Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN 2160 – Electronic Design Realization

Design Documentation(Resubmission)

Verifying the accurate placement of labels on milk packets and rejecting faulty ones

Group no (O) 41

210387D Mihiranga N.G.D.
210391J Morawakgoda M.K.I.G.

07 July 2024

Contents

1.Hardware Design

1.1. Introduction

Our project aims to develop an automated system capable of detecting misprints on product packaging. The primary objective of this project is to identify instances where critical information such as expiration dates, manufacturing dates, and price tags deviate from the designated white space on the packaging. Upon detection of misprints or deviations from the expected placement of information within the designated white space, the system will trigger a mechanism to reject the affected packaging from the conveyor belt.

1.2. Circuit design

1.2.1. PCB Design

Micro Controller Unit

The MCU schematic includes the Atmel ATmega328P IC. The J1 connector is for an external reset button. The J2 connector is for motor driver 1 signals. The J3 connector is for motor driver 2 signals. The J4 connector is for input signals from the limit switches.

USB to TTL

The CH340C IC is used for real-time communication between the computer, which is used for image processing, and the MCU. For this we have used a USB Type-A port(J7). Two LEDs are included on the PCB for indicating transmission and reception.

Power

There are two ways to input power to this PCB: one is via the USB port, and the other is through a direct 12V input from the power supply. For regulation, we used the LM7800 power regulator IC.

We have designed 2-layer PCB for this project.

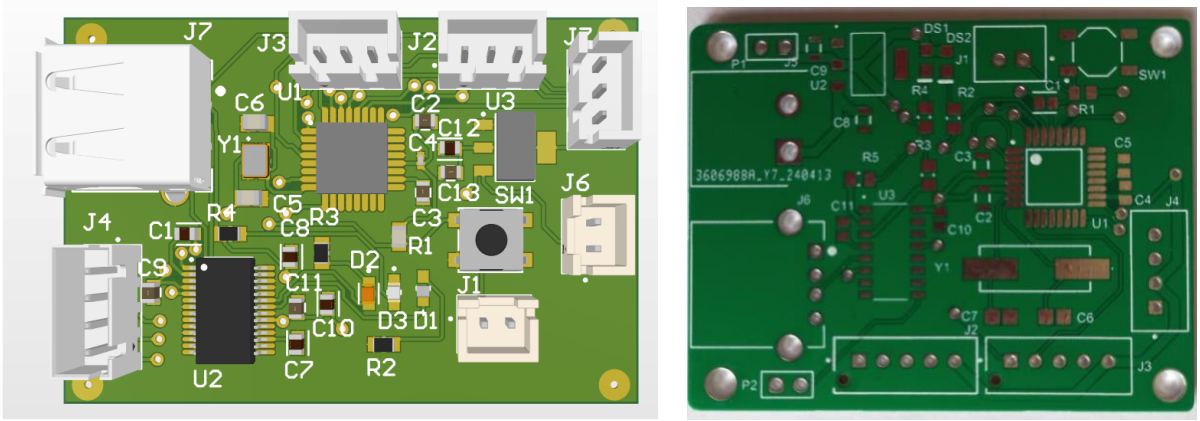


Figure 1: 3D view of the PCB design and the Printed PCB

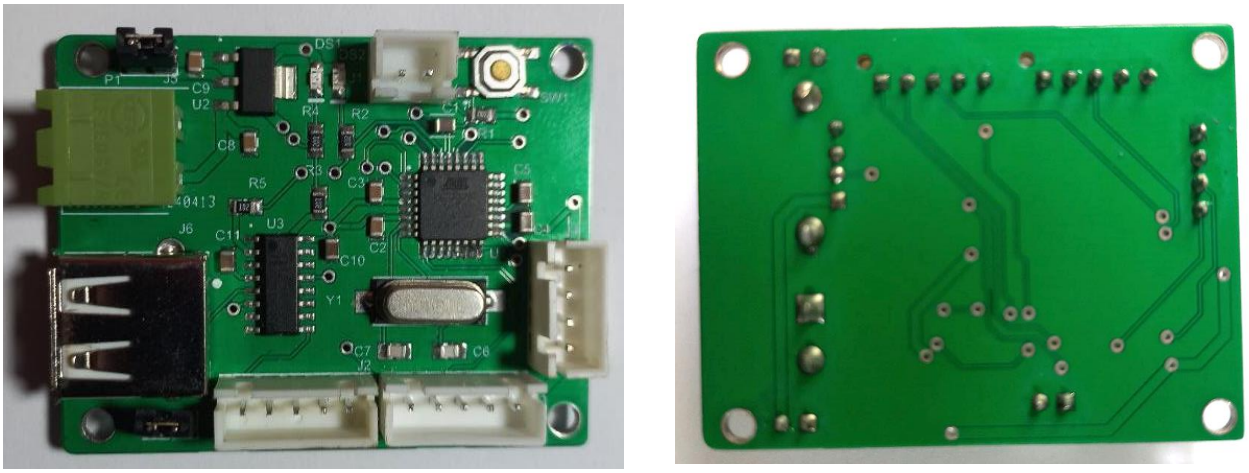


Figure 2: Soldered PCB

Components of PCB

Item	Designator	Part
1	U1	ATmega328P-AU 8-bit AVR Microcontroller
2	U2	1A Low Dropout Voltage Regulator
3	U3	CH340C IC
4	C1	0.1uF Ceramic Capacitors
5	C2, C3, C8, C9	10uF Ceramic Capacitors
6	C4, C5, C10	100nF Ceramic Capacitors
7	C6, C7	22pF Ceramic Capacitors
8	P1	Connector
9	R1	Chip Resistor, 10 KOhm
10	R2, R3, R4, R5	Chip Resistor, 1 KOhm
11	DS1, DS2	Chip LED
12	J1	CONN HEADER VERT 2POS 2.5MM
13	J2, J3	CONN HEADER VERT 5POS 2.5MM
14	J4	CONN HEADER VERT 4POS 2.5MM
15	J5	Connector
16	J6	1 Port Right Angle Female Type A USB Connector
17	Y1	16MHz Crystal Oscillator
18	SW1	SPST Tactile Switch

PCB Testing

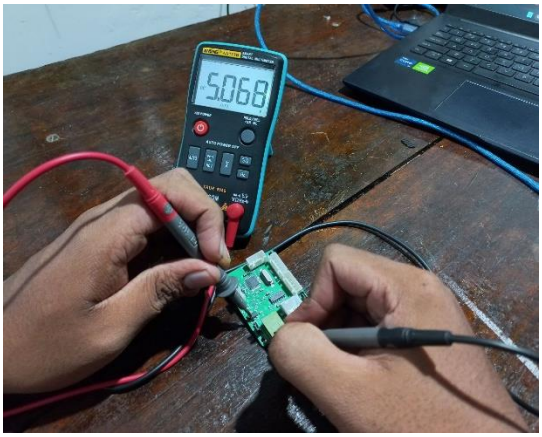


Figure 3: PCB testing - Input voltage from USB port

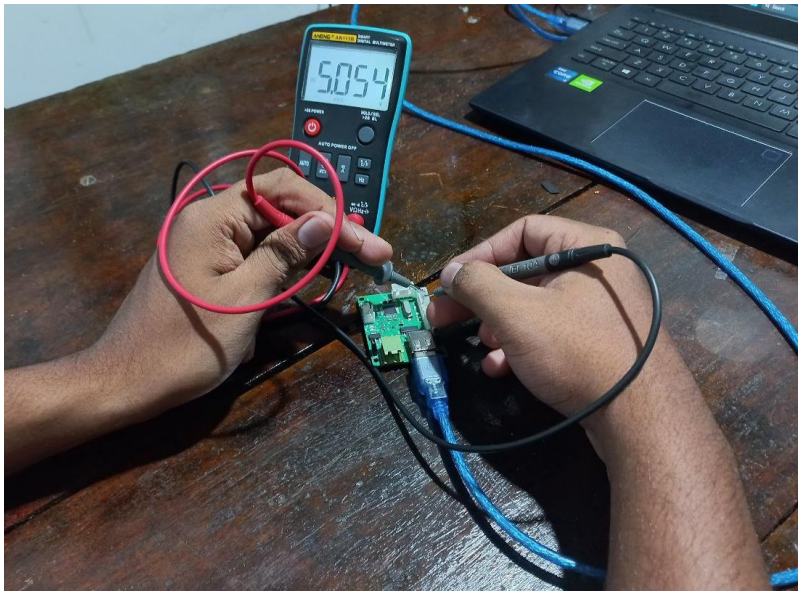


Figure 3: PCB testing - Output Voltage

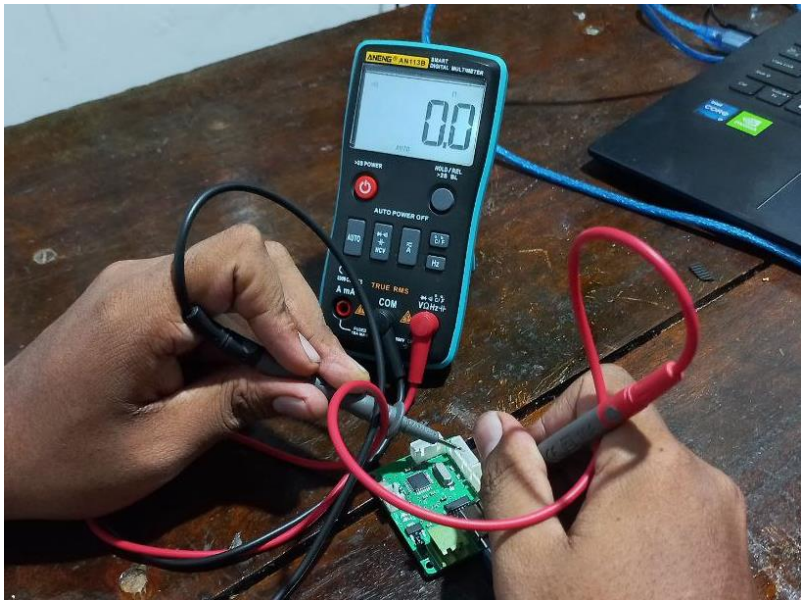


Figure 4: PCB testing -Continuity

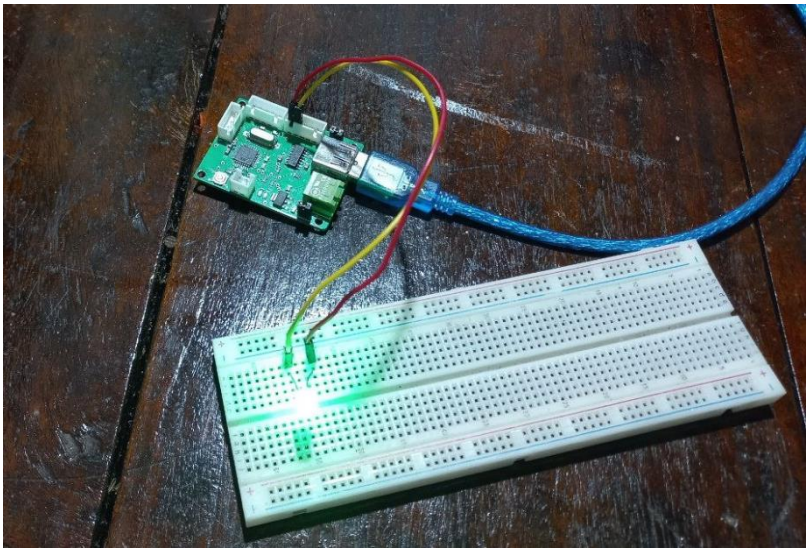


Figure 6: PCB testing - USB to TTL

1.2.2 Overall Circuit Design

Motors

We are using two NEMA 17 Stepper motors, one for conveyor belt and another one for pushing mechanism.

Continuous Input Current: 1.8 A

Figure 7: NEMA17 Motor

		IP 20		
NEMA		17S	17M	17L
Input Power, Nominal (±10%)	VDC	14–48	14–48	14–48
Auxiliary Input Power, Nominal (±10%)	VDC	6–24	6–24	6–24
Auxiliary Input Power, Maximum	W	1	1	1
Detent Torque	mNm	15	25	25
Thrust Load Limit	kg	0.28	0.36	0.6
Overhung Load Limit (from shaft end)	N	20	20	20
Rotor Inertia	g·cm2	57	82	123
Holding torque at continuous current	Nm	0.35	0.45	0.65
Holding torque at peak current	Nm	0.5	0.6	1.05
Continuous Output Current	A	1.8	1.8	1.8
Peak Output Current (application dependent)	A	3.5	3.5	3.5
Step Angle	deg	1.8	1.8	1.8
Magnetic Encoder, Resolution	ppr	4096	4096	4096
Circuit Loss	W	6	6	6
Weight	kg	0.37	0.44	0.59
Connection Hardware Screw Size/Torque	Nm	0.63	0.63	0.63

Figure 8: Data Sheet NEMA17

Motor Drivers

Two TB6600 motor drivers are used to drive the two NEMA17 Motors.

Figure 9: TB6600 Motor Driver

This is a two-phase stepper motor driver. It supports speed and direction control. This can set its micro step and output current with 6 DIP switch. There are 7 kinds of micro steps (1, 2 / A, 2 / B, 4, 8, 16, 32) and 8 kinds of current control (0.5A, 1A, 1.5A, 2A, 2.5A, 2.8A, 3.0A, 3.5A) in all. And all signal terminals adopt high-speed optocoupler isolation, enhancing its anti-high-frequency interference ability.

Electrical Specifications:

Input Current	0~5.0A
Output Current	0.5-4.0A
Power (MAX)	160W
Micro Step	1, 2/A, 2/B, 4, 8, 16, 32
Temperature	-10～45℃
Humidity	No Condensation
Weight	0.2 kg
Dimension	96*56*33 mm

Camera

To capture the placement of labels, an HD web camera is used. This is powered by the computer.

Figure 10: Web Camera

LED Light

For the accuracy of capturing of the web camera LED light is used.
Power Rating = 1.2W
Input Voltage = 12V
P = VI
Input Current = 0.01A

Figure 11: Led Light

Fan

Due to the heating of the power supply and the motor drivers, a 0.1A cooling fan was included for the enclosure.

Figure 12: Cooling Fan

Power Supply

Calculations:

Current for motors = 1.8A x 2 = 3.6A
Current for light = 0.01A
Current for fan = 0.1A
Current for PCB = 1 A (Approximately Maximum)
Total Current required = 4.71A
After adding 30% Safety Margin = 6.123 A

Since the lowest available power supply in the market greater than 6.123A is 10A, we have used a 10A power supply.

Figure 13: Power Supply

Limiting Switches
Two limiting switches are added to limit the ends of the pushing mechanism.

Figure 14: Limiting Switch

Main Switch
For the 230V AC current input, a 10A rocker switch is used.

Figure 15: Main Switch

Power Input
For power in 10A 230V 3-Pin Male Panel Mount AC Power Supply Socket is used.

Figure 16: Power Socket

1.2.3 Circuit Functionality

The first function of the circuit is to take input from a camera and send the information to a computer for processing. After processing the data and detecting if a milk packet is faulty, a signal is sent to our pushing mechanism. The output signal from the computer is then sent to our Atmel ATmega 328P chip, which is part of the PCB. We use a USB connector(J7) to facilitate communication between the PCB and the computer.

After processing the data in the Atmel chip, the actuator movement signal is sent from the PCB to the actuator motor driver. We use JST connectors (J6) for this purpose. Finally, the output signal from the motor driver goes to the motor, activating the pushing mechanism.

The design ensures efficient detection and removal of faulty milk packets by integrating camera input, computer processing, and actuator control through the PCB and motor driver. This setup provides a reliable and automated solution for quality control in milk packet processing.

1.3 Solidworks Design

1.3.1 Enclosure design

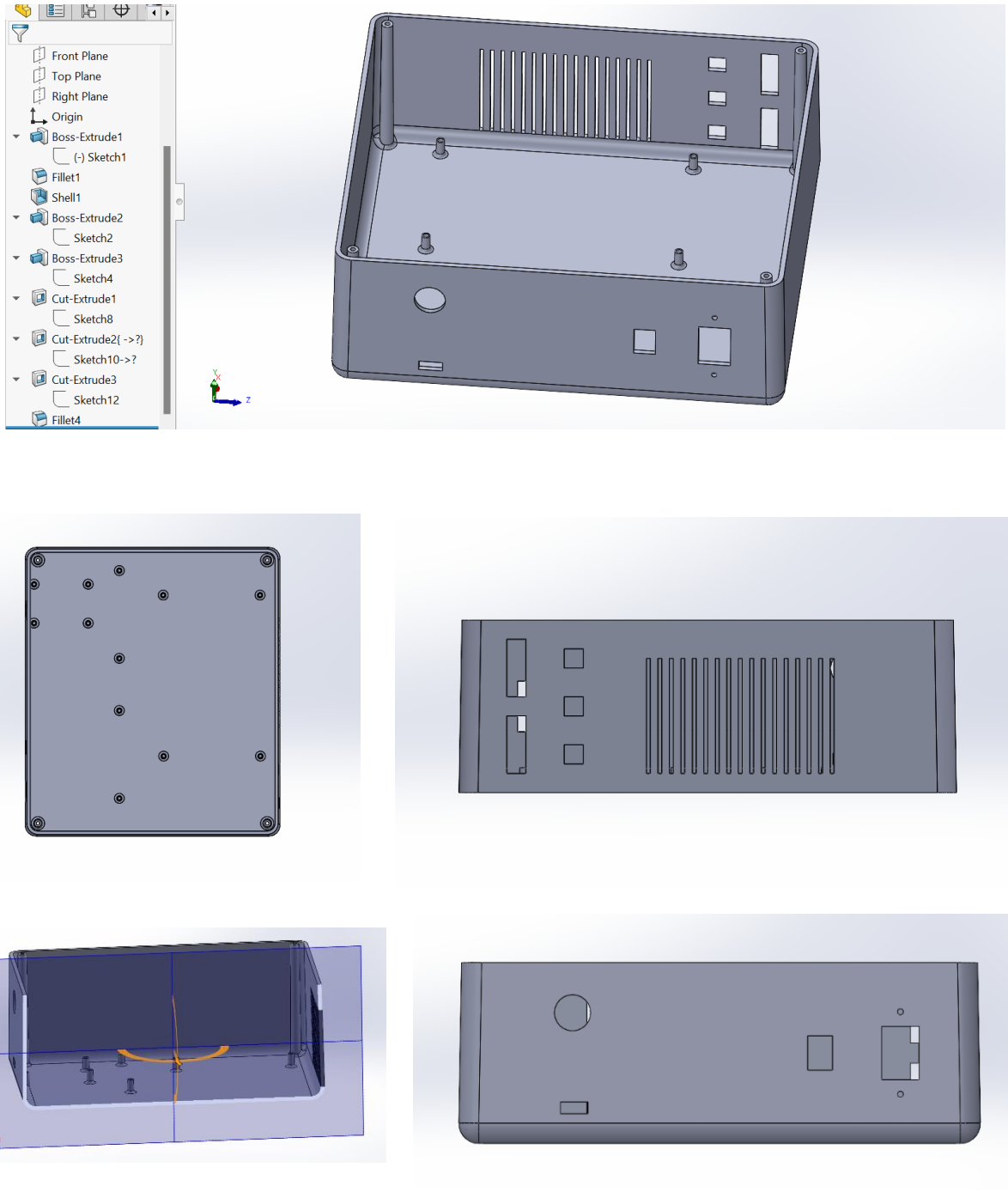
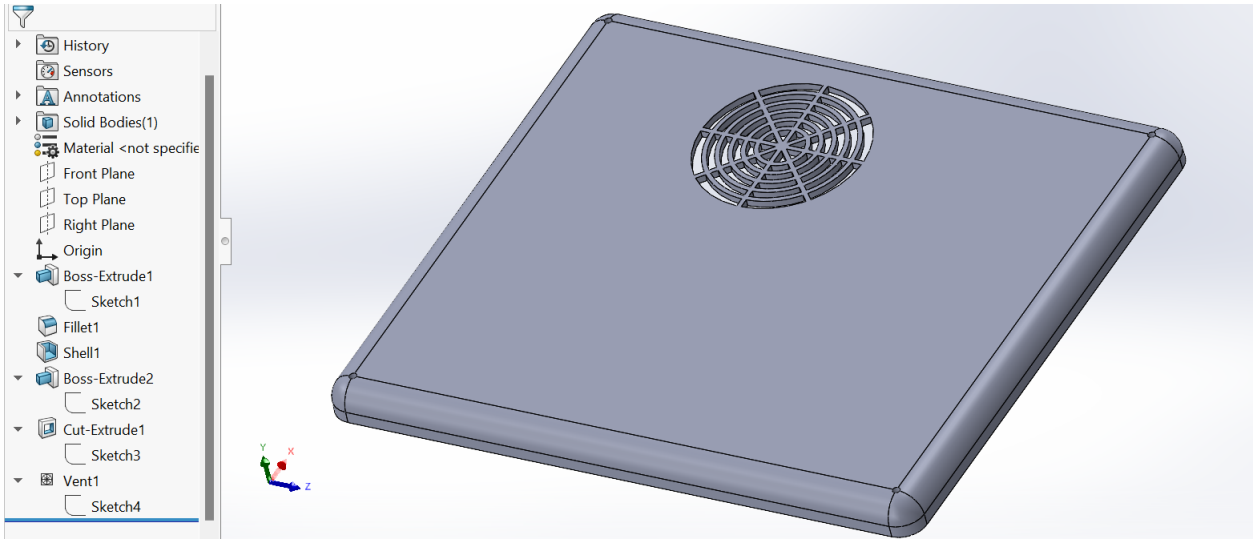
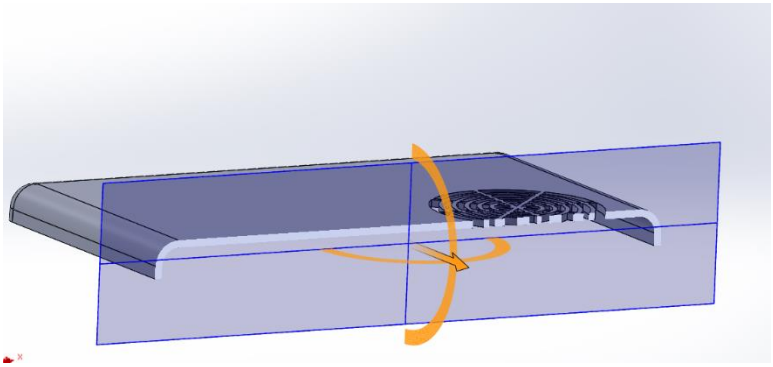
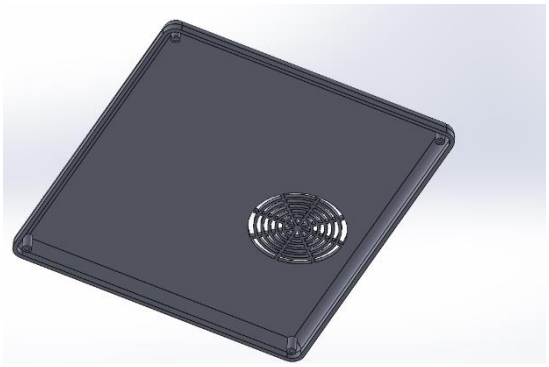


Figure 17: Solidworks design - bottom part of the enclosure







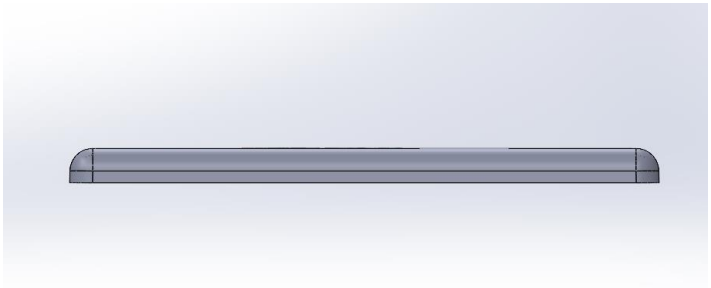
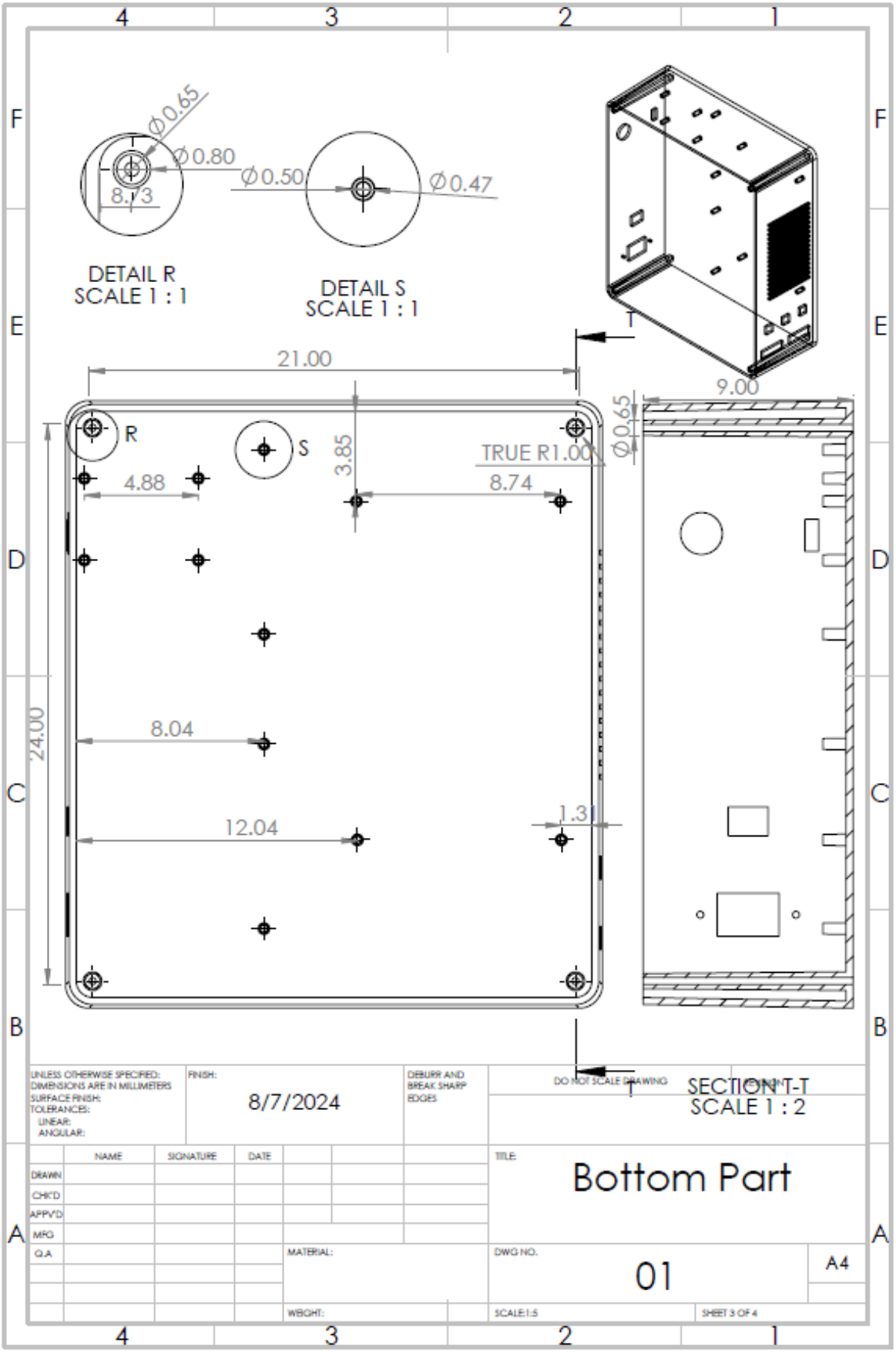
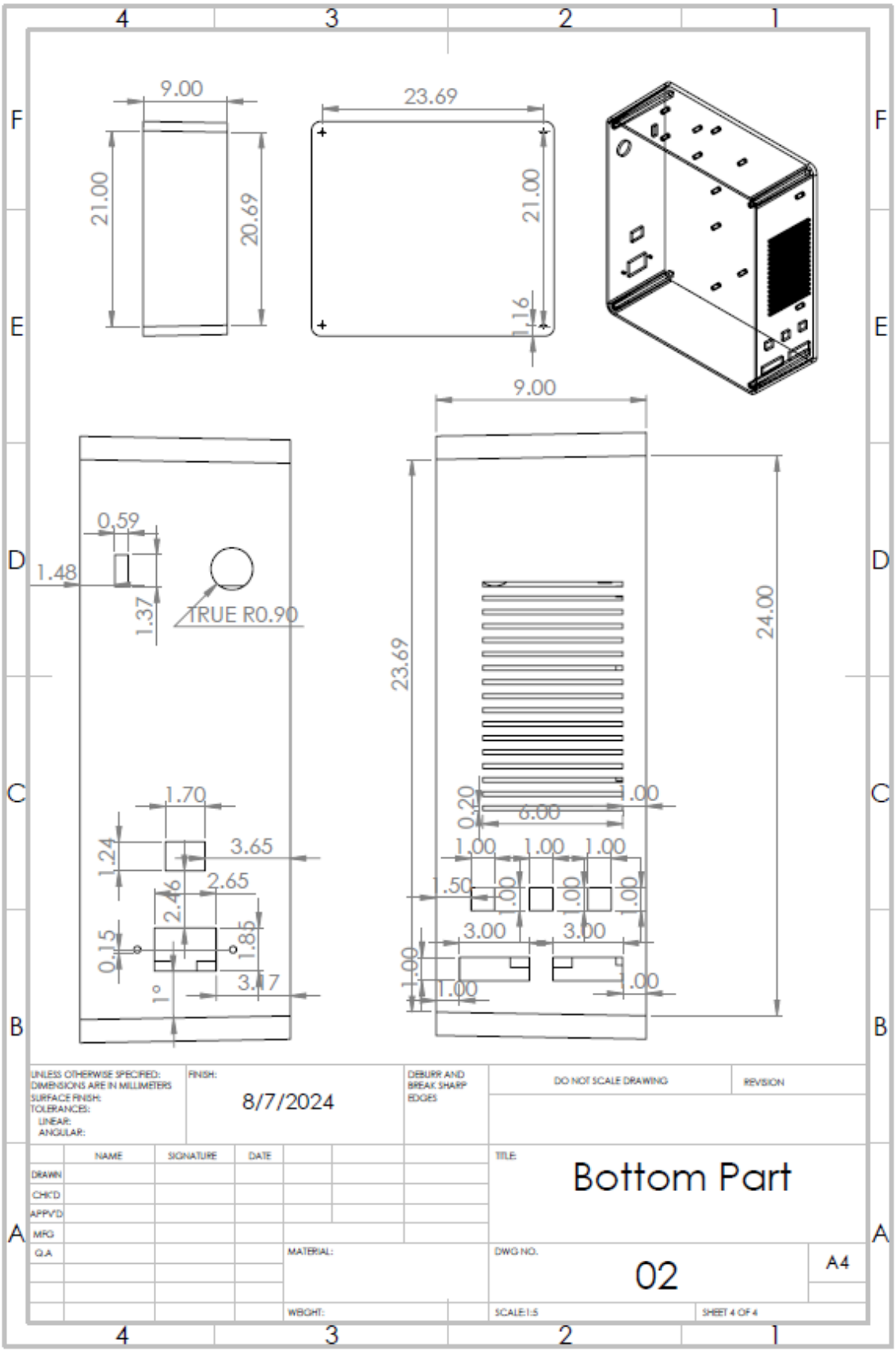
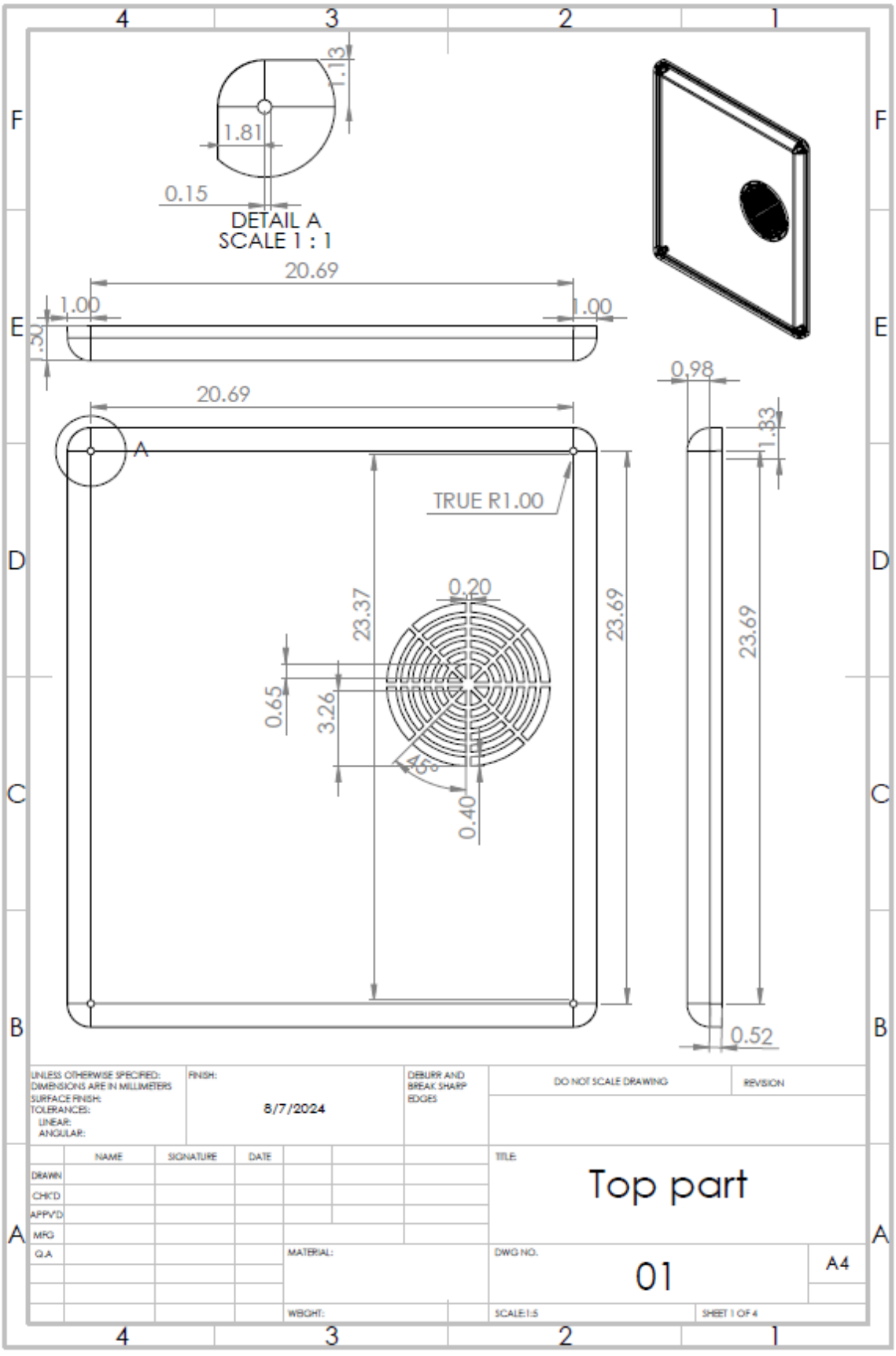


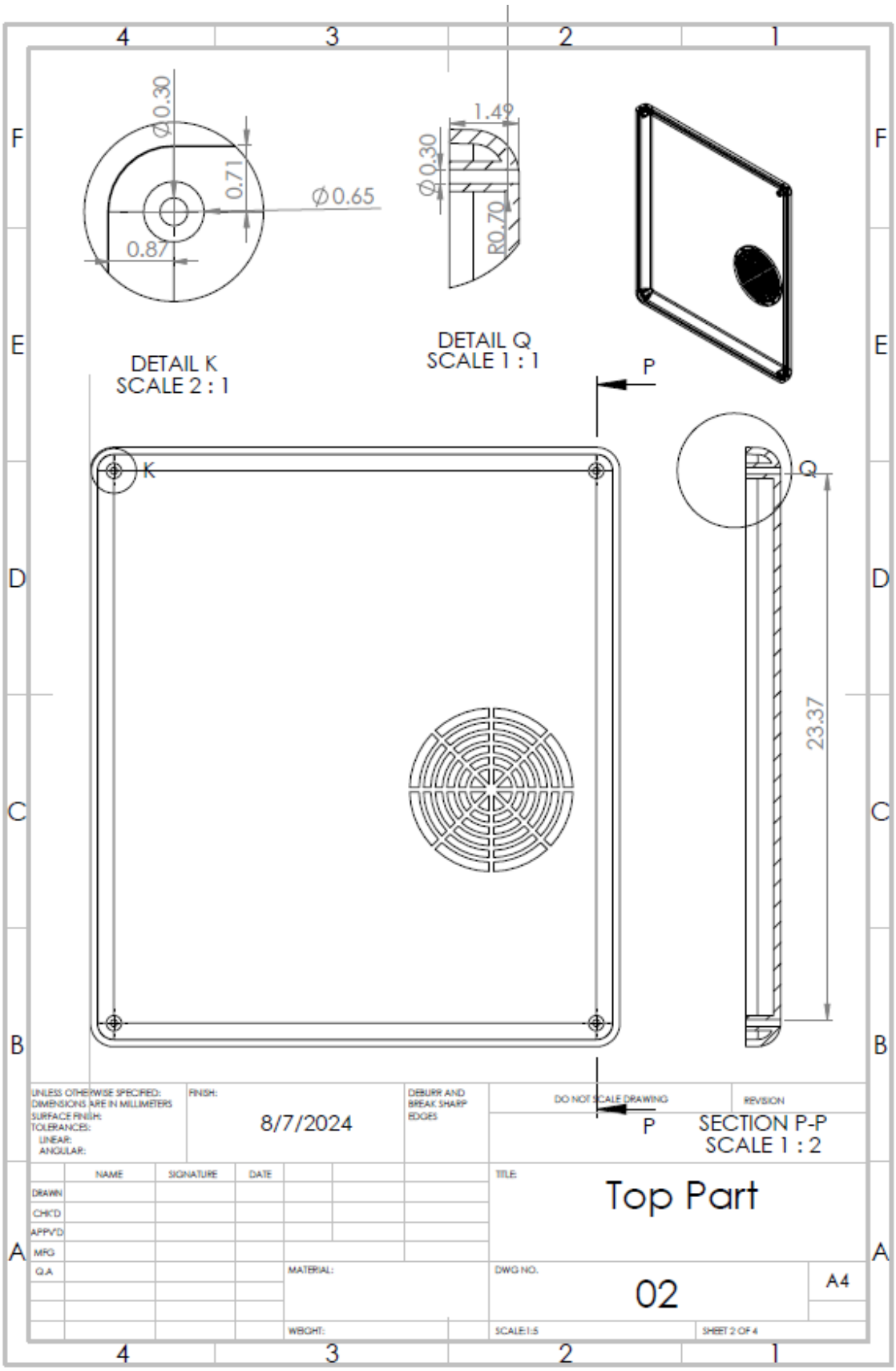
Figure 18: Solidworks design - top part of the enclosure

1.3.2. Design Drawings




9.00
23.69
21.00
20.69
21.00
1.16
0.59
1.48
1.37
TRUE R0.90
9.00
23.69
24.00
1.70
1.24
3.65
2.65
2.46
0.15
1.85
1°
3.17
0.20
6.00
1.00
1.00
1.00
1.00
1.50
1.00
1.00
1.00
3.00
3.00
1.00
1.00
1.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
8/7/2024
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Bottom Part
MATERIAL:
DWG NO.
02
A4
WEIGHT:
SCALE:1:5
SHEET 4 OF 4


1.13
1.81
0.15
DETAIL A
SCALE 1 : 1
20.69
1.00
1.00
1.50
20.69
A
TRUE R1.00
0.20
23.37
23.69
0.65
3.26
45°
0.40
0.98
1.83
23.69
0.52
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
8/7/2024
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
Top part
DWG NO.
01
A4
SCALE:1:5
SHEET 1 OF 4


4
3
2
1
F
E
D
C
B
A
Ø0.30
0.71
Ø0.65
0.87
DETAIL K
SCALE 2 : 1
1.49
Ø0.30
R0.70
DETAIL Q
SCALE 1 : 1
P
K
Q
23.37
P
SECTION P-P
SCALE 1 : 2
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
8/7/2024
DEBURR AND
BREAK SHARP
EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Top Part
MATERIAL:
DWG NO.
02
A4
WEIGHT:
SCALE:1:5
SHEET 2 OF 4

1.3.2. Enclosure Assembly

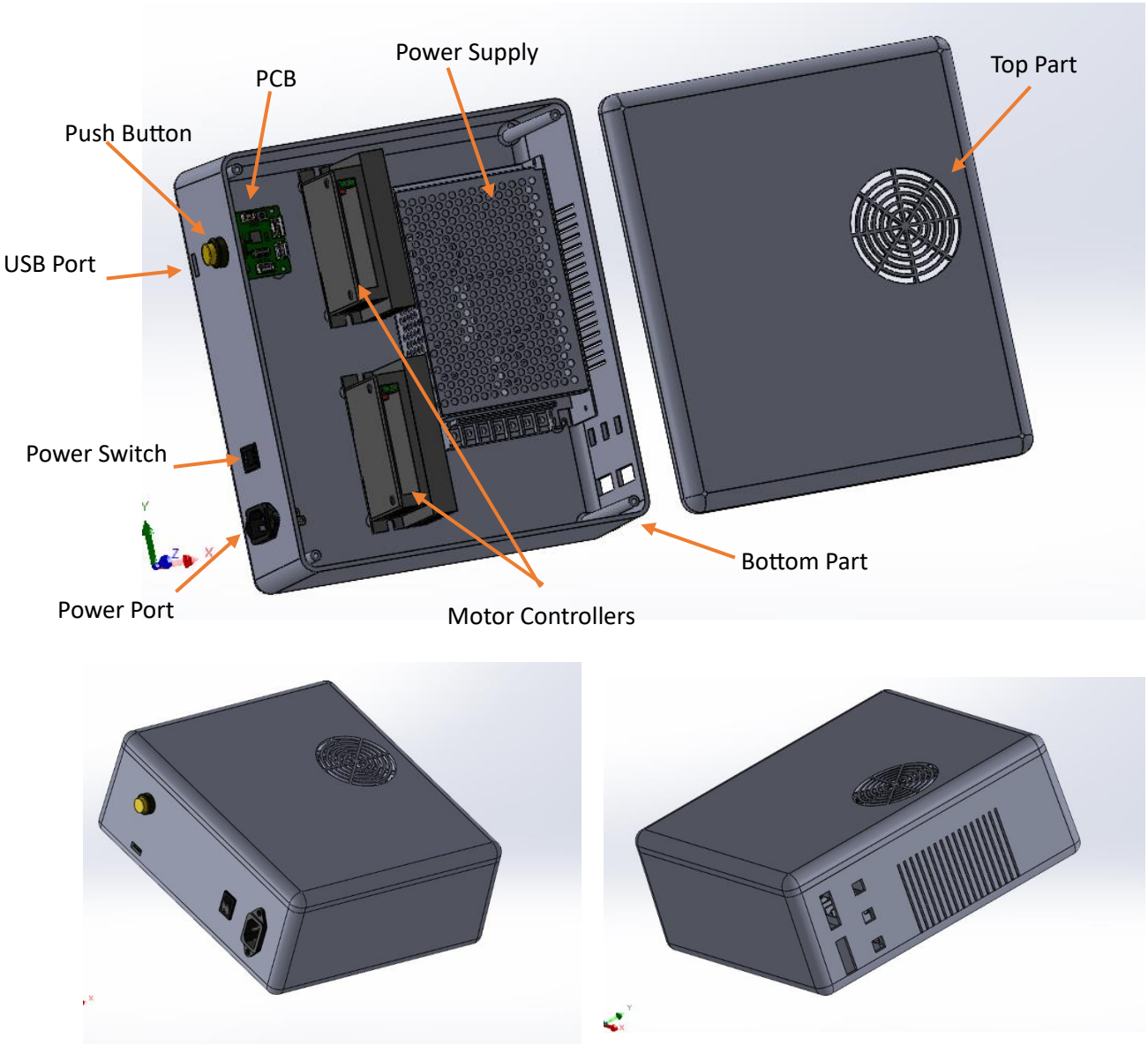


Figure 19: Enclosure Assembly

1.3.2. Mechanical Parts Design

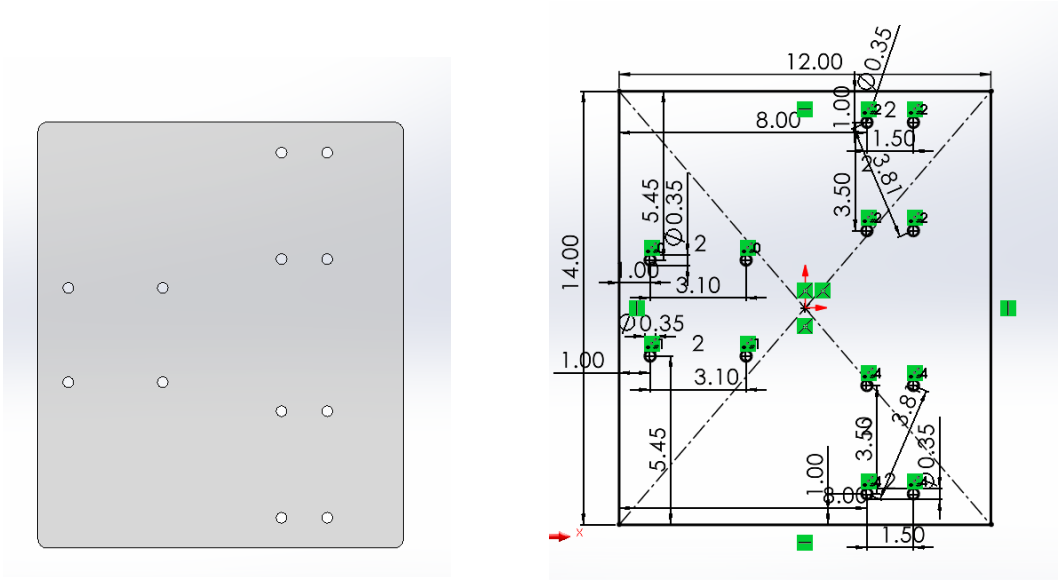


Figure 20: Fixed Plate 1

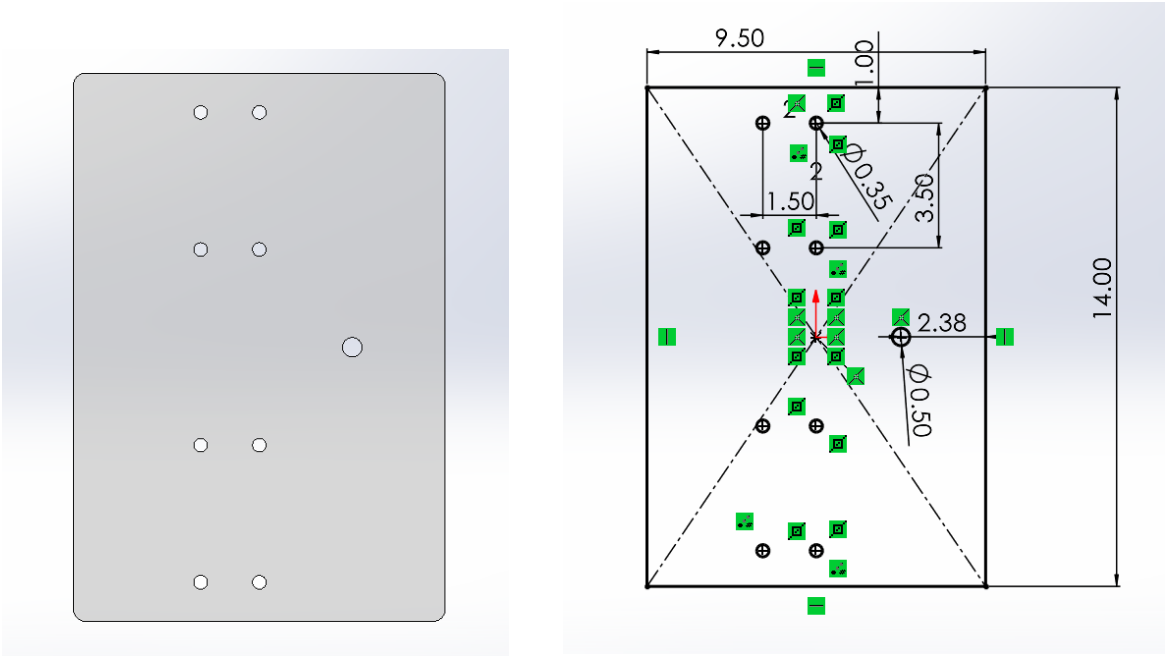


Figure 21: Fixed Plate 2

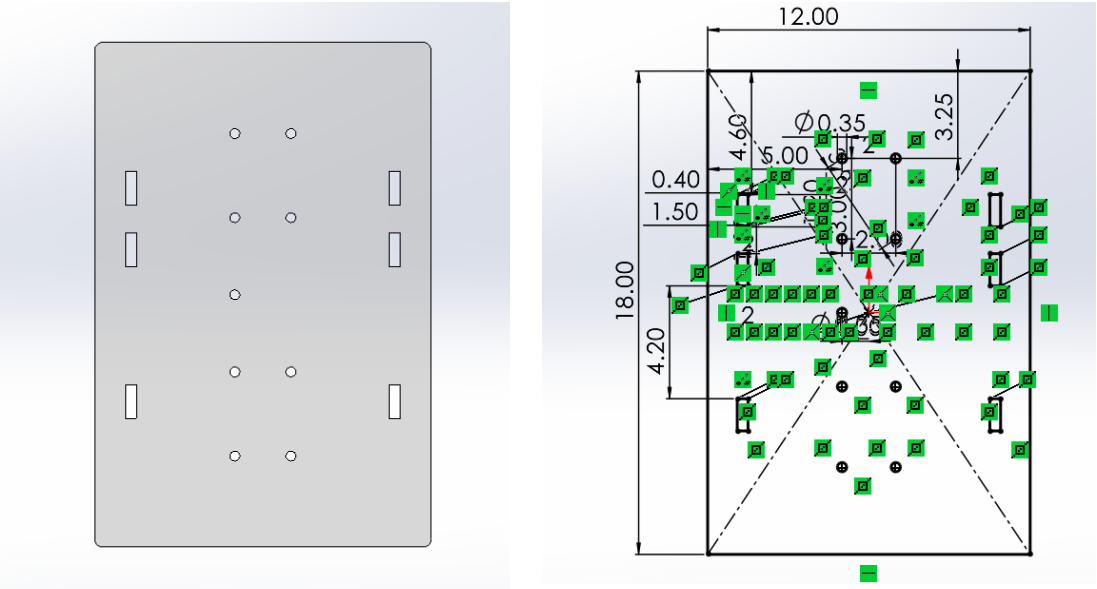


Figure 22: Moving Plate

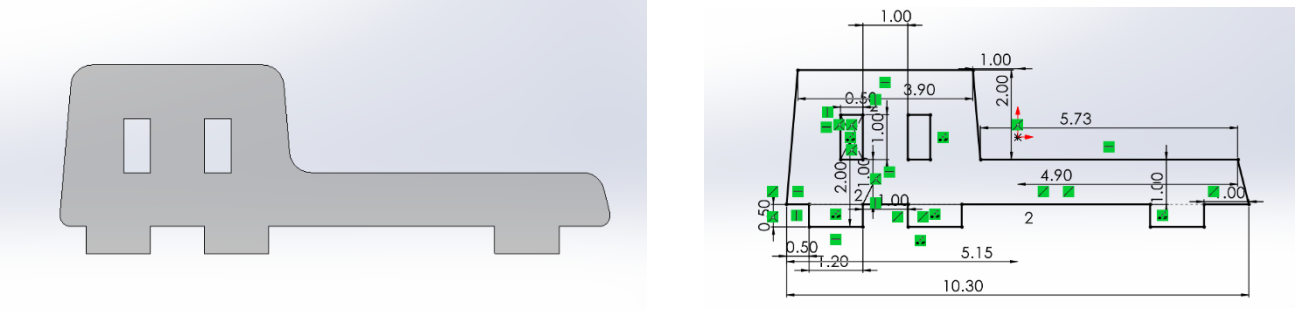


Figure 23: Supporting Plate

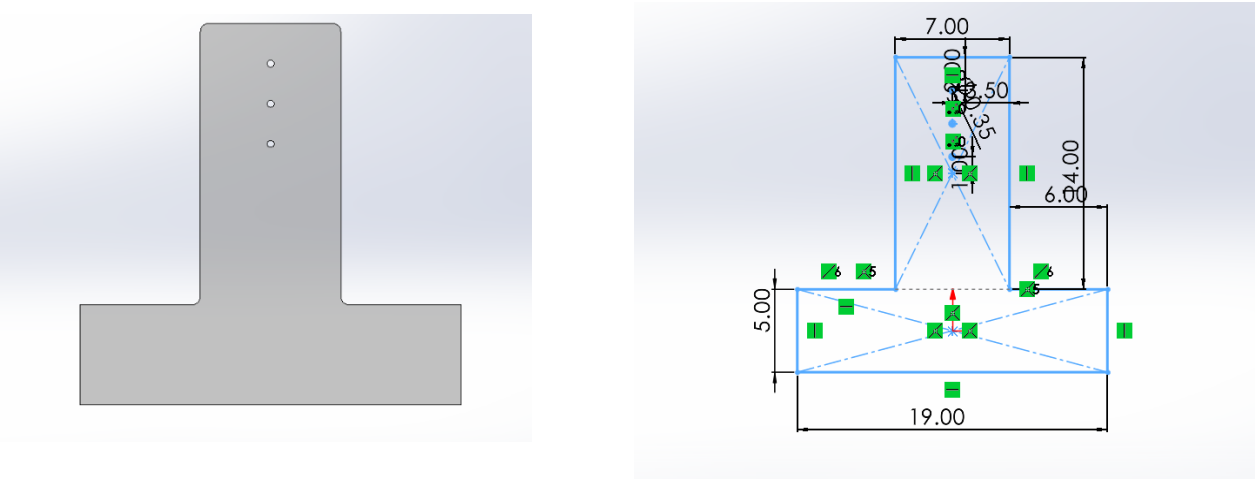


Figure 23: Pushing Plate

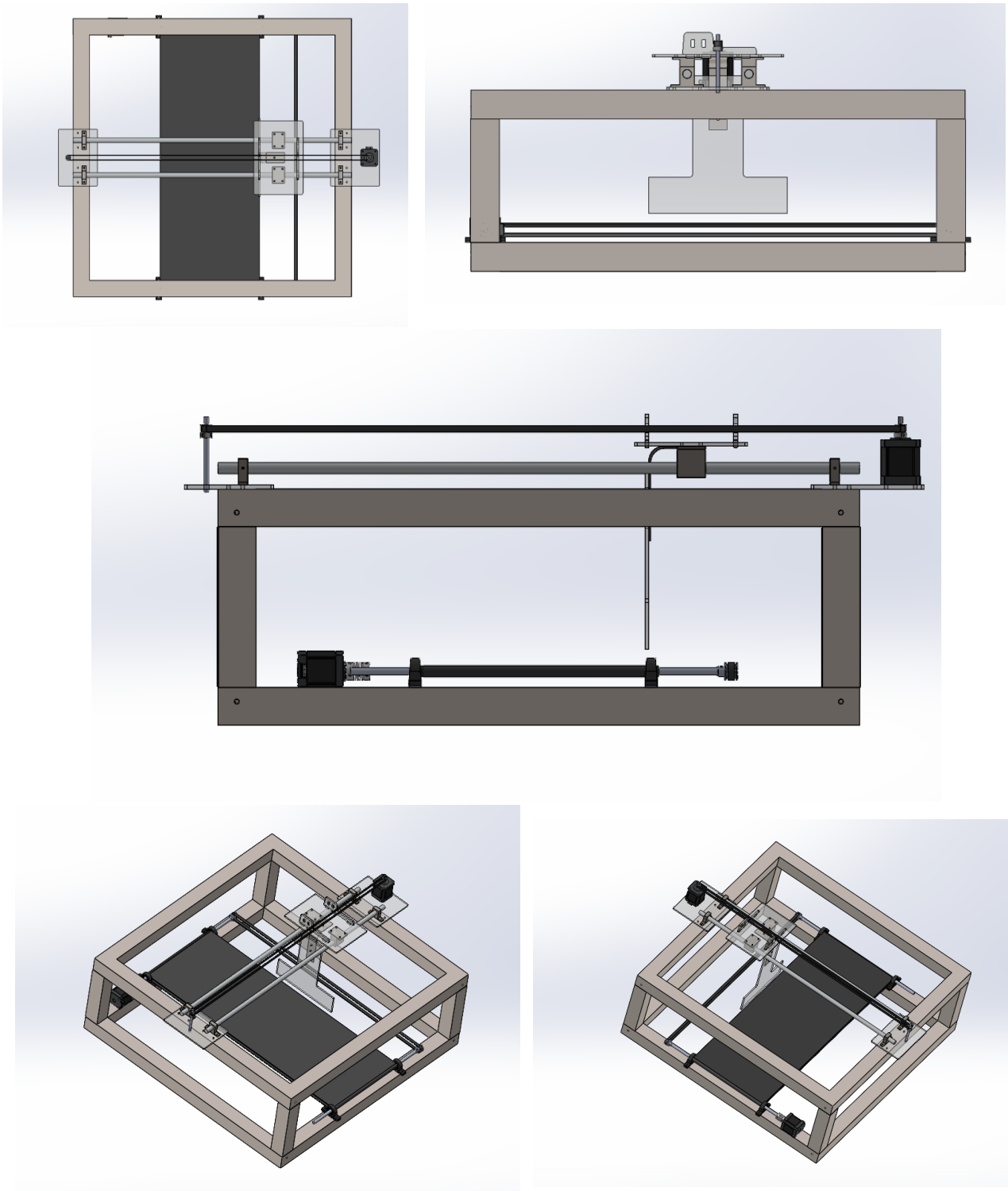


Figure 24: Assembly of the mechanical designs

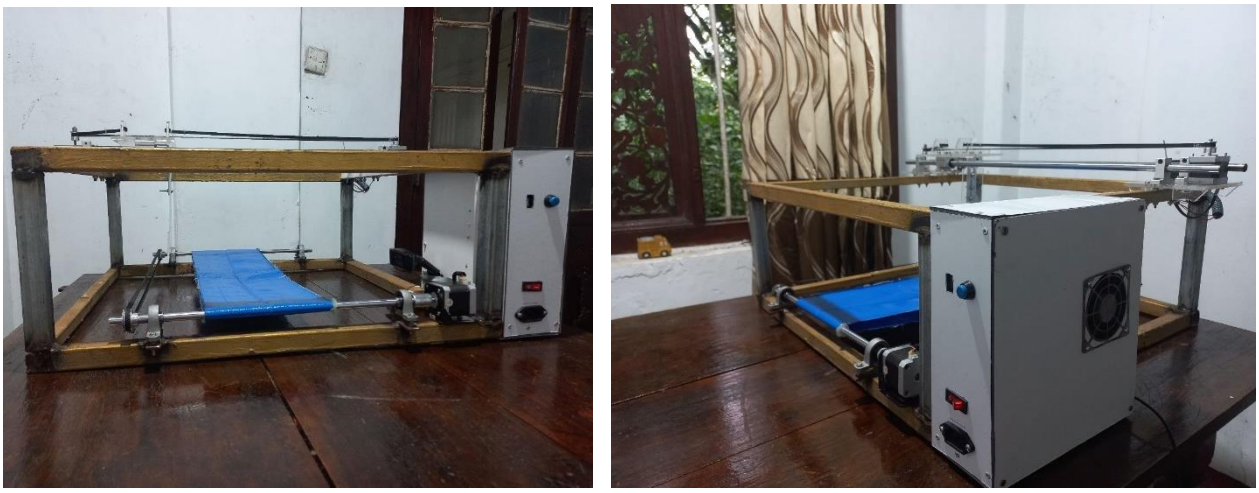
1.4 Implemented Mechanical design

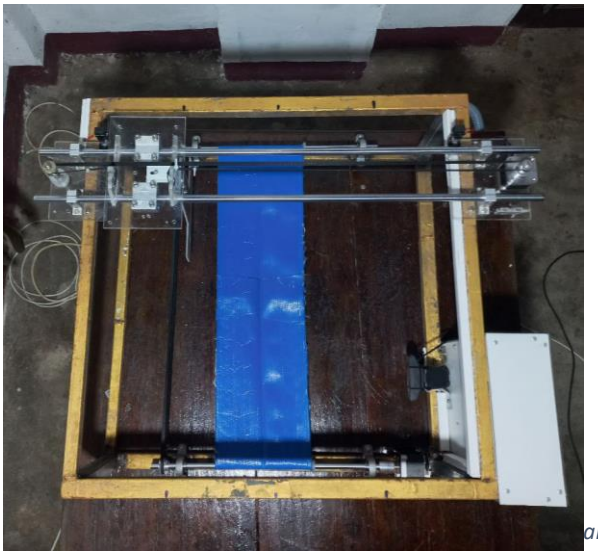
In the mechanical design of the components, there are three main components the main structure, the enclosure, the conveyor belt and the pushing actuator.

Main Structure

The main structure of the design consists of iron box bars, providing a durable frame for the entire system. The dimensions of the main structure are 60cm x 60cm x 20cm.

Figure 17: Initial Main Structure





al Mechanical structure

The conveyor belt roller mechanism uses 10mm pillow bearings and 10mm axles, driven by a NEMA 17 motor. The motor is mounted on the main structure using a NEMA 17 motor mount. The motor is connected to the axle using a coupling. To transfer energy to the other axle, we have used another belt system. The belt of the conveyor is made of canvas. This setup ensures smooth and reliable movement of the conveyor belt, allowing for precise positioning of the milk packets.

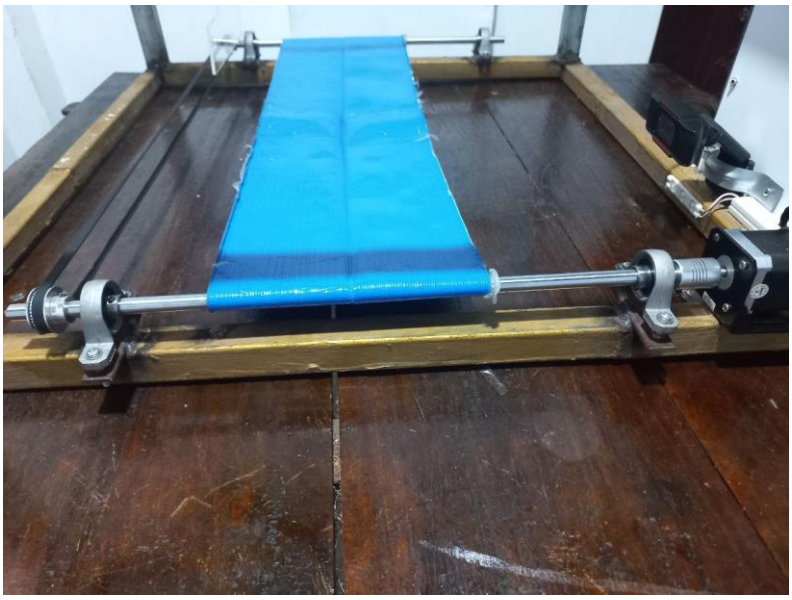
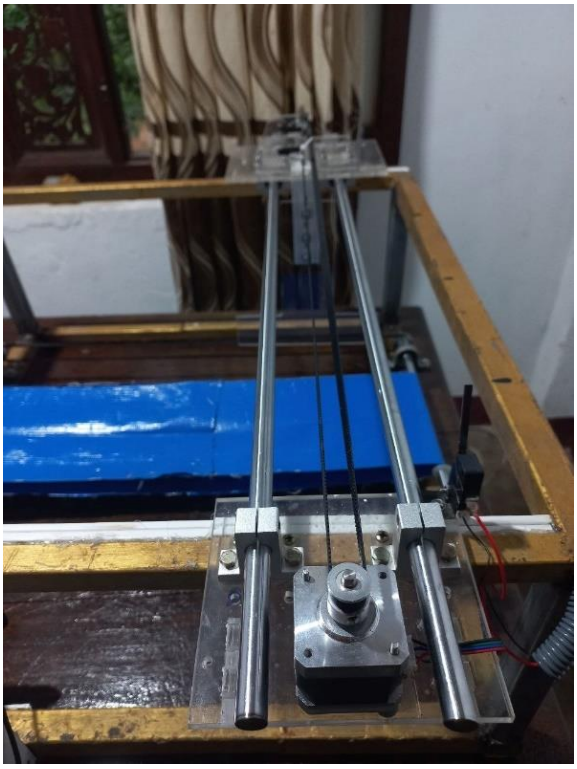


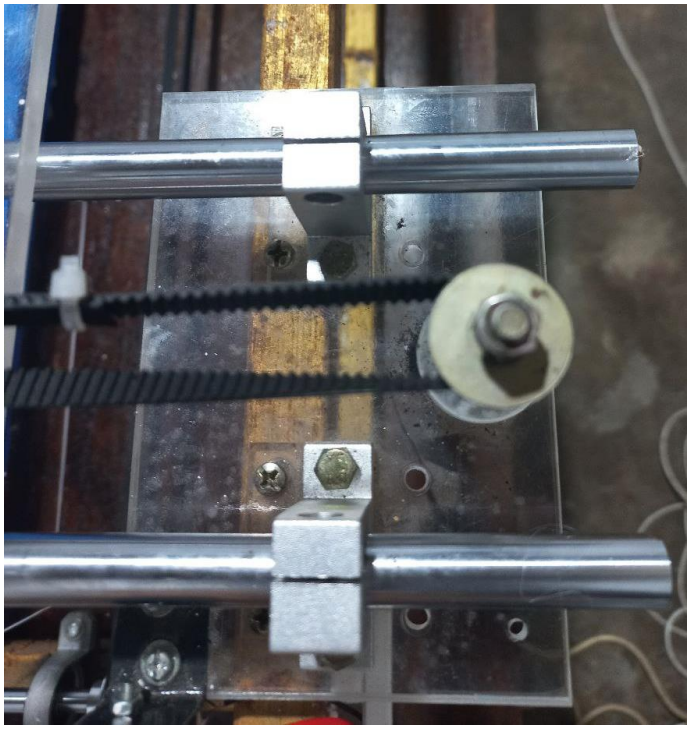
Figure 19: Conveyor Belt

Pushing Actuator

The pushing actuator is also controlled by a NEMA 17 motor, which is mounted to a custom-designed acrylic sheet. The pushing mechanism is like the mechanism used in CNC machines. We have used 4 shaft supports to hold 2 shafts, 2 supports per shaft. The mechanism operates via a belt. When a signal is received, the belt moves and activates the pushing mechanism. We have also used a custom-designed pushing plate, which provides the necessary force to remove faulty milk packets from the conveyor belt. The actuator mechanism is designed for a quick and accurate response, ensuring that faulty packets are promptly removed from the production line.

Figure 20: Pushing Plate





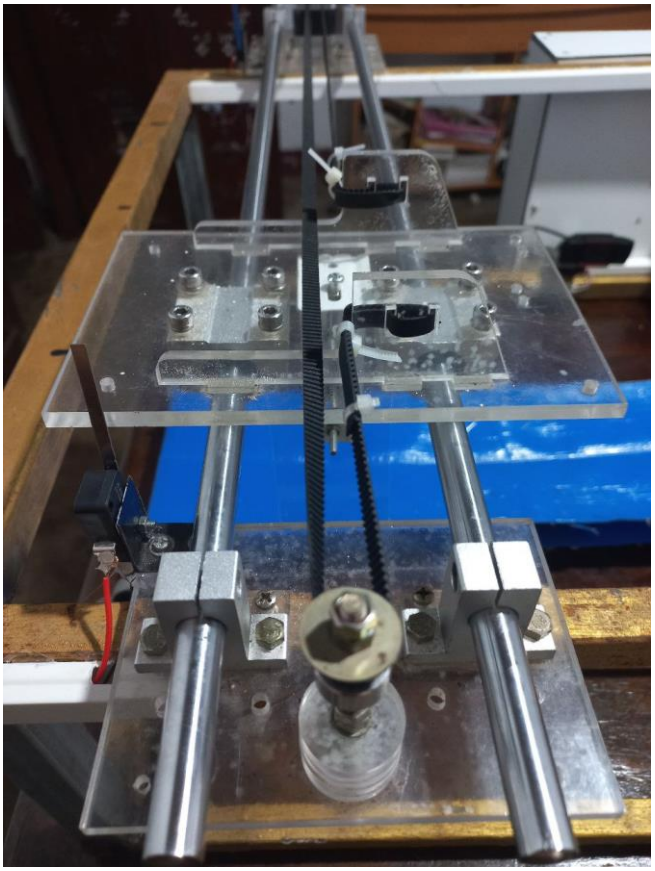


Figure 21: Pushing Mechanism

Enclosure

We have designed an enclosure that includes a fan to ensure proper ventilation. Inside the enclosure, there is the power supply unit, motor drivers and the PCB.

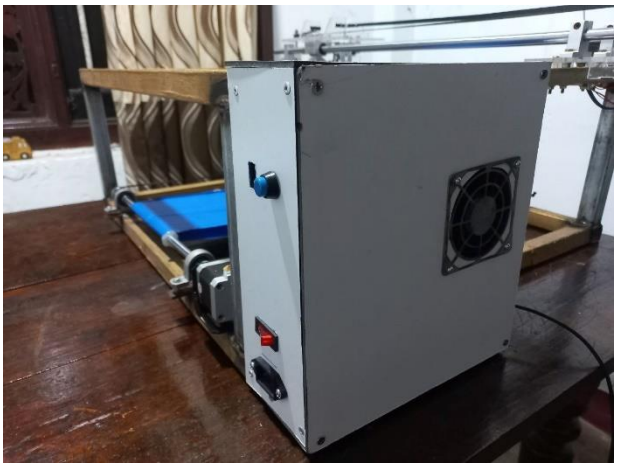


Figure 22: Enclosure

Figure 23: Component placement

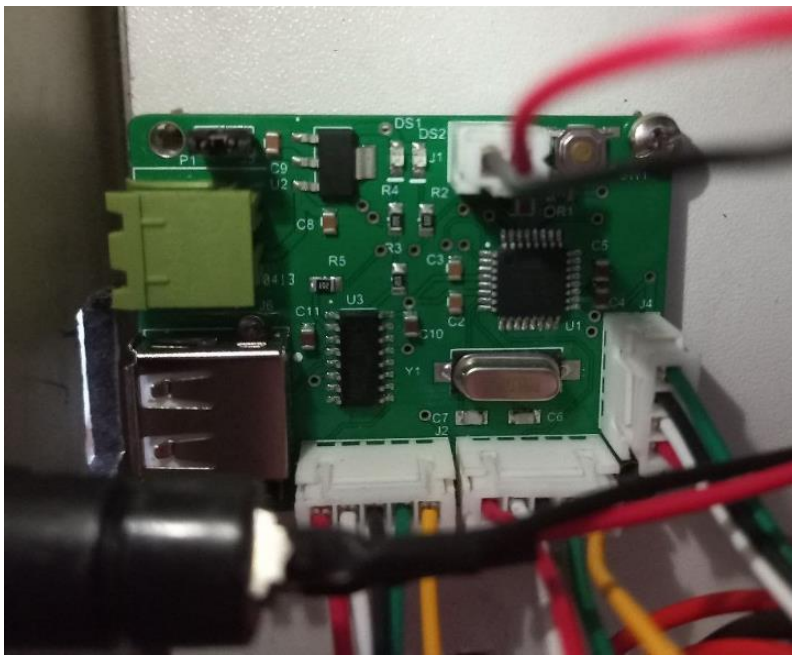


Figure 24: PCB placement

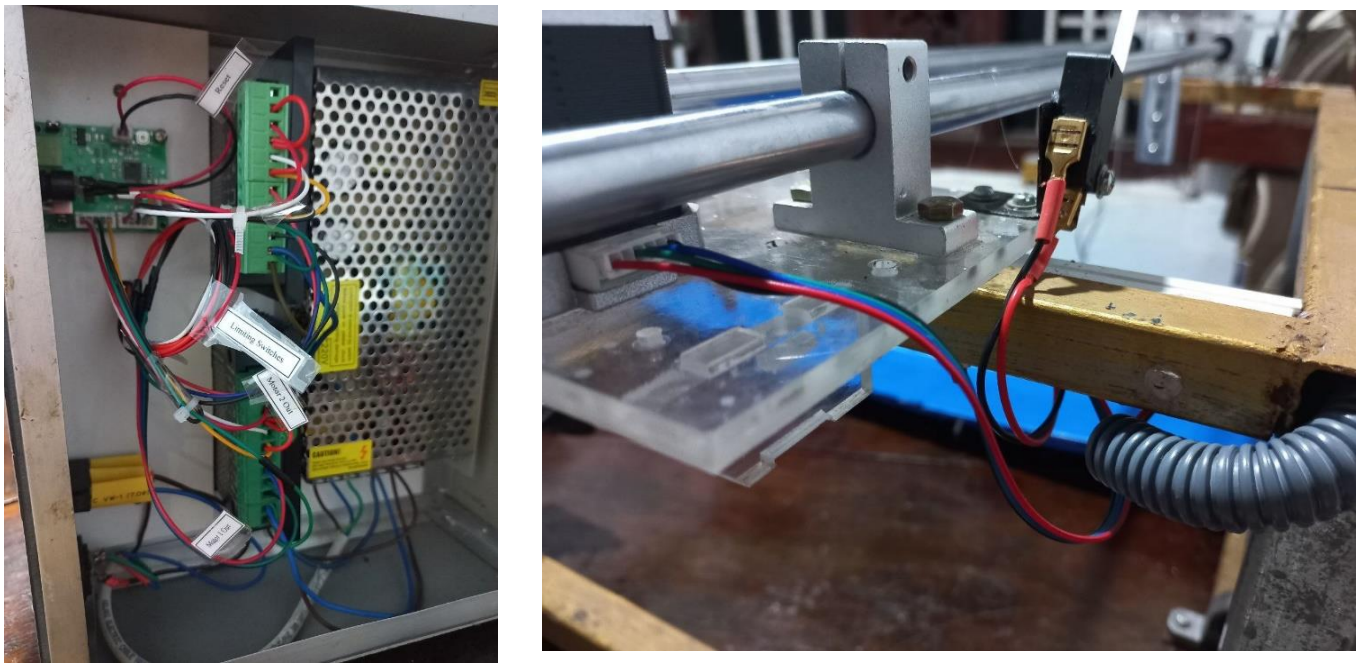


Figure 25: Cable Arrangement

2.Software Design

2.1. Introduction

In this software designing part we aim to deploy an object detection model to identify misprinted labels on milk packets, specifically focusing on labels that are not printed within the designated white space area. Ensuring the correct positioning of labels on milk packets is crucial for maintaining product quality, brand integrity, and compliance with regulatory standards. Misprinted labels can lead to misidentification of products, miscommunication of product information, and potential consumer safety issues.

To address this challenge, we are developing a robust object detection system that can automatically inspect milk packets for correct label placement in real-time

2.2. Background Research

At the initial stage of this project, we explored two potential solutions for detecting misprinted labels:

1) **Detect the misprinted labels using Hough lines and the Canny edges without using any deep learning techniques**:

This traditional computer vision approach involves using edge detection (Canny edge detector) and line detection (Hough Transform) techniques to identify the boundaries of the labels.

Challenges:

- Low Accuracy: The accuracy of this method is significantly lower because the results can vary depending on external conditions like lighting and camera angles.
- External Conditions: Variations in lighting, shadows, and camera positioning can drastically affect the edge detection and line detection results, leading to inconsistent performance.

2) **Detect the misprinted labels by developing an object detection model using deep learning techniques**:

This approach involves training a deep learning model on a custom dataset of milk packet images to detect misprinted labels.

Advantages:

- High Accuracy: Deep learning models can be trained to handle varying conditions by including diverse examples in the training dataset. This makes them more robust to changes in lighting, camera angles, and other external factors.
- Industry Standard: Most similar projects in the industry are using deep learning techniques due to their superior performance in accuracy and robustness.

Challenges:

- Data Collection: Creating a large dataset manually is time-consuming and requires significant effort. The dataset needs to include images of correctly printed labels and misprinted labels under various conditions.

- Training Time: Training deep learning models can be computationally intensive and time-consuming, requiring powerful hardware (GPUs) and considerable time for experimentation and tuning.

Considering the importance of accuracy in detecting misprinted labels and the industry's trend towards using deep learning techniques, we decided to proceed with the second solution. Despite the challenges in data collection and training, the potential for achieving high accuracy and robustness makes deep learning the preferred approach for this project.

2.3. Project Workflow and Implementation Plan

Since we opted for a deep learning approach, we selected YOLOv8 as the model and decided to implement image segmentation, an advanced form of object detection. Our choice of YOLOv8 was driven by its balance of speed and accuracy, making it suitable for real-time applications like detecting misprinted labels on milk packets.

We chose Google Colab for the training and evaluation phases due to its accessibility to free GPU resources, seamless integration with deep learning libraries and support for collaborative work. For deploying the model into real-time predictions, we planned the workflow as follows:

1) **Dataset Preparation**: Gathered a comprehensive dataset of milk packet images, including examples of correctly printed labels and misprinted labels outside the white space area.
2) **Data Annotation**: Manually annotated the dataset to indicate the location of labels and their misprints.
3) **Training Phase**: Utilized Google Colab to train the YOLOv8 model on the annotated dataset, leveraging its GPU capabilities for faster training.
4) **Evaluation Phase**: Evaluated the trained model's performance using metrics to ensure it meets the required accuracy for detecting misprinted labels.
5) **Real-Time Object Detection**: Deployed the trained model using Jupyter Lab in a local environment to perform real-time object detection on incoming video streams or images of milk packets.
6) **Integration with Ardunio for Automated Response**: Integrated the detection system with a microcontroller to send signals upon detecting misprinted labels, enabling automated quality control in production lines.

2.4. Dataset Preparation

2.4.1. Preparing the Initial Data

Dataset preparation is a key challenge in this project because, to achieve sufficient accuracy with the YOLOv8 model, we need at least 1000+ images per class (for CORRECT and WRONG labels). Initially, we prepared the data with 10 images for each class:

- Correct Labels: We simply took 10 photos of milk packets with correctly printed labels.
- Wrong Labels: Finding milk packets with misprinted labels was difficult. Therefore, we took 10 images of milk packets and edited these images to simulate misprinted labels using ***Photoshop*** software.

Figure 263: Correct Label

Figure 27: Wrong Label

2.4.2. Data Augmentation

To increase the number of images from our initial set to 1000+ per class, we utilized data augmentation. Data augmentation is a technique used to artificially expand the size of a dataset by creating modified versions of images in the dataset. This can include transformations such as rotations, flips, translations, and changes in brightness or contrast, among others.

Here is the process we used:

1) ***Initial data*** Folder: We created a folder containing the initial 10 images for each class.
2) ***preview*** Folder: We created a new folder called ***preview*** in the same directory as the ***Initial data*** folder.
3) Augmentation Process: Using the following process, we generated 2020 images (101 images from each of the 20 initial images) and saved them in the ***preview*** folder.

Augmentation Process

1. Installing & Importing Required Libraries

Before running the data augmentation script, it is essential to install all the necessary libraries. The following commands can be used to install the required libraries:

```
!pip install numpy tensorflow pillow
```

Following code imports necessary libraries:

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from PIL import Image
```

- `os` for file and directory operations.
- `numpy` for numerical operations.
- `ImageDataGenerator` and `load_img` from `tensorflow.keras.preprocessing.image` for image augmentation and loading.
- `Image` from `PIL` for image manipulations.

2. Custom Functions for Image Conversion

```
def custom_img_to_array(img, data_format='channels_last', dtype='float32'):
    """Converts a PIL Image instance to a Numpy array."""
    x = np.asarray(img, dtype=dtype)
    if data_format == 'channels_first':
        if x.ndim == 3:
            x = x.transpose(2, 0, 1)
        elif x.ndim == 4:
            x = x.transpose(0, 3, 1, 2)
    return x
```

```
def custom_array_to_img(x, data_format='channels_last', scale=True):
    """Converts a Numpy array to a PIL Image instance."""
    if data_format == 'channels_first':
        x = x.transpose(1, 2, 0)
    if scale:
        x = (x - x.min()) / (x.max() - x.min()) * 255
    return Image.fromarray(x.astype('uint8'))
```

These functions handle the conversion between PIL images and Numpy arrays:

- `custom_img_to_array`: Converts a PIL image to a Numpy array, with an option for different data formats.
- `custom_array_to_img`: Converts a Numpy array back to a PIL image, with optional scaling.

3. Ensuring the Preview Directory Exists

```
# Ensure the preview directory exists
save_dir = 'preview'
if not os.path.exists(save_dir):
    os.makedirs(save_dir)
```

This section ensures that the directory where augmented images will be saved (`preview`) exists. If not, it creates the directory.

4. Initializing the ImageDataGenerator

```python
# Initialize the ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

The `ImageDataGenerator` is configured with various augmentation parameters:

- `rotation_range`: Randomly rotate images up to 20 degrees.
- `width_shift_range` and `height_shift_range`: Randomly shift images horizontally and vertically by up to 10% of the total width/height.
- `shear_range`: Apply random shear transformations.
- `zoom_range`: Apply random zoom within a small range (±5%).
- `horizontal_flip`: Randomly flip images horizontally.
- `fill_mode`: Specifies how to fill points outside the boundaries of the input when the image is transformed (here, using 'nearest').

5. Path to the Dataset Directory

```python
# Path to the dataset directory
dataset_dir = 'Initial data'
```

This line sets the path to the directory containing the initial set of images.

6. Iterating Through and Augmenting Images

```python
# Iterate through all images in the dataset directory
for filename in os.listdir(dataset_dir):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
        img_path = os.path.join(dataset_dir, filename)
        img = load_img(img_path)  # This is a PIL image
        x = custom_img_to_array(img)
        x = x.reshape((1,) + x.shape)

        # Generate and save batches of randomly transformed images manually
        i = 0
        for batch in datagen.flow(x, batch_size=1):
            transformed_img = custom_array_to_img(batch[0])
            save_name = f"{os.path.splitext(filename)[0]}_{i}.jpg"
            fname = os.path.join(save_dir, save_name)
            transformed_img.save(fname)
            i += 1
            if i > 100:
                break  # Otherwise, the generator would loop indefinitely
```

This section processes and augments each image in the dataset directory:

- **File Iteration**: Iterates through all image files in the dataset directory.
- **Image Loading and Conversion**: Loads each image and converts it to a Numpy array.
- **Data Augmentation Loop**: Uses `datagen.flow` to generate batches of augmented images from each original image. For each batch:
    - Converts the augmented Numpy array back to a PIL image.
    - Constructs a filename for the augmented image.
    - Saves the augmented image in the `preview` directory.
    - Limits the number of augmented images per original image to 100 to prevent indefinite looping.

2.5. Data Annotation

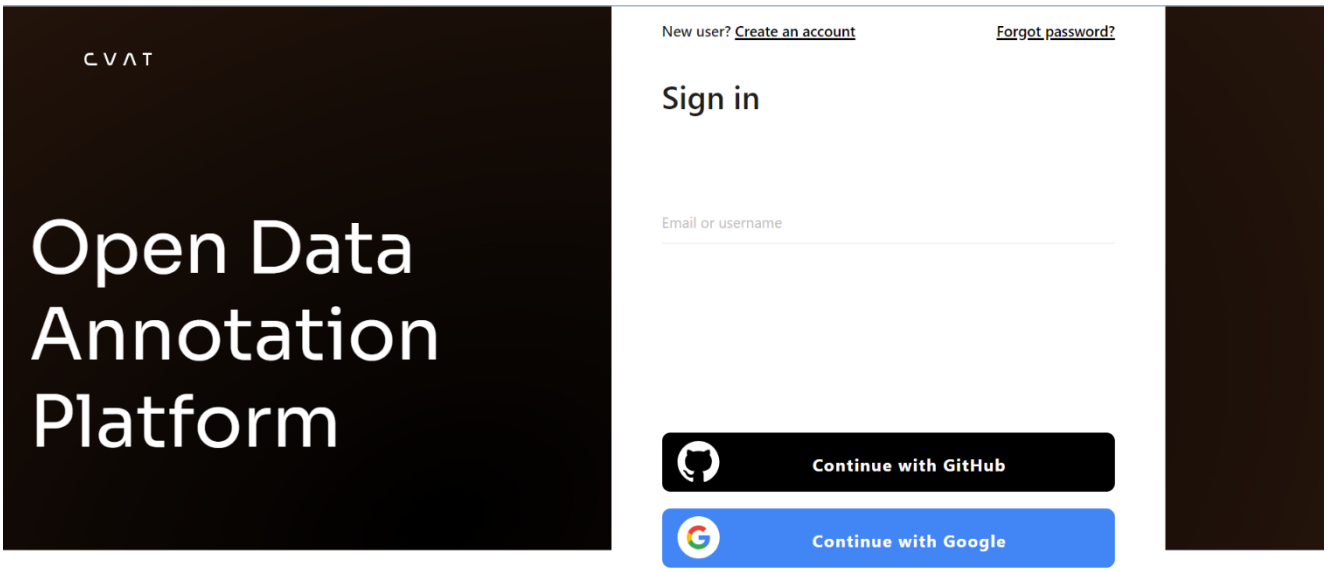
After the data augmentation process, a new folder called images was created, and all images from both the preview and initial data folders were copied into it. This resulted in a dataset of 2040 images, with 1020 images per class. The next step is to create segmentation masks for the dataset and convert them into labels.

2.5.1 Creating the Segmentation Masks

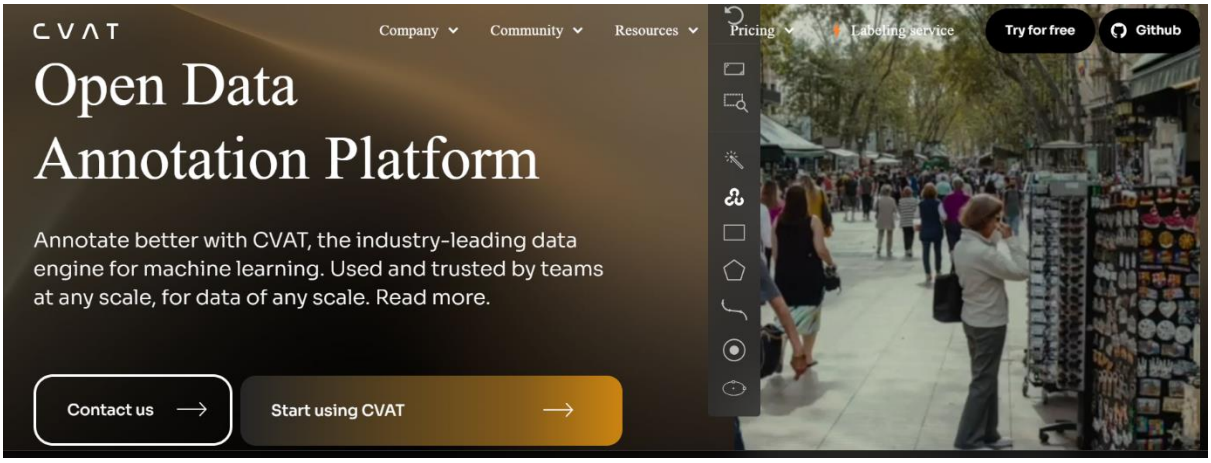
For creating segmentation masks for our custom dataset, we used an open data annotation platform called CVAT (https://www.cvat.ai/).

Process Followed:

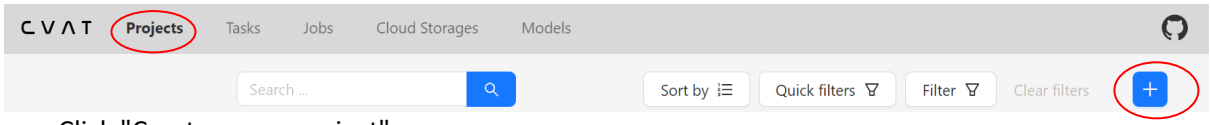
1. **Create an Account**:
    - Sign up using Google or GitHub.



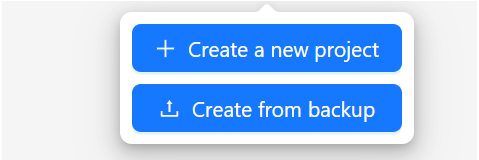
2. **Start Using CVAT**:
   - Click "Start using CVAT" after logging into your account.



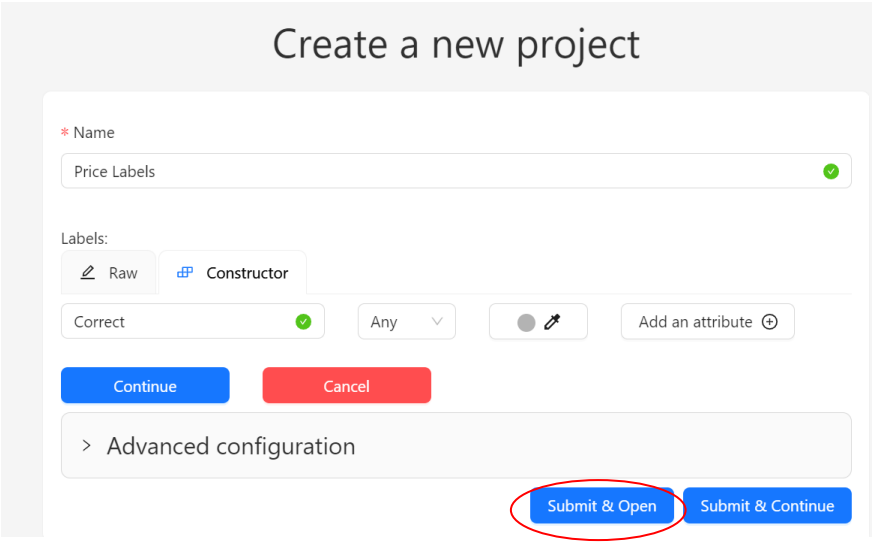
3. **Create a New Project**:
   - Go to the Projects section and click the "+" button.



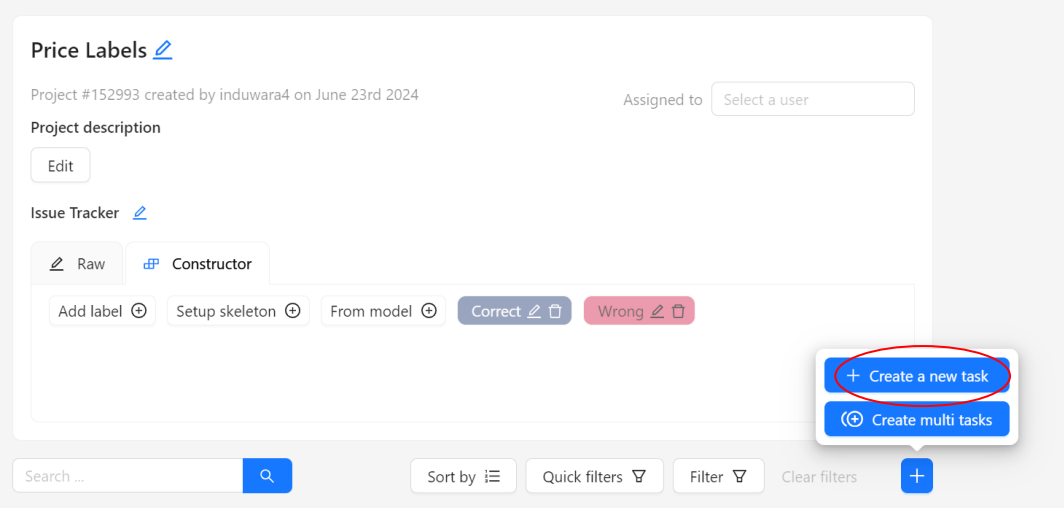
   - Click "Create a new project".



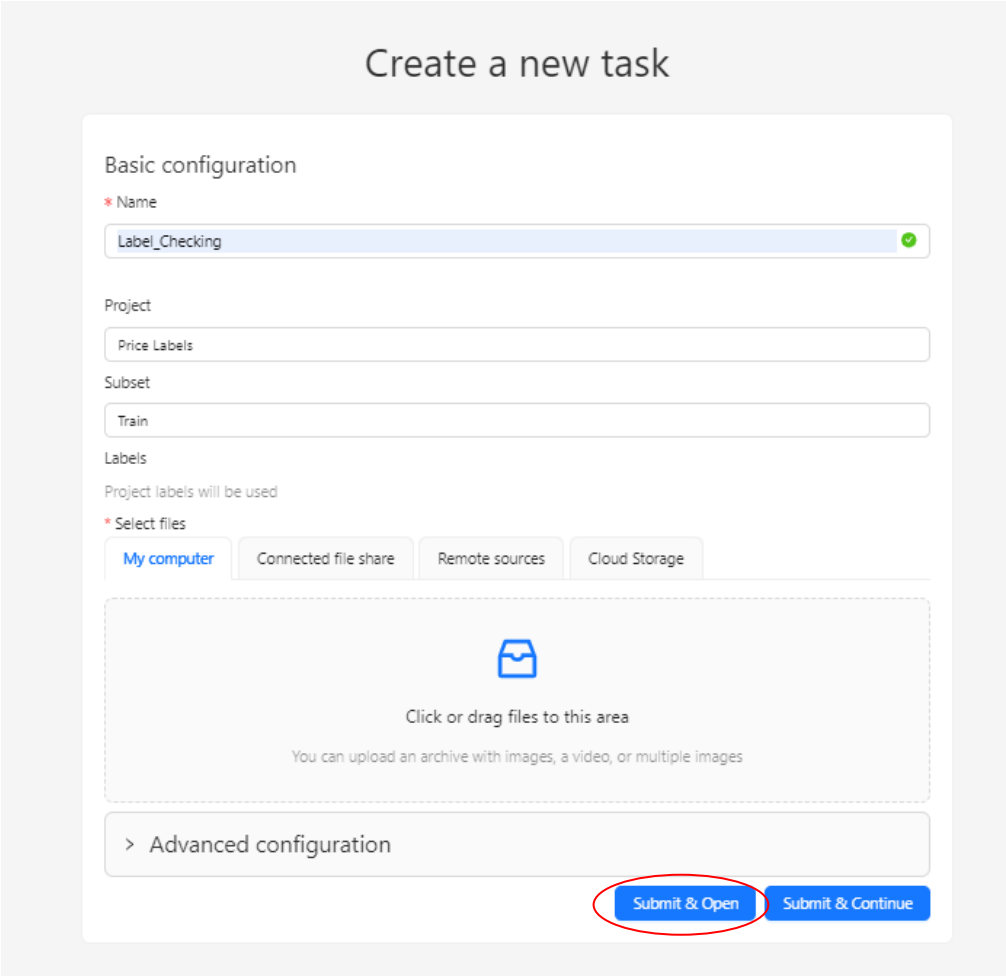
   - Name the project (e.g., "Price Labels").
   - Add labels:
     - Click "Add label" in the Labels field, enter "Correct" as the label name, and click "Continue".
     - Enter "Wrong" as the label name and click "Continue".
     - Click "Cancel" and then "Submit and open".



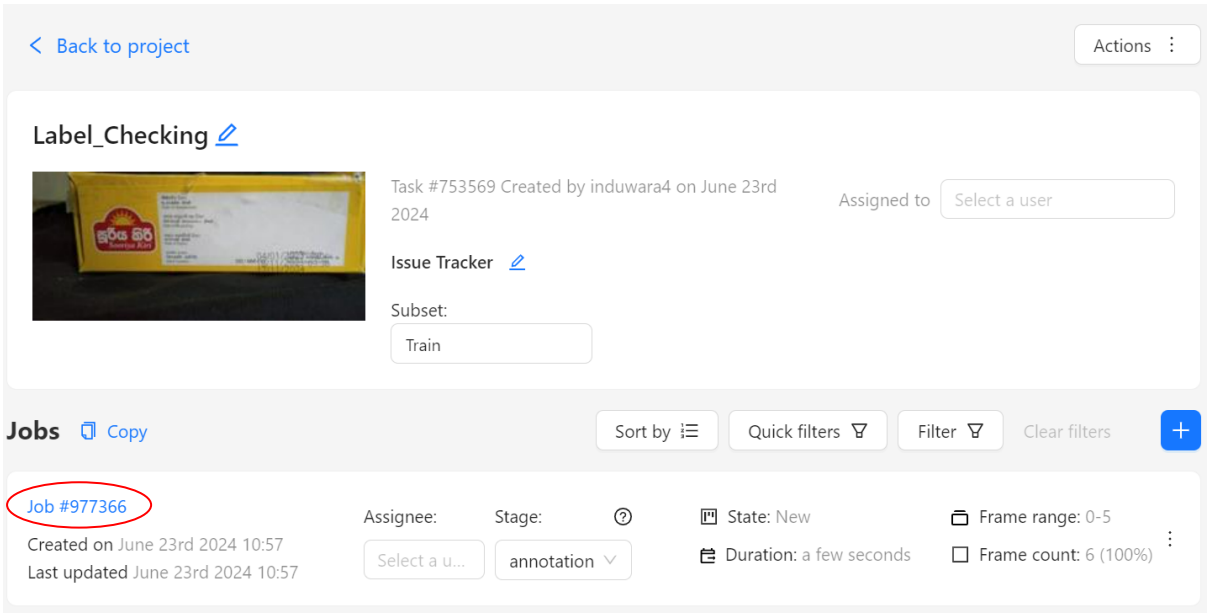
4. **Create a New Task**:
   - Click the "+" button and select "Create new task".



   - Name the task ("Label_Checking").
   - In the Subset field, select "Train".
   - Open the ***images*** folder, select all 2040 images, and drag and drop them into the "Select files" field.
   - Click "Submit and open".

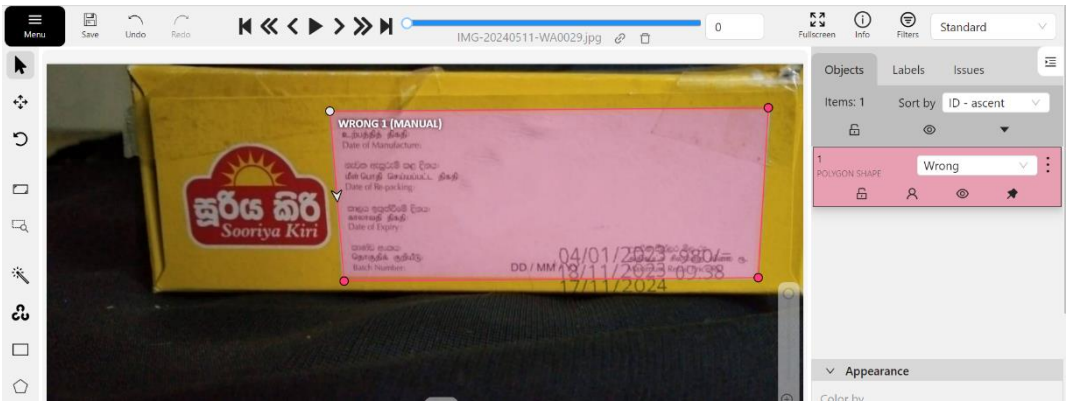


5. **Annotate Images**:
   - Click the job number in the Jobs field to open the task.

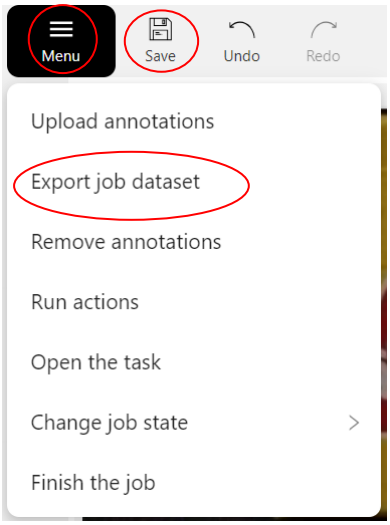


   - Click the polygon icon.
   - Select the appropriate label (Correct or Wrong), set the number of points to 4, and click "Shape".

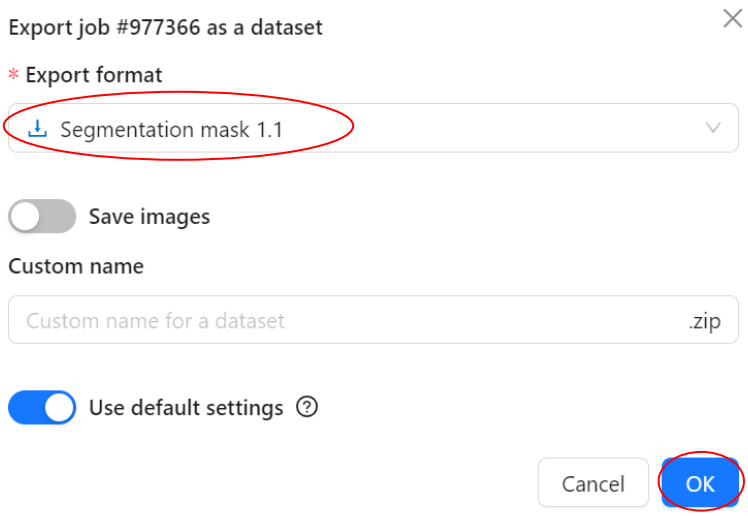
   - Draw a polygon around the white space and press "F" to go to the next image.
   - Annotate all images accordingly.



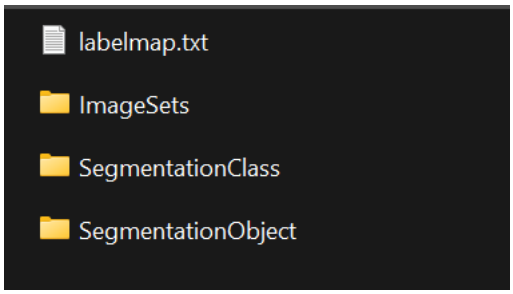
6. **Save and Export Annotations**:
   - After annotating all images, click "Save".
   - Under the menu, click "Export job dataset".



   - In the Export format section, select "Segmentation mask 1.1" and click "OK" to download the segmentation masks for the dataset.



7. **Extract and Review Masks**:
   - Extract the downloaded zip file and open it.



- The ***labelmap.txt*** file contains the colors used for each class in RGB format.

```
# label:color_rgb:parts:actions
Correct:85,104,148::
Wrong:224,88,121::
background:0,0,0::
```

- The ***SegmentationClass*** folder, contains the segmentation masks for each image.

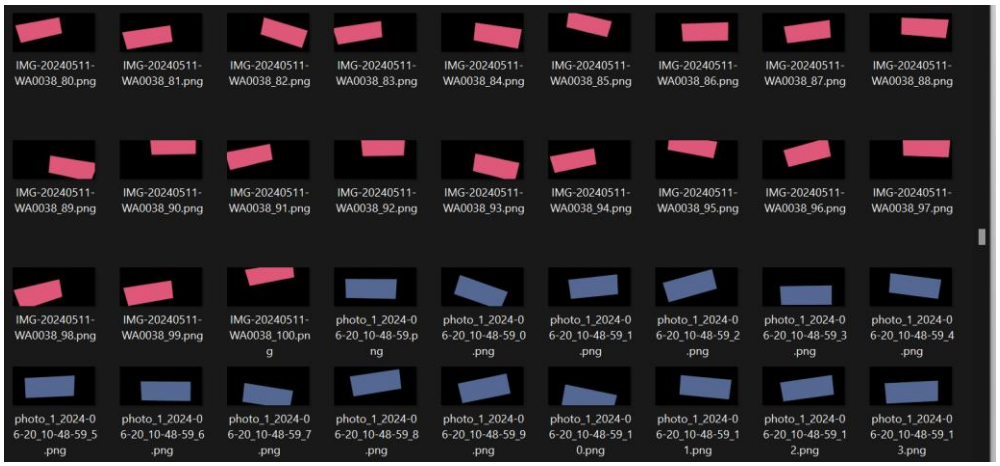


*Figure 5: Created Masks in **SegmentationClass** folder*

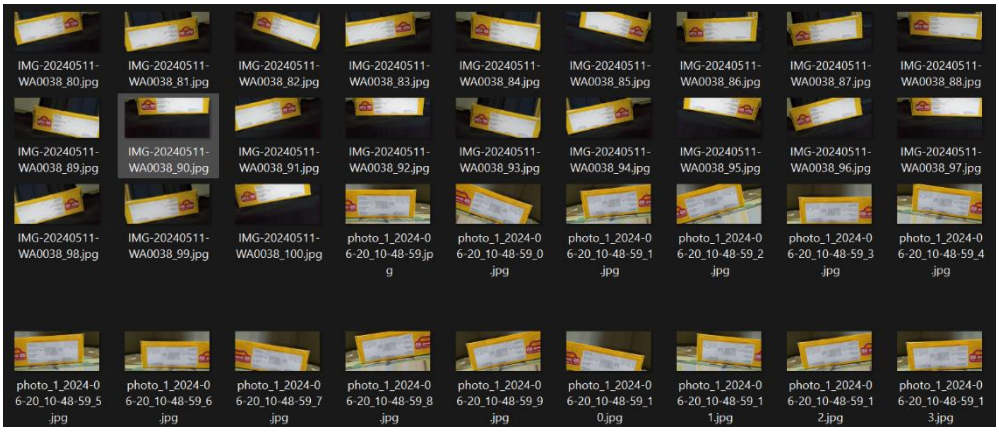


*Figure 4: Actual images in **images** folder*

2.5.2 Creating the Labels

After creating segmentation masks, the next step is to convert these masks into labels suitable for training our model. Here is how this process was done:

1. A new folder called ***masks*** was created, and all segmentation masks from the ***SegmentationClass*** folder were copied into it.
2. Another folder called ***labels*** was created to store the converted label files.
3. The following code was used to convert the segmentation masks into label files and save them in the ***labels*** folder:

Code to Convert Masks into Labels :

```
import os
import cv2
import numpy as np

input_dir = './tmp/masks'
output_dir = './tmp/labels'

# Define class RGB colors
class_colors = {
    'Correct': (148, 104, 85),
    'Wrong': (121, 88, 224),
}

# Map class names to numeric labels
class_labels = {
    'Correct': 0,
    'Wrong': 1,
}
```

```
def mask_to_polygons(mask, class_label):
    # Find contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Convert contours to polygons
    H, W = mask.shape
    polygons = []
    for cnt in contours:
        if cv2.contourArea(cnt) > 200:  # Filter out small contours
            polygon = []
            for point in cnt:
                x, y = point[0]
                polygon.append(x / W)
                polygon.append(y / H)
            polygons.append((class_label, polygon))
    return polygons
def convert_mask_to_labels(input_dir, output_dir, class_colors, class_labels):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(input_dir):
        image_path = os.path.join(input_dir, filename)
        mask = cv2.imread(image_path)

        all_polygons = []
        for class_name, color in class_colors.items():
            class_label = class_labels[class_name]

            # Create binary mask for each class
            binary_mask = cv2.inRange(mask, color, color)

            # Get polygons for this class
            polygons = mask_to_polygons(binary_mask, class_label)
            all_polygons.extend(polygons)

        # Write to label file
        label_filename = os.path.splitext(filename)[0] + '.txt'
        output_path = os.path.join(output_dir, label_filename)
        with open(output_path, 'w') as f:
            for class_label, polygon in all_polygons:
                f.write(f"{class_label} ")
                for p_, p in enumerate(polygon):
                    if p_ == len(polygon) - 1:
                        f.write(f"{p}\n")
                    else:
                        f.write(f"{p} ")

# Run the conversion
convert_mask_to_labels(input_dir, output_dir, class_colors, class_labels)
```

Explanation of the Code:

1. **Library Imports**:
   - `os`: For file and directory operations.
   - `cv2`: For image processing (OpenCV).
   - `numpy`: For numerical operations.
2. **Class Colors and Labels**:
   - `class_colors`: A dictionary mapping class names to their corresponding colors.(In here we use BGR format)
   - `class_labels`: A dictionary mapping class names to numeric labels.
3. **Functions**:
   - `mask_to_polygons`: Finds contours in the binary mask and converts them to polygons normalized to the image dimensions.
   - `convert_mask_to_labels`: Iterates through each mask in the `input_dir`, extracts polygons for each class, and writes them to a label file in the `output_dir`.
4. **Processing**:
   - For each mask, the function creates binary masks for each class based on the RGB colors.
   - It then finds and converts contours to polygons, which are saved to corresponding label files.

2.6. Training Phase

2.6.1 YOLOv8

What is YOLOv8?

YOLOv8 (You Only Look Once version 8) is the latest iteration in the YOLO family of object detection models. YOLO models are renowned for their speed and accuracy, making them ideal for real-time object detection tasks. YOLOv8 continues this legacy by incorporating state-of-the-art techniques to improve detection accuracy and processing speed.

How Does YOLOv8 Work?

YOLOv8, like its predecessors, divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell. The main innovations in YOLOv8 include:

- **Improved Backbone**: The backbone network in YOLOv8 is optimized for better feature extraction.
- **Efficient Neck**: The neck component of the model refines the features further to improve detection accuracy.
- **Advanced Head**: The detection head predicts bounding boxes, class probabilities, and objectness scores with higher precision.
- **Enhanced Loss Function**: The loss function in YOLOv8 has been fine-tuned to minimize errors more effectively.

Why Use YOLOv8 for This Project?

- **Accuracy**: YOLOv8 provides high detection accuracy, crucial for identifying misprinted labels on milk packets.
- **Speed**: The model's ability to process images quickly makes it suitable for real-time applications.
- **Flexibility**: YOLOv8 can be fine-tuned to detect custom objects, making it ideal for this specific task.

2.6.2 Initial Setup

To set up the training environment, the following steps were taken:

1. **Folder Structure**:
   - A new folder called ***Label_Checking*** was created in Google Drive.
   - Under this folder, three subfolders/files were created:
     1. ***tmp***: This folder contains the ***images*** and ***labels*** folders created previously.
     2. ***dataset.yaml***: This file contains the dataset configuration for YOLOv8.
     3. **training.ipynb**: This Jupyter Notebook file is used for training the model in Google Colab.
2. **Dataset Configuration (*dataset.yaml*)**: The ***dataset.yaml*** file was created using a text editor and contains the following configuration:

```yaml
# dataset.yaml
path: '/content/gdrive/My Drive/Label_Checking/tmp' # dataset root dir
train: images
val: images

nc: 2  # Number of classes

names: ['Correct', 'Wrong']
```

   - `path`: The root directory of the dataset.
   - `train` and `val`: Subdirectories containing the training and validation images, respectively.
   - `nc`: Number of classes (2 in this case: 'Correct' and 'Wrong').
   - `names`: List of class names.

3. **Google Colab Setup**:
   - A new Jupyter Notebook file named ***training.ipynb*** was created in Google Colab.
   - The notebook was used to train the YOLOv8 model using the dataset stored in Google Drive.

2.6.3 Training Script

The provided script sets up and trains a YOLOv8 model in Google Colab using a custom dataset stored in Google Drive. Below is a step-by-step explanation of the script:

1. **Mount Google Drive**:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

   - The first two lines mount Google Drive to the Colab environment. This allows access to the dataset and saving results directly to Google Drive. The `drive.mount` function prompts for authorization and mounts the Google Drive at `/content/gdrive`.

2. **Install YOLOv8 Library**:

```
!pip install ultralytics
```

   - This line installs the `ultralytics` package, which includes the YOLOv8 model and other utilities. The `!pip install` command is used to install Python packages in the Colab environment.

3. **Import Libraries**:

```
import os

from ultralytics import YOLO
```

   - The `os` module is imported for handling file and directory operations.
   - The `YOLO` class from the `ultralytics` package is imported to load and train the YOLOv8 model.

4. **Load Pretrained YOLOv8 Model**:

```
model = YOLO('yolov8n-seg.pt')  # load a pretrained model (recommended for training)
```

   - A pretrained YOLOv8 segmentation model is loaded using the `YOLO` class. The model file `'yolov8n-seg.pt'` is a lightweight version of YOLOv8 optimized for segmentation tasks.

5. **Train the Model**:

```
model.train(data='/content/gdrive/My Drive/Label_Checking/dataset.yaml', epochs=10, imgsz=640)
```

   - The `train` method is called on the YOLO model object to start the training process.
   - The `data` parameter specifies the path to the dataset configuration file (`dataset.yaml`), which contains information about the dataset structure.
   - The `epochs` parameter is set to `10`, indicating that the training will run for 10 epochs.
   - The `imgsz` parameter is set to `640`, specifying the image size to be used for training.

2.6.4. Model Architecture

The summary provided below gives a detailed layer-by-layer breakdown of the YOLOv8 segmentation model, which is used for training to identify misprinted labels on milk packets.

```
                 from  n    params  module                                       arguments
  0                -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11           [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14           [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17          [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20           [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22      [15, 18, 21]  1   1004470  ultralytics.nn.modules.head.Segment          [2, 32, 64, [64, 128, 256]]
YOLOv8n-seg summary: 261 layers, 3264006 parameters, 3263990 gradients, 12.1 GFLOPs
```

Breakdown of YOLOv8 Model Components

1. **Convolutional Layers (Conv):**
   - **Purpose:** Extract features from input images by applying filters.
   - **Example:**
     - `ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]`: This layer takes an input with 3 channels, applies 16 filters of size 3x3, and downsamples the image by a factor of 2.
2. **C2f Blocks:**
   - **Purpose:** Implement a more complex feature extraction by stacking multiple convolutional layers and adding shortcut connections.
   - **Example:**
     - `ultralytics.nn.modules.block.C2f [32, 32, 1, True]`: This block uses 32 filters and processes the feature maps in a more refined manner.
3. **Spatial Pyramid Pooling Fast (SPPF):**
   - **Purpose:** Combines features from different scales to improve the model's ability to detect objects of varying sizes.
   - **Example:**
     - `ultralytics.nn.modules.block.SPPF [256, 256, 5]`: Applies pooling operations to feature maps to aggregate contextual information.
4. **Upsampling Layers:**
   - **Purpose:** Increase the spatial resolution of feature maps to allow finer object localization and segmentation.
   - **Example:**
     - `torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']`: Doubles the size of the feature maps using nearest-neighbor interpolation.
5. **Concatenation Layers (Concat):**
   - **Purpose:** Merge feature maps from different layers to combine information from various stages of the network.
   - **Example:**

        - `ultralytics.nn.modules.conv.Concat [1]`: Concatenates feature maps from two or more layers.
6. **Segmentation Head:**
    - **Purpose:** Generates segmentation masks that delineate the precise boundaries of detected objects.
    - **Example:**
        - `ultralytics.nn.modules.head.Segment [2, 32, 64, [64, 128, 256]]`: Outputs segmentation masks for the two classes (Correct and Wrong) using multiple channels and feature scales.

Model Statistics

- **Total Layers:** 261
- **Total Parameters:** 3,264,006
- **Trainable Parameters:** 3,263,990
- **GFLOPs (Giga Floating Point Operations):** 12.1

2.7. Evaluation Phase

After the training is complete, the trained model weights and the metrics of the training process are saved in the `content/runs` directory by default. To evaluate the results, the following code can be used to visualize the training outputs.

Code for Evaluating and Plotting Results

```
import matplotlib.pyplot as plt
import glob

# Function to plot the images
def plot_results(file_path):
    img = plt.imread(file_path)
    plt.figure(figsize=(10, 10))
    plt.imshow(img)
    plt.axis('off')
    plt.show()

# Define source path where results are saved
source_path = '/content/runs'

# Check if source path exists
if os.path.exists(source_path):
    print(f"Source path exists: {source_path}")
    result_files = glob.glob(os.path.join(source_path, '**', '*.png'), recursive=True)

    # Plot each result image
    for file_path in result_files:
        print(f"Plotting: {file_path}")
        plot_results(file_path)
else:
    print(f"Source path does not exist: {source_path}")
```

Explanation of the code:

1. **Import Libraries:**
   - `matplotlib.pyplot`: Used for plotting and visualizing images.
   - `glob`: Used to retrieve files/pathnames matching a specified pattern.
   - `os`: Used for interacting with the operating system (e.g., checking if a directory exists).
2. **Function Definition - `plot_results`:**
   - **Input:** Takes the file path of an image.
   - **Process:** Reads and displays the image using `matplotlib`.
   - **Output:** Displays the image without axes for a cleaner look.
3. **Set the Source Path:**
   - Defines the directory where the training results are stored (`/content/runs`).
4. **Check Directory Existence:**
   - Uses `os.path.exists` to check if the specified directory exists.
   - If the directory exists, it retrieves all PNG files in the directory (and its subdirectories) using `glob`.
5. **Plotting the Results:**
   - Iterates through each result file found in the directory.
   - Calls the `plot_results` function to display each image.

Results Validation:

After running the above code, the following results were obtained:

1. **Loss Curves:**

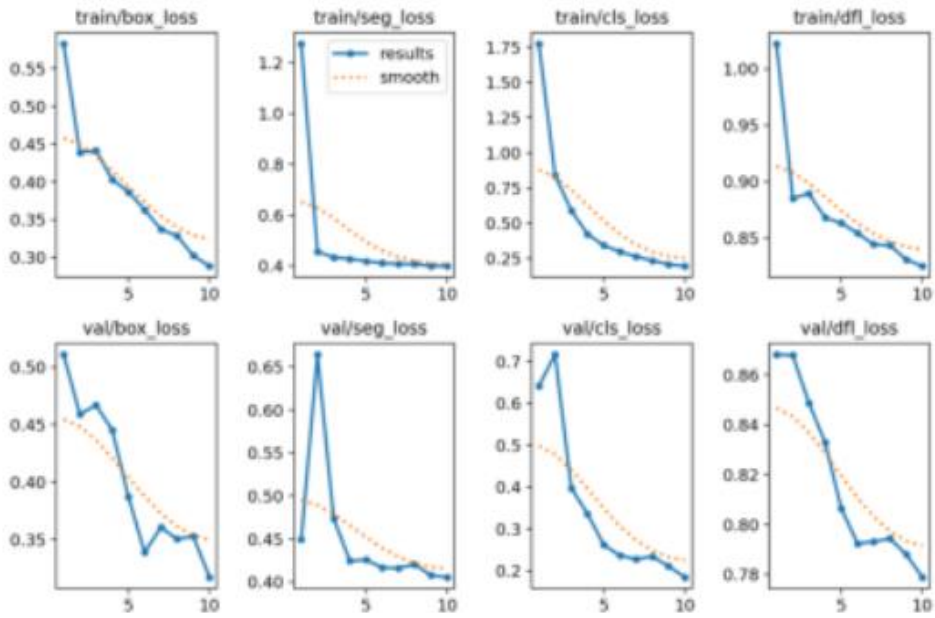


Figure 6: Loss curves

   - Loss curves are crucial for understanding how well the model is learning during training.

- **Box Loss**: Measures the localization error (i.e., how well the model predicts the bounding boxes).
- **Segmentation Loss**: Pertains to the accuracy of the segmentation masks, which are relevant if instance segmentation is used.
- **Class Loss**: Reflects how well the model classifies objects within bounding boxes.
- **DFL Loss**: If used, it focuses on additional features like domain-specific features or adaptations.

- The loss curves for both training and validation sets (Box Loss, Segmentation Loss, Class Loss, DFL Loss) consistently decreased across epochs, indicating that the model was learning effectively.

2. **Confusion Matrix**

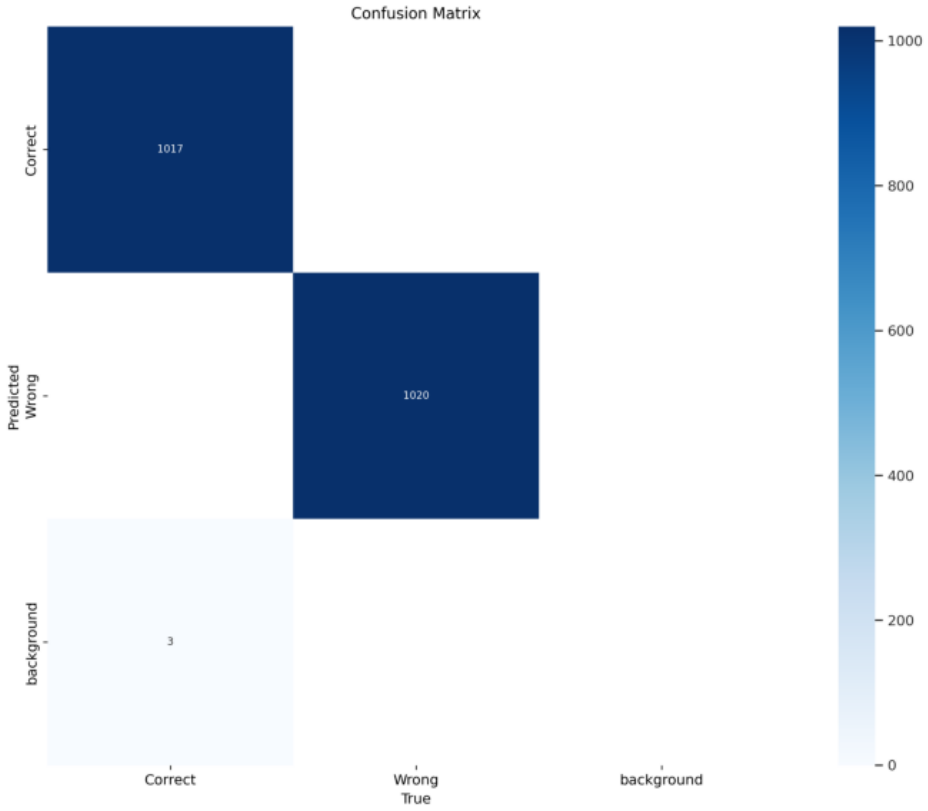


Figure 7: Confusion Matrix

- The confusion matrix provided insights into the model's classification performance:
  - **True Positives (TP)**: High values indicated accurate identification of misprinted labels.
  - **True Negatives (TN)**: Demonstrated correct rejection of non-misprinted labels.
  - **False Positives (FP)** and **False Negatives (FN)**: Were minimized, indicating fewer instances of misclassification.

2.8. Real Time Predictions

2.8.1. Downloading Model Weights

To facilitate real-time predictions in a local environment, the trained model weights (`best.pt` and `last.pt`) need to be downloaded from Google Colab using the following code:

```
from google.colab import files

# Define paths to the model weights
best_model_path = '/content/runs/segment/train/weights/best.pt'  # Update this path if necessary
last_model_path = '/content/runs/segment/train/weights/last.pt'  # Update this path if necessary

# Check if the files exist and then download them
if os.path.exists(best_model_path):
    files.download(best_model_path)
else:
    print(f"File not found: {best_model_path}")

if os.path.exists(last_model_path):
    files.download(last_model_path)
else:
    print(f"File not found: {last_model_path}")
```

2.8.2. Implementing Real-Time Object Detection

To perform real-time object detection using the downloaded YOLOv8 model, we implemented the following code in the local environment with jupyter notebook:(prediction.ipnb)

Code Explanation:

1. **Importing Libraries**:

```
import os
from ultralytics import YOLO
import cv2
```

- `os`: Provides functionalities to interact with the operating system, used here to manage file paths.
- `ultralytics.YOLO`: Imports the YOLO object detection model from the Ultralytics library for use in the script.
- `cv2`: OpenCV library for handling real-time video capture and image processing.

2. **Setting up Video Capture**:

```
# Set up the video capture for the webcam
cap = cv2.VideoCapture(0)  # 0 is the default camera
```

- Initializes video capture from the default camera (`0` denotes the default camera device).

3. **Checking Video Capture**:

```
# Check if the webcam is opened correctly
if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()
```

- Checks if the camera capture is successful (`cap.isOpened()`). If not, prints an error message and exits the script.

4. **Defining Model Path and Loading Model**:

```
# Path to the trained model
model_path = os.path.join('.', 'runs', 'segment', 'train', 'weights', 'last.pt')

# Load the custom model
model = YOLO(model_path)
```

- Constructs the path to the trained YOLO model (`last.pt` file).
- Loads the YOLO model using the specified path.

5. **Setting Detection Threshold**:

```
threshold = 0.5
```

- Defines the confidence threshold above which detected objects will be displayed.

6. **Real-Time Detection Loop**:

```
while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Could not read frame from webcam.")
        break
```

- Enters a loop to continuously read frames (`cap.read()`) from the webcam.
- If unable to read a frame, prints an error message and breaks out of the loop.

7. **Performing Object Detection**:

```
    results = model(frame)[0]
```

- Passes the current frame (`frame`) to the YOLO model (`model`) for object detection.
- `model(frame)` returns detection results, and `[0]` selects the first (and typically only) batch of results.

8. **Drawing Boxes and Labels**:

```
    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result

        if score > threshold:
            label = results.names[int(class_id)].upper()
            color = (0, 255, 0) if label != "WRONG" else (0, 0, 255)  # Green for correct, red for wrong

            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 4)
            cv2.putText(frame, label, (int(x1), int(y1 - 10)),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.3, color, 3, cv2.LINE_AA)
```

- Iterates through each detection result (`results.boxes.data.tolist()`).
- Checks if the confidence score (`score`) exceeds the threshold.
- Draws a bounding box (`cv2.rectangle`) around the detected object with a label (`cv2.putText`).

9. **Displaying Real-Time Detection**:

```
    # Display the frame with detection boxes
    cv2.imshow('YOLO Real-Time Detection', frame)
```

- Displays the current frame (`frame`) with overlaid detection boxes and labels using `cv2.imshow`.

10. **Exiting Real-Time Detection**:

```
    # Press 'q' to exit the real-time detection
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

- Releases the video capture (`cap`) and closes all OpenCV windows (`cv2.destroyAllWindows()`).

2.8.3. Results of the real time detection

Running above code enabled us to successfully identify the correct and wrong labels, as shown in the below

.

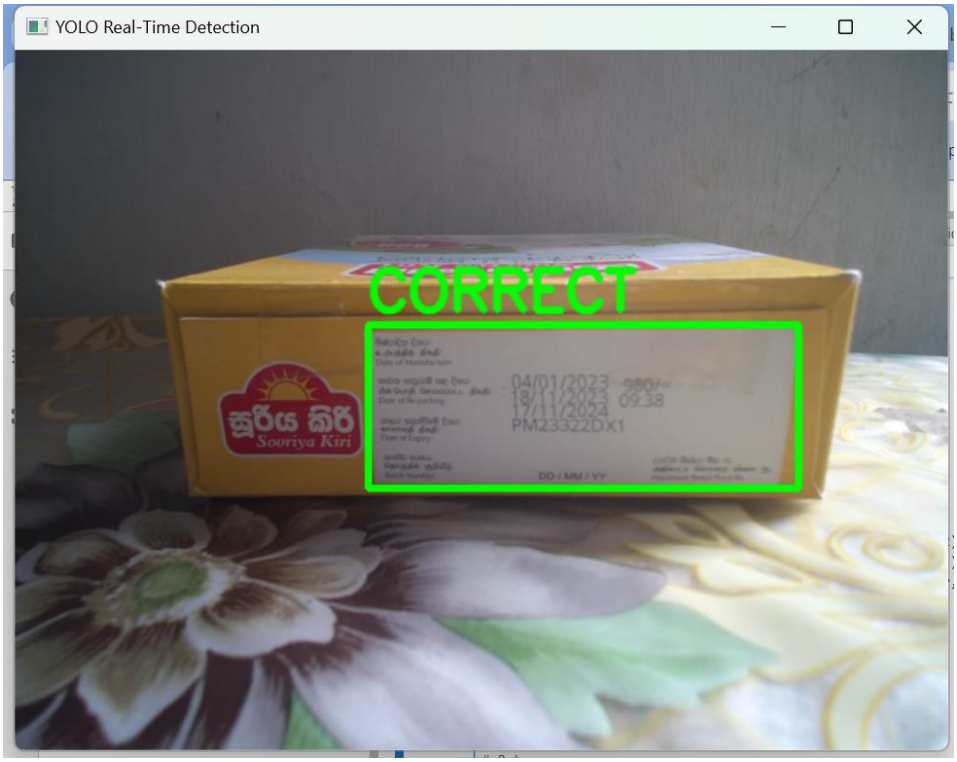


Figure 28: Real time detection of a correct label

Figure 298: Real time detection of a wrong label

2.9. Integration with the PCB for Automated Response

In this section, we describe the integration of YOLO real-time object detection with our PCB to enable automated responses based on detected labels.

2.9.1. Python Code Integration

This Python script uses a webcam and YOLO object detection to monitor real-time video for specific labels on milk packets moving on a conveyor belt. If a "WRONG" label is detected, it starts a 16-second period to count "CORRECT" labels. Depending on the count of "CORRECT" detections, it either triggers or does not trigger a mechanism via the PCB. The system operates continuously until manually stopped, displaying the detection results on the video feed. This ensures that the chance of a correct label being identified as a wrong label is reduced. Despite the low framing rate of the camera, the process is designed to ensure accurate detection.

```python
import os
from ultralytics import YOLO
import cv2
import serial
import time

# Initialize Arduino serial communication
arduino_port = 'COM3'
baud_rate = 9600
ser = serial.Serial(arduino_port, baud_rate)
time.sleep(2)  # Wait for the serial connection to initialize

# Set up the video capture for the webcam
cap = cv2.VideoCapture(0)  # 0 is the default camera

# Check if the webcam is opened correctly
if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()

# Path to the trained model
model_path = os.path.join('.', 'weights', 'last.pt')

# Load the custom model
model = YOLO(model_path)

threshold = 0.5

# Initialize detection tracking variables
wrong_detected = False
```

```
correct_count = 0
start_time = time.time()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Could not read frame from webcam.")
        break

    if not wrong_detected:
        results = model(frame)[0]

        for result in results.boxes.data.tolist():
            x1, y1, x2, y2, score, class_id = result

            if score > threshold:
                label = results.names[int(class_id)].upper()
                color = (0, 255, 0) if label != "WRONG" else (0, 0, 255)
                if label == "WRONG":
                    wrong_detected = True

                cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 4)
                cv2.putText(frame, label, (int(x1), int(y1 - 10)),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.3, color, 3, cv2.LINE_AA)

        # If wrong label detected, reset correct count and start tracking time
        if wrong_detected:
            correct_count = 0
            start_time = time.time()
```

```
    else:
        # Count correct detections for 16 seconds after wrong detection
        if time.time() - start_time <= 16:
            results = model(frame)[0]

            for result in results.boxes.data.tolist():
                x1, y1, x2, y2, score, class_id = result

                if score > threshold:
                    label = results.names[int(class_id)].upper()
                    color = (0, 255, 0) if label != "WRONG" else (0, 0, 255)
                    cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 4)
                    cv2.putText(frame, label, (int(x1), int(y1 - 10)),
                                cv2.FONT_HERSHEY_SIMPLEX, 1.3, color, 3, cv2.LINE_AA)


                    if label == "CORRECT":
                        correct_count += 1

        else:
            # Decision based on correct count after 16 seconds
            if correct_count >= 5:
                # Do not trigger pushing mechanism
```

```
                print("No action taken. Correct detections were sufficient.")
            else:
                # Trigger pushing mechanism
                ser.write(b'W')
                print("Pushing mechanism triggered.")

            # Reset variables for next detection cycle
            wrong_detected = False
            correct_count = 0

    # Display the frame with detection boxes
    cv2.imshow('YOLO Real-Time Detection', frame)

    # Press 'q' to exit the real-time detection
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
ser.close()
cv2.destroyAllWindows()
```

2.9.2. Integration with AVR Microcontroller Using Serial Communication

This C++ code configures an AVR microcontroller to control two stepper motors and a pushing mechanism on a conveyor belt system. The code continuously runs the conveyor belt and activates the pushing mechanism upon receiving a specific signal ('W') via serial communication, indicating a wrongly labeled milk packet. The code utilizes AVR port manipulation and register programming for efficient and low-level control of the microcontroller.

```
 1 #include <avr/io.h>
 2 #include <util/delay.h>
 3
 4 #define F_CPU 16000000UL  // Define CPU frequency for delay.h
 5
 6 const int stepDelay = 1000;  // Delay between steps in microseconds
 7 int count = 1;  // Initialize count as a mutable variable
 8 int push = 0;
 9 int t = 0;
10
11 char serial_read() {
12   while (!(UCSR0A & (1 << RXC0))); // Wait for data to be received
13   return UDR0;  // Get and return received data from buffer
14 }
15
16 int ADC_Read(uint8_t channel) {
17   // Select ADC channel with safety mask
18   ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);
19
```

```
  // Start single conversion
  ADCSRA |= (1 << ADSC);

  // Wait for conversion to complete
  while (ADCSRA & (1 << ADSC));

  return ADC;
}

int main() {
  // Setup
  // Set direction and step pins as outputs for motor 1 and motor 2
  DDRD |= (1 << PD2) | (1 << PD3) | (1 << PD4) | (1 << PD5); //

  // Set initial direction for motor 2
  PORTD |= (1 << PD4);  // Set PD4 high

  // Initialize serial communication for debugging
  UBRR0H = 0;
  UBRR0L = 103; // Baud rate 9600
  UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Enable receiver and
transmitter
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit data

  // ADC Initialization
  // AVcc with external capacitor at AREF pin
  ADMUX = (1 << REFS0);

  // ADC Enable and prescaler of 128
  // 16000000/128 = 125000
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

  while (1) {
    // Loop
    if (UCSR0A & (1 << RXC0)) { // Check if data is received
      char signal = serial_read();
      if (signal == 'W') {
        push = 1;
      }
    }

    if (push == 1) {
      int sensorValueA4 = ADC_Read(4); // Read analog pin A4
      int sensorValueA5 = ADC_Read(5); // Read analog pin A5

      // Reset count based on sensor values
      if (sensorValueA4 > 1022 && t == 1) {
        count = 0;  // Set count to 0 if sensorValueA4 > 1022
      } else if (sensorValueA5 > 1022) {
        count = 1;  // Set count to 1 if sensorValueA5 > 1022
        PORTD &= ~(1 << PD2); // Set PD2 low

        for (int i = 0; i < 50; i++) {
          PORTD |= (1 << PD3) | (1 << PD5);  // Set PD3 and PD5 high
          _delay_us(stepDelay);
          PORTD &= ~((1 << PD3) | (1 << PD5)); // Set PD3 and PD5 low
```

```
76            _delay_us(stepDelay);
77          }
78
79          push = 0;
80        }
81
82        if (sensorValueA4 > 1022) {
83          t = 1;
84        }
85
86        // Set PD2 low if count is 1, otherwise set it high for motor 1
87        if (count == 1) {
88          PORTD &= ~(1 << PD2); // Set PD2 low
89        } else {
90          PORTD |= (1 << PD2);  // Set PD2 high
91        }
92
93        // Step both motors simultaneously
94        PORTD |= (1 << PD3) | (1 << PD5);  // Set PD3 and PD5 high
95        _delay_us(stepDelay);
96        PORTD &= ~((1 << PD3) | (1 << PD5)); // Set PD3 and PD5 low
97        _delay_us(stepDelay);
98      } else {
99        // Default behavior when no serial signal is received
100       PORTD |= (1 << PD5);  // Step motor 2 (set PD5 high)
101       _delay_us(stepDelay);
102       PORTD &= ~(1 << PD5);  // Set PD5 low
103       _delay_us(stepDelay);
104     }
105   }
106
107   return 0;
108 }
```

3. System Integration

Figure 30: Correct Label Detection

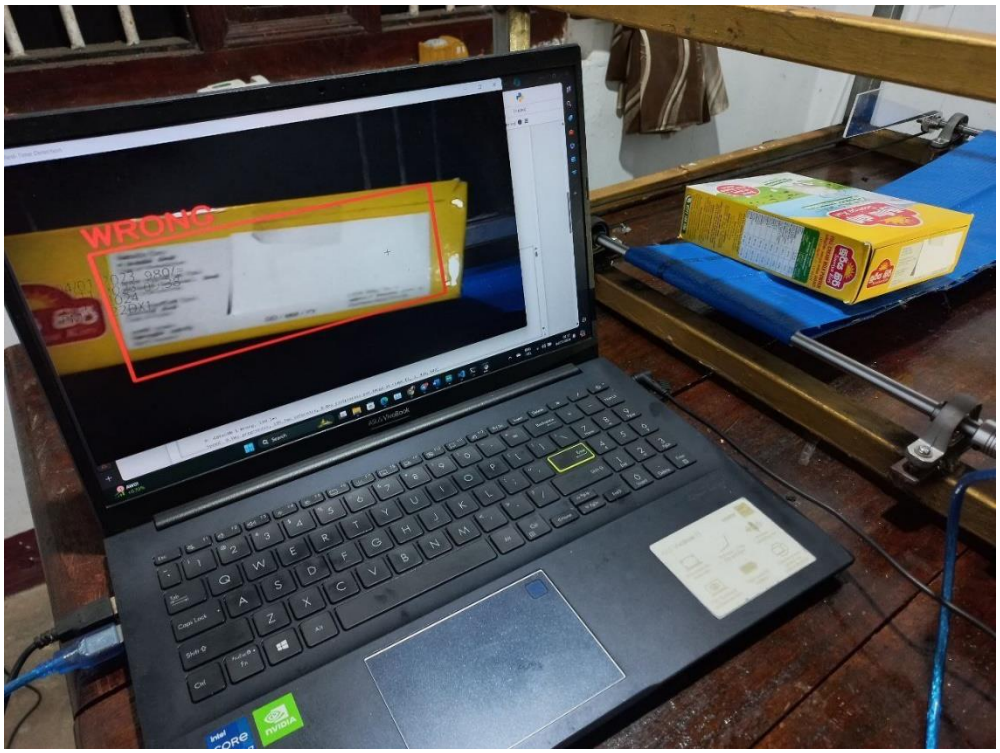


Figure 31: Wrong Label Detection

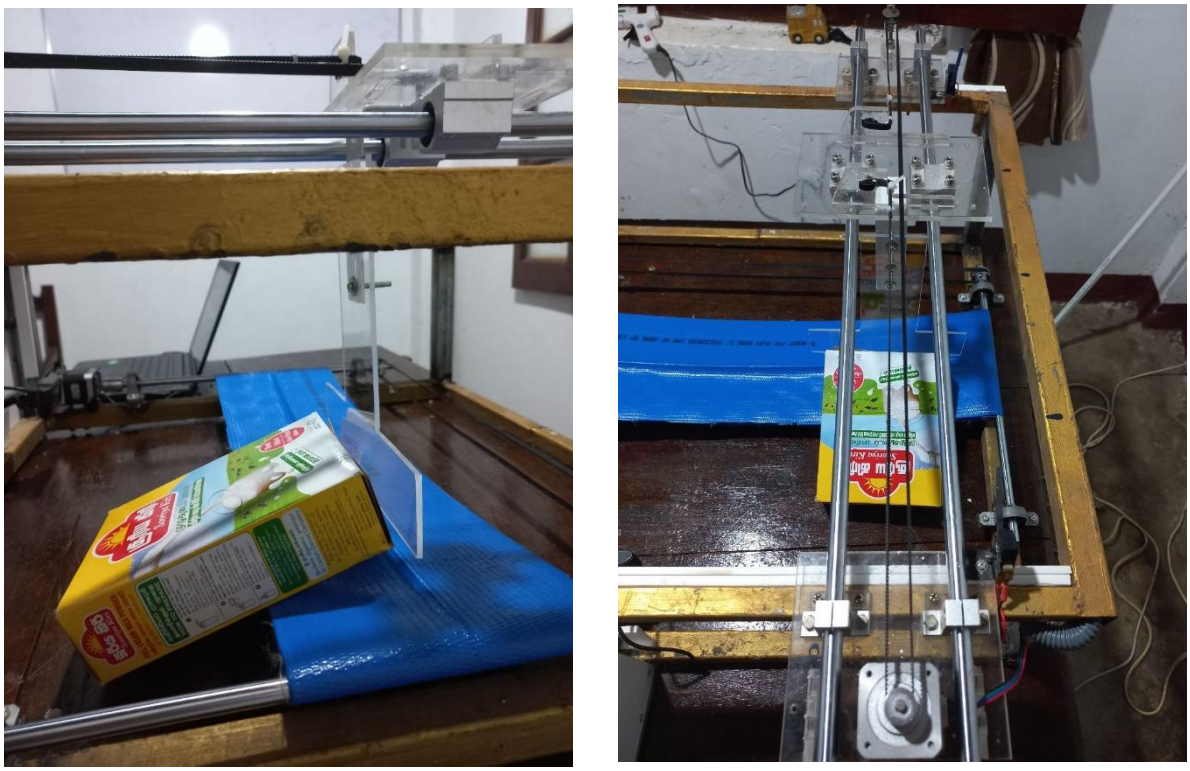


Figure 32: Pushing the Wrong Labels by the mechanism.

Results and Observations

The final product of our project, along with the results and observations, can be accessed at the following links:

- **Project demonstration Video:** https://uniofmora-my.sharepoint.com/:f:/g/personal/mihirangangd_21_uom_lk/EnhfXuybQFBHpfi9hA1jpi4BiTE904bJCvYNSIOq6mRapA?e=T8gVtB
- **Github :** https://github.com/InduwaraGayashan001/Misprinted-Label-Detection

Appendix: Daily Progress

January 29-February 4

Initial Submission

- **January 29:** Project proposal drafted and submitted for initial review.
- **January 30:** Feedback received; revisions made to proposal.
- **January 31:** Finalized project proposal and submitted.
- **February 1-4:** Awaiting approval and preparing initial research materials. During this time, we gather resources to ensure a smooth start once the proposal is approved.

February 5-February 11

Finding Information - Market and Other Analysis

- **February 5:** Began market analysis; identified key competitors and market needs. Understanding the market landscape helps in defining our project's unique value proposition.
- **February 6-7:** Conducted detailed analysis of existing solutions and their limitations. This analysis reveals gaps and opportunities in the current market.
- **February 8:** Compiled data on market trends and potential demand. Accurate data is vital for making informed decisions.
- **February 9:** Identified key features needed for our system to stand out. These features will differentiate our product from competitors.
- **February 10-11:** Summarized findings and prepared a report on market analysis.

February 11-February 18

Finding Information - Stakeholders

- **February 11:** Identified potential stakeholders and their interests.
- **February 12:** Reached out to industry experts for insights. Expert opinions provide valuable guidance and validation.
- **February 13-15:** Conducted interviews and surveys with potential users. Direct feedback from users helps in shaping user-centric solutions.
- **February 16:** Analyzed feedback from stakeholders. This analysis highlights critical needs and expectations.
- **February 17-18:** Compiled a comprehensive report detailing stakeholder requirements and expectations. A detailed report ensures stakeholder needs are addressed in our design.

February 19-February 25

Initial PCB Design and Software Setup

- **February 19:** Started conceptual design of PCB and set up software development environment.

- **February 20-21:** Determined essential components for PCB and selected Python and OpenCV for image processing.
- **February 22:** Developed initial script to capture and preprocess images from the camera.
- **February 23-24:** Created initial schematic diagrams and implemented basic image segmentation techniques. These diagrams and techniques form the basis of our system's functionality..
- **February 25:** Reviewed and refined PCB design; tested initial image processing pipeline with sample data.

February 26-March 3

Research and Software Development

- **February 26:** Researched available sensors and their integration into the system.
- **February 27-28:** Evaluated different microcontrollers for system compatibility.
- **March 1:** Identified suitable image processing libraries for misprint detection; integrated YOLO object detection model.
- **March 2-3:** Investigated communication protocols between PCB and software; trained YOLO model on sample misprint data.

March 4-March 10

Conceptual Design Basic Structure and Software Optimization

- **March 4:** Drafted the basic structural layout of the system.
- **March 5-6:** Defined the workflow for detecting misprints and triggering the rejection mechanism; optimized detection accuracy and speed. Optimized workflows improve system efficiency.
- **March 7-8:** Designed the initial mechanical setup for the conveyor belt system. The mechanical setup is essential for material handling.
- **March 9-10:** Integrated feedback to refine the conceptual design; developed interface for real-time detection and feedback. Real-time feedback enhances user interaction and system performance.

March 11-March 17

Research and Software-Hardware Integration

- **March 11:** Explored advanced image processing techniques for accurate detection. Advanced techniques improve detection accuracy.
- **March 12:** Studied machine learning models for improving detection accuracy; integrated software with PCB for signal transmission. Machine learning models enhance the system's ability to learn and adapt.
- **March 13-15:** Tested different algorithms for real-time processing. ing various algorithms ensures the best performance.
- **March 16-17:** Compiled research findings and prepared a summary report; refined software based on hardware integration tests. Continuous refinement based on tests ensures system reliability.

March 18-March 24

Prototype Design Development

- **March 18:** Developed a detailed PCB layout, Developed the initial Solidworks design for the enclosure. Detailed designs are crucial for accurate prototyping.
- **March 19-21:** Simulated the PCB design for validation. Rechecked and Reviewed the solidworks design.
- **March 22:** Made necessary adjustments based on simulation results and reviews.
- **March 23-24:** Prepared the final design for prototype production.

March 25-March 31

Prototype Assembly and Testing

- **March 25-26:** Assembled initial hardware prototype; implemented additional software features for misprint classification.
- **March 27:** Integrated PCB designs with sensors and microcontroller; enhanced real-time processing capabilities. Integration and enhancement improve overall system performance.
- **March 28-30:** Conducted initial tests on the prototype designs; refined real-time processing software.
- **March 31:** Documented prototype assembly and testing results.

April 1-April 7

Prototype Refinement and Integration Testing

- **April 1-3:** Continued refining the prototype based on test results. Refinement ensures the prototype meets all requirements.
- **April 4-5:** Implemented software integration for real-time processing; conducted comprehensive software-hardware integration tests. Comprehensive testing ensures all components work seamlessly together.
- **April 6-7:** Conducted comprehensive testing of the prototype; finalized and documented software for deployment. Finalizing software is crucial for a stable and reliable system.

April 8-April 14

Ordering Components

- **April 8:** Finalized the list of required components. Finalizing the list ensures all necessary parts are accounted for.
- **April 9:** Sourced suppliers and obtained quotes. Sourcing and quotes help manage the budget effectively.
- **April 10-11:** Placed orders for all necessary components. Timely ordering ensures no delays in the project timeline.
- **April 12-14:** Followed up on order confirmations and delivery schedules. Following up ensures all components are delivered on time.

April 15-April 21

Ordering PCB and Final Initial Mechanical Parts Design

- **April 15:** Finalized PCB design based on prototype testing feedback.
- **April 16-17:** Placed order for PCBs.
- **April 18:** Designed final initial mechanical parts for the conveyor belt system. Designing mechanical parts ensures smooth material handling.
- **April 19-21:** Ordered mechanical parts and confirmed delivery dates. Confirming delivery dates helps in planning the assembly phase.

April 29-May 5

Design Document Review and PCB Soldering

- **April 29:** Completed the design document and prepared it for review.
- **April 30:** Submitted the design document for review by another group.
- **May 1:** Received feedback and signed declaration from the review group.
- **May 2:** Incorporated feedback and finalized the design document.
- **May 3:** PCB and parts received.
- **May 4-5:** Soldered components onto the PCB and performed initial tests.

Appendix: Previous code

```
const int dirPin = 2;         // Direction pin for motor 1
const int stepPin = 3;        // Step pin for motor 1
const int dirPin1 = 4;        // Direction pin for motor 2
const int stepPin1 = 5;       // Step pin for motor 2
const int stepDelay = 1000;    // Delay between steps in microseconds
int count = 1;      // Initialize count as a mutable variable
int push = 0;
int t = 0;
int i = 0;

const int sensorPinA4 = A4;    // Analog pin A4
const int sensorPinA5 = A5;    // Analog pin A5

void setup() {
  pinMode(dirPin, OUTPUT);     // Set direction and step pins as outputs for
motor 1
  pinMode(stepPin, OUTPUT);

  pinMode(dirPin1, OUTPUT);    // Set direction and step pins as outputs for
motor 2
  pinMode(stepPin1, OUTPUT);

  digitalWrite(dirPin1, HIGH); // Set initial direction for motor 2

  Serial.begin(9600);          // Initialize serial communication for
debugging
```

```
}

void loop() {
  if (Serial.available() > 0) {
    char signal = Serial.read();
    if (signal == 'W') {
      push = 1;
    }
  }



    if (push == 1) {  // Respond to serial signal 'W'



      int sensorValueA4 = analogRead(sensorPinA4);
      int sensorValueA5 = analogRead(sensorPinA5);




      // Reset count based on sensor values
      if (sensorValueA4 > 1022 && t==1) {
        count = 0;  // Set count to 0 if sensorValueA4 > 1000
      } else if (sensorValueA5 > 1022) {
        count = 1;  // Set count to 1 if sensorValueA5 > 1000
        digitalWrite(dirPin, LOW);

         for (int i = 0; i < 50; i++) {
        digitalWrite(stepPin, HIGH);
        digitalWrite(stepPin1, HIGH);
        delayMicroseconds(stepDelay);
        digitalWrite(stepPin, LOW);
        digitalWrite(stepPin1, LOW);
        delayMicroseconds(stepDelay);
      }


        push = 0;
      }

      if (sensorValueA4 > 1022) {
        t=1;
      }

      // Set dirPin LOW if count is 1, otherwise set it HIGH for motor 1
      if (count == 1) {
        digitalWrite(dirPin, LOW);
      } else {
        digitalWrite(dirPin, HIGH);
      }

      // Step both motors simultaneously
      digitalWrite(stepPin, HIGH);
      digitalWrite(stepPin1, HIGH);
      delayMicroseconds(stepDelay);
```

```
      digitalWrite(stepPin, LOW);
      digitalWrite(stepPin1, LOW);
      delayMicroseconds(stepDelay);
    }
   else {
    // Default behavior when no serial signal is received
    digitalWrite(stepPin1, HIGH);   // Step motor 2
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin1, LOW);
    delayMicroseconds(stepDelay);
  }
}
```

References

- Ultralytics, "YOLO Object Detection Documentation," Ultralytics, 2024. [Online]. Available: https://docs.ultralytics.com.
- G. Jocher, A. Chaurasia, J. Qiu, and A. Stoken, "YOLOv5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5.
- OpenCV, "OpenCV Documentation," OpenCV, 2024. [Online]. Available: https://docs.opencv.org/.

Reviews

Reviewer	Index No	Signature
LAKSHAN W.D.T.	**210335T**	
GUNAWARDANA W.M.T.V.	**210198A**	
WICKRAMASINGHE M.M.M.	**210707L**	
RAJAPAKSHA N.N.	**210504L**