potential causes are listed in Table 26 below.

Possible Cause	Description
Electromagnetic Interference (EMI)	High-frequency radar signals and other equipment generate EMI, introducing noise into the ADC input.
Power Supply Fluctuations	Variations in power supply voltage cause instability, leading to irregular spikes in ADC readings.
Ground Loops	Improper grounding or multiple grounding points introduce noise, causing inaccurate ADC spikes.
Crosstalk Between Signals	Interference from adjacent signal lines, particularly high-speed signals, results in ADC spikes.
Impedance Mismatch	Mismatch between source and ADC input impedance causes reflections, leading to signal distortions.
Temperature Variations	Rapid temperature changes affect ADC performance, causing temporary spikes.
Improper Shielding	Lack of proper shielding allows external noise to couple into the ADC input, causing high spikes.
High-Speed Switching	Rapid switching of radar signals or digital circuits induces noise into the analog signal path.
Sampling Rate Issues	Unsynchronized sampling rate with the signal of interest causes aliasing or under-sampling spikes.
PCB Layout Issues	Poor PCB layout practices lead to noise coupling into the ADC input, resulting in high spikes.

Table 26 : Possible causes for ADC spikes.

To examine this anomaly several tests were done under different environmental conditions to develop a solution.

I. Test in a Faraday Cage

A Faraday cage is an enclosure made of conductive material that blocks external electric fields and electromagnetic radiation, creating an isolated environment free from electromagnetic interference (EMI).

This allowed for a more accurate assessment of the system's performance by ensuring that any anomalies, such as ADC spikes, are not influenced by external electromagnetic noise but are instead inherent to the system itself. In the Faraday Cage available on NXP premises, the unfiltered, load cell reading values from the ADC were recorded for around 12 minutes and then compared with the values for the same time period in the radar validation lab environment.

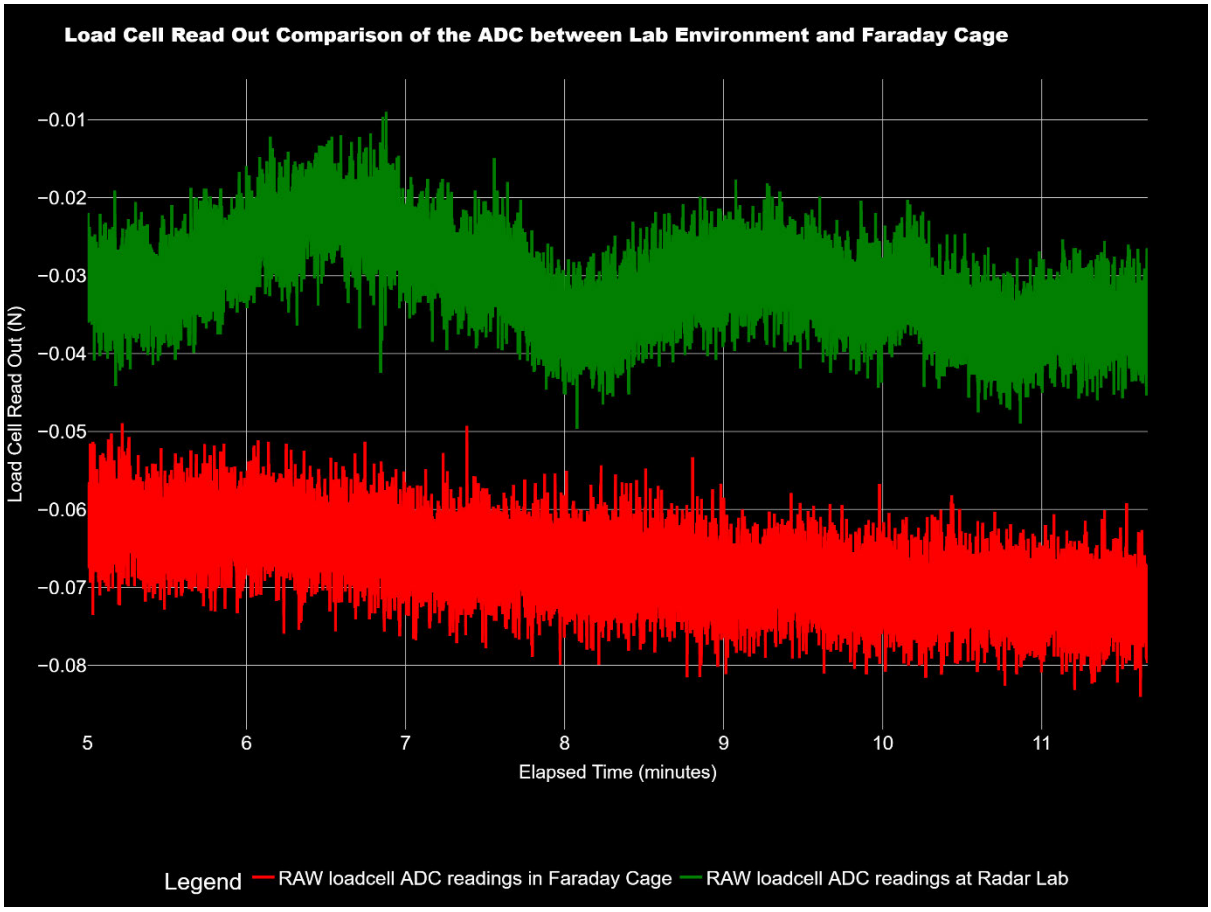

Figure 29 : ADC unfiltered values comparison between Lab and Faraday Cage

In Figure 29, it is observed that the readings taken in the Faraday cage have almost no significant relative spikes compared to the readings taken in the lab environment. Also, the average absolute deviation from the mean of the readings in the Faraday cage and lab is calculated to be respectively 0.0039 N and 0.0045 N, which is almost the same value.

II. High ADC Spike Readings

During another test of load cell data readouts, very high spikes were captured, which are denoted in Figure 30 below. These random spikes are significantly higher in amplitude and would cause the control system to malfunction.

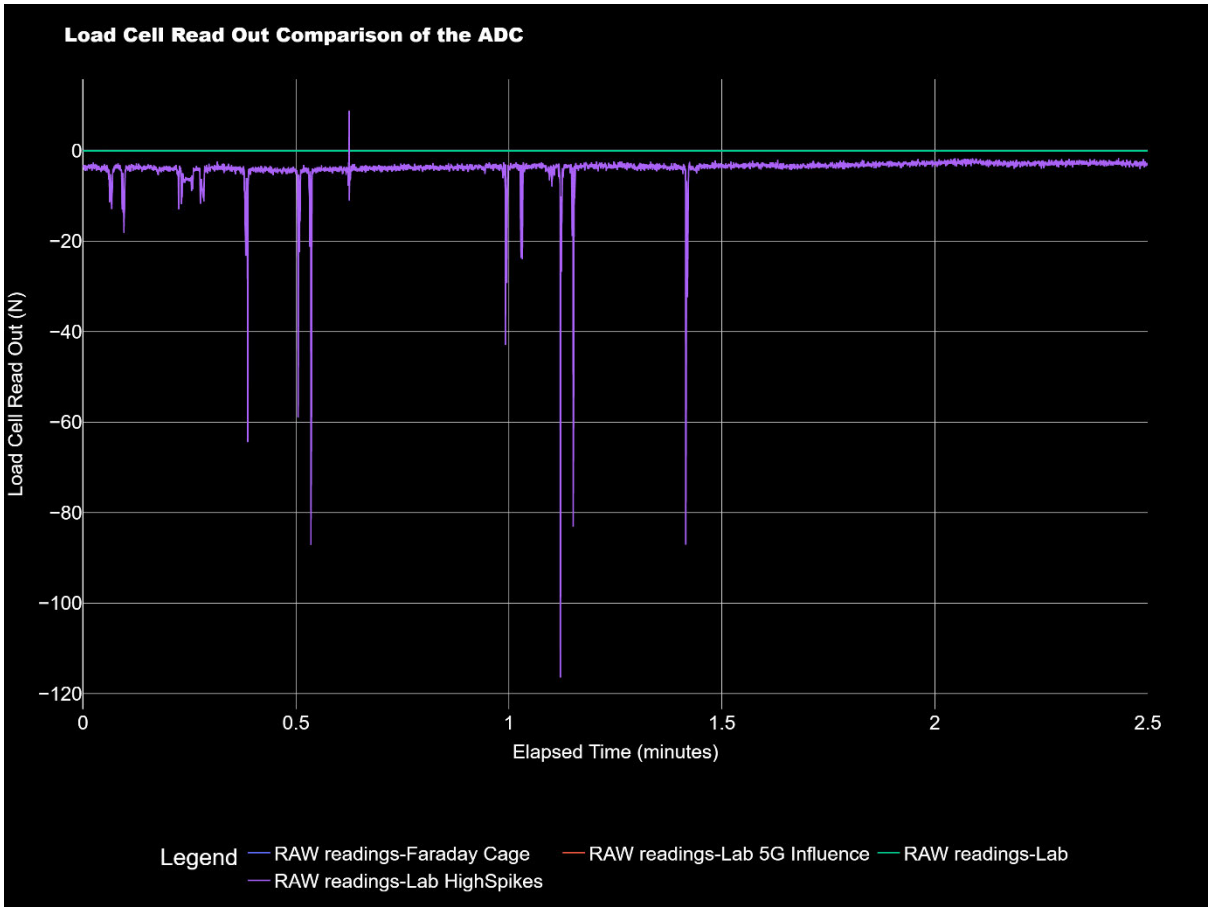

Figure 30: Very high ADC spike readings

III. 5G Influence from Mobile Phone

Even though mobile phone usage within the lab area is restricted, a test was conducted to observe the influence of the 5G signal on the raw ADC values by downloading a large file on a mobile with 5G connectivity and having it in close proximity to the PCB. Figure 31 displays relatively high spikes and heightened noise levels.

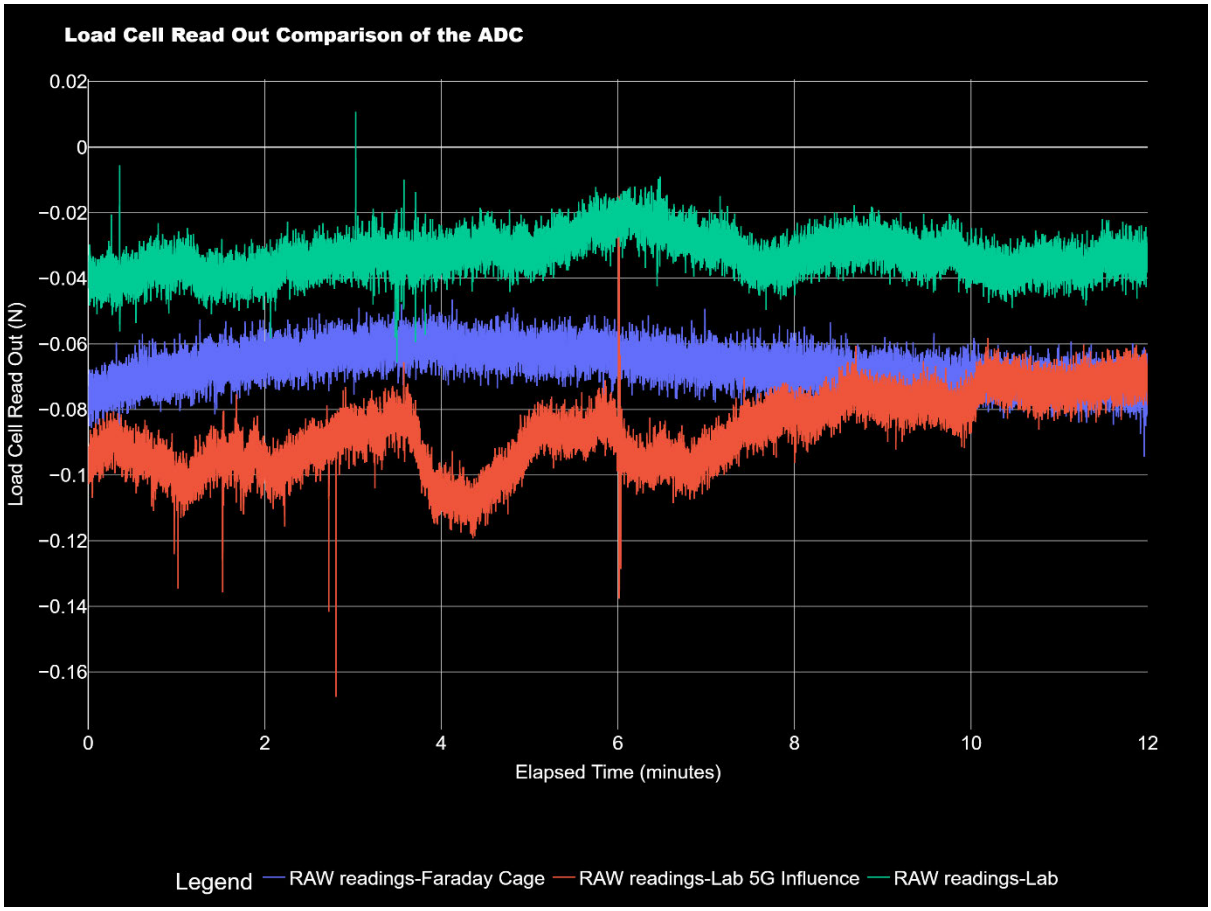

Figure 31 : Comparison of RAW ADC readouts between Faraday cage, lab, and 5G influence.

IV. Median Filtering of ADC Load Cell Values

To minimise the effect of this influence from the SW side, a median filter with a suitable window size of seven was implemented, which was able to overcome this sudden spike and also minimise the noise amplitude around a stable value. This could be observed in Figure 32 below.

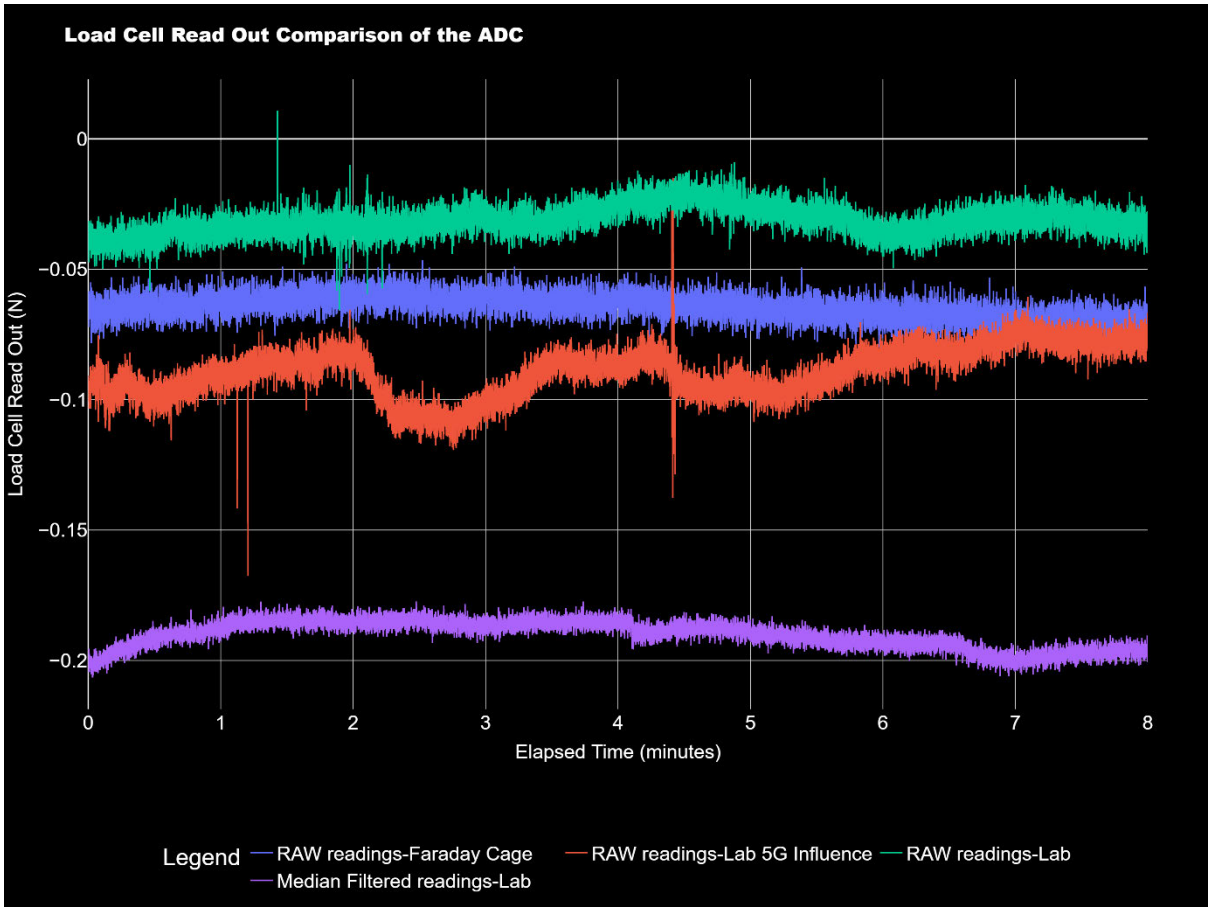

Figure 32 : Load cell readings comparison with Median filtering.

8.6.2 Motor PID Tuning Analysis

Although following the Ziegler-Nichols method for the tuning of parameters for the temperature-controlled motor worked smoothly, this was not the case when it came to tuning the PID parameters of the motor.

For the tests done on the insertion force of the plunger, Eva was disconnected from the Cobot and was kept on the socket test board lock in adapter, where when the plunger is pressing on the socket board, the assembly (Eva) travels up a bit until the locking mechanism between the test board adaptor and the locking pins is engaged. The first flat area in Figure 33, between 0.2 and 0.4 minutes, illustrates this. This could be observed in the other figures, where the insertion force is plotted.

In an insertion force operation involving the plunger and an IC, the plunger undergoes two distinct phases. The first phase occurs when the plunger, which holds the IC, descends before making contact with the socket board. During this time, the error signal between the set point

and the actual force remains constant, leading to a significant accumulation of error in the integral term. This accumulation can cause the controller output to saturate as the integral action continues to increase despite the lack of actual force feedback. In Figure 33 and Figure 34, it is observed that at around 26 N, the plot starts to stay relatively constant for some time and then again increases. This is because the plug lifts itself until the locking mechanism is tightly fitted with the board mounting plate, which is basically the exact force or weight of the Eva mechanical assembly.

As shown in Figure 33, when using the Ziegler-Nichols method for parameter tuning, it resulted in a too aggressive system that took too much time to settle and oscillate around the target force, which led to utilising a less aggressive tuning method.

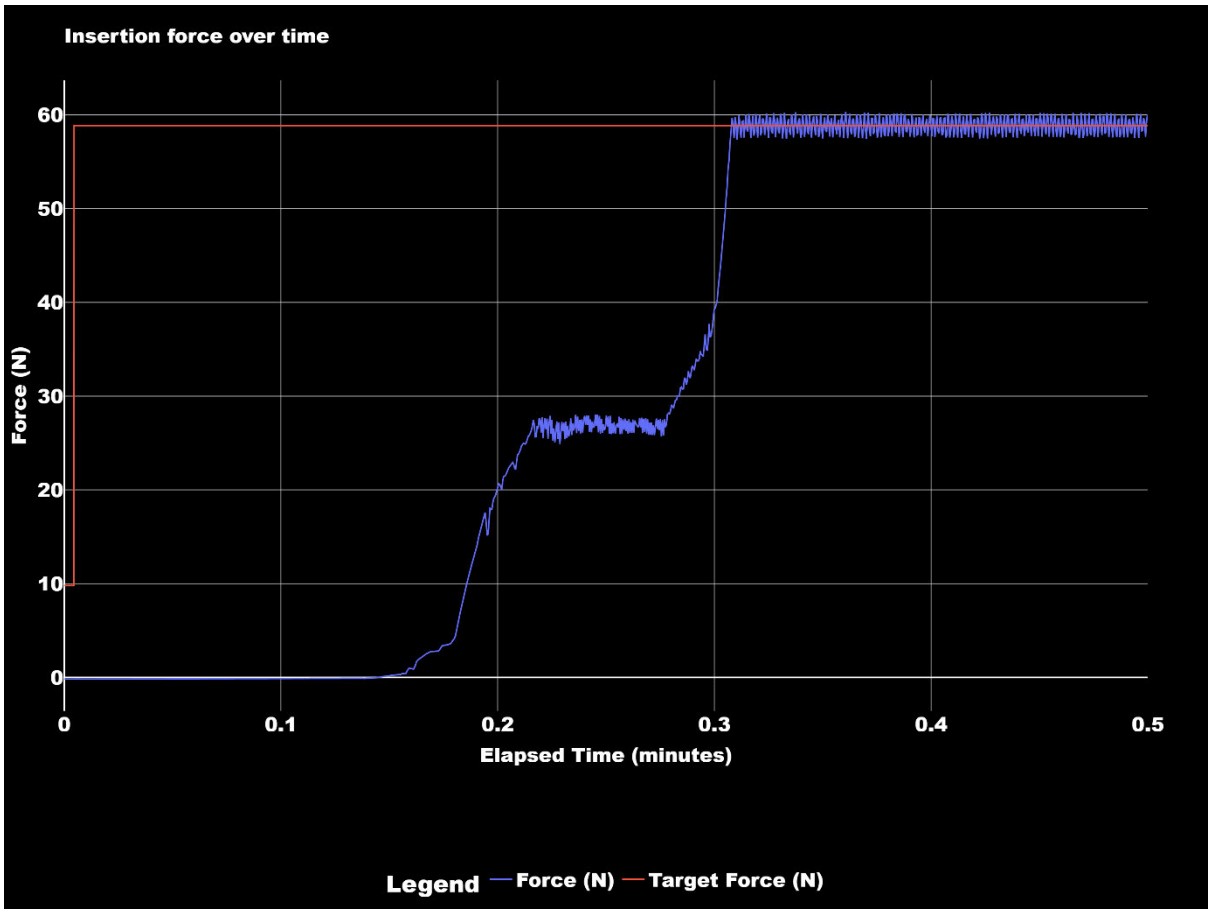

Figure 33: Unstable system with Ziegler-Nichol's method.

The Tyreus-Luyben tuning method, which resulted in a less aggressive system, was utilised, and for further fine-tuning, the parameters were manually adjusted for another operation. In Figure 34, the comparison between the Ziegler-Nichols method and the combination of

Tyrus-Luyben with manual tuning methods is illustrated. It is observed that the system settled relatively faster and did not oscillate for a longer period.

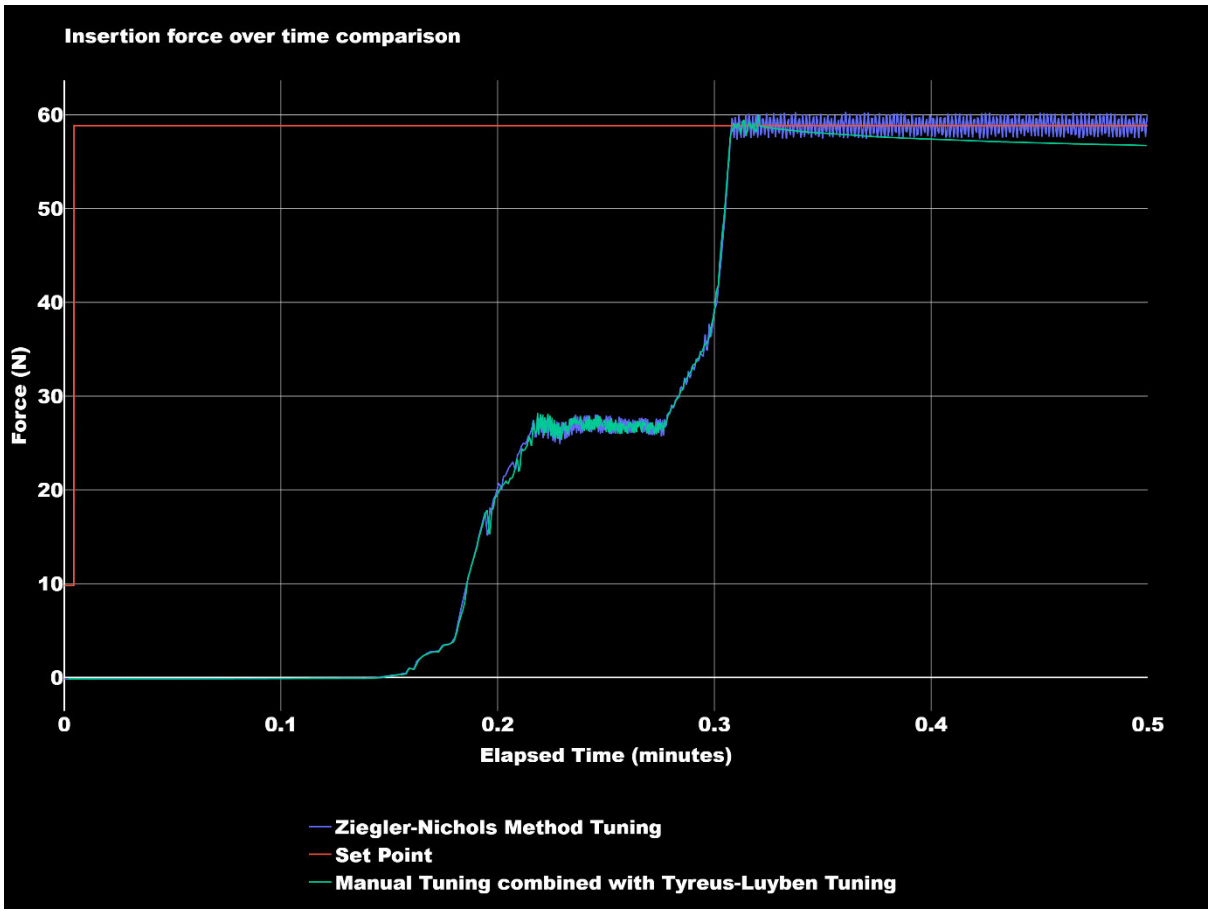

Figure 34 : Comparison between Ziegler-Nichol's method and the manually tuned Tyreus-Luyben method for system stability.

8.6.3 Sensor Libraries

While there were well-implemented and community-supported libraries available for most sensors and drivers, the two i2c sensors (magnetic encoder and pressure sensor) lacked such libraries, necessitating the creation of a custom, suitable library, which is called "i2cSensorLib". Here methods such as "exponential backoff" were utilised to overcome some transmission bugs that occurred.

9 Validation

9. 1 Requirement Completion Overview.

The functional and non-functional requirements that were developed in the requirements engineering phase and classified in the requirements chapter of this document are revisited again in this section to evaluate the requirement completion of the implemented system. The evaluation of the requirements and its deviations are listed in Table 27 below.

Requirement Identifier	Description	Satisfied (Yes/No)	Deviation If No
REQ – 1	Pick ICs	Yes	
REQ – 2	Place ICs	Yes	
REQ – 3	Press ICs (80N Max)	Yes	
REQ – 4	Heat ICs (150°C)	Yes	
REQ – 5	Low Latency	Yes	
REQ – 6	Precise Insertion Force	Yes	
REQ – 7	Precise Force Reading	No	0.0413N
REQ – 8	Accurate Force Reading	No	0.413%
REQ – 9	Precise Temperature Reading	Yes	
REQ – 10	Precise Temperature Steps	No	0.75°C
REQ – 11	Pick & Place Confirmation	Yes	
REQ – 12	Engineer & Board Safety	Yes	
REQ – 13	Cost-Efficient	Yes	
REQ – 14	Software Portability	Yes	
REQ – 15	Robust Communication	Yes	
REQ – 16	Code Comprehensibility	Yes	

Table 27 : Functional and non-functional requirements evaluation.

Several tests were carried out to benchmark and analyse whether the system met the specified requirements. While some requirements were validated using detailed technical

measurements, others did not require extensive testing because their fulfilment was more straightforward. In this section, these tests and their results are listed, and some of the approaches taken to meet the requirements are also listed.

I. Low Latency

The system consists of six parallel processes. While some of these processes do not directly interact with the user interface, others do. Operations such as sending commands, executing them, and motor controlling must operate smoothly and with minimal latency to ensure optimal performance. In Table 28, each process and its average execution loop frequency are listed.

Process Name	Average Frequency	Stack Size (bytes)
Task1Code	1 kHz	4000
Task2Code	6.5 kHz	4000
Task3Code	34 Hz	4000
Task4Code	900 Hz	4000
Task5Code	14 Hz	4000
Task6Code	150 Hz	4000

Table 28 : Average execution loop frequency.

Task1Code and Task2Code, which are responsible for processing user commands, managing state machines, and controlling the motor, operate at sufficiently high frequencies, resulting in very low latency. However, it is observed that the processes responsible for reporting status and sensor data to the serial interface, as well as heat control, are significantly slower compared to the other processes. This is due to the blocking and slowness of the temperature sensor readings in Task5Code, as well as the fact that data is reported in JSON format.

II. Precision in Insertion Force, Force Reading and Accuracy

After reviewing specifications such as non-linearity, hysteresis, repeatability, and creep for the selected load cell, we calculated the Root Sum Square (RSS) error to be approximately 0.1413 N. Given that the system's required minimum applicable step size is 10 N, this results in a measurement accuracy of 1.413%. Although the control software theoretically allows for minimum step sizes around 1 N, this would lead to reduced accuracy, and such small

insertion force steps are not necessary for the application in this project. Therefore, the requirements are evaluated as “NO” due to the deviation of 0.4 from the required value.

III. Precision in Temperature Reading and Application Step

Even though the prediction in temperature reading is met with the selected HW, the required prediction in temperature is calculated to be around 2.75 °C, as shown in Figure 35, where the plunger was set to a target heat of 120 °C and its temperature is monitored.

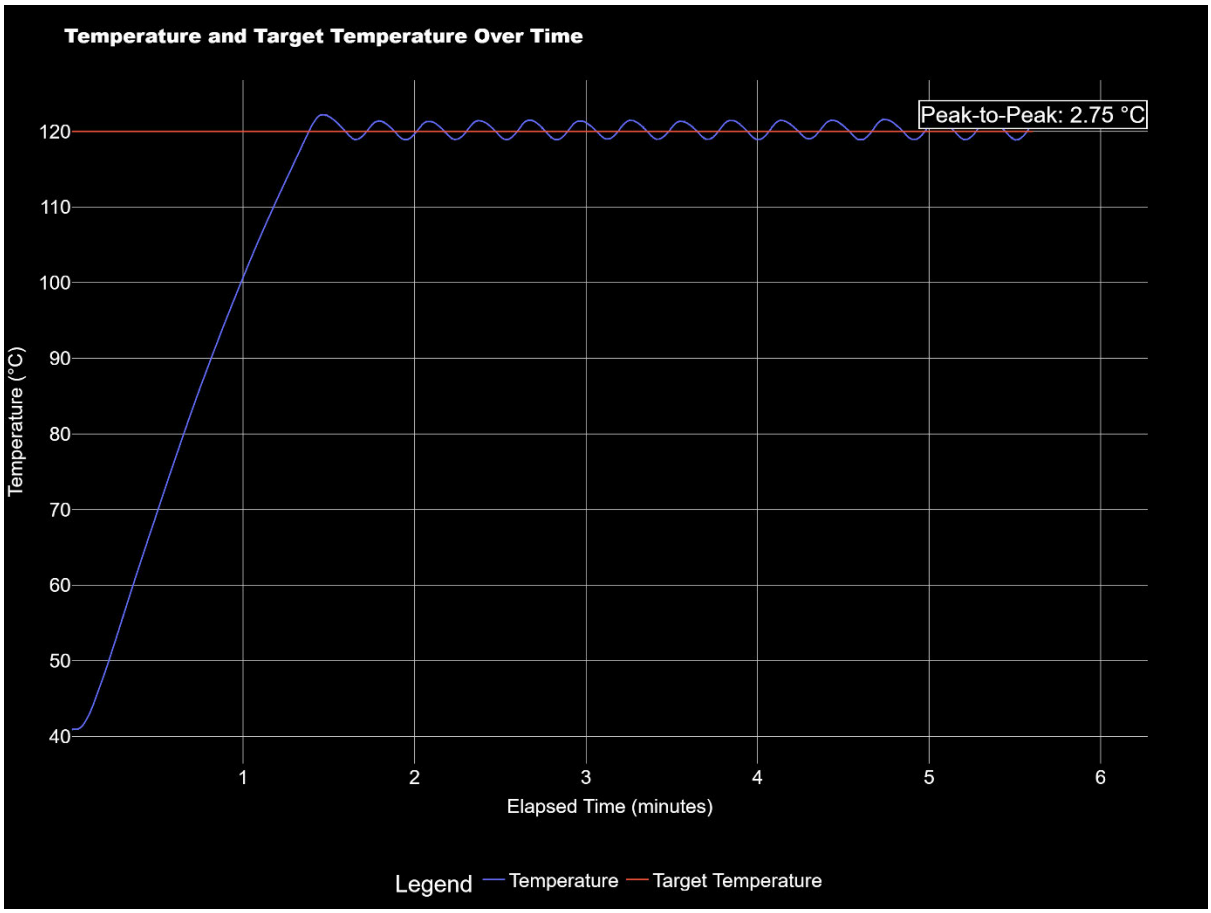

Figure 35 : Temperature of the plunger for a target heat value of 120°C.

IV. Engineer & Board Safety

The lab engineer and the test socket board could be at risk of damage if the plunger malfunctions during socket insertion, heating, or in scenarios where the engineer’s hand or another body part is in a compromising position relative to the plunger.

To address these scenarios, several measures have been implemented and are listed in Table 29 below. Although many safety measures have been taken regarding the system software, no software can be guaranteed to be perfectly stable and safe.

Solutions
1. Use of semaphores to avoid race conditions, inconsistent data states, and deadlocks in parallel processing.
2. Use “configASSERT(xReturned == pdPASS);” to ensure the successful creation of parallel tasks and semaphores, and halt the system if not.
3. Watchdog timers are used for important tasks where a fire hazard could be a problem, such as controlling the heater. If this task is not responding for a selected time interval, the system will enter panic mode and reboot.
4. An emergency stop button that cuts all power and a stop command that puts the system into a stop state were implemented.
5. Max force checking and retracting the plunger, so the socket is salvaged in case of a sudden increase in force crossing 80N (configurable) force.
6. To prevent the plunger from being pushed or retracted out of bounds, the system actively polls the state of the optical end stops during plunger movement.
7. Input command filtering and error checking. e.g., only insertion forces under 120N or temperatures under 180°C are passed on as valid commands to the system after serial command processing.
8. For PID controller stability measures such as, a. Proper parameter tuning method usage. b. Integral clamping, output clamping, and resetting of the PID parameters are done to achieve a more stable controller. c. Use of a median filter to minimise the effect of random ADC spikes influencing the controller. d. Limiting the motor PIDs operation for the linear range, where the PID starts to calculate after around 20N of force is read. e. When PID control is active, the system is designed to call this function for calculating the values with the same frequency (if not, it would cause malfunctions in the controller due to the integral and derivative parts).

Table 29 : Measures to for improved safety of the system.

V. SW Portability, Power, Code Clarity and Communication

The rest of the requirements are listed below in Table 30 along with the measures taken for successful implementation of them.

Requirement	**Measures Taken**
Software Portability	• Use of Free-RTOS for portability [27].
Optimized Power	• Task sleeping instead of busy waiting, • Motor is turned on only when needed and in most of the time remains off.
Code Comprehensibility	• Commenting when required, and Doxygen documentation of the source code. • State diagrams of the FSM and flowchart to understand the flow of events in the system.
Cost-Efficient	• Not accounting NRE costs and in house printed 3D structures, the costs come to under 350€ for one Eva Lab Handler excluding the mounting robotic arm.

Table 30 : Requirements and measures taken to meet them.

9.2 System performance and validation through lab test cases

Along with meeting functional and non-functional requirements to validate the system, this system should match or exceed the efficiency and precision of the process where the validation engineer manually swaps samples and uses a lid to securely place the IC in the socket.

The measurements listed below in the next section are conducted on the NXP's Automotive Radar chip called SAF85XX, illustrated in Figure 36, which is yet in preproduction and being actively validated in my department. The Integrated Circuit (IC) is utilised in a variety of advanced automotive applications, including adaptive cruise control (ACC), autonomous emergency braking (AEB), blind spot detection (BSD), door open warning (DoW), front collision warning (FCW), front cross traffic alert (FCTA), lane change assistance (LCA), park assist (PA), rear cross traffic alert (RCTA), and reverse autonomous emergency braking (R-AEB) [28].

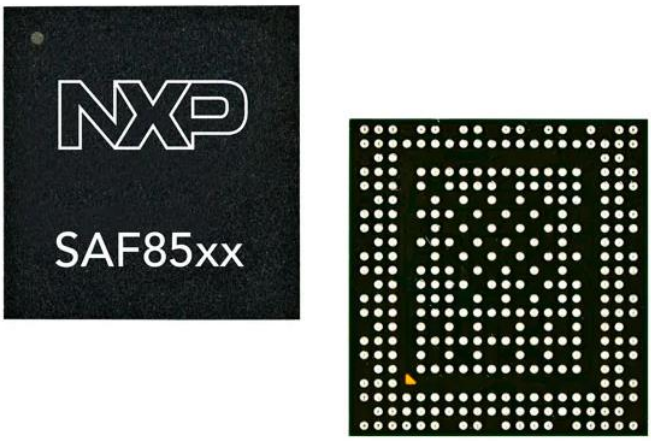

Figure 36 : High Performance 77GHz RFCMOS Automotive Radar One-Chip SoC [28].

9.2.1 IC Transmitter Power Test Case

The IC was pressed into the socket with around 70N of force, and then the transmitter power of the IC was measured over time with regulated force by activating the KEEP_PRESSING function, which maintains the set insertion force over time and unregulated force. This was compared with the case where the engineer swaps the IC manually and uses the specified lid to push the IC into the socket. This was done in three temperature test cases, namely ambient (25°C), hot (150°C), and cold (-40°C), which are the three temperature measurements done in the validation lab.

For confidential reasons, no absolute values of any of the measures are plotted, and only the ratio with respect to the reference case is plotted using the formula (2) below. The reference case is 0 dB in this case due to the difference between itself as a test case being zero.

$$\text{Power Ratio (dB)} = 10 \cdot \log_{10}\left(\frac{\text{Test Case Power (mW)}}{\text{Ref Case Lid Power (mW)}}\right) \qquad (2)$$

During a test, the IC heats up when transmitting, and this, as well as heating up the device for hot measurements and cooling it down for cold measurements, expands or shrinks the metal plunger. This will influence the insertion force of the IC on the socket and could cause deviations in power due to the change in impedance of the pogo pins of the socket.

I. Test at Ambient Temperature (25 °C)

As shown in Figure 37, it is clearly observed that when the force is regulated, the TX power is relatively constant compared to the unregulated test case, where the power has a slightly increasing line.

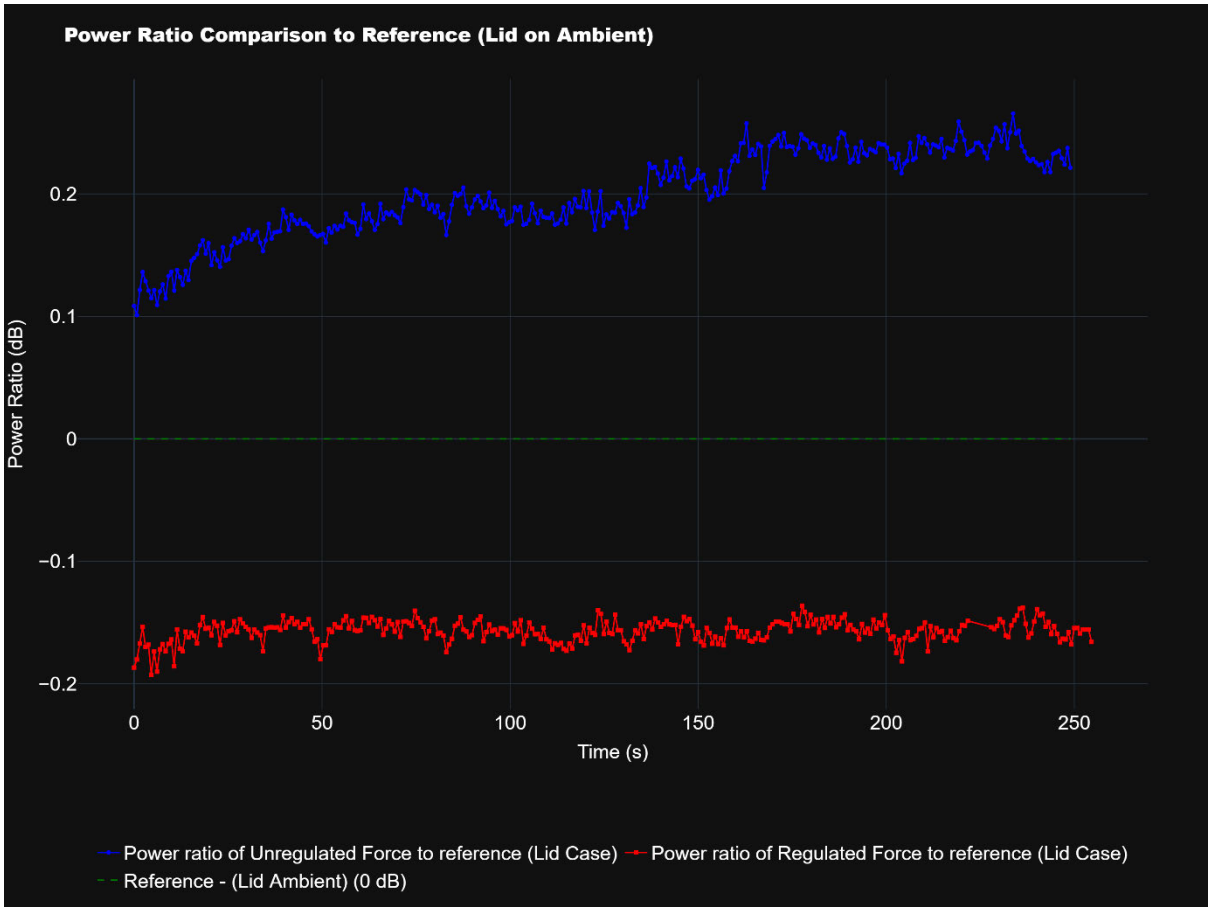

Figure 37: TX power comparison at ambient.

II. Test at Cold Temperature (-40 °C)

As illustrated in Figure 38, the temperature readout from the IC was cooled down to around -40 °C, and the TX power was recorded. In this case, the regulated force performed better in power around +3.5 dBs compared to the unregulated force test case.

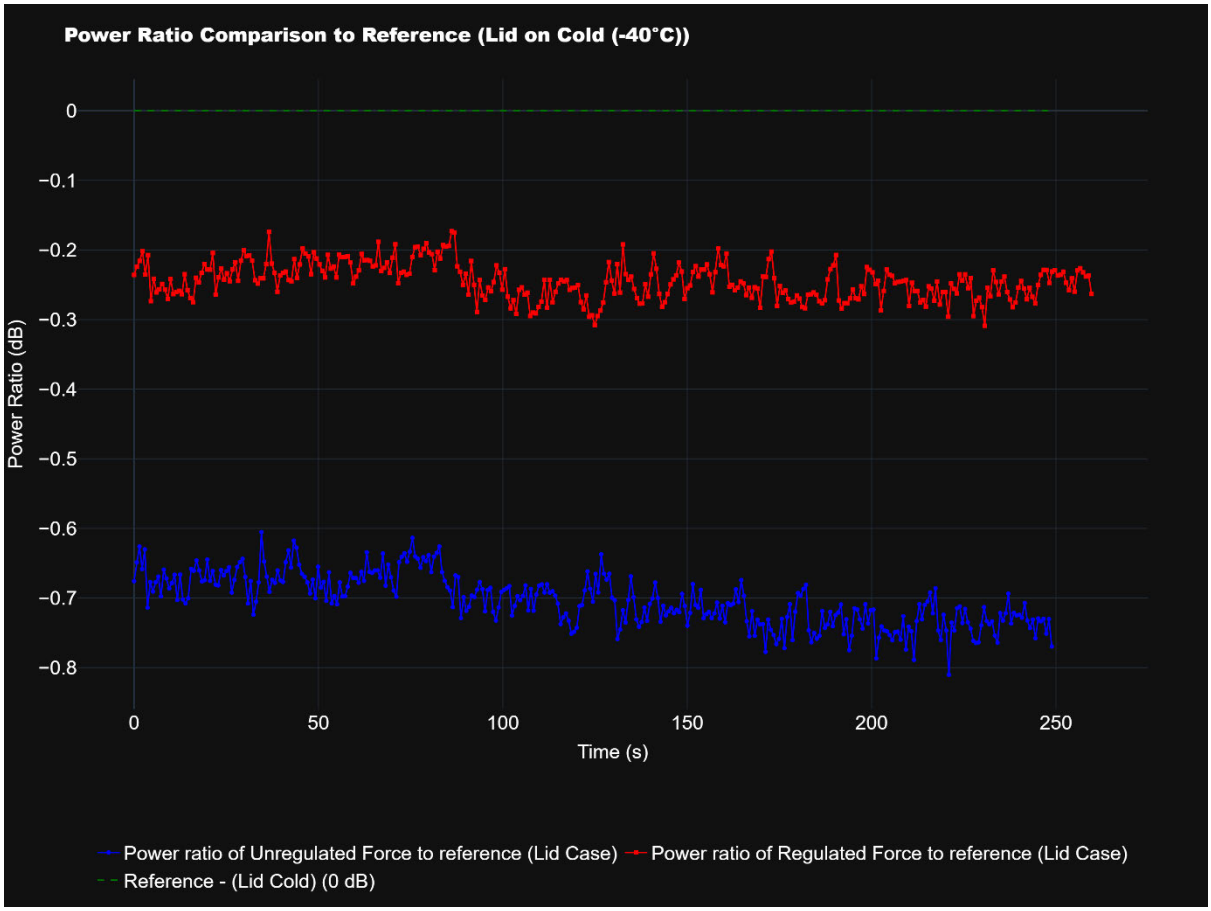

Figure 38 : TX power comparison at -40°C.

III. Test at Hot Temperature (150 °C)

As illustrated in Figure 39, the temperature readout from the IC was heated up to around 150 °C, and the TX power was recorded. In this case, the regulated force performed only slightly better, with almost no difference in power compared to the unregulated force test case.

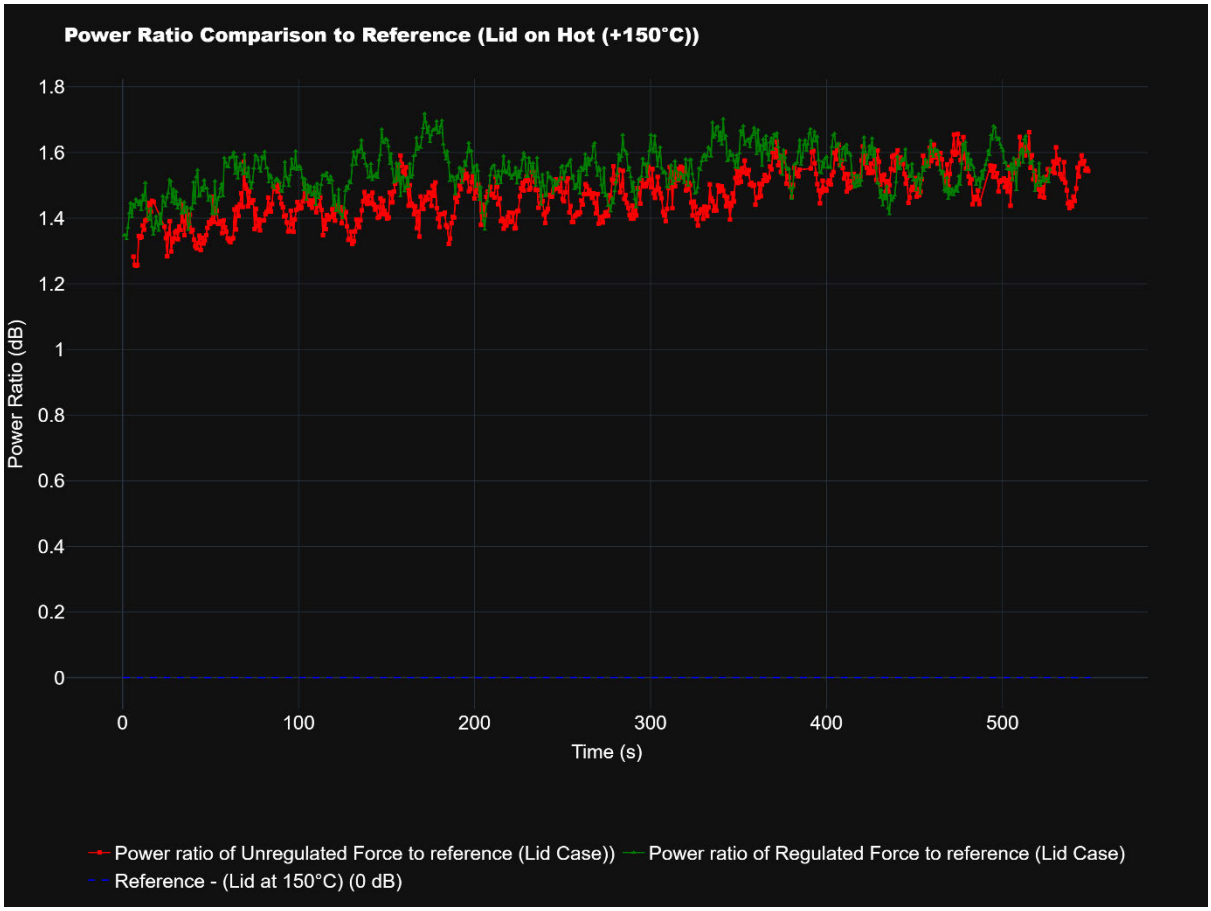

Figure 39 : TX power comparison at 150°C.

10 Summary & Outlook

10.1 Summary and Key Findings of Analysis

Based on the analysis presented in 8.6 Analysis of Challenges in Implementation, as well as insights gained from various interactions and findings encountered during the system's implementation and testing phases, the table below outlines the lessons learnt and recommended steps for avoiding anomalies and deviations in future iterations.

Anomaly / Deviation	Possible Solutions
ADC Spikes	• Shielding of the traces in PCB. • Twisted pair traces in PCB. • Short and direct traces on the PCB between the ADC and the load cell. • EMI shielded enclosure for the PCB. • Shielded wires from load cell to PCB connector.
Accuracy	• Load cells with narrower but more accurate full-scale ranges.
Slow Temperature Reads	• Although for the current Temperature PID and the heater the temperature sensors reading frequency is good enough, a faster reading sensor would be better and would be able to get the required temperature steps.

Table 31 : Suggested solutions for the next iteration of the project.

To avoid the anomalies and deviations encountered in this iteration, which were difficult to counteract with SW solutions, Table 31 above lists potential solutions for consideration in the next iteration.

10.2 Conclusion

This thesis successfully designed, developed, implemented, and validated an RTOS system that uses PID control for socket insertion force and heater temperature control, meeting nearly all functional and non-functional requirements of the lab handler project with minimal deviations. The system is now suitable for use in IC chip validation in conjunction with a Cobot.

This conclusion was reached after thoroughly reviewing the validation section where the system is tested and analysed to check for functional and non-functional requirements, and also the overall system with its current state has been compared with the reference test case where the engineer swaps the samples manually.

10.3 Future Work and Recommendation

Although the system has successfully met most of the project requirements, there is still room for improvement and optimization. This section lists some potential improvements that could have a positive impact on the system in addition to the listing included in the Summary and Key findings section.

1. For further safety of the temperature PID control, where a possible thermal runaway scenario could happen due to faulty connection or malfunction of the temperature sensor, a SW solution with a redundant temperature sensor could be implemented for a safer system with higher MTBF.
2. Redundant load cell sensors for safety and higher MTBF of the system in socket insertion force control.

Bibliography

[01] reichelt elektronik G. I. Team (webmaster@reichelt.de), ‘UF XARM5 LITE - UFactory xArm5 Lite’, Elektronik und Technik bei reichelt elektronik günstig bestellen. Accessed: Sep. 09, 2024. [Online]. Available: https://www.reichelt.de/ufactory-xarm5-lite-uf-xarm5-lite-p299080.html

[02] ‘PCB Prototype & PCB Fabrication Manufacturer - JLCPCB’. Accessed: Sep. 09, 2024. [Online]. Available: https://jlcpcb.com/

[03] R. Malan and D. Bredemeyer, ‘Functional Requirements and Use Cases’, Dec. 2001.

[04] Y. H. Hee, M. K. Ishak, M. S. M. Asaari, and M. T. A. Seman, ‘Embedded operating system and industrial applications: a review’, *Bull. Electr. Eng. Inform.*, vol. 10, no. 3, Art. no. 3, Jun. 2021, doi: 10.11591/eei.v10i3.2526.

[05] S. Fischmeister and P. Lam, ‘Time-Aware Instrumentation of Embedded Software’, *IEEE Trans. Ind. Inform.*, vol. 6, no. 4, pp. 652–663, Nov. 2010, doi: 10.1109/TII.2010.2068304.

[06] K. W. Batcher and R. A. Walker, ‘Interrupt Triggered Software Prefetching for Embedded CPU Instruction Cache’, in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’06)*, Apr. 2006, pp. 91–102. doi: 10.1109/RTAS.2006.24.

[07] M. Nahas, ‘Implementation of highly-predictable time-triggered cooperative scheduler using simple super loop’, *Int. J. Comput. Sci. Eng.*, Jul. 2011.

[08] M. Pont, S. Kurian, H. Wang, and T. Phatrapornnant, *Selecting an appropriate scheduler for use with time-triggered embedded systems*. 2007, p. 618.

[09] S. Kurian and M. J. Pont, ‘The maintenance and evolution of resource-constrained embedded systems created using design patterns’, *J. Syst. Softw.*, vol. 80, no. 1, pp. 32–41, Jan. 2007, doi: 10.1016/j.jss.2006.04.007.

[10] P. Hambarde, R. Varma, and S. Jha, ‘The Survey of Real Time Operating System: RTOS’, in *2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies*, Jan. 2014, pp. 34–39. doi: 10.1109/ICESC.2014.15.

[11] X. Guan, Q. Xing, and L. Feng, ‘Implementation of embedded system platform based on μC/OS-II and S3C44B0X microprocessor’, in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Aug. 2011, pp. 2205–2208. doi: 10.1109/MEC.2011.6025929.

[12] ‘FreeRTOS documentation - FreeRTOS™’. Accessed: Sep. 09, 2024. [Online]. Available: https://freertos.org/Documentation/00-Overview

[13] ‘Zephyr Project Documentation — Zephyr Project Documentation’. Accessed: Sep. 09, 2024. [Online]. Available: https://docs.zephyrproject.org/latest/index.html

[14] ‘NuttX Documentation — NuttX latest documentation’. Accessed: Sep. 09, 2024. [Online]. Available: https://nuttx.apache.org/docs/latest/

[15] K. Andersson and R. Andersson, ‘A comparison between FreeRTOS and RTLinux in embedded real-time systems’.

[16] Y. Neuhard, ‘A Comparison of Real-time Operating Systems for Embedded Computing’, Technische Universität Kaiserslautern, Department of Computer Science, Summer term 2022. [Online]. Available: https://es.cs.rptu.de/publications/datarsg/Neuh22.pdf

[17] M. H. Qutqut, A. Al-Sakran, F. Almasalha, and H. S. Hassanein, ‘Comprehensive survey of the IoT open-source OSs’, *IET Wirel. Sens. Syst.*, vol. 8, no. 6, pp. 323–339, 2018, doi: 10.1049/iet-wss.2018.5033.

[18] A. Serino and L. Cheng, ‘A Survey of Real-Time Operating Systems’.

[19] B. Boulet, *Introduction to Feedback Control Systems*, REV 0. 2000.

[20] ‘The PID Controller & Theory Explained’. Accessed: Sep. 09, 2024. [Online]. Available: https://www.ni.com/en/shop/labview/pid-theory-explained.html

[21] B. M. Sarif, D. V. A. Kumar, and M. V. G. Rao, ‘Comparison Study of PID Controller Tuning using Classical/Analytical Methods’, vol. 13, no. 8, 2018.

[22] G. Ellis, ‘Chapter 6 - Four Types of Controllers’, in *Control System Design Guide (Fourth Edition)*, G. Ellis, Ed., Boston: Butterworth-Heinemann, 2012, pp. 97–119. doi: 10.1016/B978-0-12-385920-4.00006-0.

[23] M. L. Luyben, ‘Essentials of process control’, *No Title*, Accessed: Sep. 09, 2024. [Online]. Available: https://cir.nii.ac.jp/crid/1130000794396931328

[24] S. Nikita and M. Chidambaram, ‘Tuning of PID Controllers for time Delay Unstable Systems with Two Unstable Poles’, *IFAC-Pap.*, vol. 49, no. 1, pp. 801–806, Jan. 2016, doi: 10.1016/j.ifacol.2016.03.155.

[25] M. O. Okelola, D. O. Aborisade, and P. A. Adewuyi, ‘Performance and Configuration Analysis of Tracking Time Anti-Windup PID Controllers’, *J. Ilm. Tek. Elektro Komput. Dan Inform.*, vol. 6, no. 2, p. 20, Jan. 2021, doi: 10.26555/jiteki.v6i2.18867.

[26] ‘Velocity Open-Loop’, Arduino-FOC. Accessed: Sep. 09, 2024. [Online]. Available: https://docs.simplefoc.com/velocity_openloop

[27] R. M. Gomes and M. Baunach, ‘A Model-Based Concept for RTOS Portability’, in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, Oct. 2018, pp. 1–6. doi: 10.1109/AICCSA.2018.8612862.

[28] ‘Snapshot’. Accessed: Sep. 09, 2024. [Online]. Available: https://www.nxp.com/products/radio-frequency/radar-transceivers-and-socs/high-performance-77ghz-rfcmos-automotive-radar-one-chip-soc:SAF85XX

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

_____	_____	_[redacted]
City	Date	Signature