ME 100 FINAL PROJECT

Robotic Shoe Remover and Shelfer

Department of Mechanical Engineering

University of California, Berkeley

ME 100: Electronics for the Internet of Things

Professor George Anwar

Group 27 – The Bay: Aarush Panda, Andrew Maramag, Ricky Young

December 17, 2025

Abstract

Removing shoes upon entering a home presents challenges for many people, whether due to inconvenience, limited mobility, or difficulty bending down. Existing solutions such as shoe horns, adjustable footwear, or assisted help all offer partial relief but lack a fully hands-free, autonomous option. This paper outlines the development and testing of a robotic shoe-removal and shelving system designed to provide a smooth, touch-free experience. The following sections detail the design, implementation, and testing of a prototype that integrates load cells, linear actuators for shoe stabilization, and a piezoelectric safety indicator within a shoe entry box. Once the shoe is successfully released, an ESP32 channel triggers an omnidirectional robot car, powered by a second ESP32 which navigates using an IMU and motor control to retrieve the shoe with an infrared-guided arm. The robot then transports and places the shoe onto a designated shelf. Ultrasonic sensing within the activation box provides relevant positioning data that enhances reliability and overall system robustness.

Table of Contents

1.0 Introduction

The development of our automated shoe-removal and shelving system in this project was motivated by a common, yet frequently overlooked inconvenience in daily life: removing and storing shoes cleanly and efficiently. In homes and apartments, removing shoes is often encouraged, or even required to maintain cleanliness. Existing shoe removal solutions rely heavily on manual effort, such as shoe horns. This can be uncomfortable, inconvenient, and even inaccessible for individuals with mobility issues or who have their hands occupied. This project was conceived to address these challenges through an integrated electromechanical solution for both shoe removal and storage.

Our design focuses on delivering a fully functional, end-to-end solution combining a step-in shoe box mechanism with a mobile robot shelving system. Leveraging sensors, actuators, and IoT-enabled communication via dual ESP32 microcontrollers, our system is capable of detecting user input in the form of stepping on our shoe platform, safely removing footwear via a robotic arm, and autonomously transporting and shelving shoes with no manual intervention needed.

2.0 Design Process

Our team split the project into three parts. Part one consisted of designing a device to sense the presence of a shoe, an automatic process to take it off the user's foot, and send a signal for shelving. Part two is a robot to take the shoe, pick it up, move to the shelf, and place it down. Part three was integrating them together, making sure there were no physical or communicational errors. The figures below show the initial sketches of each part as well as the final prototype.

2.1 Initial Concept and Design

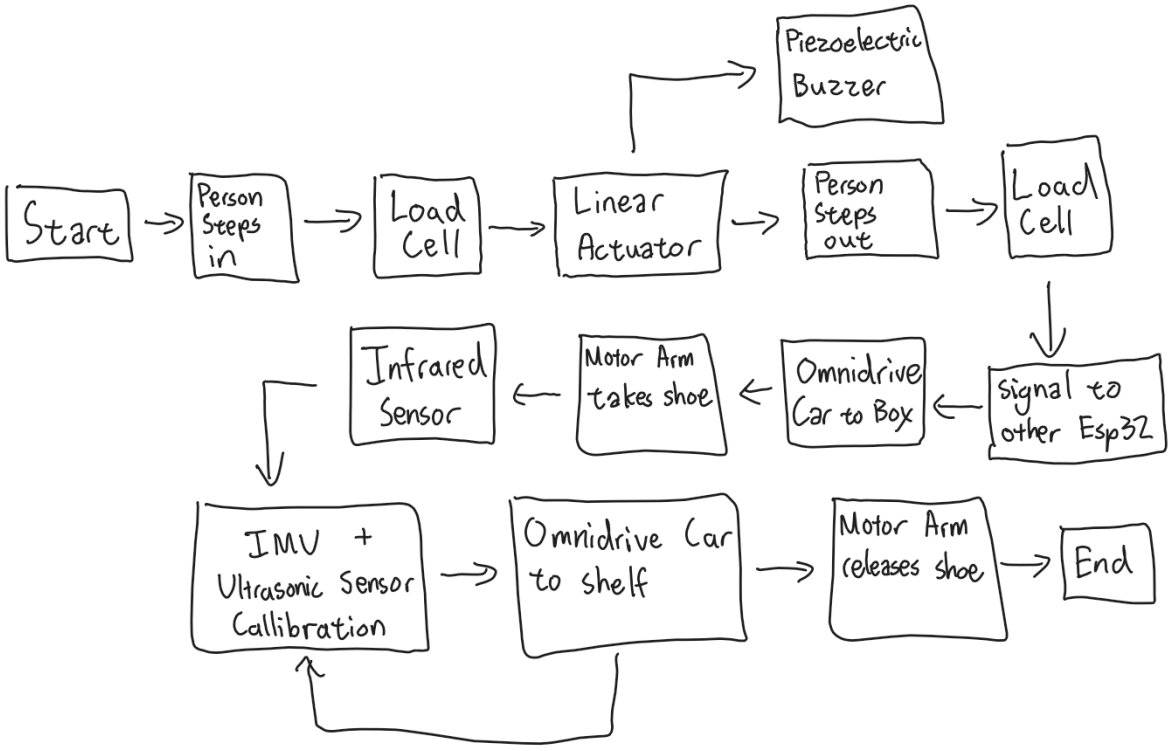


Figure 1: Initial circuit diagram for shoe remover system concept

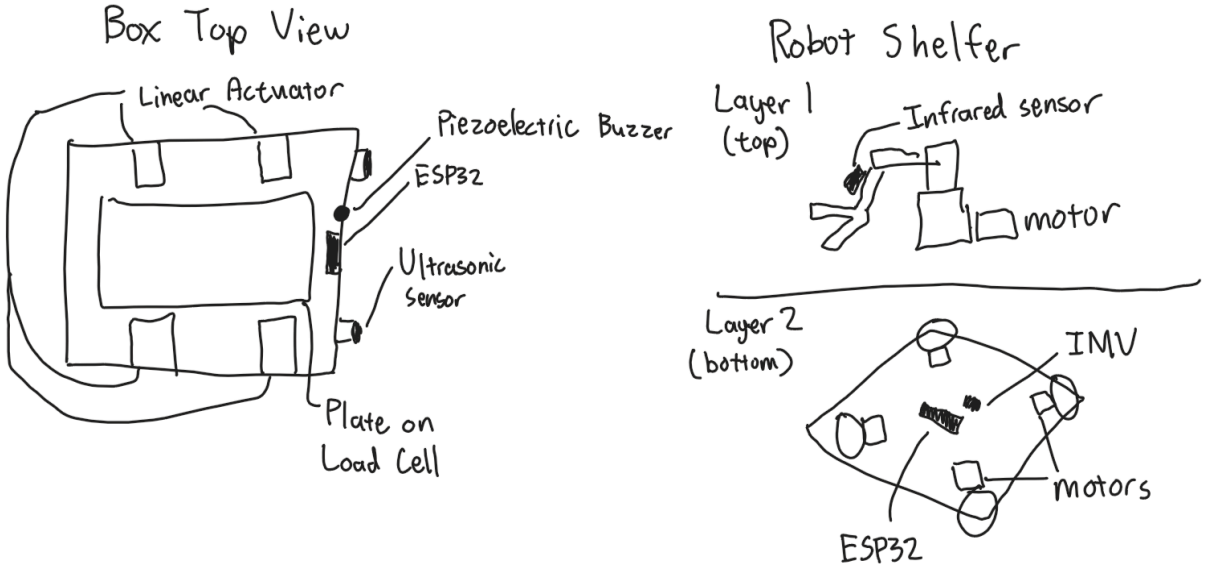


Figure 2: Initial sketched idea for shoe remover system concept

2.2 Materials and Component Selection

During this phase, we selected electronic components based on their functionality and compatibility with our system design.

2.2.1 ESP32

A small microcontroller providing the necessary compute to process positional data and control other electronic components.

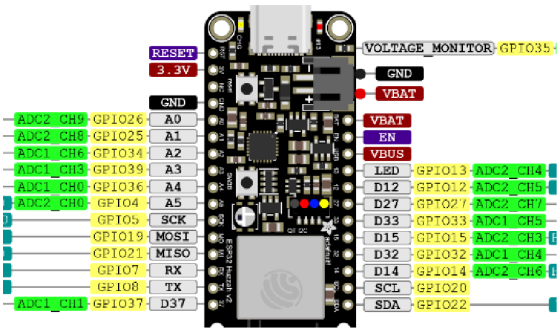


Figure 3: ESP32

2.2.2 Piezo Electronic Buzzer Alarm

The buzzer functions as an indicator that the shoe removal/racking process has begun/completed, so the user knows when to safely step out of their shoe.

Figure 4: Buzzer

2.2.3 HX-711 Load Cell Amplifier

An analog-to-digital converter specifically designed for load cell applications. Allows ESP32 to accurately measure applied force from the load cell.

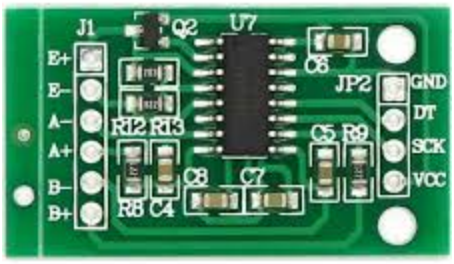


Figure 5: HX711 Load Cell Amplifier

2.2.4 Load Cell (10 KG)

A force sensor used to measure the weight applied during operation. Detects user presence via a shoe placed on the base plate.

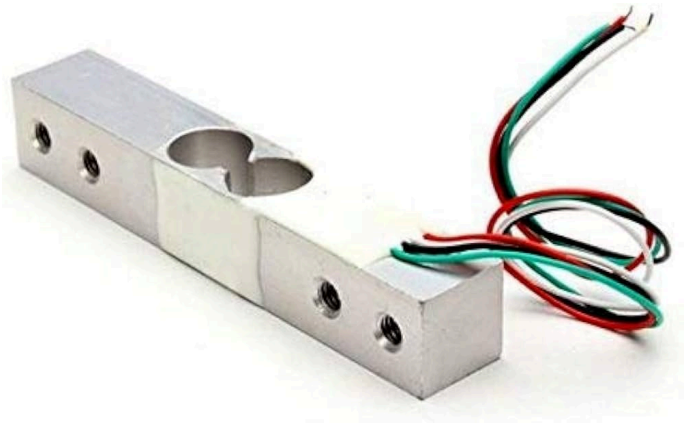


Figure 6: 10 kg Load Cell

2.2.5 BTS7960 43A High Power H-Bridge Motor Driver

High-current H-bridge motor driver capable of controlling speed and direction of high-power DC motors and actuators. Handles the high current requirement demanded by the linear actuator and enables ESP32 to control actuation via power-control signals.

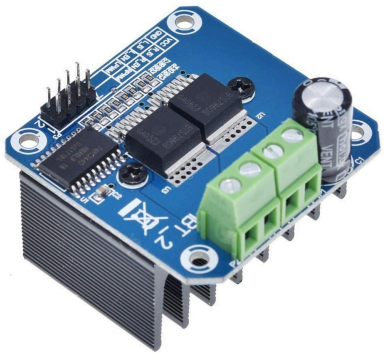


Figure 7: BTS7960 Driver

2.2.6 Linear Actuator (12V)

Provides mechanical motion necessary for shoe removal. Converts electrical energy into linear motion to assist the user in removing their shoe. Motion is regulated via BTS7960 driver and sensor feedback.

Figure 8: 12 Volt Linear Actuator

2.3 Initial Circuit Assembly/Testing

Our initial development stages consisted of assembling basic circuits with our ESP32. Each actuator and sensor was tested individually, ensuring that sensors were functioning properly and that actuators did their intended function. After each component was working, they were combined into their respective circuits.

2.4 Incorporating IMU Position tracking

The robot shelfer uses an IMU for position tracking, since a camera would be expensive and the distances it moves are too small for gps. An IMU reads even the smallest accelerations, so double integration would make it quickly drift from its real location. To compensate for this, we cut off accelerometer readings below a certain threshold, so it would only sense the initial jerk in one direction and estimate the robot's position. This would place the robot close enough to its ideal position that the ultrasonic sensor could line the robot back up for better precision.

2.5 Integration and Finalizing Circuit

Once both circuits performed separately, they were combined using ESP-NOW to perform their tasks in series. This allowed the robot to move only when there was a shoe to move. We decided to put this signal at the end of the piston retraction so the user would be able to move away without worrying about robot interference.

2.6 Component Design

Our team designed our components using Onshape and SolidWorks.

2.6.1 Base Plate for Box

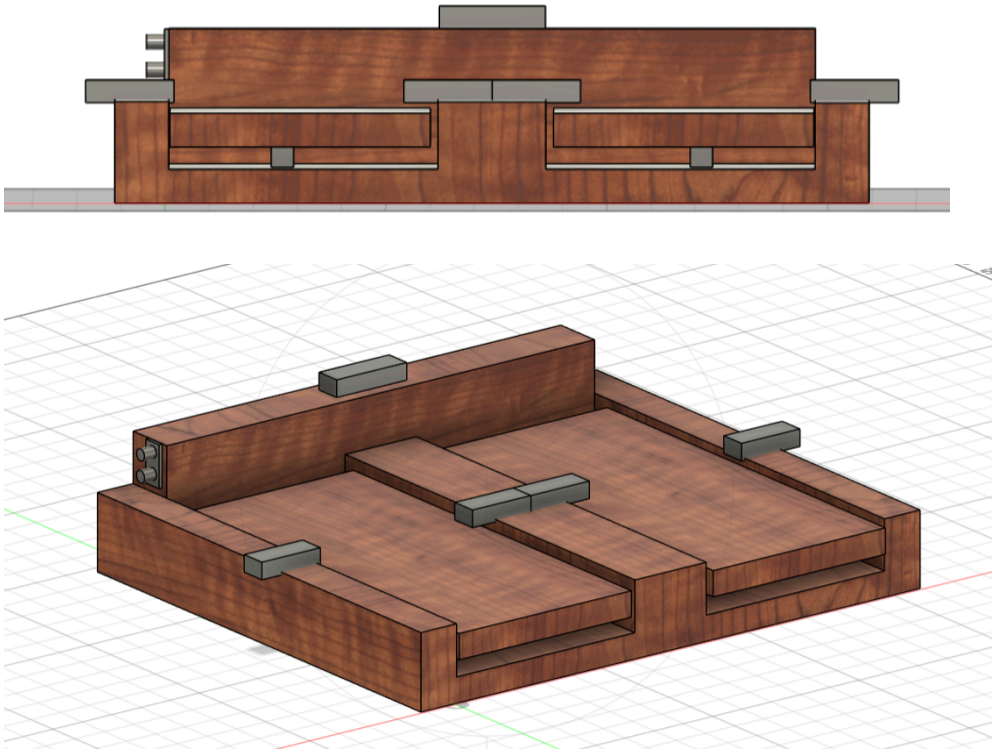


Figure 9, 10: Front and Top View (Respectively) of Shoe Plate

2.6.2 Car

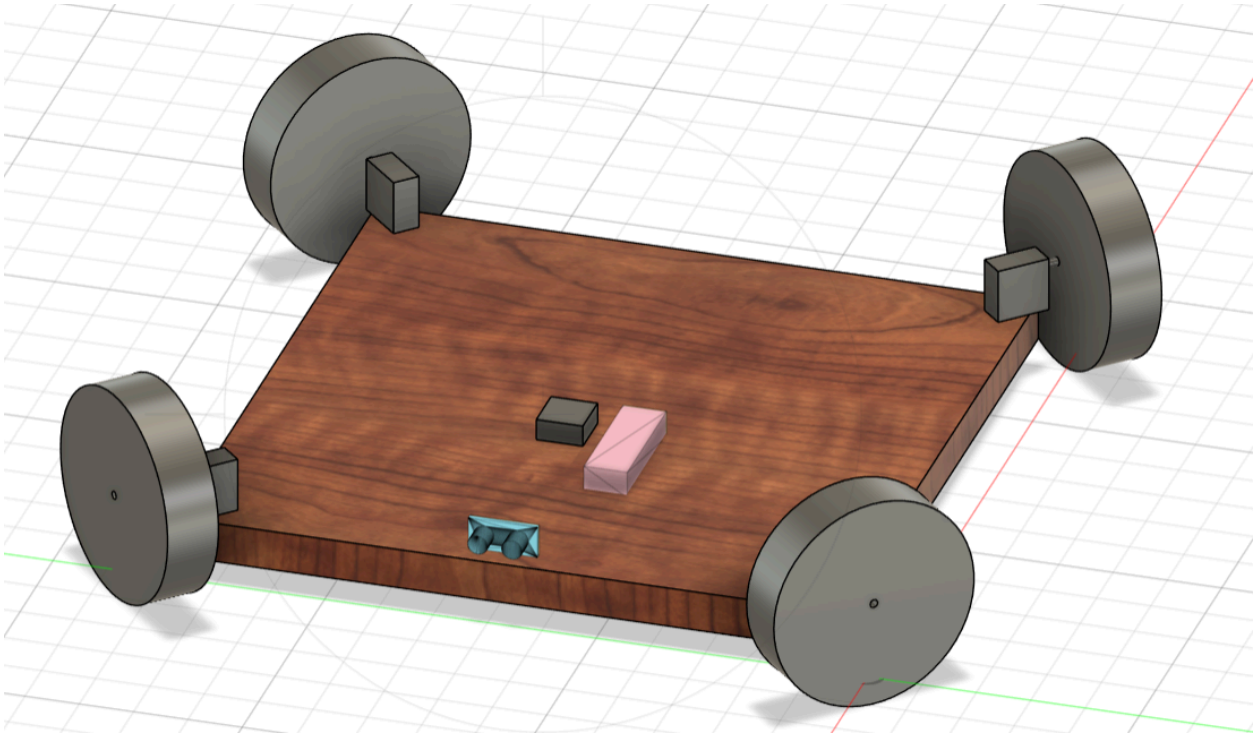
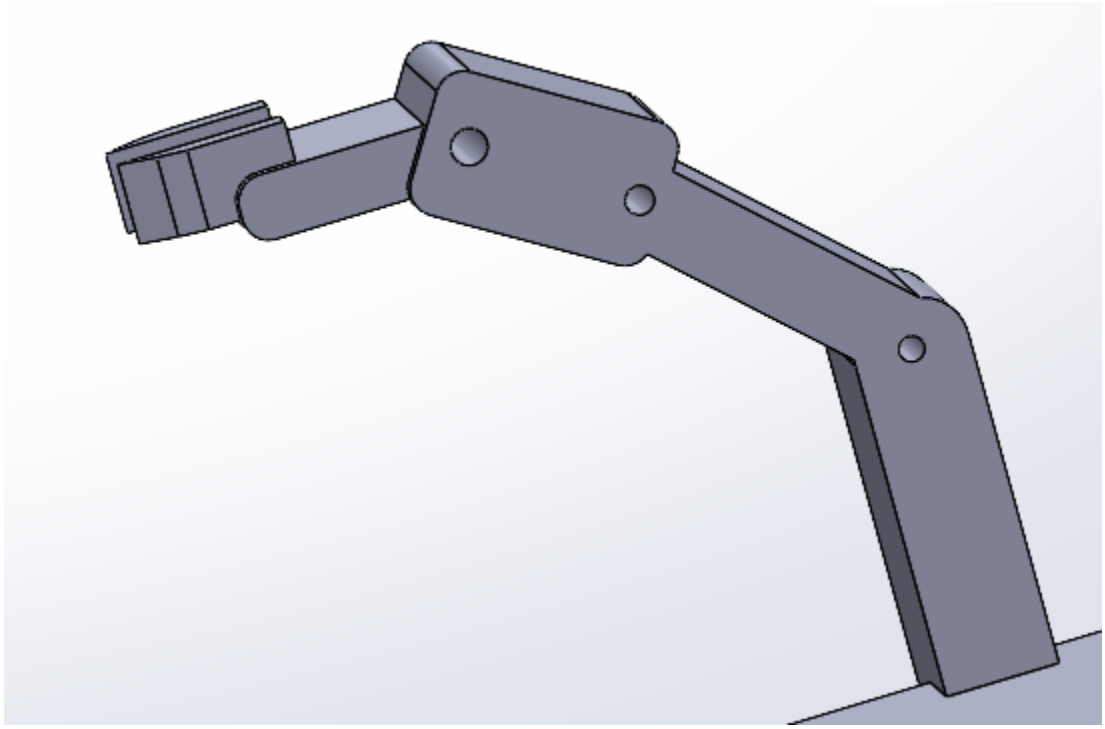


Figure 11: Robot Car

2.6.3 Arm



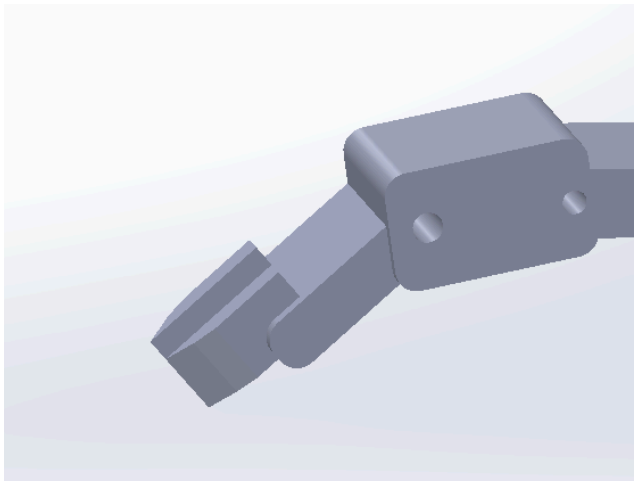


Figure 12, 13: Gripper Arm Design

3.0 Micropython Code

3.1 ESP32 Sender

3.1.1 Main.py for Sender ESP32 on shoe plate

Link:
https://drive.google.com/file/d/1_fKUCXLf57mwft8fHBil3-bp5rxhn6Wy/view?usp=sharing

The major components of this main code in chronological order include taring our load cell, causing actuation to start once a threshold of over an absolute value of 8 kilograms has been reached (represents a shoe), beeping once its safe for the user to remove their foot from the base plate (once actuation has finished expanding and before actuation will begin retracting), and then sending a signal to the ESP32 Receiver so that it can begin its sequence.

```python
from machine import Pin, PWM
from hx711 import HX711
import time
import network
import espnow
import actuator

LOADCELL_DOUT_PIN = 32
LOADCELL_SCK_PIN  = 15
BUZZER_PIN = 33

THRESHOLD_KG = 8.0
BEEP_INTERVAL = 3.0
BEEP_DURATION = 1.0
Calibration = 100
CHANNEL = 1
PEER = b'\x14\x2b\x2f\xae\xbd\xfc'

scale = HX711(Pin(LOADCELL_DOUT_PIN, Pin.IN), Pin(LOADCELL_SCK_PIN, Pin.OUT))
scale.set_scale(Calibration)


buzzer_pin = Pin(BUZZER_PIN, Pin.OUT, value=1)
buzzer = PWM(buzzer_pin)
buzzer.freq(2000)
buzzer.duty(0)

sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.disconnect()
sta.config(channel=CHANNEL)

e = espnow.ESPNow()
e.active(True)
e.add_peer(PEER)

print("Taring...")
time.sleep(2)
scale.tare()

last_beep_time = None
beep_start_time = None
buzzing = False

# Trigger state
over_threshold_since = None
actuated_this_event = False

def timed_beeps(duration):
    buzzer.freq(2000)
    buzzer.duty(512)
    time.sleep(duration)
    buzzer.duty(0)

try:
    while True:
        weight = scale.get_units(10)
        now = time.time()

        if abs(weight) > THRESHOLD_KG:
            if over_threshold_since is None:
                over_threshold_since = now
                actuated_this_event = False
                last_beep_time = None
                buzzing = False
                buzzer.duty(0)
```

```
                last_beep_time = None
                buzzing = False
                buzzer.duty(0)

            if (not actuated_this_event) and (now - over_threshold_since) >= 1.0:
                actuated_this_event = True
                #print("Actuation started - extension")
                #timed_beeps(2.0)
                actuator.retract_full()
                time.sleep(6)
                #print("Actuation paused - extension to retraction")
                timed_beeps(2.0)
                actuator.extend_full()
                #print("Actuation finished")
                #timed_beeps(2.0)

                try:
                    e.send(PEER, b"ON")
                except Exception as err:
                    print("ESP-NOW send error:", err)
#
#
#             if last_beep_time is None or (now - last_beep_time) >= BEEP_INTERVAL:
#                 buzzer.freq(2000)
#                 buzzer.duty(512)
#                 buzzing = True
#                 last_beep_time = now
#                 beep_start_time = now
#                 print("BEEP")
#
#             if buzzing and beep_start_time is not None and (now - beep_start_time) >= BEEP_DURATION:
#                 buzzer.duty(0)
#                 buzzing = False
#
#         else:
#             over_threshold_since = None
#             actuated_this_event = False
#             last_beep_time = None
#             beep_start_time = None
#
#             if buzzing:
#                 buzzer.duty(0)
#                 buzzing = False
#
#             print("OFF")
#
        print("Weight:", weight, "kg")
        time.sleep(0.2)

finally:
    try:
        buzzer.duty(0)
    except:
        pass

    print("finished")
    try:
        actuator.retract_full()
    except Exception as e2:
        print("Retract failed:", e2)

```

3.1.2 actuator.py for Sender ESP32 on shoe plate

```
from machine import Pin, PWM
import time

rpwm = PWM(Pin(12), freq=1000, duty=0)
lpwm = PWM(Pin(27), freq=1000, duty=0)
SPEED = 700

def stop():
    rpwm.duty(0)
    lpwm.duty(0)
def extend_full():
    lpwm.duty(0)
    rpwm.duty(SPEED)
    time.sleep(10)
    stop()

def retract_full():
    rpwm.duty(0)
    lpwm.duty(SPEED)
    time.sleep(10)
    stop()
if __name__ == "__main__":
    extend_full()
    time.sleep(2)
    retract_full()
```

3.1.3 hx711.py for Sender ESP32 on shoe plate

```
from machine import Pin
import time

class HX711:
    def __init__(self, dout, pd_sck, gain=128):
        self.dout = dout
        self.pd_sck = pd_sck
        self.gain = gain
        self.offset = 0
        self.scale = 1
        self.pd_sck.value(0)
        self.set_gain(gain)

    def set_gain(self, gain):
        if gain == 128:
            self.gain = 1
        elif gain == 64:
            self.gain = 3
        elif gain == 32:
            self.gain = 2
        time.sleep_ms(10)

    def is_ready(self):
        return self.dout.value() == 0
```

```
26      def read(self):
27          while not self.is_ready():
28              pass
29
30          data = 0
31          for _ in range(24):
32              self.pd_sck.value(1)
33              data = (data << 1) | self.dout.value()
34              self.pd_sck.value(0)
35
36          for _ in range(self.gain):
37              self.pd_sck.value(1)
38              self.pd_sck.value(0)
39
40          if data & 0x800000:
41              data |= ~0xffffff
42
43          return data
44
45      def tare(self, times=15):
46          total = 0
47          for _ in range(times):
48              total += self.read()
49          self.offset = total // times
50
```

```
51      def read_average(self, times=10):
52          total = 0
53          for _ in range(times):
54              total += self.read()
55          return total // times
56
57      def get_value(self, times=10):
58          return self.read_average(times) - self.offset
59
60      def set_scale(self, scale):
61          self.scale = scale
62
63      def get_units(self, times=10):
64          return self.get_value(times) / self.scale
65
```

3.2 ESP32 Receiver

3.2.1 Main.py for Receiver ESP32 on robot arm

Link:

https://drive.google.com/file/d/1-lEL5a-SwHYX7XLT96QwnAhe0WbHB_Qe/view?usp=sharing

The major components of this main code in chronological order include printing a message to indicate the ESP32 is on and waiting for the sender's message, printing a message received statement which begins the inflow of IMU data, setting the ultrasonic sensor to start reading once the IMU data reaches a manually set threshold of 0.5, and finally activating the servo once the ultrasonic sensor reaches a distance of about 8 centimeters from a target (such as a wall).

```python
import network
import espnow
import time
import math

#ESP NOW
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.disconnect()
sta.config(channel=1)

e = espnow.ESPNow()
e.active(True)

#Ultrasonic sensor
from hcsr04 import HCSR04
from time import sleep

sensor = HCSR04(trigger_pin=7, echo_pin=8, echo_timeout

#Servo
from machine import Pin, PWM

servo = PWM(Pin(12))
servo.freq(50)

def set_angle(angle):
    min_duty = 26
    max_duty = 123
    angle = max(0, min(180, angle))
    duty = int(min_duty + (angle / 180) * (max_duty - m
    servo.duty(duty)
    print(f"Angle: {angle}° | Duty: {duty}")


from lsm6dsox import LSM6DSOX

from machine import Pin, SPI, I2C

lsm = LSM6DSOX(I2C(0, scl=Pin(14), sda=Pin(22)))

print("Receiver ready. Waiting for messages...")

seen_on_before = False
active = False
last_on_ms = None

TIMEOUT_SEC = 4.5

pos_x = 0.0
pos_y = 0.0
pos_z = 0.0
vel_x = 0.0
vel_y = 0.0
vel_z = 0.0
prev_ax = 0.0
prev_ay = 0.0
prev_az = 0.0
last_time = None

bias_ax = 0.0
bias_ay = 0.0
bias_az = 0.0

def calibrate(samples=200):
    print("Calibrating IMU... keep still")
```

```
    sum_ax = 0.0
    sum_ay = 0.0
    sum_az = 0.0

    for _ in range(samples):
        ax, ay, az = lsm.accel()
        sum_ax += ax
        sum_ay += ay
        sum_az += az
        time.sleep_ms(5)

    bias_x = sum_ax / samples
    bias_y = sum_ay / samples
    bias_z = sum_az / samples
    print("Bias calibrated:", bias_x, bias_y, bias_z)
    return bias_x, bias_y, bias_z

def get_accel_cm():
    ax, ay, az = lsm.accel()

    ax -= bias_ax
    ay -= bias_ay
    az -= bias_az
    ax_cm = ax * 9.80665 / 1000 * 100
    ay_cm = ay * 9.80665 / 1000 * 100
    az_cm = az * 9.80665 / 1000 * 100

    gx, gy, gz = lsm.gyro()

    return ax_cm, ay_cm, az_cm

def update_position():
    global pos_x, pos_y, pos_z, vel_x, vel_y, vel_z, last_time

    now = time.ticks_ms()
    if last_time is None:
        last_time = now
        return pos_x, pos_y, pos_z

    dt = time.ticks_diff(now, last_time) / 1000.0
    last_time = now

    ax, ay, az = get_accel_cm()

    drift = 1
    ax = 0 if abs(ax) < drift else ax
    ay = 0 if abs(ay) < drift else ay
    az = 0 if abs(az) < drift else az

#     alpha = 0.95
#     global prev_ax, prev_ay, prev_az
#     ax = alpha * prev_ax + (1 - alpha) * ax
#     ay = alpha * prev_ay + (1 - alpha) * ay
#     az = alpha * prev_az + (1 - alpha) * az
#     prev_ax, prev_ay, prev_az = ax, ay, az

    if abs(ax) > 0:
        vel_x += ax * dt
    if abs(ay) > 0:
        vel_y += ay * dt
    if abs(az) > 0:
        vel_z += az * dt

    print("AX: ", ax, "AY: ", ay)
    pos_x += vel_x * dt
```

```
    pos_x += vel_x * dt
    pos_y += vel_y * dt
    pos_z += vel_z * dt

    return pos_x, pos_y, pos_z


def start():

    #IMU -> motors
    print("Waiting for IMU movement: -10 cm back AND +10 cm right...")

    while True:
        x, y, z = update_position()
        print("X:", x, "Y:", y)

        if abs(x) >= 0.5 and abs(y) >= 0.5:
            print("Reached 10 cm back and 10 cm right.")
            break

        #code to move motors here


    #ultrasonic sensor -> servo
    print("Waiting for ultrasonic distance < 8 cm...")

    while True:
        d = sensor.distance_cm()
        # print("Ultrasonic:", d)
        if d < 8:
            print("Ultrasonic detected < 8 cm.")
            break
        time.sleep(0.05)


    print("Servo: 150°")
    set_angle(150)
    time.sleep(2)

    print("Servo: 90°")
    set_angle(90)

    pos_x = pos_y = vel_x = vel_y = 0
    last_time = time.ticks_ms()

    print("Waiting for IMU movement: 10 cm left...")

    while True:
        x, y, z = update_position()
        print("X:", x, "Y:", y)

        if y <= -1:
            print("Reached 10 cm left.")
            break

        time.sleep_ms(20)


    print("Servo: 150°")
    set_angle(150)

    print("Sequence complete.")

```

```
193
194 while True:
195     host, msg = e.irecv(0)
196     now_ms = time.ticks_ms()
197
198 #     start()
199
200     if msg:
201         s = ""
202         try:
203             s = msg.decode("utf-8").strip()
204         except:
205             pass
206
207         if s == "ON":
208             seen_on_before = True
209             active = True
210             last_on_ms = now_ms
211             print("ON received from", host)
212
213     if active and last_on_ms is not None:
214         if time.ticks_diff(now_ms, last_on_ms) > int(TIMEOUT_SEC * 1000):
215             active = False
216             print("Beeping stopped (timeout). Load removed.")
217             start()
218
```

3.2.2 lsm6dsox.py from Lab 7 Part 1

Link:

https://drive.google.com/file/d/1vgK9JACSthi81kn3UvMj3nmE6r0JRSZ_/view?usp=sharing

3.2.3 hcsr04.py from Lab 7 Part 1

```python
import machine, time
from machine import Pin

__version__ = '0.2.0'
__author__ = 'Roberto S[illegible]chez'
__license__ = "Apache License 2.0. https://www.apache.org/licenses/LICENSE-2.0"

class HCSR04:

    # echo_timeout_us is based in chip range limit (400cm)
    def __init__(self, trigger_pin, echo_pin, echo_timeout_us=500*2*30):

        self.echo_timeout_us = echo_timeout_us
        # Init trigger pin (out)
        self.trigger = Pin(trigger_pin, mode=Pin.OUT, pull=None)
        self.trigger.value(0)

        # Init echo pin (in)
        self.echo = Pin(echo_pin, mode=Pin.IN, pull=None)

    def _send_pulse_and_wait(self):
        """
        Send the pulse to trigger and listen on echo pin.
        We use the method `machine.time_pulse_us()` to get the microseconds until the echo is received.
        """
        self.trigger.value(0) # Stabilize the sensor
        time.sleep_us(5)
        self.trigger.value(1)
        # Send a 10us pulse.
        time.sleep_us(10)
        self.trigger.value(0)
        try:
            pulse_time = machine.time_pulse_us(self.echo, 1, self.echo_timeout_us)
            return pulse_time
        except OSError as ex:
            if ex.args[0] == 110: # 110 = ETIMEDOUT
                raise OSError('Out of range')
            raise ex

    def distance_mm(self):
        """
        Get the distance in milimeters without floating point operations.
        """
        pulse_time = self._send_pulse_and_wait()

        # To calculate the distance we get the pulse_time and divide it by 2
        # (the pulse walk the distance twice) and by 29.1 becasue
        # the sound speed on air (343.2 m/s), that It's equivalent to
        # 0.34320 mm/us that is 1mm each 2.91us
        # pulse_time // 2 // 2.91 -> pulse_time // 5.82 -> pulse_time * 100 // 582
        mm = pulse_time * 100 // 582
        return mm

    def distance_cm(self):
        """
        Get the distance in centimeters with floating point operations.
        It returns a float
        """
        pulse_time = self._send_pulse_and_wait()

        # To calculate the distance we get the pulse_time and divide it by 2
        # (the pulse walk the distance twice) and by 29.1 becasue
        # the sound speed on air (343.2 m/s), that It's equivalent to
        # 0.034320 cm/us that is 1cm each 29.1us
        cms = (pulse_time / 2) / 29.1
        return cms
```

3.2.4 imutest.py

```
import time
from lsm6dsox import LSM6DSOX

from machine import Pin, SPI, I2C
# Init in I2C mode.
lsm = LSM6DSOX(I2C(0, scl=Pin(14), sda=Pin(22)))

while (True):
    print('Accelerometer: x:{:>8.3f} y:{:>8.3f} z:{:>8.3f}'.format(*lsm.accel()))
    print('Gyroscope:     x:{:>8.3f} y:{:>8.3f} z:{:>8.3f}'.format(*lsm.gyro()))
    print("")
    time.sleep_ms(100)

```

3.2.5 macadress.py from Lab 7 Part 2

4.0 Challenges

We encountered various challenges throughout our design and development process, some big and some small. We were able to address the majority of challenges through root cause analysis to determine effective solutions, but were unable to address others due to limited resources and other tertiary constraints. The main challenges we encountered are in the sections below.

4.1 Code Bugs

In our logic for the robot positioning, the IMU would only read the accelerations past a certain threshold, but since it updated very often, it would only read that threshold acceleration even if the robot was moving faster (because the robot would accelerate less as it reaches its ideal velocity). To compensate for this, we designed the robot to move at a slower speed, high enough to pass the threshold, while still making sure the threshold was large enough to filter out any noise.

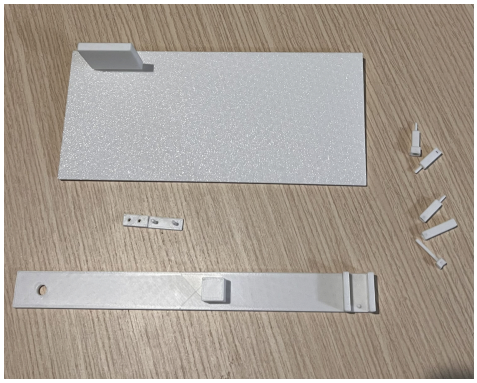
4.2 Hardware & Assembly Issues

We ran into primarily hardware and assembly issues. We were unable to get the wheels of the car fully functional due to challenges in fastening the omnidirectional wheels to our individual motors. Since the robot needed to automatically position, we couldn't use the generic one motor connected to an axle with 2 wheels. Though it would be an easier way to fasten the wheels onto

the car, it would increase any drift already caused by the IMU. We attempted to create a 3D printed hub to attach the wheel directly to the gear shaft of the motor, but we ran into a geometric dimensioning and tolerancing challenge. Since the dimensions of the hub we had created on SolidWorks were fairly small, after 3D printing the parts, the dimensions were very off compared to our small tolerances. We found this was due to many different variables, such as nozzle, material, and printer, but keeping tolerances in mind, we were able to account for these inaccuracies and give room for extra clearance or under sizing certain parts.

4.3 3D printer Issues

The 3D printer we used had a print bed size of 256mm by 256mm, which limited our ability in a few ways. One of which was for the base of the scissor link and making it long enough to be fastened on each end of the linear actuator. This would have been used to turn a one directional force into a squeeze from both sides of the shoe, for easier removal. Since we couldn't use a scissor clamping method, we needed a wall to push the shoe against that would be attached to the base, which led to another challenge from the stress and strain constraints of PLA. We were unable to create a proper base to go with the load cell for shoe removal, as we tried to find methods to print smaller parts and fasten them together, but since the force of the linear actuator squeezing the shoe was significantly large, the fastening methods we tried weren't good enough and snapped. Due to this, we tried to print the base and wall as a single part, but were unable to in the end due to dimensions and other various printing issues.



Various 3d printed parts that could not be used.

5.0 Final Prototype

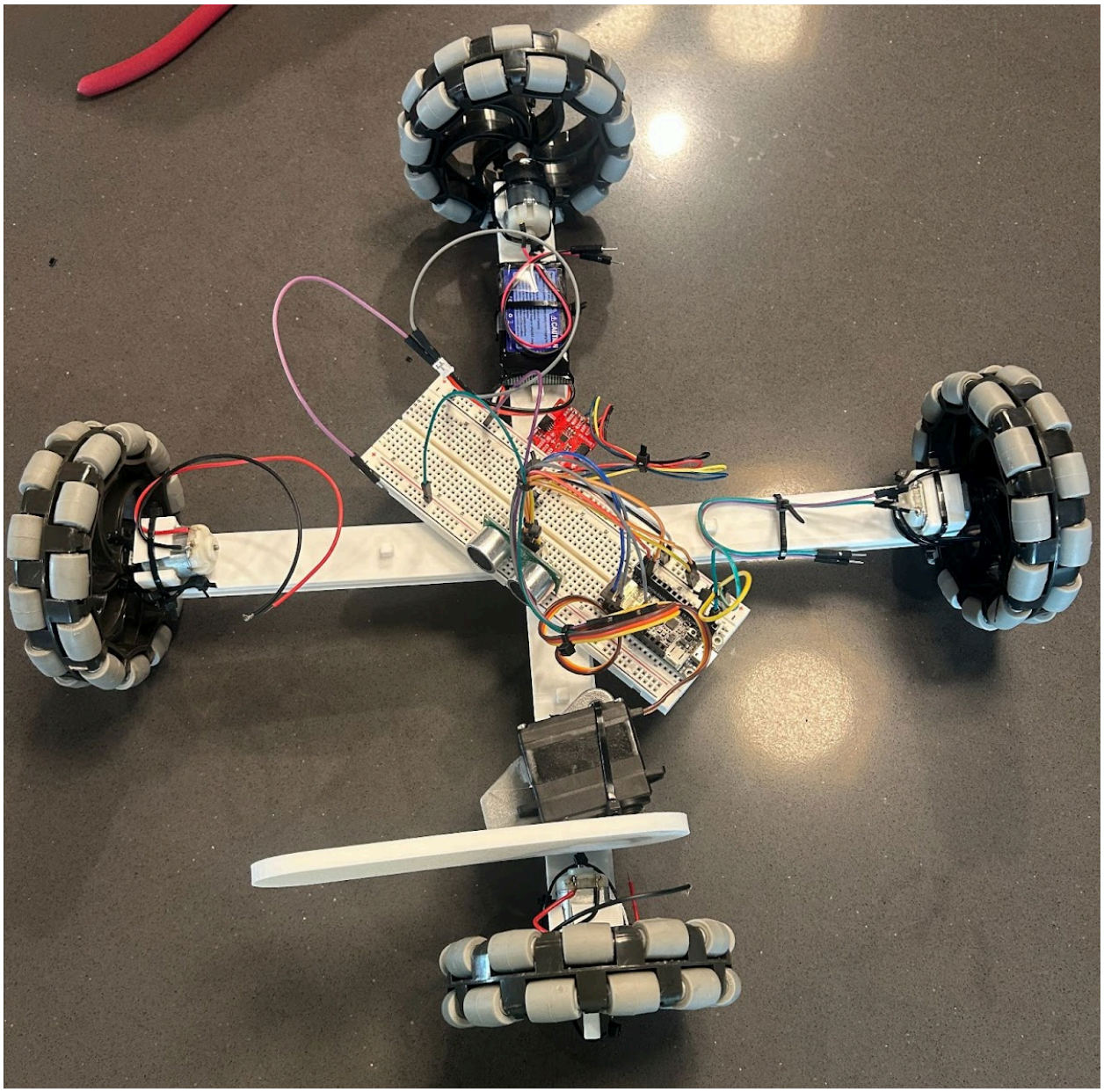


Figure 14: Final prototype of robot shelfer

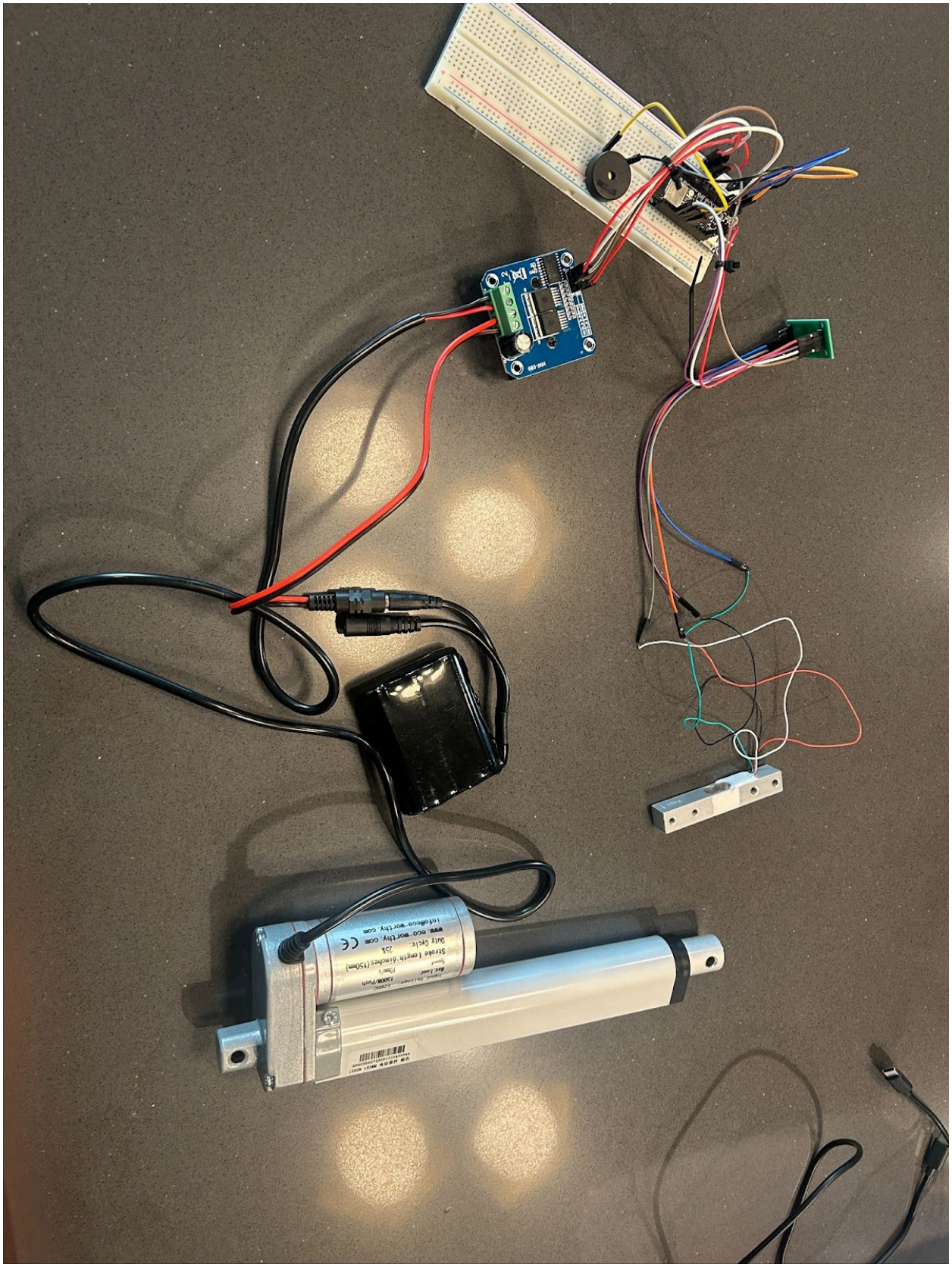


Figure 15: Final prototype of shoe box system

6.0 Improvements

To improve our prototype, we would've liked to condense our circuits and components further in order to make it more marketable as a robust product. Another improvement would be simplifying hardware, and streamlining certain sensor aspects of the code.

6.1 Better Arm Design

Our current arm is a simple lever attached to a servo, which scoops the shoe from the inside and places it down by angling downward. With more time and resources, we would create a better design that can lift the shoe past ground level. We had a few designs in mind, such as the CAD of the grabber arm we originally designed, connected to multiple servos or motors to allow for multi-directional travel, allowing the shoe to be placed on higher levels of the shoe rack

6.2 Box design

Our current designs failed because of fragile connections between the load cell and linear actuator. Given better engineering and the right materials, these connections could be strengthened for a more cohesive design.

7.0 Demonstration Video

https://youtu.be/eVaiJQHA-1U

8.0 References

- https://randomnerdtutorials.com/esp32-load-cell-hx711/
- https://esp32io.com/tutorials/esp32-piezo-buzzer
- https://docs.micropython.org/en/latest/library/espnow.html
- https://docs.micropython.org/en/latest/library/machine.PWM.html
- https://microcontrollerslab.com/esp32-esp-now-tutorial-arduino-ide/
- https://esp32io.com/tutorials/esp32-actuator
- https://esp32io.com/tutorials/esp32-ultrasonic-sensor
- https://esp32io.com/tutorials/esp32-servo-motor
- https://esp32io.com/tutorials/esp32-dc-motor