Tecnológico de Monterrey

Instituto Tecnológico de Estudios Superiores de Monterrey
Mexico City Campus

Challenge Technical Document

Courses

Troubleshooting Processes	**MR2005B.601**
Industrial Automation	**MR2006B.601**

Students:

Frida Sophia Chávez Juárez	**A01665457**
Victor Manuel Galicia Hernández	**A01666471**
Oscar Gadiel Ramo Martínez	**A01665807**
Leonardo Valero Perales	**A01658513**
Karl Maximilian Waldthausen Botello	**A01660005**

Instructors:

David Navarro Durán	**PLCs Module**
Israel Ulises Cayetano Jiménez	**Microcontrollers Module**
Ricardo Ganém Corvera	**Thermofluids Module**
Rodrigo Regalado García	**Applied Electronics Module**

December 7, 2025

Abstract

In this project, we present the integration between an automated viscosity adjustment system and a conveyor belt design. Building upon the conceptual station design developed in our first deliverable, "Challenge Deliverable," and the functional specifications for the viscometer and dosing system defined in the complementary requirements, we propose a fully coordinated architecture capable of detecting, measuring, adjusting, transporting, and classifying liquid samples based on predefined viscosity ranges. Our system combines a PLC-driven bottle-handling subsystem with a microcontroller-based viscometer that performs torque-dependent viscosity estimation, iterative water dosing, and intelligent stirring routines. The solution incorporates UART communication between controllers, safety mechanisms, sensor-based decision making, and a LabVIEW Human Machine Interface for monitoring and supervisory control. This technical report outlines the conceptual integration of all mechanical, electronic, and software elements that guided the manufacturing, programming, and validation phases of the project.

Keywords: Automated Viscosity Control, Mechatronic, System Integration, Sensor-Based Decision Making, PLC–Microcontroller Communication, LabVIEW Human-Machine Interface

Contents

1. Introduction

1.1. Problematic Context

Industrial automation plays a central role in modern manufacturing environments, where efficiency, repeatability, and product quality depend on the coordinated interaction of mechanical, electrical, and computational subsystems. In industries that handle viscous fluids such as cleaning products, cosmetics, food processing, and chemical manufacturing, viscosity becomes a critical property to control, as it directly affects product performance, consistency, and user experience. Automated viscosity-adjustment systems, therefore, represent a relevant application of mechatronic engineering, allowing the measurement, correction, and classification of fluid properties with minimal human intervention while ensuring reliability and traceability.

In this project, we address the challenge of designing a fully integrated station capable of transporting containers, measuring their viscosity, performing controlled dilution cycles when required, and classifying each container according to predefined viscosity ranges. The system brings together two major subsystems: a PLC-based automated handling and sorting module that moves each bottle through the different stages of the process by a conveyor belt, and a microcontroller-based viscometer subsystem responsible for measuring the viscosity of the container, iterative adjustment, and automated cleaning routines of the spindle used to measure. Achieving seamless coordination between these subsystems is central to the problem we aim to solve.

One of the main difficulties lies in ensuring that the mechanical, electronic, and computational components operate reliably as a single automated unit. Any error in detection, communication, timing, or actuation may result in misclassification, inaccurate viscosity readings, or mechanical interference. For this reason, we developed a unified integration strategy that aligns the behaviour of all actuators, sensors, and controllers under both manual and automatic modes.

Another main difficulty is the coordination between the different microcontrollers, as there cannot be a direct communication between them using the protocols reviewed in class, such as I2C, UART, Bluetooth, or another communication method. One of the main applications of the system was to make a modularization of the stations, so if one of them fails, there won't be the coordination we need within the systems and the final integration won't be complete.

Our project objectives guide this integration by establishing the expected performance of the station, the functional role of each subsystem, and the criteria for successful operation. These objectives include developing a reliable bottle handling mechanism, implementing a Brookfield-based viscometer with dosing capabilities, establishing robust communication between controllers, incorporating safety and feedback mechanisms, and deploying a supervisory interface for monitoring and control. The scope of this report focuses on the conceptual and preliminary design stages that

will support subsequent phases of manufacturing, programming, and testing. This includes the structural design of the bottle handling and sorting system, the definition of the viscometer's operational logic, the selection and justification of actuators and sensors, the communication strategy between the PLC and microcontroller, and the integration of safety elements and human–machine interaction.

Finally, this introduction contextualizes the broader engineering foundations involved in the project. Concepts such as motor actuation, position sensing, PID control, serial communication, electronic signal conditioning, and thermo fluid properties, including viscosity measurement and volumetric flow, are essential for understanding how the station operates as a cohesive automated system. Together, these elements establish the technical framework for the design, development, and analysis presented in the following sections.

1.2. Objectives

1.2.1 General Objective

To design and integrate an automated mechatronic system capable of transporting, measuring, adjusting, and classifying liquid samples according to viscosity specifications by combining PLC driven handling mechanisms with a microcontroller based viscometer and a supervisory LabVIEW HMI.

1.2.2 Specific Objectives

- Develop a coordinated bottle handling subsystem capable of dispensing, transporting, positioning, and sorting containers according to the system workflow.
- Implement a microcontroller based viscometer capable of torque derived viscosity measurement, iterative dilution control, and automated cleaning routines.
- Establish a reliable UART communication interface between the PLC system and the microcontroller to synchronize container flow, measurement cycles, and process status.
- Integrate all required sensors, actuators, and safety mechanisms including emergency stops, visual indicators, and feedback devices into a unified automation architecture.
- Design and configure a LabVIEW based human machine interface (HMI) that enables real time monitoring, manual and automatic operation modes, and system alarms.

1.3. Technical Foundations of the System

1.3.1 PLCs

Programmable Logic Lontrollers are rugged, industrial computers used to automate and control manufacturing processes, machines, and other activities. They function by reading inputs from sensors, executing a pre-programmed set of rules, and then sending commands to output devices to perform tasks, such as starting a motor or opening a valve. PLCs have largely replaced older relay-based control systems because they are more reliable, easier to program and troubleshoot, and allow for changes to be made through software instead of rewiring. [1] They are usually programmed in ladder diagrams (LD) as opposed to traditional coding languages like Python and C as these are harder to troubleshoot quickly since they require more education and training as opposed to the visually intuitive design of LDs.

Most commercial PLC programming software like Siemen's TIA Portal and Rockwell's Studio 5000 is costly and often limited to the manufacturer's own ecosystem. An free, open-source alternative is OpenPLC which allows users to program their own PLCs in a plethora of different languages and then upload them to commonly used prototyping boards and microcontrollers like Arduinos and ESPs. Due to its free, open source, and wide availability, this is the software that will be used for the system's PLC programming.

1.3.2 Actuators

The actuators are the components of a machine that are responsible for moving and controlling a mechanism or system. They are *transducers* that convert energy (e.g., Electrical, Mechanical, Fluid Pressure, Thermal, Chemical, Magnetic) into motion [2].

They can be classified into [2]:

- Energy received: Electric, Hydraulic, Pneumatic.
- Motion type: Linear or rotary
- Materials used: Rigid or soft.

A **Brush DC Motor** is an electric motor that converts direct current (DC) power into mechanical energy using a simple, internally-mounted mechanical commutation system involving brushes and a commutator [3]. The working principle is based on the Electromagnetic induction principle: The commutator contacts carbon brushes that apply current to the motor's armature. Current flows through the armature windings, creating a magnetic field. Permanent magnet or field windings produce a stationary field that interacts with this magnetic field. This interaction generates torque, which in turn causes the armature to coil [4].

They can be classified into [2]:

- **Permanent Magnet**: Uses a permanent magnet in the stator to provide the fixed magnetic field, eliminating the need for a separate field winding and power supply [5].
- **Shunt**: The field winding is connected in parallel with the armature winding. The field winding has high resistance and many turns of fine wire, so the field current is nearly constant and independent of the armature current [5].
- **Series**: Here, the field winding is connected in series with the armature winding. The field winding has low resistance and fewer turns of thick wire, designed to carry the full armature current [5].
- **Compound**: It combines the characteristics of both series and shunt motors by incorporating both a series and a shunt field winding. This hybrid design offers a balance of their features [5].

A stepper motor is a brushless DC motor that converts digital electrical pulses into precise, discrete mechanical angular movements, also called steps. This design allows for extremely accurate positioning control, typically without needing a feedback sensor. The working principle of a stepper motor is based on the interaction of electromagnetic fields: a motor controller sequentially energizes the stationary coils (stator phases), creating a rotating magnetic field. The rotor, which is either a permanent magnet or a toothed piece of iron, is attracted to this energized magnetic field and aligns itself with the active stator poles. By precisely timing the sequence of electrical pulses sent to the stator phases, the rotor moves in incremental, discrete angular steps, allowing for accurate positioning control without continuous mechanical commutation [6].

They can be classified into [2]:

- **Permanent Magnet (PM)**: The rotor is a permanent magnet that aligns with the magnetic field generated by the stator circuit. This solution guarantees a good torque and also a detent torque. This means the motor will resist, even if not very strongly, to a change of position regardless of whether a coil is energized. The drawbacks of this solution are that it has a lower speed and a lower resolution compared to the other types [7].
- **Variable Reluctance (VR)**: The rotor is made of an iron core and has a specific shape that allows it to align with the magnetic field. With this solution, it is easier to reach a higher speed and resolution, but the torque it develops is often lower, and it has no detent torque [7].
- **Hybrid Synchronous (HB)**: This kind of rotor has a specific construction and is a hybrid between permanent magnet and variable reluctance versions. The rotor has two caps with alternating teeth and is magnetized axially. This configuration allows the motor to have the advantages of both the permanent magnet and variable reluctance versions, specifically high

resolution, speed, and torque. This higher performance requires a more complex construction, and therefore a higher cost [7].

1.3.3 Sensors

A sensor is an element of a measuring system that is directly affected by a phenomenon, body, or substance carrying a quantity to be measured. It is a type of transducer that converts the physical phenomenon into an electrical signal [8].

They can be classified by [8]:

- Measurand (e.g., length, position, velocity, angular velocity, mass, flow, temperature, electric current, level, current, force, light intensity, image, geolocation, etc.).
- Application (e.g., automotive, industrial, consumer, medical, aerospace).
- Sensor mean or principle (e.g., mechanical, optical, capacitive, inductive, resistive).
- Sensor characteristics (e.g., passive or active, analog or digital output signal, discrete or integrated, proprioceptive or exteroceptive, MEMs).

The sensor signal can be classified into [8]:

- **Analog**: Translate the magnitude of a physical phenomenon into a continuous, proportionally equivalent signal.
- **Digital**: They only have two states, ON and OFF. The sensor triggers a signal when a specific threshold is crossed and is often used in applications where the presence or absence of a particular condition is monitored.

The resulting signal can be sent via a communication protocol or expressed directly via an output pin as a HIGH-LOW or PWM signal.

Ultrasonic sensors primarily work on the Time-of-Flight (ToF) principle. They act like miniature SONAR (Sound Navigation and Ranging) systems, measuring the time it takes for sound waves to travel to an object and return to calculate the distance.

The transducer emits a high-frequency sound pulse. This sound travels through the air at the speed of sound until it hits an object and bounces back as an echo. The sensor measures the elapsed time (ToF) between emission and reception [9]. The distance is calculated using the formula:

$$Distance = \frac{Speed of Sound \cdot Time of Flight}{2} \tag{1}$$

Temperature variations affect the speed of sound, so some advanced sensors include internal temperature compensation for increased accuracy. [9]

Colour sensors detect the intensity of light at specific wavelengths to identify the colour of a surface in ambient light. Most of the colour sensors utilize a white light source to illuminate the target object. The object absorbs some wavelengths and reflects others based on its colour. The reflected light passes through internal Red, Green, and Blue (RGB) filters, and is measured by photodiodes (light detectors). The sensor then analyzes the intensity ratio of the RGB components to determine the specific colour value [10].

A motor encoder is an electromechanical sensor that attaches to a motor's shaft to provide real-time feedback on its rotational position, speed, and direction. This feedback is crucial for open-loop systems, allowing a control system (like a PLC or microcontroller) to precisely monitor and adjust the motor's performance in real-time [11].

They are divided by:

- **Motion**: Rotary, Linear
- **Sensing Tech**: Magnetic, Optical
- **Feedback Signal Output**: Incremental, Quadrature, Absolute

The quadrature encoder is a specific type of incremental encoder that relies on having two output channels (A and B) where the electrical signals are precisely 90 degrees out of phase with each other. This phase shift creates a unique sequence of signal transitions that allows a decoding circuit to monitor which channel leads the other. By tracking the order of these transitions, the system can unambiguously determine both the speed and direction of the movement. Furthermore, the use of two channels in quadrature allows for higher resolution measurement of position than a single channel could provide, typically quadrupling the effective pulse count. [12]

1.3.4 Control

A PID controller is an instrument that receives input data from sensors, calculates the difference between the actual value and the desired setpoint, and adjusts outputs to control variables such as the velocity or the position of a motor. It does this through three mechanisms: [13]

- **Proportional Control**: Reacts to current error
- **Integral Control**: Addresses accumulated past errors
- **Derivative Control**: Predicts future errors

The PID controller sums those three components to compute the output. This architecture allows PID controllers to efficiently maintain process control and system stability [13].

The **proportional gain** K_p determines the ratio of output response to the error signal. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate. If K_p is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control. [13].

The **integral response** sums the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State error to zero. Steady-state error is the final difference between the process variable and the set point. A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero. [13].

The **derivative component** causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Increasing the derivative time (K_d) parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use very small derivative time (K_d), because the Derivative Response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control system unstable. [13].

1.3.5 Communication

Universal Asynchronous Receiver-Transmitter ($UART$) allows the communication between devices via a USB-UART converter (COM port). It start and stop bits on each package (overhead). It is Full-Duplex, uses asynchronous time, uses the RX, TX, and GND pins, can go up to 1500000 bauds per second, can go up to 10 meters, but can be expanded by using RS-485 to up to 1000 kilometers, and connect one-to-one devices, but can be expanded up to 1-to-32 using network topologies [14].

The **Inter-Integrated Circuit** ($I2C$) relies on addressing (Slave/Node address, data register address), frames can be as long as required, and "Acknowledge" allows to verify data communication. It is Half-Duplex, Synchronous, uses SDA-SCL, and GND wiring, can go to 1000 kHz, up to 1 meter of distance, and can connect single-multiple main and up to 127 or 1023 nodes [14].

Bluetooth is a wireless protocol that exchanges data over short distances using UHF radio waves in the 2.4 GHz frequency band. Its power is medium, latency is 50 ms, can to up to 3 Mbps,

its network topology is Piconet, and its family is 802.15.1 [14].

1.3.6 Applied Electronics: Operational Amplifiers (Op-Amps)

An Operational Amplifier (Op-Amp) is a high gain, integrated analog circuit designed to allow a wide variety of useful electronics circuits to be built. Internally, it is composed of a large number of transistors which control currents and voltages to achieve its specific electrical characteristics. [17]

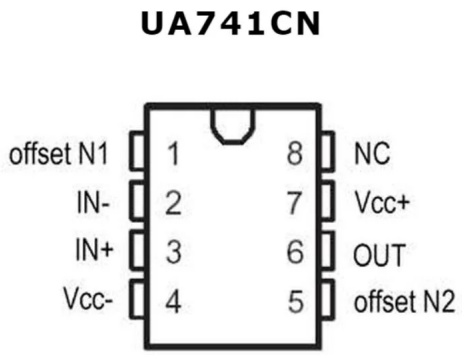


Figure 1: Most common Op-Amp Pinout.

For simplified analysis, the ideal Op-Amp model assumes the following properties:

- **Ideal Characteristics and Operational principles.**
  - **Infinite Input Impedance** ($Z_{in} = \infty$): No current flows into the input terminals ($I_{in} = 0$).
  - **Zero Output Impedance** ($Z_{out} = 0$): The output can drive any load without voltage drop.
  - **Infinite Open-Loop Gain** ($A_{OL} = \infty$).
  - **Infinite Bandwidth** ($BW = \infty$).
- **Operational Principles (Negative Feedback):**
  - **Virtual Short Circuit:** With negative feedback applied, the voltage difference between the inverting terminal (V_n) and the non-inverting terminal (V_p) tends towards

zero:

$$V_d = V_p - V_n \approx 0 \quad \Rightarrow \quad V_p \approx V_n$$

– **Virtual Ground:** If the non-inverting terminal is connected to ground ($V_p = 0$), then the inverting terminal is also at ground potential ($V_n \approx 0$).

Signal conditioning equation

The primary goal of signal conditioning in a linear electronic system is to map a measurable input variable (x) to a usable output voltage (Y) via a linear transformation. This relationship is defined by the Conditioning Equation, which resembles the slope-intercept form of a line:

$$Y = mx + k$$

Where:

- Y: The system **Output Voltage** (V_{out}).
- m: The **Gain** (A_v) of the conditioning stage. This factor scales the input signal.
- x: The **Input Voltage** (V_{in} or differential input V_d).
- k: The **Offset Voltage** (or V_{rest}, defined as the subtracted voltage in the differential stage). This value shifts the entire signal range.

Operational amplifier circuits are designed to precisely control the values of m and k through the relation between external resistors.

Principal Op-Amp configurations

A. Non-Inverting Amplifier: The **Non-Inverting Amplifier** is used for **voltage amplification** where the output signal (V_{out}) is in phase with the input signal (V_{in}) [17]. Its main advantage is its extremely **high input impedance**. The voltage gain (A_v) is determined by the feedback resistor (R_f) and the gain-setting resistor (R_1):

$$A_v = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_1}$$

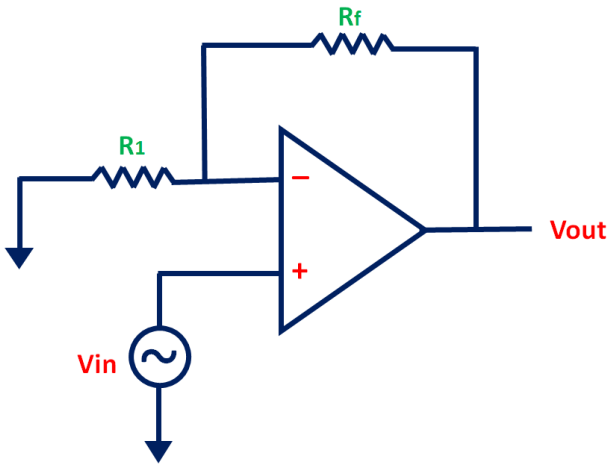
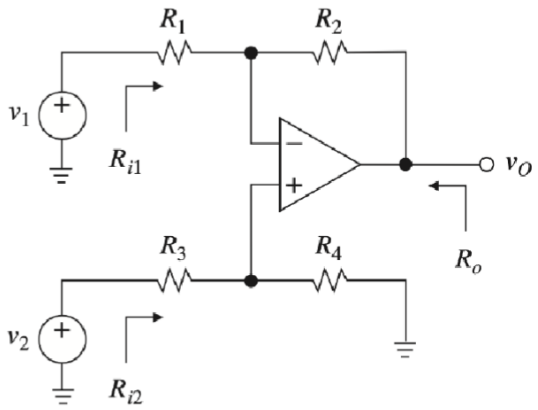


Figure 2: Non-inverting Op-Amp Configuration

B. Differential Amplifier (Subtractor): The **Differential Amplifier** is used to amplify the voltage difference between two input signals (V_2 and V_1). Assuming a matched resistor network ($R_1 = R_3$ and $R_2 = R_4$), the output voltage (V_{out}) is proportional to the difference between the two inputs:

$$V_{out} = \left(\frac{R_2}{R_1}\right)(V_2 - V_1)$$



$$v_O = \frac{R_2}{R_1}\left(\frac{1 + R_1/R_2}{1 + R_3/R_4}v_2 - v_1\right)$$

Si R2/R1 = R4/R3

$$vo = \frac{R_2}{R_1}(v_2 - v_1)$$

$$v_d = v^+ - v^-$$

$$vo = \frac{R_2}{R_1}(v_d)$$

Figure 3: Differential Op-Amp configuration

Op-Amp Saturation.

Saturation is a non-linear operating condition where the Op-Amp's output voltage attempts to exceed the limits imposed by the DC power supply rails (V_{CC+} and V_{CC-}). The output is

driven to its limit, or "clipped," at the saturation voltage (V_{sat}). The output signal is distorted, and the device ceases to operate as a linear amplifier.

1.3.7 Printed Circuit Board (PCB)

A **Printed Circuit Board (PCB)** is the mechanical support structure used to electrically connect and mechanically support electronic components using conductive pads, tracks, and features etched from copper sheets laminated onto a non-conductive substrate material, typically. The main purposes are to replace point-to-point wire connections and to reduce the size of circuits [18].

Anatomy of a PCB A PCB is constructed from multiple layers and features.

- **Substrate:** The non-conductive core material, typically fiberglass-epoxy.
- **Copper:** The conductive layers etched to form **traces** (tracks) and **planes** (large areas for power or ground).
- **Soldermask:** A polymer layer (usually green) that coats the board to prevent accidental solder bridges between traces.
- **Silkscreen:** A layer (usually white) used for placing component legends, designators, and symbols.
- **Pad:** A copper area where a component lead is soldered.
- **Via:** A plated hole that provides an electrical connection between different copper layers of the PCB.

Trace width and Thickness The physical dimensions of the copper tracks are critical for the reliability and performance of the system:

- **Trace Width (w):** Primarily determines the **current carrying capacity** and DC resistance. Wider traces are essential for high-current paths to minimize heating and voltage drop. Dimensions are generally expressed in **mils** or **thou** (thousandths of an inch), where 1 mil = 25.4 μm.
- **Copper Thickness (t):** Measured in ounces per square foot (oz/ft^2). Standard thickness is 1 oz ($\approx$ 35 μm), crucial for high-power applications, corresponding to approximately 1.4 mils.

1.3.8 Thermofluids

Viscosity quantifies how difficult it is to move a fluid or resist flow, and when a fluid/solid is subjected to shear stress, it undergoes deformation. All these three concepts can be related to the **Newton's Law of Viscosity**, which states that [15]:

$$\tau = \mu \cdot \frac{du}{dy} \tag{2}$$

In which μ is the dynamic viscosity. τ the shear force, and $\frac{du}{dy}$ is the velocity gradient.

Volumetric flow rate (Q) is the volume of a fluid that passes through a specific area per unit of time [16]. Its formula is:

$$Q = \frac{\Delta(volume)}{\Delta(t)} \tag{3}$$

1.4. Materials and Software Used

Component	Quantity
GM 25-370 Motor with Encoder 12 V DC 330 RPMs	3
5840-31ZY Motorreductor 12V 260 RPM	6
JSS57HS76 Stepper Motor	1
Lineal Actuator 12 V 150 KgF 100 mm	1
Lineal Actuator 12 V 20 KgF 200 mm	3
L298N H-Bridge Motor Driver	5
L293D H-Bridge Motor Driver	2
Microstep Driver TB6600	1
Small Water Pump	1
Fan DC motor	1
RGB LED	4
HC-SR04 Ultrasonic Sensor	1
FC - 51 IR Sensor	24
TCS3475 Color Sensor	2
Protoboard	1
PCB	4
ESP32D	1
Arduino Mega	3
Arduino UNO	1
screw terminal blocks	50
WAGO 50 22 AWG solid-core cable 50 m	50
22 AWG braided cable	50 m
1/8 in MDF 2x1 m planks	2
Flex Glue 370 ml	3
Roller bearings	24
9/16 in aluminum rod	1 m
1 in hollow aluminum rod	1 m

Table 1: Materials used

Software Used
Visual Studio Code
MATLAB
Festo FluidSim
OpenPLC
Arduino IDE
Proteus 9 Professional

Table 2: Software used

2. Development

The development phase of this project focuses on the complete design, integration, and justification of the three major subsystems that compose the automated viscosity adjustment station. These subsystems work together sequentially, forming a continuous flow in which each bottle is dispensed, transported, analyzed, adjusted if necessary, and finally sorted according to its viscosity classification.

The first subsystem, the *Bottle Stack Dispenser*, is responsible for introducing containers into the processing line one at a time. It is composed of three lines, each representing the three different containers with the different colors to represent the three different viscosities. They are directed to the main conveyor belt that takes the containers from the bottle stack dispenser station to the measurement station - viscometer system.

The second subsystem, the *Viscometer System*, constitutes the analytical and corrective core of the station. It measures the viscosity of each sample using spindle torque estimation and encoder feedback. When the measured viscosity exceeds acceptable limits, a dosing pump injects a predefined amount of water into the bottle to correct the mixture. This requires precise actuation, iterative decision making, and safe mechanical motion.

Finally, the third subsystem, the *Box Sorting System*, classifies each bottle based on its final viscosity range. A set of three linear actuators redirects bottles into corresponding collection boxes. This ensures that each processed sample is grouped according to its measured properties. Across all subsystems, the development process includes the selection of sensors and actuators, mechanical feasibility through CAD modeling, PLC microcontroller communication, and the implementation of safety features. Together, these components form a modular and scalable automated station capable of performing viscosity measurement and classification reliably.

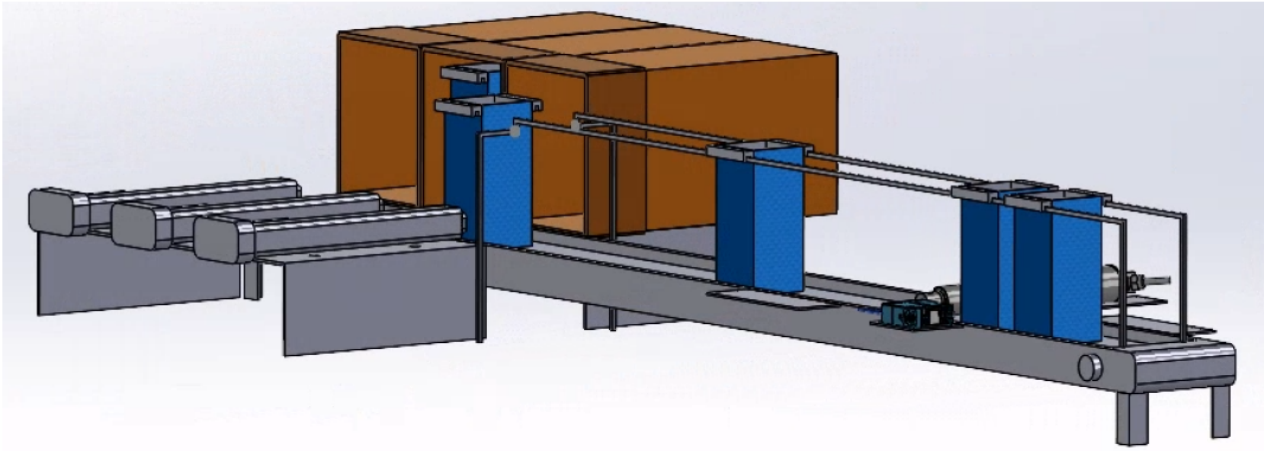


Figure 4: First CAD draft with all the stations united

2.1. Bottle Stack Dispenser

2.1.1 Mechanical Design and Construction

The design was based on the container we were going to use, a 7.00 x 7.00 cm square base cylinder. Since the stack system was primarily used to transport containers to the viscometer station, a conveyor belt was designed, varying only in the belt's longitude. In total, four conveyor belts were made: three of the same size with a capacity of storing five containers, each in a different color, and one that would transport the containers to the second station. The measures were thought out in the material we had available for previous projects to save spending and actually use the left ones, and with commercial sizes to ease the manufacturing. The CAD design obtained is shown in the next images.

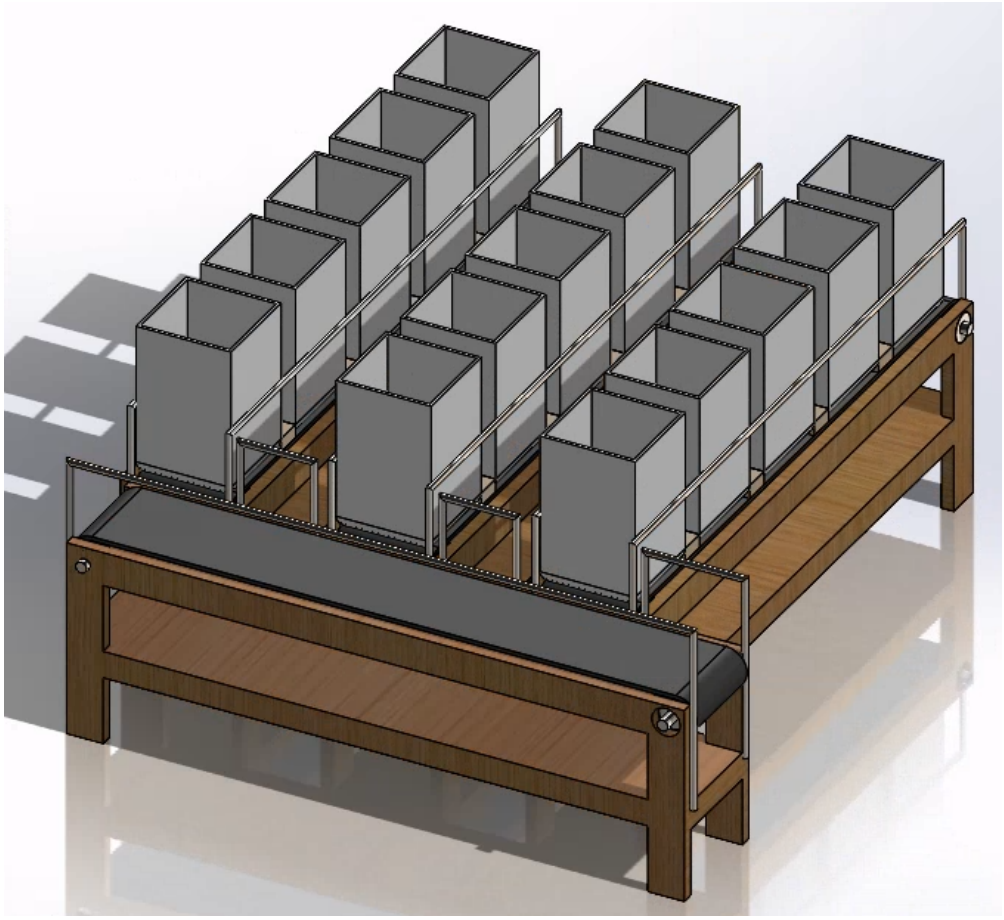


Figure 5: Trimetric view of the Stack Station.

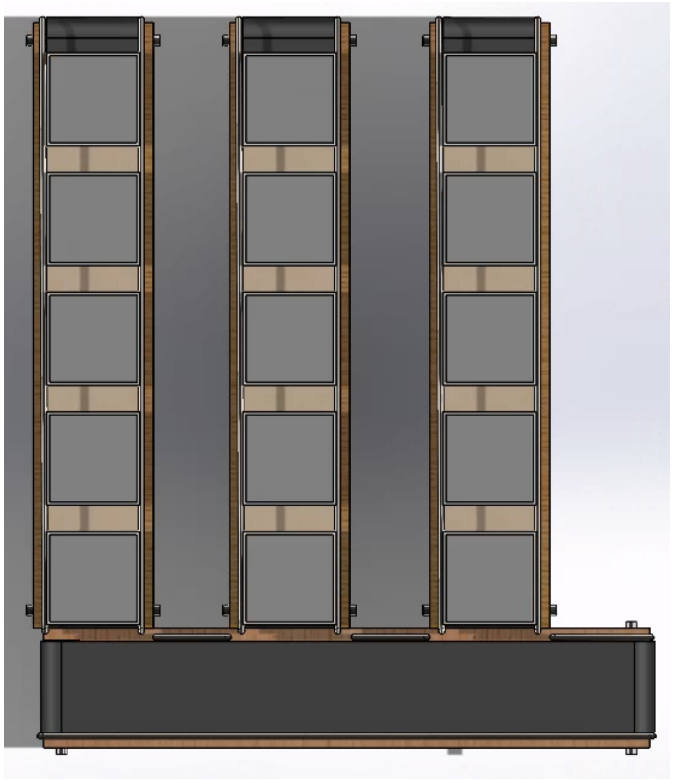


Figure 6: Top view of the Stack Station

To see the exact measures, go to Appendix A

For the manufacturing, we developed a robust mechanical conveyor structure built primarily from MDF. This material was selected due to its rigidity, machinability, and dimensional stability, which allowed us to fabricate straight, well-aligned frames that ensured smooth interaction between the bottles, the belt system, and the FC-51 infrared sensors.

Each conveyor lane required a reliable motion mechanism; for this reason, we manufactured two aluminum rollers for every lane. These rollers were machined to a uniform diameter and drilled axially to allow secure mounting. One roller was permanently fixed to the belt using welding or strong adhesive, enabling it to transmit torque directly from the motor, thus serving as the driven element of the system.

The second roller was mounted on ball bearings to minimize friction and promote smooth belt rotation. This driven–free-roller configuration provided stable, low-resistance movement and reduced mechanical stress on the motor, ensuring consistent and reliable conveyor performance.

All structural elements (MDF panels, roller supports, motor brackets, and sensor mounts) were assembled using specialized MDF adhesive, rivets, and nails. This hybrid joining method ensured both strength and long-term durability, preventing misalignment or loosening during extended operation. The resulting construction provided a solid mechanical foundation that complemented

the infrared detection system and enabled the bottle-handling subsystem to operate accurately and efficiently.

Figure 7: Construction model for the conveyor belt.

2.1.2 Electronics

To ensure continuous operation, an infrared proximity sensor monitors the presence of bottles at the release point. If no bottle is detected within a predefined interval, an LED indicator alerts the operator to reload the stack. The mechanical frame is manufactured using rigid materials to maintain alignment and prevent jamming, ensuring consistent bottle flow. To drive the conveyor motion consistently across all lanes, we selected a 12V DC gear motor model 5840-31ZY, operating at 260 RPM. This motor provided the necessary torque to rotate the aluminum rollers and maintain steady belt movement even under load. Using the same motor model for every conveyor ensured uniform performance, simplified the mechanical design, and reduced integration complexity by allowing all drive units to share identical mounting geometry and electrical characteristics. Its torque output, compact form factor, and proven reliability made it well suited for the repetitive-duty operation required in our automated bottle-handling system.

Figure 8: Motor for each conveyor belt.

To ensure accurate detection of containers throughout the conveyor system, we integrated the FC-51 infrared reflective sensor at key points along each conveyor segment. The FC-51 infrared proximity sensor was selected because it provides an optimal balance between cost, precision, and integration simplicity. Its low cost enables the deployment of multiple sensing nodes along the system, while its rapid and reliable switching behavior ensures consistent detection of container movement, even under variable ambient lighting conditions. Additionally, the project team had previously employed this sensor in earlier development stages, providing familiarity with its wiring requirements, calibration procedures, and operational characteristics. This prior experience minimized integration time and contributed to stable performance throughout construction and testing.

These sensors are positioned at the entrance and exit of every conveyor lane, allowing the control system to verify when a container arrives, remains in the correct position, and successfully transitions to the next stage of the process. Each FC-51 module emits infrared light and measures the reflected signal from nearby objects. When a container passes in front of the sensor, the module generates a digital trigger that the PLC or microcontroller uses to update the conveyor logic. This enables the system to regulate bottle spacing, prevent premature advancement, and synchronize movements between consecutive subsystems. By placing sensors both at the beginning of each conveyor lane and at the interface to the main transfer belt, we created a precise and redundant detection scheme that improves overall flow reliability.

To implement the PLC-related electronics, a printed circuit board (PCB) was designed specifically for the Arduino Mega. This board consolidated all necessary microcontroller connections and included terminal blocks to facilitate the organized connection of sensors and actuators. Once the PCB was fabricated, it was mounted onto the station, and each component was wired in an orderly manner to ensure mechanical stability, electrical reliability, and overall system maintainability.

Proteus 9 Professional Software was used for the schematic and PCB design. The schematic, the PCB design, and the manufactured PCB images are shown.

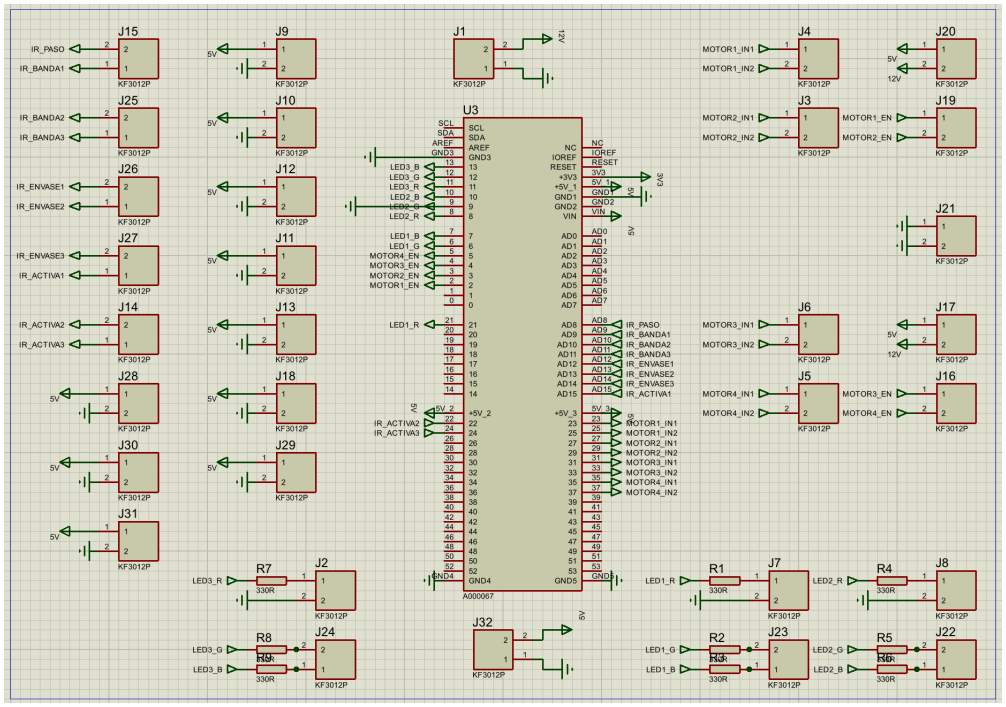


Figure 9: Schematic of the Stack System

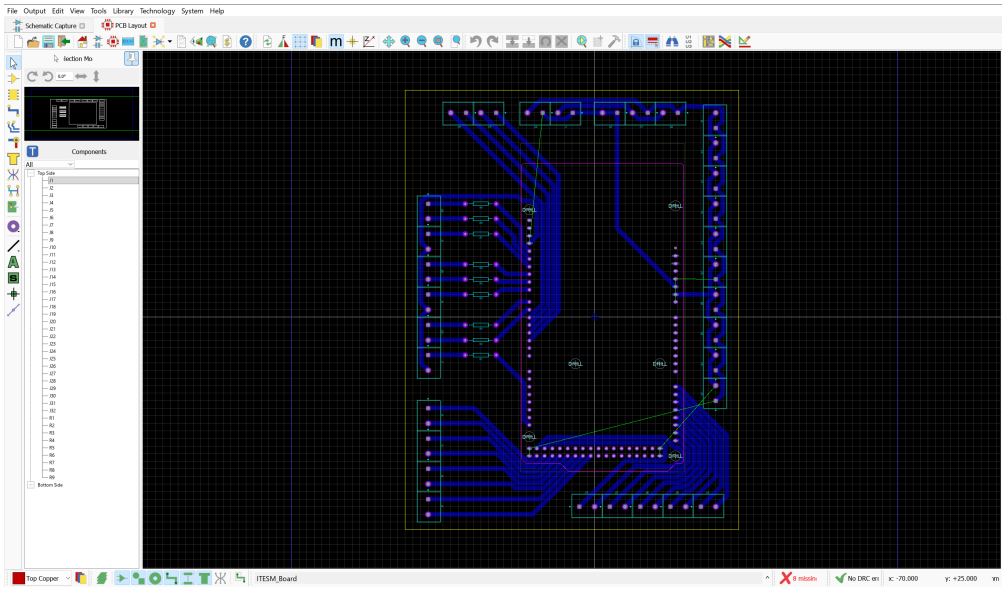


Figure 10: PCB design of the stack system

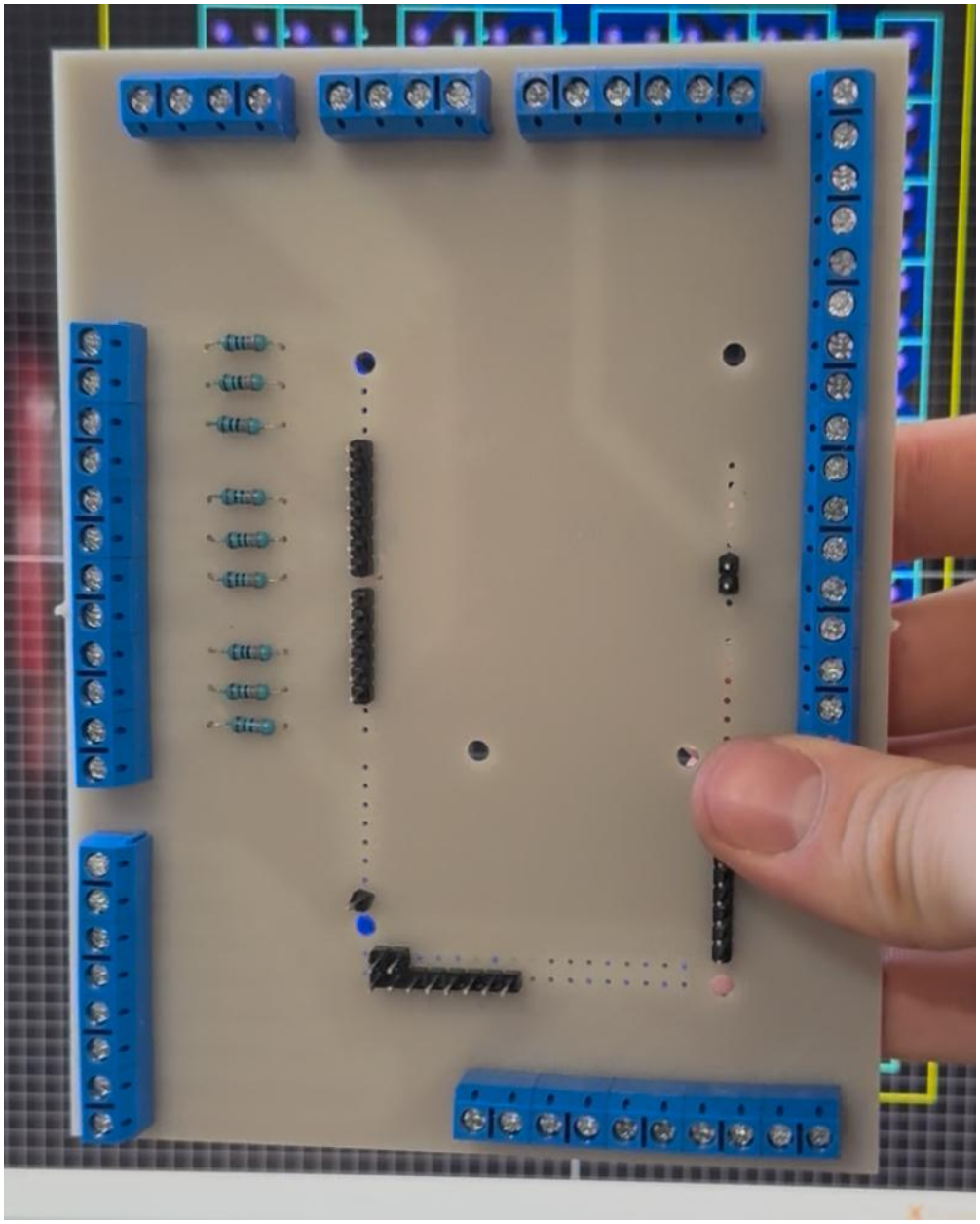


Figure 11: Manufactured PCB for the Stack system

The final stack system with the actuators and sensors mounted is shown in the following image

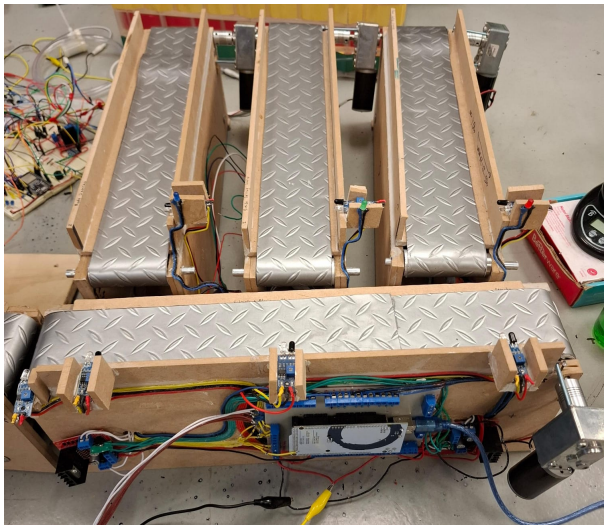


Figure 12: Stack System mounted with the actuators and sensors.

To ensure that all connections, sensors, and actuators were functioning correctly, a test program was implemented in OpenPLC. This program enabled the system to receive sensor inputs and individually control each actuator within the station. By activating components one at a time, it was possible to verify the correct operation of the entire electronic subsystem and identify any irregular behavior. This testing methodology allowed the team to confirm proper integration and, when necessary, to apply corrective adjustments before proceeding with full system operation.

2.1.3 PLC

For the development of the PLC logic, the system was divided into several functional modules to simplify implementation and ensure maintainability. First, a monitoring routine was designed in which an indicator LED would blink whenever its corresponding sensor was not detecting any object. This behavior had to be replicated three times, one for each viscosity measurement station. Thus, whenever a sensor registered no detection, its associated LED was required to remain off.

Additionally, the timing between container transitions was considered to prevent false indication states. To address this, a timer-based filtering mechanism was implemented so that LED state changes occurred only after a predefined minimum time interval had elapsed. This approach reduced the likelihood of transient readings triggering incorrect visual feedback.

The PLC program structure for the three sensors and LEDs was standardized and optimized to allow all components to be managed through a single reusable logic block. This eliminated the need to create independent routines for each LED–sensor pair and contributed to a more modular and efficient control architecture.

The second component of the control system determines the sequence in which the containers exit each station, taking into account that each selection button can only be pressed twice. When a button is pressed, its value is automatically registered in a state machine governed by internal counters. Although every button press is propagated to all states, the counters ensure that only

the state corresponding to the total number of button activations is enabled. In this way, each state stores and processes the value assigned to it during the selection phase.

Transition between states is triggered by an input signal generated by the system's final station. This signal indicates that the first processing cycle has been completed, thereby authorizing the state machine to advance to the next state and update the corresponding output behavior.

Immediately after the state transition occurs, the conveyor belt associated with the color assigned to that state is activated. When the container reaches the main conveyor, a sensor detects its presence, stops the corresponding individual belt, and initiates the main conveyor to transport the container to the designated station. Once the container reaches the end of the main conveyor, a final sensor halts the entire system, placing it in a waiting condition until the next state transition is triggered.

2.2. Viscometer System

The Viscometer System is the analytical component responsible for determining and adjusting the viscosity of the liquid sample. A piston is used to move the viscometer in the Y-axis, a stepper motor to move it in an angular direction in the Z-axis, and a DC motor spins at a fixed RPM with PID control to measure the viscosity of the containers. When the viscosity is not within expected values, a dosing pump injects water into the bottle, followed by a mixing cycle and a re-evaluation of viscosity. The spindle is mounted on a movable structure that enables transitions between the measurement area and a cleaning station. Homing routines with limit switches ensure precise positioning.

2.2.1 Mechanical Design and Construction

To build the viscometer subsystem, we designed the majority of its mechanical components for 3D printing, allowing us to obtain custom geometries, precise tolerances, and a lightweight assembly that accommodates the spinning platform and the motor arrangement. All structural and alignment-critical parts—including the rotating disc, the central support column, and the motor mounts—were printed using PLA, ensuring dimensional consistency while keeping the overall mass low to avoid excessive load on the drive motor. The only non-printed structural components were the base and support table, which we constructed using a combination of wood and MDF. These materials provided a rigid foundation for the rotating assembly, and we joined them using nails, wood adhesive, and screws to ensure resistance to vibration during operation.

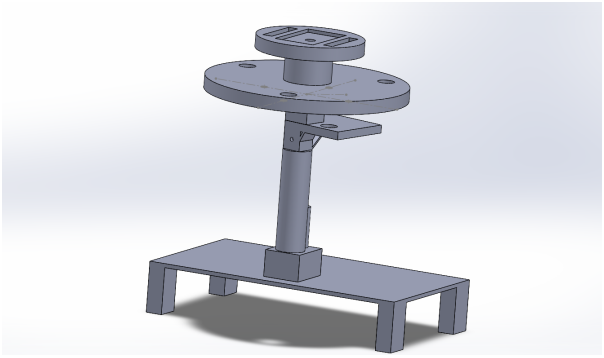


Figure 13: CAD model for Viscosmeter system.

In Appendix B can be found the viscometer CAD drawings

2.2.2 Electronic System Implementation

This section details the selection and integration of electronic components, including sensors and actuators, required for the viscometer operation.

Signal Conditioning Circuit and Calibration

The viscometer utilizes a methodology analogous to a **Brookfield Viscometer**, where viscosity is measured indirectly by quantifying the torque required to spin a submerged spindle. This torque is linearly proportional al motor's current draw.

Current-to-Voltage Conversion To measure the current, a **1 Ω shunt resistor** was placed in series with the H-Bridge driver and the stepper motor. When the motor draws current (I), a voltage drop (V_{shunt}) is generated across the resistor ($V_{shunt} = I \cdot R_{shunt}$), converting the current into a measurable voltage signal. The system's task is to condition this small differential voltage (V_{in}) into a usable range for the ESP32's 3.3 V Analog-to-Digital Converter (ADC).

Calibration Parameters and Required Gain The measurement range was determined by empirical testing:

- **Minimum (water - 25%):** $V_{in,min} = 62.1$ mV (Target $V_{out} = 0$ V)
- **Maximum (Viscous fluid (Salvo) - 100%):** $V_{in,max} = 69.5$ mV (Target $V_{out} = 3.3$ V)

The total voltage delta is $\Delta V_{in} = 7.4$ mV. The required total system gain ($A_{v,Total}$) is:

$$A_{v,Total} = \frac{\Delta V_{out}}{\Delta V_{in}} = \frac{3.3 \text{ V}}{0.0074 \text{ V}} \approx 445.95$$

Two-Stage Conditioning Design A single stage Op-Amp with a gain of 445.95 would require a large offset voltage ($k \approx 29$ V), leading to inevitable **saturation** of the Op-Amp outputs. To prevent this, the required total gain was split into two stages, aligning with the **Conditioning Equation** ($Y = mx + k$) principles discussed previously.

1. **Phase 1: Non-Inverting Amplifier and Filtering** The goal was to set the output voltage (V_{out1}) to 0.3 V at the maximum input (69.5 mV), yielding a reduced gain $A_{v1} \approx 42.85$ and a reduced offset voltage $k_1 \approx 2.678$ V.
   - **Resistors Chosen:** $R_2 = 10$ KΩ and $R_1 = 220\ \Omega + 33\ \Omega$.
   - **Low-Pass Filter (LPF):** A 47 μF capacitor was placed in parallel with the feedback resistor (R_2). This configuration creates a **Low-Pass Filter** to remove high-frequency noise and voltage spikes generated by the stepper motor's operation.
2. **Phase 2: Differential Amplifier (Subtractor)** This stage provides the remaining gain needed and performs the subtraction of the offset voltage (k_1) to establish the 0 V reference. The remaining required gain is $A_{v2} \approx 10$. The output of the Non-Inverting stage (V_{out1}) connects to the non-inverting input (V_p), and the offset voltage ($k_1 \approx 2.678$ V) connects to the inverting input (V_n).
   - **Resistors Chosen (Gain 10):** $R_1 = R_3 = 6.8$ KΩ and $R_2 = R_4 = 68$ KΩ.

This cascaded design successfully yields an output range of 0 V to ≈ 3.3 V, accurately mapping the input current variation to the ESP32 ADC range.

Actuators and drivers

- **Stepper Motor (JSS57HS76):** This high-torque motor was reused from a previous project.
- **Stepper Motor Driver (TB6600):** Used to control the JSS57HS76 motor. This driver is favored for its ability to handle the motor's current requirements, and providing micro-stepping functionality.
- **DC Gear Motor with Encoder (GM 25-370 12 V DC, 330 RPM):** Recommended by Professor Regalado. The integrated **encoder** provides closed-loop feedback, allowing the system to track angular position and velocity accurately.
- **Linear Actuator (12 V, 10 cm):** Used as the one one actuator to control the movement of our viscometer in the Z-axis, this model was chosen because of the length it can achieve.
- **H-Bridge Driver (L298N):** Utilized to control the **12 V DC motors** (GM 25-370 and the Linear Actuator). The L298N can handle the required current and provides simple control over direction and speed via the micro-controller.
- **Small DC Motors (5 V):**
  - **Submersible Water Pump (5 V):** Used for controlled fluid handling (Viscosity adjustment).
  - **DC Motor (5 V):** Used for the drying station.

- **Low-Current Motor Driver (L293D):** An integrated circuit used to control the activation of the small 5 V motors (water pump and drying motor). It is suitable for their lower current requirements.

Sensors

- **Color Sensor (Adafruit):** This sensor serves a crucial logical function by determining the color of the fluid container.
- **Digital Infrared Sensors:** Selected due to their simplicity. These sensors only require three pins (power, ground, and digital output), which simplifies their use and management within the ESP32 GPIO pins. Crucially, they already contain internal conditioning circuitry, eliminating the need for external resistor arrangements common with bare phototransistor sensors (like the CNY70 or QRD1114).
- **Ultrasonic Sensor (HC-SR04):** Used to measure the platform height of the viscometer. This sensor was chosen due to its immediate availability.
- **Bidirectional Logic Level Converter (5 V to 3.3 V):** This component is critical for **protecting the ESP**32. The encoder attached to the GM $25-370$ motor is powered by 5 V, meaning its output logic high spikes would reach 5 V. Connecting this directly to the ESP32 (which operates at 3.3 V logic) would risk damaging the micro-controller. The level converter reliably steps down the logic high signal from 5 V to a safe 3.3 V before it reaches the ESP32 GPIOs.

Schematic and PCB design

The system's schematic integrates all the components listed above, connecting the sensors and motor drivers to the ESP32 micro-controller.

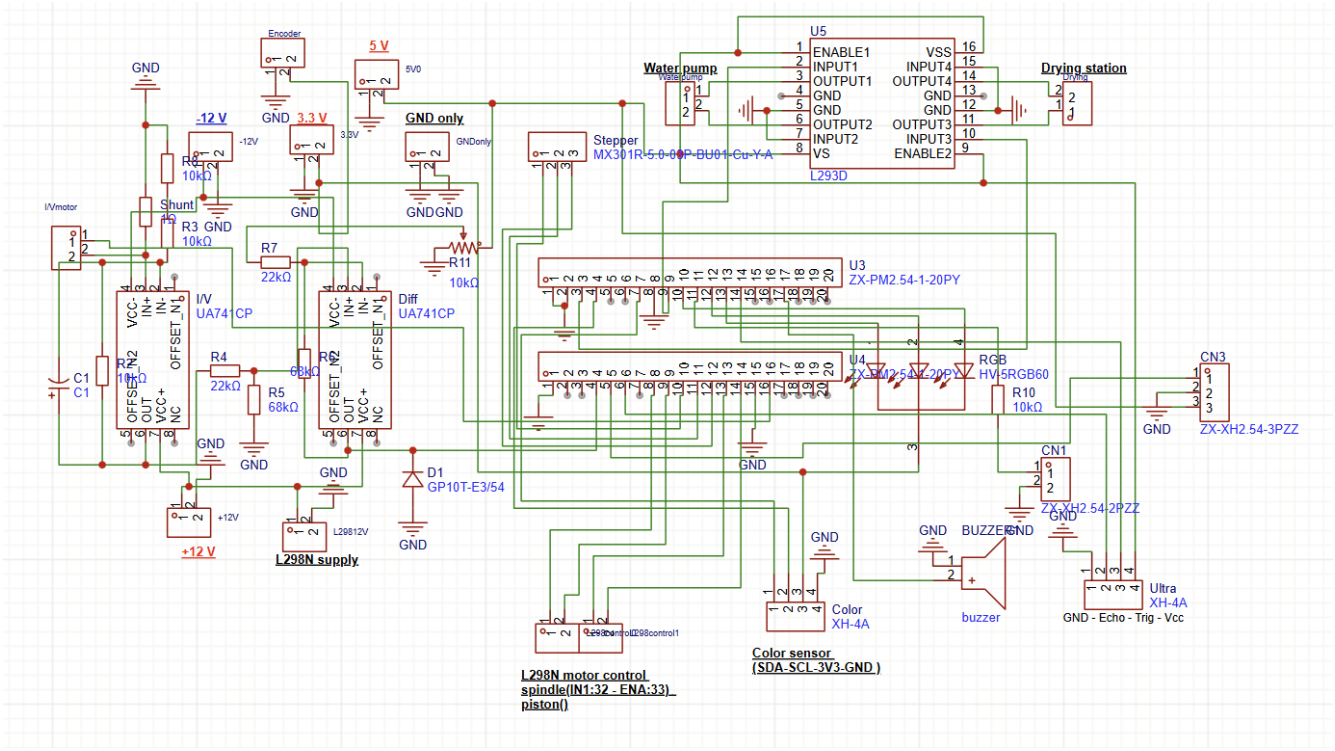


Figure 14: Schematic of the viscometer.

PCB design: By using EasyEDA software to design both the schematics and the PCB for the system, we space efficient PCB.

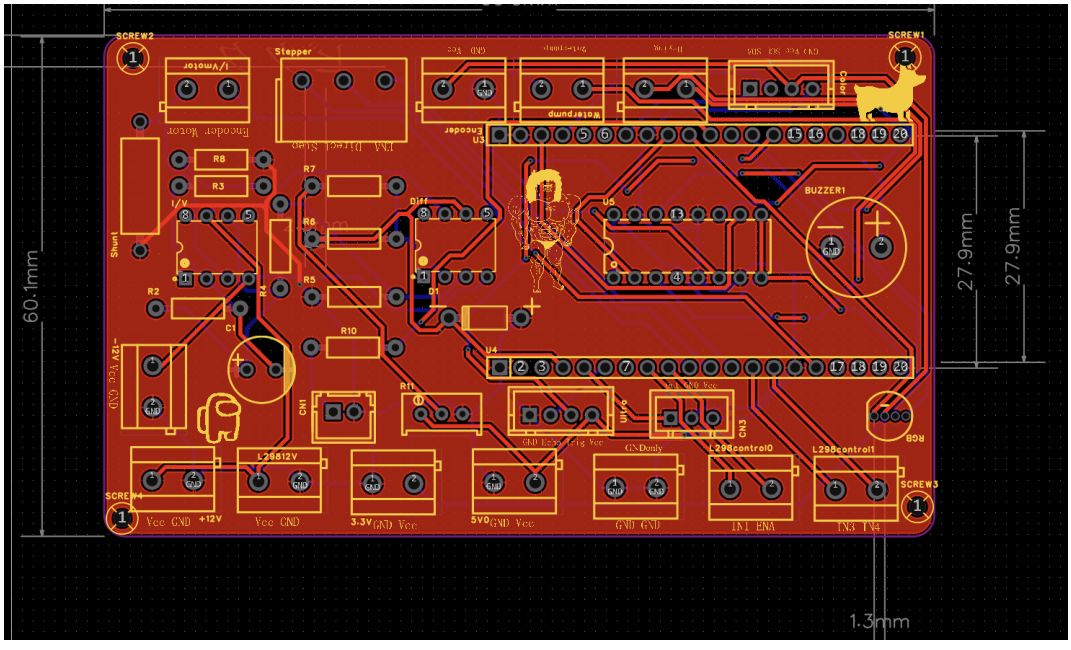


Figure 15: Final design of the PCB for the viscometer system.

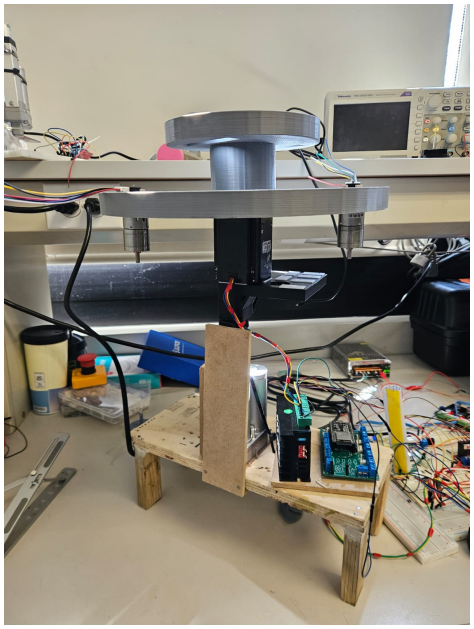


Figure 16: Viscometer mounted with the actuators and sensors.

2.2.3 PID control for DC motor

The core objective of the development phase was to accurately model the DC motor and subsequently tune a Proportional-Integral-Derivative (PID) controller to achieve precise speed regulation. This process involved empirical data collection, system identification in MATLAB, and controller tuning.

Empirical Data Collection and Feedforward Modeling

The motor was first subjected to a series of increasing Pulse Width Modulation (PWM) duty cycles to establish the relationship between the control input and the steady-state angular velocity.

- **Minimum Operating PWM:** The experimentation began by determining the minimum PWM required to overcome static friction and initiate motor rotation, which was found to be **20**%.
- **Data Collection:** A sweep of PWM values, from 20% up to 100%, was applied. For each step, the steady-state encoder frequency (**Freq**) was recorded, the actual speed (**RPM′s**) was measured using an external tachometer, and the analytical RPM (**RPM analtico**) was calculated using the motor's PPR = 291 for validation.
- **Output Validation:** The final output variable, **Deg/s** (Degrees per Second), was calculated and validated against the velocity readings obtained from the microcontroller's serial terminal.

The collected data used for modeling is presented in the next Table:

Feedforward Equation

Table 3: Experimental Data for DC Motor Speed vs. PWM

PWM	**Freq (enc) [Hz]**	**RPM's**	**RPM analítico**	**RPM AVG**	**Deg/s**
0	0	0	0	0	0
19	0	0	0	0	0
20	57.8	12.01	11.917526	11.963763	71.782577
25	128.9	26.96	26.57732	26.76866	160.61196
30	206.4	43.3	42.556701	42.928351	257.5701
35	286.2	59.95	59.010309	59.480155	356.88093
40	365.12	76.3	75.282474	75.791237	454.74742
45	436.45	91.39	89.989691	90.689845	544.13907
50	513.34	107.45	105.8433	106.64665	639.8799
55	593.2	124.22	122.30928	123.26464	739.58784
60	671.86	140.2	138.52784	139.36392	836.18351
65	742.41	155.23	153.07423	154.15211	924.91268
70	819.16	171.15	168.89897	170.02448	1020.1469
75	895.47	187.2	184.63299	185.91649	1115.499
80	972	203.56	200.41237	201.98619	1211.9171
85	1043	217.5	215.05155	216.27577	1297.6546
90	1117	233	230.30928	231.65464	1389.9278
95	1208	251.75	249.07216	250.41108	1502.4665
100	1326	275	273.40206	274.20103	1645.2062

A linear regression of the data was performed to establish a relationship between the input (PWM) and the output (Deg/s). The resulting equation served as the **feedforward** component for the control system:

$$\text{PWM}_{\text{required}} = 0.0542 \cdot (\text{Deg/s}_{\text{desired}}) + 14.289$$

This equation provides the necessary steady-state PWM to achieve a desired velocity and establishes the foundation for the open-loop control structure.

System Identification and PID Controller Tuning

With the empirical data prepared, the next step was to derive the motor's dynamic model and determine the optimal PID gains.

Transfer Function Estimation using MATLAB

The columns for PWM (input U) and Deg/s (output Y) were loaded into MATLAB. The motor's dynamic behavior was modeled using the System Identification Toolbox.

- **Data Import:** The input and output data were imported from the spreadsheet.
- **Model Estimation:** The **tfest** (Transfer Function Estimation) function was used to fit a continuous-time transfer function $G(s)$ to the experimental data. Assuming a standard DC

motor model, a **first-order system** was estimated:

$$G(s) = \frac{K}{\tau s + 1}$$

The **tfest** function utilizes the loaded data to estimate the steady-state gain (K) and the time constant (τ), resulting in the plant model $G(s)$.

- **Controller Input:** The derived $G(s)$ was then passed to the automated tuning utility.

PID Controller Autotuning

The **pidTuner** application in MATLAB was used to systematically synthesize the PID gains based on the estimated plant model $G(s)$.

- **Tuning Strategy:** The tuning focused on achieving a rapid **Step Response** while minimizing **Overshoot** to ensure stability and precise tracking of the reference speed.
- **Optimization:** By iteratively adjusting the **Response Time** and **Transient Behavior** sliders within the **pidTuner** interface, the closed-loop performance was optimized. The final step response demonstrated fast rise and settling times with minimal transient error.

Final PID Gains and Performance

The tuning process resulted in the following proportional-integral-derivative (**PID**) gains, which are implemented in the embedded system operating at a sampling period of $T_s = 0.001$ s (1 kHz):

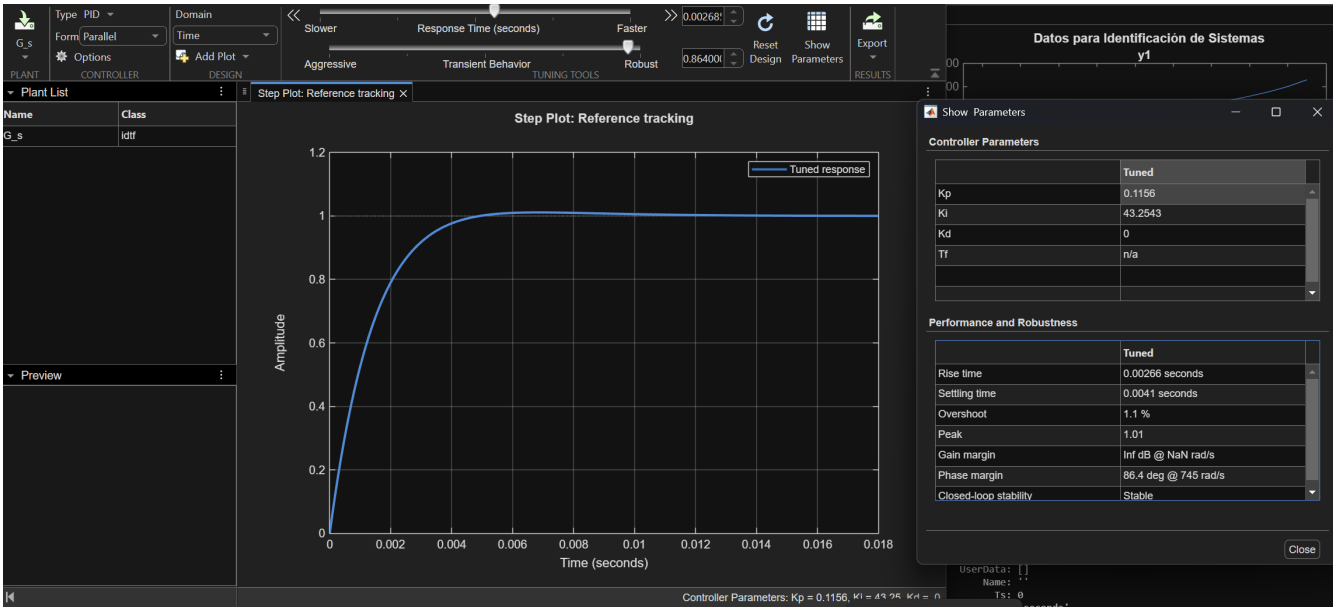


Figure 17: Step response got from Matlab PID tune tool.

The derivative term ($\mathbf{K_d}$) was set to zero, resulting in a PI Controller to avoid amplifying noise typically present in velocity measurements. The key performance metrics achieved with these gains, as simulated by the tuner, were:

Table 4: Optimized PID Controller Gains

Parameter	Value
$\mathbf{K_p}$ (Proportional Gain)	0.1156
$\mathbf{K_i}$ (Integral Gain)	43.2543
$\mathbf{K_d}$ (Derivative Gain)	0

- **Rise Time:** 0.00286 seconds
- **Settling Time:** 0.0041 seconds
- **Overshoot:** 1.1%

The used code can be found in Appendix E

2.2.4 C++ Code

Two different codes were made for the Viscometer, which were later integrated together. The first one was the Manual Controlling, which did not follow any state machine model, and instead, we were able to control all the actuators and monitor the sensors in the LabVIEW interface.

The Automatic Controlling did follow a state machine model in which the cycle was:

1. **Homing routine**: Before any measurement or container positioning, there was a homing positioning state in which the piston positions the viscometer height 30 cm from the ground so the container would not clash with the spindles, the stepper motor being in 0°, knowing that it is positioned at the exact angle because of an IR sensor functioning as a limit switch, and the DC motors with the spindles spinning at no velocity. This state could also work as the Calibration instead of the Homing. The RGB turns yellow and blinks to let the User know that the viscometer is homing or calibrating.

2. **Automatic Idle**: In this state, nothing is moving, and it is waiting for the conveyor belt to advise the viscometer that the container is positioned. The RGB is turned off, and all the actuators are stopped.

3. **Container positioning**: The conveyor belt had two IR sensors that detected when the container was positioned at the middle of the viscometer. When the container was in the good position, then with another IR sensor, the conveyor belt notified the viscometer that the container was positioned to start the measurements. When this happens, the RGB turns green and waits for 5 seconds to confirm that the container is still positioned.

4. **Detect tag color**: The color sensor starts to detect the color of the container, and depending on the detected color, it will catalog the viscosity objective as low, medium, or high viscosity. The minimum and maximum accepted ranges will be registered so that when measuring the

viscosity, the system knows that the cPs obtained has to be within the range. Otherwise, there are three more iterations. If the viscosity is not still within the ranges after three iterations, the container is disposed

5. **Measure viscosity**: To do this, the piston is places so that the spindle submerge in the liquid, then is starts to spin at 130 RPM to stir the liquid for 30 seconds. After that, the spindle starts to spin at 30 RPM for 10 seconds to measure the viscosity, and finally, the average viscosity value is stored

6. **Dossing adjustment**: If the viscosity obtained was within the range, it passes directly to the next state machine. If not, we have three iterations to do: dose 85% water in the first iteration and 5% water in the second and third iterations. When the iterations happen, the state is set again in the *Measure viscosity* state to evaluate. After all the iterations, if the viscosity is not yet within the desired ranges, the container is disposed of. The viscometer notifies the belt that it has finished the measuring process with an IR sensor, and if the container needs to be disposed of, a pin from the ESP32 microcontroller to the Arduino microcontroller sends HIGH to notify the conveyor belt that it has to be disposed of.

7. **Cleaning station**: The stepper positions the spindle at 120° to go to the cleaning station. Then, the spindle is submerged and lies there for 60 seconds without spinning to be cleaned with the water.

8. **Drying station**: The spindle is raised, and then the stepper motor positions it to the 240° angle to be now in the drying station. After arriving, a DC motor with a fan will be turned on while the spindle spins at 30 RPM. After that, the whole process is restarted.

The next diagrams show the Manual and automatic logic of the viscometer.

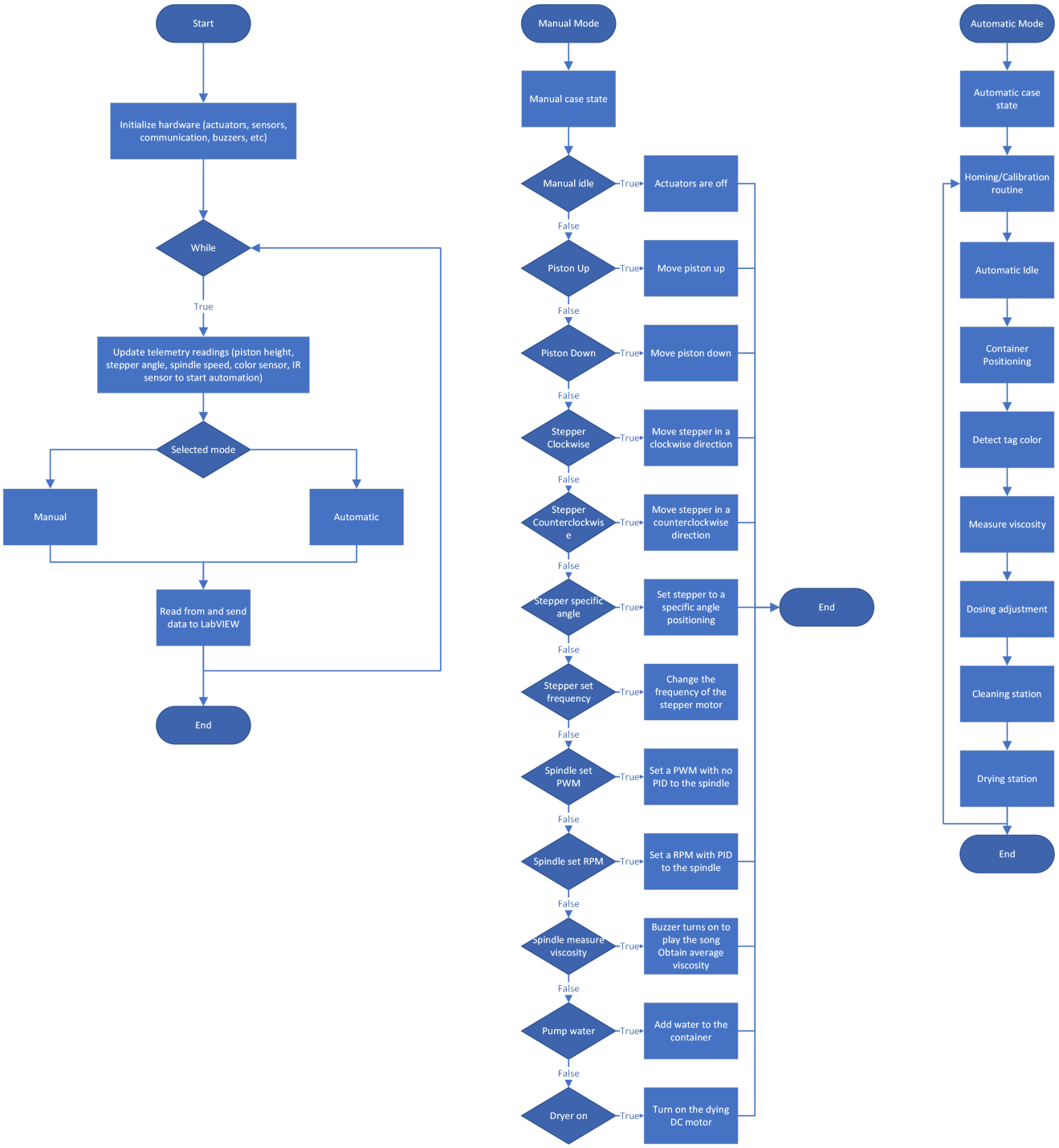


Figure 18: Flowcharts of the viscometer. On the left, the general flowchart. In the center, the manual mode flowchart. On the right, the automatic mode flowchart

The code with the definitions.h and main.cpp can be found in Annex D

2.2.5 LabVIEW VI

While the code is loaded to the ESP32 microcontroller, the LabVIEW interface is used to control (in manual mode) and monitor (both the manual and automatic mode) all the actuators and sensors of the viscometer. It has also graphs in which we can visualize the current the spindle is taking, or the position and velocity angles of the stepper motor. To change between the manual and automatic mode, the *Mode* button has to be pushed.

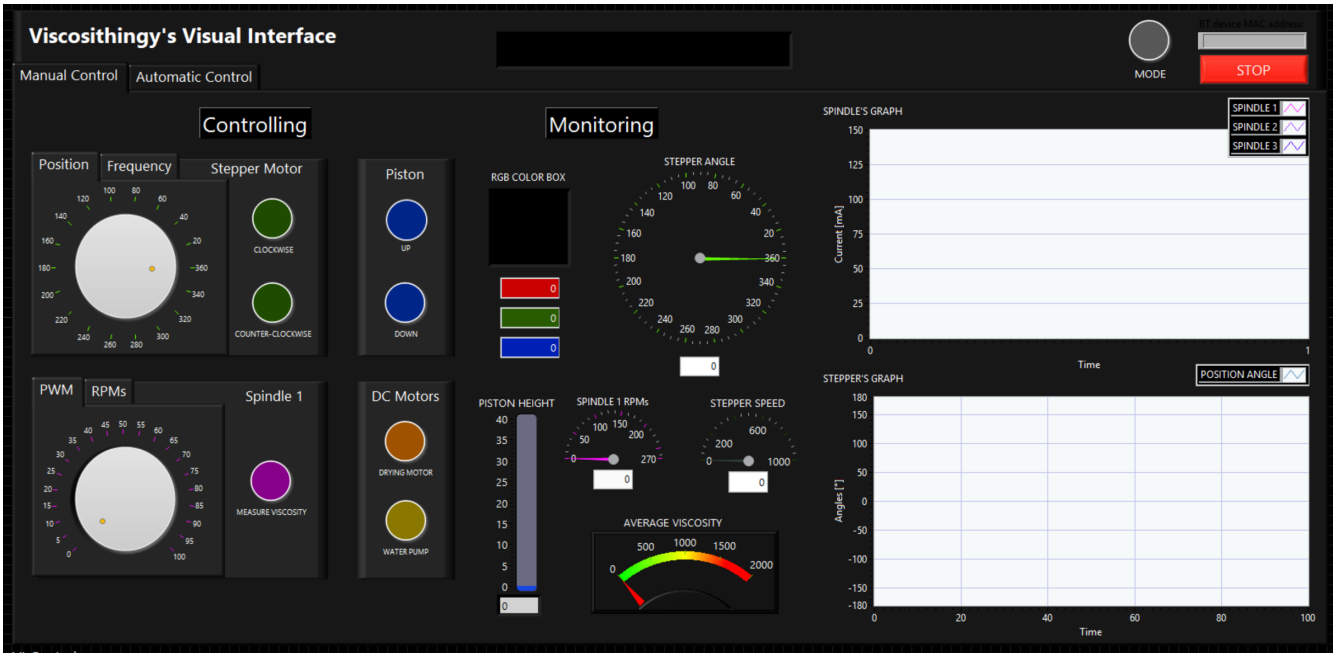


Figure 19: LabVIEW front panel

For the Block Diagram, we use Bluetooth protocol to connect with the ESP32 microcontroller. There is no logic implementation in this Block Diagram, as all the logic was coded in the C++ files. In here, we only read the variables for the monitoring part, which are the sensors, and we write in the controlling part, moving all the actuators.

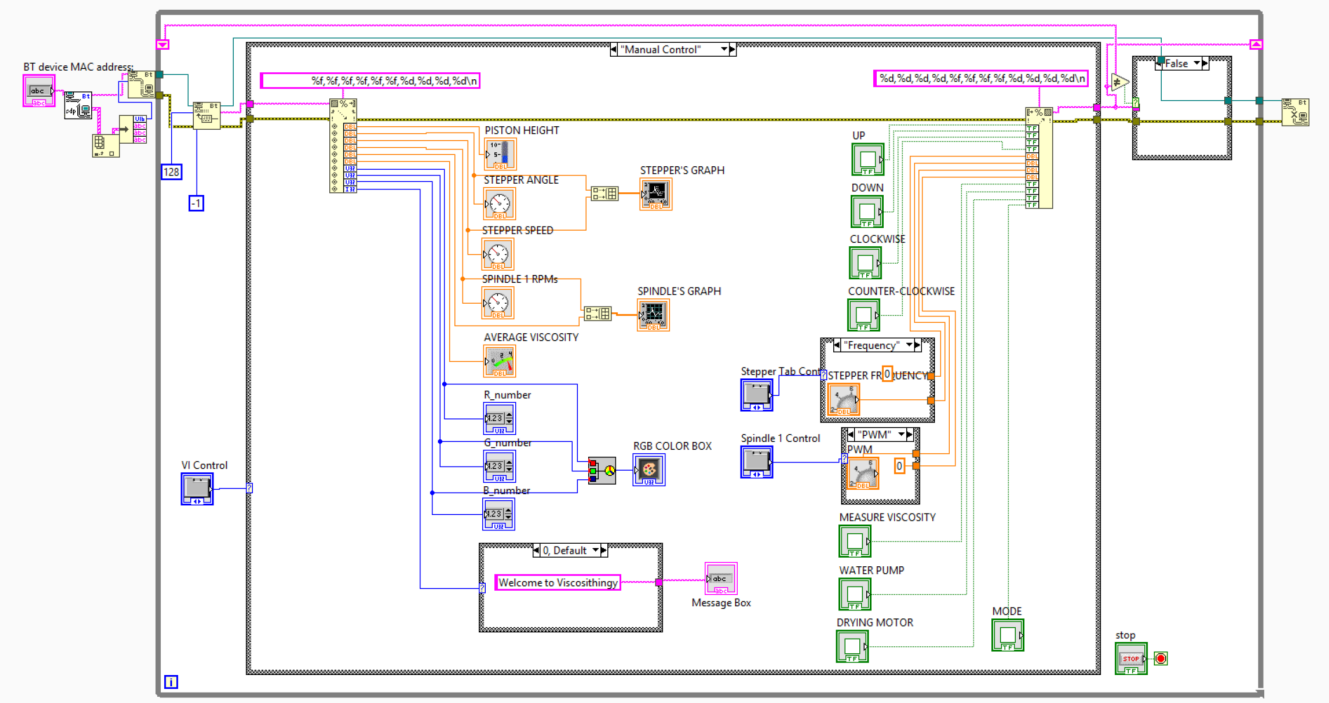


Figure 20: LabVIEW Block Diagram

2.2.6 PLC

In comparison with the other modules, this station's system was the simplest to implement. Its operation begins with an initial sensor that activates the conveyor belt. At the center of the mechanism, two infrared sensors are positioned such that the belt stops only when both are triggered simultaneously. This configuration ensures that the container is properly centered, allowing the viscometer to perform its measurement cycle with accurate alignment.

Once the viscometer completes its process, the ESP32 activates an infrared LED, which is detected by a corresponding sensor on the Arduino. This signal indicates that the measurement has finished and that the container may continue through the system. A manual button is also included to allow operators to proceed in cases where the viscometer is unable to transmit the completion signal.

After the signal is received, the conveyor resumes operation until the container reaches the end of the belt and transfers to the subsequent station. At this point, the main conveyor motor is stopped, ensuring proper synchronization between stations.

2.3. Box Sorting System

The Box Sorting System classifies bottles according to their final viscosity range. Three linear actuators are placed along the conveyor belt, each responsible for pushing a bottle into a designated box. When the PLC receives the viscosity classification, the conveyor halts at the appropriate position and the corresponding actuator extends, redirecting the bottle into its assigned container.

The collection boxes are placed on a height-aligned platform. If sliding occurs during actuation, optional L-shaped stoppers can be added to stabilize the boxes.

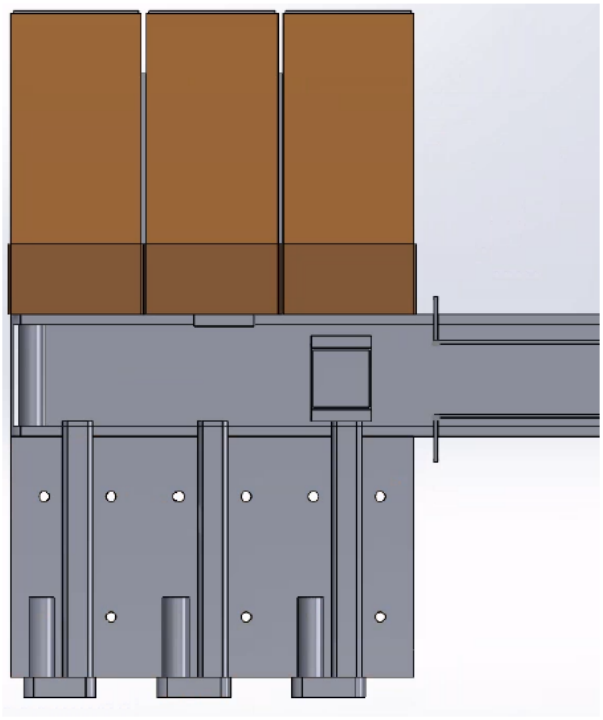


Figure 21: CAD model of the Box Sorting System.

2.3.1 Construction

The structural framework of the Box Sorting System follows the same construction criteria applied in the previous stations, while incorporating additional elements required to support actuation, impact resistance, and repetitive mechanical cycles. The main body of the sorting module was assembled using 1/8 in MDF panels, selected due to their dimensional stability, ease of machining, and consistent behavior under load. These panels form the external housing, internal guides, and the base platform for each actuator. To ensure adequate rigidity, all MDF joints were reinforced using a combination of industrial wood adhesive, nails, and aluminum rivets. Adhesive provided surface bonding, while nails and rivets ensured mechanical locking, especially along structural edges subjected to vibration and repeated motion.

2.3.2 Electronics

Table 5: Sensor Selection for Box Sorting System

Sensor	Range	Type	Detection Rate	Resolution	Protocol
ISL29125	RGB with IR rejection	Digital (I^2C ADC)	Depends on ADC setting	12/16-bit	I^2C, 3.0–3.6 V
TCS34725	Visible spectrum photodiodes	Digital (I^2C ADC)	Programmable 2.4 ms steps	Up to 16-bit	I^2C, adjustable gain
TCS3200	400–700 nm	Digital (freq. output)	2 Hz–500 kHz	Counting-based	Requires controlled lighting

The third station of the system requires precise detection of incoming containers before actuating the sorting mechanism. As in previous stages, we employed the FC-51 reflective infrared sensor due to its proven reliability, low cost, and ease of integration with simple digital logic. This module operates by emitting an infrared beam and measuring the reflected intensity to determine whether an object is present within a configurable distance threshold.

In this section, each FC-51 sensor is positioned directly in front of its corresponding sorting lane, where it continuously monitors the arrival of the container. When a container passes over the sensing point, the infrared module triggers a digital signal that instructs the microcontroller to activate the linear actuator associated with that specific lane. This ensures that each item is diverted into the correct box in real time, avoiding misclassification or mechanical conflicts between lanes.

Because this sensor had already been validated in the first station and demonstrated consistent performance across different lighting and surface conditions, it was again selected for the sorting module. Its adjustable sensitivity and robust LM393 comparator allow accurate detection even when the containers differ slightly in color or reflectivity. The low unit price also made it possible to deploy one sensor per sorting lane without increasing overall project cost.

Another sensor used in this station was the Adafruit TCS34725 RGB color sensor which detected the labels for the incoming containers in order to give the order of moving them only into their respective boxes. It works through an I2C communication protocol which would later prove to be a significant hassle as Open PLC doesn't directly support this protocol through ladder logic.

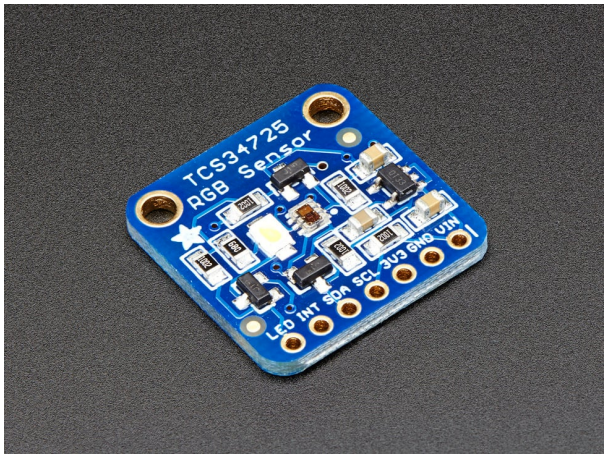


Figure 22: TCS34725 RGB color sensor.

To implement the PLC-related electronics, a printed circuit board (PCB) was designed specifically for the Arduino Mega. This board consolidated all necessary microcontroller connections and included terminal blocks to facilitate the organized connection of sensors and actuators. Once the PCB was fabricated, it was mounted onto the station, and each component was wired in an orderly manner to ensure mechanical stability, electrical reliability, and overall system maintainability.

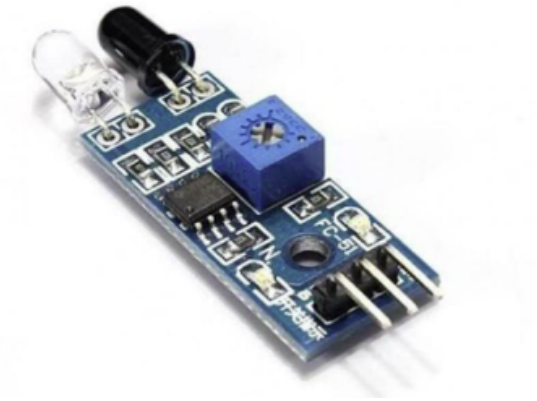


Figure 23: IR Sensor.

To ensure that all connections, sensors, and actuators were functioning correctly, a test program was implemented in OpenPLC. This program enabled the system to receive sensor inputs and individually control each actuator within the station. By activating components one at a time, it was possible to verify the correct operation of the entire electronic subsystem and identify any irregular behavior. This testing methodology allowed the team to confirm proper integration and, when necessary, to apply corrective adjustments before proceeding with full system operation.

To guarantee secure and reliable connections between the Arduino and the electronics present at the station another PCB was designed and manufactured with the intent of being placed as a shield directly above the Arduino whose traces connected to a series of terminal blocks, one for each used pin on the Arduino.

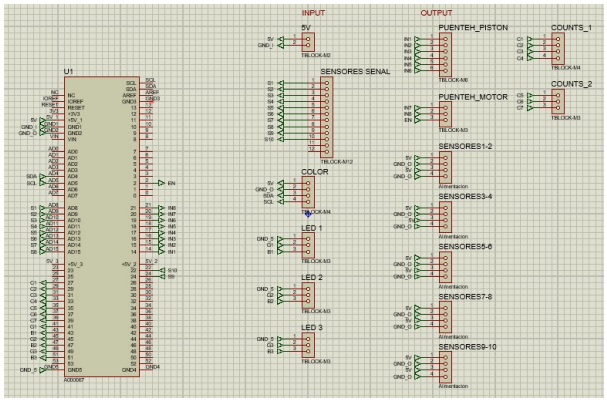


Figure 24: Schematic of the Sorting System.

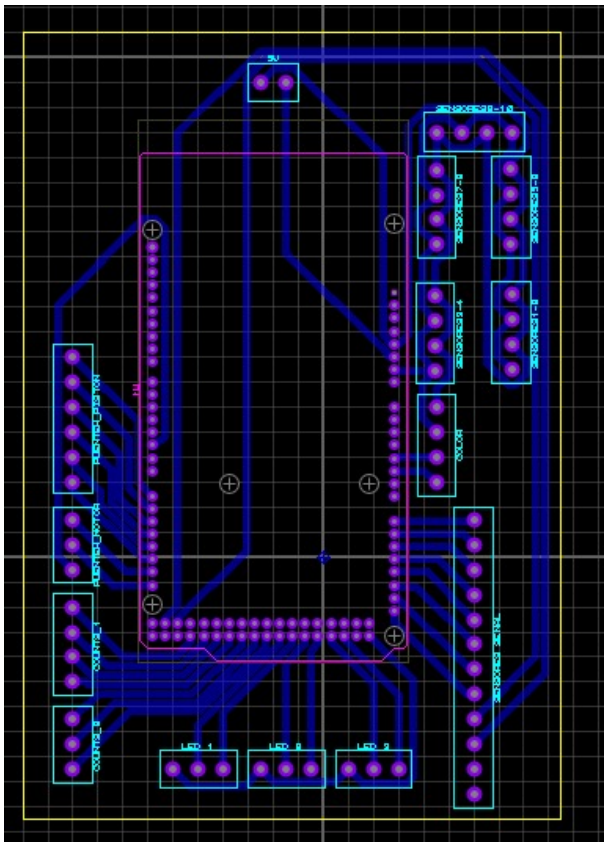


Figure 25: PCB design of stach system.

2.3.3 PLC

To initiate this station, a color-detection sensor programmed in Arduino Extension C++ and integrated with an Arduino UNO was implemented to identify the color of each container as it moved along the conveyor. Once the sensor detected the container's color, it transmitted a corresponding digital signal to a dedicated input pin on the main controller. The controller was configured to receive one of three possible signals: red, green, or blue, each representing a different container type.

Reading and sending the color would prove to be much more difficult than originally expected because of the previously mentioned limitation of the TCS34725 sensor only working through I2C and OpenPLC not handling this protocol directly through ladder logic. Instead, a feature within OpenPLC called "Arduino Extension" had to be used. This feature enables users to program in normal Arduino C++ code and then export the measured variables into ladder logic. Getting the sensor to work would be quite a hassle that will be discussed at lenght in the Discussion section but a workaround would eventually be implemented and the sensor would be properly read by the ladder diagram as 3 variables, one for each color.

Upon receiving the color signal, the system enabled only the sensor and pneumatic actuator associated with the detected color, thereby preventing unintended activation of the remaining mechanisms. After the color was identified, the main conveyor resumed motion to transport the container to its designated station. When the container reached the sensor corresponding to its color, the main conveyor halted, and the assigned piston extended to position the container correctly, after which the piston then retracted to its initial state. Both extension and contraction of the actuators were controlled by TON timers with a 20 seconds duration. At the end of this sequence, the station transmitted a digital high signal (1) to the first station, indicating that it could release the next container in the process.

In cases where the viscosity measurement was incorrect, the ESP32 microcontroller sent a signal to the Arduino to activate the conveyor until the container reached the end of the line and was ejected. Once the container exited the system, the conveyor stopped, and the Arduino signaled the first station to begin a new processing cycle.

Unlike earlier subsystems, this station required a high-strength upper surface capable of withstanding direct sliding contact and occasional impacts from moving containers. For this purpose, a textured aluminum sheet was integrated into the platform. This material offers increased abrasion resistance, improved stiffness, and reduced friction, allowing containers to move smoothly into their designated lanes. The sheet was precisely cut to size and fastened using rivets and adhesive to prevent deformation during actuator operation.

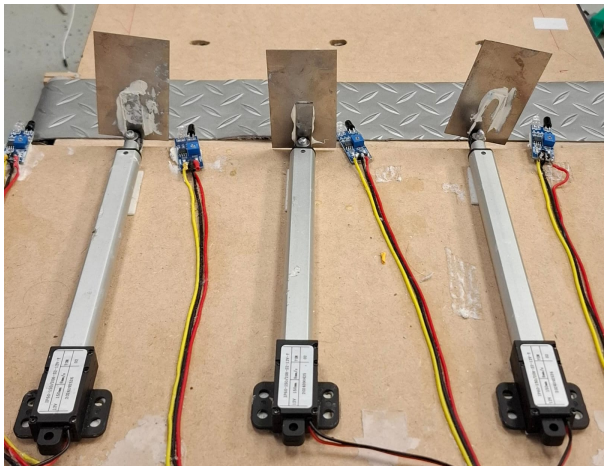


Figure 26: Aluminum plates for greater contact area.

A defining component of this module is the 12 V DC linear actuator, responsible for redirecting containers into their assigned lanes upon receiving input from the infrared sensor. Each actuator was mounted using metal brackets anchored to the MDF frame, while the telescopic rod connected to an aluminum flap acting as a diversion gate. Integrated limit switches ensure controlled extension and retraction, preventing over-travel and mechanical stress. To support actuator forces, additional aluminum plates were installed at mounting points to distribute load and avoid localized deformation.

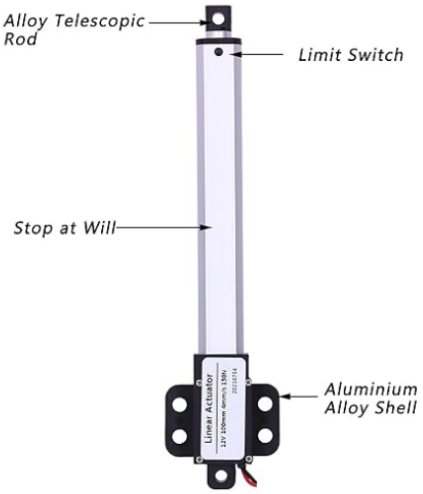


Figure 27: Linear Actuator for Box Sorting System.

Although constructed with similar materials as the previous stations, the Box Sorting System incorporates higher mechanical demands due to the actuation forces involved. The combination of MDF panels, aluminum reinforcement, and linear actuators results in a compact and durable structure capable of continuous operation. The infrared sensor provides the timing signal for actuator activation, while the mechanical design ensures consistent routing of containers with minimal wear across multiple cycles.

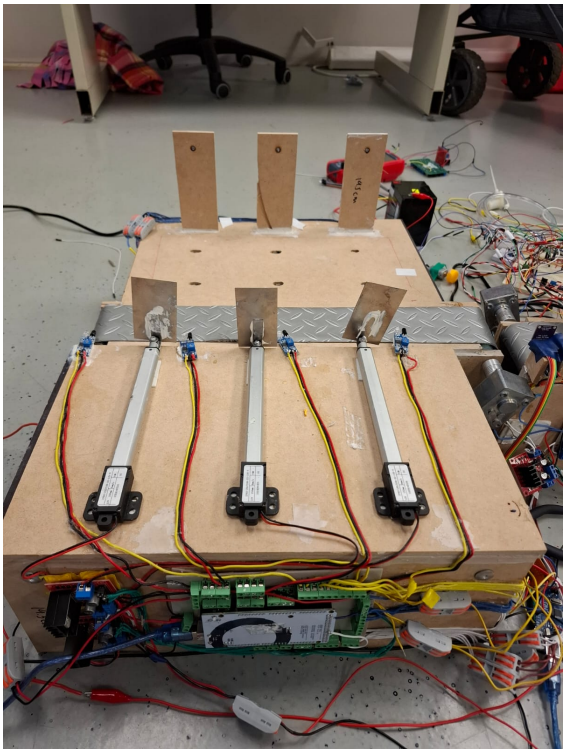


Figure 28: Box sorting System.

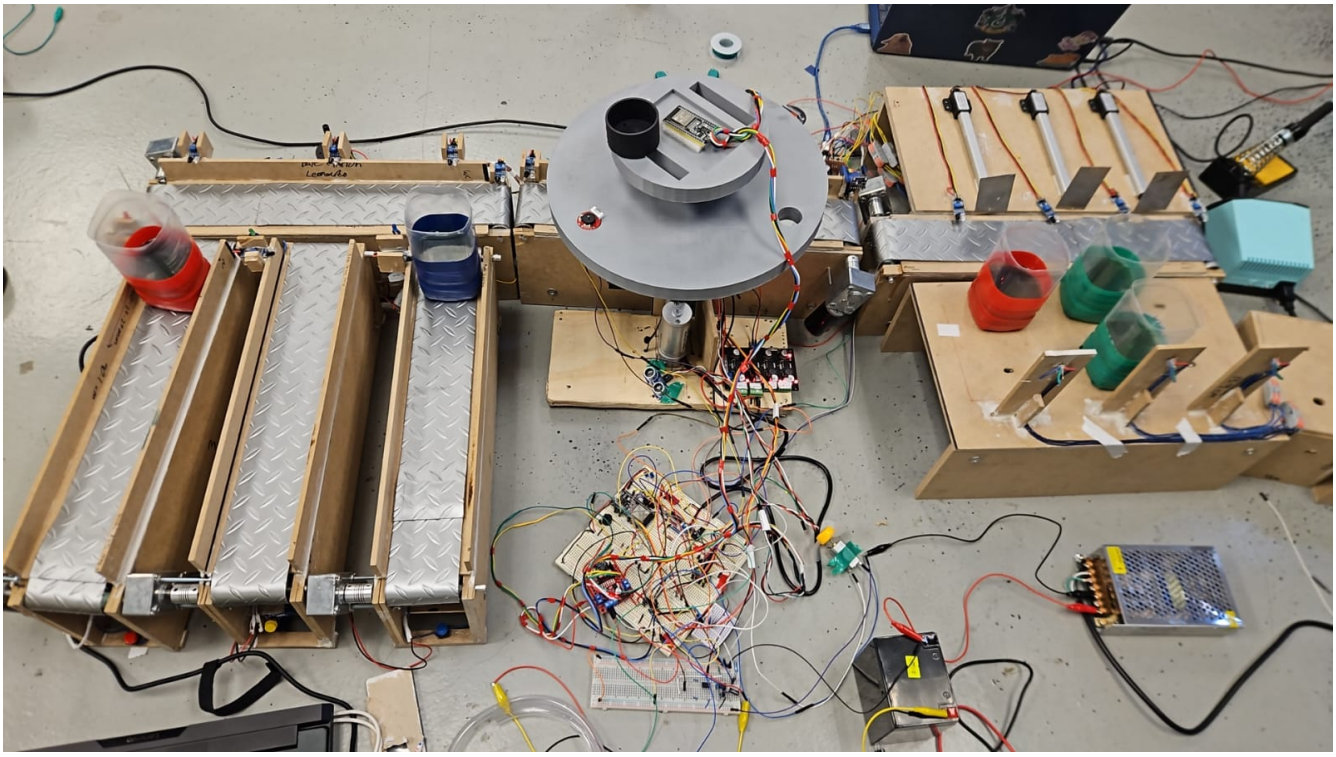


Figure 29: Photo of the system integrated with the conveyor belt and the viscometer

3. Results

3.1. Bottle Stack Dispenser

The development of both the PLC program and the electronic system for the stack station was completed successfully. During implementation, several modifications had to be made to the PCB, including the addition of wire bridges on the underside of the board. These adjustments were necessary because the sensor signals connected to certain pins exhibited instability. Furthermore, some of the signals received from the sorting station were affected by electrical noise, which required modifications to the PLC logic to ensure reliable operation. Despite these challenges, all corrections were implemented effectively, and the system operated as expected.

The performance of the stack station produced the anticipated and favorable results. The system fulfilled its objective of dispensing containers in the specified order and at the appropriate moment. However, several issues were observed during testing. First, the presence indicator LED for the blue-container conveyor flickered with low intensity and short duration. This behavior resulted from the Arduino supplying only 1.8 V to that output, insufficient for proper LED operation. The cause of this voltage drop is still under investigation.

Additionally, mechanical limitations were identified within the conveyor system. The belts moved with significant friction, causing irregular motion and generating excess torque on the motors. This led to current spikes, instability in container positioning, and generally slow and uneven movement. Moreover, during the transition from one conveyor to another, containers

occasionally fell, indicating a need for improved alignment or support between conveyor sections.

3.2. Viscometer System

Before starting manufacturing of our own prototype we made baseline measurements with a professional-grade viscometer and round containers resulting in this measurements:

Percentage of Water (%)	**Viscosity [cP]**
0%	1984
5%	1376–1380
10%	976–980
15%	672–676
20%	484–488
25%	338–340

Table 6: Viscosity measurements with professional viscometer.

For our own prototype we used Electrolit bottles which are closer to a square than a circle. This worried us on how it would affect the viscosity readings so we also made measurements using said containers giving the following results:

Percentage of Water (%)	**Viscosity [cP]**
0%	2034
5%	1308
10%	994
15%	679
20%	501
25%	352

Table 7: Viscosity measurements with Electrolit bottles.

The difference was barely a few percentage points between the round and semi-squared containers so we decided on using the latter for ease of use.

The results of the viscometer were mostly as expected, with some differences and changes from the very beginning. Throughout its development, the construction of the CAD models and their manufacturing, as well as the programming and wiring, were challenging stages. In each of these phases, we had to plan carefully and follow a solid process in order to understand the best way to approach each task.

Based on this, during the construction of the model, fixing the piston to the wooden base provided better stability and solved the issue of lifting the other component. Additionally, the base attached to the piston was very helpful in supporting both the motor and the part where the

spindle was located. Having a firm and functional structure allowed us to begin integrating the electronic components. As a result, during the initial tests of the viscometer's movement and the spindle, we were able to measure the correct viscosity for both the 100 percent Salvo soap solution and the mixture of 75 percent soap and 25 percent water. We also achieved precise control of the motor movement and developed a model that allowed the base where the motors and spindles were located to rotate.

On the other hand, the control of the motors where the spindles were located was similar, and the main difficulty, as mentioned before, was the calibration of viscosity and the available measurement range. One of the mistakes we made during the first measurements was having the measurement values inverted, but after this small adjustment, we were able to achieve better development. Because of this, the development of the LabVIEW interface became somewhat easier, and by the end of the project we were able to demonstrate the manual mode of the control panel and its functionality. More specifically, the development of the entire viscometer section was precise, and although the process was somewhat tedious, we successfully achieved its operation and are pleased with the idea of further improving the model.

3.3. Box Sorting System

The operation of the sorting station was successful, as it consistently achieved its objective of automatically classifying the containers according to their color. Although the mechanical components of the conveyor performed adequately, they exhibited issues similar, though less severe, to those observed in the stack station, particularly in the smoothness and stability of belt movement.

Regarding the development of the PCB, modifications and wire bridges were again required to correct routing issues and signal inconsistencies. After these adjustments, the board functioned properly.

One of the aims of the whole system was to have the ability of having each part work separately in case a certain section stopped working or another problem arose. One decision that led us to make was having two RGB color sensors, one for the viscometer station which worked through the ESP32 present in said station, and another sensor present at the start of the sorting station. This station only had an Arduino MEGA

4. Discussion

4.1. Mechanical design

The design upon which the bases were built mostly accomplished their purpose of working as a stable, sturdy, and reliable platform upon which to mount the system's different parts. They had the advantage of being capable of being mass manufactured as all 3 stations due to only having 3 distinct parts: one for the side supports, one for the conveyor belt to rest upon, and one as a central stabilizer, and they all only depended on simple rectangular shapes and 90° cuts. Each station used at least one of the standard model with the conveyor belt with the final sorting station and the middle viscometer station using additional models for the viscometer itself and mounting the linear actuators and container boxes respectively.

For the viscometer a custom turret that elevated and rotated was developed. The first two designs had great disadvantages because when we needed the motor to be stable, when taking the measurements, it would become highly unstable and therefore we couldn't measure the RPM's. On our third design we solved this problem in its totality managing to measure and have a stable base for the viscometer to work.

For the sorting station, an additional 2 bases were manufactured where the linear actuators and the packing boxes would rest upon. These also had very simple designs to manufacture as they only consisted of the two supports to the sides, and the base plate itself.

An area of opportunity was to use a thicker MDF as the 1/8 in thickness proved too slim for providing stable support and 2 boards had to be pasted together to obtain better stability which took time away from the manufacturing process. Starting off with 3/4 or similar thickness MDF would have helped cut manufacturing time while not increasing costs.

However, by far the greatest area for improvement was everything related to the conveyor belts themselves, from the motor couplers, to the bearings, to the bands themselves, they all had at least one issue starting with the axel that connected the couplers to the bearings which was composed of a thin metal rod that went trough a pair of bearing which were themselves mounted to a hollow cylinder about 1 in in diameter. Now, the first problem arose when we tried connecting those pieces as they were all made out of aluminum which turned out to be extremely hard to MIG weld properly as it would immediately cave in and deform. We then tried finding other ways to conjoin the pieces, eventually landing on using epoxy plasticine. The next problem arose with the bands themseves as joining one end to the other proved difficult as stapling them together made that section too straight for the bearing-induced rotation, and pasting using glue turned that area too thick and hard, but it was the version we ended up using. This stiffness caused hiccups and a considerable amount of momentary jamming, stressing out the band and demanding the motors draw more current, resulting in an unstable speed. Finally, at first we had 3 different kinds of motors which caused us to have to manufacture different couples for each model from scratch. When we eventually settled on using the 5840-31ZY motors we also bought some commercial flexible couples which solved the problem but by then the time spent on manufacturing

the custom ones had already been wasted.

4.2. PLCs and electronic design

During the development of the three stations, several recurring issues were encountered, the most significant being the instability of certain sensors when connected to specific Arduino pins. In several cases, sensors produced continuous output signals even when no object was present. Identifying the cause required extensive testing, during which multiple potential solutions were evaluated, including isolating the sensors from ambient light fluctuations, adding pull-up resistors, replacing sensors entirely, and verifying wiring, continuity, and voltage levels. Ultimately, the only effective solution was to reassign the sensor output to different Arduino pins.

Throughout the process, various connection-related problems also emerged. Each issue required careful analysis to identify its root cause, followed by targeted corrections to ensure system reliability. For instance, the RGB LEDs for the sorting station which indicated the capacity left for the packing boxers would routinely fail to turn on one of their colors, a problem that sometimes went away by itself but other times required very extensive testing to fix.

In terms of PLC programming, several modifications were made during development in response to behaviors observed during testing. For instance, timers were incorporated to achieve smoother and more stable operation. As an example, when certain sensors presented instability in their passive readings TON timers had to be implemented to guarantee an output was only given after the sensor maintained a consistent reading. By far the biggest hiccup was implementing the RGB color sensor on the sorting station's Arduino as the ladder diagram didn't seem capable of reading the presented color despite it achieving the task successfully when testing on an Arduino Uno. At least 2 days were wasted trying to fix the error until it became clear that the problem was not on the PLC itself but on the fact that when designing the PCB we set aside the same pins that were used on the Uno tests, however, the SDA and SCL pinout changes from the Uno platform to the Mega board which forced us to switch to a double-Arduino setup with the majority of the station's logic present on the Mega board and an additional Uno board soley responsible for the color sensor. This approach also had the advantage of allowing us to live debug as the feature is unavailable on PLCs with an Arduino extension active. Overall, however, the implementation of the PLC logic was straightforward and successful.

A separate set of challenges arose when uploading code to the Arduinos. On multiple occasions the devices failed to upload properly, the interface became unresponsive, or the system would not enter live debug mode. These issues were resolved through a combination of resetting the system, switching to a different COM port, and uploading a simple program to "unblock" the Arduino before loading the intended code. Although these failures occurred frequently, they did not follow a predictable pattern.

An additional improvement opportunity identified by the professor involved modifying the actuator control approach. Initially, the extension and retraction of the pistons operated strictly on timing-based logic. The recommended enhancement was to transition to a sensor-based feedback

implementation. Incorporating real-time feedback could increase system reliability and improve operational efficiency and precision. However, the team's counterpoint is that we already dealt with a plethora of issues relating to sensor malfunction and calibration and making another section depend on a finicky set of constraints could disable those systems from properly working. There's also the fact that the LD logic would become more complex as fallback measures would have to be implemented to prevent issues like the actuators stopping when the containers first reach the sensors as opposed to when they're centered. Testing needs to be carried out in order to determine the better approach.

4.3. Viscometer

When doing tests with the automatic mode, the viscometer failed to communicate with LabVIEW, and we were not able to present this final mode. We think that the reason may be that instead of using two ESP32s like all the other teams, we only implemented one and made all the connections in there. At the beginning, we were testing with the UART protocol the communication between LabVIEW and the ESP32 (hence the viscometer), obtaining the desired results. However, when integrating with the conveyor belt, implementing the IR sensors to communicate between the viscometer and the PLC, we used pins left like the ones made for UART communication: RX and TX. Theoretically, if UART was not implemented, those pins were available, and because we were going to change from UART to Bluetooth, we thought it was correct to make those connections. Perhaps this was not true, and maybe these connections affected the final communication between the ESP32 and LabVIEW.

Seeing that the other teams implemented two ESP32, mainly because of the timers and channels, and we did not have problems with that. Maybe the best approach was to use two ESP32 to occupy safer pins and have more available, instead of having just one and encountering the possible problem we had.

5. Conclusions

5.1. Oscar

This semester was the most challenging for me because of the variety of subjects we learned, the quantity of them to learn and master for this project, and our professional development, and because of the Tec Model, in which we see the subjects so fast that most of the times we can not fully understand them and we get left with questions that were solved during the theorical classes. However, due to the rhythm in which we are, we sometimes forget even the easiest actions we need to do to solve problems we encounter.

However, while we did not complete the final integration between the ESP32 and the PLCs modules, I feel satisfied because I was able to work in all the areas (mechanics design, schematic design, PCB design, the full programming of the final implementation of the viscometer, and full LabVIEW design), so my learning in this semester was beyond my expectations. I think that we successfully integrated the first system, which was the mechanics, electronics, programming, and the Visual Interface of the viscometer, and that is not a little, but actually a great achievement.

And considering that due to personal problem I had to take two jobs and work around 9 hours, with the school project, and considering that this was the most challenging semester of the career (at least according to ex-alumni and the professors), I'm glad that I was able to learn and participate in all the areas, and obtain maybe not a perfect work, but a very good work we do obtained.

5.2. Leo

What a ride. This was far and away the hardest semester of our degree and I would like to reflect for but a moment upon it. I would like to start off by stating some personal reasons that made it even harder than it already was. Due to financial hardships my friend Oscar had to undertake two different 30 hrs a week jobs, one remote and one all the way to Santa Fe. Naturally this rendered him permanently exhausted and it also meant he missed at least 9 or 10 hours that could have otherwise been dedicated to academic subjects like this project. On the other hand Frida has been part of multiple dance companies and teams within Tec that force her to spend a lot of time practicing with them or risk losing her place. This meant that she was unavailable to work for long periods of time each and every day. Finally, I started a research assistantship with the Massachusetts Institute of Technology that demanded a minimum commitment of 25 hours a week, which often times turned into a number closer to 30 and also detracted from focusing as much on this project and other academic endeavors as I would have liked. This meant that 3 of our team members were effectively part-timers which ended up with the consequence of leaving Victor with a lot more work than he otherwise would have, who himself dedicates many hours

to his bodybuilding career. As professor Navarro graciously expressed "we shot ourselves on the foot" by choosing this lineup. Not because we are lazy or don't work, but because of the opposite. Perhaps we bit more than we could chew, but I still absolutely feel proud of what we accomplished.

Going back to more technical aspects, I enjoyed this project quite a bit. I never felt as lost as I did with last semester's AGV and I always had at least a basic notion of what everyone else was doing unlike last semester where everyone had to pick a specialty and soley focus on that. I personally focused on the manufacturing, PLC programming, and electrical design.

On the manufacturing design, I was afraid of our decision of making the stations out of wood as I had never handled big machinery like the tablesaw and I thought I would need help from a professor each and every time I needed to cut something which started off as being the case, however I exhausted Quique, on of the laboratorists at the wood workshop that he decided to teach and certify me in the use of the heavy machinery after which manufacturing was mostly a breeze except for the rollers and conveyor belts themselves.

In PLC programming, I grew more confident in my ability to program LDs and ran into very few issues with them. However, my greatest challenge would present itself when I tried to implement the color senor on the same program as the rest of the sorting system. This forced me to become an expert in I2C, Arduino IDE file management, cache management, and OpenPLC's Arduino extension in order to fix the problem to which I was eventually able to find a workaround but only after spending way to many days on it.

5.3. Frida

Overall, the development of this automated system was both a highly challenging and deeply rewarding experience. Working through the design, implementation, and troubleshooting of the three stations required sustained effort, creativity, and perseverance. Despite the technical difficulties, such as sensor instability, mechanical inconsistencies, PCB modifications, and communication issues between microcontrollers, the process was genuinely enjoyable. Each obstacle became an opportunity to learn, experiment, and strengthen my understanding of electronics, automation, PLC programming, and system integration.

I can confidently say that I learned an enormous amount throughout this project, not only about the technical components involved but also about structured problem-solving, debugging methodologies, and collaborative design. However, I also recognize that the time allocated to complete a project of this scale is quite limited. Given the overall academic workload of the semester, achieving a fully functional system demanded extensive hours and constant dedication. With additional time, many aspects could have been refined further, especially in terms of mechanical optimization and transitioning from timing, based control to sensor, driven feedback.

Even so, the project was an enriching experience that strengthened my confidence and enthusiasm for automation and control systems.

5.4. Victor

From my point of view, the project managed to meet its main objectives, especially in terms of integrating different subsystems into a single automated process. Being able to coordinate mechanical components, electronics, control logic, and communication helped me understand how complex automation systems function in practice. Even though we faced challenges during calibration, wiring, and mechanical alignment, the final system demonstrated that the core idea of measuring, adjusting, and classifying viscosity was successfully achieved.

Regarding industrial applications, I can see this system being useful in environments where fluid consistency is critical. Industries such as cleaning products, food processing, or chemical manufacturing could benefit from an automated viscosity control station to improve quality control and reduce manual intervention. With more robust components and further optimization, this concept could be adapted to real production lines. Overall, the project helped me better connect theoretical knowledge with real world automation challenges and applications.

5.5. Max

I believe this project achieved its main objectives in a solid and meaningful way. The system we developed successfully integrates mechanical design, electronics, control, and communication into a single automated process capable of measuring, adjusting, and classifying viscosity. Seeing all subsystems work together helped me understand how theoretical concepts from automation, thermofluids, and control translate into a real, functional solution. Even with the challenges we faced during calibration, wiring, and mechanical stability, the final result showed that our design decisions were aligned with the original goals of reliability, modularity, and process automation.

From an industrial perspective, I can clearly see how this system could be adapted to real production environments. Industries such as food processing, cleaning products, cosmetics, and chemicals could benefit from an automated viscosity control station like this to improve quality control, reduce human error, and increase process repeatability. With more robust materials, improved calibration procedures, and industrial grade sensors and actuators, the system could scale beyond an academic prototype and become part of a continuous production line. This project helped me better understand the complexity and value of integrated automation systems and reinforced the relevance of mechatronic solutions in modern industrial applications.

It was also difficult having to work at a proyect that has complex in many way´s and have to work in my part time job, which made difficult the time managing for all the semestrer, as well, the late time when I had to finish both deliverables for my job and school. Although it was tough, I would like to practice much more in all topics and apply it as much as I can in my future.

References

[1] K. Collins and S. Willis, “DRAFT PLC Programming for Industrial Automation,” 2014.

[2] I. Cayetano, ’Actuators’, Tecnológico de Monterrey Mexico City Campus, 2021.

[3] N. Jones, "Brush DC Motor Guide". Anaheim Automation. https://anaheimautomation.com/blog/post/brush-dc-motor-guide

[4] "Types of DC Motors", Monolithic Power Systems, https://www.monolithicpower.com/en/learning/mpscholar/electric-motors/dc-motors/types

[5] "Brushed vs brushless DC motors: Key differences", Arrow. https://www.arrow.com/en/research-and-events/articles/which-dc-motor-is-best-for-your-application

[6] "Stepper Motor Basics & Working Principle", ATO.com, https://www.ato.com/stepper-motor-basics-working-principle

[7] C. Fione, "Stepper Motors Basics: Types, Uses, and Working Principles". Monolithic Power Systems. https://www.monolithicpower.com/en/learning/resources/stepper-motors-basics-types-uses

[8] I. Cayetano, ’Sensor classification, characterization and calibration’. Tecnológico de Monterrey, Mexico City Campus, 2023.

[9] J. Cook, "Ultrasonic Sensors: How They Work (and How to Use Them with Arduino)". Arrow. https://www.arrow.com/en/research-and-events/articles/ultrasonic-sensors-how-they-work-and-how-to-use-them-with-arduino

[10] "What is a Color Sensor: Principles, Comparisons, and Diverse Applications". Bedook Sensors. https://www.bedooksensors.com/what-is-a-color-sensor/

[11] "What is the Motor Encoder?", Manufacturing Tomorrow, https://www.manufacturingtomorrow.com/news/2023/12/18/what-is-the-motor-encoder/21919/

[12] W. Burke, "Understanding Quadrature Encoders: A Comprehensive Guide for Mechanical Engineers". Five Flute. https://www.fiveflute.com/guide/understanding-quadrature-encoders-a-comprehensive-guide-for-mechanical-engineers/

[13] "The PID Controller & Theory Explained", National Instruments. https://www.ni.com/en/shop/labview/pid-theory-explained.html?srsltid=AfmBOoo71Q1CMOhwGRk6C1mLinzcNlby7O3yL2Z5kvX_vubSwGW9u1EB

[14] I. Cayetano, ’Communication Protocols for Microcontrollers’, Tecnológico de Monterrey Mexico City Campus, 2025

[15] R. Ganem, 'Viscosity Fall 2025'. Tecnológico de Monterrey Mexico City Campus, 2025

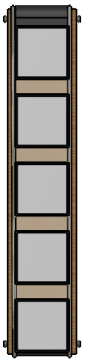
[16] R. Ganem, 'Mass and Energy Conservation Bernoulli equation'. Tecnológico de Monterrey Mexico City Campus, 2025

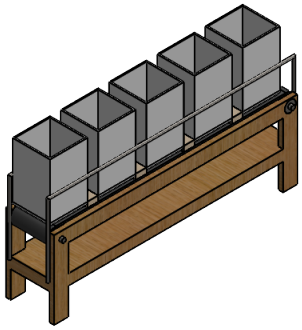
[17] R. R. Garcia, 'Operational Amplifiers', Tecnológico de Monterrey Mexico City Campus.

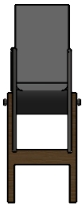
[18] I. Cayetano, 'PCB Design and Manufacturing', Tecnológico de Monterrey, Mexico City Campus, 2021.

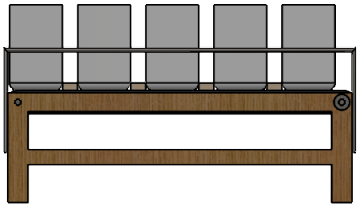
A. Stack System Measurements

A.1. Container Conveyor Belt Drawings

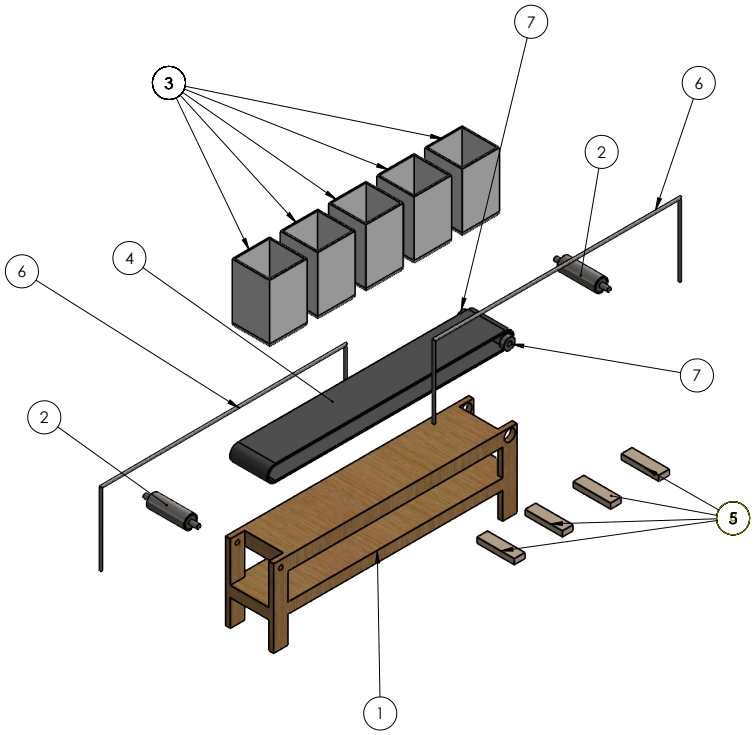








		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS	DRAWN			
		TOLERANCES: FRACTIONAL ±	CHECKED			TITLE:
		ANGULAR: MACH ± BEND ±	ENG APPR.			
		TWO PLACE DECIMAL ±	MFG APPR.			
		THREE PLACE DECIMAL ±	Q.A.			
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:			SIZE B / DWG. NO. Assembly / REV
		MATERIAL				
NEXT ASSY	USED ON	FINISH				SCALE: 1:5 / WEIGHT: / SHEET 1 OF 9
APPLICATION		DO NOT SCALE DRAWING				



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	base		1
2	Roller_Assembly		2
3	Envase		5
4	cinturon		1
5	separadores		4
6	Barandal G		2
7	Balero		2

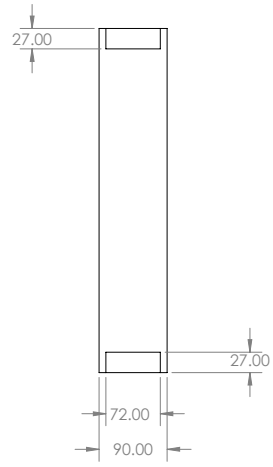
PROPRIETARY AND CONFIDENTIAL

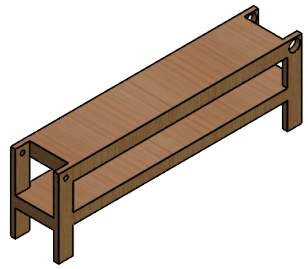
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

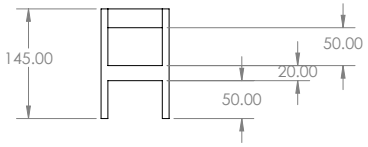
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

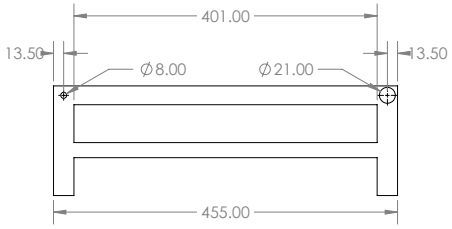
TITLE:

SIZE	DWG. NO.	REV
B	Exploded	
SCALE: 1:5	WEIGHT:	SHEET 2 OF 9





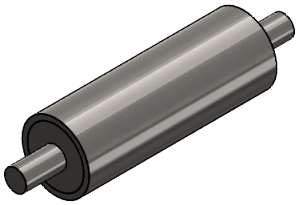




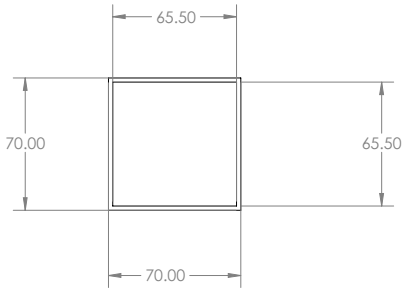
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES:	CHECKED		
		FRACTIONAL ±	ENG APPR.		
		ANGULAR: MACH ± BEND ±	MFG APPR.		
		TWO PLACE DECIMAL ±	Q.A.		
		THREE PLACE DECIMAL ±	COMMENTS:		
		INTERPRET GEOMETRIC TOLERANCING PER:			
		MATERIAL: Madera			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

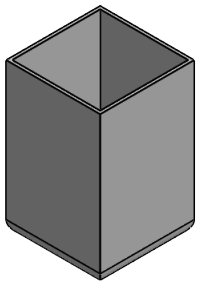
TITLE:

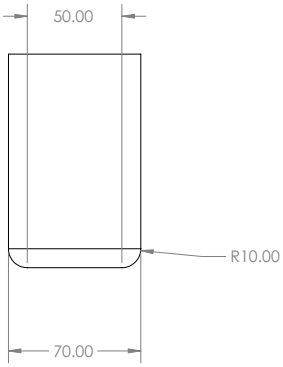
SIZE	DWG. NO.	REV
B	Base	
SCALE: 1:5	WEIGHT:	SHEET 3 OF 9

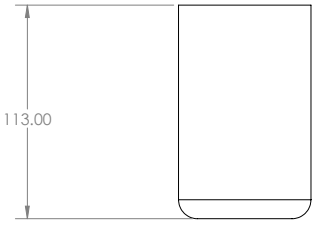


		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS	DRAWN			
		TOLERANCES:	CHECKED			TITLE:
		FRACTIONAL ±	ENG APPR.			
		ANGULAR: MACH ± BEND ±	MFG APPR.			
		TWO PLACE DECIMAL ±	Q.A.			
		THREE PLACE DECIMAL ±	COMMENTS:			SIZE DWG. NO. REV
		INTERPRET GEOMETRIC TOLERANCING PER:				B Roller Assembly
		MATERIAL See its technical draw				
NEXT ASSY	USED ON	FINISH				SCALE: 1:1 WEIGHT: SHEET 4 OF 9
APPLICATION		DO NOT SCALE DRAWING				









PROPRIETARY AND CONFIDENTIAL

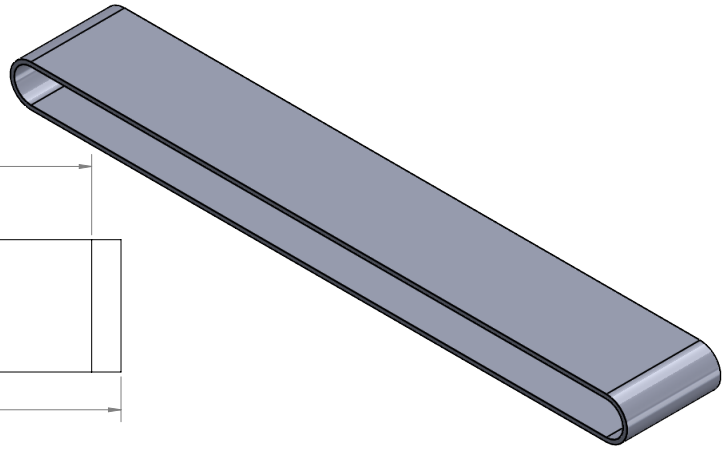
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

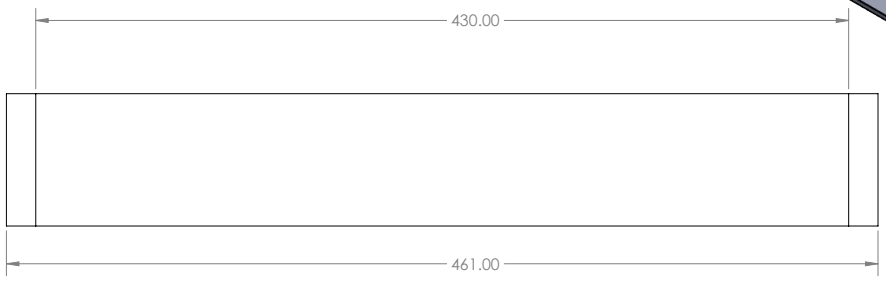
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL Plastic			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

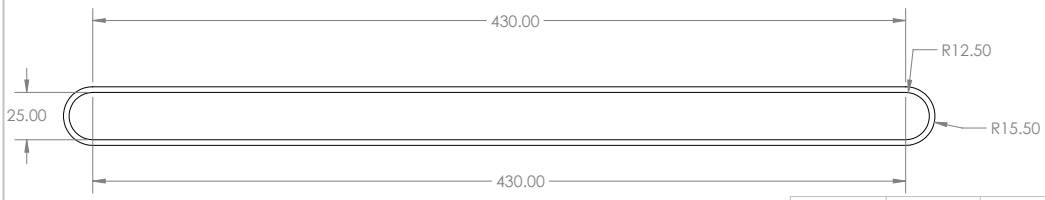
TITLE: Electrolit

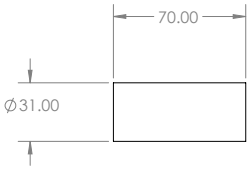
SIZE	DWG. NO.	REV
B	Envase	

SCALE: 1:2 | WEIGHT: | SHEET 5 OF 9





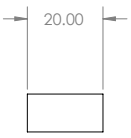


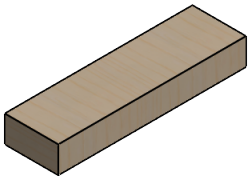


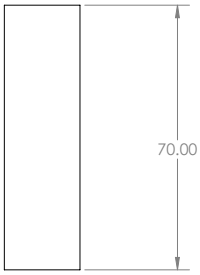
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Rubber	COMMENTS:		
		FINISH			
NEXT ASSY	USED ON				
APPLICATION		DO NOT SCALE DRAWING			

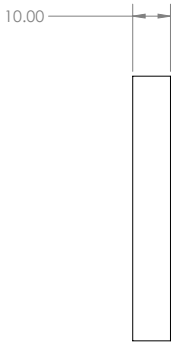
TITLE:

SIZE	DWG. NO.	REV
B	Cinturon	
SCALE: 1:2	WEIGHT:	SHEET 6 OF 9

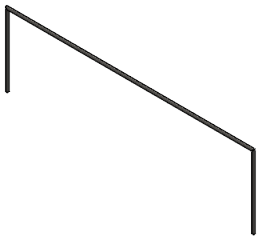


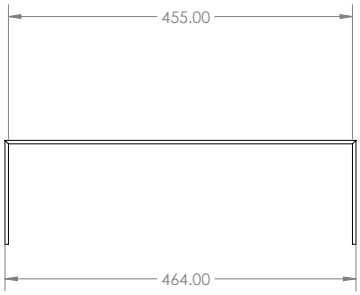


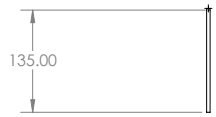




		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS	DRAWN			
		TOLERANCES: FRACTIONAL ±	CHECKED			TITLE:
		ANGULAR: MACH ± BEND ±	ENG APPR.			
		TWO PLACE DECIMAL ±	MFG APPR.			
		THREE PLACE DECIMAL ±	Q.A.			
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:			
		MATERIAL Madera				SIZE B / DWG. NO. Separadores / REV
NEXT ASSY	USED ON	FINISH				
APPLICATION		DO NOT SCALE DRAWING				SCALE: 1:1 / WEIGHT: / SHEET 7 OF 9



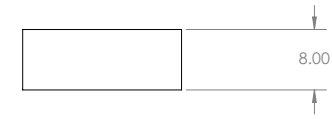


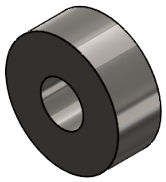


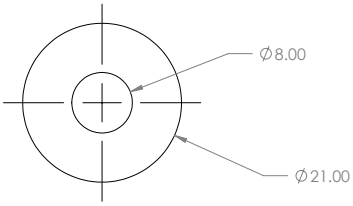
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL Acero			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
B	Barandal G	
SCALE: 1:5	WEIGHT:	SHEET 8 OF 9





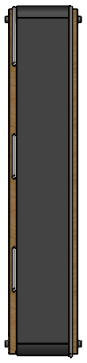


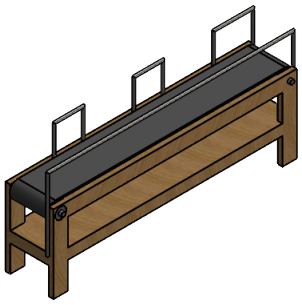
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES:	CHECKED		
		FRACTIONAL ±	ENG APPR.		
		ANGULAR: MACH ± BEND ±	MFG APPR.		
		TWO PLACE DECIMAL ±	Q.A.		
		THREE PLACE DECIMAL ±	COMMENTS:		
		INTERPRET GEOMETRIC TOLERANCING PER:			
		MATERIAL Acero/Aluminio			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

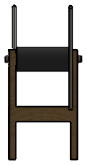
SIZE	DWG. NO.	REV
B	Balero	
SCALE: 2:1	WEIGHT:	SHEET 9 OF 9

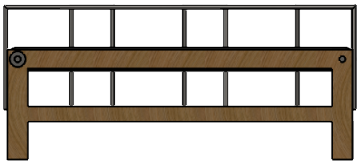
A.2. Main Conveyor Belt Drawings





ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	base		1
2	Roller_Assembly		2
3	cinturon		1
4	barandal G		1
5	barandal P		2
6	barandal M		1
7	roller_bearing		2

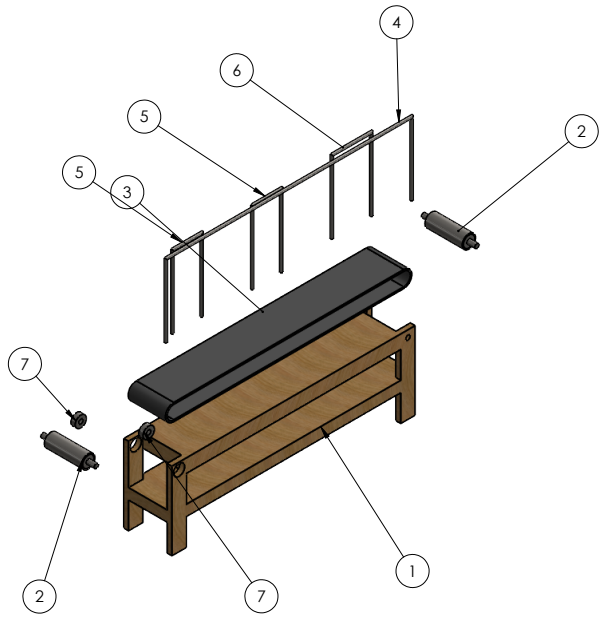




PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS	DRAWN			
		TOLERANCES: FRACTIONAL ±	CHECKED			TITLE:
		ANGULAR: MACH ± BEND ±	ENG APPR.			
		TWO PLACE DECIMAL ±	MFG APPR.			
		THREE PLACE DECIMAL ±	Q.A.			
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:			
		MATERIAL				SIZE B — DWG. NO. Assembly — REV
NEXT ASSY	USED ON	FINISH				
APPLICATION		DO NOT SCALE DRAWING				SCALE: 1:5 — WEIGHT: — SHEET 1 OF 9



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	base		1
2	Roller_Assembly		2
3	cinturon		1
4	barandal G		1
5	barandal P		2
6	barandal M		1
7	roller_bearing		2

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MILIMETERS
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

INTERPRET GEOMETRIC TOLERANCING PER:

MATERIAL

FINISH

DO NOT SCALE DRAWING

NEXT ASSY | USED ON | APPLICATION

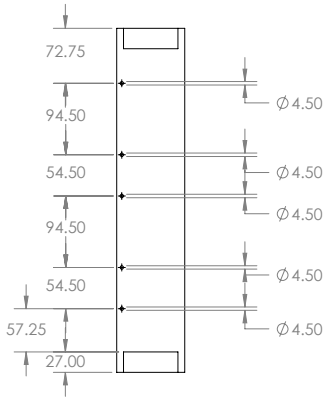
	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

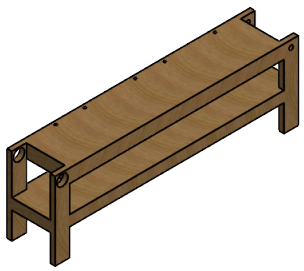
COMMENTS:

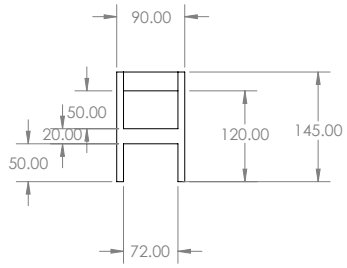
TITLE:

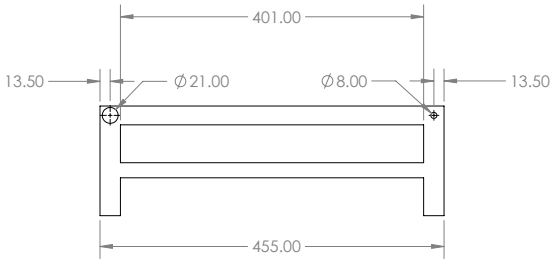
SIZE B | DWG. NO. Exploded View | REV

SCALE: 1:5 | WEIGHT: | SHEET 2 OF 9

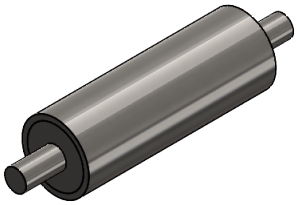




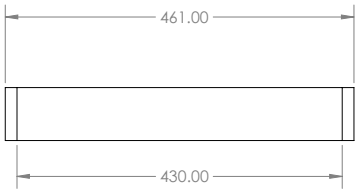


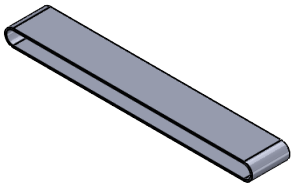


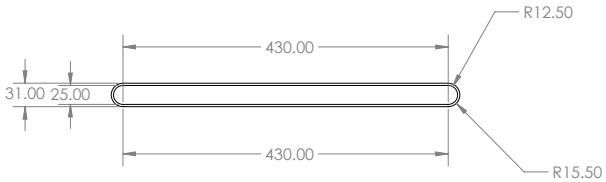
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN			TITLE:
			CHECKED			
			ENG APPR.			
			MFG APPR.			
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
		MATERIAL Madera	COMMENTS:			SIZE B / DWG. NO. Base / REV
NEXT ASSY	USED ON	FINISH				SCALE: 1:5 / WEIGHT: / SHEET 3 OF 9
APPLICATION		DO NOT SCALE DRAWING				

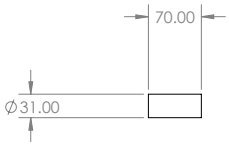


		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN MILIMETERS	DRAWN			
		TOLERANCES: FRACTIONAL ±	CHECKED			TITLE:
		ANGULAR: MACH ± BEND ±	ENG APPR.			
		TWO PLACE DECIMAL ±	MFG APPR.			
		THREE PLACE DECIMAL ±	Q.A.			
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:			
		MATERIAL See its Tech Drawing				SIZE B DWG. NO. Roller Assembly REV
NEXT ASSY	USED ON	FINISH				
APPLICATION		DO NOT SCALE DRAWING				SCALE: 1:1 WEIGHT: SHEET 4 OF 9





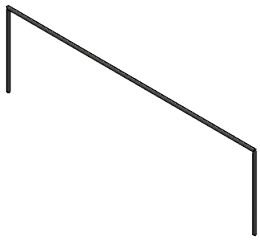


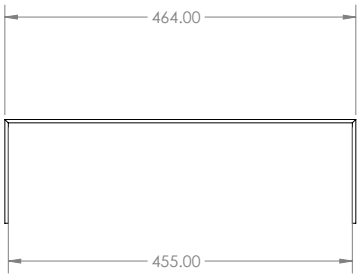


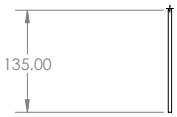
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL Goma			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
B	Cinturon	
SCALE: 1:5	WEIGHT:	SHEET 5 OF 9



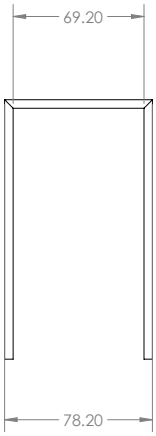


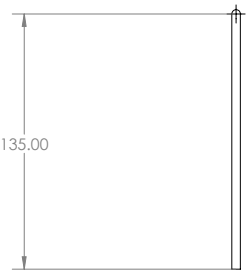


		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL: Acero			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
B	Barandal G	
SCALE: 1:5	WEIGHT:	SHEET 6 OF 9

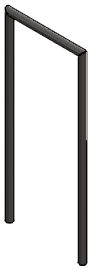


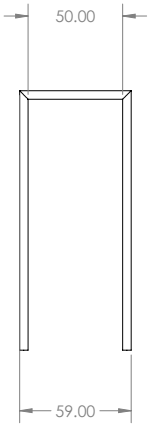


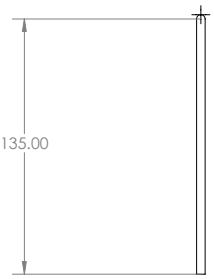
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL Acero			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
B	Barandal M	
SCALE: 1:2	WEIGHT:	SHEET 7 OF 9



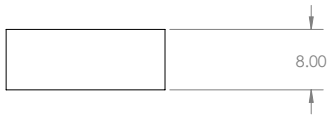


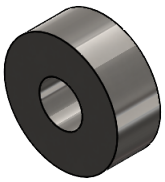


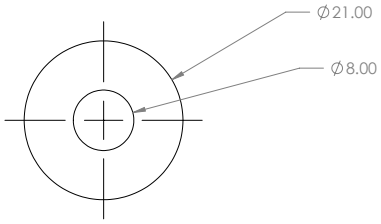
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS	DRAWN		
		TOLERANCES: FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
		MATERIAL: Acero			
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
B	Barandal P	
SCALE: 1:2	WEIGHT:	SHEET 8 OF 9





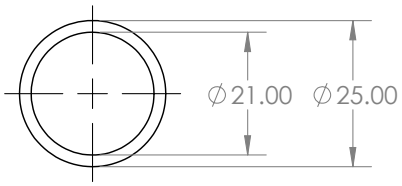


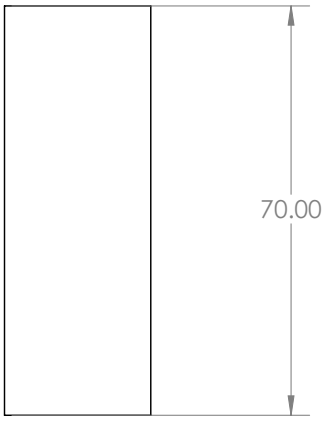
UNLESS OTHERWISE SPECIFIED:		NAME / DATE
DIMENSIONS ARE IN MILIMETERS	DRAWN	
TOLERANCES:	CHECKED	
FRACTIONAL ±	ENG APPR.	
ANGULAR: MACH ± BEND ±	MFG APPR.	
TWO PLACE DECIMAL ±	Q.A.	
THREE PLACE DECIMAL ±	COMMENTS:	
INTERPRET GEOMETRIC TOLERANCING PER:		
MATERIAL: Aluminio		
FINISH		
NEXT ASSY / USED ON		
APPLICATION	DO NOT SCALE DRAWING	

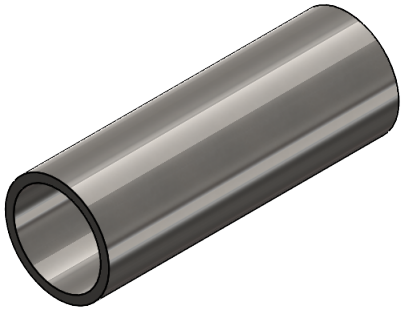
TITLE:

SIZE	DWG. NO.	REV
B	Balero	
SCALE: 2:1	WEIGHT:	SHEET 9 OF 9

A.3. Fixed Tube Drawings



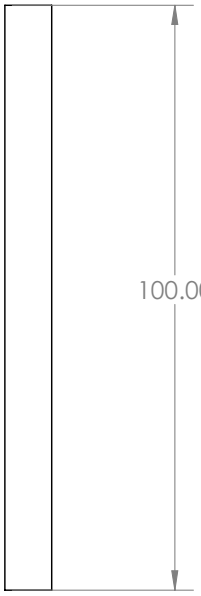


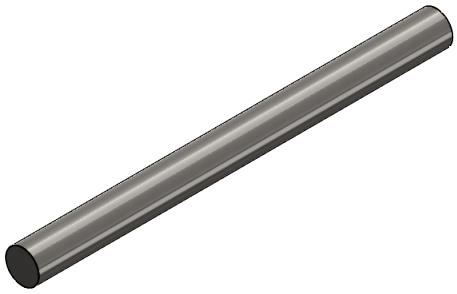


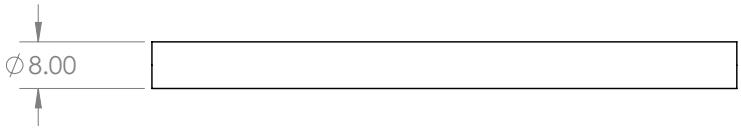
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Acero	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
A	Tubo	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 4







NEXT ASSY	USED ON	
APPLICATION		

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MILIMETERS
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±

INTERPRET GEOMETRIC TOLERANCING PER:

MATERIAL: Acero

FINISH:

DO NOT SCALE DRAWING

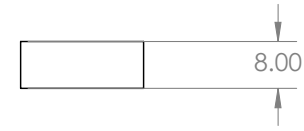
	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

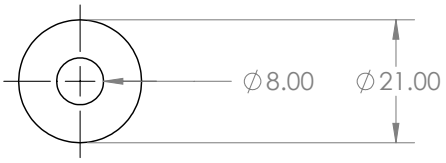
COMMENTS:

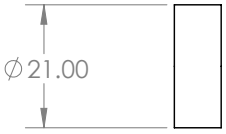
TITLE:

SIZE	DWG. NO.	REV
A	Eje - Varilla	

SCALE: 1:1	WEIGHT:	SHEET 2 OF 4



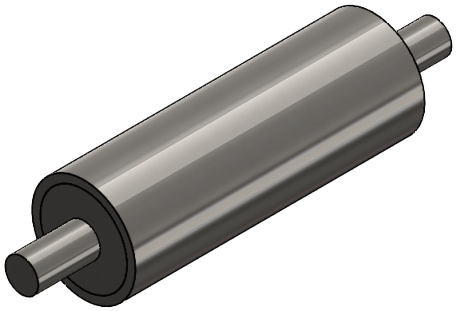




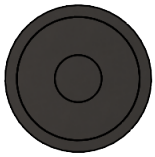
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Acero	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

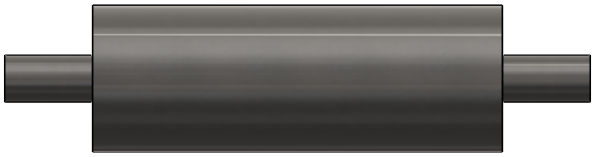
TITLE:

SIZE	DWG. NO.	REV
A	Hoja	
SCALE: 1:1	WEIGHT:	SHEET 3 OF 4



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	roller_eje		1
2	roller_tube		1
3	roller_Hoja_Acero		2



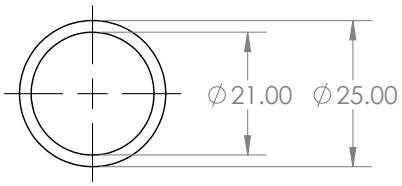


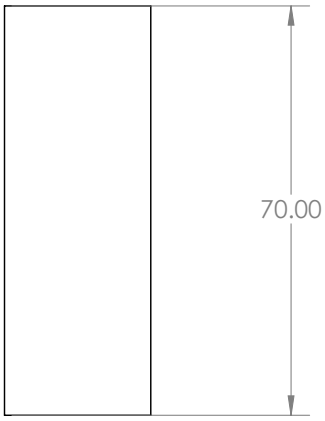
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

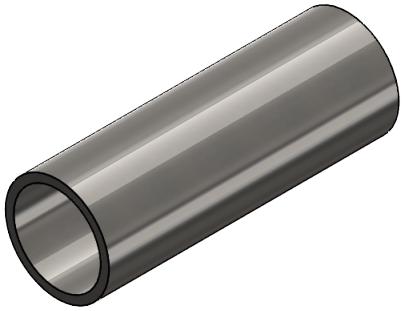
TITLE:

SIZE	DWG. NO.	REV
A	Assembly	
SCALE: 1:1	WEIGHT:	SHEET 4 OF 4

A.4. Spinning Tube Drawings



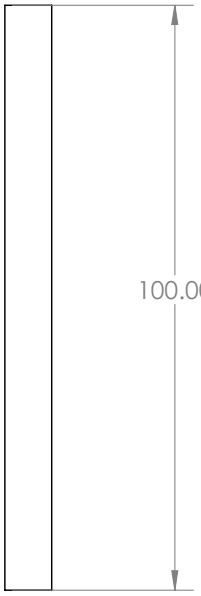


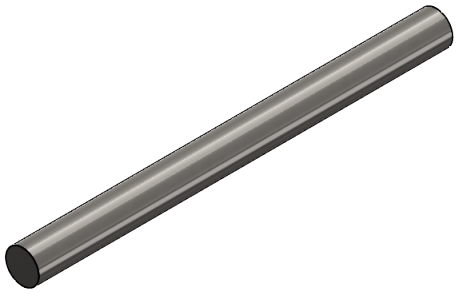


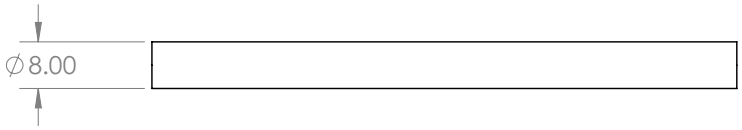
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Acero	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
A	Tubo	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 4



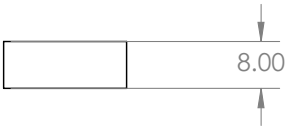


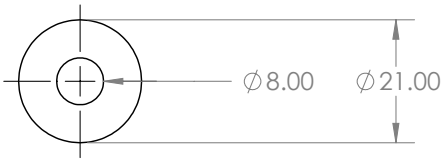


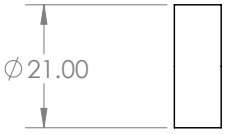
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Acero	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
A	Eje - Varilla	
SCALE: 1:1	WEIGHT:	SHEET 2 OF 4



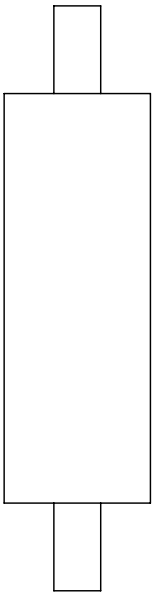


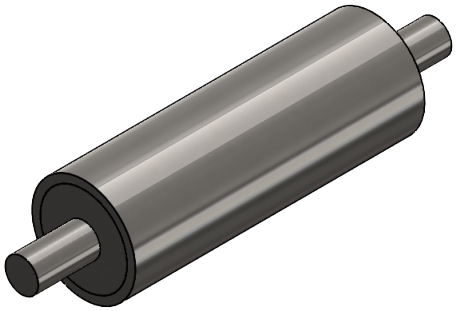


		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN MILIMETERS TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL Aluminio	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

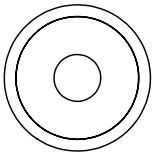
TITLE:

SIZE	DWG. NO.	REV
A	Balero	
SCALE: 1:1	WEIGHT:	SHEET 3 OF 4





ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	roller_eje		1
2	roller_tube		1
3	roller_bearing		2



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE **A** | DWG. NO. Assembly | REV

SCALE: 1:1 | WEIGHT: | SHEET 4 OF 4

B. Viscometer CAD drawings

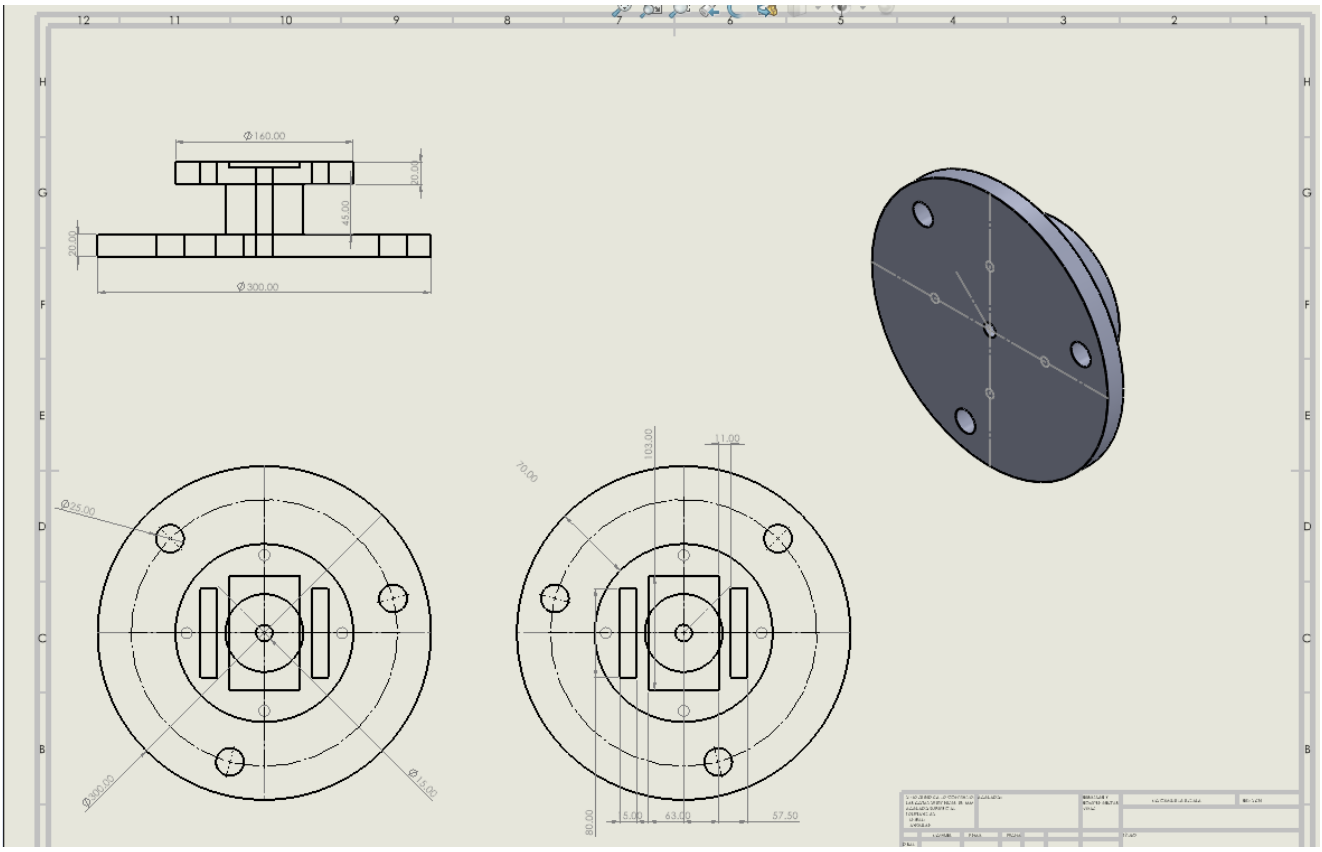


Figure 30: CAD Model 1

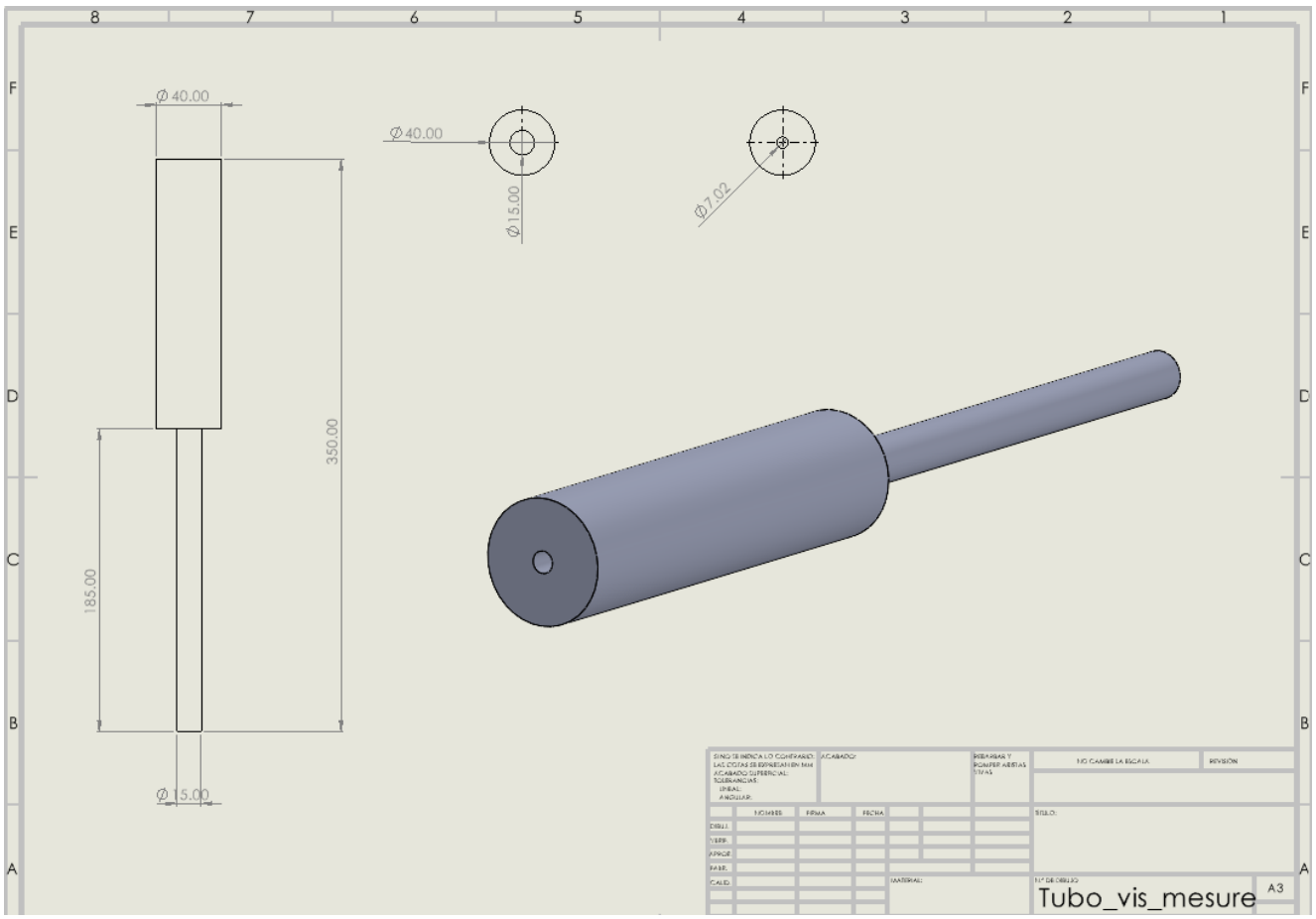


Figure 31: CAD Model 2

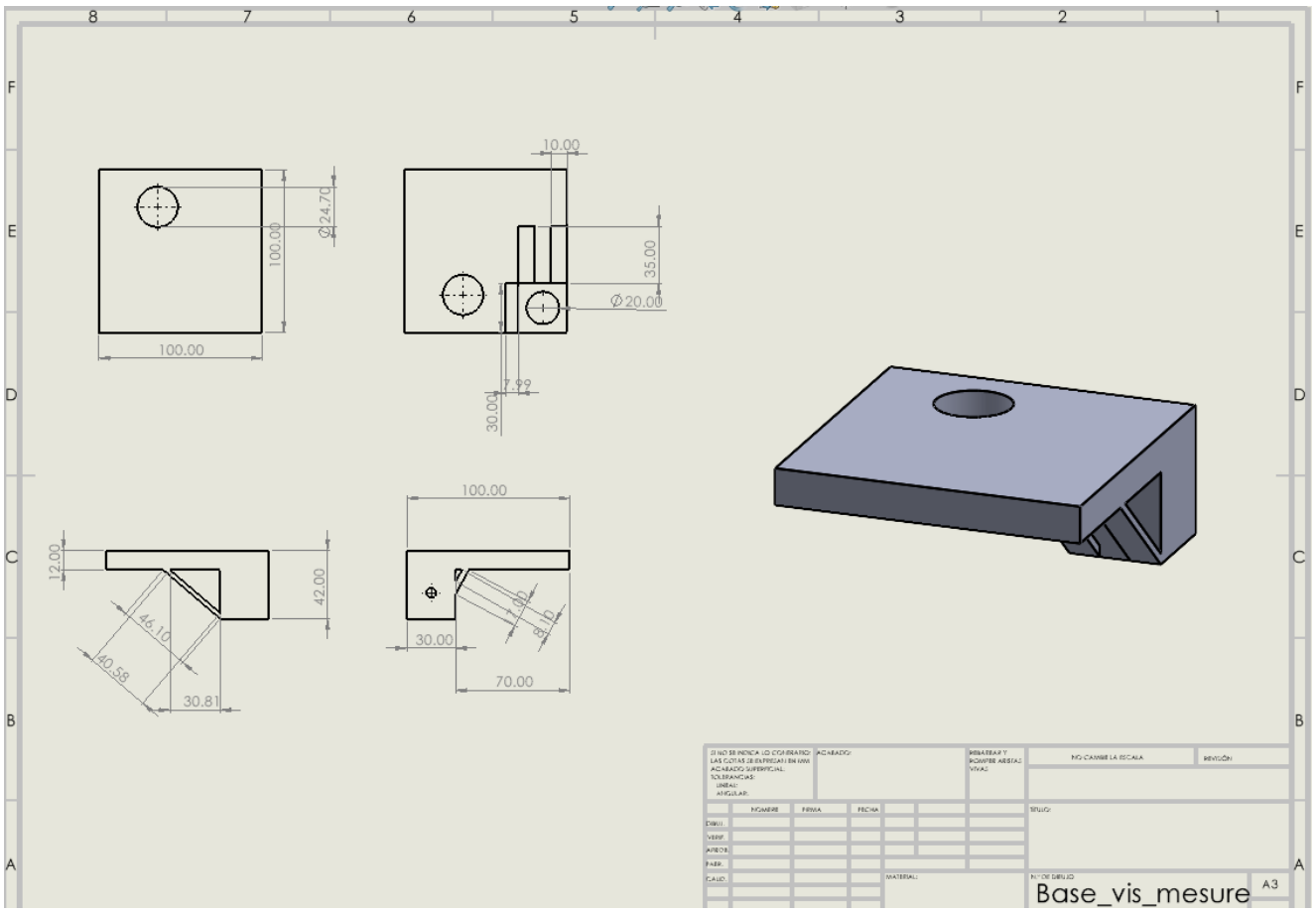


Figure 32: CAD Model 3

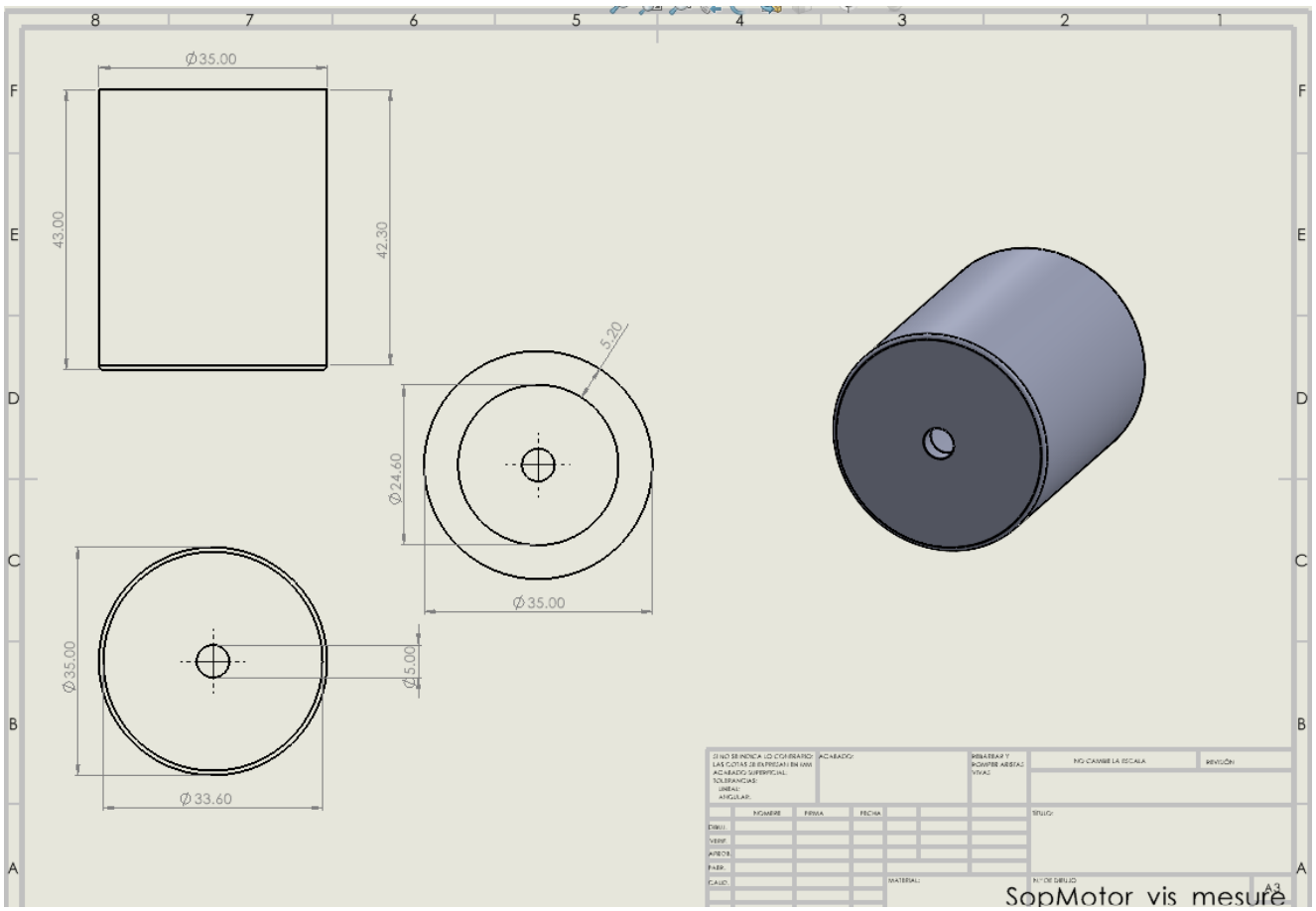


Figure 33: CAD Model 4

C. Viscometer C++ Code

C.1. definitions.h

Listing 1: definitions.h C++ code

```
#ifndef _DEFINITIONS_H_
#define _DEFINITIONS_H_

// ================================================================================
// SYSTEM INCLUDES
// ================================================================================
#include <esp_timer.h>
#include <esp_task_wdt.h>
#include <math.h>
#include <string.h>
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include <cstring>
#include <cstdio>

// ================================================================================
// LIBRARY INCLUDES
// ================================================================================
#include "SimpleADC.h"
#include "SimpleGPIO.h"
#include "SimplePWM.h"
#include "SimpleRGB.h"
#include "PistonDC.h"
#include "dcMotorController.h"
#include "stepperMotor.h"
#include "esp_system.h"
#include "SpindleVisc.h"
#include "SimpleUART.h"
#include "AxelF_CrazyFrog.h"
#include "HBridge.h"
#include "SimpleI2C.h"
#include "Ultrasonic.h"
#include "RGBColorSensorTCS34725.h"
#include "SimpleSerialBT.h"

```

```
37 // ============================================================================
38 // ENUM & LABVIEW INPUT/OUTPUT DEFINITIONS
39 // ============================================================================
40
41 // Machine states of the manual control system
42 enum ManualState
43 {
44     MANUAL_IDLE = 0,
45     PISTON_UP = 1, // Button
46     PISTON_DOWN = 2, // Button
47     STEPPER_CW = 3, // Button
48     STEPPER_CCW = 4, // Button
49     STEPPER_SPECIFIC = 5, // UART
50     STEPPER_FREQ = 6,
51     SPINDLE_PWM = 7, // Knob
52     SPINDLE_RPMs = 8, // Knob
53     SPINDLE_MEASURE_VISCOSITY = 9, // Button
54     PUMP_WATER = 10, // Button
55     DRYER_ON = 11, // Button
56 };
57
58 // Switch case state variable
59 ManualState manual_case_state = MANUAL_IDLE;
60 ManualState previous_state = MANUAL_IDLE; // Track previous state for state transitions
61
62 // Machine states of the automatic control system
63 enum AutomaticState
64 {
65     HOMING_ROUTINE = 0,
66     AUTOMATIC_IDLE = 1,
67     CONTAINER_POSITIONING = 2,
68     DETECT_TAG_COLOR = 3,
69     MEASURE_VISCOSITY = 4,
70     DOSSING_ADJUSTMENT = 5,
71     CLEANING_STATION = 6,
72     DRYING_STATION = 7
73 };
74
75 // Switch case state variable
76 AutomaticState automatic_case_state = AUTOMATIC_IDLE;
77
78 // LabVIEW inputs
```

```
79 int btn_piston_up = 0;
80 int btn_piston_down = 0;
81 int btn_stepper_cw = 0;
82 int btn_stepper_ccw = 0;
83 float knob_stepper_angle = 0.0f;
84 float knob_stepper_frequency = 0.0f;
85 float knob_spindle_pwm = 0.0f;
86 float knob_spindle_rpm = 0.0f;
87 int btn_measure_viscosity = 0;
88 int btn_water_pump = 0;
89 int btn_dryer_on = 0;
90 int state_machine_case = 0;
91
92 // LabVIEW outputs
93 float piston_position = 0.0f;
94 float stepper_angle = 0.0f;
95 float stepper_speed = 0.0f;
96 float spindle_rpms = 0.0f;
97 float spindle_current = 0.0f;
98 float viscosity_avrg = 0.0f;
99 uint16_t r_sensor_output = 0, g_sensor_output = 0, b_sensor_output = 0, c_sensor_output
       = 0;
100 int message_box_case = 0;
101
102 // ============================================================================
103 // PIN & CLASSES DEFINITIONS
104 // ============================================================================
105
106 // UART
107 SimpleUART uart(115200);
108
109 // BT
110 SerialBT bt;
111
112 // ================================ ACTUATORS ================================
113
114 // DC Motor Timer Configuration (shared by Spindle, Water Pump, Dryer Motor)
115 TimerConfig dcMotorTimerConfig =
116 {
117     .timer = LEDC_TIMER_0,
118     .bit_resolution = LEDC_TIMER_8_BIT,
119     .mode = LEDC_HIGH_SPEED_MODE
```

```
};

// Spindle DC motor
uint8_t dcMotor_pins[2] = {32, 33}; // IN1, IN2
// Spindle uses DC motor timer 0 (high speed) and channels 0,1
uint8_t dcMotor_channels[2] = {0, 1};

uint8_t encoder_pins[2] = {2, 13}; // ENC_A (Amarillo), ENC_B (Verde)

float pulses_per_rev = 291.0f; // datasheet = 341.2f, real = 291.0f

float degrees_per_edge = 360.0f / (pulses_per_rev * 2);

int64_t encoder_timeout_us = 100000; // 100 ms

#define VISCOSITY_ADC_PIN 36 // C-V - Changed from pin 2 to avoid conflicts
float max_voltage = 1900.0f; // mV at 100% soap
float min_voltage = 33.0f;    // mV at 0% soap
float max_water_percentage = 25.0f;
float min_water_percentage = 0.0f;

float gains[3] = {0.1156f, 43.2543f, 0.0f}; // PID gains: Kp, Ki, Kd. DEFINE WITH TF
float spindle_dt_s = 0.001f; // dcMotorController internal timing (NOT main loop dt)

float m_slope = 0.0542f;    // Feedforward slope (deg/s to PWM)
float b_intercept = 14.289f; // Feedforward intercept (PWM at 0 deg/s)

float m_conditioning = 1.0f; // Slope for current conditioning
float b_conditioning = 2.51f; // Intercept for current conditioning

// Spindle configuration struct - groups all related parameters
SpindleConfig spindle_config = {
    .hbridge_pins = dcMotor_pins,
    .hbridge_channels = dcMotor_channels,
    .motor_config = &dcMotorTimerConfig,
    .encoder_pins = encoder_pins,
    .degrees_per_edge = degrees_per_edge,
    .encoder_timeout_us = encoder_timeout_us,
    .viscosity_adc_pin = VISCOSITY_ADC_PIN,
    .max_voltage = max_voltage,
    .min_voltage = min_voltage,
    .max_water_percentage = max_water_percentage,
```

```
162     .min_water_percentage = min_water_percentage,
163     .m_slope = m_slope,
164     .b_intercept = b_intercept,
165     .m_conditioning = m_conditioning,
166     .b_conditioning = b_conditioning
167 };
168 
169 SpindleVisc spindle;
170 
171 // Stepper Motor - Controls the X-movement
172 uint8_t steppper_gpio_pins[2] = {26, 27}; // DIR, ENA
173 uint8_t stepper_pul_pin = 25; // PUL
174 uint8_t stepper_pul_feedback_pin = 35; // Pulse feedback from TB6600 (GPIO 35 - ADC
        input)
175 uint8_t stepper_pwm_channel = 6; // Changed from 2 to 6 to avoid conflict with water
        pump
176 float stepper_duty_percentage = 50.0f; // 50% duty cycle for square wave
177 
178 TimerConfig stepper_vel_config =
179 {
180     .timer = LEDC_TIMER_1,
181     .frequency = 5000,  // Higher base frequency for better PWM control
182     .bit_resolution = LEDC_TIMER_10_BIT,  // Reduced from 12-bit to prevent duty errors
183     .mode = LEDC_HIGH_SPEED_MODE
184 };
185 
186 uint32_t stepper_frequency = 100; // Initial frequency
187 
188 // Calculated from test: 1600 gave 270   for 180   cmd, so 1600    (180/270) = 1067
189 uint32_t pulses_per_rev_stepper = 800; // Empirically calibrated
190 
191 float gains_stepper[3] = {1.0f, 0.0f, 0.0f}; // PID gains: Kp, Ki, Kd. Increased Kp for
        faster response
192 
193 int stepper_case = 0; // test
194 
195 StepperMotor stepper;
196 
197 // H-Bridge - Y-axis piston control
198 uint64_t PistonPins[2] = {14, 12}; // IN1, IN2
199 gpio_mode_t PistonModes[2] = {GPIO_MODE_OUTPUT, GPIO_MODE_OUTPUT};
200 
```

```
201 PistonDC piston;
202
203 // Water Pump
204 uint8_t water_pump_pins[2] = {19, 255};
205 // Water pump shares DC motor timer 0 (high speed), uses channels 2,3
206 uint8_t water_channels[2] = {2, 3};
207
208 HBridge water_pump;
209
210 // Drying motor
211 uint8_t dryer_pins[2] = {23, 255};
212 // Dryer motor shares DC motor timer 0 (high speed), uses channels 4,5
213 uint8_t dryer_channels[2] = {4, 5};
214
215 HBridge dryer_motor;
216
217 // RGB LED Configuration
218 uint8_t RGB_pins[3] = {18, 17, 16};
219 // RGB has dedicated timer 2 low speed, channels 0,1,2 (three channels)
220 uint8_t RGB_channels[3] = {0, 1, 2};
221 TimerConfig RGB_config =
222 {
223     .timer = LEDC_TIMER_2,
224     .frequency = 40,
225     .bit_resolution = LEDC_TIMER_12_BIT,
226     .mode = LEDC_LOW_SPEED_MODE
227 };
228
229 SimpleRGB RGB_led;
230
231 // ================================ SENSORS ================================
232
233 // Ultrasonic Sensor
234 #define TRIG_PIN 4
235 #define ECHO_PIN 34
236 // Ultrasonic has dedicated timer 3 low speed, channel 0
237 uint8_t ultrasonic_trig_channel = 0;
238 TimerConfig ultrasonic_timer_config =
239 {
240     .timer = LEDC_TIMER_3,
241     .frequency = 40,
242     .bit_resolution = LEDC_TIMER_12_BIT,
```

```
    .mode = LEDC_LOW_SPEED_MODE
};

Ultrasonic ultrasonic;

// Start routine IR Sensor
#define START_IR_SENSOR_PIN 39

SimpleGPIO start_routine_sensor;

// End routine IR Sensor
#define IR_SENSOR_PIN 5

SimpleGPIO end_routine_sensor;

// Homing IR Sensor
#define HOMING_IR_SENSOR_PIN 1

SimpleGPIO homing_ir_sensor;

// Dispose Sensor
#define DISPOSE_IR_SENSOR_PIN 3  // GPIO 3 (RX0, safe if not using Serial for debugging)

SimpleGPIO dispose_sensor;

// Color Sensor
uint8_t color_sensor_pins[2] = {21, 22}; // SDA, SCL

TCS3475_RGB color_sensor;

// Communication
// IR para cuando acabe el proceso
// GPIO para el arduino en caso de que uno no jale

// ================================ OTHERS ================================

// Axel F - Buzzer Class
#define BUZZER_PIN 15
// Buzzer shares timer 3 low speed with ultrasonic, channel 1
uint8_t buzzer_channel = 1;
TimerConfig buzzer_timer_config =
{
```

```
    .timer = LEDC_TIMER_3,
    .frequency = 440,
    .bit_resolution = LEDC_TIMER_8_BIT,
    .mode = LEDC_LOW_SPEED_MODE
};

AxelF_CrazyFrog crazy_frog;  // Renamed from 'song' to avoid conflict


// ============================================================================
// SYSTEM VARIABLES & TIMING
// ============================================================================

// UART communication variables
char buffer[256];
char msg_buffer[256];
uint64_t message_length;
const char* txt_msg;
int len = 0;

// Time polling variables
uint64_t prev_time, current_time;
uint64_t dt_us = 100000; // 100ms = 10Hz update rate (matches working test code)

// Viscosity time polling variables
uint64_t visc_prev_time, visc_current_time;
uint64_t visc_dt_us = 5000000; // 5 s

// RGB LED colors
int r = 0, g = 0, b = 0;

// Automatic Control System Variables
float piston_target_height_cm = 0.0f;
int start_automation = 0;
float min_viscosity = 0.0f;
float max_viscosity = 0.0f;

// RPMs test
float speed = 0.0f, angle = 0.0f, RPM = 0.0f, error = 0.0f, reference = 0.0f;

#endif //_DEFINITIONS_H_
```

C.2. main.cpp

Listing 2: definitions.h C++ code

```
#include "definitions.h"

// ================================================================================
// UTILITY FUNCTIONS
// ================================================================================

// TODO IN FUTURE PROJECTS: Add error handling in classes and functions

// Initialize all hardware components
bool init_hardware()
{
    // Install global GPIO ISR service once for all interrupt-based sensors
    // (must be done before any gpio_isr_handler_add calls)
    gpio_install_isr_service(ESP_INTR_FLAG_IRAM);

    // Actuators
    spindle.setup(spindle_config);
    spindle.setupPID(gains, spindle_dt_s);

    stepper.setup(steppper_gpio_pins, stepper_pul_pin, stepper_pul_feedback_pin,
        stepper_pwm_channel,
                  &stepper_vel_config, stepper_duty_percentage, stepper_frequency,
                      pulses_per_rev_stepper, 0);

    stepper.setupPID(gains_stepper, spindle_dt_s);

    piston.setup(PistonPins, PistonModes);

    water_pump.setup(water_pump_pins, water_channels, &dcMotorTimerConfig);

    dryer_motor.setup(dryer_pins, dryer_channels, &dcMotorTimerConfig);

    RGB_led.setup(RGB_pins, RGB_channels, &RGB_config, 1);

    // Sensors
    ultrasonic.setup(ECHO_PIN, TRIG_PIN, ultrasonic_trig_channel,
        ultrasonic_timer_config);

```

```
36      start_routine_sensor.setup(START_IR_SENSOR_PIN, GPIO_MODE_INPUT);
37      end_routine_sensor.setup(IR_SENSOR_PIN, GPIO_MODE_OUTPUT);
38      homing_ir_sensor.setup(HOMING_IR_SENSOR_PIN, GPIO_MODE_INPUT);
39      dispose_sensor.setup(DISPOSE_IR_SENSOR_PIN, GPIO_MODE_OUTPUT);
40
41      color_sensor.setup(color_sensor_pins);
42      color_sensor.calibrate(TCS34725_INTEGRATIONTIME_101MS, TCS34725_GAIN_16X);
43
44      // Others
45      crazy_frog.setup(BUZZER_PIN, buzzer_channel, &buzzer_timer_config);
46
47      bt.begin("OscarRamo");
48
49      prev_time = esp_timer_get_time();
50      while (esp_timer_get_time() - prev_time < 5000000)
51      {
52          // Wait 5 seconds for hardware to stabilize
53      }
54
55      return true;
56  }
57
58  // Update all telemetry readings from sensors
59  // TODO: Add viscosity, IR sensor, color sensor readings, current for monitoring
60  bool update_telemetry()
61  {
62      piston_position = ultrasonic.getDistance();
63
64      stepper_angle = stepper.readAngle();
65      stepper_speed = stepper.readSpeed();
66
67      spindle_rpms = (spindle.getSpeed() / 360.0f) * 60.0f;
68
69      spindle_current = spindle.getCurrent();
70
71      color_sensor.getRawRGBC(&r_sensor_output, &g_sensor_output, &b_sensor_output,
            &c_sensor_output);
72
73      start_automation = start_routine_sensor.get();
74      return true;
75  }
76
```

```
77 // Handle UART RX/TX data with LabVIEW
78 bool RX_TX_data()
79 {
80     // Scale RGBC to 8-bit (0-255) for LabVIEW; omit C channel if not used
81     uint16_t max_rgbc = r_sensor_output;
82     if (g_sensor_output > max_rgbc) max_rgbc = g_sensor_output;
83     if (b_sensor_output > max_rgbc) max_rgbc = b_sensor_output;
84     if (c_sensor_output > max_rgbc) max_rgbc = c_sensor_output;
85     if (max_rgbc == 0) max_rgbc = 1; // avoid divide by zero
86     uint8_t r8 = (uint8_t)((r_sensor_output * 255u) / max_rgbc);
87     uint8_t g8 = (uint8_t)((g_sensor_output * 255u) / max_rgbc);
88     uint8_t b8 = (uint8_t)((b_sensor_output * 255u) / max_rgbc);
89
90     // Device -> LabVIEW: numeric fields first, then status text (CSV, one line)
91     // Format: piston,stepper,rpms,viscosity,r8,g8,b8,msg
92     message_length = snprintf(
93         buffer,
94         sizeof(buffer),
95         "%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%u,%u,%u,%d\n",
96         piston_position,
97         stepper_angle,
98         stepper_speed,
99         spindle_rpms,
100        spindle_current,
101        viscosity_avrg,
102        static_cast<unsigned>(r8),
103        static_cast<unsigned>(g8),
104        static_cast<unsigned>(b8),
105        message_box_case
106    );
107    bt.write(buffer, message_length);
108
109    len = bt.available(); // Check for incoming data
110    if (len)
111    {
112        // Reset ALL input values before parsing new data
113        btn_piston_up = 0;
114        btn_piston_down = 0;
115        btn_stepper_cw = 0;
116        btn_stepper_ccw = 0;
117        knob_stepper_angle = 0.0f;
118        knob_stepper_frequency = 0.0f;
```

```
119         knob_spindle_pwm = 0.0f;
120         knob_spindle_rpm = 0.0f;
121         btn_measure_viscosity = 0;
122         btn_water_pump = 0;
123         btn_dryer_on = 0;
124 
125         bt.read(buffer, len);
126         buffer[len] = '\0';  // Null-terminate the buffer to prevent corruption
127         printf("UART received: %s\n", buffer);
128         // Format: LabVIEW inputs from top to bottom (CSV, one line)
129         sscanf(buffer, "%d,%d,%d,%d,%f,%f,%f,%f,%d,%d,%d,%d\n",
130                 &btn_piston_up,
131                 &btn_piston_down,
132                 &btn_stepper_cw,
133                 &btn_stepper_ccw,
134                 &knob_stepper_angle,
135                 &knob_stepper_frequency,
136                 &knob_spindle_pwm,
137                 &knob_spindle_rpm,
138                 &btn_measure_viscosity,
139                 &btn_water_pump,
140                 &btn_dryer_on,
141                 &state_machine_case);
142 
143         printf("Parsed: Pist_Up=%d, Pist_Dn=%d, Step_CW=%d, Step_CCW=%d, Angle=%.2f,
                Freq=%.2f, PWM=%.2f, RPM=%.2f, Visc=%d, Water=%d, Dryer=%d\n",
144             btn_piston_up, btn_piston_down, btn_stepper_cw, btn_stepper_ccw,
145             knob_stepper_angle, knob_stepper_frequency, knob_spindle_pwm,
                    knob_spindle_rpm,
146             btn_measure_viscosity, btn_water_pump, btn_dryer_on);
147 
148         if (state_machine_case == 0)
149         {
150             // Manual control mode
151             // Determine state based on current values (only when new data arrives)
152             if (btn_piston_up) manual_case_state = PISTON_UP;
153             else if (btn_piston_down) manual_case_state = PISTON_DOWN;
154             else if (btn_stepper_cw) manual_case_state = STEPPER_CW;
155             else if (btn_stepper_ccw) manual_case_state = STEPPER_CCW;
156             else if (knob_stepper_angle > 0.1f) manual_case_state = STEPPER_SPECIFIC;
                    // Added tolerance 0.1f
157             else if (knob_stepper_frequency > 0.1f) manual_case_state = STEPPER_FREQ;
```

```
                // Added tolerance 0.1f
158             else if (knob_spindle_pwm > 0.1f) manual_case_state = SPINDLE_PWM;  //
                    Added tolerance 0.1f
159             else if (knob_spindle_rpm > 0.1f) manual_case_state = SPINDLE_RPMs;  //
                    Added tolerance 0.1f
160             else if (btn_measure_viscosity) manual_case_state =
                    SPINDLE_MEASURE_VISCOSITY;
161             else if (btn_water_pump) manual_case_state = PUMP_WATER;
162             else if (btn_dryer_on) manual_case_state = DRYER_ON;
163             else manual_case_state = MANUAL_IDLE;  // Default: IDLE when nothing is
                    pressed
164         }
165         else
166         {
167             // Automatic control mode
168         }
169
170     }
171
172     return true;
173 }
174
175 // ================================================================================
176 // MANUAL CONTROL SYSTEM STATE MACHINE
177 // ================================================================================
178
179 // Control system state machine
180 bool manual_system_state_machine()
181 {
182     // If we are leaving viscosity mode, ensure related actions are stopped once
183     if (previous_state == SPINDLE_MEASURE_VISCOSITY && manual_case_state !=
            SPINDLE_MEASURE_VISCOSITY) {
184         spindle.setSpeed(0.0f);
185         spindle.setPWM(0.0f);
186         crazy_frog.stopSong();
187     }
188
189     switch(manual_case_state)
190     {
191         case MANUAL_IDLE:
192             // Do nothing
193             r = 0; g = 0; b = 0; // Off
```

```
194             RGB_led.setColor(r, g, b);
195             piston.stop();
196             stepper.stop();
197             spindle.setPWM(0.0f);
198             spindle.setSpeed(0.0f);
199             spindle.resetViscosityMeasurement();
200             water_pump.setStop();
201             dryer_motor.setStop();
202             crazy_frog.stopSong();
203             message_box_case = 1; // System idle
204             break;
205         case PISTON_UP:
206             r = 0; g = 255; b = 0; // Green
207             RGB_led.setColor(r, g, b);
208             piston.moveUp();
209             message_box_case = 2; // Piston moving up
210             break;
211         case PISTON_DOWN:
212             r = 0; g = 255; b = 0; // Green
213             RGB_led.blinkColor(r, g, b, 500);
214             piston.moveDown();
215             message_box_case = 3; // Piston moving down
216             break;
217         case STEPPER_CW:
218             r = 0; g = 0; b = 255; // Blue
219             RGB_led.setColor(r, g, b);
220             stepper.setCW();
221             message_box_case = 4; // Stepper moving clockwise
222             break;
223         case STEPPER_CCW:
224             r = 0; g = 0; b = 255; // Blue
225             RGB_led.blinkColor(r, g, b, 500);
226             stepper.setCCW();
227             message_box_case = 5; // Stepper moving counter-clockwise
228             break;
229         case STEPPER_SPECIFIC:
230             r = 0; g = 0; b = 255; // Blue
231             RGB_led.blinkColor(r, g, b, 200);
232             error = stepper.desiredAngle(knob_stepper_angle);
233             // When target reached (error = 0), transition to IDLE and clear knob value
234             if (error == 0.0f) {
235                 knob_stepper_angle = 0.0f;  // Clear the input to prevent re-triggering
```

```
236                 manual_case_state = MANUAL_IDLE;
237             }
238             crazy_frog.stopSong();
239             message_box_case = 6; // Stepper moving to specific angle
240             break;
241         case STEPPER_FREQ:
242             r = 0; g = 0; b = 255; // Blue
243             RGB_led.blinkColor(r, g, b, 100);
244             stepper.setFrequency(knob_stepper_frequency);
245             message_box_case = 7; // Stepper frequency set
246             break;
247         case SPINDLE_PWM:
248             r = 255; g = 255; b = 0; // Yellow
249             RGB_led.setColor(r, g, b);
250             spindle.setPWM(knob_spindle_pwm);
251             message_box_case = 8; // Spindle PWM set
252             break;
253         case SPINDLE_RPMs:
254             r = 255; g = 255; b = 0; // Yellow
255             RGB_led.blinkColor(r, g, b, 500);
256             spindle.setSpeed(knob_spindle_rpm);
257             message_box_case = 9; // Spindle RPMs set
258             break;
259         case SPINDLE_MEASURE_VISCOSITY:
260             // Reset song on first entry into this state
261             if (previous_state != SPINDLE_MEASURE_VISCOSITY) {
262                 crazy_frog.stopSong();
263             }
264             r = 255; g = 255; b = 0; // Yellow
265             RGB_led.blinkColor(r, g, b, 200);
266             spindle.setSpeed(30.0f); // Maintain constant speed for viscosity
                    measurement
267             viscosity_avrg = spindle.measure_viscosity_readings();
268             spindle_rpms = (spindle.getSpeed() / 360.0f) * 60.0f; // Update RPM display
269             crazy_frog.playSong();
270             message_box_case = 10; // Measuring viscosity
271             break;
272         case PUMP_WATER:
273             r = 255; g = 0; b = 255; // Magenta
274             RGB_led.setColor(r, g, b);
275             water_pump.setDuty(100.0f); // Full duty cycle
276             message_box_case = 11; // Water pump ON
```

```
277             break;
278         case DRYER_ON:
279             r = 255; g = 165; b = 0; // Orange
280             RGB_led.setColor(r, g, b);
281             dryer_motor.setDuty(100.0f); // Full duty cycle
282             message_box_case = 12; // Dryer ON
283             break;
284     }
285 
286     previous_state = manual_case_state; // Remember current state for next iteration
287     return true;
288 }
289 
290 // ================================================================================
291 // AUTOMATIC CONTROL SYSTEM STATE MACHINE
292 // ================================================================================
293 bool move_piston_to_height(float target_height_mm)
294 {
295     piston_position = ultrasonic.getDistance();
296 
297     error = target_height_mm - piston_position;
298 
299     if (fabs(error) <= 5.0f)  // 5mm tolerance
300     {
301         piston.stop();
302         return true; // Target reached
303     }
304     else if (error > 5.0f) piston.moveUp();
305     else if (error < -5.0f) piston.moveDown();
306 
307     return false; // Still moving to target
308 }
309 
310 bool stepper_homing()
311 {
312     stepper.setFrequency(200.0f);
313     homing_ir_sensor.set(1);
314     if (homing_ir_sensor.get() == 1)
315     {
316         stepper.stop();
317         stepper.homePosition();
318         return true; // Homing complete
```

```
319     }
320     else
321     {
322         stepper.setCW();
323     }
324 
325     return false; // Still homing
326 }
327 
328 bool water_dossing(int iteration, float dose_percentage)
329 {
330     // Iterative water dosing with configurable iteration and dose percentage
331     // iteration: 1, 2, or 3 (which iteration we're on)
332     // dose_percentage: percentage of total water to add (e.g., 85.0 for iteration 1,
        5.0 for iterations 2 & 3)
333 
334     float target_viscosity = (min_viscosity + max_viscosity) / 2.0f;  // Target
        midpoint of range
335 
336     // Exponential model coefficients (from SpindleVisc calibration)
337     float A = 1975.6f;   // Coefficient A (viscosity at 0% water)
338     float B = -0.071f;   // Coefficient B (exponential decay rate)
339 
340     // Calculate initial water percentage needed
341     float required_water_percentage = logf(target_viscosity / A) / B;
342 
343     // Clamp to valid range (0-25% water as per calibration limits)
344     if (required_water_percentage > max_water_percentage) {
345         required_water_percentage = max_water_percentage;
346     }
347     if (required_water_percentage < min_water_percentage) {
348         required_water_percentage = min_water_percentage;
349     }
350 
351     // Initial soap volume
352     float initial_soap_volume_mL = 100.0f;  // Initial volume of pure soap (0% water)
353 
354     // Calculate total water volume needed for target
355     float total_water_needed_mL = initial_soap_volume_mL * (required_water_percentage /
        (100.0f - required_water_percentage));
356 
357     // Calculate water to add this iteration
```

```
358     float iteration_water_mL = total_water_needed_mL * (dose_percentage / 100.0f);
359
360     // Water pump flow rate (calibrate this experimentally!)
361     float pump_flow_rate_mL_per_s = 10.0f;  // Example: 10 mL/s at 100% duty cycle
362
363     // Calculate pump run time
364     float pump_time_s = iteration_water_mL / pump_flow_rate_mL_per_s;
365
366     printf("\n=== ITERATION %d: Adding %.1f%% (%.2f mL, %.1f s) ===\n",
367            iteration, dose_percentage, iteration_water_mL, pump_time_s);
368     printf("Target: %.1f cPs (Range: %.1f - %.1f cPs)\n", target_viscosity,
           min_viscosity, max_viscosity);
369
370     // Dispense water
371     water_pump.setDuty(100.0f);
372     uint64_t start_time = esp_timer_get_time();
373     while (esp_timer_get_time() - start_time < (uint64_t)(pump_time_s * 1e6)) {
374         message_box_case = 12 + iteration; // 13=iter1, 14=iter2, 15=iter3
375     }
376     water_pump.setStop();
377
378     // Stir and measure viscosity
379     printf("Stirring and measuring viscosity...\n");
380     spindle.setSpeed(30.0f);  // Stir at constant speed
381     vTaskDelay(pdMS_TO_TICKS(5000));  // Stir for 5 seconds
382     viscosity_avrg = spindle.measure_viscosity_readings();
383     spindle.setSpeed(0.0f);  // Stop stirring
384
385     // Calculate error
386     float error_percent = ((viscosity_avrg - target_viscosity) / target_viscosity) *
           100.0f;
387     printf("Measured: %.1f cPs | Target: %.1f cPs | Error: %.1f%%\n",
388            viscosity_avrg, target_viscosity, error_percent);
389
390     // Check quality control criteria
391     // Pass: error within  10 % AND no overshoot (error >= 0)
392     if (fabs(error_percent) <= 10.0f && error_percent >= 0.0f) {
393         printf("    SUCCESS - Within tolerance! Directing to destination.\n");
394         message_box_case = 16; // Quality pass
395         return true;
396     }
397
```

```
398     // Check if overshooting (too much water)
399     if (error_percent < 0.0f) {
400         printf("    OVERSHOOT - Too much water added. Directing to reject
                warehouse.\n");
401         message_box_case = 17; // Quality reject - overshoot
402         return false;
403     }
404 
405     // Still not enough water - check if we can continue
406     if (iteration >= 3) {
407         printf("    FAILURE - Target not reached after 3 iterations. Directing to
                reject warehouse.\n");
408         message_box_case = 17; // Quality reject - insufficient iterations
409         return false;
410     }
411 
412     printf("    Proceeding to iteration %d...\n", iteration + 1);
413     return true;  // Continue to next iteration
414 }
415 
416 bool automatic_system_state_machine()
417 {
418     switch(automatic_case_state)
419     {
420         case HOMING_ROUTINE:
421             r = 255; g = 255; b = 0; // Yellow
422             RGB_led.blinkColor(r, g, b, 500);
423             move_piston_to_height(300.0f);  // 30cm = 300mm
424             stepper_homing();
425             automatic_case_state = AUTOMATIC_IDLE;
426             end_routine_sensor.set(0); // Deactivate end routine sensor
427             dispose_sensor.set(0); // Deactivate dispose sensor
428             break;
429 
430         case AUTOMATIC_IDLE:
431             r = 0; g = 0; b = 0; // Off
432             RGB_led.setColor(r, g, b);
433             piston.stop();
434             stepper.stop();
435             spindle.setPWM(0.0f);
436             spindle.setSpeed(0.0f);
437             spindle.resetViscosityMeasurement();
```

```
438             water_pump.setStop();
439             dryer_motor.setStop();
440             crazy_frog.stopSong();
441             message_box_case = 1; // System idle
442             break;
443         case CONTAINER_POSITIONING:
444             if (start_automation == 1)
445             {
446                 r = 0; g = 255; b = 0; // Green
447                 RGB_led.setColor(r, g, b);
448                 prev_time = esp_timer_get_time();
449                 while (esp_timer_get_time() - prev_time < 5000000)
450                 {
451                     message_box_case = 2; // Container positioned
452                 }
453                 automatic_case_state = DETECT_TAG_COLOR;
454             }
455             break;
456 
457         case DETECT_TAG_COLOR:
458             color_sensor.getRawRGBC(&r_sensor_output, &g_sensor_output,
                    &b_sensor_output, &c_sensor_output);
459             if (color_sensor.getColorDetected() == Color::COLOR_RED)
460             {
461                 message_box_case = 3; // Low Viscosity
462                 min_viscosity = 0.0f;
463                 max_viscosity = 800.0f;
464             }
465             else if (color_sensor.getColorDetected() == Color::COLOR_GREEN)
466             {
467                 message_box_case = 4; // Medium Viscosity
468                 min_viscosity = 800.0f;
469                 max_viscosity = 1400.0f;
470             }
471             else if (color_sensor.getColorDetected() == Color::COLOR_BLUE)
472             {
473                 message_box_case = 5; // High Viscosity
474                 min_viscosity = 1400.0f;
475                 max_viscosity = 2000.0f;
476             }
477             else
478             {
```

```
479                 message_box_case = 6; // No dominant color detected
480             }
481             automatic_case_state = MEASURE_VISCOSITY;
482             break;
483 
484         case MEASURE_VISCOSITY:
485             move_piston_to_height(210.0f);  // 21cm = 210mm
486             // Measure initial viscosity before dosing
487             message_box_case = 8; // Measuring viscosity
488             spindle.setSpeed(30.0f);
489             prev_time = esp_timer_get_time();
490             while (esp_timer_get_time() - prev_time < 30000000) // Stir
491             {
492                 // Wait for stirring to complete
493             }
494             viscosity_avrg = spindle.measure_viscosity_readings();
495             spindle.setSpeed(0.0f);
496             printf("Initial viscosity: %.1f cPs\n", viscosity_avrg);
497             automatic_case_state = DOSSING_ADJUSTMENT;
498             break;
499 
500         case DOSSING_ADJUSTMENT:
501             {
502                 bool quality_pass = false;
503                 float target_visc = (min_viscosity + max_viscosity) / 2.0f;
504 
505                 // Iteration 1: 85%
506                 water_dossing(1, 85.0f);
507                 float error1 = ((viscosity_avrg - target_visc) / target_visc) * 100.0f;
508                 if (fabs(error1) <= 10.0f && error1 >= 0.0f) {
509                     quality_pass = true;
510                     message_box_case = 9; // Viscosity within range
511                 } else if (error1 < 0.0f) {
512                     // Overshoot - reject
513                     message_box_case = 10; // Viscosity out of range, discarding
514                     move_piston_to_height(300.0f);  // 30cm = 300mm
515                     dispose_sensor.set(1);
516                     automatic_case_state = AUTOMATIC_IDLE;
517                     break;
518                 } else {
519                     // Iteration 2: 5%
520                     water_dossing(2, 5.0f);
```

```
521                 float error2 = ((viscosity_avrg - target_visc) / target_visc) *
                        100.0f;
522                 if (fabs(error2) <= 10.0f && error2 >= 0.0f) {
523                     quality_pass = true;
524                     message_box_case = 9; // Viscosity within range
525                 } else if (error2 < 0.0f) {
526                     // Overshoot - reject
527                     message_box_case = 10; // Viscosity out of range, discarding
528                     move_piston_to_height(300.0f);  // 30cm = 300mm
529                     dispose_sensor.set(1);
530                     automatic_case_state = AUTOMATIC_IDLE;
531                     break;
532                 } else {
533                     // Iteration 3: 5%
534                     water_dossing(3, 5.0f);
535                     float error3 = ((viscosity_avrg - target_visc) / target_visc) *
                            100.0f;
536                     if (fabs(error3) <= 10.0f && error3 >= 0.0f) {
537                         quality_pass = true;
538                         message_box_case = 9; // Viscosity within range
539                     } else {
540                         message_box_case = 10; // Viscosity out of range, discarding
541                     }
542                 }
543             }
544 
545             if (quality_pass) {
546                 automatic_case_state = CLEANING_STATION;
547             } else {
548                 printf("REJECT: Quality control failed\n");
549                 move_piston_to_height(300.0f);  // 30cm = 300mm
550                 dispose_sensor.set(1);
551                 automatic_case_state = AUTOMATIC_IDLE;
552             }
553         }
554         break;
555 
556     case CLEANING_STATION:
557         stepper.desiredAngle(120.0f); // Move to cleaning station
558         move_piston_to_height(210.0f);  // 21cm = 210mm
559         prev_time = esp_timer_get_time();
560         while (esp_timer_get_time() - prev_time < 60000000) // 60 seconds
```

```
            {
                message_box_case = 11; // Cleaning station active
                spindle.setPWM(0.0f);
            }
            move_piston_to_height(300.0f);  // 30cm = 300mm
            automatic_case_state = DRYING_STATION;
            break;

        case DRYING_STATION:
            stepper.desiredAngle(240.0f); // Move to drying station
            move_piston_to_height(210.0f);  // 21cm = 210mm
            dryer_motor.setDuty(100.0f); // Activate dryer
            prev_time = esp_timer_get_time();
            while (esp_timer_get_time() - prev_time < 60000000) // 60 seconds
            {
                message_box_case = 12; // Drying station active
                spindle.setSpeed(30.0f);
            }
            dryer_motor.setStop();
            move_piston_to_height(300.0f);  // 30cm = 300mm
            automatic_case_state = AUTOMATIC_IDLE;
            start_automation = 0; // Reset automation start flag
            break;
    }
    return true;
}

// ================================================================================
// MAIN FUNCTION
// ================================================================================

extern "C" void app_main()
{
    esp_task_wdt_deinit(); // Disable the watchdog timer

    if (!init_hardware())
    {
        printf("ERROR: Hardware initialization failed!\n");
        return;
    }
    else
    {
```

```
603         printf("Hardware initialization successful!\n");
604     }
605 
606     prev_time = esp_timer_get_time();
607 
608     while (true)
609     {
610         current_time = esp_timer_get_time();
611         if (current_time - prev_time >= dt_us)
612         {
613             // Update all sensor telemetry readings
614                 // All function calls are commented out for isolation.
615                 // Update all sensor telemetry readings
616                 if (!update_telemetry())
617                 {
618                     printf("ERROR: Telemetry update failed!\n");
619                     return;
620                 }
621 
622                 if (state_machine_case == 0)
623                 {
624                     // Manual control mode
625                     if (!manual_system_state_machine())
626                     {
627                         printf("ERROR: Manual system state machine failed!\n");
628                         return;
629                     }
630                 }
631                 else
632                 {
633                     // Automatic control mode
634                     if (!automatic_system_state_machine())
635                     {
636                         printf("ERROR: Automatic system state machine failed!\n");
637                         return;
638                     }
639                 }
640 
641                 if (!RX_TX_data())
642                 {
643                     printf("ERROR: RX/TX data handling failed!\n");
644                     return;
```

```
645                 }
646
647             prev_time = current_time; // Update previous time
648
649         }
650     }
651 }
```

D. Sorting system FluidSim

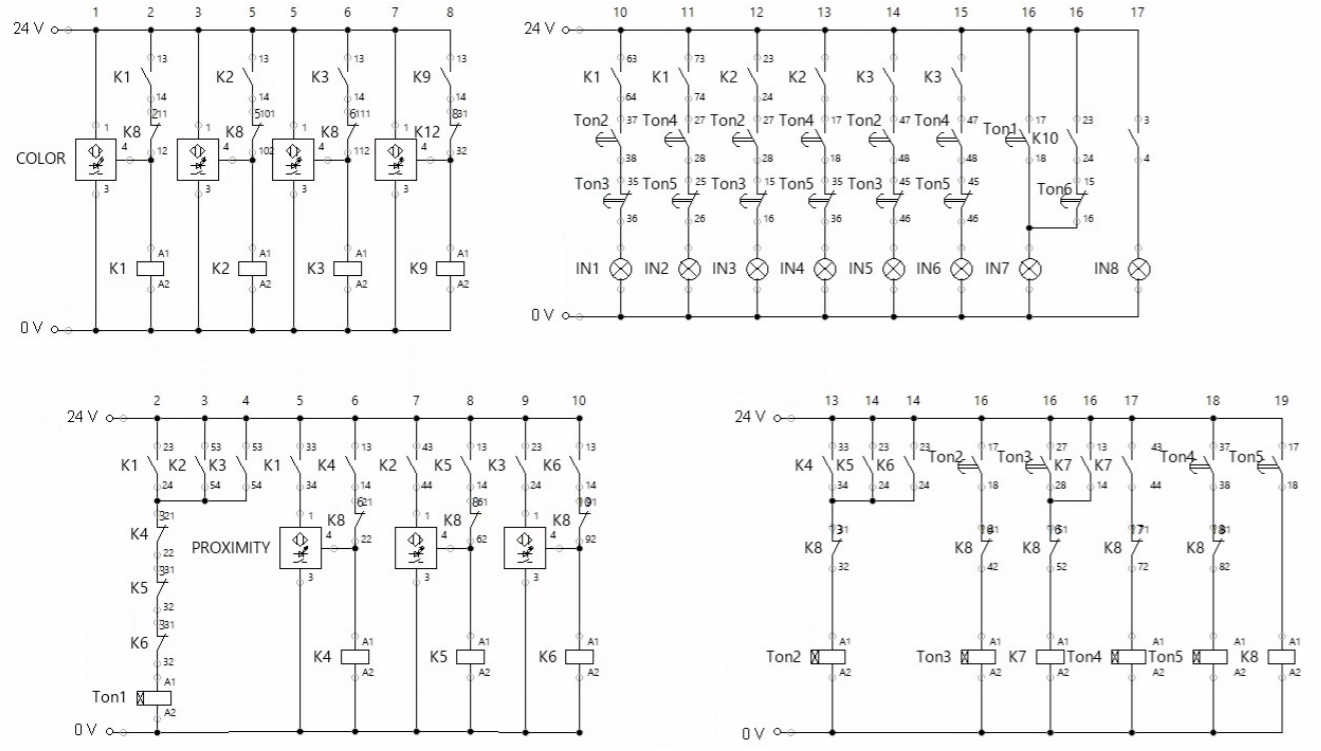


Figure 34: Sorting system Fluidsim pt.1

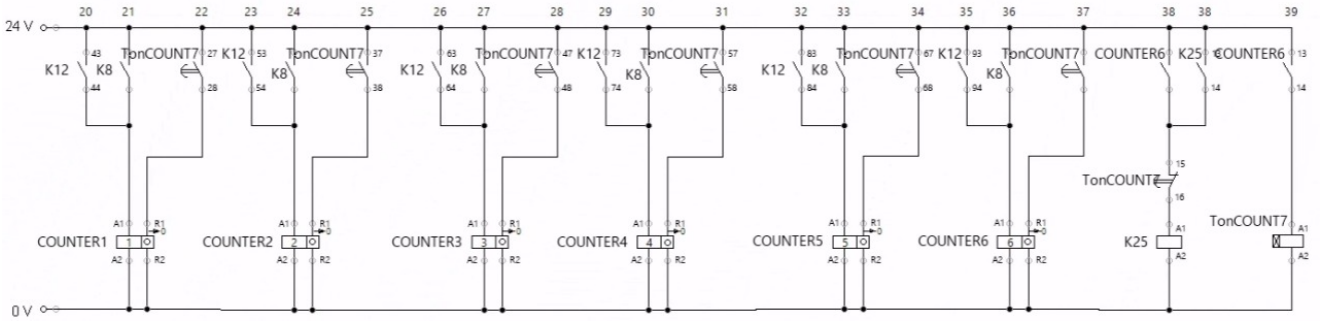


Figure 35: Sorting system Fluidsim pt.2

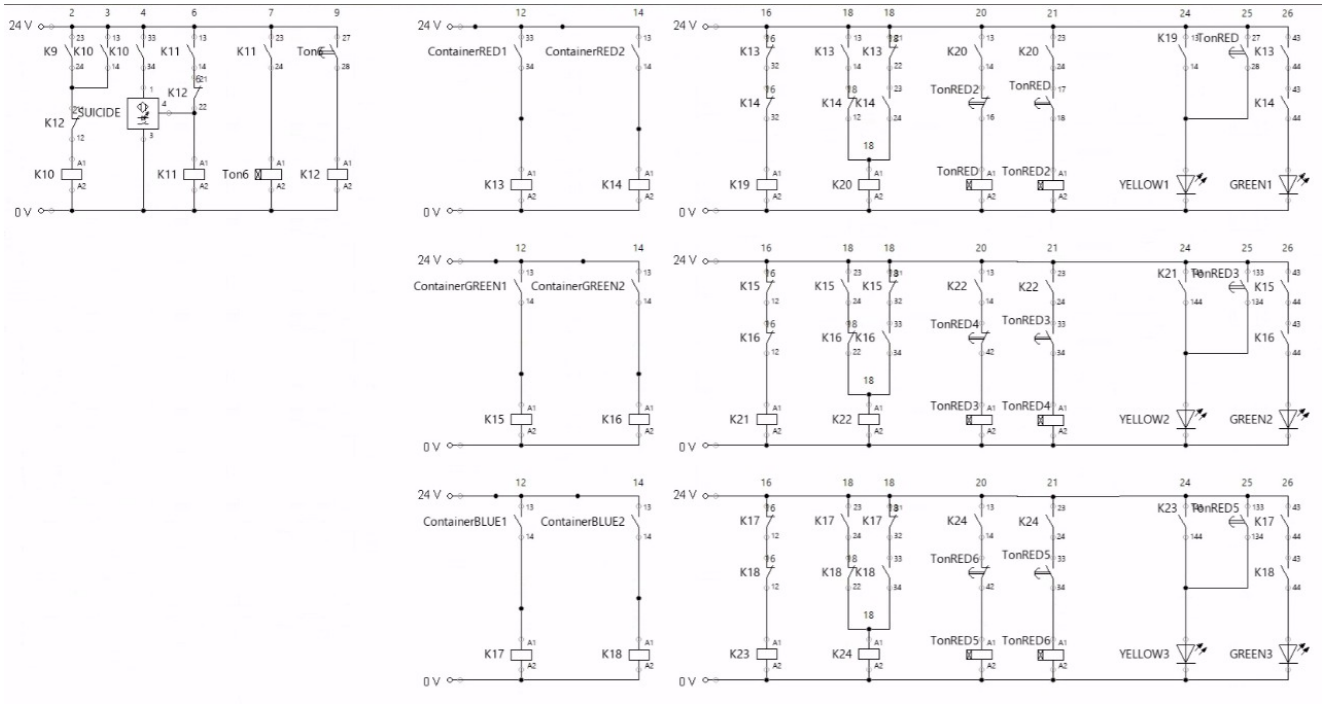


Figure 36: Sorting system Fluidsim pt.3

E. Matlab code for PID control

```
PWM = unnamed;
Degs = unnamed1;
Ts = 0.001;


% iddata(salida, entrada, Ts)
datos = iddata(Degs, PWM, Ts);


G_s = tfest(datos, 1, 0);

figure;
compare(datos, G_s);
title('Datos Reales vs. Modelo');
legend('Datos Reales', 'Modelo Estimado');

%Autotune PID
pidTuner(G_s);
```