M5 Code and Systems Analysis

By

Zachary Eix

Student, Mechanical Engineering,

Edward E. Whitacre Jr. College of Engineering, Texas Tech University

May 1, 2025

Table of Contents

Introduction

This project, sponsored by L3Harris, focuses on the development of a system capable of accurately measuring the moment of inertia (MOI) of various objects using a trifilar pendulum approach. The goal was to design a self-contained, automated platform that could perform precise mass measurements, apply controlled rotational displacements, record free oscillations, and calculate inertial properties with minimal operator intervention.

The team developed a fully integrated solution combining mechanical hardware, control electronics, and custom Python code. Tension sensors measure the system's mass properties, while a stepper motor and electromagnets perturb the suspended plate in a controlled manner. An inertial measurement unit (IMU) records angular motion, and custom analysis scripts process the collected data to determine the MOI.

This report presents an overview of the system architecture, detailed summaries of the software and hardware components, and a review of the testing campaign carried out to validate system performance. The discussion section will examine the results, system limitations, and challenges encountered during testing, while the conclusion will summarize key findings and outline possible areas for future improvement.

System Overview

The trifilar pendulum system consists of a suspended circular plate supported by three symmetrically spaced cables attached to a rigid frame. Objects placed on the plate cause a change in the system’s inertial properties, which can be determined by analyzing the plate's free oscillations following a controlled disturbance.

The measurement platform integrates several key subsystems working in coordination:

- **Tension Cell Subsystem:**

  Three tension cells are mounted between the plate and frame to measure the forces in each supporting cable. These readings are used to determine the mass of the plate alone, and the combined mass of the plate and test object.

- **Rotation Initiation Subsystem:**

  A stepper motor connected through a motor driver imparts a small, controlled rotation to the plate. Electromagnets secure the plate before the test begins, and release simultaneously when the motor steps are completed, allowing free oscillation without external interference.

- **Motion Capture Subsystem:**

  An inertial measurement unit (IMU) mounted at the center of the plate continuously records angular velocity data around the vertical axis. High sampling rates ensure that even subtle rotational movements are captured accurately.

- **Central Control Subsystem:**

  A Raspberry Pi 5 runs custom Python scripts that control the system. The Pi activates the motor and electromagnets through a relay module, reads sensor data, records IMU output, and performs data analysis to calculate the system’s moment of inertia.

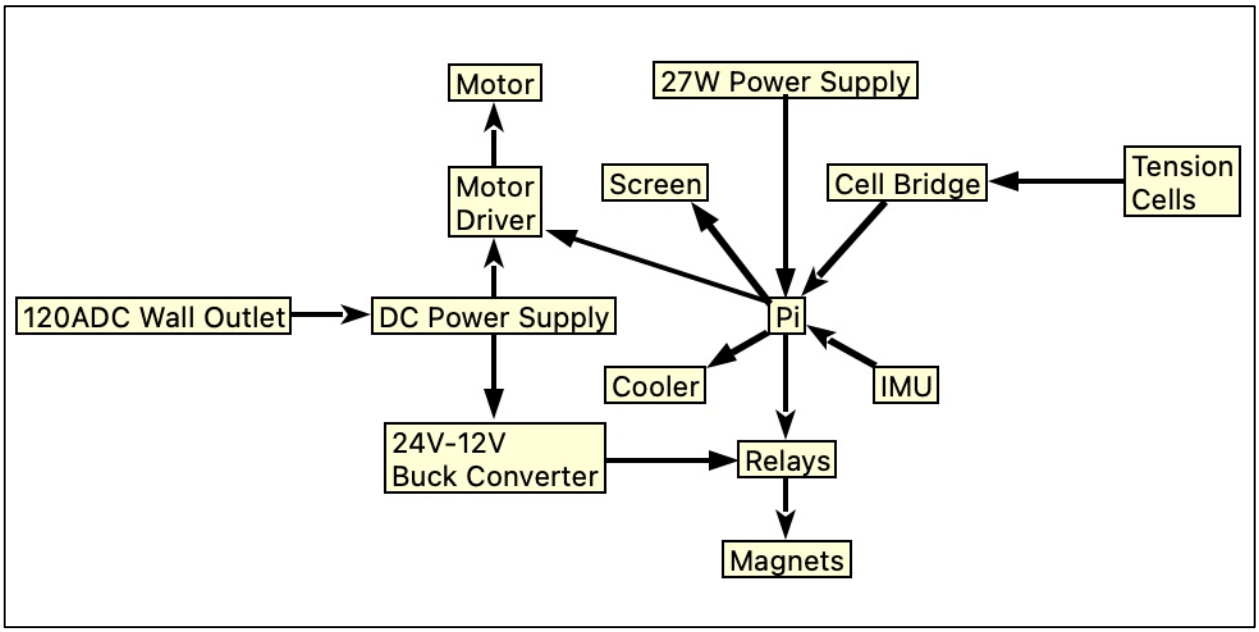

Figure 1. System Diagram

The combination of these subsystems allows for the full automation of the measurement process: from initial mass acquisition to plate perturbation, to data recording and processing. Proper synchronization between mechanical and electronic components ensures that high-quality, repeatable measurements can be achieved with minimal manual intervention. The trifilar pendulum design was selected for its unique advantages in measuring rotational inertia. Compared to other dynamic testing methods, the trifilar configuration isolates rotational motion while minimizing translation and tilting, which improves measurement accuracy. Its relatively simple mechanical structure, combined with the ability to measure small oscillations with high precision, makes it particularly well-suited for capturing the inertial properties of a wide range of objects. Additionally, the design allows for quick resetting between tests, enabling high repeatability and efficient testing cycles.

Code Summary

The control and data acquisition processes are managed through three primary Python scripts: *TensionCells.py*, *IMU.py*, and *M5CodeFinal.py*. Each script is modular, with a clearly defined role in the overall system operation.

- **TensionCells.py:**

  This script initializes and manages communication with the three Phidget load cells. It continuously listens for voltage ratio changes on each channel and maintains a real-time dictionary of force readings. These values are used to calculate the mass of the plate and the plate-plus-object configuration. Automatic error handling detects sensor disconnection or instability during measurement.

- **IMU.py:**

  This script handles the communication with the inertial measurement unit (WTGAHRS2) via a USB serial interface. The IMU transmits angular velocity data packets at a baud rate of 230400 bits per second, enabling fast, high-resolution data transfer. IMU.py extracts the Z-axis angular velocity, applies scaling to convert raw sensor values to degrees per second, and provides a live feed of rotational data for the system. Proper calibration at startup ensures that small offsets or biases are removed before recording begins.

- **M5CodeFinal.py:**

  Serving as the main program, this script integrates mass measurement, motor and magnet control, IMU data recording, and post-processing analysis. It follows a structured sequence:

  1. Measures the baseline and object mass using the tension cells.
  2. Waits for user confirmation to begin recording.

3. Commands the stepper motor to apply a controlled rotation while electromagnets release the plate.
4. Records IMU angular velocity data throughout the oscillation period.
5. Analyzes the IMU data to calculate the average oscillation period and settling time.
6. Computes the object's moment of inertia based on mass, cable length, plate radius, and oscillation characteristics.

The use of modular scripting ensures that individual components can be tested, debugged, and updated independently without disrupting the full system. Moreover, the high sampling rates and fast serial communications allow the system to capture fine dynamic behavior without significant data loss, which is critical for producing accurate and reliable results.

Test Campaign

Prior to full system integration, each major component was individually tested to validate its functionality and ensure reliability. The tension cells, IMU, stepper motor, motor driver, relay module, and electromagnets were all evaluated independently using a combination of Python and C-based testing programs. Spyder was configured on the Raspberry Pi to support live code execution and easier debugging during this phase.

Testing involved verifying basic sensor readings, confirming correct motor stepping behavior, ensuring clean relay switching, and checking stable IMU data acquisition at high sampling rates. Diagnosed issues, such as communication errors or mechanical misalignments, were addressed immediately by making necessary adjustments to the hardware assembly or refining software parameters.

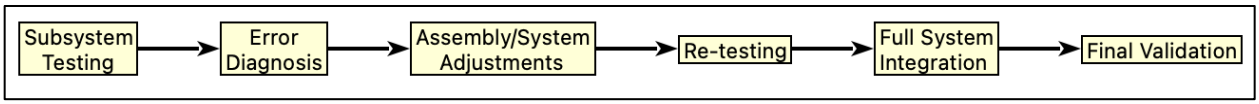

Figure 2. Testing Campaign Block Diagram

Once each subsystem was verified, they were progressively integrated into a unified system. Full-system testing focused on confirming that all hardware and scripts operated cohesively, with proper timing, triggering, and data recording. Adjustments were made as needed to optimize performance, eliminate data loss, and ensure smooth coordination between the control and measurement subsystems. Final validation confirmed that the system could complete full test cycles automatically with high reliability and repeatability.

Results

The automated trifilar-pendulum system was evaluated through a series of validation and range tests. The key findings are summarized below.

1. Reference Validation

- **Test object:** A precision-machined hollow cylinder with a certified MOI of 0.0278 $kg \cdot m^2$.
- **Measured MOI:** 0.0288 $kg \cdot m^2$
- **Error:** +3.6% (overestimation)

This single-object test demonstrates that, under ideal mid-range conditions, the system can measure inertia within a few percent of the true value.

2. Mass-Range Performance

- **Light objects (< 3 lbs.):**
    - Lower angular momentum yields smaller oscillation amplitudes.
    - Signal-to-noise ratio decreases, causing error to rise above the baseline 3.6%.
- **Heavy objects (> 50 lbs.):**
    - Increased structural and cable damping shortens the free-decay response.
    - Non-ideal exponential behavior in the decay curve leads to larger fit errors.

Across both extremes, errors systematically increase, indicating the practical mid-range of reliable operation lies between roughly 3 lbs. and 50 lbs.

3. Effect of Object Placement (Parallel-Axis Theorem)

- By deliberately placing the reference object off-center and applying the parallel-axis theorem in post-processing, measured MOI values remained within the system's baseline uncertainty.
- **Conclusion:** Precise centering of the test object on the plate is not required, simplifying setup without sacrificing accuracy.

4. Overall Accuracy and Usable Range

- **Mid-range masses (3–50 lbs.):** MOI error consistently below 5%, with repeatable measurements on consecutive runs.
- **Extremes (< 3 lbs. or > 50 lbs.):** Errors exceed 5% due to physical limits of angular momentum and damping.

These results confirm that the platform meets its design goal of rapid, hands-off inertia measurements with acceptable accuracy for most small-to-mid-weight test articles.

Discussion

The completed trifilar pendulum system demonstrated strong performance during testing, successfully automating the mass measurement, rotation initiation, and oscillation recording processes. Individual component testing proved essential to identifying and resolving early issues, such as minor wiring inconsistencies, IMU communication stability, and mechanical clearances within the assembly. Integrating Python and C-based diagnostic tools allowed for rapid troubleshooting and system refinement throughout the development process.

While overall system operation was consistent and reliable, certain limitations were observed. Sensitivity to external vibrations and slight cable stretch during setup occasionally introduced variability into oscillation measurements. Additionally, the IMU's performance was highly dependent on careful offset calibration at the start of each run. Despite these challenges, the system produced repeatable and accurate moment of inertia calculations across multiple test cycles, validating the underlying design approach.

The results confirmed that modular scripting, fast communication rates, and careful mechanical setup were critical to achieving high-quality measurements. Areas for further improvement, such as enhanced vibration isolation or automated leveling adjustments, are discussed in more detail in the conclusion and future work sections.

Conclusion

The trifilar pendulum system developed by the M5 group successfully met the project objectives set forth by L3Harris. The combination of modular Python scripts, robust hardware components, and systematic testing allowed the team to build a platform capable of accurately measuring the moment of inertia of various objects. Through individual component validation, iterative troubleshooting, and full-system integration, a reliable and repeatable measurement process was achieved.

Key factors contributing to the system's success included precise motor control, high-speed data acquisition from the IMU, and accurate mass determination using tension cells. The system demonstrated the ability to operate autonomously with minimal user input, consistently producing high-quality measurement data. While minor challenges were encountered during testing, such as sensitivity to external disturbances and setup variations, these issues were effectively mitigated through procedural adjustments.

Overall, the project highlights the importance of thorough system validation, cross-disciplinary integration of mechanical and electronic components, and careful attention to data quality in experimental setups. Future improvements and potential enhancements are discussed in the following sections.

Further Work

While the current system meets the primary objectives, areas for improvement and further development have been identified to enhance performance, reliability, and ease of use:

- **Vibration Isolation:**

  Implementing additional damping or isolation measures could reduce the impact of environmental vibrations during testing, improving measurement consistency.
- **Automated Plate Leveling:**

  Integrating an electronic leveling system or automatic turnbuckle adjustment would ensure the plate remains perfectly level without manual intervention, reducing setup variability.
- **Wireless Data Acquisition:**

  Future iterations could explore wireless IMU systems or remote tension cell interfaces to eliminate potential cable drag and improve system cleanliness.
- **Expanded Mass Range Testing:**

  Additional testing with heavier and lighter objects would help characterize the system's dynamic range and sensitivity limits.
- **Graphical User Interface (GUI) Development:**

  For this project, we did construct a GUI, however it is rather simple, and does not produce a plot. Further work could include a more fleshed out GUI.

Pursuing these enhancements would further increase the system's robustness, accuracy, and versatility for broader future applications.

Appendix A – Python Scripts

Appendix A.1 – TensionCells2.py

```python
from Phidget22.PhidgetException import *
from Phidget22.Phidget import *
from Phidget22.Devices.Log import *
from Phidget22.LogLevel import *
from Phidget22.Devices.VoltageRatioInput import *
import time

# Enable logging once at import time
Log.enable(LogLevel.PHIDGET_LOG_INFO, "phidgetlog.log")

# Global dictionary to store latest readings
_latest_readings = {}

#-----------------------------------------------------------
# Event Handlers
#-----------------------------------------------------------
def onVoltageRatioChange(self, voltageRatio):
    channel = self.getChannel()
    _latest_readings[channel] = voltageRatio

def onAttach(self):
    print(f"Attach [Channel {self.getChannel()}]!")

def onDetach(self):
    print(f"Detach [Channel {self.getChannel()}]!")

def onError(self, code, description):
    print(f"Error on Channel {self.getChannel()}: Code {code} - {description}")

#-----------------------------------------------------------
# Module Functions
#-----------------------------------------------------------
def setup_tension_cells(serial_number, channels=1, timeout=5000):
    """
    Initializes and opens the specified number of VoltageRatioInput channels.
    """
    cells = []
    for ch in range(channels):
        try:
            cell = VoltageRatioInput()
            cell.setDeviceSerialNumber(serial_number)
            cell.setChannel(ch)
```

```
            cell.setOnVoltageRatioChangeHandler(onVoltageRatioChange)
            cell.setOnAttachHandler(onAttach)
            cell.setOnDetachHandler(onDetach)
            cell.setOnErrorHandler(onError)
            cell.openWaitAttachment(timeout)
            cells.append(cell)
        except PhidgetException as ex:
            print(f"Error initializing tension cell on channel {ch} (SN:
{serial_number}): {ex}")
    return cells

def get_latest_forces():
    """
    Returns a copy of the latest raw voltage ratio readings.
    """
    return dict(_latest_readings)


def close_tension_cells(cells):
    """
    Closes all provided VoltageRatioInput channels.
    """
    for cell in cells:
        try:
            cell.close()
        except PhidgetException as ex:
            print(f"Error closing channel {cell.getChannel()}: {ex}")
```

Appendix A.2 – IMU.py

```python
import time
import serial
import struct

# Open serial port to the IMU
ser = serial.Serial("/dev/ttyUSB0", 230400, timeout=0.5)

def read_gyro_z():
    """
    Reads a single gyro Z-axis measurement from the IMU.
    This function waits until a valid packet is received,
    then unpacks and returns the scaled gyro Z angle in degrees.
    """
    while True:
        # Sync to header byte
        if ser.read(1) == b'\x55':
            data_type = ser.read(1)
            if data_type == b'\x53':  # Angle data packet
                data = ser.read(8)
                if len(data) == 8:
                    # Unpack as four signed 16-bit integers (pitch, roll, yaw, temp)
                    ax, ay, az, temp = struct.unpack('<hhhh', data)
                    return az / 32768.0 * 180  # Scale to degrees

def get_initial_offset():
    """
    Returns an initial gyro Z reading as the offset.
    """
    return read_gyro_z()

if __name__ == "__main__":
    # Only run the following code if this module is executed directly.
    # When imported, this block will not be executed.
    initial_offset = get_initial_offset()
    print(f"Initial Gyro Z Offset: {initial_offset:.2f}")
    # Continuously read gyro Z and print at a controlled rate.
    gyro_z_values = []  # Store gyro values for reference
    sampling_rate = 100  # Hz
    print_rate = 0.1     # Print rate (in seconds)
    try:
        start_time = time.time()
        while True:
            gyro_z = read_gyro_z() - initial_offset
            gyro_z_values.append(gyro_z)
            # Print every print_rate seconds
            if time.time() - start_time >= print_rate:
                print(f"Calibrated Z Angle: {gyro_z_values[-1]:.2f}")
                start_time = time.time()
            time.sleep(1 / sampling_rate)
    except KeyboardInterrupt:
        print("\nStopped by user.")
        ser.close()
```

Appendix A.3 – M5Code.py

```
import time
import threading
import numpy as np
import matplotlib.pyplot as plt
import PySimpleGUI as sg
from scipy.signal import find_peaks
from scipy.optimize import curve_fit
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from periphery import GPIO
# --- Custom Modules ---
from TensionCells2 import setup_tension_cells, get_latest_forces, close_tension_cells
from IMU import read_gyro_z, ser

# ---------------------------- GPIO SETUP -----------------------------
GPIO_CHIP = "/dev/gpiochip0"
STEP_PIN, DIR_PIN = 18, 23
EM1_PIN, EM2_PIN, EM3_PIN = 5, 6, 13
STEP = GPIO(GPIO_CHIP, STEP_PIN, "out")
DIR  = GPIO(GPIO_CHIP, DIR_PIN,  "out")
EM1 = GPIO(GPIO_CHIP, EM1_PIN, "out")
EM2 = GPIO(GPIO_CHIP, EM2_PIN, "out")
EM3 = GPIO(GPIO_CHIP, EM3_PIN, "out")
# Safe defaults
STEP.write(False); DIR.write(False)
EM1.write(False); EM2.write(False); EM3.write(False)

# ---------------------------- CONSTANTS -----------------------------
sampling_rate      = 100
IMU_RECORD_TIME    = 30
TENSION_GAIN       = 3.3508e4
TENSION_OFFSET     = -1.5682e-5
MOTOR_TRIGGER_TIME = 1.0
STEADY_DELAY       = 2.0
R                  = 0.4572
L                  = 1.5

# ---------------------- Motor/EM Function ---------------------------
def trigger_motor():
    EM1.write(True); EM2.write(True); EM3.write(True)
    time.sleep(1)
    DIR.write(True)
    for _ in range(120): STEP.write(False); time.sleep(0.0015); STEP.write(True);
time.sleep(0.0015)
    DIR.write(False)
```

```
    for _ in range(300): STEP.write(False); time.sleep(0.0015); STEP.write(True);
time.sleep(0.0015)
    EM1.write(False); EM2.write(False); EM3.write(False)
    time.sleep(1)
    DIR.write(True)
    for _ in range(180): STEP.write(False); time.sleep(0.0015); STEP.write(True);
time.sleep(0.0015)

# ------------------- Tension Cell Measurements ----------------------
def measure_plate_mass(serial_number=716326, channels=3):
    cells = setup_tension_cells(serial_number=serial_number, channels=channels)
    time.sleep(2)
    samples = []
    for _ in range(20):
        forces = get_latest_forces()
        if forces:
            vals = [(raw - TENSION_OFFSET)*TENSION_GAIN for raw in forces.values()]
            samples.append(np.mean(vals))
        time.sleep(0.1)
    close_tension_cells(cells)
    return 3 * np.mean(samples)

def measure_object_mass(baseline, serial_number=716326, channels=3):
    cells = setup_tension_cells(serial_number=serial_number, channels=channels)
    time.sleep(2)
    samples = []
    for _ in range(20):
        forces = get_latest_forces()
        if forces:
            vals = [(raw - TENSION_OFFSET)*TENSION_GAIN for raw in forces.values()]
            samples.append(np.mean(vals))
        time.sleep(0.1)
    close_tension_cells(cells)
    return 3 * np.mean(samples) - baseline

# ----------------------- IMU Data & Analysis ----------------------
def record_imu_data():
    offset = read_gyro_z()
    ser.reset_input_buffer()
    times, gyro_z = [], []
    trig = False
    start = time.time()
    while time.time() - start < IMU_RECORD_TIME:
        t = time.time() - start
        if t >= MOTOR_TRIGGER_TIME and not trig:
            trigger_motor(); trig=True
        val = read_gyro_z() - offset
        times.append(t); gyro_z.append(val)
```

```
        time.sleep(1/sampling_rate)
    return np.array(times), np.array(gyro_z)

def analyze_imu_data(times, gyro_z, object_mass):
    steady = MOTOR_TRIGGER_TIME + STEADY_DELAY
    mask = times >= steady
    st, gz = times[mask], gyro_z[mask]
    # settling
    def exp_decay(t, A, tau, C): return A*np.exp(-t/tau)+C
    try:
        popt, _ = curve_fit(exp_decay, st[:len(gz)], np.abs(np.convolve(gz,
np.ones(10)/10, mode='valid')), maxfev=10000)
        settling = 4 * popt[1]
    except:
        settling = None
    # period & MOI
    peaks, _ = find_peaks(gz, height=0.1)
    if len(peaks)>1:
        period = np.mean(np.diff(st[peaks]))
        moi = object_mass * R**2 * period**2 / (4*np.pi**2*L)
    else:
        period = moi = None
    # figure
    fig, ax = plt.subplots(figsize=(4,3))
    ax.plot(st, gz, label='ω (deg/s)')
    ax.axvline(steady, color='r', linestyle='--', label='Steady Start')
    ax.set_xlabel('Time (s)'); ax.set_ylabel('ω (deg/s)')
    ax.legend(); ax.grid()
    return settling, period, moi, fig

# ---------------------------- GUI SETUP ------------------------------
sg.set_options(font=('Helvetica',14))
layout = [
    [sg.Text('🔧 Pendulum Control', expand_x=True, justification='center')],
    [sg.Button('1.Measure Plate', key='PLATE'), sg.Text('', key='PSTAT')],
    [sg.Button('2.Measure Object', key='OBJ', disabled=True), sg.Text('',
key='OSTAT')],
    [sg.Button('3.Perform MOI', key='MOI', disabled=True), sg.Text('', key='MSTAT')],
    [sg.Button('4.Show Results', key='RES', disabled=True), sg.Button('Run Again',
key='RUN', disabled=True)],
    [sg.Canvas(key='CANVAS')],
    [sg.Multiline('', key='RESULTS', size=(50,5), disabled=True)]
]
window = sg.Window('Pendulum Kiosk', layout, finalize=True)

# helpers to draw figure
canvas_elem = window['CANVAS']
canvas = canvas_elem.TKCanvas
```

```
def draw_figure(canvas, figure):
    for child in canvas.winfo_children(): child.destroy()
    agg = FigureCanvasTkAgg(figure, canvas)
    agg.draw(); agg.get_tk_widget().pack()

# state
baseline = object_mass = None
imu_data = None

# reset function
def reset():
    global baseline, object_mass, imu_data
    baseline = object_mass = imu_data = None
    for k in ('PSTAT','OSTAT','MSTAT','RESULTS'): window[k].update('')
    window['PLATE'].update(disabled=False)
    window['OBJ'].update(disabled=True)
    window['MOI'].update(disabled=True)
    window['RES'].update(disabled=True)
    window['RUN'].update(disabled=True)
    draw_figure(canvas, plt.figure())

reset()

# event loop
while True:
    event, vals = window.read()
    if event in (sg.WIN_CLOSED, 'Exit'): break
    if event=='RUN': reset()
    if event=='PLATE':
        window['PSTAT'].update('Measuring...')
        threading.Thread(target=lambda: window.write_event_value('PDONE',
measure_plate_mass()), daemon=True).start()
    elif event=='PDONE':
        baseline = vals[event]
        lb = baseline*2.205
        window['PSTAT'].update(f'{baseline:.3f} kg / {lb:.2f} lb')
        window['PLATE'].update(disabled=True); window['OBJ'].update(disabled=False)
    elif event=='OBJ':
        window['OSTAT'].update('Measuring...')
        threading.Thread(target=lambda: window.write_event_value('ODONE',
measure_object_mass(baseline)), daemon=True).start()
    elif event=='ODONE':
        object_mass = vals[event]
        lb = object_mass*2.205
        window['OSTAT'].update(f'{object_mass:.3f} kg / {lb:.2f} lb')
        window['OBJ'].update(disabled=True); window['MOI'].update(disabled=False)
    elif event=='MOI':
```

```
        window['MSTAT'].update('Recording...')
        threading.Thread(target=lambda: window.write_event_value('MDONE',
record_imu_data()), daemon=True).start()
    elif event=='MDONE':
        imu_data = vals[event]
        window['MSTAT'].update('Done')
        window['MOI'].update(disabled=True); window['RES'].update(disabled=False)
    elif event=='RES':
        times, gyro = imu_data
        settling, period, moi, fig = analyze_imu_data(times, gyro, object_mass)
        # draw inside GUI
        draw_figure(canvas, fig)
        # show text
        lines = [f'Object Mass: {object_mass:.3f} kg',
                 #f'Settling Time: {settling:.2f} s' if settling else 'Settling: N/A',
                 f'Period: {period:.3f} s' if period else 'Period: N/A',
                 f'MOI: {moi:.4f} kg·m²' if moi else 'MOI: N/A']
        window['RESULTS'].update('\n'.join(lines))
        window['RES'].update(disabled=True); window['RUN'].update(disabled=False)

window.close()
```

Appendix B – Additional Figures

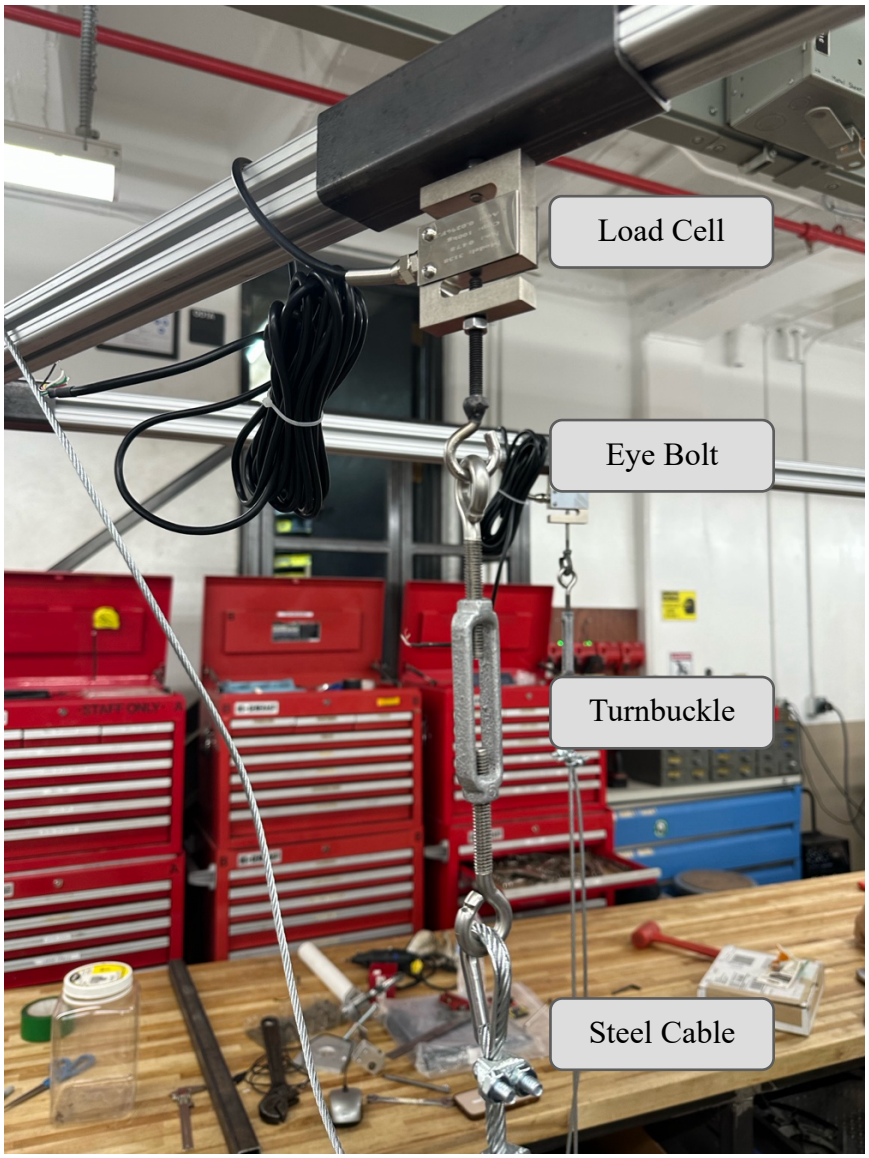

Figure 3. Cable System

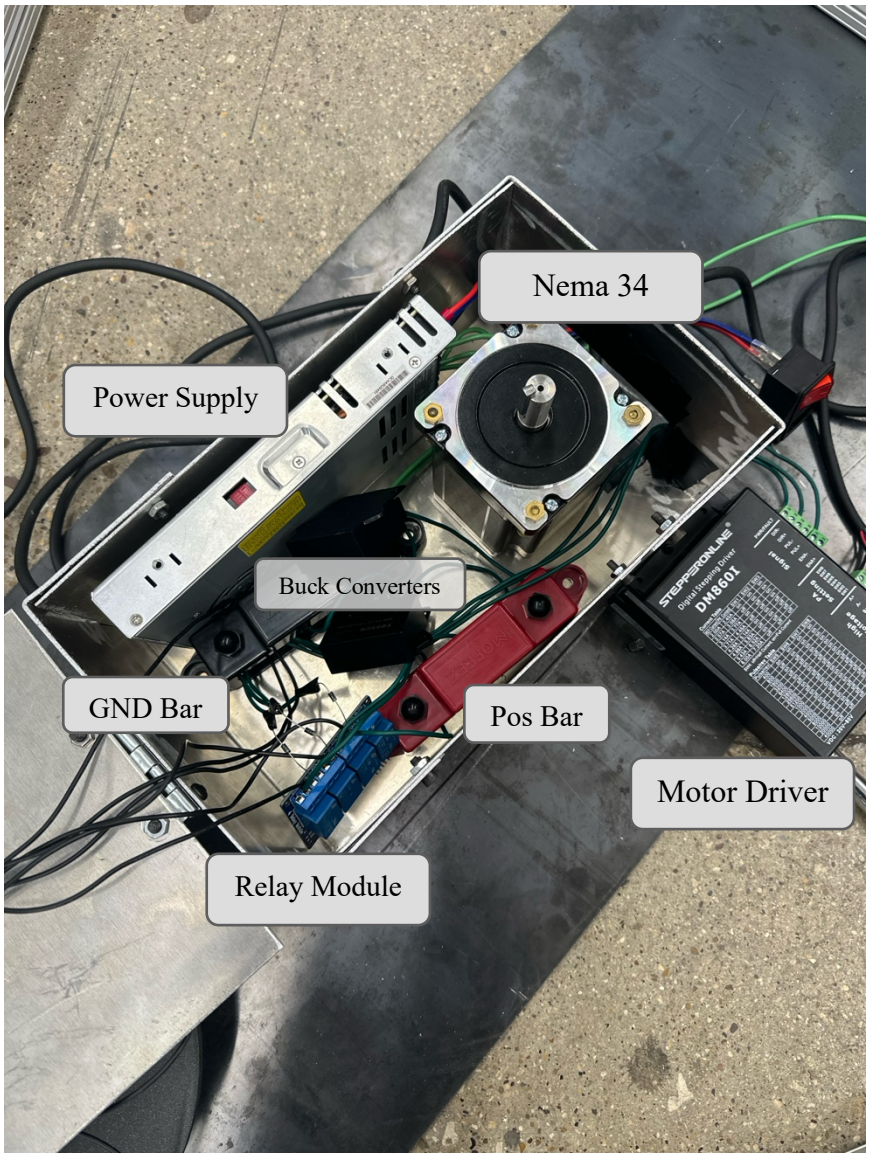

Figure 4. System Electrical Enclosure

Appendix C – System Setup and Operation Guide

Assembly Pre-Setup

- Ensure the frame and plate are level.
- If the plate is not level, adjust the cable lengths using the turnbuckles.
- Verify that all aluminum beams are properly seated into their corner joints.
- Make sure wiring is clear of all moving components within the assembly.
- When placing an object for measurement, place it precisely at the center of the plate.

Before Starting

- Confirm all hardware (tension cells, IMU, stepper motor, electromagnets) are securely connected.
- Power on the Raspberry Pi and wait until the touchscreen interface is fully loaded.

Performing The Measurement

1. Open Terminal

Click the Terminal icon to open a new Terminal window.

Alternatively, use the keyboard shortcut: *Ctrl + Alt + T.*

2. Activate the Python Environment (M5env)

In Terminal, type:

```
conda activate M5env
```

You should see (*M5env*) appear at the beginning of the terminal prompt.

3. Navigate to the Script Directory

Type the following to move to the correct directory:

`cd ~/Code`

4. Pre-Run Checklist

Before running the code:

- Ensure the plate is completely still.

- Ensure the magnets are in line with the steel tabs beneath the plate.

- Ensure the surrounding environment is clear of disturbances.

- Ensure the system is safe for motor and magnet activation.

This is critical to obtain accurate and reliable measurements.

5. Run the Measurement Script

- In terminal, type:

    - Non-GUI with graph: `python3 M5Code.py`
    - GUI: `python3 M5GUI.py`

7. Follow the On-Screen Instructions

Tension Measurement:

- The script will first measure the plate weight.

- You will be prompted to press *Enter* to measure the object weight.*

IMU Recording:

- After object measurement, press *Enter* again to start recording.*

- Stand clear as the motor and magnets will activate automatically.

Data Analysis:

- Results and plots* will be automatically generated.

** - GUI will not generate plot and will not prompt user for input. If using GUI, simply push the respective buttons.*

Preparing for a Second Reading

- Allow the script to fully finish.

- Close any open plots.

- Reset the plate and position any new object.

- Press the up-arrow key in Terminal to recall the last command.

- Press Enter to rerun the script.

Safely Shutting Down the Raspberry Pi

Once all tests are complete:

- Type the following command:

```
sudo shutdown -h now
```

- Wait until the screen goes black before unplugging the Pi.

Common Errors and Fixes

Terminal:

- "*Tension cells not found*": Tension cells could not be detected. Check wiring and USB connections. Reboot if necessary.
- "*No steady-state IMU data recorded*": No useful data after motor activation. Ensure IMU is working, and the system remains completely still.
- "*Settling time estimation failed*": Curve fitting was unsuccessful. Reduce noise and verify that the system is undisturbed.
- "*Period detection failed*": Oscillations too small to detect. Verify motor rotation and magnet release.
- "*MOI calculation failed*": Insufficient data for moment of inertia calculation. Verify proper plate rotation
- "*Serial connection error with IMU*": IMU serial connection lost. Check cable connection and restart the script.

GUI:

- *N/A value for period/MOI*: Insufficient data, likely not enough oscillations. Verify proper plate rotation.
- *Abnormally low MOI*: Due to unknown factors, during some runs the IMU records erroneous data values, like 350º. Restart Test.

System:

- *Low plate rotation*: Verify that electromagnets are in line with the steel tabs beneath the plate.

Quick Start Summary

- Open Terminal: Click Terminal Icon or use *Ctrl + Alt + T*

- Activate Environment: `conda activate M5env`

- Navigate to Script Directory: `cd ~/Code`

- Run Script:

    - Non-GUI with graph: `python3 M5Code.py`

    - GUI: `python3 M5GUI.py`

- Shutdown Raspberry Pi: `sudo shutdown -h now`