Missile Guidance System

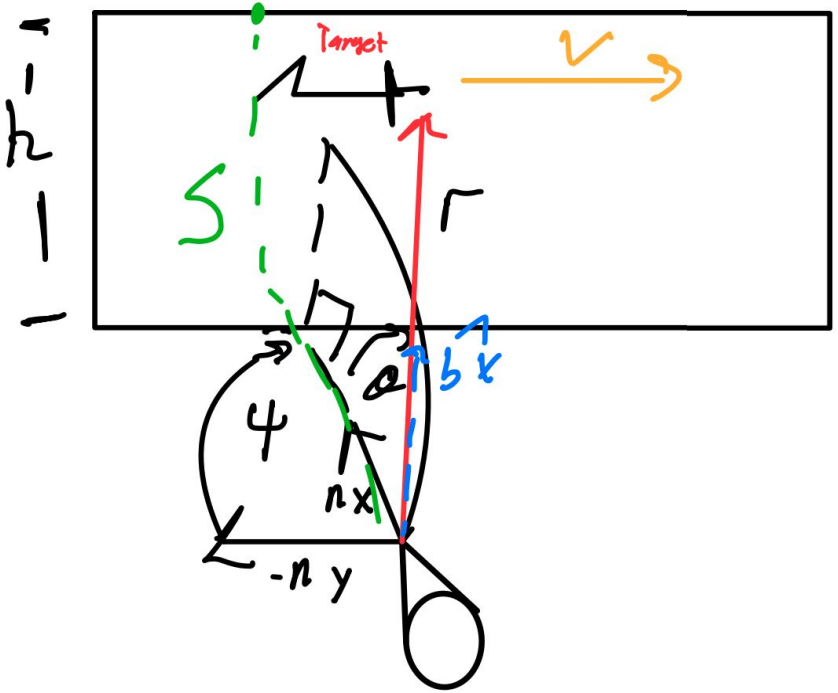
Dev Dhruv, William Cai, Hayden Schaufel, and Alex Wille

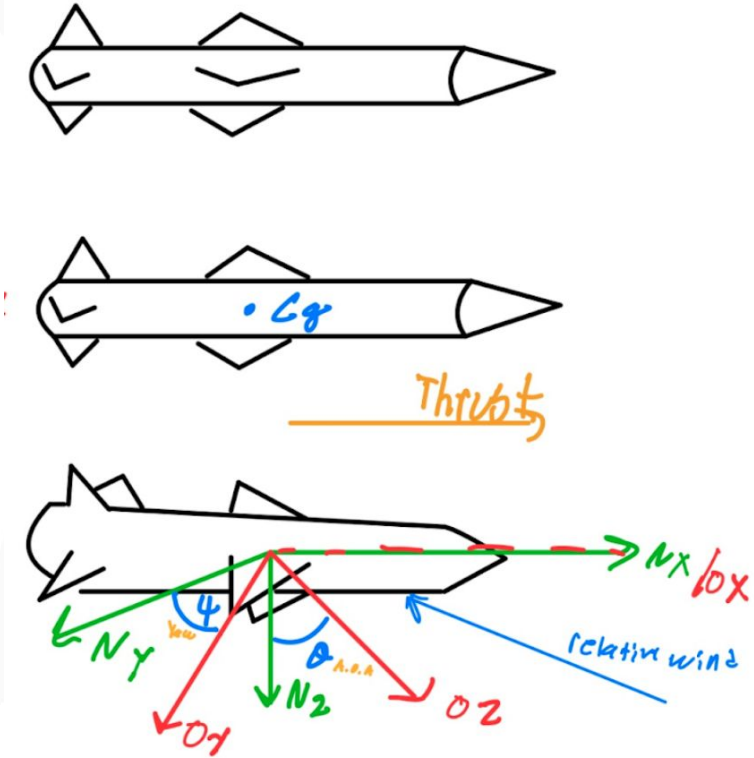
Project Introduction and Intent

Primary Purpose: Design, build, and simulate a missile guidance system to track a non-maneuvering target in 3D space and orient the missile to track the moving target asymptotically.

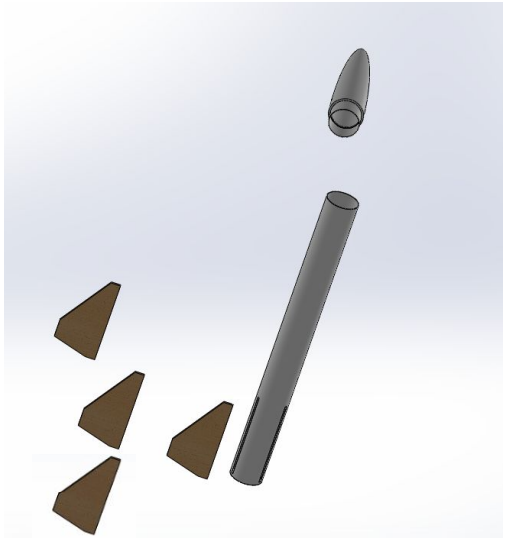
Project Objective: Research and develop a missile guidance system using rigid body dynamics concepts such as basic tracking of a moving target using intuition from flight dynamics (i.e. pitch, roll, and yaw), and tracking dynamics (two degree of freedom radar problem).

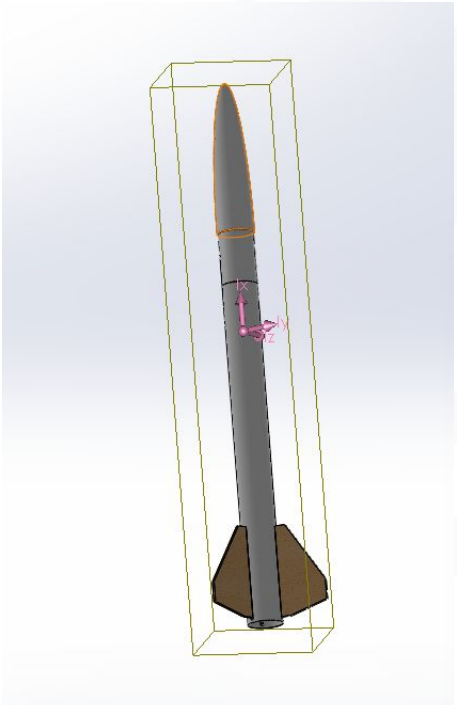
Conceptual Design





Assembly Design



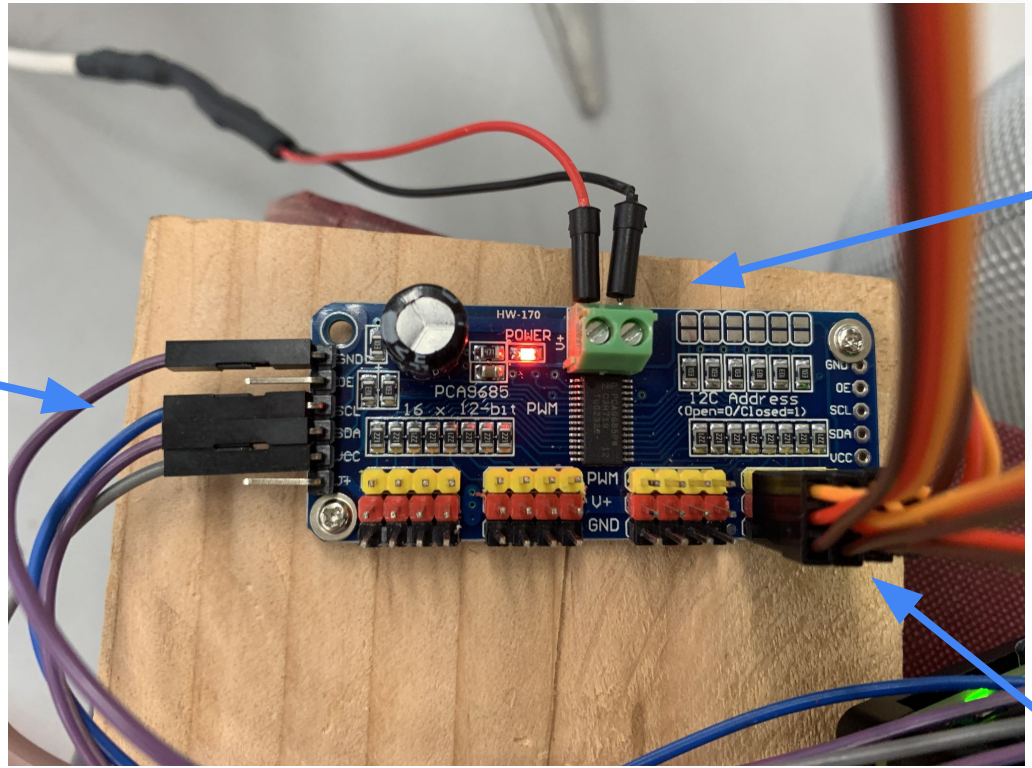


Rocket Assembly

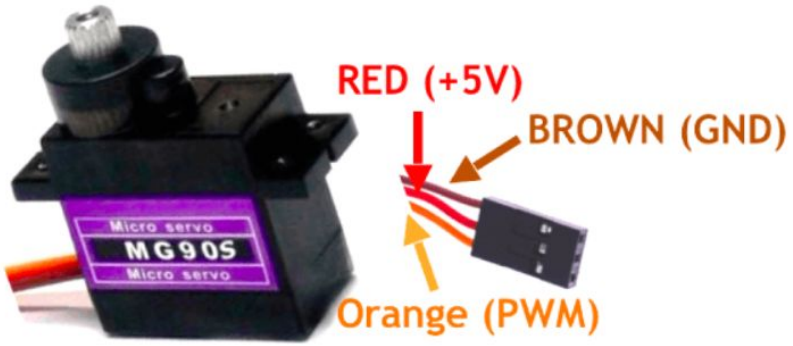
Servo-motor Mounting Assembly

Primary Controller Board (Raspberry Pi 5):

Servo control board (Adafruit PCA9685):

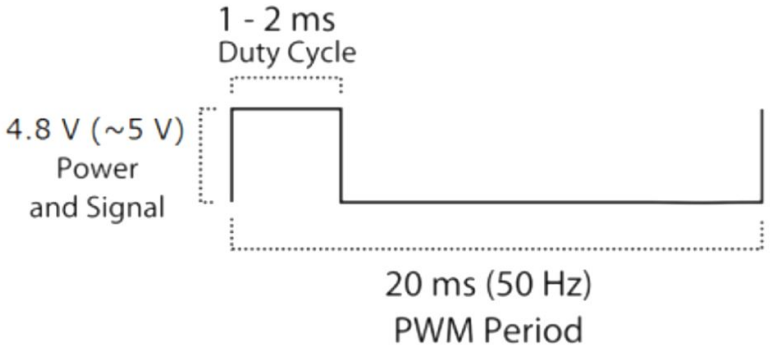


MG90S Servo

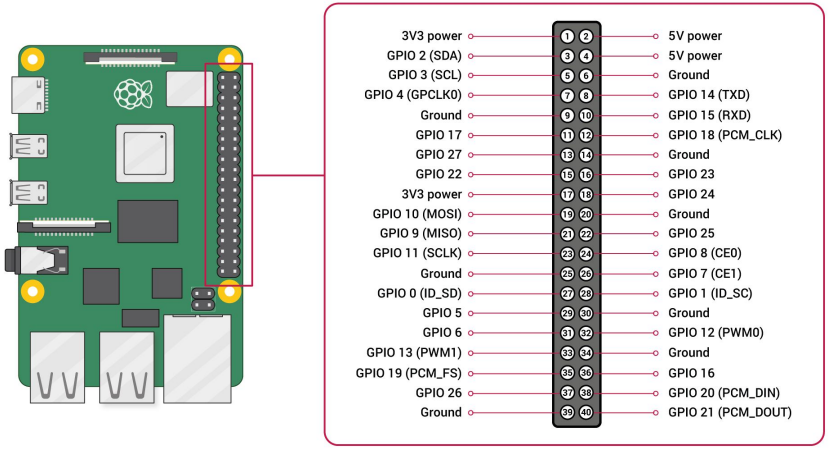


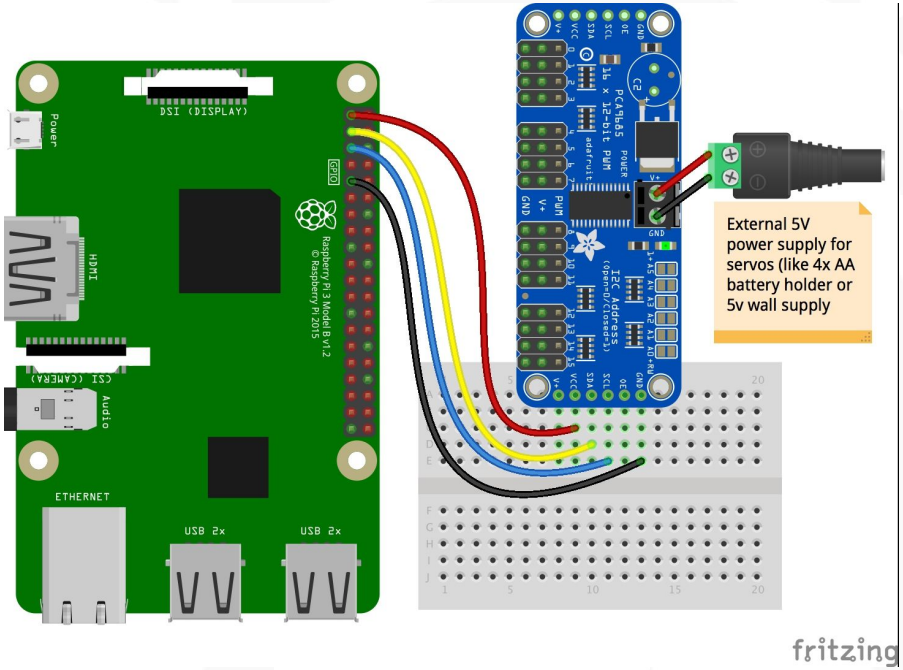
Specifications

- Operating Voltage: 4.8 V to 6V (ideally 5V)
- Rotation: 0 to 180 degrees
- Operating Speed: 0.1s/60°
- Pulse width modulation (PWM) signal produced is 50 Hz
    - By varying the duty cycle (on time), we can control the servo +/- 15° to actuate the missile fins



Rpi5 to PCA9685 Servo Driver Board Wiring Schematic





SJSU SAN JOSÉ STATE UNIVERSITY

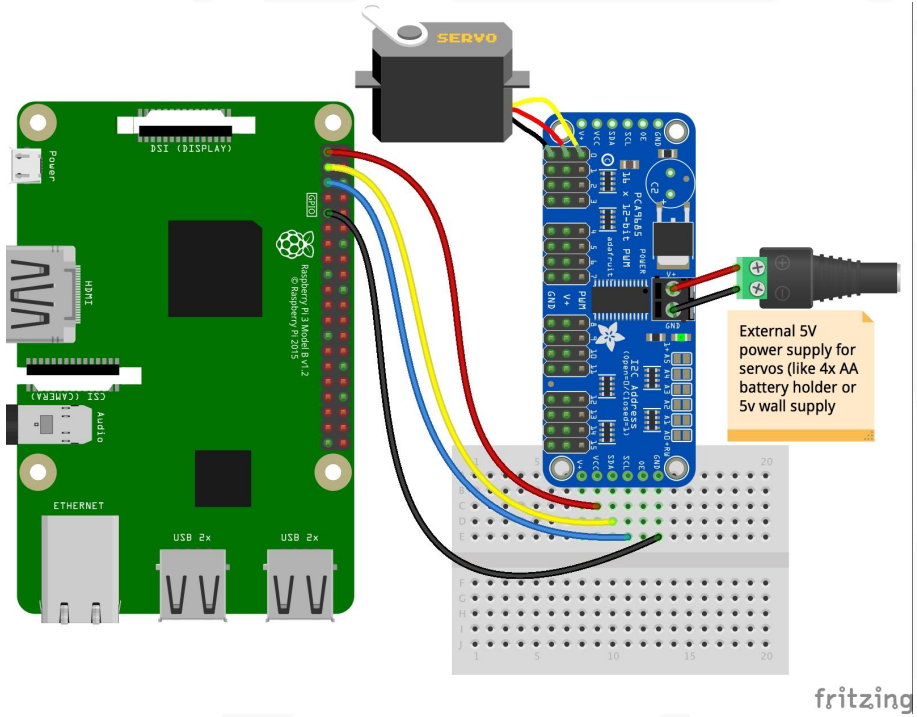
MG90S Servo Connections to PCA9685 Board

Hooking up Rpi 5 to PCA9685 (Servo-Driver Board) Using i2C protocol

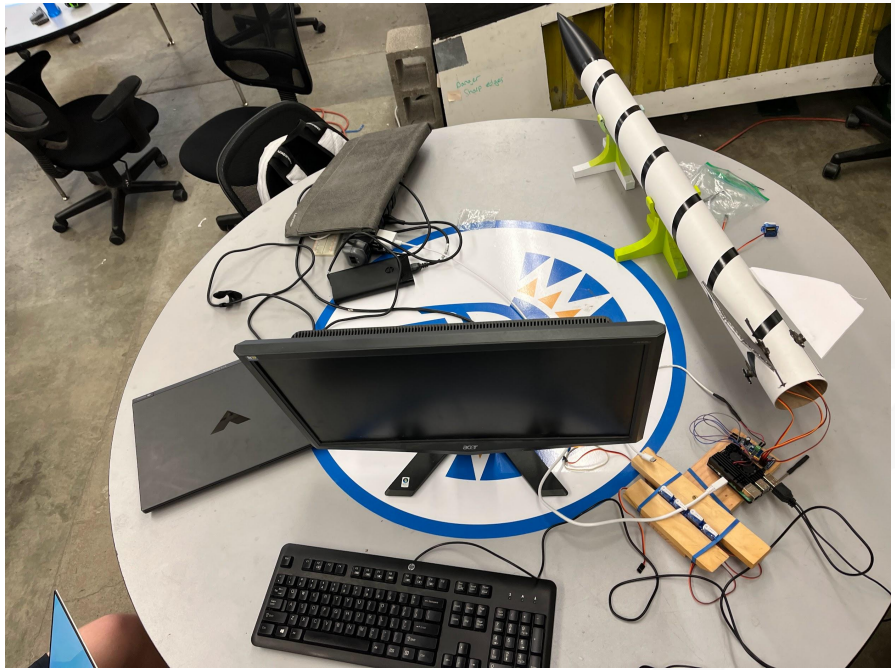
- RPi 5 3v3 pin to PCA9685 VCC pin
- RPi 5 SCL pin to breakout SCL pin
- RPi 5 SDA pin to breakout SDA pin
- RPi 5 GND pin to breakout GND pin

Checking Servo-Motor Connections

- Servo orange wire to PWM pin on channel 0
- Servo red wire to V+ pin on channel 0
- Servo brown wire to ground on channel 0



Hardware Connections and Wiring Setup Result



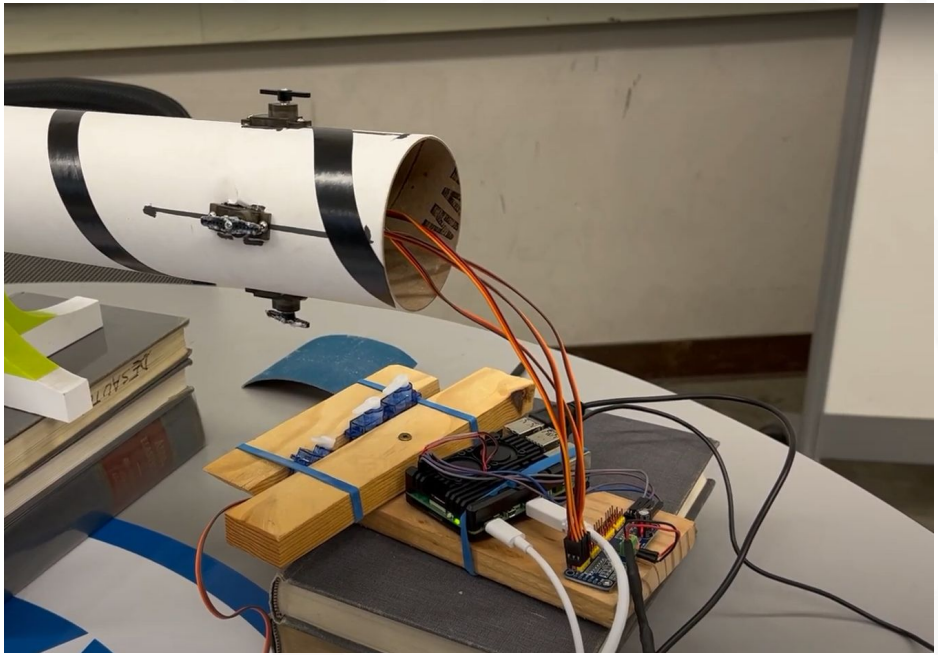
Project Development Process

William Cai
- MATLAB to Python software and hardware interfacing on Raspberry Pi 5 board
- Embedded systems design

Dev Dhruv
- Missile flight dynamics
- Troubleshooted servo control movements
- Assisted with software and hardware interfacing

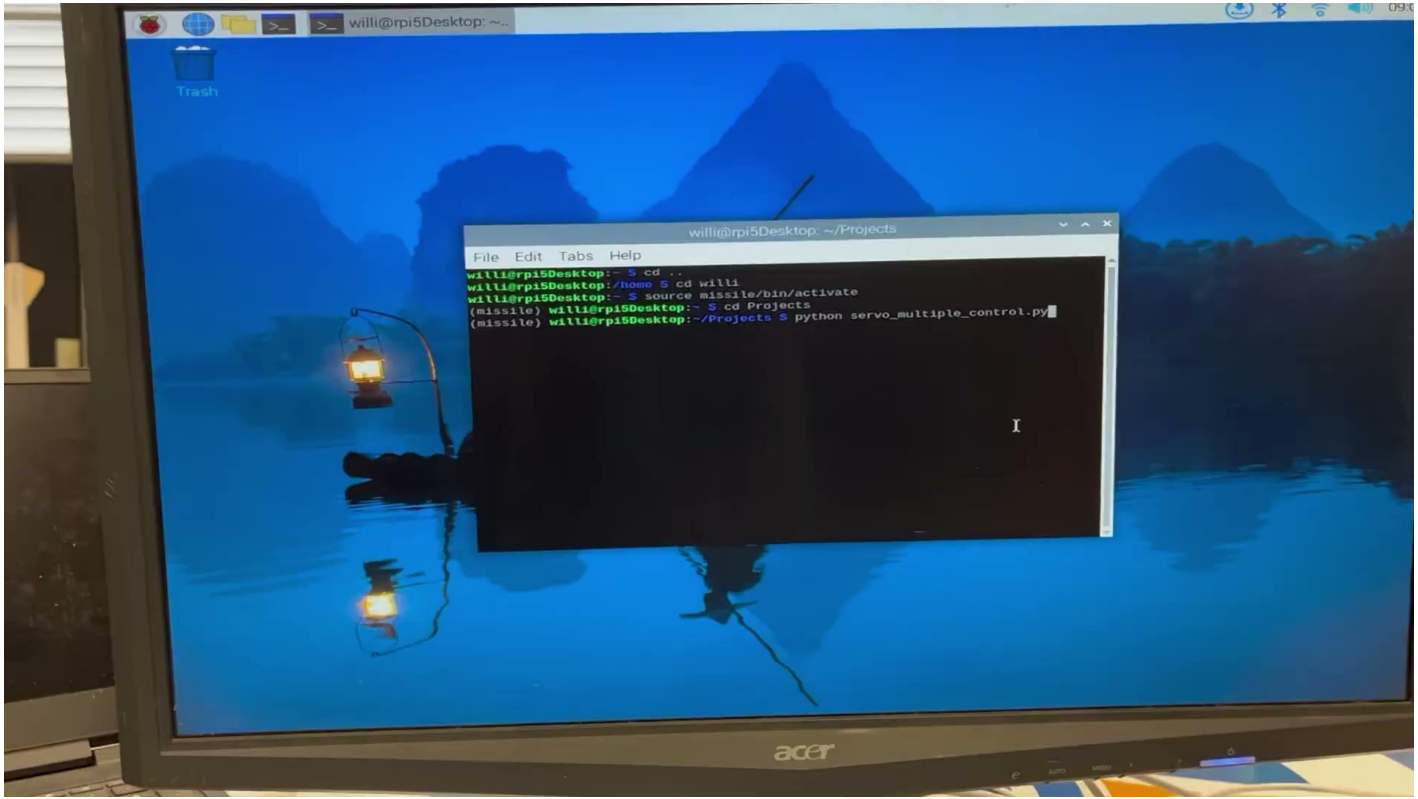
Hayden Schaufel
- Missile flight dynamics in MATLAB
- Servo Motor mount assembly
- Fin attachments

Alex Wille
- Missile target tracking dynamics in MATLAB
- MATLAB simulation and liveplots



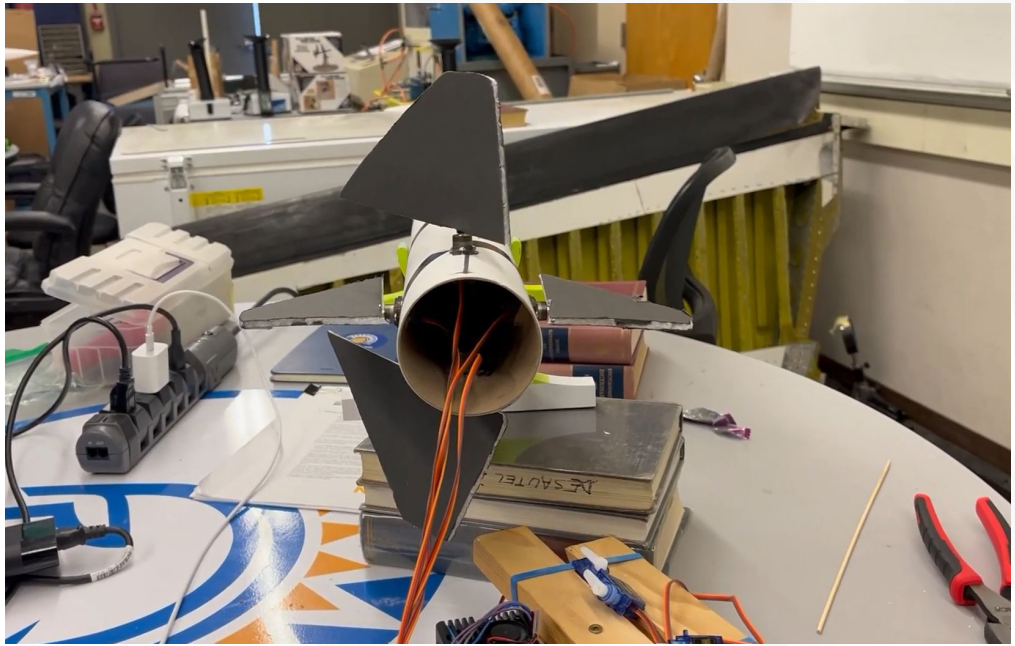
Final Product

Software to Hardware Interfacing in RPiOS on Test Servos



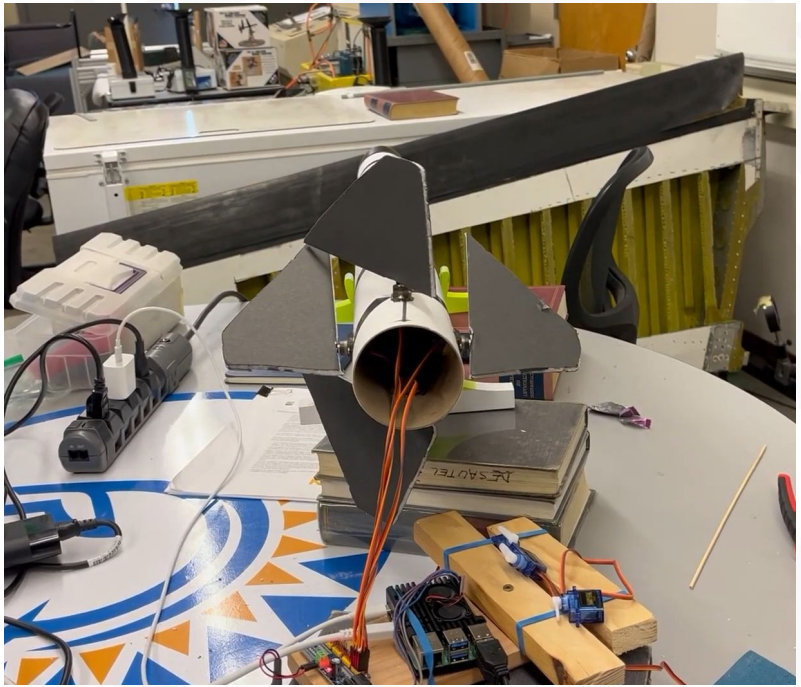
Missile Fin Actuation Movements

- Top and bottom fins on the missile control yaw motions (i.e. left and right banking for the missile)

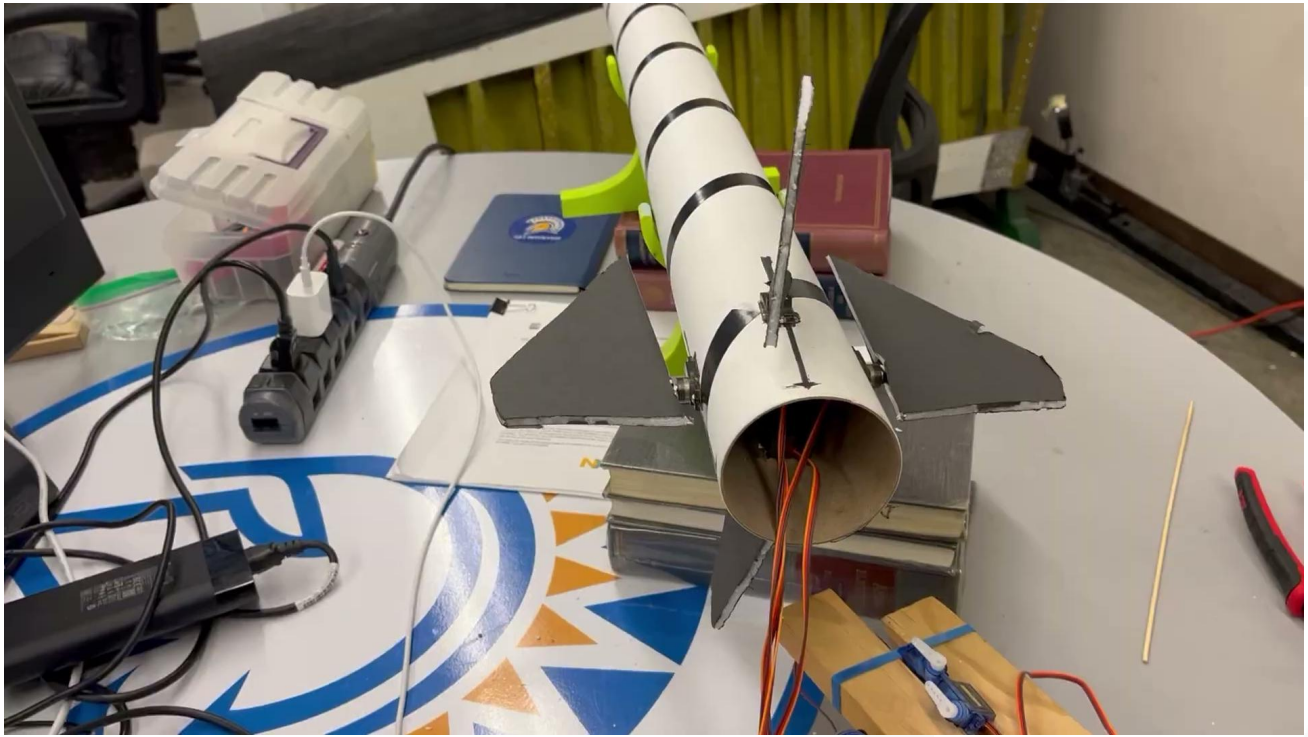


Missile Fin Actuation Movements

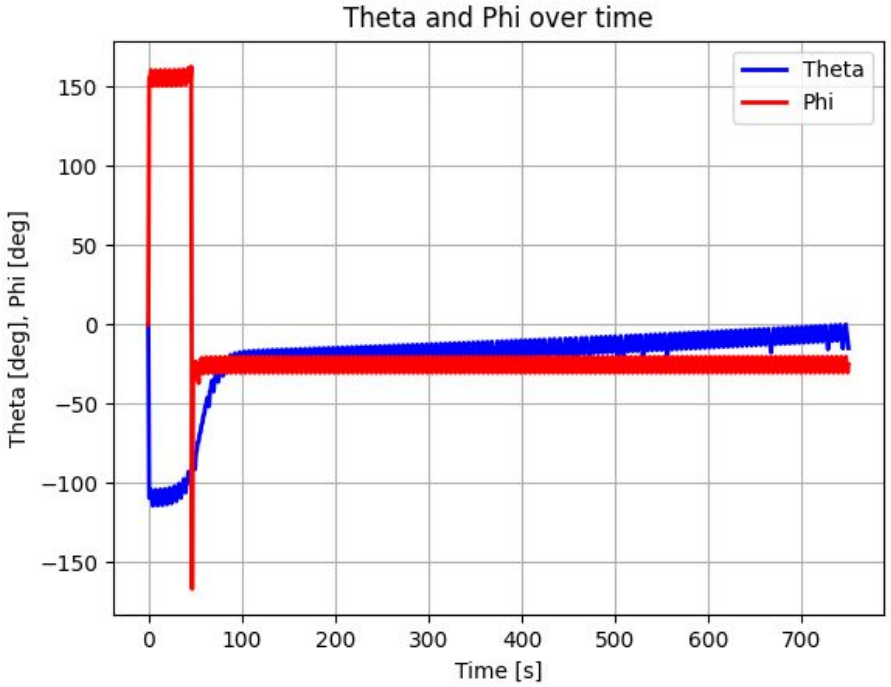
- Left and right fins on the missile control elevation (i.e. pitch up and pitch down movements)



Hardware Demonstration



Angular Displacement Plots



Python Tracking Code for Servo Calibration

Import Python Libraries

```
import RPi.GPIO as GPIO # install
import time
from time import sleep

import board
from board import SCL, SDA # install
import busio # install

import adafruit_pca9685
from adafruit_pca9685 import PCA9685 # install

from adafruit_servokit import ServoKit
kit = ServoKit(channels = 16)

import numpy as np
from numpy.linalg import norm
from numpy.random import randint
import random

import matplotlib.pyplot as plt
```

Python Tracking Code for Servo Calibration

```
# Initialize I2C bus and PCA9685 instance
i2c_bus = busio.I2C(SCL, SDA)
pca = PCA9685(i2c_bus)

# Set the PWM frequency (Hz)
pca.frequency = 50

# Initialize I2C bus and PCA9685 instance
i2c_bus = busio.I2C(SCL, SDA)
pca = PCA9685(i2c_bus)

# Set the PWM frequency (Hz)
pca.frequency = 50

# White Fin on Bottom
SERVO_1_CHANNEL = 0
kit.servo[SERVO_1_CHANNEL].actuation_range = 180
kit.servo[SERVO_1_CHANNEL].set_pulse_width_range(500, 2500)

#White Fin on Top
SERVO_2_CHANNEL = 1
kit.servo[SERVO_2_CHANNEL].actuation_range = 180
kit.servo[SERVO_2_CHANNEL].set_pulse_width_range(500, 2500)

# Black Fin on Left
SERVO_3_CHANNEL = 2
kit.servo[SERVO_3_CHANNEL].actuation_range = 180
kit.servo[SERVO_3_CHANNEL].set_pulse_width_range(500, 2500)

# Black Fin on Right
SERVO_4_CHANNEL = 3
kit.servo[SERVO_4_CHANNEL].actuation_range = 180
kit.servo[SERVO_4_CHANNEL].set_pulse_width_range(500, 2500)
```

Python Servo Calibration

```
# Clearing all variables and closing all figures
plt.close('all')

try:
        while True:
                kit.servo[SERVO_1_CHANNEL].angle = 90
                time.sleep(1)
                kit.servo[SERVO_1_CHANNEL].angle = 60
                time.sleep(1)
                kit.servo[SERVO_1_CHANNEL].angle = 120
                time.sleep(1)

                kit.servo[SERVO_2_CHANNEL].angle = 90
                time.sleep(1)
                kit.servo[SERVO_2_CHANNEL].angle = 60
                time.sleep(1)
                kit.servo[SERVO_2_CHANNEL].angle = 120
                time.sleep(1)

                kit.servo[SERVO_3_CHANNEL].angle = 90
                time.sleep(1)
                kit.servo[SERVO_3_CHANNEL].angle = 60
                time.sleep(1)
                kit.servo[SERVO_3_CHANNEL].angle = 120
                time.sleep(1)

                kit.servo[SERVO_4_CHANNEL].angle = 90
                time.sleep(1)
                kit.servo[SERVO_4_CHANNEL].angle = 60
                time.sleep(1)
                kit.servo[SERVO_4_CHANNEL].angle = 120
                time.sleep(1)
except KeyboardInterrupt:
        exit()
```

Python Tracking Code

Rocket and Target Initial Conditions

```
## rocket conditions

rocket_speed = 7.0
max_vertical_speed = 10.0
V_adjustment = 0.1

# Initial positions and velocities
rocket_position = np.array([0.0, 0.0, random.uniform(0.0, 1001.0)])
rocket_velocity = np.array([0.0, 0.0, 0.0])

target_position = np.array([random.uniform(-300.0, 301.0), random.uniform(-300.0, 301.0), random.uniform(100.0, 501.0)])

target_velocity = np.array([random.uniform(-5.00, 5.10), random.uniform(-5.00, 5.10), 0.0])

theta = [0.0]
phi = [0.0]

rocket_path = rocket_position.reshape(1, -1)
target_path = target_position.reshape(1, -1)
```

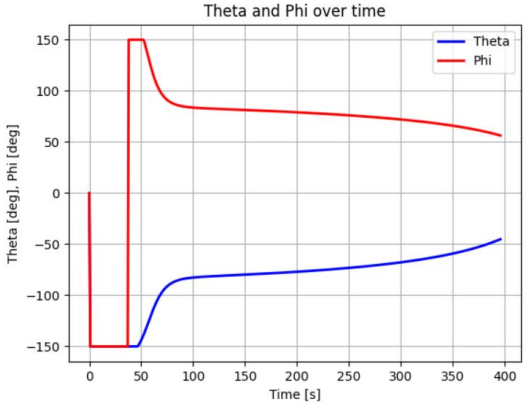
Python Code for Target Tracking

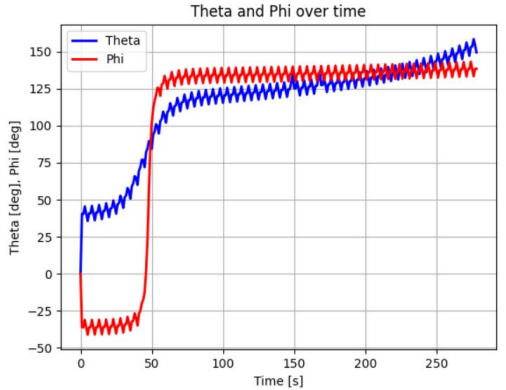
Adjusting Fins

```
phi.append(phiNext)
servo3Angle = phiNext / 5.0 + 90.0
servo4Angle = -phiNext / 5.0 + 90.0
if abs(phiNext / 5.0) < 5:
    servo3Angle = 2.0 * phiNext / 5.0 + 90.0
    servo4Angle = -2.0 * phiNext / 5.0 + 90.0
if abs(phiNext / 5.0) > 30:
    servo3Angle = 0.5 * phiNext / 5.0 + 90.0
    servo4Angle = -0.5 * phiNext / 5.0 + 90.0

kit.servo[SERVO_3_CHANNEL].angle = int(servo3Angle)
kit.servo[SERVO_4_CHANNEL].angle = int(servo4Angle)

# Update positions based on velocity
rocket_position += rocket_velocity * step_interval
rocket_path = np.vstack([rocket_path, rocket_position])
target_position += target_velocity * step_interval
target_path = np.vstack([target_path, target_position])
```

Python Plotting

```
# Theta and phi graphs
fig2, ax2 = plt.subplots()
ax2.grid(True)
ax2.set_xlabel('Time [s]')
ax2.set_ylabel('Theta [deg], Phi [deg]')
ax2.set_title('Theta and Phi over time')
ax2.plot(range(endStep + 1), theta, '-b', linewidth=2, label='Theta')
ax2.plot(range(endStep + 1), phi, '-r', linewidth=2, label='Phi')
ax2.legend(loc='best')
plt.show()
```





Questions or Feedback?

Thank You!