Wireless servomotor control for Lathes

Using a simplex motion servomotor and wireless Wi-Fi interface for safe and flexible field machining.

Degree project in the Mechatronics Engineering Program.

Yngve Larsson

Elias Lindén

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Acknowledgments

This degree project was carried out at Chalmers University of Technology as part of the three-year engineering program in Mechatronics. The project marks the final step of our education and serves as the concluding work of the program. The work has been conducted in close collaboration with Metalock for whom the final product was developed. Additionally, the project involved technical cooperation with Simplex Motion, the supplier of the integrated servomotor used in the design.

We would like to extend our sincere thanks to Johan Lundgren at Metalock for his consistent guidance and hands-on support throughout the process. We are also grateful to Pär Jalbin at Simplex Motion for sharing his technical expertise and for assisting with the motor integration. Special thanks go to Henrik Karlsson at Puketorps Bud for his help in acquiring the necessary components.

We also wish to thank Åke Larsson for his guidance and expertise regarding the ESP32 microcontroller and the TFT display, which were essential to the development of our system. Furthermore, we are grateful to Sven Larsson for acting as a valuable sounding board and for his help with tools, component assembly, soldering, and various practical tasks.

Finally, we would like to express our appreciation to our examiner, Rob Maaskant at Chalmers, for his feedback and academic supervision during this project.

Gothenburg, May 2025

Abstract

This project presents the development of a compact, wireless control system for an automated feed mechanism driven by a servomotor. The work was carried out as part of a bachelor's thesis in mechatronics engineering at Chalmers University of Technology. The goal was to design a self-contained unit capable of precise motion control and reliable wireless communication, while meeting requirements like portability, runtime, and safety. The final system integrates a Simplex Motion SE010 servomotor and an ESP32-based T-Display S3 microcontroller module. Communication between components is established via serial interface (RS232, TTL-level), and the system operates independently using a Wi-Fi access point hosted by the microcontroller. A lithium-polymer battery pack supplies power, offering over seven hours of continuous use, surpassing the original five-hour runtime target. Key features include a local display for system monitoring, a web interface for wireless control, and emergency stop functionality. The system has undergone testing, demonstrating reliability, stable communication, and effective performance across various feed rates. The solution is adaptable to work with several different Simplex Motion motor variants and power levels, making it suitable for a range of industrial or research applications. Future improvements may include enhanced safety features, such as obstacle detection and positional sensors, as well as better control at low speeds.

Table of Contents

1. Introduction

Feeding systems, which control the movement of cutting tools or workpieces during machining, play an important role in mobile machining operations. This report presents the development of a wireless-controlled servo motor system for axial feed. The focus lies on implementing wireless remote control, ensuring reliable operation, adaptability, and compliance with safety standards.

1.1 Background

Metalock is a company that manufactures a wide range of specialized components and tools on demand, while also providing services on existing, installed machinery. One of their key areas of expertise is on-site turning of round components of all sizes, using portable precision equipment that can be brought and placed on the machinery being repaired. Typical applications include shafts, bearing seats, and flanges. By offering these services on-site, Metalock helps minimize transport needs and reduce downtime for the companies they serve. However, some of these on-site operations still offer potential for automation and further efficiency improvements.

1.2 Purpose

The purpose of this thesis is to design a prototype that automates one of the manual processes involved in on-site turning. Specifically, the task of manually feeding the cutting blade in or out, which is currently done by turning a small handwheel, as seen in Figure 1.

Figure 1. Leadscrew with handwheel from Metalock (image by the author)

By developing a small, portable wireless device that automates the feeding function with a motor, the need for workers to physically enter the turning zone is eliminated. This results in safer and more efficient operation.

1.3 Boundaries

The level of the design is limited to a conceptual prototype, meaning the product does not need to be fully functional or ready for commercial use. The focus of this thesis is to ensure that the design meets the functional and safety requirements provided by Metalock, such as implementing a working emergency stop and input control for the motor. It is also important to note that Metalock's turning equipment comes in various sizes, and this design is specifically tailored for a relatively small unit. Therefore, adaptations would be required for use with larger machines.

1.4 Clarification of Research Questions

Turning operations require a high level of accuracy, which makes the choice of motor critical. Metalock Engineering has already selected a suitable motor manufacturer, Simplex Motion, a Gothenburg-based company specializing in compact servomotors ideal for this application. The objective is to use one of these motors to drive a mechanism that feeds the cutting blade forward and backward during the turning process.

One of the main challenges in the design of this device is the requirement for it to be wireless. This introduces several questions, particularly concerning safety, most notably how to implement a reliable and regulation-compliant emergency stop. Since emergency-stop systems must adhere to strict standards, part of the thesis work involves identifying a viable solution that meets those requirements.

Another design consideration is the implementation of wireless communication between the remote control interface and the servomotor. There are multiple options for achieving a wireless connection, and a key task will be to evaluate and select the most suitable technology for this specific use case.

Additionally, the device must be portable, and battery powered. The chosen battery solution must be rechargeable, flight-legal, and powerful enough to meet performance demands. However, battery development is not within the scope of this thesis; instead, the focus is on designing a concept that integrates commercially available batteries that meet the company's requirements.

1.5 Outline of the Report

A general layout of the report can be seen in figure 2 below.

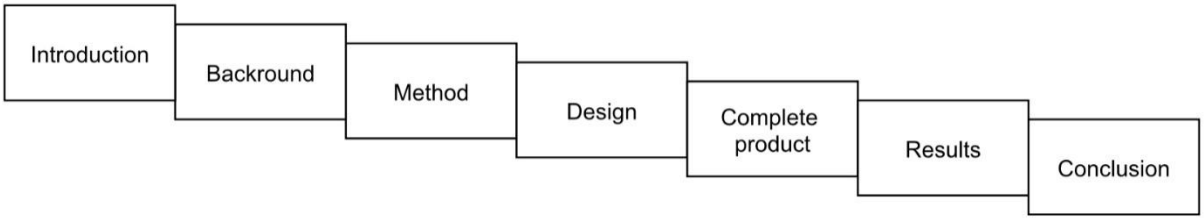

Figure 2, general layout of the report.

2. Background Material

The background material has been structured into several key parts. The following sections cover the selection of motor, battery solution, communication method, emergency stop functionality and contains explanations of each part works.

2.1 Lathe Processes

A lathe is a machine tool that removes material from a workpiece by rotating it against a fixed cutting tool. This process can be applied to the outside of the object to reduce its diameter or to the inside to create a circular hole [1]. It is a widely used process in manufacturing due to its versatility and precision and is especially suitable for producing round objects or holes that require high dimensional accuracy [2].

When using a lathe, cutting speed is an important factor that affects the surface finish of the workpiece. The appropriate speed depends on several factors, such as the type and sharpness of the cutting tool, the material of the workpiece, and the required surface quality [3].

The use of manual lathes in manufacturing has become less common, as Computer Numerical Control (CNC) machining offers higher efficiency and precision. CNC machines can be fully automated, allowing continuous operation with minimal human intervention [4].

To automatize the lathe process, you can use a servomotor instead of manually feeding the cutting tool. In order to find a suitable motor for this there are some facts about servomotors and their different types in the next chapter.

2.2 Servomotors

Servomotors are essential components in modern automation and control systems, as they operate in a closed-loop system that provides continuous feedback. This feedback enables precise control of position, velocity and acceleration. Compared to stepper motors, servomotors offer higher torque at high speeds and typically require an encoder and gearbox for improved accuracy. Due to their precise and easy control, servomotors are widely used in various fields, including CNC machinery and automotive applications [5].

2.2.1 Basics of the Servomotor

A servomotor consists of an AC or DC motor combined with a sensor, typically an encoder, for position feedback. It operates using Pulse Width Modulation (PWM), where the width of the input signal determines the shaft's angle of rotation from its starting position [6].

Inside the servo, a potentiometer and a set of gears help control the shaft’s position and movement. The encoder monitors the shaft's speed and position to ensure accurate motion [6].

The motor functions by generating a rotating magnetic field in the stator windings, which the rotor (connected to the shaft) follows. When the rotor and the stator's magnetic field rotate at the same speed, the motor is classified as a synchronous motor [6].

2.2.2 Types of Servomotors

2.2.2.1 AC and DC Servomotors

Servomotors are commonly divided into two main types: AC (alternating current) and DC (direct current). The primary difference between them is the type of current used to power the motor, but they also differ in performance characteristics and typical applications [7].

AC servomotors generally have lower efficiency (around 5–20%) but can deliver high torque and speed. They are known for stable and smooth operation, minimal stability issues, and the absence of radio frequency noise. These features make them well-suited for indoor environments where noise reduction is important. AC servomotors are also relatively compact and lightweight [7].

In contrast, DC servomotors offer higher efficiency but have limitations in speed and torque. They tend to generate more noise and radio frequency interference and may experience greater stability challenges. DC servomotors are typically larger and heavier, making them more appropriate for applications where efficiency is critical and high speed or torque is less important [7].

2.2.2.2 Brushless Motors

Brushless motors have become widely used in various industries due to their high efficiency, durability, and performance. Unlike brushed motors, brushless motors operate without mechanical brushes and rely on electronic control instead [8].

A brushless motor consists of a rotor with permanent magnets and a stator containing windings. The stator generates a rotating magnetic field through electronic commutation, which drives the rotor. Most commonly, brushless motors use a three-phase configuration, meaning the stator contains three separate windings [8].

2.2.5 Torque-Speed Characteristics

Simplex Motions website shows the torque and rotational speed characteristics of the Simplex Motion SE010 servomotor. A curve defines the motor's operating range, with a peak torque of 0.2 Nm and a power limit of 75 W for short-term operation, approximately three times the nominal continuous rating. The maximum speed depends on the supply voltage. Compared to traditional stepper motors, this servomotor maintains more consistent torque over a wider speed range, making it suitable for applications that require both precise low-speed control and fast high-speed movement [9].

The motor requires electricity and since the servomotor will be part of a portable design, a battery is the best way to meet this need. +

2.3 Lithium Batteries

Lithium-based batteries are commonly used in modern embedded systems due to their high energy density, compact size, and reliable performance. Lithium, the lightest metal in the periodic table, has high electrochemical reactivity, allowing the batteries to provide a typical cell voltage of around 3.7 V and low self-discharge rates [11].

The most common type of lithium-based battery is the lithium-ion (Li-ion) battery, which includes variants such as lithium-polymer (Li-Po). Li-ion batteries typically use a liquid electrolyte and are valued for their high efficiency and energy density. These characteristics make them suitable for applications like electric vehicles and stationary energy storage, where long-lasting power is essential. Variants such as lithium iron phosphate ($LiFePO_4$) offer improved safety and longer cycle life, while nickel manganese cobalt (NMC) batteries provide high energy density and strong power output [12].

Li-Po batteries are a subtype of Li-ion batteries that use a gel or solid polymer electrolyte instead of a liquid one. This enables more flexible and lightweight designs, making them ideal for drones, wearable devices, and other applications where size, weight, and safety are important. However, Li-Po batteries generally have lower energy density, and a shorter lifespan compared to some other Li-ion chemistries [12].

The batteries can store sufficient amounts of energy, but due to different voltage requirements for the components there is a need for a voltage regulator.

2.4 Voltage Regulator

A voltage regulator is a device that provides a stable output voltage from varying input voltage, ensuring the correct supply for electrical components [13].

The Luxorparts adjustable switching voltage regulator converts an input voltage between 3 V and 40 V DC to an adjustable output voltage ranging from 1.25 V to 35 V. It can deliver up to 2.5 A of current with an efficiency of over 90%, resulting in low heat generation. Due to its high efficiency, minimal cooling is required under normal conditions [14].

In addition to the electrical components of the system, in order to control the different parts, there is a need for communication, in this case a wireless one.

2.5 Wireless Communication Alternatives

Bluetooth:

Bluetooth is a wireless communication technology widely used in devices such as cars, speakers, computers, phones, and digital cameras. It is designed for short-range communication and often replaces physical cables in existing equipment. It offers a low-cost and user-friendly method for data transmission where it serves as a modern alternative to older communication protocols like RS-232 serial ports, providing greater flexibility and convenience [15].

Bluetooth operates in the license-free 2.4 GHz ISM (Industrial, Scientific, and Medical) frequency band. It uses Frequency Hopping Spread Spectrum (FHSS), switching between frequencies 1600 times per second to reduce interference and enhance communication security [15].

An example of a module that provides Bluetooth functionality is the ESP32-WROOM-32E, based on Espressif Systems' ESP32 microcontroller. It features a dual-core 32-bit Xtensa® LX6 processor running at up to 240 MHz, along with embedded flash memory, SRAM, and various digital and analog interfaces, including GPIO, UART, SPI, and I²C. The module is optimized for low power consumption and supports multiple sleep modes, making it suitable for power-sensitive applications [16].

Thanks to its compact form factor and integrated wireless capabilities, the ESP32-WROOM-32E is commonly used in embedded systems that require Bluetooth connectivity for device-to-device communication without dependence on a central network [16].

Wi-fi:

Wi-Fi is a wireless communication technology used to transfer data between devices over distances of approximately 50–100 meters. It is based on the IEEE 802.11 standard and allows devices to connect to a local network without the need for physical data cables. Wi-Fi typically operates in the 2.4, 3.6, 5, and 60 GHz frequency bands [17].

A Wi-Fi network usually consists of an access point (AP) that manages communication with multiple stations (STAs). Communication can occur in a centralized mode, where all data passes through the AP, or in an ad hoc mode, where devices communicate directly without fixed infrastructure [17].

Although Wi-Fi can face challenges such as security vulnerabilities and reduced performance with many connected users, its low infrastructure cost, ease of deployment, and support for mobility have made it popular in homes, offices, and public spaces. Recent advancements have significantly improved data throughput, increasing it from 11 Mbps to 54 Mbps and enhancing overall performance [17].

An example of a microcontroller module with integrated Wi-Fi functionality is the T-Display S3, developed by LilyGO. It is based on the ESP32-S3 system-on-chip, which includes a dual-core 32-bit Xtensa LX7 microprocessor, built-in 802.11 b/g/n Wi-Fi, Bluetooth Low Energy, onboard flash memory, and various I/O interfaces. The ESP32-S3 is designed for embedded systems and is commonly used in applications such as automation and remote sensing [18].

The T-Display S3 includes built-in USB connectivity for programming and data transfer. It also features a 1.9-inch TFT display, which operates using a grid of rows and columns of electrodes to form individual pixels. Each pixel is controlled by a thin-film transistor (TFT), which improves switching speed and enhances display performance [19].

The T-Display S3 module provides general-purpose input/output (GPIO) pins for connecting external components such as sensors, actuators, and communication peripherals. It can function as a standalone device or as part of a network system. Its microcontroller architecture supports real-time processing alongside wireless communication [18].

This type of module enables integration into wireless systems without requiring external Wi-Fi chips or interfaces, making it well-suited for compact, self-contained embedded applications. Additionally, the ESP32-S3 can operate in access point mode, allowing the module to create its own Wi-Fi hotspot for wireless connection with other devices [18].

2.6 Communication with Modbus-protocol

Modbus is an open communication protocol that has become widely used due to its simplicity and ease of implementation. It follows a master-slave model, where a single master communicates with up to 247 slave devices. Slaves are typically Modicon PLCs with EIA-232C interfaces, while masters are usually host-computers or programming panels [19].

Because EIA-232C has limited communication range, EIA-485 (RS-485) is often used for longer-distance communication [20].

There are several versions of Modbus: RTU, ASCII, and TCP. Modbus RTU uses a compact binary format over serial connections; Modbus ASCII transmits data in a readable text format, making it easier to interpret manually; and Modbus TCP operates over Ethernet, making it suitable for modern networked systems [20].

In the Modbus protocol, only the master initiates communication. Slave devices cannot send data unless requested by the master. Slaves are addressed from 1 to 247, while the master has no address, as it does not receive commands [20].

2.6.1 Serial Communication – RS232 and TTL

RS-232 is a widely used communication standard that transmits data using two wires: one for transmitting (TX) and one for receiving (RX). A shared ground connection is required, as communication is based on voltage levels relative to ground. In RS-232, a digital '1' is represented by a voltage between –5 V and –15 V, while a digital '0' ranges from +5 V to +15 V [21].

Data is sent serially, meaning one bit at a time with one line in each direction. To synchronize communication, each data transmission begins with a start bit (logic 0). RS-232 commonly uses a 9-pin D-sub (DB9) connector, which allows direct connection to a computer's COM port [21].

Transistor-Transistor Logic (TTL) is a UART (Universal Asynchronous Receiver/Transmitter) communication standard in which a logic HIGH is defined by the device's supply voltage (typically +3.3 V or +5 V), and a logic LOW is defined as 0 V (ground). TTL specifies both the voltage levels and the data packet structure, enabling reliable communication between compatible devices [22].

To maintain safety in this wirelessly controlled device, we need an emergency stop. This is also a requirement in the Swedish regulation.

2.7 Emergency Stop

In industrial environments and workplace safety, a high level of protection is essential. One key safety component when operating machinery with moving parts is the emergency stop system. Its primary function is to quickly shut down the machine to prevent accidents. Emergency stops can also be used to stop operation in the event of a detected fault, helping to prevent material damage and reduce potential costs.

The device developed in this project qualifies as a machine, defined as a product with one or more moving parts powered by an external energy source. As such, it is subject to specific legal safety requirements during both development and future use [23].

During the prototyping phase, the product must comply with Annex 1 of AFS 2023:11, a Swedish regulation that outlines essential health and safety requirements for machinery under development [24].

According to Section 2.7 of AFS 2023:11, machinery must, where relevant, be equipped with an emergency stop device capable of immediately halting any hazardous motion. The device must remain in the stop position until it is manually reset, and it must be clearly visible and easily accessible [24].

As the product is intended for use beyond the prototype stage and may eventually be introduced to the market, it must also comply with the European Machinery Directive [25].

Until January 20, 2027, the applicable regulation is Directive 2006/42/EC. After that date, it will be replaced by Regulation (EU) 2023/1230 on machinery [25].

According to Directive 2006/42/EC, clause 1.2.4.3, machinery must be equipped with one or more emergency stop devices to allow the prevention of imminent or existing hazards. The emergency stop function must override all other controls, must not create additional risks, and must require manual resetting before the machinery can resume operation [25].

Regulation (EU) 2023/1230 maintains the core principles established in the earlier directive. It reiterates that emergency stop devices must be designed to safely halt machine operations,

must not automatically reset or restart processes, and must be clearly identifiable and operable under all expected use conditions [26].

To ensure future flexibility and readiness for market release, it is important that the device complies with all relevant standards and directives. Meeting these requirements would make the device largely compliant with CE marking regulations.

3. Method

The project starts with a requirements specification, created together with Metalock Engineering. It defines key criteria for the feeder system, such as speed, torque, and size. These are divided into essential (base) and optional (extra) requirements. Extra features will only be included if there is enough time.

The prototype is developed on a conceptual level, which means there is room for improvement. Regular meetings with Metalock help ensure the project meets their needs and stays on track. This also helps confirm that the design is a possible solution.

The focus of the thesis is to select suitable components and develop control logic for the feeder. If time allows, the physical design of the prototype may also be included.

3.1 Project plan and timeline

One of the first steps for the project was to develop a simple timeline, as seen in figure 3, in order to keep the project within its scope and to make sure that the deadline was met. Due to some delays and setbacks this timeline was not always followed, but did provide valuable information on how to handle these setbacks and what to prioritize.

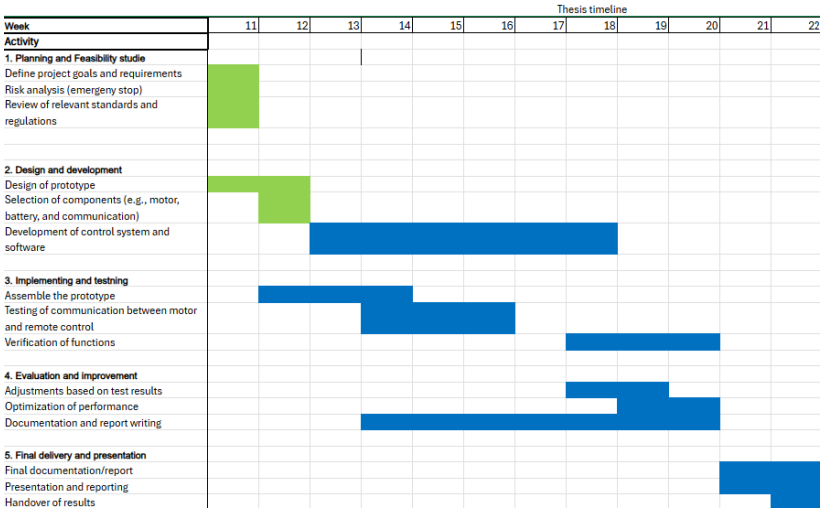

Figure 3. Timeline of the project.

3.2 Requirement-driven component selection

To develop the complete systems design, a comparative requirements-based approach was used A complete requirement specification was created in collaboration with Metalock which covers the basic safety functions needed, size, communication requirements, and more. Furthermore, each choice of a design solution can be directly traced back to this list ensuring that the goal, scope, and requirements were followed.

3.3 Data collection and analysis

Relevant data for the project were collected in several different ways.

Document reviewing played a big part of collecting relevant data and was used throughout the project. Specifically, the technical datasheets for the Simplex Motion motor and the LILYGO ESP32 were studied in detail in order to correctly implement a working wiring solution and build the logic for the microcontroller using the Modbus RTU protocol.

Some interviews with Simplex Motion were held regarding the functionality of their servomotor, the SE010. This provided some basic knowledge on how to implement the motor in our solution, which laid the ground for our own experiments and prototyping.

A qualitative assessment in order to ensure ease of integration, safety considerations and physical form factor will be done in the following chapters using an elimination matrix to finalize the selection of components.

3.4 Implementation protocol

Using the methods mentioned in 3.2 and 3.3 together with the time plan, the work was divided into five steps, as seen in figure 4.

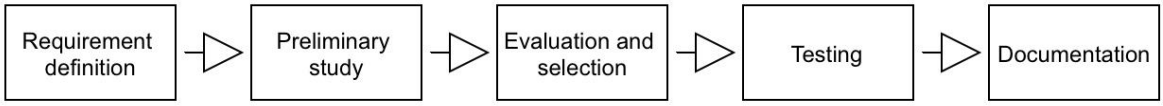

Figure 4, The general workflow.

4. Design and Evaluation of Solution Alternatives

The design and selection process in this project has been structured into several key parts to ensure a clear and systematic development approach. It begins with the requirement specification, which outlines the fundamental needs and constraints for the system. Subsequent sections cover the selection of the motor, communication method, battery solution, and emergency stop functionality.

4.1 Requirement Specification

Metalock requires the ability to advance a cutting tool on a lathe using a wirelessly controlled servomotor. A set of technical specifications outlining this requirement is presented in Table 1.

Table 1. Device specifications and requirements provided from Metalock

Power	25-100 Watts	Base requirement
Variable speed	0,01-0,24 mm/min (axial tool movement during engagement)	Base
Rapid feed	500 mm/min (during movement)	Base requirement
Minimal form factor	As small and compact as possible	Base requirement
Customizable, with stronger versions available		Base requirement
Environmentally resistant	Dust	Extra
Price		Extra
Should be a servomotor		Base requirement
Must be airline approved	Max 100Wh	Base requirement
Battery time	At least 5 hours	Extra
Nondependent on extern Wi-Fi connection		Base requirement
Reliable		Extra
Fast		Extra
Emergency stop	That meets machine directives	Base requirement

4.2 Choice of Servomotor

The servomotor used in this thesis is the Simplex Motion SE010, an integrated brushless DC servomotor. The choice was made in collaboration with Metalock Engineering. The SE010 is a compact, open-frame motor that suits the design of the feeder system well.

The motor provides a continuous output of 25 W and up to 0.2 Nm of torque. For short periods, it can deliver up to 75 W, which gives flexibility during high-demand operations. It reaches speeds of up to 4000 RPM, depending on the supply voltage. These performance levels are more than sufficient for the needs of the feeder.

Unlike traditional stepper motors, the SE010 offers relatively stable torque across a wide range of speeds. This makes it suitable for applications that require both precise control at low speeds and fast motion when needed.

The motor includes several input options, such as analog input, digital I/O, RS485, and RS232 (TTL level) communication. All settings can be configured using the Simplex Motion Tool, a PC software that allows full control of the motor parameters.

Because of an integrated microprocessor, the SE010 has the ability to operate as a stand-alone unit. This supports the goal of building a compact, wireless, and self-sufficient feeder system. Its brushless design also contributes to long-term reliability and resistance to dust and debris.

The full register map as well as the technical specifications for the motor can be found on Simplex Motions official website [9][10].

4.3 Choice of Communication

4.3.1 First Selection

To enable wireless control of the SE010 servomotor from Simplex Motion, a Bluetooth-based communication approach is being considered. The motor supports serial communication via RS232 (TTL level) which a Bluetooth module can interface with through a direct serial connection.

An ESP32-WROOM-32 module has been selected for this purpose. It features built-in support for Bluetooth and programmable serial communication. Only the module itself is used, not a complete development board. It is programmed separately and connected directly to the motor, which also supplies power to the module.

Bluetooth is considered due to its low energy consumption, broad compatibility with mobile devices, and sufficient data throughput for simple command-based control. In this setup, commands could be sent to the motor from a phone or computer, using the motor’s RS232 interface.

A practical challenge related to this solution is the development of an HMI (Human-Machine Interface). While several mobile apps can connect to Bluetooth Low Energy (BLE) devices, many of them reset or disconnect when re-establishing a connection. An alternative would be to create a custom interface accessible from both computers and smartphones, but this is not addressed at the current stage of the project.

4.3.2 Second Selection

As an alternative to Bluetooth, a Wi-Fi-based communication solution was evaluated. The selected module is the T-Display S3 developed by LilyGO, which was introduced in Section 2.5. This device integrates a dual-core ESP32-S3 microcontroller and built-in Wi-Fi connectivity, making it a suitable choice for embedded control systems. A visual representation of the module is shown in Figure 5.

Figure 5. ESP32 and TFT display (T-Display S3) from LilyGo (image by the author).

During operation, the T-Display S3 runs in access point (AP) mode, allowing a mobile device or computer to connect directly without the need for external infrastructure. A browser-based interface hosted on the module enables real-time adjustment of motor parameters such as speed and direction.

The module also features a 1.9-inch TFT display, which enhances local interactivity by providing visual feedback during system development, testing, and operation. This display improves the user experience and supports on-site debugging without the need for additional peripherals.

The design of the ESP32-S3 allows it to run the Wi-Fi communication and control interface in parallel, enabling responsive and seamless interaction. The setup is also designed to be modular and scalable, allowing future integration into more complex environments such as remote diagnostics or multi-device networks.

4.3.3 Elimination Matrix

Table 2. Elimination Matrix. Weighted score = ∑ (Weight * Raw score). The highest total score results in the best alternative in regard to the requirements.

Criterion	Weight	ESP32-WROOM-32	ESP32+Display

Combability with servomotor	4	4	4
Ease of integration	5	3	4
Programmability	3	2	3
Size and physical space	4	3	2
Debugging capabilities	3	2	4
Access to support	3	1	2
Total (raw score)		15	19
Total (weighted score)		58	71

4.3.3 Final Choice of Communication

The final communication method selected for the system is based on the T-Display S3 module from LILYGO, which integrates an ESP32-S3 microcontroller with support for both Wi-Fi and Bluetooth. This module was selected based on the results of the elimination matrix (Table 2), where it achieved the highest score. In addition to this, the integrated 1.9-inch TFT display, although not included as a criterion in the matrix, provided a valuable advantage by enabling local feedback and interaction during both development and operation.

The system design of the device now consists of the Simplex Motion SE010 servomotor and the LILYGO ESP32 T-Display S3 module. The motor provides a 14-pin interface with support for serial communication via RS232 (at TTL level). Communication between the motor and the ESP32 is established directly through their respective RX/TX lines via UART, which eliminates the need for an external RS485 transceiver.

Power to the ESP32 module is supplied from the main battery via a voltage converter, which steps down the voltage from 14.8V to 4 V, which is the required operating voltage for both the microcontroller and its display. Please see figure 6 below for the complete schematic of the design.

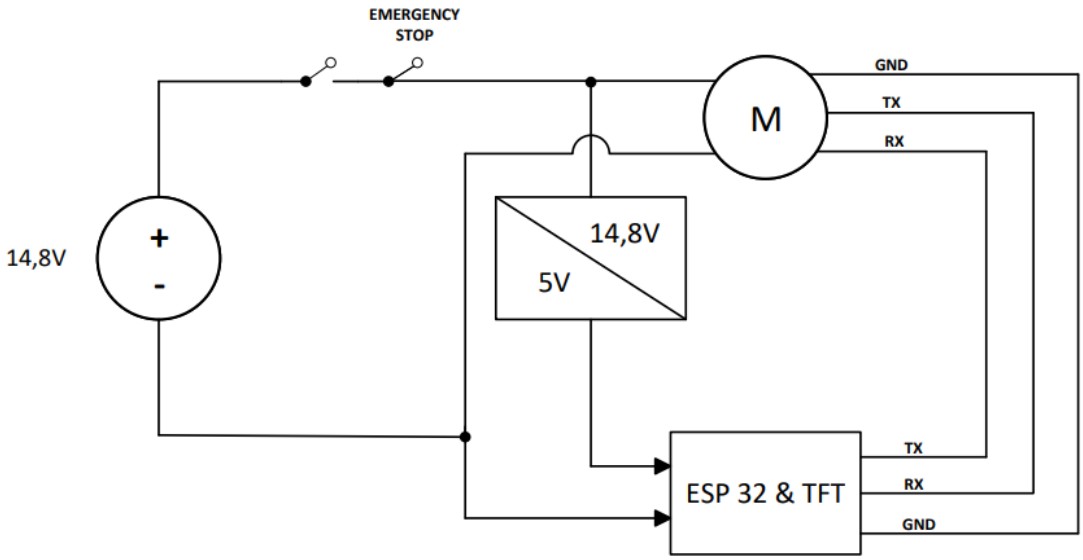

Figure 6. Complete schematic of the design.

Choosing Wi-Fi as the communication method addressed one of the major challenges encountered during development, the design of a suitable human-machine interface (HMI). Existing mobile applications for Bluetooth Low Energy (BLE) were found to be unreliable, often resetting upon each connection, and creating a custom application from scratch was not possible within the given timeframe.

By using Wi-Fi and the T-Display S3 module's built-in hotspot functionality, it became possible to host a custom-built website from the microcontroller itself. This solution enables direct access to the motor control interface via any web browser, without requiring external infrastructure or software installations. The only requirement is a device with Wi-Fi capability and access to the network password. The LilyGo T-display S3 is designed in such a way that the last uploaded code will be stored and loaded each time a power source is plugged in, fulfilling the needs of the stand-alone device.

This approach provides high flexibility and accessibility, allowing multiple types of devices to interact with the system.

4.4 Choice of Battery

A compact and reliable power source is essential for the system, particularly given the portability requirements and the motor’s operational demands. As described in the background section, lithium-based batteries offer high energy density and low weight, which makes them ideal for embedded applications.

Of the available options, lithium-polymer (Li-Po) batteries were selected for this design. Compared to traditional cylindrical lithium-ion cells, Li-Po batteries provide a slimmer

profile and greater design flexibility, key benefits when working within the constraints of a portable system.

The SE010 servomotor requires a minimum supply voltage of 12V, and the design specification calls for at least five hours of runtime. To meet these requirements, two Li-Po batteries were connected in series.

Selected battery solution:

- 7.4 V 3900 mAh 25C 2S Li-Po Battery (RGT EX86100)

Each battery delivers 7.4V and 28.9~Wh. In series, the combined voltage reaches 14.8V, while the total energy remains 28.9~Wh. This solution is enough to power both the motor and supporting electronics, including the ESP32 module, which is powered via a voltage converter that steps down the voltage to 4V. The configuration remains well below the 100~Wh threshold for air travel, ensuring compliance with transport regulations.

The final decision to use Li-Po batteries was based on their compact size, sufficient energy capacity, and favorable integration into the overall system design.

4.5 Emergency Stop Solution

To meet the safety requirements regarding emergency stops, and to prepare the product for potential CE certification, an emergency stop device with a red push-button, as seen in Figure 7, has been selected. The button will be installed in an easily accessible location, although its exact placement may vary depending on the operational environment.

When the emergency stop is engaged, all power to the motor is immediately disconnected, preventing any operation. To restore functionality, the button must be manually reset by pulling it back to its original position.

This implementation aligns with the requirements specified in AFS 2023:11 and the European Machinery Directive, both of which state that emergency stop devices must be clearly identifiable, manually resettable, and capable of stopping hazardous motion without delay.

Figure 7. Emergency stop-solution for lathe controller. The emergency stop button is located on top of the application, allowing for easy access if needed. (Image by the author)

5. The Complete Product

The final version of the feeding device is a fully wireless control system for an integrated servomotor. It allows the user to adjust motor speed, direction, and feed mode via a dedicated web interface. The system is built around an ESP32 microcontroller, which acts as the central hub for communication between the operator, motor, and the integrated TFT display, as seen in Figure 8.

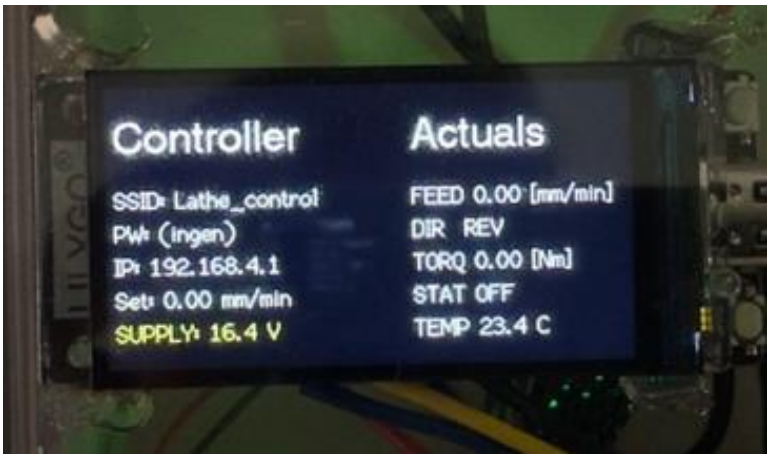

Figure 8. TFT display (image by author).

The system is easy to understand and is based on programmed logic executed by the microcontroller. It also offers the ability to upload custom code and make modifications via USB-C, allowing direct communication between a computer and the ESP32. See Figure 9 for complete setup.

Figure 9. The final design of the system (image by author)

5.1 Final System Design and Development Environment

The software is developed in Visual Studio Code using an extension called PlatformIO, which is specifically designed for IoT development. PlatformIO simplifies the process of handling libraries, compiling the firmware, uploading to hardware, and serial monitoring for debugging. Please see figure 10 for the communication flow of the final system design.

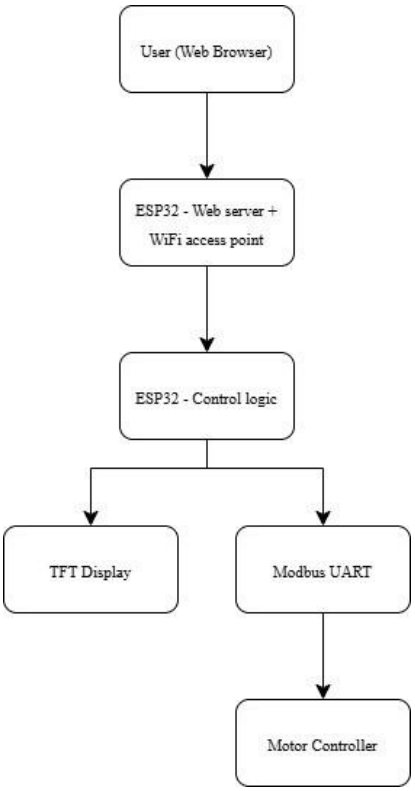

Figure 10. Communication flow between the user interface, ESP32, and servomotor (Image created by the author).

The codebase is written in C++, and the logic is modularized into several classes, each responsible for handling different parts of the system, including motor control, WiFi communication, and data display. This modular design improves both the readability and maintainability of software. See Appendix A-D for complete source code with comments.

5.2 Wireless User Interface via Wi-Fi

The ESP32 is configured to operate as a Wi-Fi Access Point (AP) under the network name Lathe_control. When a user connects to this network and opens a browser, an HTML-based interface is served locally from the ESP32.

User interactions such as setting speed, direction, or feed mode are handled via HTTP POST requests, which are interpreted by the WiFiAccessPointServer class. Based on these commands, the system updates the motor state in real-time

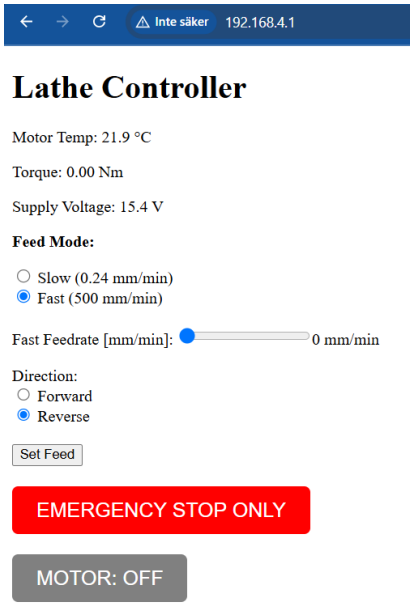

Figure 11. Screenshot of the web-based user interface used to control the ESP32 and servomotor (image created by the author).

The web interface provides the operator with a range of control options, as shown in Figure 11. Each button and action are clearly labeled for intuitive and self-explanatory use. However, operating the motor through the interface does require some basic understanding.

The motor can be turned off at any time by clicking the "MOTOR:" button. When the toggle indicates "OFF", the motor is completely disabled and will not respond to any commands. When the motor is enabled, the operator can adjust the feed mode, direction, and feedrate.

To ensure reliable communication and avoid unexpected behavior, it is recommended to first reduce the feedrate to zero and update the setting before making major changes, such as switching between fast and slow feed modes. Once the mode is changed, the desired feedrate can be set.

The "EMERGENCY STOP ONLY" button is intended for use in critical situations. Pressing this button forces the servomotor into an immediate halt by activating its emergency mode (mode 5 in the register map). It should be noted that after triggering the virtual emergency stop, the system may require a manual reboot via USB-C using VS Code and PlatformIO to restore normal operation.

It is recommended to set the feedrate to zero before turning off the motor power supply. Otherwise, the motor may resume operating at the previous speed when power is reapplied.

5.3 Built-in Safety: Ping-Pong Check

In order to enhance operational safety, the system includes a "ping-pong" safety mechanism: The web interface sends a ping signal every second while a user is connected. If the ESP32 does not receive a ping, it assumes the connection has been lost and immediately stops the motor. This ensures that the motor does not continue running without supervision.

5.4 Communication with the Servomotor

The motor (Simplex Motion SE010) is controlled using the Modbus RTU protocol over a UART2 serial connection. Communication takes place between the ESP32 (master) and the motor (slave, address 1), using the following settings:

- Baud rate: 57600
- Data bits: 8
- Parity: Even
- Stop bits: 1

Communication is implemented via the ModbusMaster library in PlatformIO. Motor commands are executed by writing values to the motor's predefined register map. All logic related to Modbus communication and register handling is encapsulated in the MotionController class. For detailed information on register addresses, the complete register map can be found on Simplex Motions webpage.

5.5 Graphical Feedback via TFT Display

The 1.9" TFT display is used to show real-time system feedback. It updates every 500 milliseconds and fetches values from the motor status registers. These values include speed, direction, fault codes, and are partly mirrored in the web interface as well.

The graphics are rendered using the TFT-eSPI library with sprites, allowing efficient refreshment without flickering. The design of the TFT layout can be found in the code, see appendix A. The display helps with diagnostics and usability during both development and operation.

5.6 Summary and Modularity

Each part of the system, Wi-Fi communication, motor control, display, and safety checks, is implemented as a separate software class. This structure enables independent development and testing of each subsystem and makes it easy to extend the codebase with new features in the future without disrupting the core functionality.

6. Results

6.1 Testing and Evaluation

Extensive testing was carried out to evaluate the performance, stability, and reliability of the developed system. All tests were performed under realistic operating conditions over an extended period of time. The primary areas of focus included wireless communication, motor control responsiveness, battery performance, and overall system robustness.

6.1.1 System Stability and Reliability

The system was operated in over 100 complete test cycles without experiencing any critical software crashes, hardware malfunctions, or overheating issues. Even during long-duration test sessions lasting several hours, all core functionalities remained operational. Minor communication delays or brief disconnections were observed in a few cases, but the system recovered gracefully without requiring a manual restart. These results indicate a high level of system robustness suitable for real-world use.

6.1.2 Wireless Communication and Interference Resistance

The system relies entirely on wireless communication via the ESP32 access point. During testing, a stable connection was maintained in approximately 95% of all test cases, even in environments with potential sources of interference such as other wireless networks or electronic equipment. In the remaining cases, brief signal interruptions were observed, but these were automatically handled by the system's safety mechanisms (such as ping-pong monitoring), which ensured a controlled stop of the motor. Overall, the communication link proved highly reliable and well-suited for the application.

6.1.3 Control Responsiveness

Control commands issued through the interface were executed with minimal delay. When operated via a computer connected directly to the ESP32 access point, response times were virtually instantaneous. When using a mobile device, a delay of approximately one second was occasionally observed. This slight latency was still within acceptable limits and did not negatively affect the functionality of the system.

6.1.4 Safety Note: Mode Change Handling

During motor control testing, a key usability issue was identified regarding the transition between different feedrate modes (slow and fast). Since both modes share the same control slider in the user interface, switching between them without resetting the slider can cause abrupt and unintended speed changes.

To ensure safe operation, the feedrate slider should be manually set to zero before changing feed mode. Failure to do so may result in the previous mode's feedrate value being applied to the new mode, which can cause sudden and unsafe acceleration. While this does not

compromise functionality, it should be either adjusted in future software revisions or clearly communicated as an operational guideline for users.

6.1.5 Communication Range

Range testing in an open environment showed that the system maintains a reliable connection up to approximately 8 meters without issues. The actual range could be estimated to be up to 20 meters, but the connection at this distance could be weak and slow to respond. However, this provides the operator with significant flexibility and freedom to operate the device remotely, without physical restrictions.

6.1.6 Battery Life

Battery life was tested by running the motor continuously at a feedrate of 500 mm/min (millimeters per minute). Under this load, the system operated for approximately seven hours before the batteries were depleted. This result exceeds the minimum requirement of five hours and confirms that the system is capable of supporting a full working session on a single charge. Due to our timeline, this test was only carried out once. In order to properly assess the battery's runtime. More extensive and multiple tests should be conducted.

6.2 Final Results

The following section outlines the performance of the final system, evaluated against the key design requirements outlined earlier in the project. The product consists of a Simplex Motion SE010 servomotor and a T-Display S3 microcontroller module, integrated into a compact unit with wireless control functionality and battery operation. The evaluation is structured according to the key requirement categories: performance, form factor, communication, battery, price, and safety.

6.2.1 Performance and Functionality

The servomotor used in the final design meets the base requirement of being able to provide a power output in the range of 25–100~W. It supports variable speed control and has been tested to deliver feed rates between 0.01 mm/min and 0.24 mm/min during engagement, fulfilling the requirements for precision machining. Additionally, the motor is capable of operating at a rapid feed rate of 500 mm/min, as specified.

The system design allows for customization and scaling, with the possibility to replace the motor with stronger variants if needed. Although please note that the system is built on a logic working for Simplex Motion servomotors. If servomotors from other manufacturers are to be used, the software will most likely need changes. This flexibility ensures adaptability for future versions with higher force or speed demands.

6.2.2 Size and Environmental Resistance

The integrated design consists of the motor, a compact microcontroller with built-in display, a voltage converter, and Li-Po batteries resulting in a minimal form factor. This compact

design allows for installation in environments with limited space and supports the project's focus on portability.

The Simplex Motion SE010 is a brushless servomotor, which inherently reduces internal wear and particle generation compared to brushed motors. While it is not officially rated for extreme environmental conditions, its compact and sealed design offers a reasonable degree of durability for light industrial or workshop use. However, for environments with significant dust exposure, additional protective measures may be necessary.

6.2.3 Communication

A key requirement was to develop a self-contained communication system that does not rely on external infrastructure. This has been successfully achieved using the ESP32 microcontroller, which operates in access point mode, allowing users to connect directly to the system without external Wi-Fi. Wireless communication has proven reliable in practice, with stable connections between the user device and the motor control interface.

Compared to the initially investigated Bluetooth solution, the Wi-Fi-based approach offers higher data bandwidth, enabling faster updates and a smoother control experience. The integrated web interface, accessed via the module’s IP address, provides a platform-independent control method accessible through any standard browser.

6.2.4 Battery and Power

The system is designed to run on a rechargeable battery that complies with airline transport regulations, including the requirement of a maximum 100Wh capacity. Testing indicates that the system can operate continuously for at least five hours, meeting the requirement for an extended runtime.

Power is supplied to the motor directly from the battery. The ESP32 microcontroller with TFT display is also powered by the battery, using a voltage converter in order to receive the correct voltage. The solution provides enough power to support the device and allows it to act as a stand-alone design.

6.2.5 Price

The total hardware cost of the product is 6200SEK, which is considered cost-effective given the system's functionality, level of integration, and adaptability. This pricing makes it competitive for small-scale industrial or research use and supports further development or upscaling.

6.2.6 Safety

The system includes a software-based emergency stop function that immediately halts motor operation upon command. As mentioned before, several safety functions are integrated within the software's logic. To comply with safety regulations, specifically the EU Machinery Directive, the system has been supplemented with a physical emergency stop. This ensures

full compliance with current legal requirements for commercial use. The physical emergency stop is integrated directly into the system enclosure for quick and safe access in emergency situations. Worth mentioning is that the placement of the emergency stop is not ideal and will need to be changed depending on the circumstances of the workspace.

7. Conclusion and Discussion

This project has successfully resulted in the development of a wireless control system for driving a lead screw in a lathe application. The system includes a simplex motion servomotor, a battery solution, graphical user interface as well as both software-based and physical emergency stop functions. Testing has demonstrated that the system operates reliably under normal conditions and fulfills its intended functionality.

7.1 Conclusions

Although the system includes both a software-based and a physical emergency stop function, its implementation still presents practical challenges due to the physical setup of the application. The product is designed to be mounted on an arm within a lathe, making it difficult to position the emergency stop button in an easily accessible location. While the most convenient solution from the client's perspective would be to have even easier access to the physical emergency stop, this remains challenging given the mounting constraints. As such, the issue remains unresolved and must be addressed in future iterations, either by redesigning the mounting solution to ensure the emergency stop is always within easy reach or by integrating alternative compliant safety mechanisms that meet regulatory standards.

Another issue that must be addressed before the product is used, or at the very least understood to ensure safe operation, is the way the torque and supply voltage values are displayed on the website. These parameters do not update in real time; instead, they are only refreshed when the operator manually changes the feed settings using the "Set Feed" button. Unfortunately, this limitation was discovered too late in the development timeline for a solution to be implemented. However, it provides a valuable starting point for future improvements to the prototype.

A minor issue has been observed when reversing direction in slow feed mode: the motor speed occasionally exceeds the intended range of 0.01–0.24 mm/min. Due to time constraints, this issue has not yet been resolved. However, the reverse functionality in slow feed mode is not critical to the system's core operation. Importantly, forward motion in slow feed mode performs as expected and without fault.

One of the system’s main strengths lies in its scalability. The software architecture, wireless interface, and display module are all compatible with a wide range of servo motors from Simplex Motion. This includes not only the SE010 model used in this project, but also higher-performance variants such as the SM100, which features a continuous output of 160 W and torque levels of up to 0.51 Nm at 3000 rpm. The modular nature of the solution allows it to adapt to different power levels and application needs without significant changes to the core hardware or software.

7.2 Scalability and Recommendations for Future Work

Several areas for improvement have been identified during development and testing. Firstly, the inclusion of positional sensors could greatly enhance operational precision by detecting when the lead screw has reached predefined limits. This would also allow for automated stopping, reducing the risk of mechanical collisions. Secondly, the system could benefit from additional safety mechanisms such as obstacle detection sensors, preventing the machine from proceeding if an obstruction is detected. Also, the system's stability at very low speeds could be improved. Although the motor includes internal regulation features that can mitigate this, additional fine-tuning of the control parameters or feedback loops may further increase the precision and responsiveness of the tool.

In addition to these points, it is worth mentioning that the motor used in the system is capable of delivering up to 25 W of mechanical power at its nominal speed of 4000 RPM. While the system supports a fast feed rate of up to 500 mm/s, the cutting operation is limited to a much lower feed rate of 0.24 mm/s. Delivering the full 25 W of power at this low-speed mode would require a high gear ratio, which is considered impractical due to mechanical complexity and efficiency losses. Although achieving maximum output power at the screw was not a design requirement, this observation could be relevant in future developments. If higher mechanical output is needed during low-speed operation, the transmission system may need to be redesigned to accommodate that demand, perhaps by using a planetary gear box.

References

[1] Woodbury RS. The Origins of the Lathe. *Scientific American*. April 1963. Available from: https://www.jstor.org/stable/10.2307/24940936

Accessed 2025-04-10.

[2] Virasak L. Chapter 2: Lathe Machine. In: *Manufacturing Processes 4-5*. Open Oregon Educational Resources; 2019.

Accessed 2025-04-08.

[3] Virasak L. Unit 2: Speed and Feed. In: *Manufacturing Processes 4-5*. Open Oregon Educational Resources; 2019.

Accessed 2025-04-08.

[4] Virasak L. Chapter 8: CNC. In: *Manufacturing Processes 4-5*. Open Oregon Educational Resources; 2019.

Accessed 2025-04-08.

[5] Baballe MA, Bello MI. Different Types of Servo Motors and Their Applications. In: Proceedings of ICEANS; 2022. Available from: https://www.researchgate.net/publication/370070680_Different_Types_of_Servo_Motors_and_Their_Applications

Accessed 2025-04-15.

[6] Cao W, Bukhari SAA, Aarniovuori L. Review of Electrical Motor Drives for Electric Vehicle Applications. *Mehran Univ Res J Eng Technol*. 2019. Available from: https://www.researchgate.net/publication/337610056

Accessed 2025-04-18.

[7] Baballe MA, Bello MI, Umar AA, Shehu AK, Bello D, Abdullahi FT. A Look at the Different Types of Servo Motors and Their Applications. *Scientific Journal of Engineering and Computer Science*. 2022 Apr 27. Available from: SJECS-3-4-9.pdf

Accessed 2025-03-21.

[8] Millett P. Brushless vs. Brushed DC Motors: When and Why to Choose One Over the Other. *Monolithic Power Systems*; 2021. Available from: 2021-brushless-vs-brushed-dc-motors-when-and-why-to-choose-one-over-the-other_r1.0.pdf

Accessed 2025-03-21.

[9] Simplex Motion. SE-Series Datasheet. Simplex Motion; 2023 Jan 27. Available from: SimplexMotionTechnicalDataSE.pdf

Accessed 2025-05-01.

[10] Simplex Motion. Simplex Motion Manual; Simplex Motion; 2023 Jan 27. Available from: https://www.simplexmotion.com/documentation/manual/SimplexMotionManual.pdf

Accessed 2025-05-01.

[11] Warner JT. *Lithium-Ion Battery Chemistries: A Primer*. Elsevier; 2019. Available from: https://books.google.se/books?hl=sv&id=2laXDwAAQBAJ

Accessed 2025-04-27.

[12] Bhawna, Phogat P, Shreya, Jha R, Singh S. Advancements and Challenges in Lithium-Ion and Lithium-Polymer Batteries: Towards Sustainable Energy Storage Solutions. *Ionics*. Springer; 2025 Apr 25. Available from: https://link.springer.com/article/10.1007/s11581-025-05315-6

Accessed 2025-04-26.

[13] Bhuiyan MAS, Hossain MR, Minhad KN, Haque F, Hemel MSK, Dawi OM, et al. CMOS Low-Dropout Voltage Regulator Design Trends: An Overview. *MDPI Electronics*. 2022 Jan 9. Available from: https://www.mdpi.com/2079-9292/11/2/202

Accessed 2025-03-30.

[14] Luxorparts. Luxorparts Variable Voltage Regulator, Switched. *Clas Ohlson*; n.d. Available from: https://www.kjell.com/se/produkter/el-verktyg/strmforsrjning/spnningsregulatorer/luxorparts-variabel-spnningsregulator-switchad-p44392

Accessed 2025-03-19.

[15] Bisdikian C. An Overview of the Bluetooth Wireless Technology. *IEEE Communications Magazine*. 2002 Aug 7. Available from: https://ieeexplore.ieee.org/document/1023781

Accessed 2025-03-28.

[16] Espressif Systems. ESP32-WROOM-32E Datasheet. Espressif; n.d. Available from: esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf

Accessed 2025-03-31.

[17] Banerji S, Chowdhury RS. On IEEE 802.11: Wireless LAN Technology. *Int J Mobile Netw Commun Telemat*. 2013 Jul 10. Available from: https://arxiv.org/abs/1307.2661

Accessed 2025-04-12.

[18] Lilygo. T-Display-S3. *LILYGO®*; n.d. Available from: https://www.lilygo.cc/products/t-display-s3

Accessed 2025-03-22.

[19] Lalama SJ. Flat Panel Display Manufacturing Overview. *IEEE Transactions on Electronics Packaging Manufacturing*. 2002 Aug 6. Available from: https://ieeexplore.ieee.org/document/1023782

Accessed 2025-03-25.

[20] Thomas G. Introduction to the Modbus Protocol. Contemporary Controls; 2008. Available from: https://www.ccontrols.com/pdf/Extv1.pdf

Accessed 2025-04-01.

[21] Sonnenberg J. Serial Communications RS232, RS485, RS422. *Raveon Technologies*; 2018. Available from: AN236(SerialComm).pdf

Accessed 2025-03-26.

[22] LaMeres BJ. *Embedded Systems Design using the MSP430FR2355 LaunchPad*. Springer; 2020 Jun 20. Available from: https://link.springer.com/chapter/10.1007/978-3-030-45234-1_7

Accessed 2025-04-05.

[23] Prevent. Machine Safety – Risks, Safety and Protection. *Prevent*; n.d. Available from: https://www.prevent.se/maskinsakerhet

Accessed 2025-03-23.

[24] Arbetsmiljöverket. AFS 2023:11 – Work Equipment and Personal Protective Equipment. *Arbetsmiljöverket*; last revised 2025 Jan 1. Available from: https://www.av.se/arbetsmiljoarbete-och-inspektioner/publikationer/foreskrifter/afs-2023-11/

Accessed 2025-03-20.

[25] European Agency for Safety and Health at Work. Directive 2006/42/EC - Machinery Directive. *EU-OSHA*; updated 2024 Jun 13. Available from: https://osha.europa.eu/en/legislation/directives/2006-42-ec

Accessed 2025-03-18.

[26] European Agency for Safety and Health at Work. Regulation 2023/1230/EU - Machinery Regulation. *EU-OSHA*; updated 2025 Jan 16. Available from: https://osha.europa.eu/en/legislation/regulations/2023-1230-eu

Accessed 2025-03-18.

Appendices

Appendice A: main.cpp

```
//main.cpp

#include <Arduino.h>
#include <SPI.h>
#include <TFT_eSPI.h>
#include "WiFiAccessPointServer.hpp"
#include "MotionController.hpp"

extern unsigned long lastClientPing;//Timestamp of last received client ping, used to
detect connection loss

// === PIN DEFINITIONS ===
#define PIN_LCD_BL   38//Pin controlling the LCD backlight
#define RXD2         18//UART RX pin for motor communication
#define TXD2         17//UART TX pin for motor communication
#define MOTOR_BAUD   57600// Baud rate for communication with the motor controller

//DISPLAY LAYOUT COORDINATES
#define LINE_1_Y 20
#define LINE_2_Y 60
#define LINE_3_Y 80
#define LINE_4_Y 100
#define LINE_5_Y 120
#define LINE_6_Y 140
#define COLUMN_1_X 5
#define COLUMN_2_X 5
#define LCD_WIDTH   320
#define LCD_HEIGHT  170

//TIMING INTERVALS
#define SCREEN_UPDATE_TIME     500//Screen refresh interval in milliseconds
#define WEB_TEMP_UPDATE_TIME  2000//Interval for updating temperature to web interface

//OBJECT INITIALIZATION
TFT_eSPI       tft;//Main display object
TFT_eSprite    spriteLeft(&tft);//Left section of the display
TFT_eSprite    spriteRight(&tft);//Right section of the display
TFT_eSprite    spriteRpm(&tft);//Area for displaying set RPM
TFT_eSprite    spriteSupply(&tft);//Area for displaying supply voltage

HardwareSerial     hwSerial(2);//UART interface for motor (UART2)
ModbusMaster       node;//Modbus protocol handler
MotionController   motor(node);//Motor control abstraction layer
WiFiAccessPointServer web("Lathe_control", "");// Web server acting as Wi-Fi access point

//MOTOR STATE TRACKING
int previousDirection = 1;//Default motor direction (1 = forward)

//TIMERS
unsigned long lastScreen = 0;// Last screen update timestamp
```

```
unsigned long lastWebTemp = 0;// Last web temperature update timestamp

void setup() {
  //Turn on power and LCD backlight pins
  pinMode(15, OUTPUT);
  digitalWrite(15, 1);//Activates external power to display

  pinMode(38, OUTPUT);
  digitalWrite(38, 1);//Turns on LCD backlight

  //Initialize serial communication for debugging
  Serial.begin(115200);
  while(!Serial) delay(1);//Wait for serial monitor to connect
  Serial.println("\n=== System start ===");

  //Initialize UART for motor communication
  hwSerial.begin(MOTOR_BAUD, SERIAL_8E1, RXD2, TXD2);
  node.begin(1, hwSerial);

  //Initialize motor controller
  motor.begin();
  motor.setMaxRPM(1000.0f);
  motor.enableSpeedRamp();

  //Initialize TFT display
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(TFT_BLACK);

  //Start the Wi-Fi access point
  web.begin();

  //UI Initialization

  //Left side: connection info
  spriteLeft.setTextColor(TFT_WHITE, TFT_BLACK);
  spriteLeft.createSprite(LCD_WIDTH, LCD_HEIGHT);
  spriteLeft.fillSprite(TFT_BLACK);
  spriteLeft.drawString("Controller", COLUMN_1_X, LINE_1_Y, 4);
  spriteLeft.drawString("SSID: Lathe_control", COLUMN_1_X, LINE_2_Y, 2);
  spriteLeft.drawString("PW: (ingen)", COLUMN_1_X, LINE_3_Y, 2);
  spriteLeft.drawString("IP: " + WiFi.softAPIP().toString(), COLUMN_1_X, LINE_4_Y, 2);
  spriteLeft.pushSprite(0, 0);

  //Right side: system telemetry display
  spriteRight.setTextColor(TFT_WHITE, TFT_BLACK);
  spriteRight.createSprite(LCD_WIDTH, LCD_HEIGHT);

  //RPM display area
  spriteRpm.setTextColor(TFT_WHITE, TFT_BLACK);
  spriteRpm.createSprite(LCD_WIDTH/2 - 20, LCD_HEIGHT/4);

  //Voltage display area
  spriteSupply.setTextColor(TFT_YELLOW, TFT_BLACK);
  spriteSupply.createSprite(LCD_WIDTH / 2 - 20, LCD_HEIGHT / 8);
```

```
}

void loop() {
  unsigned long now = millis();
  web.handleClient();//Handle incoming web commands

  //CLIENT CONNECTION MONITORING
  static bool connectionLost = false;
  static unsigned long lastLossPrint = 0;
  bool clientActive = (millis() - lastClientPing <= 5000);

  //Stop motor if connection is lost
  if (!clientActive && web.isMotorEnabled()) {
    if (!connectionLost) {
      Serial.println("No active client motor stopped!");
      motor.stop();
      connectionLost = true;
      lastLossPrint = now;
    } else if (now - lastLossPrint > 3000) {
      Serial.println("No client motor still stopped");
      lastLossPrint = now;
    }
  } else if (clientActive && connectionLost) {
    Serial.println("Client reconnected");
    connectionLost = false;
  }

  //PERIODIC WEB DATA UPDATE
  if (now - lastWebTemp > WEB_TEMP_UPDATE_TIME) {
    lastWebTemp = now;
    float Tm = motor.readTemperature();//Read motor temperature
    web.setUserPresentationTemperature(Tm);//Update web display
  }

  //Read user commands from web interface
  float reqRPM = web.getUserSetSpeed();
  int dir = web.getFeedDirection();

  //MOTOR MODE CONTROL
  static bool lastMotorState = false;
  static bool lastFeedMode = false;

  bool motorNow = web.isMotorEnabled();
  bool lowSpeedNow = !web.isFastFeedEnabled();

  //Switch motor mode if enabled state or speed mode changed
  if (motorNow != lastMotorState || (motorNow && lowSpeedNow != lastFeedMode)) {
    if (motorNow) {
      motor.enableMotor(lowSpeedNow);//Switch to mode 33 or 35
      Serial.print("Motor ENABLED, mode: ");
      Serial.println(lowSpeedNow ? 35 : 33);
    } else {
      motor.disableMotor();//Disable motor (mode 0)
      Serial.println("Motor DISABLED");
    }
```

```
    lastMotorState = motorNow;
    lastFeedMode = lowSpeedNow;
  }

  //Clear emergency stop if speed is requested
  if (reqRPM > 0.0f && motorNow && !connectionLost) {
    web.clearUserStop();
  }

  //EMERGENCY STOP HANDLING
  bool isStopped = web.isUserStopRequested();
  if (isStopped) {
    motor.emergencyStop();//Engage emergency stop (mode 5)
  }

  //MOTOR SPEED CONTROL
  if (reqRPM > 0.0f && motorNow && !connectionLost) {
    //Perform soft stop before changing direction
    if ((dir != previousDirection) && (abs(motor.getActualSpeed()) > 1.0f)) {
      Serial.println("Soft stop before direction change");
      motor.stop();
      delay(700);
    }
    motor.setSpeed(reqRPM, dir);//Set motor speed and direction
    previousDirection = dir;
  } else if (motorNow && !connectionLost) {
    motor.stop();//Stop motor if no speed is set
  }

  //DISPLAY UPDATE
  if (now - lastScreen > SCREEN_UPDATE_TIME) {
    lastScreen = now;

    float feed = isStopped ? 0.0f : reqRPM;
    float rpm  = motor.getActualSpeed();
    float tq   = motor.getActualTorque();
    float Tm   = motor.readTemperature();
    float Vcc  = motor.readSupplyVoltage();

    String status = isStopped ? "STOP" :
                    (connectionLost ? "LOST" :
                    (motorNow ? (reqRPM > 0.0f ? "RUN" : "IDLE") : "OFF"));

    //Update values for web display
    web.setUserPresentationTorque(tq);
    web.setUserPresentationVoltage(Vcc);

    //Display live motor data
    spriteRight.fillSprite(TFT_BLACK);
    spriteRight.drawString("Actuals", COLUMN_2_X + 30, LINE_1_Y, 4);
    spriteRight.drawString("FEED " + String(rpm, 2) + " [mm/min]", COLUMN_2_X + 30,
LINE_2_Y, 2);
    String dirStr = (dir >= 0) ? "FWD" : "REV";
    spriteRight.drawString("DIR  " + dirStr, COLUMN_2_X + 30, LINE_3_Y, 2);
```

```
    spriteRight.drawString("TORQ " + String(tq, 2) + " [Nm]", COLUMN_2_X + 30, LINE_4_Y,
2);
    spriteRight.drawString("STAT " + status, COLUMN_2_X + 30, LINE_5_Y, 2);
    spriteRight.drawString("TEMP " + String(Tm, 1) + " C", COLUMN_2_X + 30, LINE_6_Y, 2);
    spriteRight.pushSprite(LCD_WIDTH / 2 - 10, 0);

    //Update set feed speed display
    spriteRpm.fillSprite(TFT_BLACK);
    spriteRpm.drawString("Set: " + String(feed, 2) + " mm/min", 5, 0, 2);
    spriteRpm.pushSprite(0, LINE_5_Y);

    //Update supply voltage display
    spriteSupply.fillSprite(TFT_BLACK);
    spriteSupply.drawString("SUPPLY: " + String(Vcc, 1) + " V", 5, 0, 2);
    spriteSupply.pushSprite(0, LINE_6_Y);
  }
}
```

Appendice B: MotionController.cpp

```
//MotionController.cpp
#include "MotionController.hpp"

//Constructor that links a ModbusMaster reference to the controller
MotionController::MotionController(ModbusMaster& modbus)
  : node(modbus)
{}

//Initializes the motor controller's acceleration/deceleration ramp
void MotionController::begin() {
  //Limit acceleration and deceleration to e.g., 300 RPM/s
  float rpmPerSec = 300.0f;

  //Convert RPM/s to internal unit value used by the controller
  uint16_t accVal = uint16_t(rpmPerSec * 4096.0f / 256.0f / 60.0f + 0.5f);

  //Write ramp values to Modbus registers
  node.writeSingleRegister(353, accVal);//RampAccMax
  node.writeSingleRegister(354, accVal);//RampDecMax

  Serial.print("Ramp acc/dec set to: ");
  Serial.println(accVal);
}

//Sets the maximum allowed RPM
void MotionController::setMaxRPM(float rpm) {
  maxRPM = rpm;

  //Convert RPM to ramp value format
  uint16_t rampMax = uint16_t(rpm * 4096.0f / (16.0f * 60.0f) + 0.5f);

  //Write to controller's speed and acceleration limits
  node.writeSingleRegister(350, rampMax);//RampSpeedMax
  node.writeSingleRegister(352, rampMax);//RampAccMax
}

//Enables the motor in speed ramp mode (low or high speed)
void MotionController::enableSpeedRamp(bool lowSpeedMode) {
  uint8_t mode = lowSpeedMode ? 35 : 33;
  node.writeSingleRegister(399, mode);//Set mode
  Serial.print("Mode set to: ");
  Serial.println(mode);
}

//Sets the target speed and direction of the motor
void MotionController::setSpeed(float rpm, int direction) {
  //Limit RPM to maximum allowed value
  if (maxRPM > 0.0f) {
    if (rpm > maxRPM) rpm = maxRPM;
  }

  //Set motor direction using TargetMul register
  int16_t mul = (direction >= 0) ? 1 : -1;
```

```
  node.writeSingleRegister(453, mul);

  //Convert RPM to pulses per second, then to internal format
  float pps = rpm * 4096.0f / 60.0f;
  int32_t tgt = int32_t(pps / 16.0f + 0.5f);

  //Write 32-bit value to TargetInput
  writeInt32(449, tgt);
}

//Stops the motor smoothly using ramp-down
void MotionController::stop() {
  writeInt32(449, 0); // Set target speed to 0
}

//Immediately stops the motor using emergency mode
void MotionController::emergencyStop() {
  node.writeSingleRegister(399, 5);//Mode 5 = Quickstop
}

//Helper function to write a 32-bit integer to two 16-bit Modbus registers
bool MotionController::writeInt32(uint16_t regHi, int32_t val) {
  uint16_t hi = uint16_t(uint32_t(val) >> 16);//High word
  uint16_t lo = uint16_t(uint32_t(val) & 0xFFFF);//Low word
  bool ok_hi = node.writeSingleRegister(regHi,     hi) == node.ku8MBSuccess;
  bool ok_lo = node.writeSingleRegister(regHi + 1, lo) == node.ku8MBSuccess;
  return ok_hi && ok_lo;
}

//Reads motor temperature in Celsius
float MotionController::readTemperature() {
 //Read temperature from register 101
  if (node.readHoldingRegisters(101, 1) == node.ku8MBSuccess) {
    return node.getResponseBuffer(0) * 0.01f;
  }
  return NAN;
}

//Returns actual motor speed in RPM
float MotionController::getActualSpeed() {
  if (node.readHoldingRegisters(201, 1) == node.ku8MBSuccess) {
    int16_t raw = int16_t(node.getResponseBuffer(0));
    return raw * 16.0f * 60.0f / 4096.0f;//Convert from internal format to RPM
  }
  return 0;
}

//Returns actual motor torque in Nm
float MotionController::getActualTorque() {
  if (node.readHoldingRegisters(202, 1) == node.ku8MBSuccess) {
    int16_t raw = int16_t(node.getResponseBuffer(0));
    return raw * 0.001f;//Convert from mNm to Nm
  }
  return 0;
}
```

```cpp
//Reads motor supply voltage in Volts
float MotionController::readSupplyVoltage() {
  if (node.readHoldingRegisters(99, 1) == node.ku8MBSuccess) {
    return node.getResponseBuffer(0) * 0.01f;  // e.g., 234 → 23.4 V
  }
  return NAN;
}

//Enables motor with selected mode (low/high speed)
void MotionController::enableMotor(bool lowSpeedMode) {
  uint8_t mode = lowSpeedMode ? 35 : 33;
  node.writeSingleRegister(399, mode);//Activate motor in selected mode
}

//Disables the motor (turns it off)
void MotionController::disableMotor() {
  node.writeSingleRegister(399, 0);//Mode 0 = OFF
}
```

Appendice C: MotionController.hpp

```
//MotionController.hpp
#ifndef MOTION_CONTROLLER_HPP
#define MOTION_CONTROLLER_HPP

#include <Arduino.h>
#include <ModbusMaster.h>

//MotionController provides a high-level interface to control a motor via Modbus.
//It supports setting speed, direction, reading status, and handling emergency stops.

class MotionController {
public:
 //Constructor: receives a reference to a ModbusMaster instance
  MotionController(ModbusMaster& modbus);

 //Initializes acceleration and deceleration ramps
  void begin();

 //Sets the maximum allowed RPM for the motor
  void setMaxRPM(float rpm);

 //Enables the speed ramp feature; optionally in low-speed mode
  void enableSpeedRamp(bool lowSpeedMode = false);

 //Sets motor speed and direction
  void setSpeed(float rpm, int direction);

 //Performs a smooth stop using acceleration ramp
  void stop();

 //Performs an emergency stop (immediate)
  void emergencyStop();

 //Reads the current motor temperature in Celsius
  float readTemperature();

 //Returns the actual motor speed in RPM
  float getActualSpeed();

 //Returns the actual motor torque in Nm
  float getActualTorque();

 //Reads the motor's current supply voltage
  float readSupplyVoltage();

 //Enables the motor (starts operation) with selected speed mode
  void enableMotor(bool lowSpeedMode);

 //Disables the motor (sets mode to OFF)
  void disableMotor();

private:
  ModbusMaster& node;//Reference to the Modbus communication handler
```

```
  float maxRPM = 0.0f;//User-defined max RPM limit

 //Helper method to write a 32-bit integer value to two Modbus registers
  bool writeInt32(uint16_t regHi, int32_t val);
};

#endif//MOTION_CONTROLLER_HPP
```

Appendice D: WiFiAccessPointServer.cpp

```
//WiFiAccessPointServer.cpp
#include "WiFiAccessPointServer.hpp"
#include <WiFi.h>

unsigned long lastClientPing = 0;//Timestamp of the most recent client "ping"

//Constructor initializes SSID, password, and HTTP server on port 80
WiFiAccessPointServer::WiFiAccessPointServer(const char* ssid, const char* password)
  : _ssid(ssid),
    _password(password),
    server(80)
{}

//Starts the Wi-Fi access point and web server
void WiFiAccessPointServer::begin() {
  WiFi.softAP(_ssid, _password);//Start ESP32 in AP mode
  Serial.print("AP IP address: ");
  Serial.println(WiFi.softAPIP());//Print the local IP address of the AP

  setupRoutes();//Define HTTP request handlers
  server.begin();//Start HTTP server
  Serial.println("Web server started.");
}

//Handles incoming HTTP client requests
void WiFiAccessPointServer::handleClient() {
  server.handleClient();
}

//Defines the HTTP routes and associates them with handler functions
void WiFiAccessPointServer::setupRoutes() {
  server.on("/", HTTP_GET, std::bind(&WiFiAccessPointServer::handleRoot, this));
//Main page
  server.on("/set", HTTP_POST, std::bind(&WiFiAccessPointServer::handleSet, this));
// Handle settings form
  server.on("/ping", HTTP_GET, [this]() {                                    //
Ping route (keeps connection alive)
    lastClientPing = millis();
    server.send(200, "text/plain", "pong");
  });
}

//Handles the main HTML page generation and dynamic UI rendering
void WiFiAccessPointServer::handleRoot() {
  float currentFeed = userSetSpeed;

  String html = "<!DOCTYPE html><html><head><title>ESP32 Lathe
Control</title></head><body>";
  html += "<h1>Lathe Controller</h1>";

 //Display motor statuses
  html += "<p>Motor Temp: " + String(userPresentationTemperature,1) + " &deg;C</p>";
  html += "<p>Torque: " + String(userPresentationTorque,4) + " Nm</p>";
```

```
  html += "<p>Supply Voltage: " + String(userPresentationVoltage,1) + " V</p>";

 //Feed control form
  html += "<form action=\"/set\" method=\"POST\">";

 //Feed mode selection
  html += "<p><strong>Feed Mode:</strong></p>";
  html += "<label><input type=\"radio\" name=\"mode\" value=\"slow\" " +
String(!fastFeedMode ? "checked" : "") + "> Slow (0.24 mm/min)</label><br>";
  html += "<label><input type=\"radio\" name=\"mode\" value=\"fast\" " +
String(fastFeedMode ? "checked" : "") + "> Fast (500 mm/min)</label><br><br>";

 //Feedrate slider based on mode
  if (!fastFeedMode) {
    html += "Feedrate [mm/min]: <input type=\"range\" name=\"feed\" min=\"0\" max=\"0.24\"
step=\"0.01\" value=\"" + String(currentFeed, 2) + "\"
oninput=\"this.nextElementSibling.value = this.value\">";
    html += "<output>" + String(currentFeed, 2) + "</output> mm/min<br><br>";
  } else {
    html += "Fast Feedrate [mm/min]: <input type=\"range\" name=\"feed\" min=\"0\"
max=\"500\" step=\"10\" value=\"" + String(currentFeed, 0) + "\"
oninput=\"this.nextElementSibling.value = this.value\">";
    html += "<output>" + String(currentFeed, 0) + "</output> mm/min<br><br>";
  }

 //Direction selection
  html += "Direction:<br>";
  html += "<label><input type=\"radio\" name=\"dir\" value=\"fwd\" " + String(feedDirection
== 1 ? "checked" : "") + "> Forward</label><br>";
  html += "<label><input type=\"radio\" name=\"dir\" value=\"rev\" " + String(feedDirection
== -1 ? "checked" : "") + "> Reverse</label><br><br>";

 //Submit button
  html += "<input type=\"submit\" value=\"Set Feed\">";
  html += "</form>";

 //Emergency STOP button
  html += "<form action=\"/set\" method=\"POST\" style=\"margin-top:20px;\">";
  html += "<input type=\"hidden\" name=\"stop\" value=\"1\">";
  html += "<input type=\"submit\" value=\"EMERGENCY STOP ONLY\" ";
  html += "style=\"background:red; color:white; font-size:20px; padding:12px 24px;
border:none; border-radius:6px; cursor:pointer;\">";
  html += "</form>";

 //Motor ON/OFF toggle
  html += "<form action=\"/set\" method=\"POST\" style=\"margin-top:20px;\">";
  html += "<input type=\"hidden\" name=\"toggle_motor\" value=\"1\">";
  html += "<input type=\"submit\" value=\"MOTOR: " + String(motorEnabled ? "ON" : "OFF") +
"\" ";
  html += "style=\"background:" + String(motorEnabled ? "green" : "gray") + "; color:white;
font-size:20px; padding:12px 24px; border:none; border-radius:6px; cursor:pointer;\">";
  html += "</form>";

 //JavaScript ping to keep connection alive
  html += "<script>";
```

```
  html += "setInterval(() => { fetch('/ping'); }, 1000);";
  html += "</script>";


 //Reset slider to 0 if emergency stop is active
  if (isUserStopRequested()) {
    html += "<script>";
    html += "document.addEventListener('DOMContentLoaded', function() {";
    html += "  const slider = document.querySelector('input[name=\"feed\"]');";
    html += "  const output = slider.nextElementSibling;";
    html += "  if (slider) { slider.value = 0; output.value = 0; }";
    html += "});";
    html += "</script>";
  }

  html += "</body></html>";

 //Send complete HTML to client
  server.send(200, "text/html", html);
}

//Handles POST requests to /set – parses form data and updates internal state
void WiFiAccessPointServer::handleSet() {
 //Emergency STOP handler
  if (server.hasArg("stop")) {
    userRequestedStop = true;
    userSetSpeed = 0.0f;
    Serial.println("Web: EMERGENCY STOP triggered!");
    server.sendHeader("Location", "/", true);
    server.send(302, "text/plain", "");
    return;
  }

 //Feed mode handler (must be processed before motor control)
  if (server.hasArg("mode")) {
    String mode = server.arg("mode");
    fastFeedMode = (mode == "fast");
    Serial.print("Feed mode set to: ");
    Serial.println(fastFeedMode ? "FAST" : "SLOW");
  }

 //Toggle motor ON/OFF
  if (server.hasArg("toggle_motor")) {
    motorEnabled = !motorEnabled;
    Serial.print("Motor toggled to: ");
    Serial.println(motorEnabled ? "ON" : "OFF");
  }

 //Feedrate input
  if (server.hasArg("feed")) {
    float feed = server.arg("feed").toFloat();
    userSetSpeed = feed;
    Serial.print("Web: requested feed = ");
    Serial.print(feed);
    Serial.println(" mm/min");
```

```
  }

 //Direction input
  if (server.hasArg("dir")) {
    String dir = server.arg("dir");
    feedDirection = (dir == "rev") ? -1 : 1;
    Serial.print("Direction set to: ");
    Serial.println(dir);
  }

 //Redirect back to main page
  server.sendHeader("Location", "/", true);
  server.send(302, "text/plain", "");
}

//=== Getters and setters for internal state ===

float WiFiAccessPointServer::getUserSetSpeed() const {
  return userSetSpeed;
}

void WiFiAccessPointServer::setUserPresentationTemperature(float temp) {
  userPresentationTemperature = temp;
}

void WiFiAccessPointServer::setUserPresentationTorque(float tq) {
  userPresentationTorque = tq;
}

void WiFiAccessPointServer::setUserPresentationVoltage(float v) {
  userPresentationVoltage = v;
}


bool WiFiAccessPointServer::isUserStopRequested() const {
  return userRequestedStop;
}


void WiFiAccessPointServer::clearUserStop() {
  userRequestedStop = false;
}

int WiFiAccessPointServer::getFeedDirection() const {
  return feedDirection;
}


void WiFiAccessPointServer::setFeedDirection(int dir) {
  feedDirection = (dir >= 0) ? 1 : -1;
}

bool WiFiAccessPointServer::isFastFeedEnabled() const {
  return fastFeedMode;
}
```

```
bool WiFiAccessPointServer::isMotorEnabled() const {
  return motorEnabled;
}
```

Appendice E: WiFiAccessPointServer.hpp

```cpp
//WiFiAccessPointServer.hpp
#ifndef WIFI_ACCESS_POINT_SERVER_HPP
#define WIFI_ACCESS_POINT_SERVER_HPP

#include <Arduino.h>
#include <WebServer.h>

//WiFiAccessPointServer sets up a local Wi-Fi network (Access Point mode)
//and hosts a web interface to monitor and control motor parameters.
class WiFiAccessPointServer {

public:
  //Constructor: initializes with SSID and password for the Wi-Fi network
  WiFiAccessPointServer(const char* ssid, const char* password);

  //Starts the access point and web server
  void begin();

  //Handles incoming HTTP client connections
  void handleClient();

  //=== Interface for main.cpp ===

  //Returns the feed speed set by the user
  float getUserSetSpeed() const;

  //Updates the displayed motor temperature on the web page
  void setUserPresentationTemperature(float temp);

  //Updates the displayed motor torque on the web page
  void setUserPresentationTorque(float tq);

  //Updates the displayed supply voltage on the web page
  void setUserPresentationVoltage(float v);

  //Returns true if the user has triggered an emergency stop
  bool isUserStopRequested() const;

  //Clears the emergency stop flag (e.g., after user sets a new speed)
  void clearUserStop();

  //Returns the desired feed direction (1 = forward, -1 = reverse)
  int getFeedDirection() const;

  //Sets the feed direction from external logic
  void setFeedDirection(int dir);

  //Returns true if the user has selected fast feed mode
  bool isFastFeedEnabled() const;

  //Returns true if the motor is currently enabled
  bool isMotorEnabled() const;
```

```
private:
  //Internal helpers

  //Configures HTTP routes for handling requests
  void setupRoutes();

  //Handles the root web page ("/")
  void handleRoot();

  //Handles user input via form submission ("/set")
  void handleSet();
 //=== Internal state ===

  const char* _ssid;//Wi-Fi SSID
  const char* _password;//Wi-Fi password
  WebServer   server;//HTTP server instance

 //Values received from the user via web interface
  float userSetSpeed = 0.0f;

 //Values presented to the user on the web page
  float userPresentationTemperature = 0.0f;
  float userPresentationTorque = 0.0f;
  float userPresentationVoltage = 0.0f;

  bool userRequestedStop = false;//Emergency stop flag
  int  feedDirection = 1;//Feed direction: 1 = forward, -1 = reverse
  bool fastFeedMode = false;//Feed mode flag: true = fast, false = slow
  bool motorEnabled = false;//Motor enable flag
};

#endif // WIFI_ACCESS_POINT_SERVER_HPP
```

Appendice F: platformio.ini

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32-c3-devkitc-02]
platform = espressif32
board = lilygo-t-display-s3
framework = arduino
monitor_speed = 115200
lib_deps =
    4-20ma/ModbusMaster@^2.0.1
    bodmer/TFT_eSPI@^2.5.43
build_flags =
    ;-DARDUINO_USB_MODE=1
    ;-DARDUINO_USB_CDC_ON_BOOT=1
    -UARDUINO_USB_CDC_ON_BOOT
;build_flags =
    ; Enable -DARDUINO_USB_CDC_ON_BOOT will start printing and wait for terminal access
during startup
    ; -DARDUINO_USB_CDC_ON_BOOT=1

    ; Enable -UARDUINO_USB_CDC_ON_BOOT will turn off printing and will not block when using
the battery
   ;-UARDUINO_USB_CDC_ON_BOOT
```