DEPARTMENT OF ELECTRONIC &
TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA

End Effector: Design Report

Name	Index Number
Muftee M. M. M.	220399B
Goonetilleke P.	220183H
Jayakody J. A. K.	220247J
Rajinthan R.	220502M
Warushavithana N. D.	220678F
Praveen V.V.J.	220491B
V. G. V. Gunasekara	220193M

Submitted in partial fulfillment of the requirements for the module
EN 2160 Electronic Design Realization

Date: 10th of July 2025

Contents

1 Abstract

This project presents the design and implementation of a flexible robotic depalletizing system optimized for use in supermarket warehouse logistics. The system combines advanced sensing techniques with pneumatic actuation to automate the handling and unloading of boxes with varying sizes, weights, and surface textures. A key innovation is the integration of Time-of-Flight (ToF) sensors for precise box detection and angular alignment, enhancing the gripping accuracy of the End-of-Arm Tooling (EOAT). The EOAT features bellows-type vacuum suction cups powered by a pneumatic ejector system, enabling reliable lifting of uneven surfaces. A microcontroller-based firmware orchestrates the entire process, including sensor data acquisition, motor control, valve switching, and communication with the host system. The system emphasizes adaptability, real-time responsiveness, and efficient integration of mechanical and electronic subsystems to achieve high performance in automated box handling tasks.

2 System Overview

The depalletizing system is centered around a robotic arm equipped with a custom-designed end effector capable of picking up and moving boxes from pallets. The system consists of the following core components:

- **Vacuum Suction Cups:** Bellows-type suction cups made of NBR-ESD material are used to adapt to various box surfaces and ensure secure gripping.
- **Vacuum Generation:** A compact pneumatic vacuum ejector is used to generate the necessary suction force, utilizing the existing compressed air infrastructure.
- **ToF Sensors:** Time-of-Flight sensors are mounted on the EOAT to detect box presence and orientation. Two methods are used for alignment: a 64×64 pixel depth array sensor and a multi-sensor triangulation setup.
- **Microcontroller Unit:** An ATmega2560 microcontroller manages real-time control tasks including sensor communication, motor control, vacuum actuation, and data transmission.
- **Solenoid Valves:** Double solenoid 5/2 valves are used to direct airflow to pneumatic cylinders and the vacuum ejector, enabling controlled movement and suction.
- **Firmware Architecture:** The microcontroller firmware is responsible for real-time task execution, including distance filtering, alignment calculations, motor actuation, vacuum control, and communication with the PC.

The integration of mechanical and electronic subsystems results in a robust and adaptable robotic platform capable of handling depalletizing tasks with high reliability and precision, even in unstructured environments.

3 Review of Existing Products for Reconfigurable End Effector

3.1 MCS-RP Gripper Series by WSR Solutions

The MCS-RP Gripper Series by WSR Solutions offers a versatile and modular approach to robotic gripping applications. These grippers are designed to handle a wide range of part geometries and sizes, making them suitable for various industrial automation tasks.

3.1.1 Key Features

- **Modular Design:** The gripper series features a modular construction, allowing for easy customization and adaptation to different application requirements.
- **High Grip Force:** The MCS-RP grippers provide a high grip force relative to their size, ensuring secure handling of parts during automation processes.

- **Precision and Repeatability:** These grippers offer high precision and repeatability, essential for tasks requiring consistent performance.
- **Versatility:** Capable of handling various part shapes and sizes, the MCS-RP grippers are suitable for diverse applications across different industries.

3.1.2 Applications

The MCS-RP Gripper Series is utilized in various industries for tasks such as:

- **Assembly Automation:** Facilitating the assembly of components in manufacturing lines.
- **Material Handling:** Efficiently moving parts within production facilities.
- **Quality Control:** Assisting in the inspection and testing of products.

3.1.3 Advantages

- **Enhanced Flexibility:** The modular design allows for quick adjustments and reconfigurations to meet changing production needs.
- **Improved Efficiency:** High grip force and precision contribute to faster and more reliable automation processes.
- **Cost-Effectiveness:** The adaptability of the grippers reduces the need for multiple specialized tools, leading to cost savings.

Figure 1: MCS-RP Gripper Series by WSR Solutions

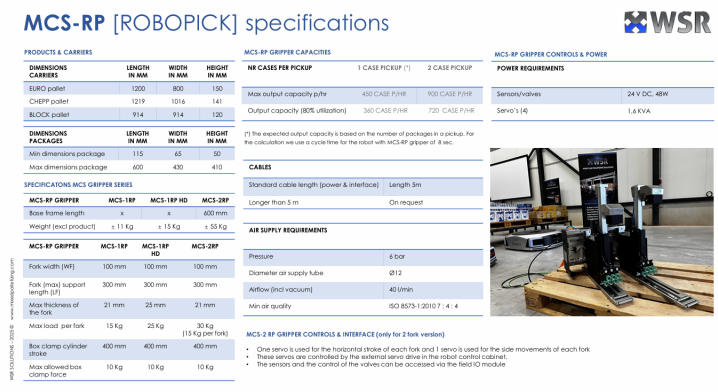

Figure 2: MCS-RP Gripper Series Specification

4 Stakeholder Mapping

Stakeholder mapping was carried out to identify and prioritize individuals and groups involved in the development of the reconfigurable end effector. Based on their level of influence and interest, stakeholders were categorized into four groups:

- **Manage Closely:** High influence and high interest
  Examples: Robotic engineers, warehouse operators, end users
- **Keep Satisfied:** High influence and low interest
  Examples: Regulatory authorities, procurement specialists
- **Keep Informed:** High interest and low influence
  Examples: Supermarket customers, equipment suppliers
- **Monitor:** Low influence and low interest
  Examples: Safety inspectors, academic researchers

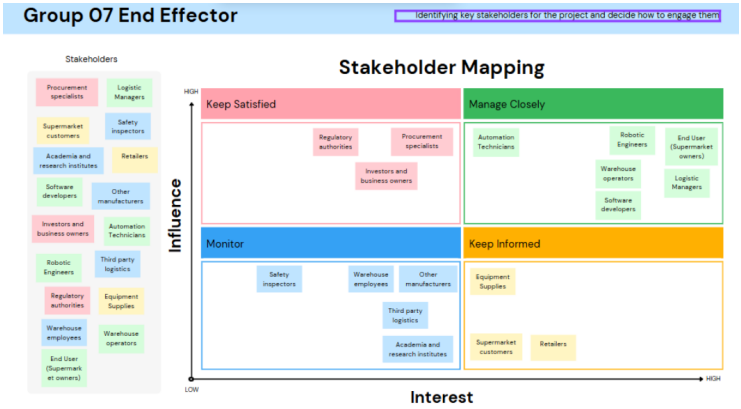

Figure 3: Stakeholder mapping

5 User Needs Analysis

An effective reconfigurable end effector for supermarket depalletizing must meet a range of user-centered requirements. These key user needs were identified based on common logistical challenges and operational demands:

- **Flexible Gripping Mechanism:** Ability to handle packages of various shapes (15–50 cm), weights (0–10 kg), and materials.
- **Dual-Sided Grasping:** Enables lifting from both top and side surfaces, which is crucial for handling stacked or irregularly shaped items.
- **Automatic Reconfiguration:** Utilizes actuated degrees of freedom to adapt the gripping strategy without requiring manual changes.
- **Safety & Stability:** Ensures safe handling of fragile or perforated packaging to avoid damage.
- **Seamless Integration with Robotic Arms:** Compatible with standard robotic systems for straightforward deployment.
- **Sensor-Driven Adaptability:** Employs Time-of-Flight (ToF) sensors and load cells for real-time object detection, weight estimation, and orientation correction.

- **Operational Efficiency:** Aims to reduce manual labor and time spent on depalletizing, improving overall logistics performance.
- **Durability & Low Maintenance:** Built to withstand frequent use in supermarkets with minimal maintenance needs.
- **Compact and Lightweight Design:** Ensures effectiveness in space-limited supermarket environments.
- **Cost-Effective Implementation:** Should be affordable and feasible for use in developing regions.

6 Conceptual Designs

6.1 Initial Sketches

The initial sketches served as a starting point for exploring different design concepts for the depalletizing system. Various ideas were considered, focusing on functionality, efficiency, and ease of integration. These sketches allowed us to visualize potential solutions and identify key features that would later be refined in the final design.

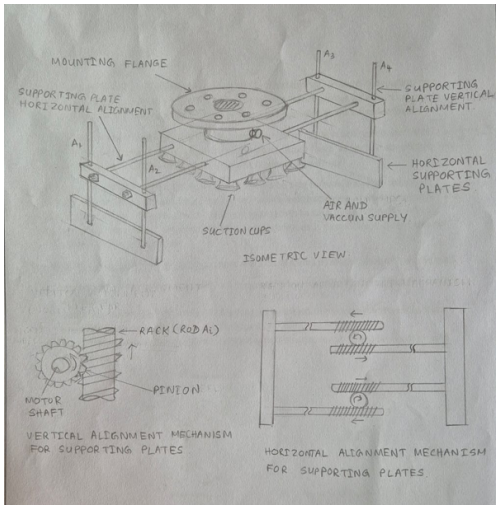

Figure 4: Conceptual Design 01

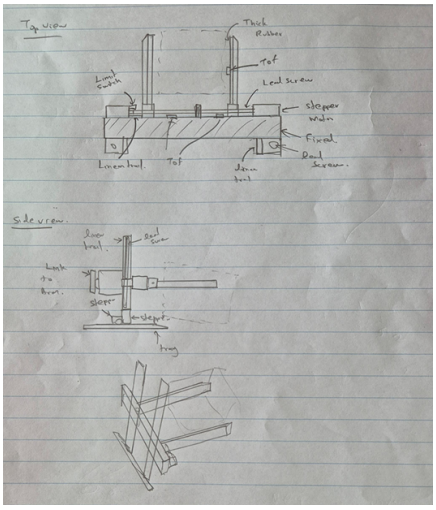

Figure 5: Conceptual Design 02

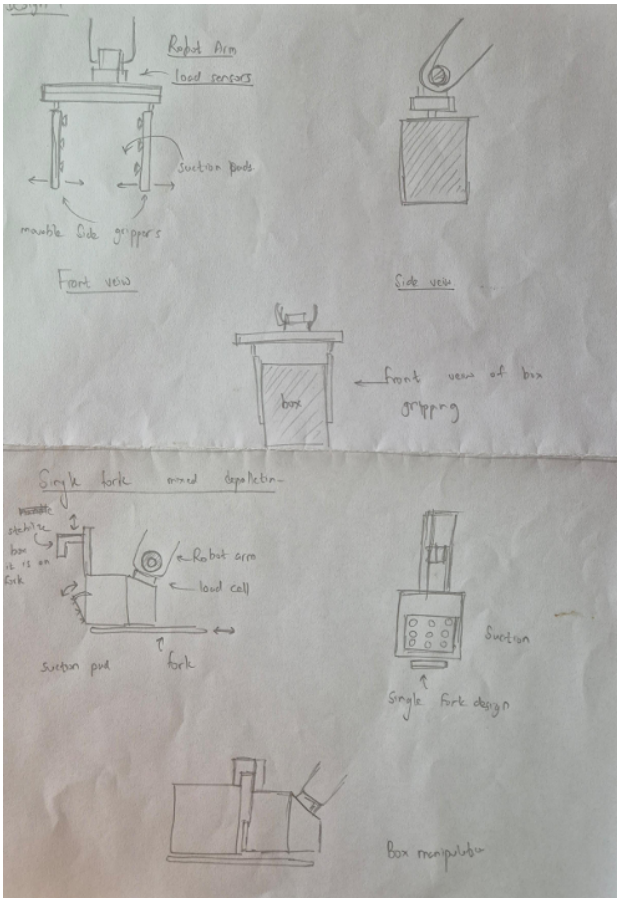

Figure 6: Conceptual Design 03

6.2 Refined Design

After evaluating and cross-pollinating the best features from several initial designs, we have created a refined conceptual sketch that incorporates the most effective elements of each. This design ensures optimal performance and reliability in depalletizing tasks.

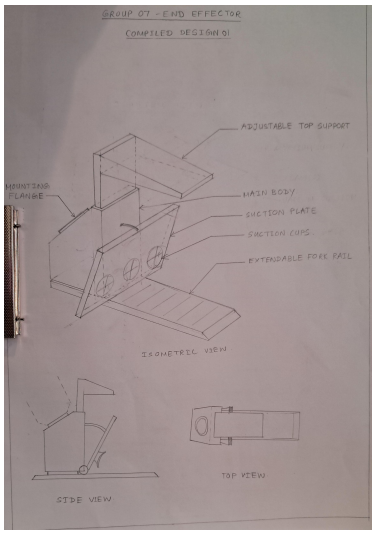

Figure 7: Final Conceptual Design

The final conceptual design features suction cups mounted vertically on a suction plate. The suction plate grips the box using the suction cups and is slightly lifted by a rodless cylinder attached to the structure. Forks are then slid under the box, and the suction plate is lowered to position the box onto the forks. To secure the box in place and prevent it from toppling, a top support is moved downward using a pneumatic cylinder, ensuring a stable grip throughout the process.

7 Functional Block Diagram

Initial Functional Block Diagram

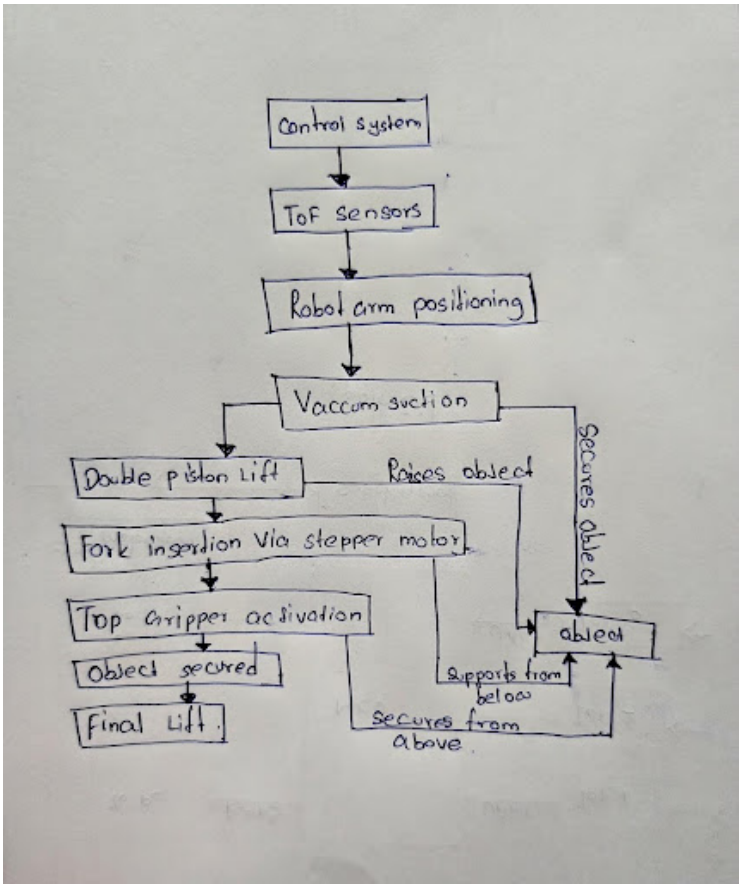

Figure 8: Functional block diagram of the End Effector

Final Block Diagram

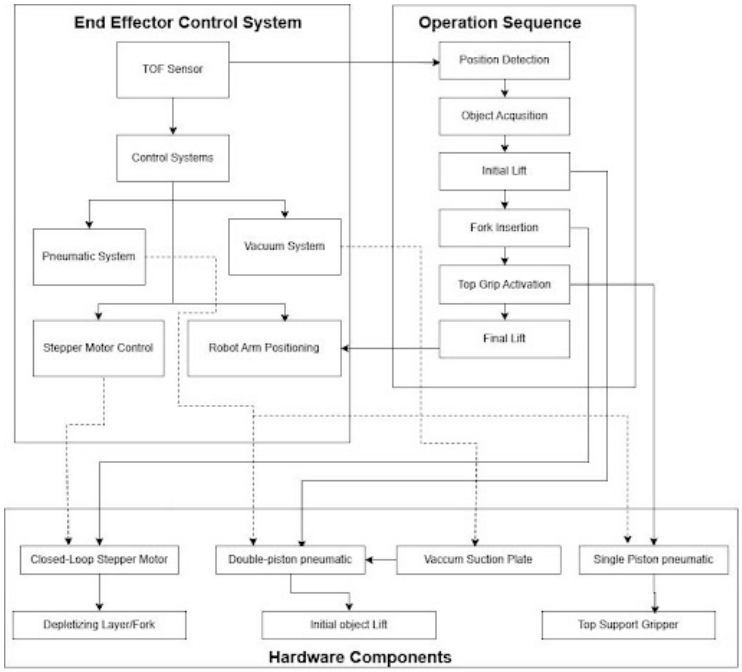

Figure 9: Functional block diagram of the End Effector

8 Component Selection

Each component in the end effector was chosen based on functionality, availability, and performance under industrial conditions. This section elaborates on the core modules.

8.1 Closed-Loop Stepper Motors and Motor Drivers

In the depalletizing robot arm end effector, a **closed-loop stepper motor** is used to insert and retract the fork mechanism. The motor is coupled to a pinion gear, which engages with a linear rack. As the motor rotates the pinion, it converts the rotary motion into linear displacement, thereby pushing or pulling the fork.

This mechanism requires **high positional accuracy** and **consistent torque delivery**, particularly under variable loads when interacting with the box in different phases of the lifting sequence. Hence, a closed-loop stepper motor was chosen to ensure reliable performance without the risk of step loss under load.

Motor Selection: 42HSE47 Closed-Loop Stepper Motor

The selected motor is the **42HSE47 closed-loop stepper motor**. Its specifications match the torque and speed demands calculated from the rack and pinion setup.

Figure 10: 42HSE47 Closed-Loop Stepper Motor

Key Specifications

- **Step Angle**: 1.8°
- **Holding Torque**: 0.6
- **Rated Current**: 2.5
- **Encoder Resolution**: 1000 lines
- **Weight**: 0.7
- **Dimensions**: 42 mm frame size (NEMA 17)

The maximum torque requirement occurs during the acceleration and deceleration phases. With a holding torque of 0.6, the **42HSE47** meets the expected requirements with a small margin, supporting safe and consistent operation.

Product details: `http://www.chinajingbo.com/en/product-detail/53ad564283dd4fe4a286167c54d5ea03#content`

Motor Driver Selection: HSE42 Closed-Loop Stepper Driver

The **HSE42 driver** is selected as it is designed to match the 42HSE47 motor and offers extensive control features.

Figure 11: HSE42 Closed-Loop Stepper Driver

Key Specifications

- **Input Voltage Range**: 24–80 VDC
- **Output Current**: 0–6.0 A
- **Pulse Frequency**: Up to 200 kHz
- **Microstepping**: 200 to 51200 steps/rev
- **Feedback Input**: Encoder A+/A, B+/B
- **Feedback Control Modes**: Position, Velocity
- **Protections**: Overcurrent, Overvoltage, Overheat, Tracking error

Features

- Built-in position and speed control mode
- Encoder feedback for real-time position correction
- Electronic gear ratio configuration for flexible control
- Digital display and buttons for easy tuning and diagnostics

Product details: `http://www.chinajingbo.com/en/product-detail/2ebcdfe755f244b8b62cabf932c3272a`

8.2 Pneumatic Cylinders

Pneumatic cylinders are integrated into two key areas of the End-of-Arm Tooling (EOAT) system of the depalletizing robot arm. These components play an essential role in both gripping and stabilizing operations during box manipulation.

First, the pneumatic cylinders are used to lift the vacuum gripper plate when they are gripping the boxes by the vacuum grippers in the initial stages of the box holding.

The second instance the pneumatic cylinders are used is to stabilize the box on the arm and avoid the boxes being toppled over the fork during the box moving phase.

1. Vacuum Gripper Plate Lifting Mechanism – TN10X50

In the initial phase of gripping, pneumatic cylinders are used to lift the vacuum gripper plate after the vacuum grippers have engaged with the box surfaces. This movement ensures secure lifting and reduces any shear force acting on the suction cups during detachment.

After evaluating multiple cylinder types—such as single/double rod cylinders, linear actuators, and rodless cylinders—the Dual Rod Double Acting Cylinder TN10X50 was selected. This choice was guided by considerations of size constraints, required degrees of freedom, load requirements, and mounting compatibility.

Selected Cylinder: Airtac TN10X50

- **Bore size**: 10 mm
- **Stroke**: 100 mm
- **Acting**: double acting
- **Operating Pressure**: 0.1 - 10 MPa
- **Mounting**: Mounting Holes
- **Cushioning**: Bumper

- **Rod end**: mounting holes

Figure 12: Airtac TN10X50 Pneumatic cylinder

The dual-rod configuration provides enhanced stability and reduces piston deflection during actuation, which is crucial for the vertical lifting motion of the gripper plate.

Data sheet: `https://airtacmalaysia.com/wp-content/uploads/media_uploads/TN-series.pdf`

2. Box Stabilizing Mechanism – MAL16X100-S

The second application of pneumatic actuation is in the box stabilization mechanism. During transport, especially in dynamic motion phases, there is a risk of the box toppling forward or backward on the fork. A pneumatic stabilizer ensures lateral constraint to prevent such occurrences.

While performance analysis indicated that a rodless cylinder would offer an optimal motion profile and compactness, budgetary constraints necessitated a more cost-effective solution. The chosen alternative was the MAL16X100-S Double Acting Cylinder, which balances functionality and affordability.

Selected Cylinder: Airtac MAL16X100-S

Figure 13: Airtac MAL16X100-S Pneumatic cylinder

- **Bore size**: 16 mm
- **Stroke**: 100 mm
- **Acting**: double acting
- **Operating Pressure**: 0.1 - 10 MPa

- **Mounting**: Rear clevis for pivoting + male mounting thread
- **Cushioning**: Bumper
- **Rod end**: Male thread for connection

Data sheet: `https://airtacmalaysia.com/wp-content/uploads/media_uploads/MAL-series.pdf`

8.3 Vacuum Suction Cups

In the vacuum system, there are a few main components: vacuum suction cups, which act as the interface between the workpiece and the vacuum system; vacuum generator; switching and valves; mounting elements and connectors.

Vacuum suction cups are attached to the end effector to grip the workpiece while the initial lifting is handled. To securely grip the workpiece, depalletizing boxes with varying surfaces and textures, we selected a bellows-type suction cup, which is optimized for adaptability and reliability. The **FSG 20 NBE-ESD-55 M5-AG** suction cup by Schmalz was chosen due to its ability to handle uneven surfaces, making it ideal for the application of depalletizing in supermarket warehouses.

> "Bellows suction cups with a round shape are particularly suited for handling uneven and arched workpieces. Bellows suction cups are used if you need to handle workpieces with different heights, uneven surfaces or even fragile parts. Bellows suction cups of these series cover a wide range of applications. Their optimized design and the fact that they are available in different diameters and materials makes flat suction cups of these series perfect all-rounders." - Schmalz

Figure 14: Vacuum suction cups for object gripping

- **Size**: 20 mm diameter
- **Material**: Nitrile rubber NBR-ESD with 55 Shore A hardness
- **Design**: 2.5 fold bellows with 9 mm stroke
- **Mounting**: M5 Male thread (M5-M) aluminium nipple
- **Suction force**: 5.20 N at -0.6 bar vacuum
- **Pull-off force**: 12.10 N
- **Hose diameter**: 2 mm (for hose length of 2 m)
- **Weight**: 3.3 g

This suction cup offers a compact and robust design that integrates easily into our end effector system, ensuring stable gripping during the complete robot arm movement.

Download datasheet: `https://www.schmalz.com/en/product/10.01.06.04345/download`

8.4 Vacuum Ejectors

The vacuum is generated either pneumatically by ejectors or electrically by pumps or blowers. For our design, we selected a vacuum ejector to utilize the pneumatic system that is already being used in pneumatic cylinders for vertical motion.

The requirements for the ejector are to work under 8 bar input pressure, provide -0.6 bar vacuum level, and sufficient output flow for the suction cups. **SCPS 07 M G02 NC M12-5 PNP** from Schmalz is best suited for the design. However, due to budget constraints, **AZH10BS-06-06** from Airbest is selected, which also satisfies all the requirements for the vacuum system.

Figure 15: AZH10BS-06-06 Vacuum ejector to create suction force

- **Nozzle diameter:** 1 mm
- **Shape:** Box type
- **Air supply port:** 6 mm
- **Vacuum port:** 6 mm
- **Rated air supply pressure:** 4.5 bar
- **Maximum vacuum level:** -88 kPa (High vacuum level)
- **Maximum vacuum flow:** 12 NL/min
- **Air consumption:** 23.5 NL/min
- **Noise level:** 68 dB
- **Working temperature:** 5–50 °C
- **Weight:** 28 g

Datasheet: AZH Series Vacuum Generator PDF

8.5 Solenoid Valves

Solenoid valves control the flow of compressed air to pneumatic components and the vacuum ejector. They are electromechanically operated using electrical signals to switch the position of internal spools and direct air to desired ports.

Solenoid valves are mainly of two types: **single solenoid** and **double solenoid**. A single solenoid has one coil and uses a spring to return the valve to its default position when power is removed; they are commonly used where a default is preferred and for quick actions. A double solenoid has two coils and remains in its last switched state when power is switched off until the other coil is energized. This is ideal for applications where both positions should be held for prolonged durations, thus being more power-efficient.

For our design, we selected a **5/2 double solenoid valve**—featuring five ports and two positions. It allows precise control of double-acting pneumatic cylinders, enabling both extension and retraction using separate signals without relying on spring return.

Figure 16: Solenoid valves for air path control

8.6 Fork

To achieve precise and smooth linear motion for the fork of our end effector, we selected a **linear rail guide combined with a rack-and-pinion mechanism**. The linear rail ensures **high rigidity and low-friction guidance**, which is essential for accurate alignment and stability when lifting or placing objects. Its compact form factor and load-bearing capability make it ideal for space-constrained robotic applications, especially in end effectors that require consistent repeatability.

We initially designed using a single linear guide rail on one end and the rack on the other end. While this design works, it can be less steady and may give rise to bending moments when an uneven distribution of weight is placed on the fork.

Therefore, we incorporated two rails on either end with the rack being set in the center. This configuration provides a balanced and steady foundation for the fork, improving structural integrity and smooth motion.

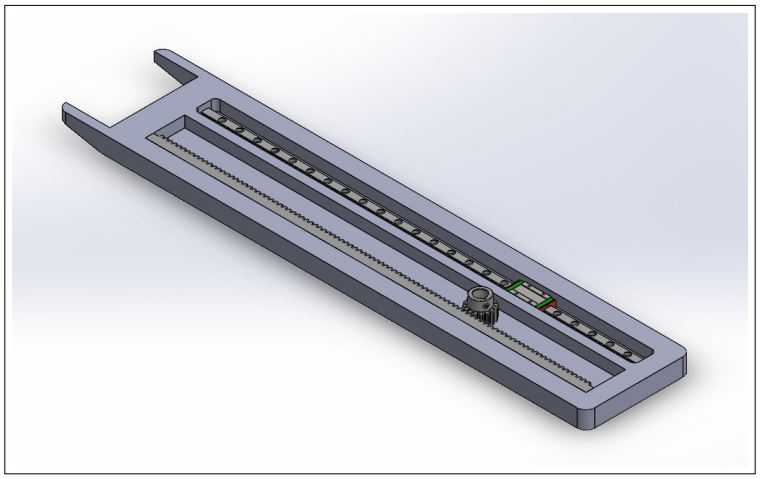

Figure 17: Fork-type linear actuator with a single rail

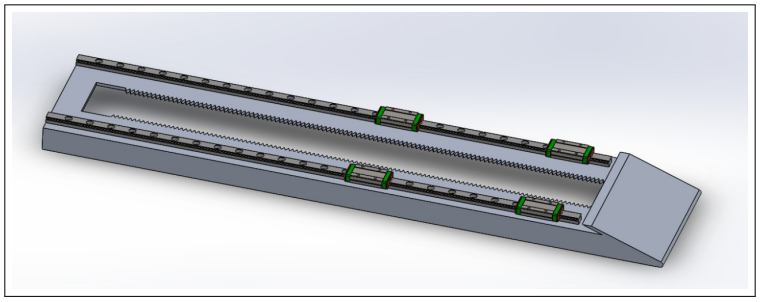

Figure 18: Fork-type linear actuator with two rails

8.7 Rack and Pinion Mechanism

The **rack-and-pinion system** complements the fork by converting rotational motion from a motor into linear movement of the fork. This setup offers **excellent mechanical advantage**, enabling quick response and reliable control over the fork's motion without slippage, which is common in belt-driven or friction-based systems. Additionally, gear-driven systems are **less sensitive to dust or wear**, increasing durability in industrial-like environments.

Moreover, this design simplifies maintenance and allows easy tuning of speed and force by adjusting gear ratios. This modularity and robustness make the linear rail and rack-and-pinion mechanism an optimal choice for the controlled and reliable actuation of the end effector's fork.

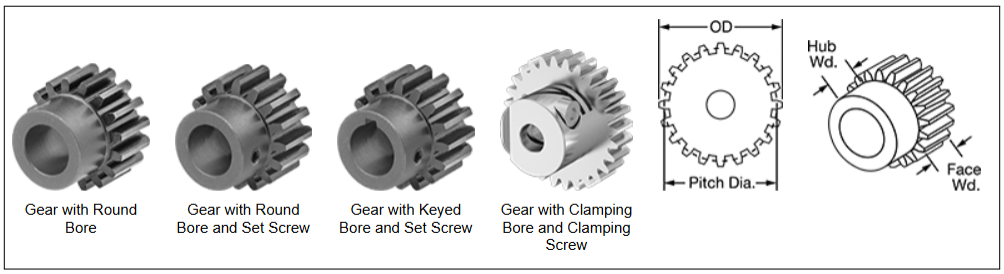

Figure 19: Metal gears used in rack-and-pinion mechanisms

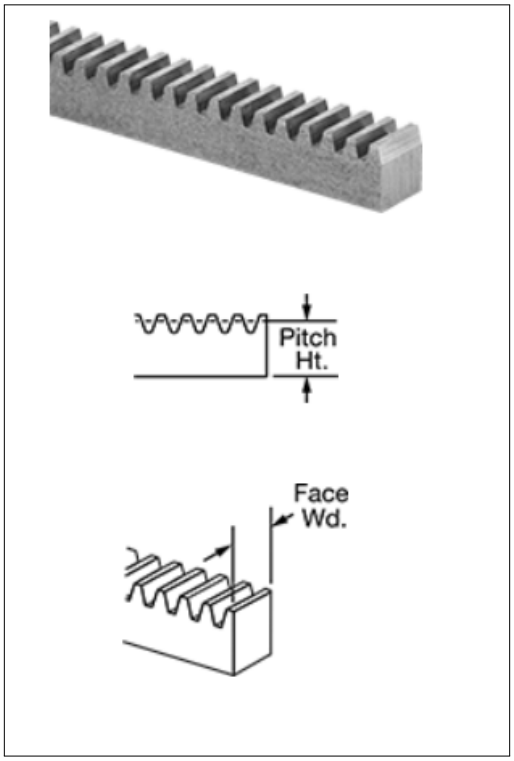

Figure 20: Metal racks corresponding to pinion gears

Gear and Rack Selection for Fork Mechanism

Below is the detailed analysis and justification for the component selection.

8.7.1 System Requirements

- **Payload:** 1 kg (500 g with 100% tolerance)
- **Stroke length:** 500 mm (0.5 m)
- **Mounting orientation:** Horizontal
- **Desired mechanism:** Rack and pinion with linear rail guide
- **Safety factor:** 1.5 (to account for friction and uncertainty)

8.7.2 Force Calculation

We assume a modest acceleration of $a = 0.5\,\mathrm{m/s^2}$ to overcome inertia. The force required to move the load is given by:

$$F = m \cdot a = 1 \cdot 0.5 = 0.5\,\mathrm{N}$$

Applying a safety factor of 1.5:

$$F_{\text{required}} = 0.5 \cdot 1.5 = 0.75\,\mathrm{N}$$

To be conservative, we round up and use $F = 1\,\mathrm{N}$ as the required linear force.

8.7.3 Torque Requirement at Pinion

Assuming a pinion with a pitch diameter of $D = 20\,\text{mm}$, the radius is:

$$r = \frac{D}{2} = 10\,\text{mm} = 0.01\,\text{m}$$

The torque required at the gear is:

$$\tau = F \cdot r = 1 \cdot 0.01 = 0.01\,\text{Nm} = 10\,\text{Ncm}$$

8.7.4 Rack and Pinion Geometry

We select a standard module $M = 1\,\text{mm}$. Choosing a pinion with 20 teeth:

$$\text{Pitch Diameter} = z \cdot M = 20 \cdot 1 = 20\,\text{mm}$$

$$\text{Linear Travel per Revolution} = \pi \cdot D = \pi \cdot 20 \approx 62.8\,\text{mm}$$

For a stroke length of 500 mm:

$$\text{Required Revolutions} = \frac{500}{62.8} \approx 7.96 \approx 8\,\text{turns}$$

8.7.5 Selected Components

- **Pinion Gear:**
  - Module: 1 mm
  - Teeth: 20
  - Pitch Diameter: 20 mm
  - Bore: To match motor shaft (keyed or clamping type recommended)
- **Rack:**
  - Module: 1 mm
  - Face width: $\geq$ 10 mm
  - Length: 500 mm

8.8 ToF Sensors

Time-of-Flight (ToF) sensors are used to detect the presence of boxes and to measure the angle relative to the box. This capability is essential to align the end effector perpendicular to the boxes, ensuring higher gripping accuracy. The sensor was chosen based on its short minimum range, high resolution, and support for well-documented communication protocols.

Based on these requirements, we selected the **TOFSense-UART** sensor by Noop-Loop, which offers the following features:

- Measuring scope: **1 cm − 5 m**
- Adjustable field of view (FOV): **15–27°**
- Resolution: **1 mm**
- Cascade: **8 UART**
- Power supply: **3.7–5.2 V** with anti-reverse protection
- Power consumption: **290 mW**

- Communication: **UART at 115200 baud**
- Weight: **2.7 g**

Figure 21: ToF sensor used for object detection and height measurement

8.9 Microcontroller Unit (MCU)

The microcontroller serves as the central processing unit in our system, managing data acquisition, processing, and communication with the external computer. It is optimized for reliable real-time data handling, ensuring continuous availability for analysis. Additionally, it requires multiple independent UART channels for reliable communication with ToF sensors.

We selected the **ATmega2560** for its robust performance and versatility in both industrial and consumer applications. With a 16 MHz clock speed, it offers sufficient processing power for handling sensor data and control logic. The microcontroller provides 256 KB of Flash memory for program storage and 8 KB of SRAM for data storage, supporting complex algorithms and data structures.

Key Features of ATmega2560

- **Performance and Connectivity:** Operates at 16 MHz with 256 KB of Flash memory and 8 KB of SRAM. Supports UART (4 independent channels), SPI, I^2C, and other communication interfaces for integration with various sensors and peripherals.
- **Built-in Peripherals:** Includes 86 programmable GPIO pins, enabling versatile configuration for interfacing with sensors, actuators, and other external devices. This flexibility is essential for tailoring applications to specific project requirements.
- **Development Support:** Fully supported by the Arduino IDE and a wide range of community libraries, which streamline development and prototyping. Arduino libraries provide ready-to-use functions for quick implementation.
- **Reliability and Cost-effectiveness:** Known for its reliability, ease of use, and cost-effectiveness, the ATmega2560 is ideal for applications requiring both stability and affordability.

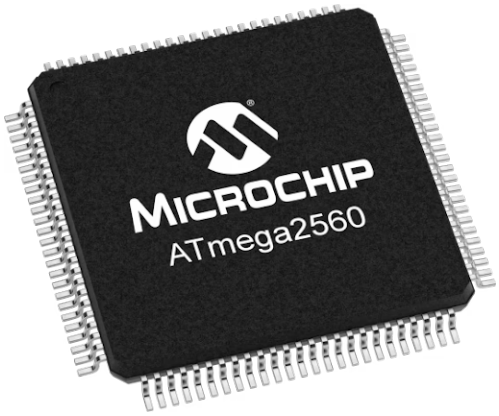

Figure 22: Microcontroller managing the system

In summary, the ATmega2560 microcontroller is a dependable choice for projects requiring reliable data processing, versatile connectivity, and robust performance in embedded systems. Its extensive community support and rich ecosystem of libraries contribute to accelerated development cycles and deployment of applications across various domains.

9 Sensor Integration

ToF Sensor-Based Alignment System for EOAT

To ensure accurate alignment of the End-of-Arm Tooling (EOAT) with the target box, Time-of-Flight (ToF) sensors are employed as the primary external environment sensing devices. These sensors enable the robot arm to detect the presence and orientation of the box before engaging with it, significantly improving the success rate of gripping and lifting operations.

ToF Sensor Functions

- **Presence Detection:** ToF sensors detect the existence of a box in the robot's field of operation.
- **Alignment Assistance:** They help ensure that the EOAT aligns correctly with the box face, minimizing positional errors during pickup.

Alignment Strategy

To accurately determine the orientation of the box surface, two methods are being explored:

Method 1: ToF Sensor with 64×64 Distance Array

This method involves the use of a ToF sensor module that outputs a 64×64 pixel distance array (e.g., a depth map). The depth map provides spatial distance information across the field of view.

Process Overview:

1. Extract the distance values from the 2D depth array.
2. Use edge detection or thresholding techniques to identify the 4 edges of the box.
3. Apply basic triangulation on the edge distances to compute the relative surface angle of the box.

This method is data-rich and allows precise angular estimation but requires significant processing power for real-time analysis.

Method 2: Multi-Sensor Triangulation (4 ToF Sensors)

The second approach uses four individual, front-facing ToF sensors, each mounted at fixed known positions on the EOAT.

Process Overview:

1. Each sensor provides a single-point distance measurement.
2. The four distances are used in a triangulation algorithm to calculate the relative tilt or rotation angle of the box surface.

This method is computationally lighter and easier to implement in embedded systems but may be less accurate in complex surface geometries or noisy environments.

Current Status

Both methods are to be under parallel development and testing. The final implementation will depend on:

- Accuracy and stability of the angle estimation
- Processing time and resource availability
- Integration compatibility with the EOAT system

10 Microcontroller Firmware

Microcontroller Firmware Functionality

The microcontroller firmware acts as the core control logic for the system, responsible for interfacing with sensors, actuators, and communication with the PC. It is designed to operate in real-time, executing a sequence of tasks that ensure coordinated and reliable operation of the full mechanical system.

Key Responsibilities

1. **Sensor Data Acquisition and Filtering:**
   The MCU communicates with four Time-of-Flight (ToF) sensors using its four independent UART channels. These sensors provide continuous distance measurements which are unpacked, filtered, and processed to calculate the relative angle between the end effector and the box surface. This angular information is critical for aligning the robot arm such that the end effector approaches the box perpendicularly.
2. **Motor Actuation and Feedback Processing:**
   The microcontroller interfaces with a motor driver to control the motor responsible for extending the rails that support the box. PWM signals are generated by the MCU to manage motor speed and direction. It also receives feedback signals from the motor, which are used to move between various states of the gripping and release sequences.
3. **Valve Switching and Vacuum Control:**
   Control of pneumatic and vacuum components is handled via GPIOs, enabling the microcontroller to:
   - Activate solenoid valves to control airflow.
   - Engage vacuum ejectors to create suction for the lifting mechanism.
   - Operate top support arms for box stabilization.

   Each valve or actuator is triggered based on sensor inputs and task state, enabling precise timing in sequences like suction engagement, lift, and box release.
4. **Serial Communication with Host System:**
   The microcontroller also maintains a serial communication link with the host PC that controls the robot arm. Through this interface, it can transmit angle data in real-time.

11 Schematic Design

11.1 Design Overview

11.1.1 ATmega2560 MCU

Selection Rationale: The ATmega2560 microcontroller was selected for this design due to its extensive I/O capabilities (86 programmable I/O pins), robust processing power, and support for multiple communication protocols, including UART, SPI, and I2C. This makes it ideal for managing the various peripherals required in this project, including time-of-flight (ToF) sensors, closed-loop stepper motors, and solenoids. The ATmega2560 handles input from ToF sensors, processes feedback from closed-loop stepper motors, and controls the switching of solenoids responsible for activating vacuum ejectors. This integration allows for precise control over the complex operations of the end effector.

11.1.2 Communication Interfaces

USB to TTL For programming the MCU and providing a serial communication interface, a USB-to-TTL converter is implemented using the CH340G chip. The CH340G was selected for its wide availability, low cost, and reliable USB-to-serial conversion capabilities. It operates with a 12MHz external crystal oscillator and is paired with decoupling capacitors for noise suppression. This circuit is also capable of providing power to the PCB via the USB connection, simplifying the overall power management design.

RS485 to TTL To enable robust, long-distance data communication, the MAX485 chip was selected for UART-to-RS485 conversion. RS485 was chosen for its noise immunity and differential signaling, which make it well-suited for industrial environments. This interface facilitates continuous data transfer from the ToF sensors to the PC, supporting real-time position feedback and control. The MAX485's small footprint and low power consumption make it an excellent choice for this application.

SPI Selection Rationale: The Serial Peripheral Interface (SPI) was chosen for its high-speed, synchronous communication, ideal for connecting high-bandwidth peripherals like external ADCs, memory modules, or future expansions. The ATmega2560's dedicated SPI pins are routed to headers J3A and J3B, providing a standard interface for SPI-compatible devices, ensuring flexibility for future design modifications.

11.1.3 Connectors

ToF Sensors The ToF sensors connect via JST BM04B-GHS headers, chosen for their compact design and secure locking mechanism. These headers are directly connected to the UART pins of the ATmega2560, ensuring reliable data transfer for precise positioning feedback.

Stepper Motor To facilitate closed-loop motor control, a 10-position connector is used, providing dedicated connections for power, ground, and control signals. This connector supports the high current requirements of the stepper motor, ensuring stable and reliable operation.

11.1.4 Solenoid Circuitry

Solenoid drivers are implemented using N-channel MOSFETs (IRLR type) paired with SS34 Schottky diodes for flyback protection. These components were selected for their low on-resistance, high current handling capabilities (up to 1A at 24V), and fast switching speeds, which are critical for precise solenoid actuation. The solenoids (Solenoid 1-5) are driven via dedicated MCU pins , each with a current-limiting resistor and a pull-down resistor to ensure defined behavior during operation.

11.1.5 Buck Converter

The ATmega2560 requires a stable 5V supply, provided by a 24V to 5V step-down buck converter based on the LM2678S-5.0. This regulator was chosen for its high efficiency (up to 92), 5A output capability, and fixed 260 kHz switching frequency, which minimizes component size and heat dissipation. The power supply design includes input capacitors (15µF), an output inductor (22µH), Schottky diode (VS-6TQ045STRR),

and output capacitors (180µF each) to minimize ripple and maintain stable operation. Additionally, a 16MHz crystal oscillator (ABM8AIG) provides the MCU's clock signal, with supporting capacitors (100nF) to filter noise and stabilize the timing circuits.

11.2 Schematics

11.2.1 Overview

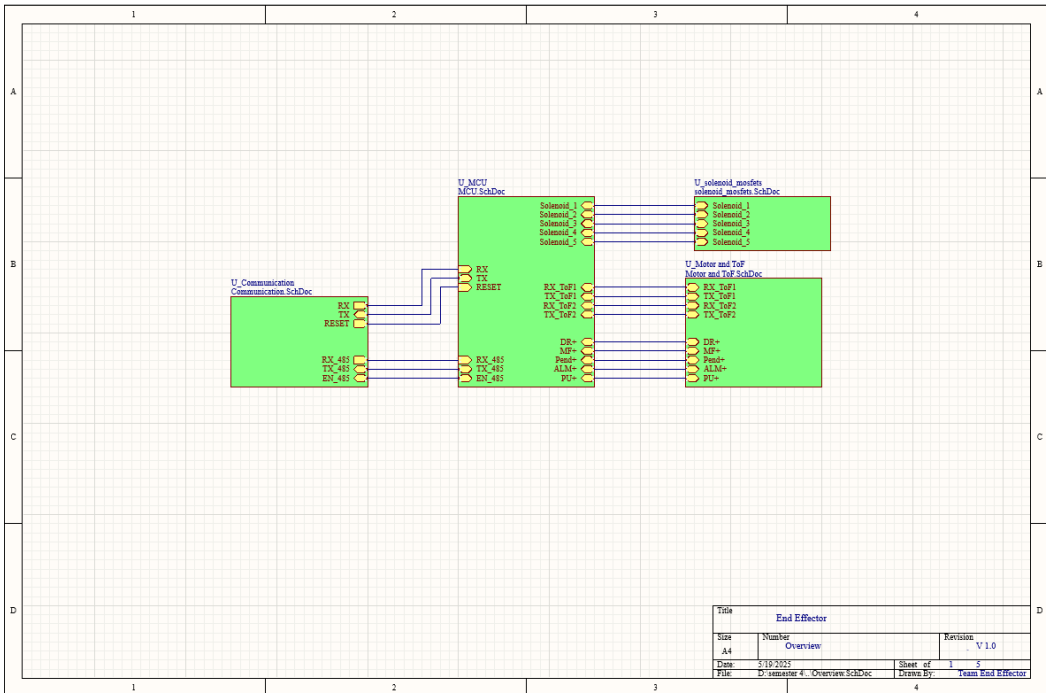

Figure 23: System Overview

11.2.2 Micro Controller Unit (ATMEGA2560)

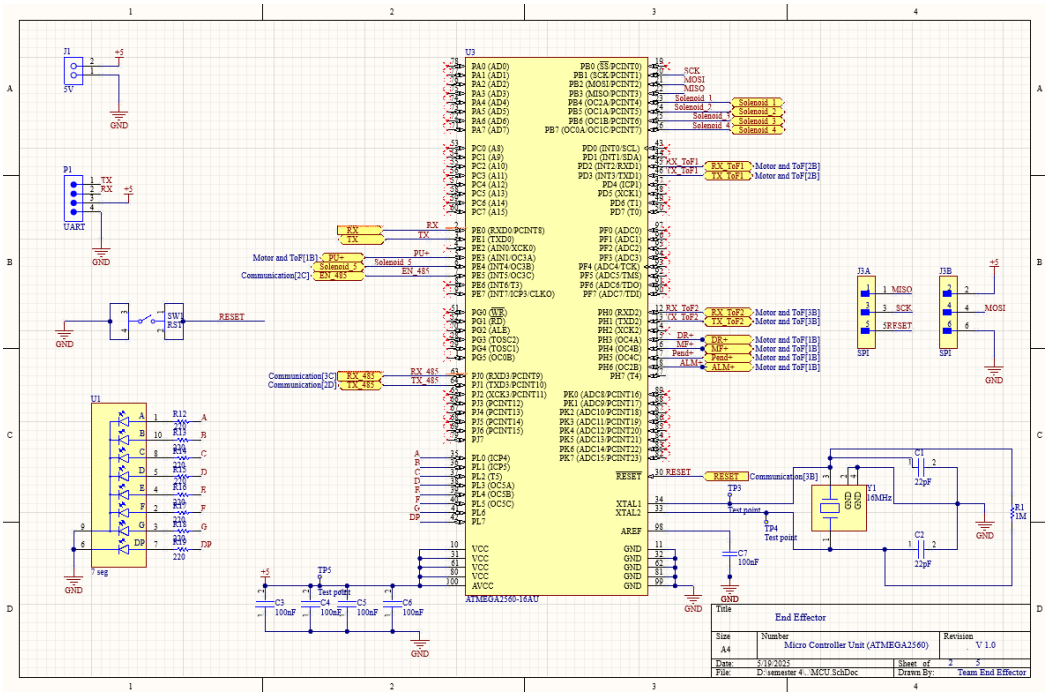

Figure 24: Micro Controller Unit (ATMEGA2560)

11.2.3 Motor and ToF headers

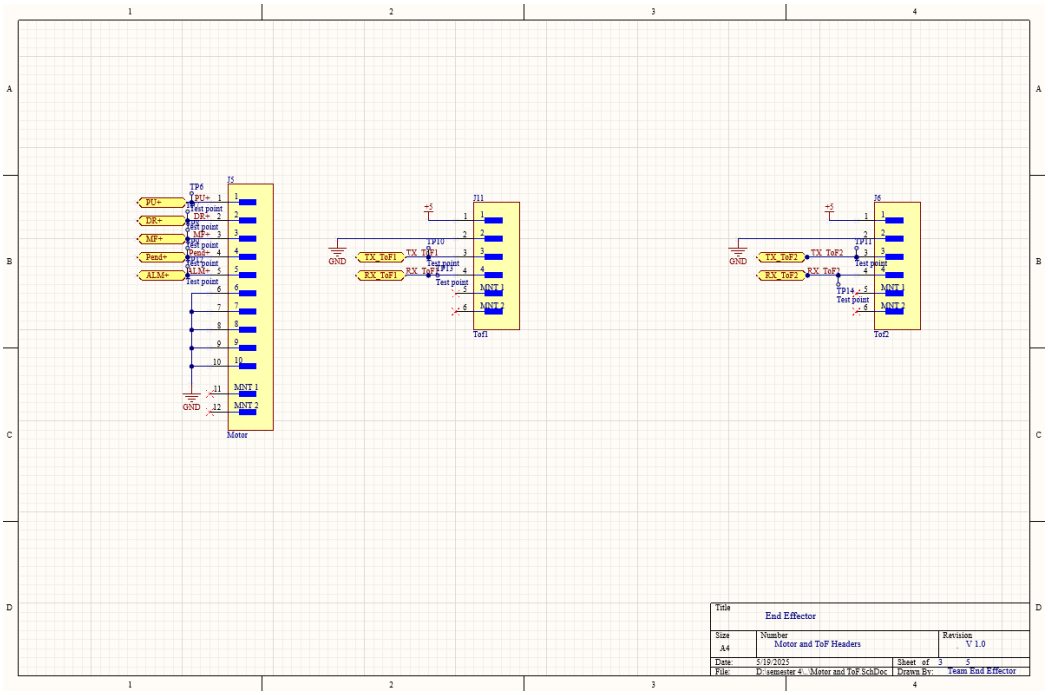

Figure 25: Motor and Tof Headers

11.2.4 Switching Circuitry for Solenoids

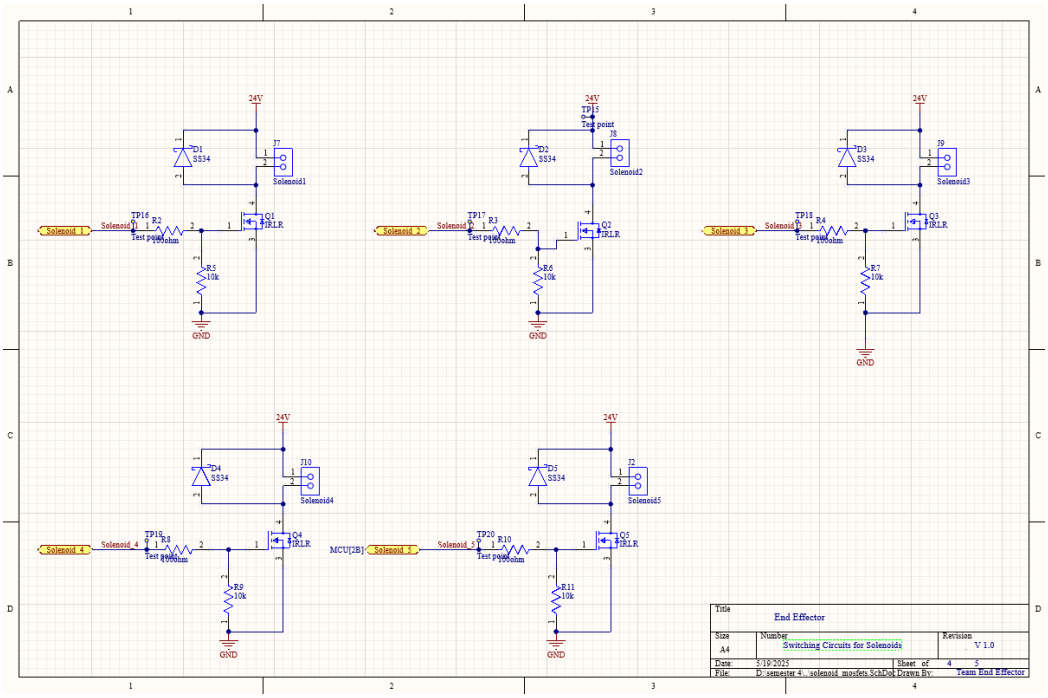

Figure 26: Switching circuitry for solenoids

11.2.5 USB-TTL and RS485-TTL Programming Interfaces

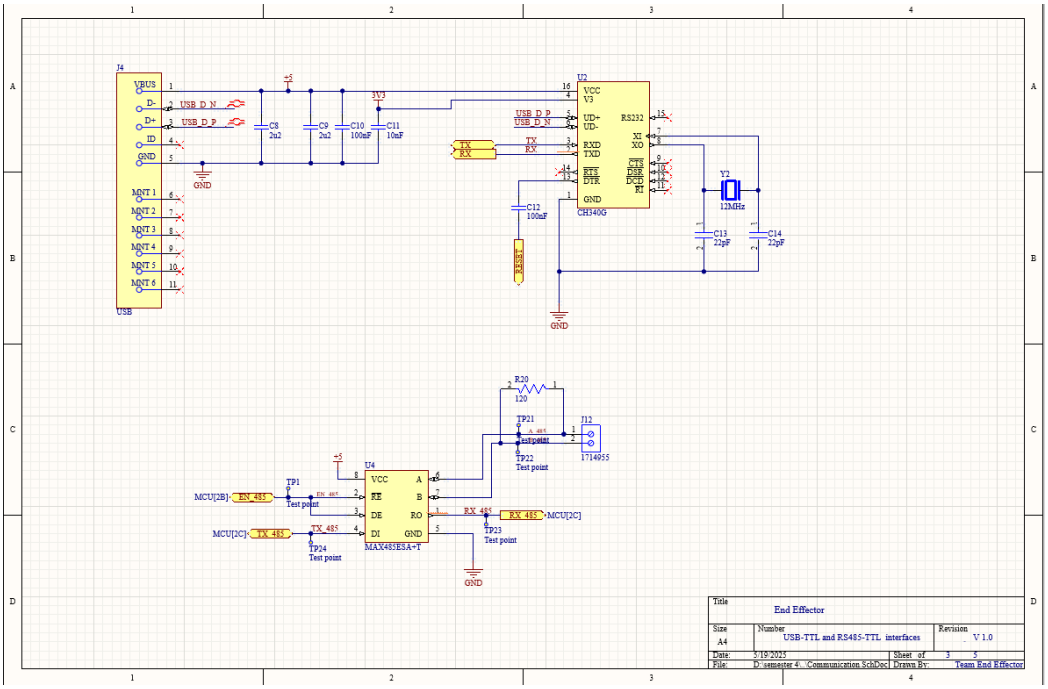

Figure 27: USB-TTL and RS485-TTL Programming Interfaces

11.2.6 24V to 5V Buck Converter

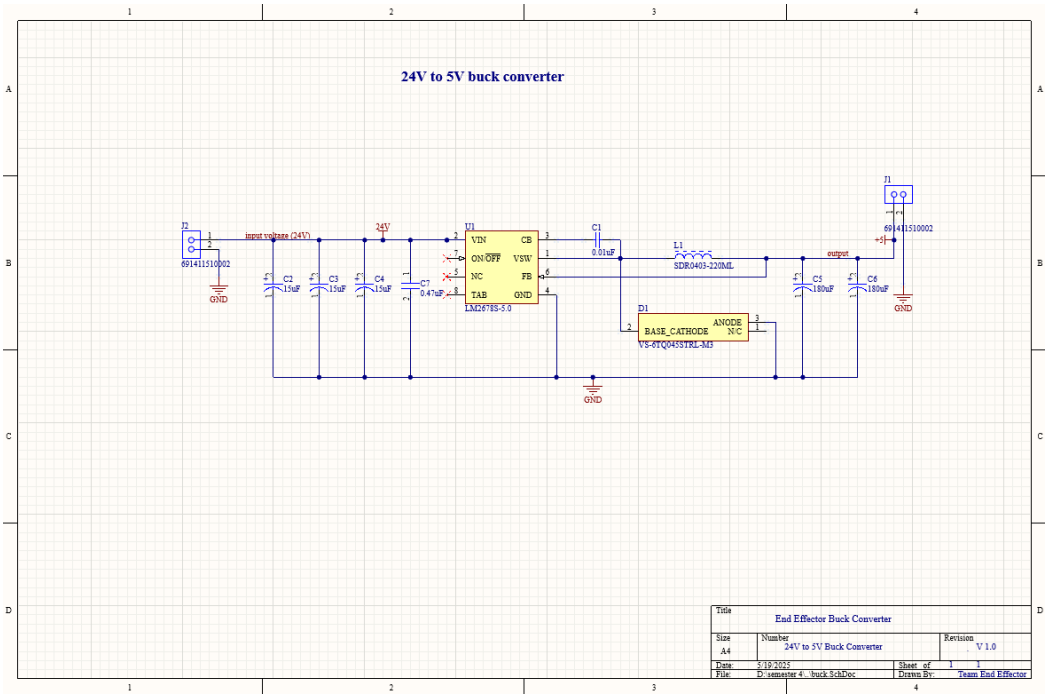

Figure 28: 24V to 5V Buck Converter

12 PCB Design

The PCB design process was carried out in two stages: development of the main control PCB and a separate power supply PCB based on a 24V–5V buck converter. Both designs followed industry-standard guidelines to ensure reliability, manufacturability, and efficient integration with the robotic system.

12.1 Main Control PCB

Once the schematic and component placement were finalized, corresponding footprints were generated for layout. The main PCB was designed as a **2-layer board** in compliance with **JLCPCB** manufacturing standards. This board integrates the Time-of-Flight (ToF) sensors, motor drivers, and RS485 communication circuitry.

Key design features of the main PCB include:

- **Test Points:** Added to all nets to facilitate debugging, validation, and signal probing.
- **Optimized Layout:** Ensured minimal signal interference and efficient routing of high-current paths.
- **RS485 Compatibility:** Routing and decoupling designed to support robust differential communication.
- **Design Rule Check (DRC):** A comprehensive DRC was performed to verify spacing, trace widths, and clearances.

Following validation, **Gerber and NC drill files** were generated for manufacturing.

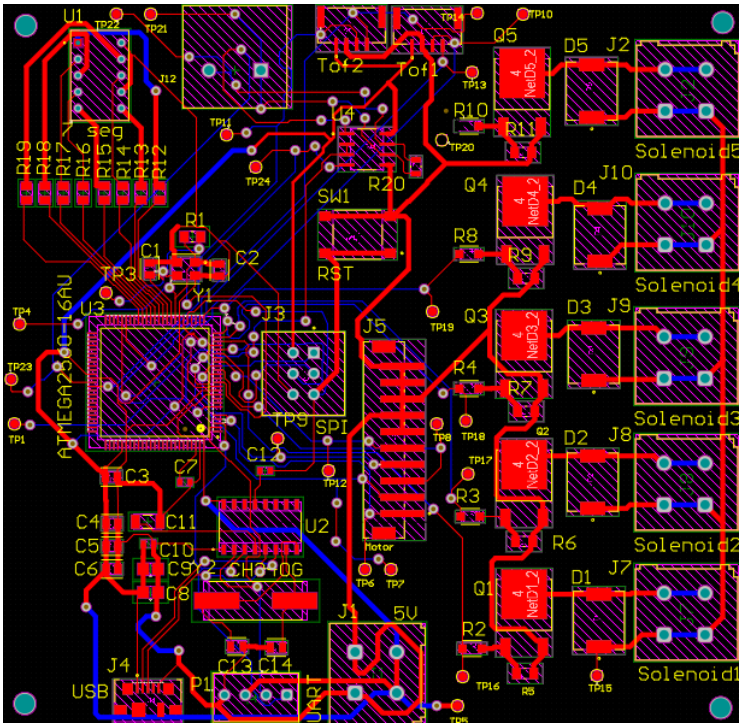

Figure 29: Routed layout of the main control PCB

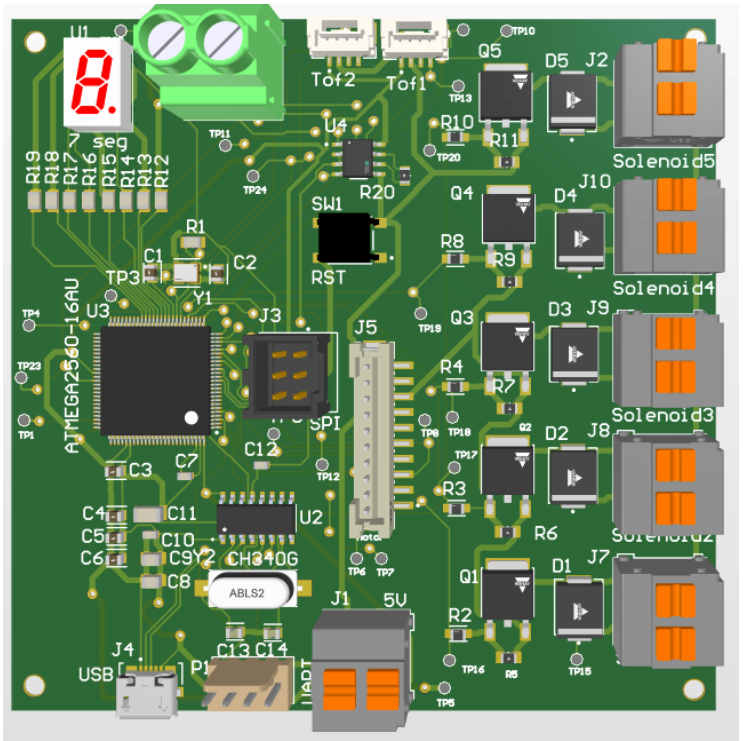

Figure 30: 3D rendered view of the main control PCB

12.2 Power Supply PCB – Buck Converter (24V to 5V)

To provide regulated power for the system, a dedicated **buck converter PCB** was designed to step down 24V input to a stable 5V output. This board was developed separately to ensure modularity and thermal isolation from the main control circuitry.

Key design aspects include:

- **High-Efficiency DC-DC Conversion:** Optimized for low ripple and heat dissipation.
- **Compact Layout:** Small form factor for easy mounting within the final enclosure.
- **Thermal Considerations:** Component spacing and copper pours were used to aid heat dissipation.

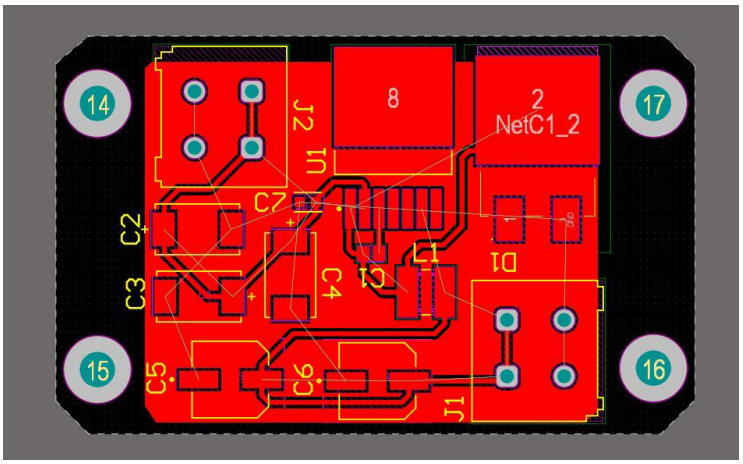

Figure 31: Top view of the 24V to 5V buck converter PCB

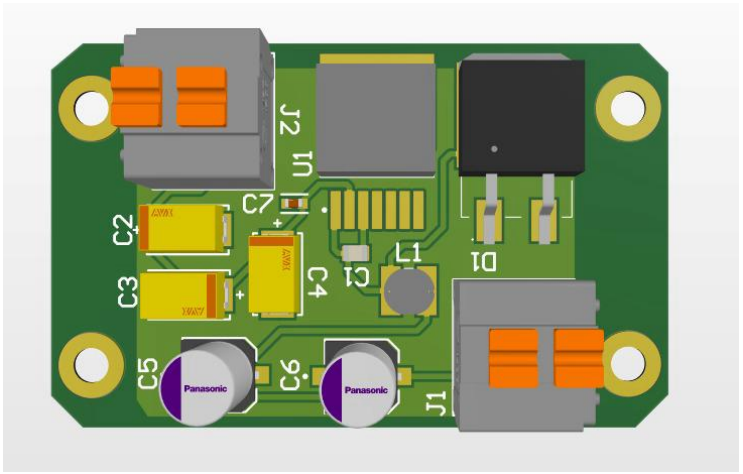

Figure 32: 3D rendered view of the buck converter PCB

13 Initial Enclosure Design and Evolution

The enclosure design process began with a 3D model developed in **SolidWorks**, focusing on compatibility with the robotic arm and end effector. This first version was created to:

- Securely house pneumatic components
- Allow easy access for assembly and maintenance

- Minimize material usage to maintain a lightweight form
- Support modularity for airflow and vacuum line routing adjustments
- Ensure spatial accuracy for robotic arm integration using SolidWorks tools

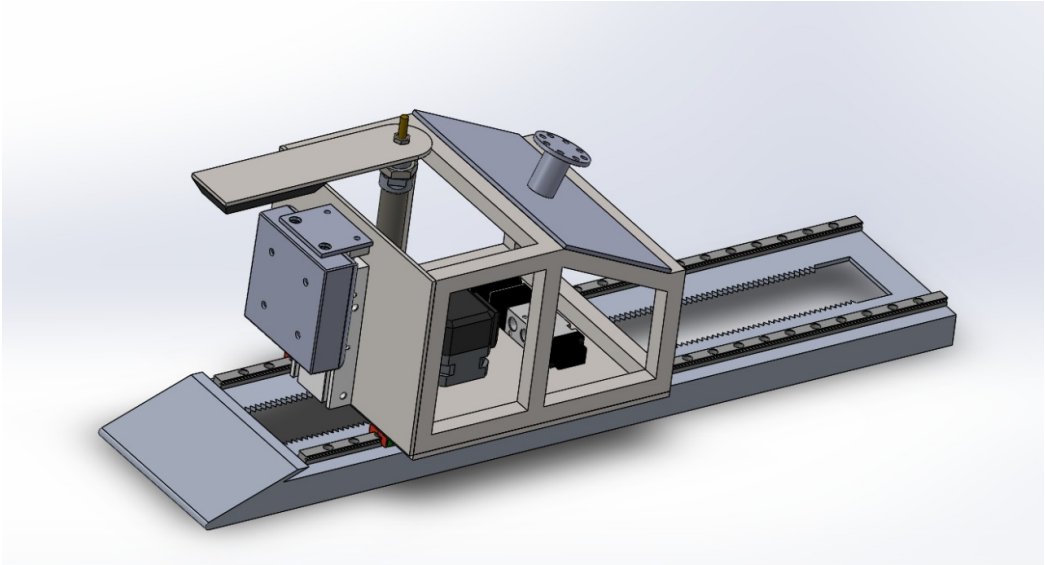

Figure 33: Initial CAD model of the enclosure designed in SolidWorks

To validate design feasibility, a **weight analysis** was performed using SolidWorks, ensuring that the structure would not hinder the robotic arm's performance.

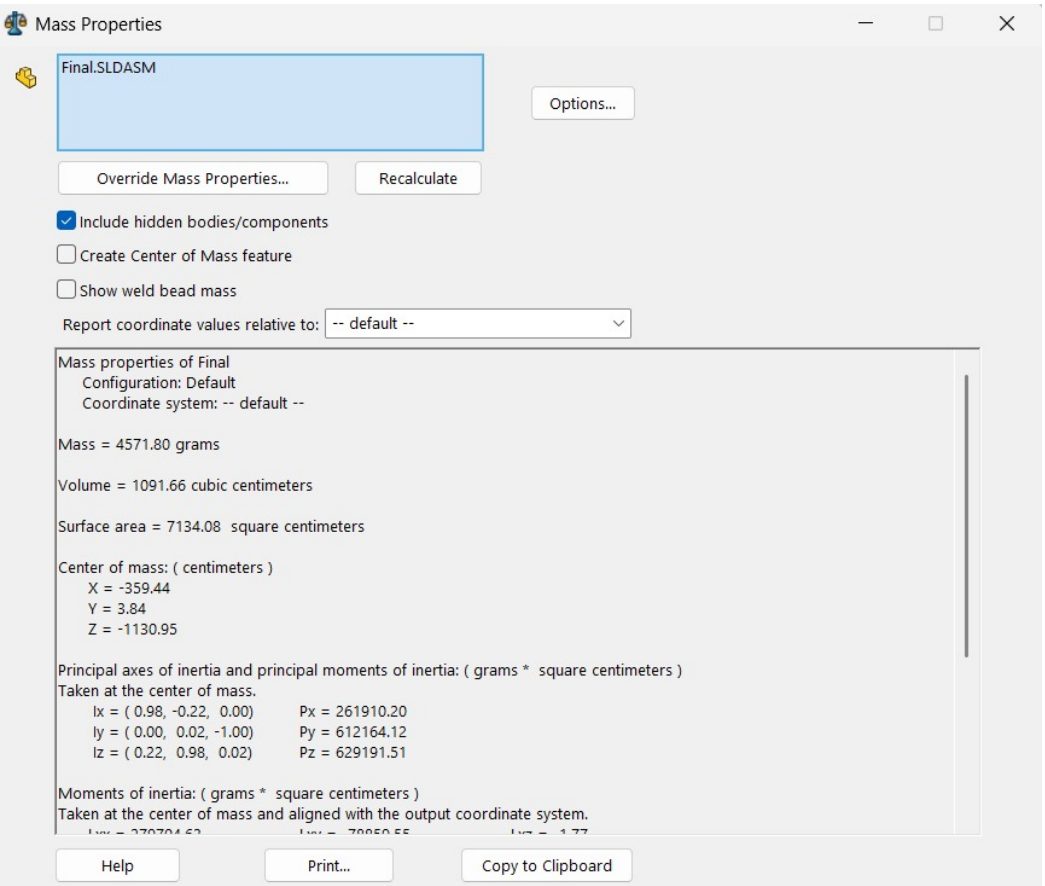

Figure 34: Weight analysis of the initial design using SolidWorks

The enclosure went through the following major transformations across three key design stages:

Design Stage Progression

- **Stage 1 – Acrylic Prototype:**
  - Used acrylic for quick prototyping and easy visual inspection
  - Allowed validation of component placement and dimensions
  - Found to be brittle and mechanically unsuitable for real-world use
- **Stage 2 – Zinc-Coated Iron Sheet:**
  - Introduced stronger material for better structural integrity
  - Increased weight and complexity of fabrication
  - Made assembly and maintenance more difficult
- **Stage 3 – Sheet Metal Refinement:**
  - Modeled entirely in SolidWorks with updated CAD geometry
  - Redesigned outer casing, flange section, and suction pad using sheet metal
  - Balanced strength with manufacturability and access to internal components
- **Final Stage – Integrated Enclosure with Electronics:**
  - Top support and suction pad transitioned from acrylic to metal

- Applied corrosion-resistant metal paint for a polished finish
- Soldered and installed the Power Supply PCB.
- Achieved a compact, deployment-ready system

14 Design Evolution

This section outlines the progressive development of the system's mechanical structure, moving from rapid prototyping to a robust and integrated final build. Each stage focused on improving durability, maintainability, and manufacturability based on real-world testing and feedback. Additionally, the evolution is framed following the Cambridge University Inclusive Design Toolkit principles, highlighting how user diversity, needs, and context were considered throughout.

14.1 Stage 1 – Acrylic Prototype

The initial stage employed **acrylic** due to its transparency and ease of machining, enabling quick prototyping and straightforward visual inspection of internal components. However, its low mechanical strength and brittleness made it unsuitable for prolonged use or load-bearing applications.

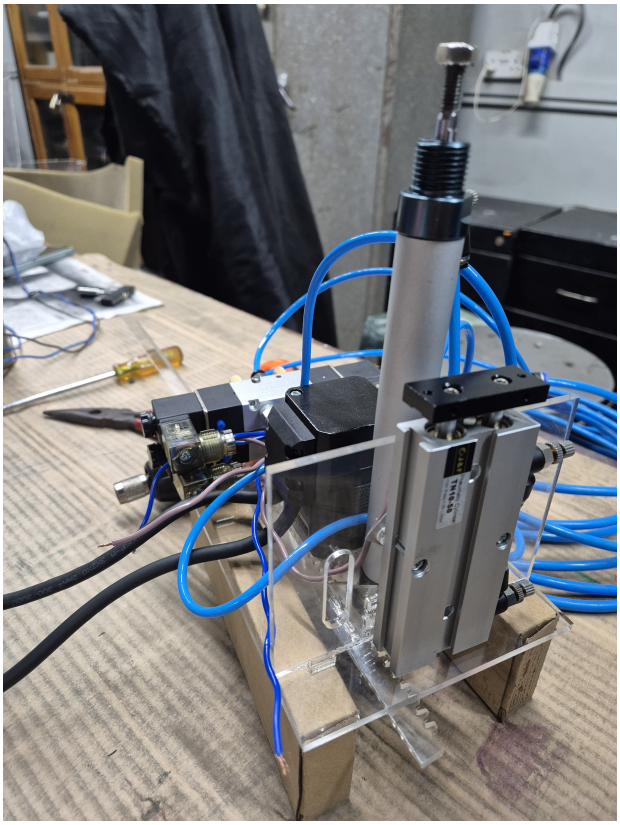

Figure 35: Stage 1: Acrylic prototype

Inclusive Design Perspective:

- *User Diversity:* Early users included technicians and developers requiring visual access for inspection and debugging.
- *User Needs:* Enabled easy visual feedback to support users with varied experience levels.
- *Context of Use:* Controlled, low-risk environments suited for quick iteration.
- *Design Response:* Transparent acrylic improved usability but lacked durability for broader contexts.

14.2 Stage 2 – Zinc-Coated Iron Sheet

To improve strength and protect internal electronics, the second stage introduced **zinc-coated iron sheets**. This material significantly enhanced rigidity and robustness. However, it also increased the weight of the assembly and introduced fabrication complexity, which affected ease of assembly and maintenance.

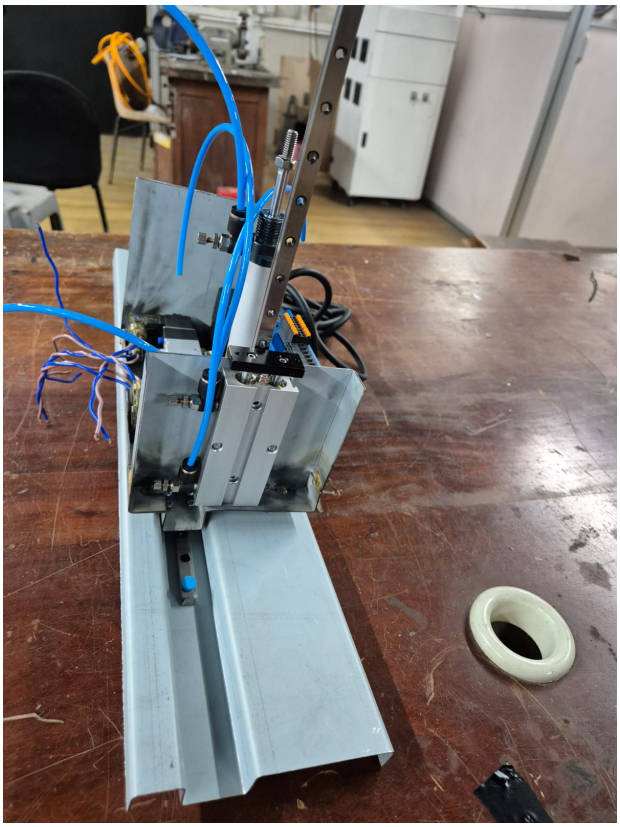

Figure 36: Stage 2: Zinc-coated iron sheet design

Inclusive Design Perspective:

- *User Diversity:* Targeted more demanding users operating in harsher or longer-term conditions.
- *User Needs:* Emphasized durability and protection while considering user handling capacity.
- *Context of Use:* Environments with increased mechanical and environmental stress.
- *Design Response:* Zinc coating improved robustness but introduced handling and maintenance challenges.

14.3 Stage 3 – Sheet Metal Enclosure

In this stage, we refined the enclosure using **custom sheet metal** to achieve a balance between structural integrity and ease of fabrication. The updated design improved both the mechanical robustness and the maintainability of the system.

Key improvements made during this stage:

- **Redesigned outer casing:** A new CAD model was developed to accommodate the sheet metal enclosure, improving overall rigidity and fit.
- **Flange section integration:** The flange section was newly modeled to ensure secure attachment points and alignment during assembly.

- **Top support and suction pad redesign:** Previously made from acrylic, these components were designed using acrylic for testing.

Below are the CAD models of the refined enclosure:

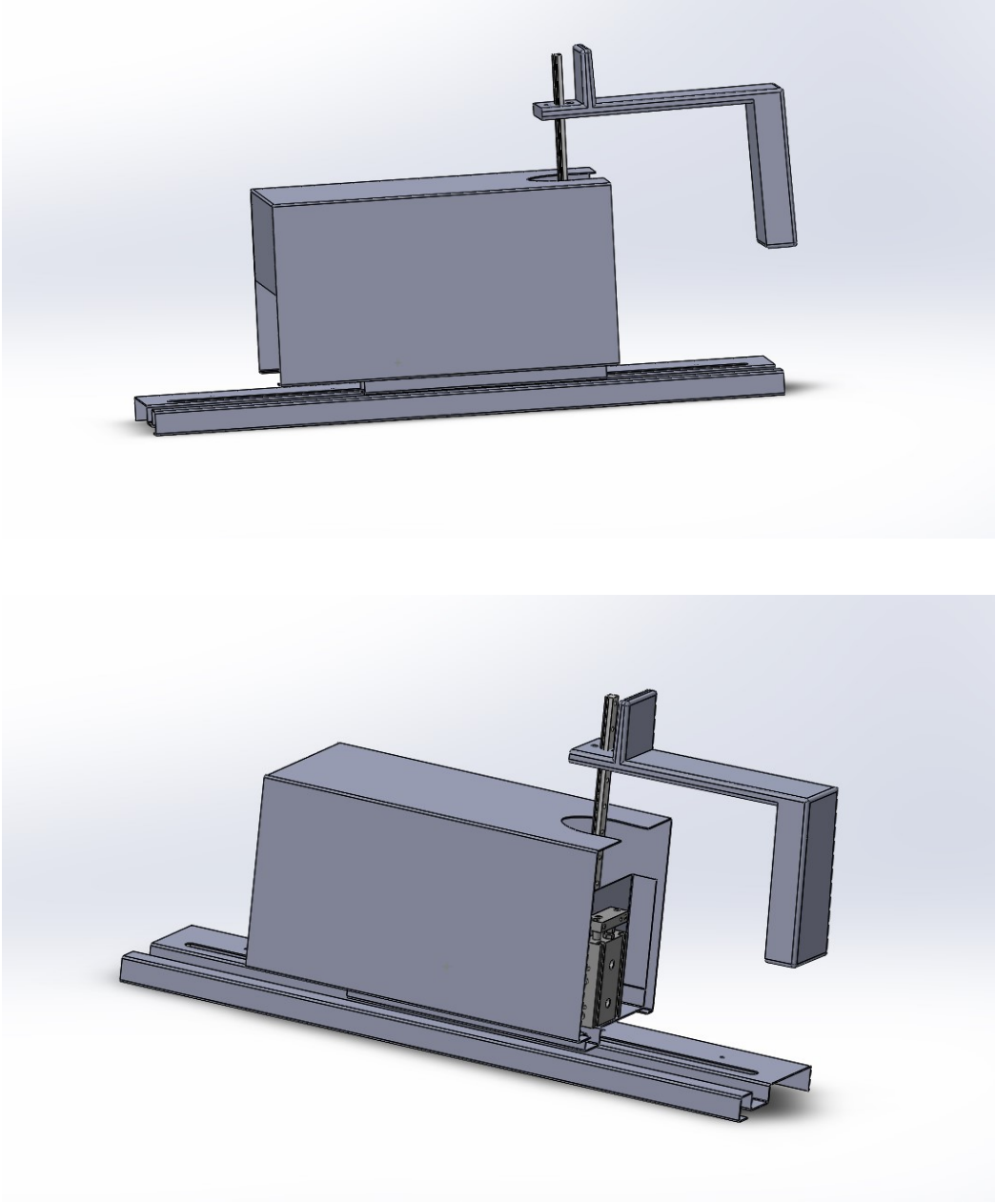

Figure 37: Stage 3: CAD models of the redesigned sheet metal enclosure

Inclusive Design Perspective:

- *User Diversity:* Designed for a broader user group including operators with varying technical skills.
- *User Needs:* Focus on maintainability and ease of assembly to reduce barriers.
- *Context of Use:* Realistic maintenance and operational environments.
- *Design Response:* Modular design and improved accessibility support diverse user capabilities.

14.4 Final Stage – Fully Integrated Metal Enclosure

The final stage focused on the complete integration and optimization of the system, transitioning from prototyping to a deployment-ready unit.

Key updates in this stage:

- **Redesigned Top Support and Suction Pad:** Previously made from acrylic, these components were reengineered using sheet metal to improve mechanical strength and stability.
- **Metal Paint Finish:** A protective metal paint coating was applied to enhance aesthetics and provide corrosion resistance.
- **Power Supply PCB Integration:** The power supply module (buck converter) was soldered and securely installed into the enclosure.
- **Improved Internal Layout:** Component positioning was optimized to ensure a more compact and maintainable design.

These enhancements resulted in a robust, self-contained system ready for final deployment.

Inclusive Design Perspective:

- *User Diversity:* Designed to accommodate end users and maintenance personnel with diverse physical and cognitive capabilities.
- *User Needs:* Emphasized durability, safety, and ease of maintenance to lower user burden.
- *Context of Use:* Intended for real-world, varied operational environments.
- *Design Response:* Holistic integration and optimized layouts support wide-ranging use cases.

Summary of Inclusive Design Approach

Inclusive Design Toolkit Element	**Implementation in Design Evolution**
User Diversity	Considered from prototype (technicians) to final wide user base including operators and maintainers
User Needs	Visibility, durability, ease of maintenance, handling, safety
Context of Use	From controlled lab conditions to real-world environments with physical and environmental stresses
Design Response	Material and structural changes to meet evolving needs, balancing usability and robustness
Feedback and Adaptation	Iterative improvements driven by user feedback and operational insights

Table 1: Inclusive design considerations through the system development stages

15 Updated Codebase

15.1 main.h

```
#ifndef __MAIN_H_
#define __MAIN_H_

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/boot.h>
#include "TofSensor.h"
#include "peripherals.h"
#include "uart.h"
#include <string.h>
#include <stdlib.h>

//extern UART uart0;
//extern UART uart1;

#endif
```

Listing 1: main.h - Project-Wide Includes and Definitions

15.2 uart.h

With the hardware finalized, the updated codebase—featuring refined firmware and control logic tailored for the new design—is presented in the following section.

```
#ifndef __UART_H_
#define __UART_H_

#include"main.h"
#include <avr/io.h>
#include <String.h>
#include <inttypes.h>

#define CMD_BUFFER_SIZE 32

extern char uart_buffer[CMD_BUFFER_SIZE];
extern volatile uint8_t uart_index;
extern volatile bool cmd_ready;

class UART{
private:
    // Private members if needed
    uint32_t baud_rate = 115200; // Default baud rate
    uint8_t ubrr_value = 16; // UBRR value for 115200 baud rate
    // UART registers (example for ATmega2560)
    volatile uint8_t *ubrrh;
    volatile uint8_t *ubrrl;
    volatile uint8_t *ucsra;
    volatile uint8_t *ucsrb;
    volatile uint8_t *ucsrc;
    volatile uint8_t *udr;
    uint32_t en_port = 0; // Port for enable pin
    uint32_t en_pin = 0; // Pin for enable pin
    bool enable_pin = false; // Flag to indicate if enable pin is used
```

```
30
31
32 public:
33     UART(volatile uint8_t *ubrrh,volatile uint8_t *ubrrl,volatile uint8_t *ucsra,
          volatile uint8_t *ucsrb,volatile uint8_t *ucsrc,volatile uint8_t *udr);
34     UART(volatile uint8_t *ubrrh,volatile uint8_t *ubrrl,volatile uint8_t *ucsra,
          volatile uint8_t *ucsrb,volatile uint8_t *ucsrc,volatile uint8_t *udr,
          volatile uint8_t en_port, uint8_t en_pin);
35
36     // Initialization function
37     void init();
38     void transmit(uint8_t data);
39     void transmit(const char *str);
40     void transmitv(long value);
41     uint8_t read();
42     bool available();
43     void transmit(float value, int precision);
44     void transmit(uint8_t *data, uint8_t size);
45     bool receive_line(char *buffer, size_t size);
46 };
47
48 #endif // __UART_H_
```

Listing 2: UART.h - UART Header File

15.3 tensor.h

```
1  #ifndef TOFSENSOR_H
2  #define TOFSENSOR_H
3
4  #define FRAME_SIZE 400
5  #define DATA_POINTS 64
6  #define POINT_SIZE 6
7  #define HEADER_BYTE_0 0x57
8  #define HEADER_BYTE_1 0x01
9
10 #include <math.h>
11 #include <stdint.h>
12 #include "main.h"
13 //#include <Arduino.h>
14
15 // Variable prototypes
16 extern uint8_t queryCommand[8];
17 extern uint8_t frame[FRAME_SIZE];
18 extern float distances[4];
19
20 float calculateAngleFromCorners(float d0, float d7, float d56, float d63);
21 int32_t parse_int24(uint8_t b0, uint8_t b1, uint8_t b2);
22 float calculateValue(uint8_t b0, uint8_t b1, uint8_t b2);
23 void printDistance(int index, float raw);
24 void sendQuery();
25 void tof_init();
26 void read_tof();
27 void UART0Init();
28 void UART1Init();
29 void UART0Transmit(const char *data);
30 void UART0Transmit(uint8_t data);
31 void UART0Transmit(uint8_t *data, uint8_t size);
32 void UART0Transmit(float value, int precision);
33 void UART1Transmit(const char *data);
34 void UART1Transmit(uint8_t data);
```

```
void UART1Transmit(uint8_t *data, uint8_t size);
void UART0Transmitc(char data);
uint8_t UART1Read();
bool UART1Available();

#endif // TOFSENSOR_H
```

Listing 3: TOFSensor.h - TOF Sensor Header File

15.4 peripherals.h

```
#ifndef __PERIPHERALS_H_
#define __PERIPHERALS_H_

#include "main.h"
#include <avr/io.h>
#include <util/delay.h>

void setup_seven_segment();
void seven_segment(int num = 0);

#endif
```

Listing 4: peripherals.h - Peripheral Control Header File

15.5 millis.h

```
#ifndef __MILLIS_H_
#define __MILLIS_H_

#include <stdint.h>

void millis_init();
uint32_t millis();

#endif
```

Listing 5: millis.h - Millisecond Timer Header File

15.6 fork.h

```
#ifndef __FORK_H_
#define __FORK_H_

#include "main.h"

void init_fork();
void drive_stepper(int distance_cm);

#endif // __FORK_H_
```

Listing 6: fork.h - Forklift Control Header File

15.7 pneumatics.h

```
#ifndef __PNEUMATICS_H_
#define __PNEUMATICS_H_

```

```
#include <inttypes.h>

void init_pneumatics();
void activate_solenoid(uint8_t solenoid);
void deactivate_solenoid(uint8_t solenoid);

#endif // __PNEUMATICS_H_
```

Listing 7: pneumatics.h - Pneumatics Control Header File

15.8 functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include "main.h"

void process_command(const char* cmd);
void pick_box();
void drop_box();
void lift(int state);
void grip(int state);
void vacuum(int state);

#endif // FUNCTIONS_H
```

Listing 8: functions.h - High-Level Control Function Declarations

15.9 main.cpp

```
#include "main.h"
#include "TofSensor.h"
#include "peripherals.h"
#include "fork.h"
#include "pneumatics.h"
#include "functions.h"
#include "millis.h"

UART uart3(&UBRR0H, &UBRR0L, &UCSR0A, &UCSR0B, &UCSR0C, &UDR0);
UART uart1(&UBRR1H, &UBRR1L, &UCSR1A, &UCSR1B, &UCSR1C, &UDR1);

UART uart0(&UBRR3H, &UBRR3L, &UCSR3A, &UCSR3B, &UCSR3C, &UDR3, PORTE, PORTE5);

int main()
{
  millis_init();
  uart0.init();
  uart3.init();
  setup_seven_segment();

  seven_segment(3);

  uart0.transmit("Booting...\n");
  init_pneumatics();
  tof_init();
  init_fork();
  uart0.transmit("End␣Effector␣Ready\n");

  uart3.transmit("UART3␣Initialized\n");

```

```
31    seven_segment(1);
32
33    while (1)
34    {
35      char command[64] = {0};
36      if (uart0.receive_line(command, sizeof(command)))
37      {
38        process_command(command);
39      }
40    }
41
42    while (1)
43    {
44      // activate_solenoid(0);
45      // activate_solenoid(1);
46      // activate_solenoid(2);
47      // activate_solenoid(3);
48      // activate_solenoid(4);
49      // _delay_ms(300);
50
51      // deactivate_solenoid(0);
52      // deactivate_solenoid(1);
53      // deactivate_solenoid(2);
54      // deactivate_solenoid(3);
55      // deactivate_solenoid(4);
56      // _delay_ms(300);
57    }
58
59    while (1)
60    {
61    }
62  }
```

Listing 9: main.cpp - Main Program Entry Point

15.10 uart.cpp

```
1  #include "uart.h"
2  #include "main.h"
3  #include "inttypes.h"
4  #include "avr/io.h"
5
6  UART::UART(volatile uint8_t *ubrrh, volatile uint8_t *ubrrl, volatile uint8_t *
       ucsra, volatile uint8_t *ucsrb, volatile uint8_t *ucsrc, volatile uint8_t *udr
       )
7      : ubrrh(ubrrh), ubrrl(ubrrl), ucsra(ucsra), ucsrb(ucsrb), ucsrc(ucsrc), udr(
           udr)
8  {
9      // Constructor code if needed
10 }
11
12 UART::UART(volatile uint8_t *ubrrh, volatile uint8_t *ubrrl, volatile uint8_t *
       ucsra, volatile uint8_t *ucsrb, volatile uint8_t *ucsrc, volatile uint8_t *udr
       , volatile uint8_t en_port, uint8_t en_pin)
13     : ubrrh(ubrrh), ubrrl(ubrrl), ucsra(ucsra), ucsrb(ucsrb), ucsrc(ucsrc), udr(
           udr), en_port(en_port), en_pin(en_pin)
14 {
15     enable_pin = true; // Set flag to indicate enable pin is used
16 }
17
18 void UART::init()
```

```
19 {
20     // Set baud rate
21     *ubrrh = (ubrr_value >> 8) & 0xFF;
22     *ubrrl = ubrr_value & 0xFF;
23 
24     if (enable_pin)
25     {
26         DDRE |= (1 << en_pin);  // Set enable pin as output
27         PORTE |= (1 << en_pin); // Set enable pin high if used
28     }
29     // Set UCSRnA for single speed mode
30     *ucsra |= (1 << U2X0); // Enable double speed mode if needed
31 
32     // Enable receiver and transmitter
33     *ucsrb |= (1 << RXEN0) | (1 << TXEN0);
34 
35     // Set frame format: 8 data bits, no parity, 1 stop bit
36     *ucsrc |= (1 << UCSZ01) | (1 << UCSZ00);
37 }
38 
39 void UART::transmit(uint8_t data)
40 {
41     if (enable_pin)
42     {
43         PORTE |= (1 << en_pin);
44     }
45     // Wait for empty transmit buffer
46     while (!(*ucsra & (1 << UDRE0)))
47         ;
48     // Put data into buffer, sends the data
49     *udr = data;
50 }
51 
52 void UART::transmit(const char *str)
53 {
54     while (*str)
55     {
56         transmit(*str);
57         str++;
58     }
59 }
60 
61 void UART::transmitv(long value)
62 {
63     char buffer[20];
64     ltoa(value, buffer, 10);
65     transmit(buffer);
66 }
67 
68 uint8_t UART::read()
69 {
70     // Wait for data to be received
71     while (!available())
72         ;
73     // Get and return received data from the buffer
74     return *udr;
75 }
76 
77 bool UART::available()
78 {
79     static unsigned long count = 0;
80     if (enable_pin)
```

```
81      {
82          if ((*ucsra & (1 << UDRE0)) && (*ucsra & (1 << TXC0)))
83          {
84              if (count > 1000)
85                  PORTE &= ~(1 << en_pin); // Set enable pin low before checking
                        availability
86              else
87                  count++;
88          }
89          else
90          {
91              count = 0;
92          }
93      }
94      // Check if data is available in the receive buffer
95      return (*ucsra & (1 << RXC0));
96  }
97
98  void UART::transmit(float value, int precision)
99  {
100     // char buffer[32];
101     // snprintf(buffer, sizeof(buffer), "%.*f", precision, (double)value);
102     // transmit(buffer);
103     char buffer[20];
104     dtostrf(value, 1, precision, buffer);
105     transmit(buffer);
106 }
107
108 void UART::transmit(uint8_t *data, uint8_t size)
109 {
110     for (uint8_t i = 0; i < size; i++)
111     {
112         transmit(data[i]);
113     }
114 }
115
116 bool UART::receive_line(char *buffer, size_t size)
117 {
118     if (!available())
119         return false; // No data available
120
121     size_t index = 0;
122     while (index < size - 1)
123     {
124         char c = read();
125         if (c == '\n' || c == '\r')
126         {
127             break; // End of line
128         }
129         buffer[index++] = c;
130     }
131     buffer[index] = '\0'; // Null-terminate the string
132     return true;          // Successfully received a line
133 }
```

Listing 10: uart.cpp - UART Class Implementation

15.11 TofSensor.cpp

```
1  #include <math.h>
2  #include "TofSensor.h"
```

```
#include "uart.h"

extern UART uart0;
extern UART uart1;

uint8_t queryCommand[8] = {0x57, 0x10, 0xFF, 0xFF, 0x03, 0xFF, 0xFF, 0x66};
uint8_t frame[FRAME_SIZE];
float distances[4];

constexpr float DEG2RAD = 3.14159265 / 180.0;

// You can adjust FoV if your sensor uses a different value
constexpr float H_FOV_DEG = 45.0;
constexpr float V_FOV_DEG = 45.0;

void tof_init()
{
    uart1.init(); // Initialize UART1
}

void read_tof()
{
    uart0.transmit("\n\n");

    sendQuery(); // Send the query command to the sensor

    // Wait for header
    while (true)
    {
        if (uart1.read() == HEADER_BYTE_0)
        {
            uart1.read(); // consume second header byte
            break;
        }
    }

    // Read the rest of the frame (FRAME_SIZE - 2 already read)
    int bytesRead = 0;
    while (bytesRead < FRAME_SIZE - 2)
    {
        if(uart1.available())
        {
            frame[bytesRead++] = uart1.read();
        }
    }
    //uart0.transmit("Frame received\n");

    // Parse and print corner distances (indexes: 0, 7, 56, 63)
    int dataStart = 7; // Skip ID (1), timestamp (3), and some flags (3)
    int distances_index = 0;
    for (int i = 0; i < DATA_POINTS; i++)
    {
        int offset = dataStart + i * POINT_SIZE;
        if (offset + 2 >= FRAME_SIZE)
            break;

        float distance = calculateValue(frame[offset], frame[offset + 1], frame[
            offset + 2]);

        // Only print four corners
        //if (i == 35 || i == 36 || i == 43 || i == 44)
        if (i == 0 || i == 7 || i == 56 || i == 63)
```

```
64          {
65              printDistance(i, distance);
66              distances[distances_index] = distance;
67              distances_index++;
68          }
69      }
70
71      calculateAngleFromCorners(distances[0], distances[1], distances[2], distances
            [3]);
72
73      //_delay_ms(2000); // Query interval
74  }
75
76  // Estimate the angle (in degrees) of the object relative to the center
77  // from 4 corner distances (in mm or cm     consistent units)
78  float calculateAngleFromCorners(float d0, float d7, float d56, float d63)
79  {
80      // Pixel grid positions for the 4 corners
81      int x0 = 0, y0 = 0;
82      int x7 = 7, y7 = 0;
83      int x56 = 0, y56 = 7;
84      int x63 = 7, y63 = 7;
85
86      // Angle per pixel
87      float hAnglePerPixel = H_FOV_DEG / 7.0;
88      float vAnglePerPixel = V_FOV_DEG / 7.0;
89
90      // Convert pixel positions to angles (centered at 0)
91      float ax0 = (x0 - 3.5) * hAnglePerPixel;
92      float ay0 = (y0 - 3.5) * vAnglePerPixel;
93      float ax7 = (x7 - 3.5) * hAnglePerPixel;
94      float ay7 = (y7 - 3.5) * vAnglePerPixel;
95      float ax56 = (x56 - 3.5) * hAnglePerPixel;
96      float ay56 = (y56 - 3.5) * vAnglePerPixel;
97      float ax63 = (x63 - 3.5) * hAnglePerPixel;
98      float ay63 = (y63 - 3.5) * vAnglePerPixel;
99
100     // Convert to 3D coordinates
101     float x_sum = d0 * tan(ax0 * DEG2RAD) +
102                   d7 * tan(ax7 * DEG2RAD) +
103                   d56 * tan(ax56 * DEG2RAD) +
104                   d63 * tan(ax63 * DEG2RAD);
105
106     float y_sum = d0 * tan(ay0 * DEG2RAD) +
107                   d7 * tan(ay7 * DEG2RAD) +
108                   d56 * tan(ay56 * DEG2RAD) +
109                   d63 * tan(ay63 * DEG2RAD);
110
111     float z_sum = d0 + d7 + d56 + d63;
112
113     // Average to get direction vector
114     float x_avg = x_sum / 4.0;
115     float y_avg = y_sum / 4.0;
116     float z_avg = z_sum / 4.0;
117
118     // Compute angle from center in degrees
119     float horizontalAngle = 10*atan2(x_avg, z_avg) * 180.0 / 3.14159265;
120     float verticalAngle = 10*atan2(y_avg, z_avg) * 180.0 / 3.14159265;
121
122     uart0.transmit("Horizontal␣Angle:␣");
123     uart0.transmit(horizontalAngle, 2);
124     uart0.transmit("␣Vertical␣Angle:␣");
```

```
    uart0.transmit(verticalAngle, 2);
    uart0.transmit("\n");

    // Optional: return horizontal only, or both as struct if needed
    return horizontalAngle; // or verticalAngle
}

// Parse int24 as per protocol: (b0 << 8 | b1 << 16 | b2 << 24) / 256
int32_t parse_int24(uint8_t b0, uint8_t b1, uint8_t b2)
{
    int32_t value = ((int32_t)b2 << 16) | ((int32_t)b1 << 8) | b0;
    // Sign extend if the top bit is set (for 24-bit signed)
    if (value & 0x800000)
        value |= 0xFF000000;
    return value;
}

float calculateValue(uint8_t b0, uint8_t b1, uint8_t b2)
{
    int32_t temp = parse_int24(b0, b1, b2);
    return temp / 1000.0; //mm
}

void printDistance(int index, float raw)
{
    float mm = raw;
    uart0.transmit("Index␣");
    uart0.transmit("␣");
    //uart0.transmit((float)index, 0);
    uart0.transmit(":␣");
    uart0.transmit(mm, 3);
    uart0.transmit("␣mm\n");
}

void sendQuery()
{
    uart1.transmit(queryCommand, 8);
}
```

Listing 11: TofSensor.cpp - Time-of-Flight Sensor Interface

15.12 peripherals.cpp

```
#include "peripherals.h"
#include "main.h"
#include "TofSensor.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <stdint.h>

void setup_seven_segment()
{
    DDRL = 0xFF; // Set all pins of PORTL as output
}

void seven_segment(int num)
{
    int number[] = {0b1110111, 0b0010010, 0b1011101,
                    0b1011011, 0b0111010, 0b1101011,
```

```
                    0b1101111, 0b1010010, 0b1111111,
                    0b1111011}; // 0-9

    int segment[] = {3, 2, 4, 6, 1, 5, 0}; // Segment mapping for seven-segment
        display

    PORTL = 0x00;
    PORTL &= ~(1 << 7);
    //_delay_ms(200);

    for (int i = 0; i < 7; i++)
    {
        if (number[num] & (1 << (6 - i)))
            PORTL |= (1 << segment[i]); // Set the corresponding segment high
                                        //_delay_ms(200);               // Wait
                                           for 200 ms
    }
}
```

Listing 12: peripherals.cpp - Seven Segment Display Control

15.13 millis.cpp

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "millis.h"

volatile uint32_t millis_counter = 0;

void millis_init()
{
    // Set up Timer0 to overflow every 1ms
    TCCR0A = 0x00;                      // Normal mode
    TCCR0B = (1 << CS01) | (1 << CS00); // Prescaler 64
    TIMSK0 = (1 << TOIE0);              // Enable overflow interrupt
    TCNT0 = 0;
    sei(); // Enable global interrupts
}

ISR(TIMER0_OVF_vect)
{
    // 16MHz / 64 = 250kHz     256 ticks = 1.024ms     1ms
    millis_counter++;
}

uint32_t millis()
{
    uint32_t m;
    cli();
    m = millis_counter;
    sei();
    return m;
}
```

Listing 13: millis.cpp - Millisecond Timer Implementation

15.14 fork.cpp

```
#include "fork.h"
#include "main.h"
```

```
3
4  void init_fork()
5  {
6    DDRH |= 0b01111000; // Configure PH3-PH6 as output - stepper motor control
7    DDRE |= 1 << 3;
8    PORTH &= ~(1 << 4);
9  }
10
11 void drive_stepper(int distance_cm)
12 {
13   const int steps_per_cm = 10000; // Adjust this value based on your stepper
       motor's configuration
14   int steps = abs(distance_cm) * steps_per_cm;
15   bool direction = (distance_cm > 0); // True for forward, false for backward
16
17   // Set direction pin (PH3) D7
18   if (direction)
19     PORTH |= (1 << 3); // Forward
20   else
21     PORTH &= ~(1 << 3); // Backward
22
23   PORTH &= ~(1 << 4);
24
25   for (int i = 0; i < steps; i++)
26   {
27     // Generate pulse on PH5 D8
28     PORTE |= (1 << 3);  // Pulse high
29     _delay_us(10);      // Adjust delay for pulse width
30     PORTE &= ~(1 << 3); // Pulse low
31     _delay_us(10);      // Adjust delay for step interval
32   }
33
34   // Disable stepper motor (PH3)
35   PORTH |= (1 << 4);
36 }
```

Listing 14: fork.cpp - Stepper Motor Control

15.15 pneumatics.cpp

```
1  #include "pneumatics.h"
2  #include <inttypes.h>
3  #include "peripherals.h"
4
5  /*
6  -PB4 - top gripper D10
7  -PB5 - suction lifter D11
8  -PB6 - suction gripper D12
9  -PB7 - reserved D13
10 */
11 void init_pneumatics()
12 {
13   DDRB |= 0b11110000;     // Set PB4-PB7 solenoids as output
14   PORTB &= ~(0b11110000); // Set PB4-PB7 solenoids low
15   // PORTB |= (0b11110000);
16   DDRE |= (1 << 4);   // Set PE4 solenoid as output
17   PORTE &= ~(1 << 4); // Set solenoid low
18 }
19
20 void activate_solenoid(uint8_t solenoid)
21 {
```

```
 switch (solenoid)
  {
  case 0:                // Top gripper
    PORTB |= (1 << 4); // Set PB4 high
    break;
  case 1:                // Suction lifter
    PORTB |= (1 << 5); // Set PB5 high
    break;
  case 2:                // Suction gripper
    PORTB |= (1 << 6); // Set PB6 high
    break;
  case 3:                // Reserved
    PORTB |= (1 << 7); // Set PB7 high
    break;
  case 4:                // Additional solenoid on PE4
    PORTE |= (1 << 4); // Set PE4 high
    break;
  default:
    break; // Invalid solenoid number
  }
  }

void deactivate_solenoid(uint8_t solenoid)
{
  switch (solenoid)
  {
  case 0:                 // Top gripper
    PORTB &= ~(1 << 4); // Set PB4 low
    break;
  case 1:                 // Suction lifter
    PORTB &= ~(1 << 5); // Set PB5 low
    break;
  case 2:                 // Suction gripper
    PORTB &= ~(1 << 6); // Set PB6 low
    break;
  case 3:                 // Reserved
    PORTB &= ~(1 << 7); // Set PB7 low
    break;
  case 4:                 // Additional solenoid on PE4
    PORTE &= ~(1 << 4); // Set PE4 low
    break;
  default:
    break; // Invalid solenoid number
  }
}
```

Listing 15: pneumatics.cpp - Solenoid Control

15.16 functions.cpp

```
#include "functions.h"
#include "main.h"
#include "fork.h"
#include "peripherals.h"
#include "uart.h"
#include "pneumatics.h"

#define LIFT_SOLENOID_1 0
#define LIFT_SOLENOID_2 1
#define GRIP_SOLENOID_1 2
#define GRIP_SOLENOID_2 3
```

```
12 #define VACUUM_SOLENOID 4
13
14 extern UART uart0;
15
16 void process_command(const char *cmd)
17 {
18   if (strncmp(cmd, "lift=", 5) == 0)
19   {
20     int state = atoi(cmd + 5);
21     lift(state);
22     if (state == 0)
23     {
24       seven_segment(0);
25     }
26     else if (state == 1)
27     {
28       seven_segment(1);
29     }
30   }
31   else if (strncmp(cmd, "grip=", 5) == 0)
32   {
33     int state = atoi(cmd + 5);
34     grip(state);
35     if (state == 0)
36     {
37       seven_segment(2);
38     }
39     else if (state == 1)
40     {
41       seven_segment(3);
42     }
43     else if (state == 2)
44     {
45       seven_segment(4);
46     }
47   }
48   if (strncmp(cmd, "vacu=", 5) == 0)
49   {
50     int state = atoi(cmd + 5);
51     vacuum(state);
52     if (state == 0)
53     {
54       seven_segment(5);
55     }
56     else if (state == 1)
57     {
58       seven_segment(6);
59     }
60   }
61   if (strncmp(cmd, "fork␣=", 5) == 0)
62   {
63     int value = atoi(cmd + 5);
64     uart0.transmitv(value);
65     drive_stepper(value);
66   }
67
68   if (strncmp(cmd, "read_tof", 8) == 0)
69   {
70     read_tof();
71   }
72 }
73
```

```
74 void lift(int state)
75 {
76   if (state == 0)
77   {
78     activate_solenoid(LIFT_SOLENOID_1);
79     deactivate_solenoid(LIFT_SOLENOID_2);
80   }
81   else if (state == 1)
82   {
83     activate_solenoid(LIFT_SOLENOID_2);
84     deactivate_solenoid(LIFT_SOLENOID_1);
85   }
86   else if (state == 2)
87   {
88     deactivate_solenoid(LIFT_SOLENOID_1);
89     deactivate_solenoid(LIFT_SOLENOID_2);
90   }
91 }
92
93 void grip(int state)
94 {
95   if (state == 0)
96   {
97     activate_solenoid(GRIP_SOLENOID_1);
98     deactivate_solenoid(GRIP_SOLENOID_2);
99   }
100  else if (state == 1)
101  {
102    activate_solenoid(GRIP_SOLENOID_2);
103    deactivate_solenoid(GRIP_SOLENOID_1);
104  }
105  else if (state == 2)
106  {
107    deactivate_solenoid(GRIP_SOLENOID_1);
108    deactivate_solenoid(GRIP_SOLENOID_2);
109  }
110 }
111
112 void vacuum(int state)
113 {
114   if (state == 0)
115   {
116     activate_solenoid(VACUUM_SOLENOID); // Activate vacuum
117   }
118   else if (state == 1)
119   {
120     deactivate_solenoid(VACUUM_SOLENOID); // Deactivate vacuum
121   }
122 }
123
124 void pick_box()
125 {
126   vacuum(1);
127   _delay_ms(1000); // Wait for 1 second
128   // move closer to the box
129   lift(1);
130   _delay_ms(1000);   // Wait for 1 second
131   drive_stepper(10); // activate bottom rail
132   lift(0);
133   _delay_ms(1000); // Wait for 1 second
134   vacuum(0);
135   _delay_ms(1000); // Wait for 1 second
```

```
136   grip(1);
137   _delay_ms(1000); // Wait for 1 second
138                    // move away from the box
139 }
140
141 void drop_box()
142 {
143   grip(0);
144   _delay_ms(1000);    // Wait for 1 second
145   drive_stepper(-10); // retract bottom rail
146 }
```

Listing 16: functions.cpp - Command Processing and Actuator Control

16 Conclusion

This report documented the end-to-end design and development process of the mechanical enclosure for a robotic vacuum gripper system. Starting from an acrylic prototype, the design evolved through multiple stages, addressing key challenges in durability, structural integrity, and ease of assembly. The use of SolidWorks for CAD modeling and simulation enabled precise spatial planning and weight analysis, ensuring the enclosure would meet mechanical requirements and fit within the constraints of a robotic manipulation platform.

The final design, constructed entirely from sheet metal, successfully housed the pneumatic components and integrated the power supply and control PCBs. A corrosion-resistant metal paint finish added both aesthetic appeal and long-term protection, resulting in a compact and robust enclosure suitable for deployment.

While the project achieved most of its technical objectives, one key goal—full mechanical and control integration with the robotic arm—remains incomplete. This integration will be addressed in future development stages, including real-time control synchronization, mounting validation, and dynamic testing under load.

Despite this limitation, the project provides a strong foundation for further enhancements. The modular and maintainable design ensures that the system can be easily adapted for complete integration in subsequent iterations. Overall, the work demonstrates a successful approach to enclosure design for robotic systems, combining mechanical reliability with thoughtful system-level engineering.