Lam RESEARCH

T-WORKS

Mahindra University
Global Thinkers. Engaged Leaders.

The Lam Research Challenge

A Nationwide Systems Engineering Competition

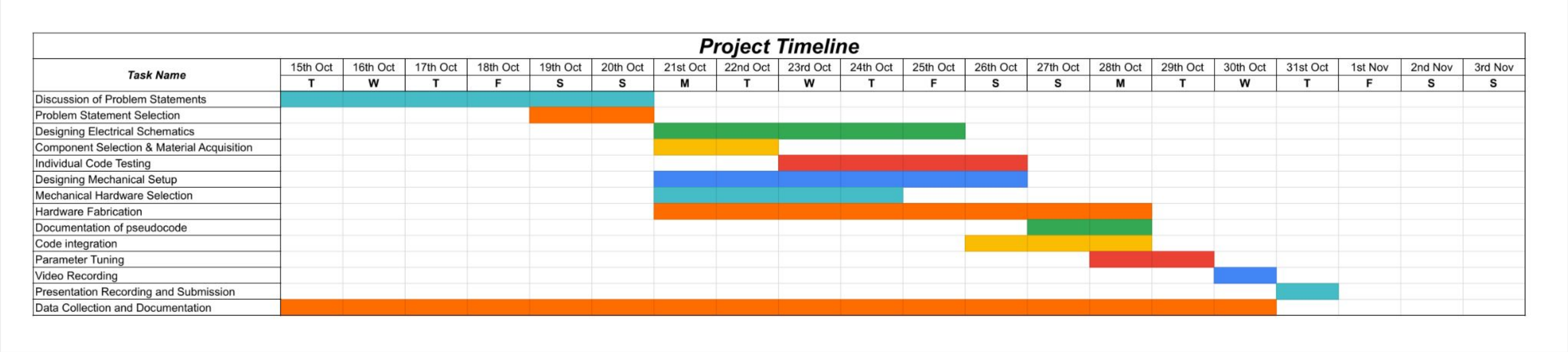
Washing Machine Bros. 2.0

Team ID: TW-LM-009
Contraption ID: LR2C1

Presentation Overview

- Some Initial Problems
- High-level Overview of Contraption
- Hardware and Mechanical System
- Electrical Schematics
- Code Explanation

Timeline and Problem Selection

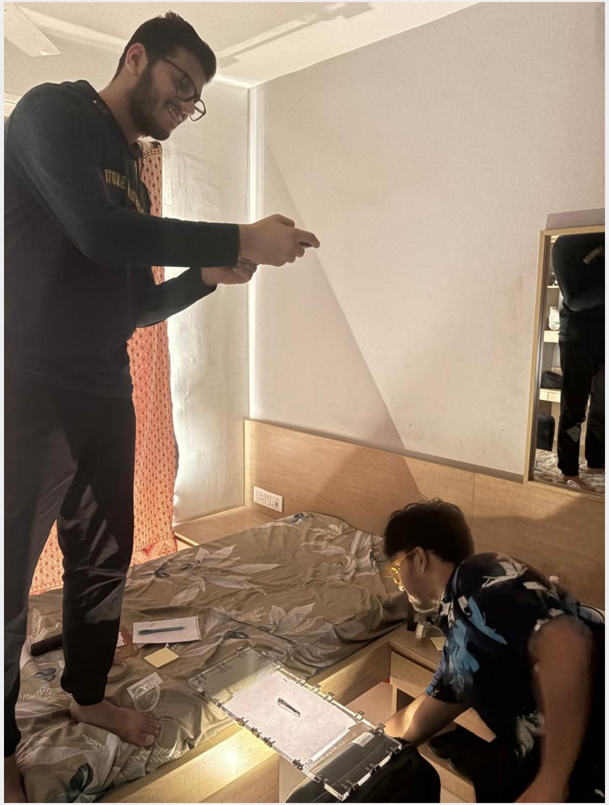


- Problem Selection took up nearly as much time as some of the more technical tasks.
- A majority of the tasks were completed in the span of less than 10 days.

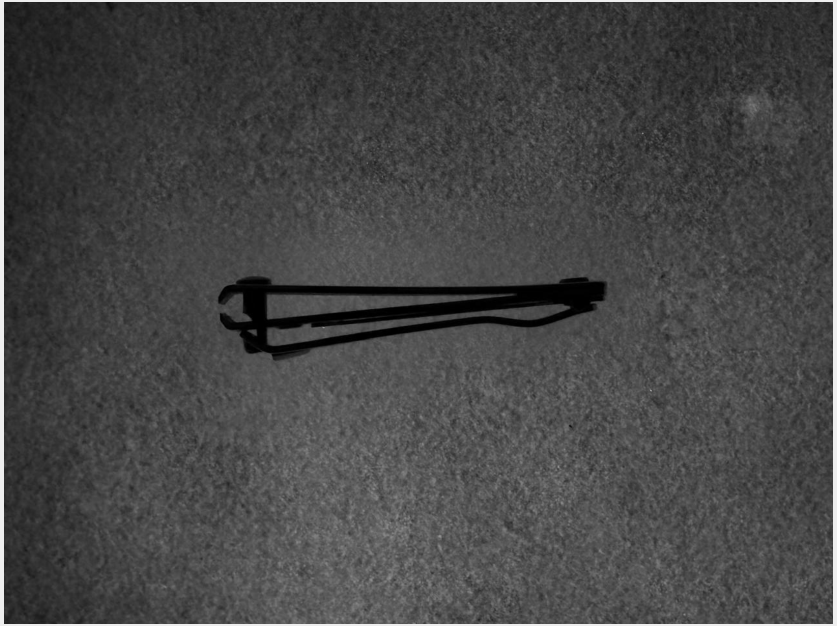
Timeline and Problem Selection

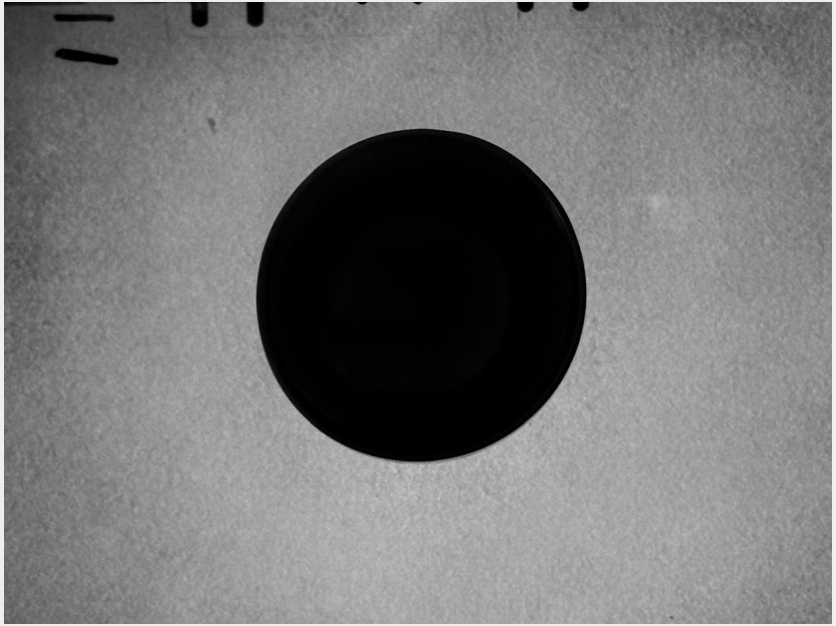
- Initially, we planned to go with problem statement number 3.
- The reason for this was because on the get go the solution methodology seemed quite straightforward with a minimal amount of hardware and electronics needed for development.
- Making changes and further refinement to a project that mostly relied on software seemed much more appealing.
- In particular, it seemed that last minute failures were not as realistic a possibility as it was with problem statements 1 and 2.
- Building a prototype model for testing and experimentation seemed very easy.

Timeline and Problem Selection

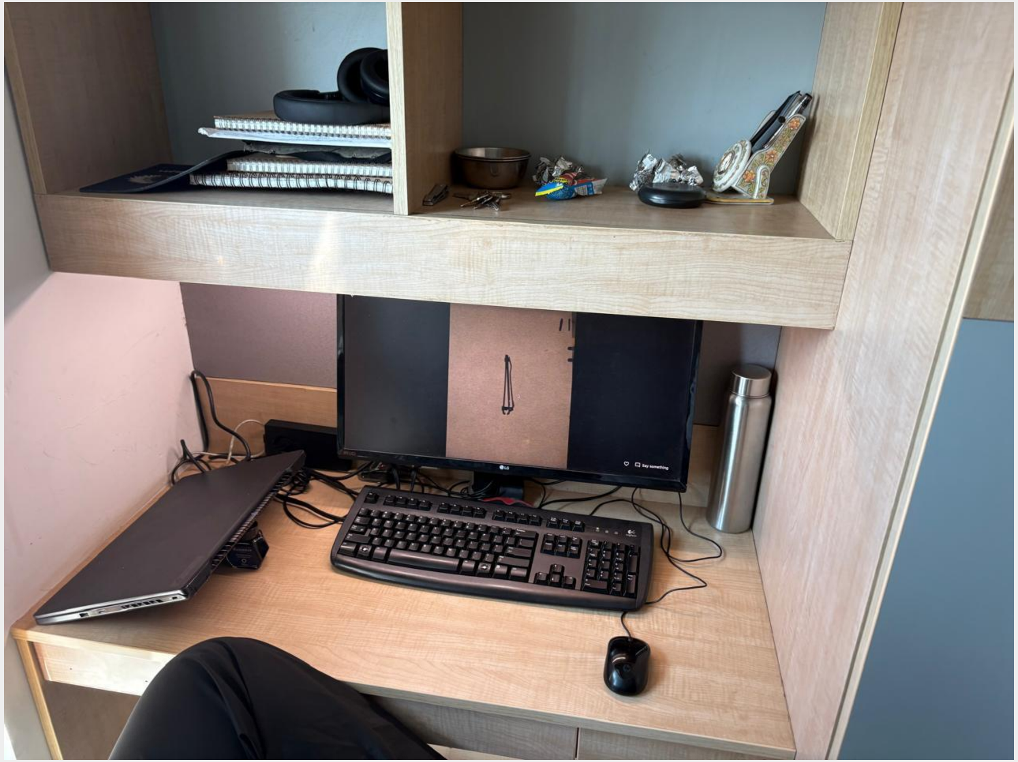


Timeline and Problem Selection



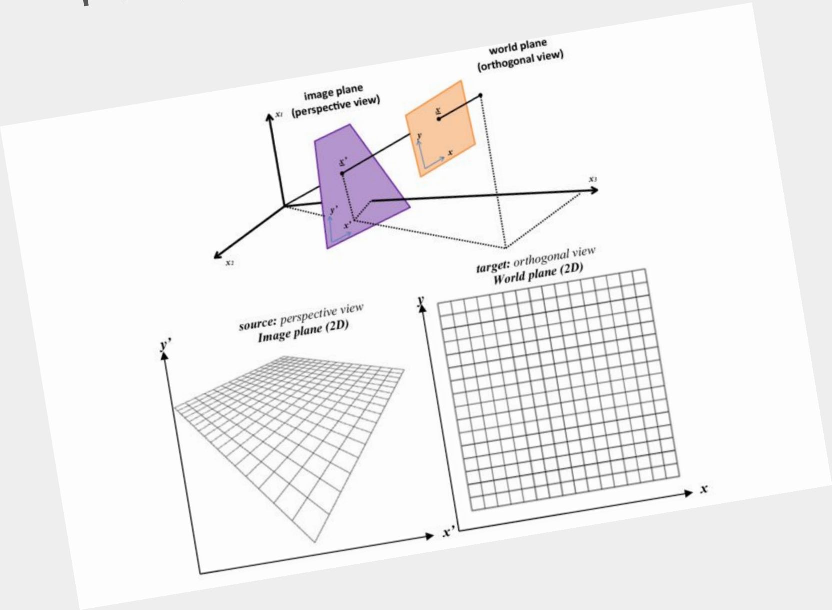


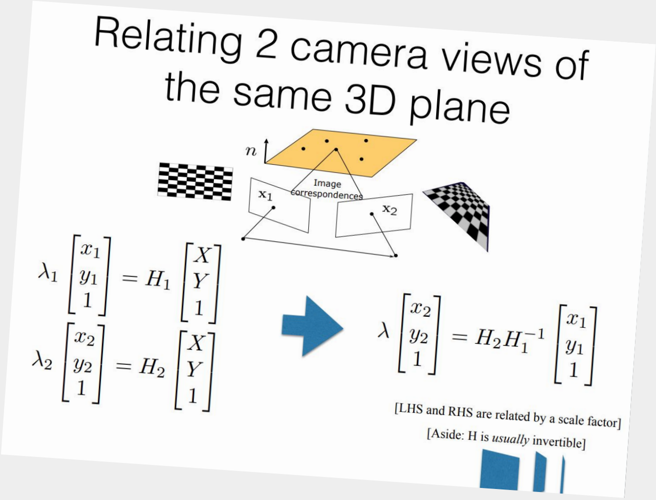
Timeline and Problem Selection



Perspective Correction

Multiple Cameras?





Moving Cameras?

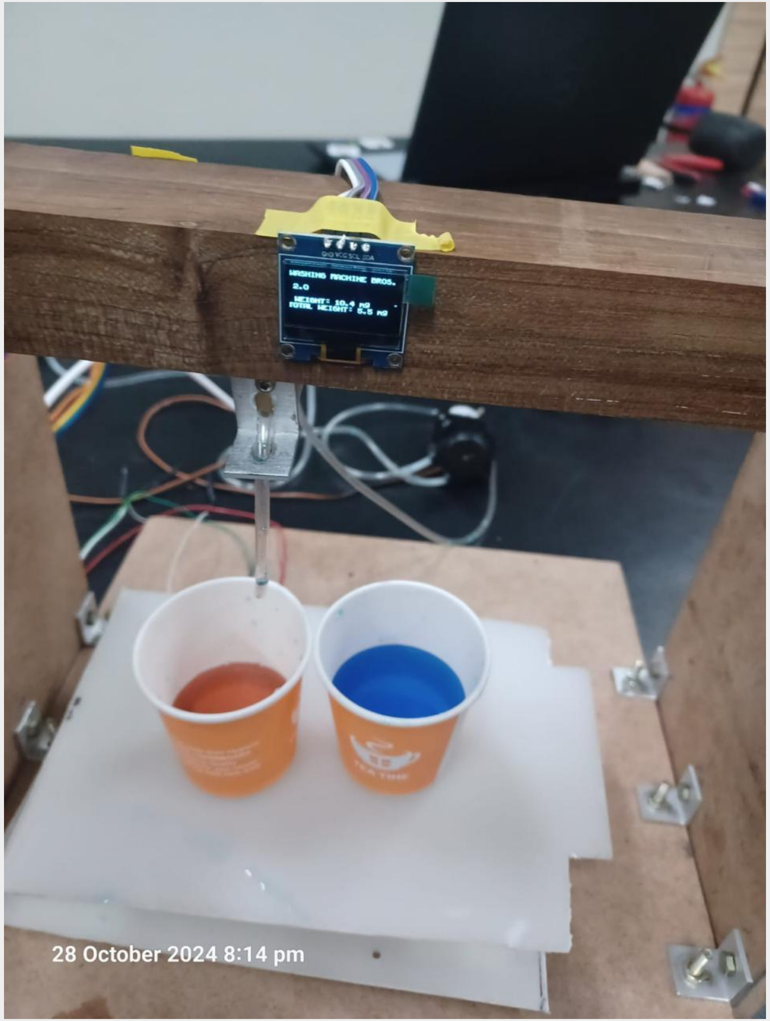
Plane to Plane Homography

Changing our Problem Statement

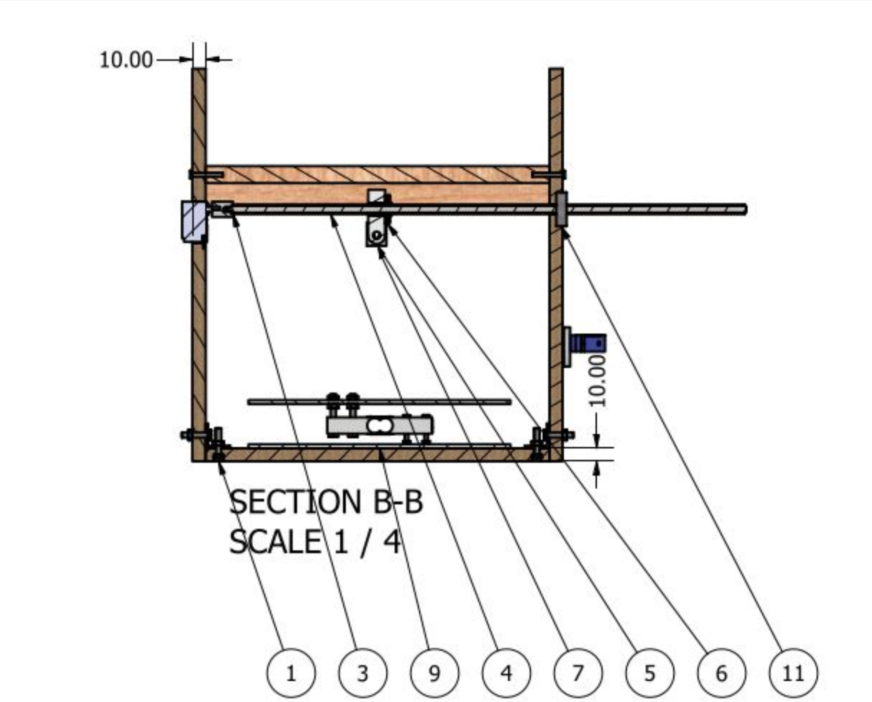
Split ourselves into two groups exploring the viability of doing:

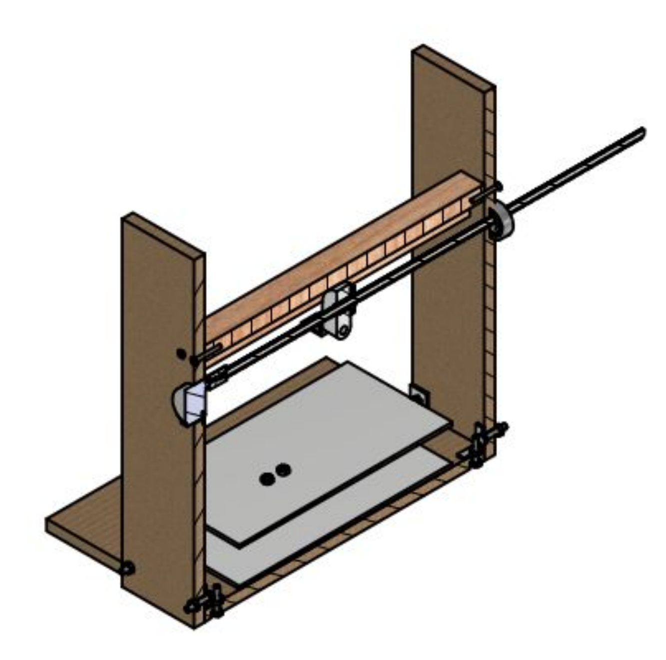
- Problem Statement 1
- Problem Statement 3

Ultimately we switched to problem statement 1.

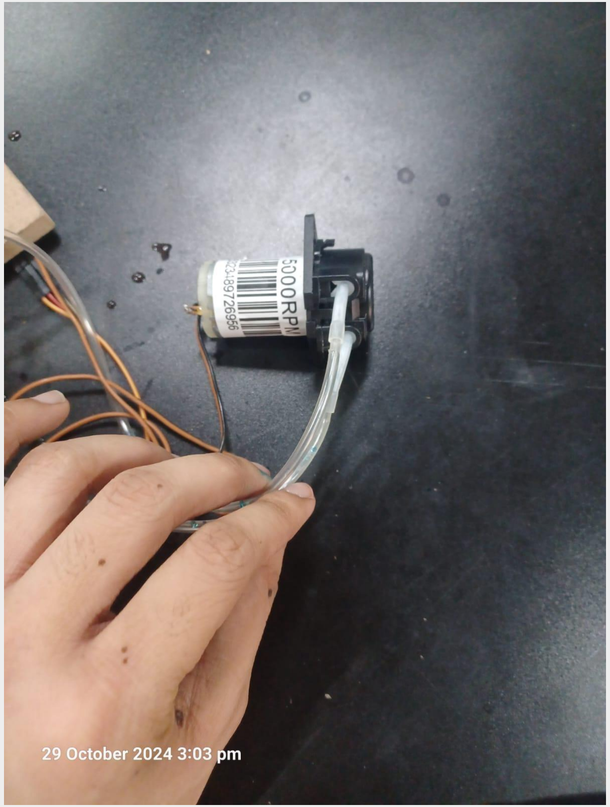

GND VCC SCL SDA
WASHING MACHINE BROS.
2.0
WEIGHT: 10.4 mg
TOTAL WEIGHT: 5.5 mg
28 October 2024 8:14 pm

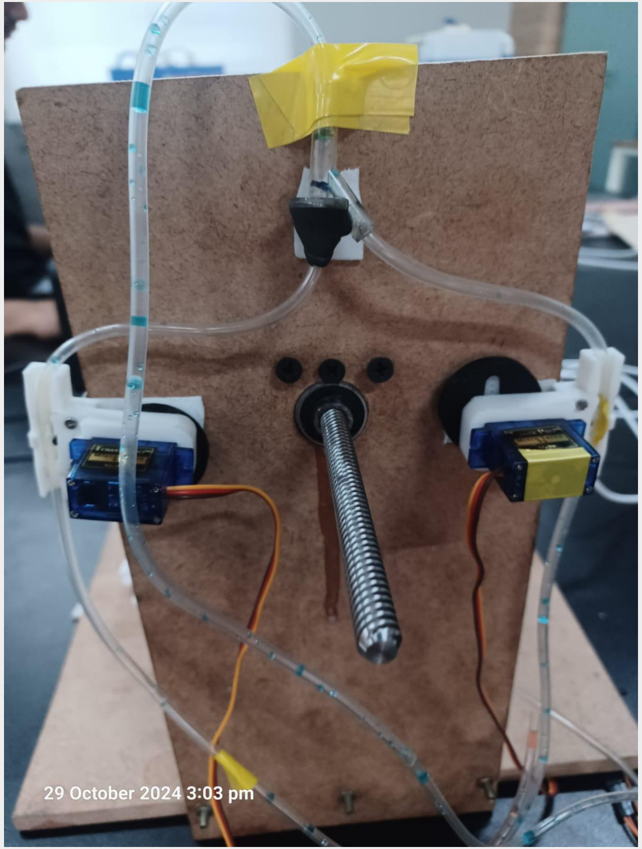
High-level Overview of Contraption





High-level Overview of Contraption





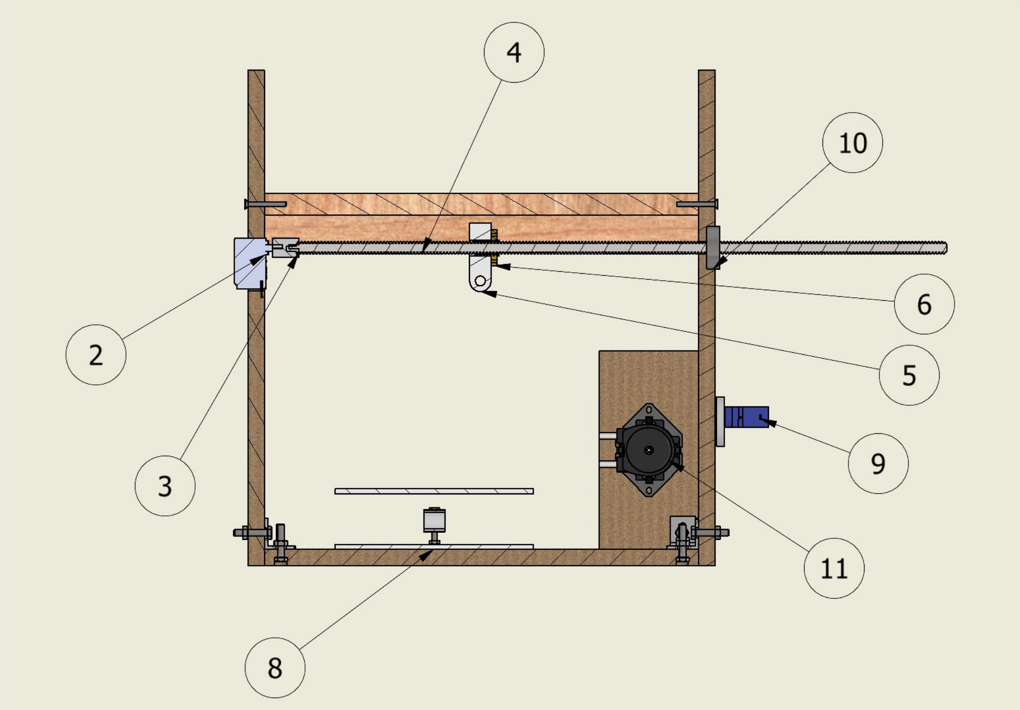
Hardware and Mechanical System

The objective is to design a simple, efficient hardware system for motor mounting that enables precise linear actuation of the spout, while prioritizing manufacturing speed and minimizing complexity

List of main components

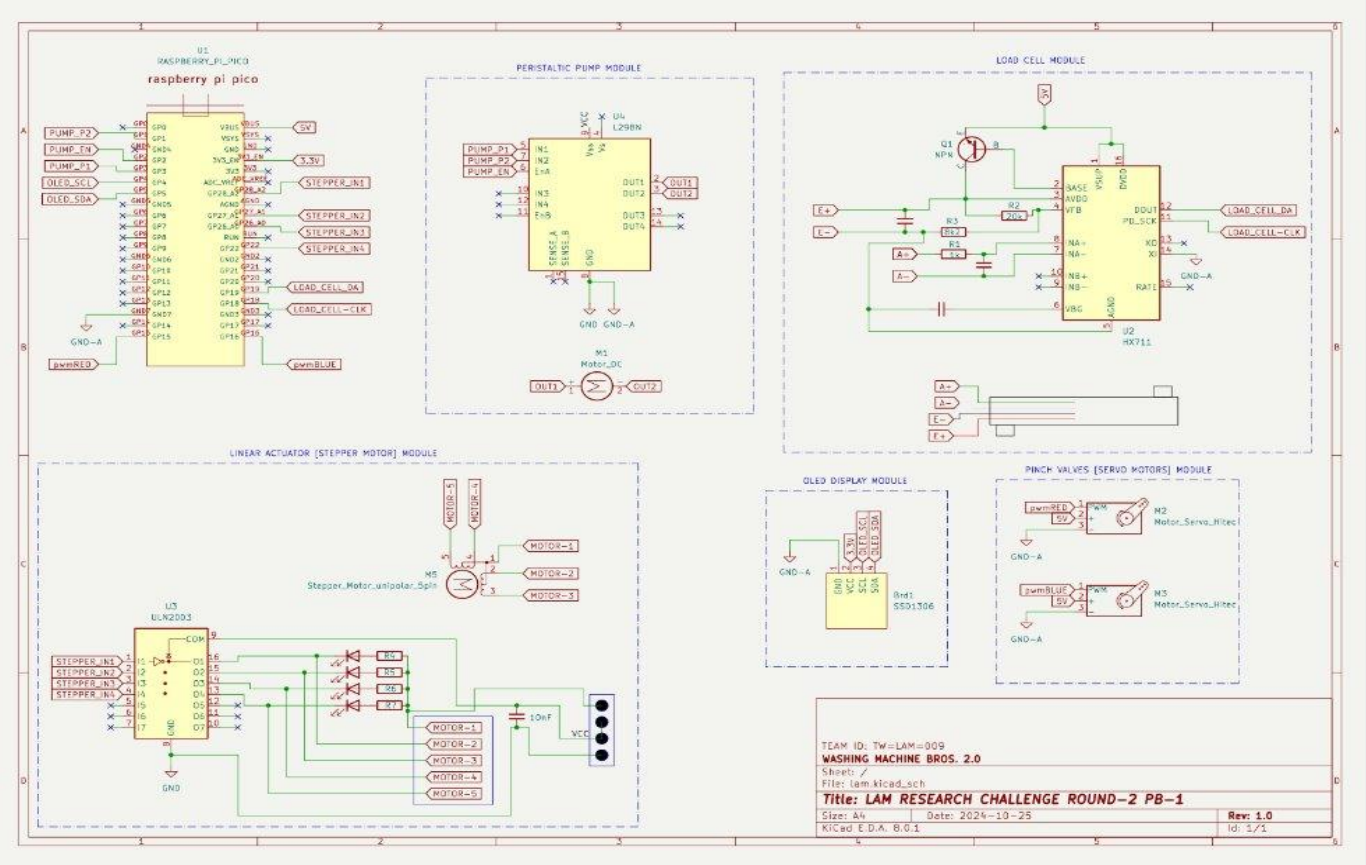
- MDF boards (base and mounts)
- Wood block (linear actuator guide rail)
- Pinch Valves
- Spout
- Clamps and Fasteners

Hardware and Mechanical System



PARTS LIST				
ITEM	QTY	PART NUMBER	DESCRIPTION	MATERIAL
1	1	Mechanical frame subassembly	MDF and wood components fixed to make the frame	
2	1	28BYJ-48_Stepper	Unipolar Stepper motor	Electronics
3	1	Coupler	Aluminium coupler between motor and lead screw	Aluminium 6061
4	1	Trapezoidal lead screw tr8x8-4	8mm dia lead screw for linear actuation	Stainless steel
5	1	pipe mount	3D ABS printed mount for spout	ABS Plastic
6	1	LeadScrew Nut 8mm x 2mmPitch	Lead screw nut used to mount the spout and linear actuation along lead screw	Brass
7	1	BS 3692 - M4	Precision hexagon nuts	Steel, Mild
8	1	1Kg Load cell	Load cell to measure the weight of the cups	Electronics
9	2	pinch valves	Valves to close differnt coloured liquids	Electronics
10	1	BS 292: Part 1 - 7000 - 10 x 26 x 8	Rolling bearings: ball bearings cylindrical and spherical roller bearings- Specification for dimensions of ball bearings, cylindrical and spherical roller bearings	Steel, Mild
11	1	Adafruit peristaltic pump assembly	peristaltic pump is used to to pump the liquds to the spout	Electronics
12	1	lclamp	L clamp is put on the spout to point into the cup	Aluminum 6061

Electrical Schematic Diagram



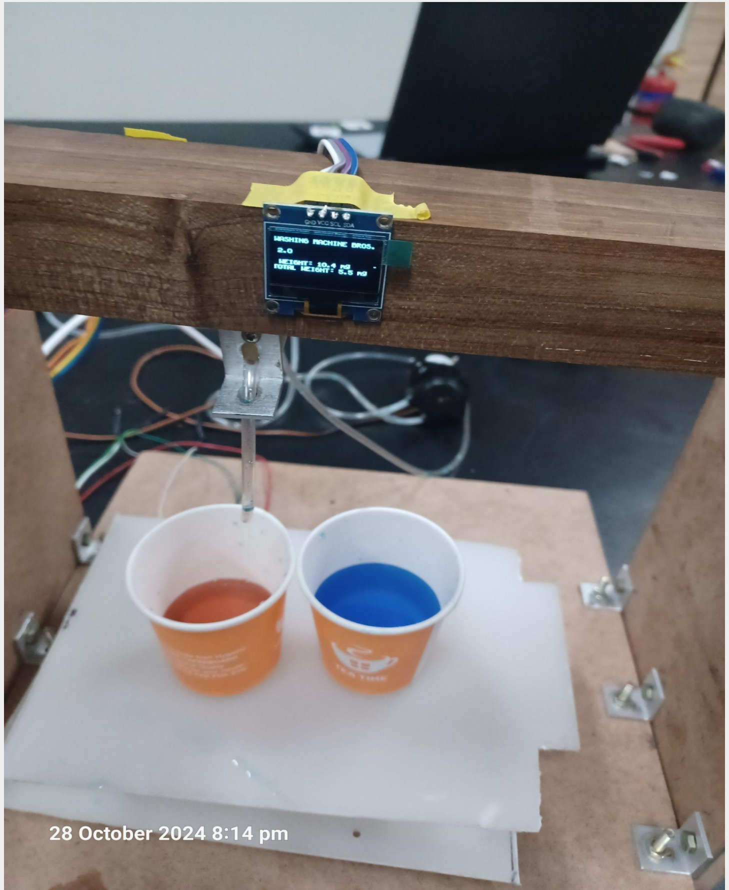
Component selection

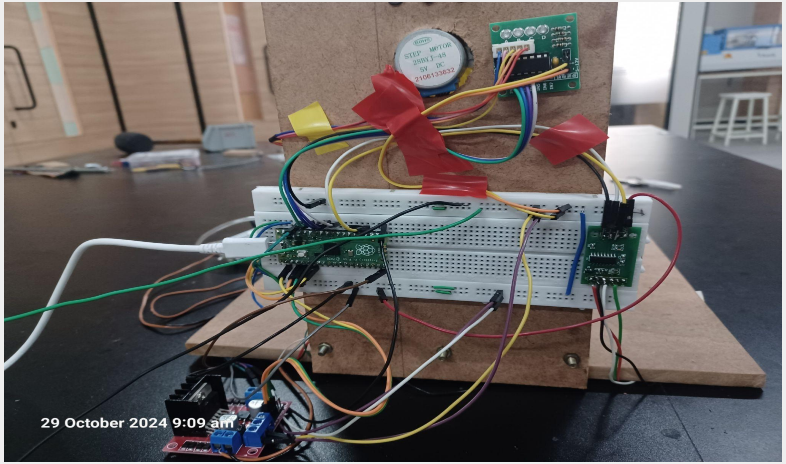
The system is designed for dispensing two liquids into 2 different containers without cross-contamination.

List of primary components:

- **Control Unit** [Raspberry Pi Pico]
- **Peristaltic Pump Module**
- **Load Cell Module**
- **Linear Actuator (Stepper Motor) Module [** 28BYJ-48 Stepper Motor and ULN2003]
- **I2C OLED Display Module**
- **Pinch Valves (Servo Motors) Module** [SG90 servo motors]

Testing the system





Code Explanation

The code has been separated in 2 files:

- **Library File**: Containing all the necessary functions and definitions.
- **Main(driver) File:** Where the library functions are imported and executed to perform the intended tasks.

This modular approach allows for better code organization, re-usability, and easier debugging and ensures that the primary logic remains concise and focused, while the underlying functionality is encapsulated within the library.

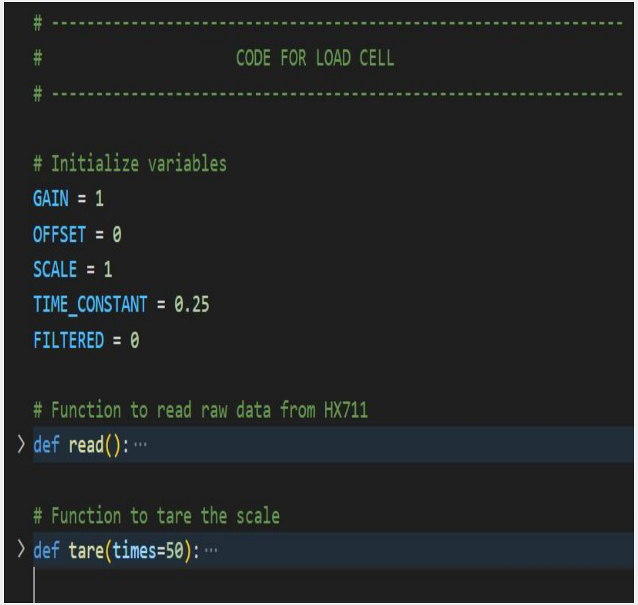
Library File

In this file we initialize the necessary setup variables and define functions for different components:

- **Load Cell:** Initialize and read raw data from the load cell sensor (HX711) to measure weight.
- **Pinch Valves (Servos):** Uses PWM to control SG90 servos that open and close tubes for specific valves.
- **Stepper Motor:** A full step sequence drives a 28BYJ-48 stepper motor for positioning.
- **OLED Display:** Initialize an OLED screen and provides functionality to display text using a custom font bitmap.
- **Peristaltic Pump:** Initialize minimum and maximum duty cycle and defining forward and backwards function to move the liquid.

Load Cell

1. **Variable Initialization**: Sets initial values for `GAIN`, `OFFSET`, and `SCALE`, which are calibration parameters, and `TIME_CONSTANT` and `FILTERED` for signal filtering.
2. `read()` Function:
   - **Purpose**: Reads raw data from the HX711 load cell amplifier.
   - **Process**:
     - **Setup**: Disables interrupt requests (IRQs) momentarily to capture data accurately.
     - **Data Collection**: Waits for the sensor to be ready and then shifts in 24 bits of data, applying the configured



Pinch Valves(Servos)

1. **PWM Frequency Setup**: Sets the PWM frequency for pwmRED and pwmBLUE to 50 Hz, matching the SG90 servo specifications.
2. **close_tube() Function**:
   - **Purpose**: Closes the tube by positioning the servo to the center.
   - **Action**: Sets servo duty to 3276, centering the pinch valve.
3. **open_tube() Function**:
   - **Purpose**: Opens the tube by rotating the servo to a 90-degree angle.
   - **Action**: Sets servo duty to 6553, adjusting the pinch valve to an open

```
# ---------------------------------------------------------------
#                 CODE FOR THE PINCH-VALVE SERVOS
# ---------------------------------------------------------------


# Setting the frequency of PWM signal as per SG90 datasheet.
pwmRED.freq(50)
pwmBLUE.freq(50)

# Defining functions to open and close the tubes with reservoir color as input
def close_tube(pwm):
    #center position
    pwm.duty_u16(3276)

def open_tube(pwm):
    #90 degree angle
    pwm.duty_u16(6553)
```

Stepper Motor

1. **Full-Step Sequence Setup**: Defines a sequence for the 28BYJ-48 stepper motor to achieve full steps, enabling precise control.
2. **Steps per Revolution**: Sets the number of steps (2048) needed for a full 360-degree rotation.
3. **set_step() Function**:
   - **Purpose**: Activates the motor coils based on input, controlling the stepper motor position.
   - **Action**: Sets values for each coil (`IN1` to `IN4`) according to the current step.
4. **rotate_stepper_360() Function**:
   - **Purpose**: Rotates the motor clockwise for the specified number of revolutions.
   - **Process**: Iterates through the full-step

```
# -------------------------------------------------------------------
#                    CODE FOR STEPPER FOR MOVING SPOUT
# -------------------------------------------------------------------
# We have used 28BYJ-48 stepper motor

# Since we are only using full rotation, only full step sequence is written
full_step_sequence = [
    [1, 1, 0, 0],  # Step 1
    [0, 1, 1, 0],  # Step 2
    [0, 0, 1, 1],  # Step 3
    [1, 0, 0, 1],  # Step 4
]

# Number of steps for full (360 degrees) rotation
steps_per_revolution = 2048

# Function to set the step on the stepper motor
def set_step(p1, p2, p3, p4):···

# Function to move stepper in clockwise direction
def rotate_stepper_360(revolutions, delay_ms = 5):···

# Rotate stepper motor in reverse
def rotate_stepper_reverse(revolutions , delay_ms = 5):···
```

OLED Display

1. **Initialization (init_display)**: Configures OLED settings like contrast, memory mode, and display setup to turn on the screen.
2. **Buffer Setup**: The display buffer holds pixel data, where each byte represents 8 vertical pixels for the 128x64 display.
3. **Core Functions**:
   - write_cmd(cmd) and write_data(data): Send commands and data to control OLED operations.
   - clear_display(): Clears the buffer to reset the screen.
4. **Drawing and Text**:
   - set_pixel(x, y, color): Sets individual pixels in the buffer.
   - draw_char(x, y, char): Draws characters using a 5x8 font format.
   - write_on_disp(text): Writes strings to the screen by drawing characters sequentially.
5. **Display Content (show())**: Loads buffer data to the

```python
SET_DISP_OFFSET     = const(0xd3)
SET_COM_PIN_CFG     = const(0xda)
SET_DISP_CLK_DIV    = const(0xd5)
SET_PRECHARGE       = const(0xd9)
SET_VCOM_DESEL      = const(0xdb)
SET_CHARGE_PUMP     = const(0x8d)

# OLED dimensions
WIDTH = 128
HEIGHT = 64
ADDR = 0x3c  # I2C address for SSD1306

# Buffer for display, here 1 byte represents every 8 vertical pixels
buffer = bytearray((HEIGHT // 8) * WIDTH)

# Function to send command to OLED
def write_cmd(cmd):···

# Function to send data to OLED
def write_data(data):···

# Initializing the OLED
def init_display():···

# Function to clear the display buffer
def clear_display():···
```

Peristaltic Pump

1. **Duty Cycle Calculation (duty_cycle)**: Calculates the duty cycle based on speed input (0-100%), translating it into a PWM value between min_duty and max_duty.
2. **Motor Direction Control**:
   - forward(speed): Sets motor direction to forward and adjusts speed with duty_cycle.
   - backward(speed): Sets motor direction to backward with adjustable speed.
   - stop(): Halts the motor by setting duty cycle to zero and turning off both control pins.

```python
# -----------------------------------------------------------------
#                   CODE FOR PERISTALTIC PUMP
# -----------------------------------------------------------------

# Set min and max duty cycle values
min_duty = 15000
max_duty = 65535

# Function to set the duty cycle based on speed
def duty_cycle(speed):
    if speed <= 0 or speed > 100:
        return 0
    else:
        return int(min_duty + (max_duty - min_duty) * (speed / 100))

# Function to move the motor forward
def forward(speed):
    enable.duty_u16(duty_cycle(speed))
    pin1.value(1)
    pin2.value(0)

# Function to move the motor backward
def backwards(speed):
    enable.duty_u16(duty_cycle(speed))
    pin1.value(0)
    pin2.value(1)

# Function to stop the motor
def stop():
```

Main File-1

Initial Setup and Calibration

1. **Display Initialization**: Sets up and clears the display.
2. **Pinch Valves**: Initializes valves to the closed position for red and blue cups.
3. **Load Cell Calibration**:
   - **Tare**: Sets the current offset as baseline.
   - **Weight Measurement**: Reads weight 10 times to calculate the average, accounting for calibration constants.
4. **Cycle Start**:
   - **Valve Control**: Opens red valve.
   - **Motor Activation**: Runs forward at full speed for initial priming

```python
# Declaring variable to store total weight of liquids
total_weight = 0

# Initialize display and clear it
init_display()
clear_display()

# Initializing Pinch Valves to closed position
close_tube(pwmRED)
close_tube(pwmBLUE)

# CODE TO CALIBRATE THE LOAD CELL
c = tare()         # Tare the scale to set the current offset

weights = []  # list to store weights

for _ in range(10):  # measure weight 10 times...

# finding average weight
weight = ((sum(weights) / len(weights)) - c) * (-0.517)     # scaling factor and co
print(f"Average Weight: {weight:.2f} grams", end="    \r")  # printing average weig

# START executing task for 5 cycles
for _ in range (5):

    open_tube(pwmRED)

    # Run at full speed to give initial speed
    backwards(100)
    time.sleep(2)
```

Main File-2

Task Execution and Weight Management

1. **Liquid Dispensing Loop**:
   - Pumps liquid until reaching weight thresholds (4.8g for red, 9.8g for blue).
   - Displays real-time weight on the screen.
2. **Cross-Contamination Avoidance**:
   - Reverses pump direction at varied speeds between cups.
3. **Cup Transition**:
   - **Stepper Motor Movement**: Rotates between red and blue cups.
4. **End-of-Cycle Updates**:
   - Adds dispensed weight to `total_weight`.
   - Tares load cell and repeats process.

```
while (weight < 4.8):  # Giving value as 4.8 instead of 5 to avoid overshoot

    weights = []

    backwards(50)

    for _ in range(10):  # measure weight 10 times
        raw_wt = read() * 0.001
        weights.append(raw_wt)
        time.sleep(0.001)  # small delay between measurements

    weight = ((sum(weights) / len(weights)) - c) * (-0.517)  # calculate average

    write_on_disp("WASHING MACHINE BROS. 2.0
    show()

# Running peristaltic pump in reverse at different speeds to avoid cross-contamina
forward(100)
time.sleep(25)
forward(40)
time.sleep(10)
forward(100)
time.sleep(25)
stop()

# Change pinch valve configuration for the Blue cup
close_tube(pwmRED)
open_tube(pwmBLUE)
```