Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Position Control of a Ping-Pong ball on a Pivot Mechanism

Team Members:

- Chris Szikszai
- Alex Reed

Project Synopsis & Expected Functionality:

The goal of this project was to display control of a physical fourth order system. To do this we took the route of controlling two consecutive second order systems in an inner-loop, outer-loop architecture.

For the inner-loop second order system we built a fan-actuated pivot mechanism with a tube as the lever. Inside of the tube is a ping-pong ball which acts as the plant for the outer-loop second order system. Inner-loop angular control of the tube results in the cascading actuation of the position of the ball within the tube, linking the two second order systems into a fourth order system.

Control of the inner-loop, angular position of the tube is described by the following second order equation: $J\theta'' + b\theta' = \tau$, where J is the rotational inertia and b is the damping constant. Control of the outer-loop linear position of the ball inside the tube is described by the following second order equation: $mx'' + bx' = mg\theta$, where g is the gravitational constant (9806.65mm/s^2), and m is the mass of the ping-pong ball. Here, a small-angle approximation was used to remove the nonlinear sine term which would be present on the right side of the equation.

To physically implement these cascading, double second order systems, we used two air-blowers controlled differently via PWM to actuate the angular position of the tube. The angular position of the tube in-turn actuates the linear position of a ping-pong ball within the tube. The angular position of the tube is measured with a rotary encoder and the linear position of the ball is measured with a time-of-flight sensor.

The PWM signals to the blowers are controlled digitally using two subsequent PID loops using the feedback of the ball position and the tube angular position. A rough Concept of Operations is shown in ***Figure 1*** below.

Additionally, a GUI was designed to visualize live feedback of the angular and linear positions and to provide a view of the control influences of the Proportional and Integral terms for each of the PI loops over time.

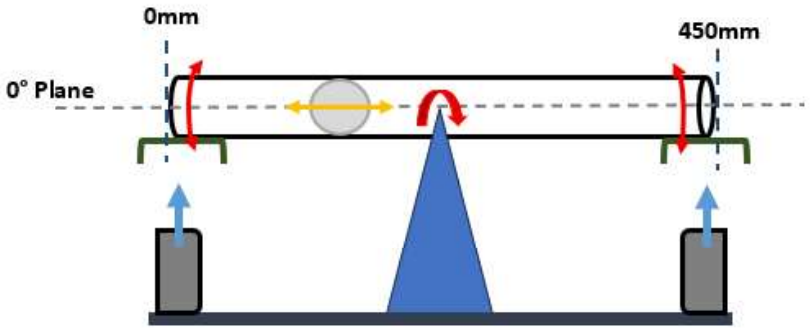


Figure 1. *Visual Concept of Operations*

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

System Overview:

Images of our system are shown below with some annotation of the key components. An in-depth discussion of the design is provided in the next section.

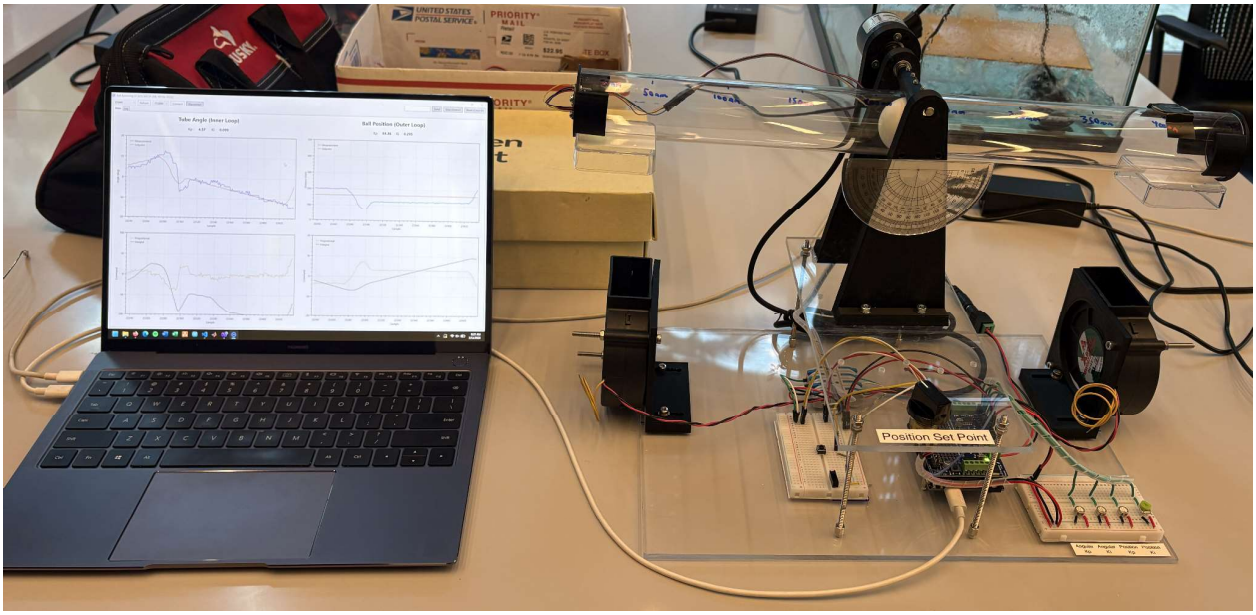


Figure 2. *Our fourth order ping-pong ball pivot system shown on the right and the accompanying control GUI shown on the left of the image.*

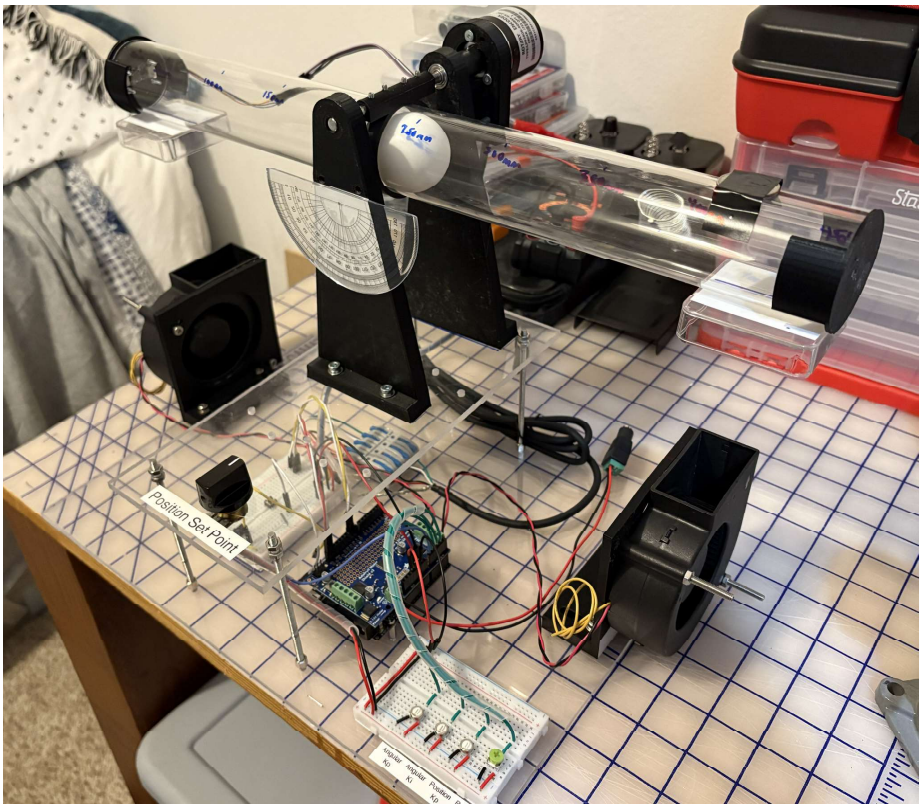


Figure 3. *Isometric view of ping-pong ball pivot system*

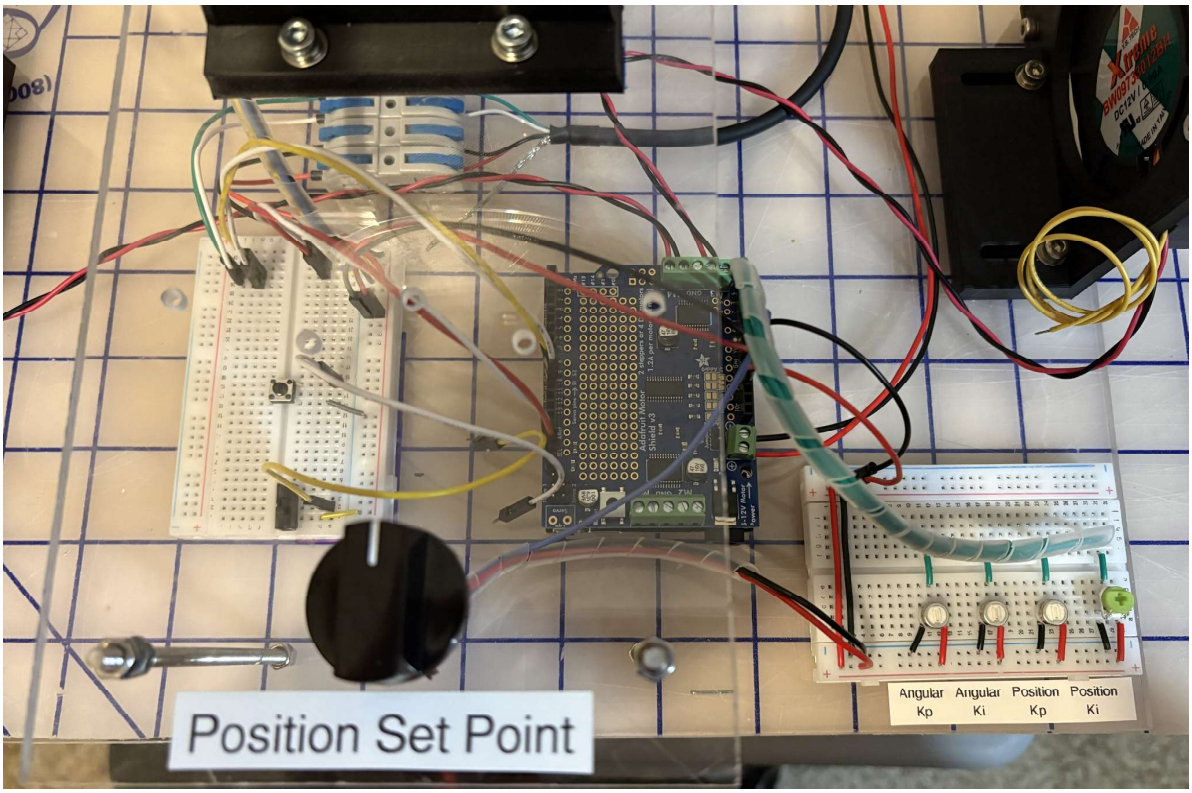


Figure 4. *Close-up view of the PCB, sensor wiring, adjustable set-point, and adjustable PI gains.*

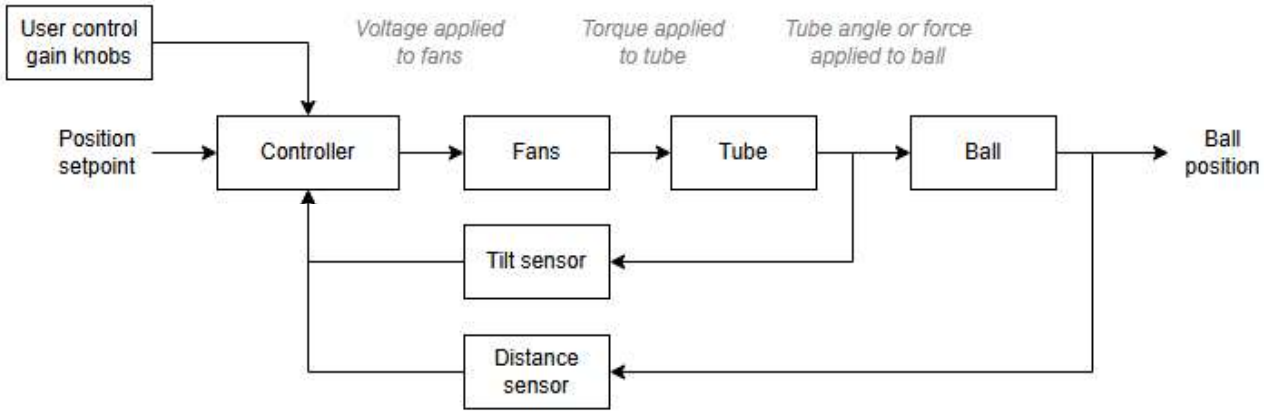


Figure 5. *High-Level Component Block Diagram*

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Design Description:

We found that success on this project was rooted in having a robust and mechanically stable physical system to control. This enabled repeatability in controlled response and allowed us to develop an effective control system. During the design process, we began by developing a functional inner-loop second order system before adding the outer-loop second order system. We had to make some design adjustments along the way, but we were ultimately able to implement our initial design concept.

To build the system we mounted the center of a 18-inch (~450mm) clear tube onto a pivoting shaft. We initially had the shaft located below the tube, which resulted in a center of gravity above the axis of rotation. This acted like an inverted pendulum and resulted in an inherently unstable system, which was by default harder to control. We ended up moving the shaft above the tube, raising the center of gravity below the axis of rotation. This resulted in a standard pendulum-like system which settled to stability and was easier to control.

The shaft was mounted around 260mm above the base plate and was fit into two bearings which enabled low-friction rotation. The shaft height enabled around plus-minus 30° of rotational motion for the tube. On the protruding portion of the shaft, we coupled a rotary quadrature encoder to provide us with angular feedback. This sensor provided us with quick and accurate angular feedback without adding rotational friction.

Table 1. *Specifications for the Rotary Quadrature encoder.*

Operating Voltage	Resolution	Output
5 VDC	600 pulses/revolution	Digital, 2-phase

We mounted two 12VDC, 0.9A air-blowers onto the acrylic baseplate under the ends of each side of the tube. The blowers were controlled differentially via PWM signals from our controller and used to actuate the angular position of the tube. The controller was using a double PI-control architecture which is further explained in the next section. To increase the lift delivered by the fans, we added small “sails” to the ends of the tube to collect the airflow.

Model No.	Bearing	Rated Voltage	Operating Voltage Range	Speed	Max. Airflow	Max. Static Pressure	Current	Power Consumption	Life at 40°C L10	P-Q Curve	Noise Level
		VDC	VDC	RPM	CFM	mm-H_2O	mA	W	Hour		dB(A)
BW09733012BL	2B	12	7~13.2	2600	25.6	22.1	400	4.80	80000	3	40.0
BW09733012BM	2B		7~13.2	2900	28.5	24.7	480	5.76	80000	2	42.5
BW09733012BH	2B		7~13.2	3300	32.4	28.1	740	8.88	75000	1	48.5
BW09733024BL	2B	24	12~26.4	2600	25.6	22.1	210	5.04	80000	3	40.0
BW09733024BM	2B		12~26.4	2900	28.5	24.7	280	6.72	80000	2	42.5
BW09733024BH	2B		12~26.4	3300	32.4	28.1	420	10.08	75000	1	48.5

Figure 6. *Datasheet for the YSTech BW09733012BH blowers.*

Inside of the tube we had a free rolling ping-pong ball, which acted as the plant for the outer-loop system. A ping-pong ball was chosen as the plant as it has low mass and inertia, which helped us keep the outer-loop system slower than the inner-loop system. To further “dampen” the outer-loop system, we added endcaps that almost completely enclosed the tube. This added an air-buffer to dampen the ball movement within the tube.

For positional feedback of the ball we ended up using a VL53L4CX time-of-flight sensor. This had an advertised range of 0-6,000mm. Practically, we found the sensor to have a lower-limit range of 20mm when used to sense the ping-pong ball. We didn't have any issues with the upper-limit range of the sensor as our tube length maxed out at 450mm. The ToF sensor was mounted to one of the endcaps and provided us with a digital output.

Table 1. Technical specification

Feature	Detail
Package	Optical LGA12
Size	4.4 x 2.4 x 1 mm
Operating voltage	2.6 to 3.5 V
Operating temperature	-30 to 85°C
Infrared emitter	940 nm
I²C	Up to 1 MHz (fast mode plus) serial bus Address: 0x52

Figure 7. *(Left) Image of the unmounted VL534CX time-of-flight sensor. (Right) Technical Specs*

We initially tried using two IR analog distance sensors for positional feedback of the ping-pong ball. These sensors had a working range of 10-80cm (4"-30"), which forced us to use them in a differential configuration to accurately monitor the position of the ball over the whole length of the tube. We had issues with this as the IR beam seemed to bounce off the tube walls resulting in crosstalk between the two sensors. Ultimately, the digital output, lighter weight, and higher precision of the time-of-flight sensor made it a better choice of sensor.

For the microcontroller we ended up using an Adafruit Metro ESP32-S3 paired with an Adafruit V3 Motor Shield on top. The Adafruit Metro provided us the ability to supply both 3.3V and 5V and provided 16-bit resolution for analog readings. The addition of the Adafruit V3 Motor Shield allowed us to supply a 12V PWM signal to the blowers.

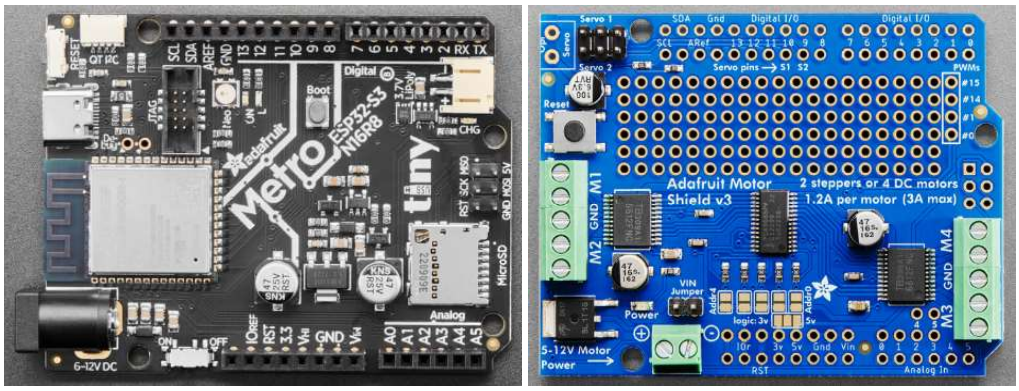


Figure 8. *(Left) Adafruit Metro ESP32-S3. (Right) Adafruit V3 Motor Shield*

Additionally, a simple UI was designed to provide visualization of the tube angular position, the ball position as well as Kp and Ki control contributions for both inner and outer loop control. To provide the user the ability to control the ball setpoint and the Kp/Ki gains for each loop we mounted potentiometers with knobs to the baseplate.

Control System Design & Performance:

To control this system, we implemented two PID control loops, first around the angle of the tube, and second around the position of the ball. The control logic was implemented in Python on the Adafruit Metro controller using their CircuitPython libraries for ease of prototyping.

The general control architecture involves the previously mentioned inner-loop / outer-loop architecture. The loops are characterized by the feedback of the angular theta-value, being read from the rotary encoder and the linear x-value read from the time-of-flight sensor.

The control system begins with a desired x-position setpoint, which is physically set via a single-turn potentiometer. The “desired x-position” is then compared to the actual x-position feedback from the ToF sensor, resulting in the “x-position error”. The error is then fed into the Outer-Loop PID controller which outputs a “desired theta” value or *q_des*. The desired theta is compared to the actual theta, read from the rotary encoder, and results in the “theta error” being output. The theta error is then fed into the Inner-Loop PID controller which outputs fan torque in the form of a PWM signal.

The fan torque then feeds into the Tube Dynamics model which is modeled by the equation: $\tau = J\theta'' + b\theta'$. The resulting output of the tube dynamics is the “actual theta” value which then fed into the Ball Dynamics model is modeled by the equation: $mg\theta = mx'' + bx'$. The output of the Ball Dynamics model is the x-position of the ping-pong ball.

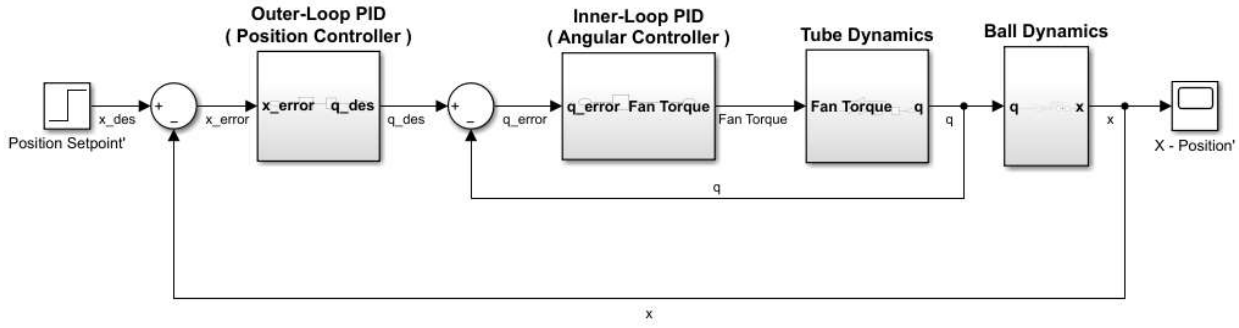


Figure 9. *Control System block diagram*

Similar to how we developed the hardware system, we also designed and tested the control system in steps. We began by implementing control over the full system angular range of +/- 30° from the center line. After proving successful angular control of the tube down to ~1° we added the outer loop control of the ping-pong ball position. For the control of the ball position we rarely saw the system exceed an angular range of +/- 10° from the center line. This makes sense as extreme angular change would result in drastic changes in ball position within the tube.

Prior to testing, we expected the position response of the ball to be on the order of 10-20 seconds for any setpoint within the tube’s 20mm-450mm range.

Software Design

This system was developed with two major software components: the embedded software for the microcontroller, and a Windows application (GUI) for the user’s viewing and monitoring pleasure.

The embedded software was written in Python and installed and run on the microcontroller using the CircuitPython architecture that was provided by Adafruit. Given the processing power of the ESP32 chip, and considering the 10-20 second settling time goal, extreme low-latency control loops were not a major priority, and so the relative inefficiencies of Python were not a concern. The primary control loop was run at 50 Hz, and while this appeared functional, the exact speed wasn't verified thoroughly. A slower loop was run at 5 Hz to send telemetry data over USB to the Windows GUI, as well as read the analog inputs to adjust the position setpoint and adjustable gains. See the appendix for a flow chart as well as the full text of the Python script used.

The Windows GUI was written in C# using Visual Studio and displayed to the user four plots: the position and setpoint for the ball, the PID control efforts for the position (outer) loop, the angle and setpoint for the tube, and the PID control efforts for the angle (inner) loop. All four plots were updated at 5 Hz and scrolled to show the latest 10 seconds of data. The current Kp and Ki gains were displayed as well. A log of telemetry data was stored in plain text in a secondary tab, making data collection simple. Screenshots of the GUI with example data are shown in the appendix.

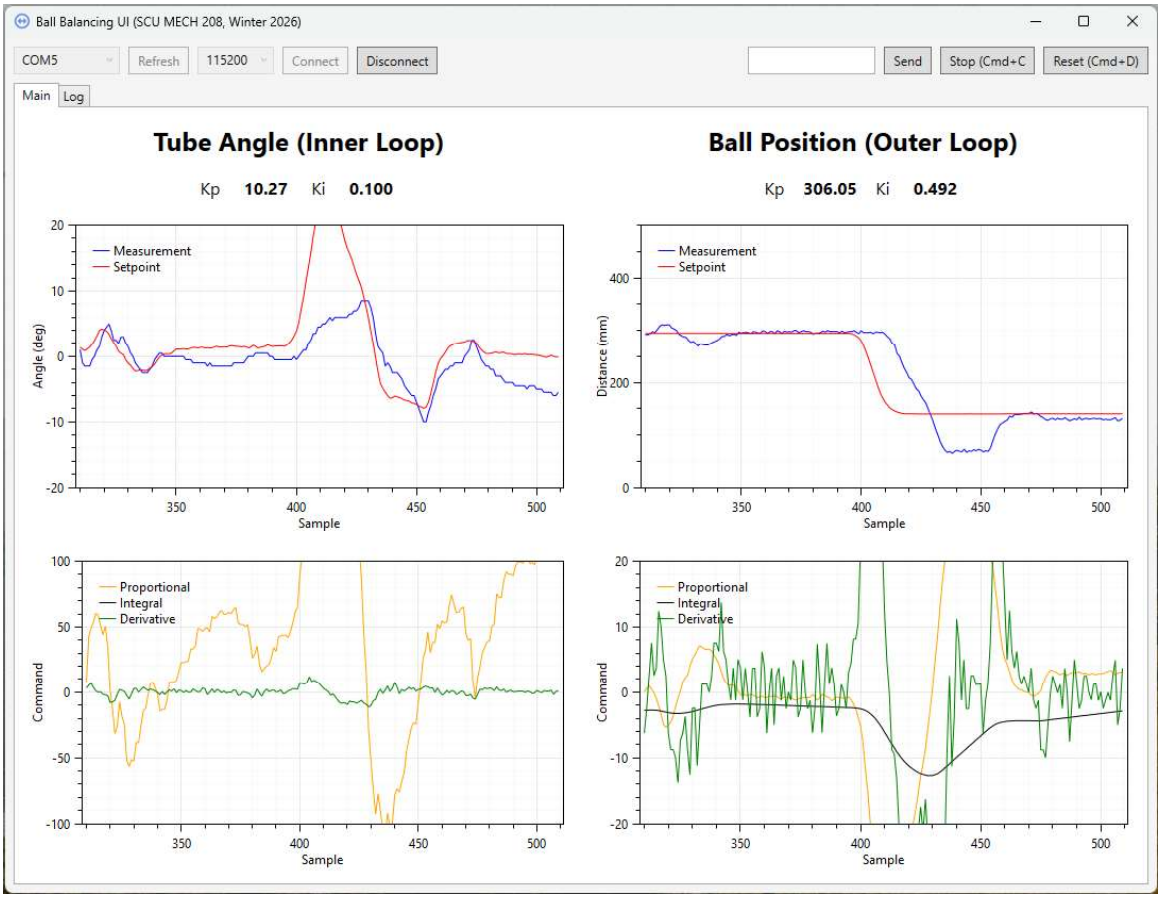


Figure 10. *Screenshot of the custom GUI*

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Simulink Modeled Behavior

To accurately build a Simulink Model of our system, we began by modeling the Inner Loop control architecture. The PID block used has Kp = 5, Ki = 0.1, and Kd = 0, which reflects our initial design choice of implementing a PI controller. For the Tube's Angular Dynamics, we estimated a damping coefficient value of 2 and an inertia value of J = ½.

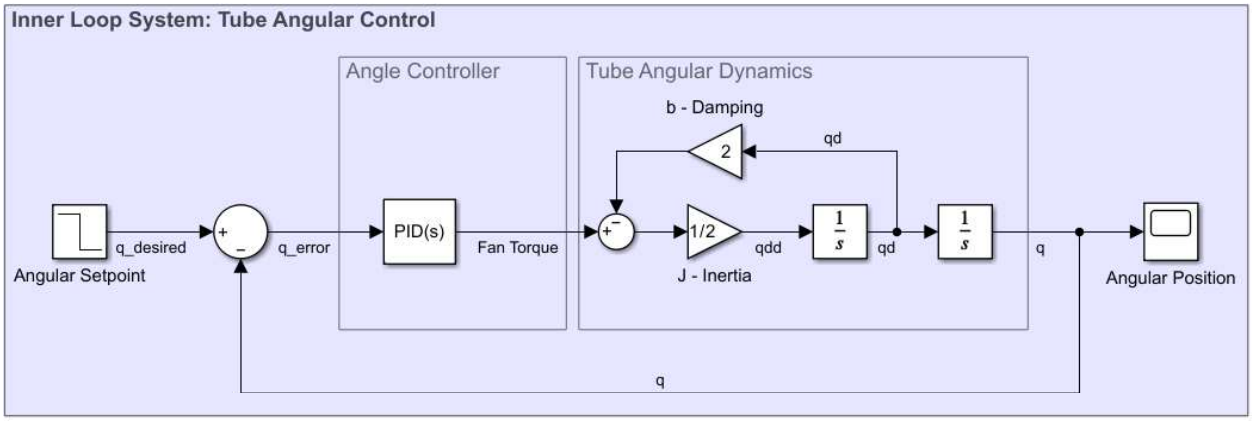


Figure 11. *Simulink Model of the Inner Loop control architecture.*

For the simulation, a step input representing the Angular Setpoint was set to an initial value of 0 radians (0°), a final value of pi/180 radians (+1°), and a step time of 1sec. Simulation plots can be seen below for a couple of Kp and Ki values.

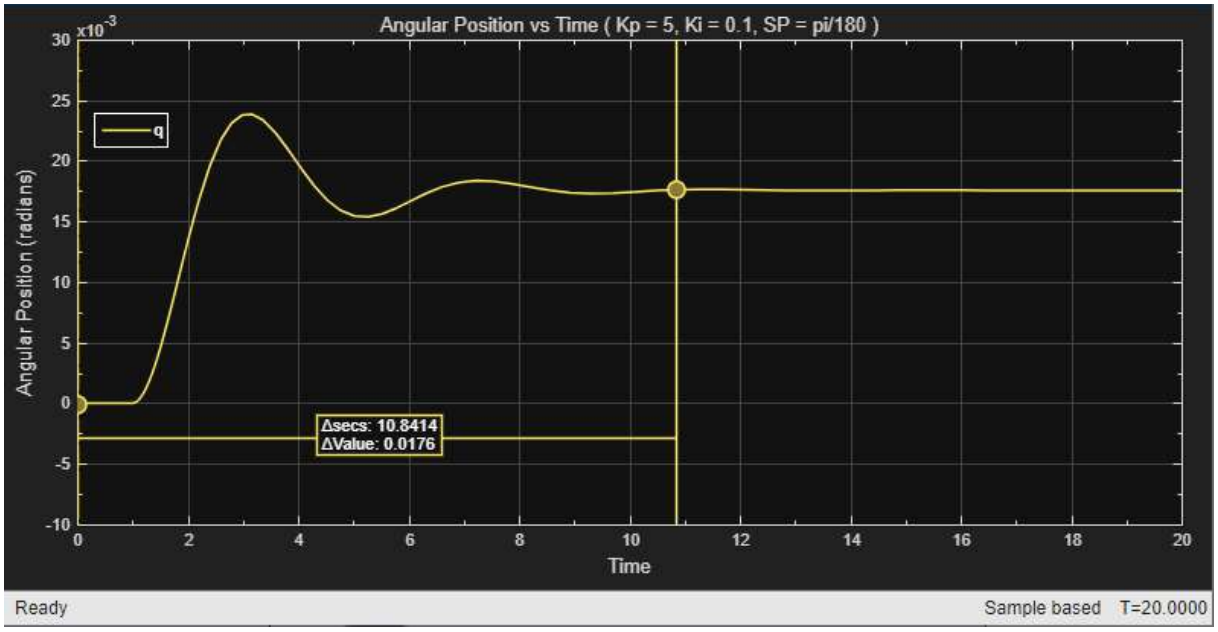


Figure 12. *Inner Loop control simulation. Kp = 5, Ki = 0.1, SP = pi/180*

For the second plot of the inner loop simulation, the overshoot and the oscillation were practically eliminated by decreasing Kp from 5 -> 2 and by adding a Kd term. Steady State of pi/180rad, or 1° was reached within 6.5 seconds.

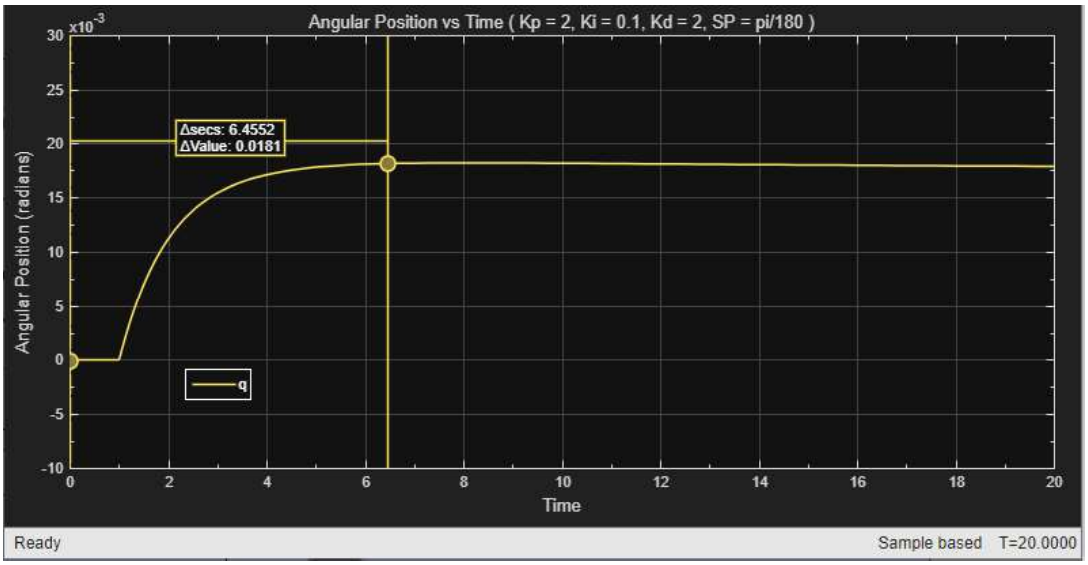


Figure 13. *Inner Loop control simulation. Kp = 2, Ki = 0.1, Kd = 2, SP = pi/180*

Next, the *Ball Dynamics* were added to the Simulink system and produced realistic looking results. The angular set-point was still set to +1° with no position feedback. In the plot you can see the ball starts at the lower-limit position of 20mm and rolls to the upper-limit of 450mm after around 12sec.

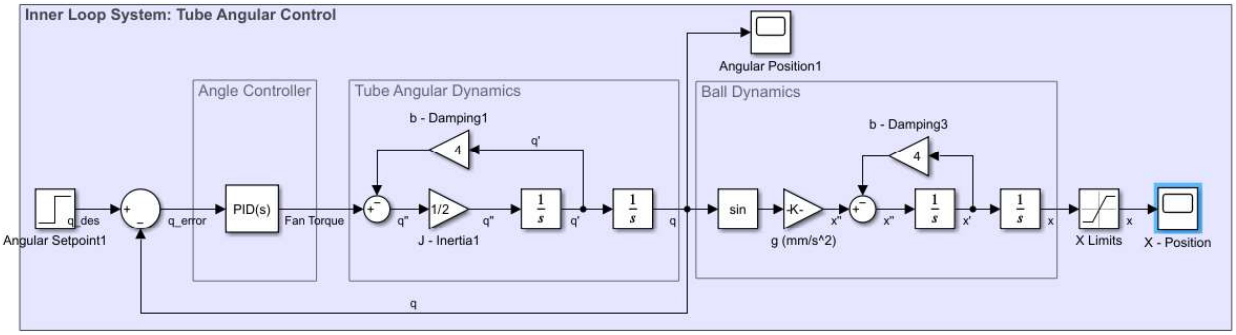


Figure 14. *Simulink Model with Inner Loop as well as the Tube Dynamics and Ball Dynamics.*

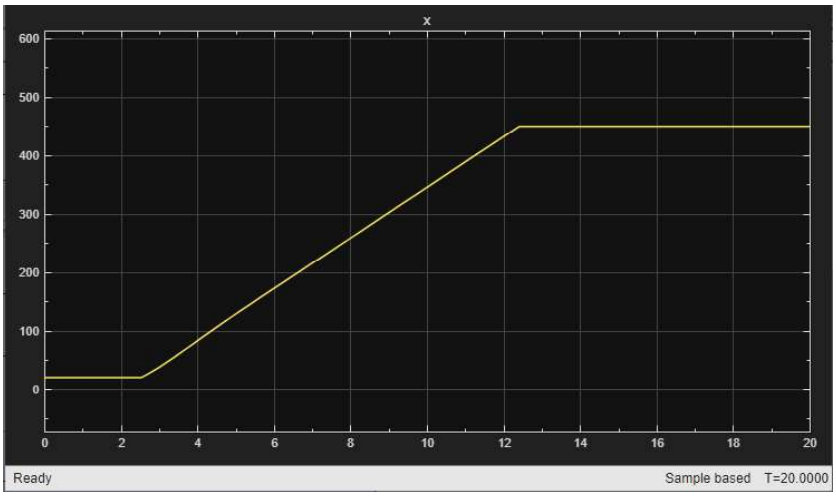


Figure 15. *Inner Loop control simulation w/ Tube & Ball Dynamics. Kp = 2, Ki = 0.1, SP = pi/180*

Next, the outer loop PID position controller was added to the model and the angular setpoint was replaced with a position setpoint. The input for the model is the desired x-position setpoint and the output of the model is the simulated x-position. Tube damping (b) = 0.4, Tube inertia (J) = 0.4. Ball damping (b) = 1.5.

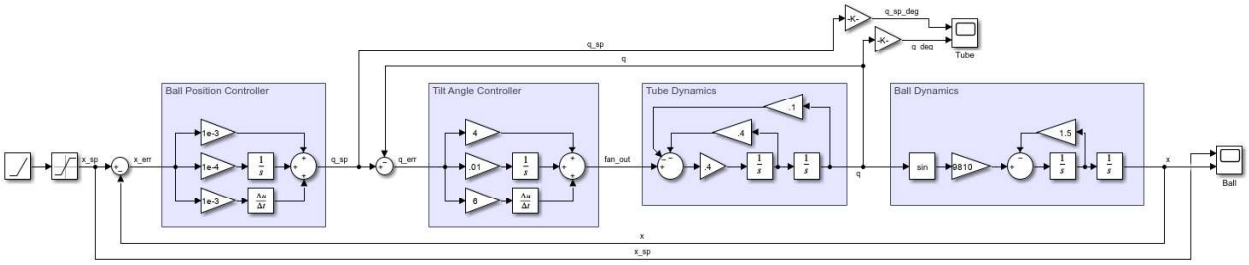


Figure 16. *Simulink Model with Inner Loop & Outer Loop PID control, Tube Dynamics and Ball Dynamics.*

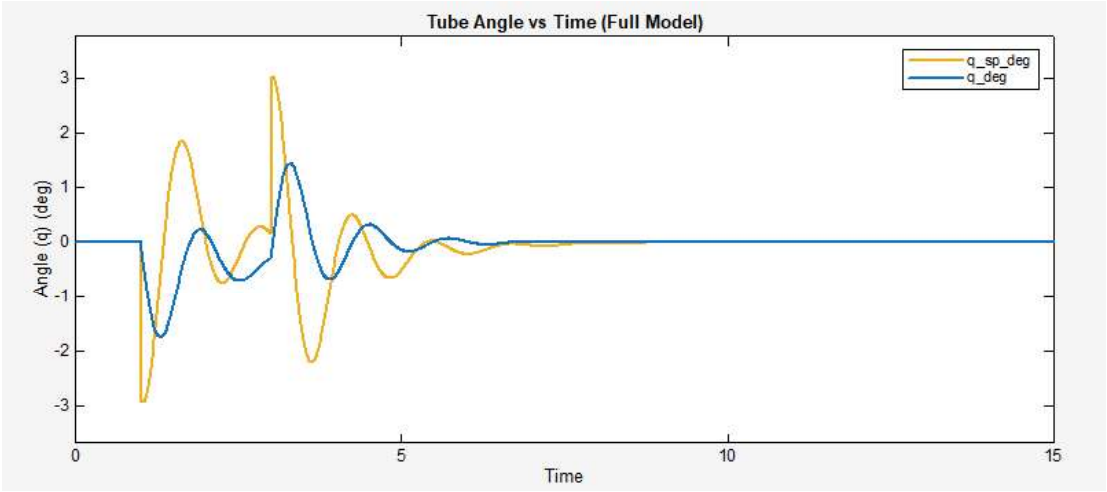


Figure 17. *Simulated plot of Tube Angle vs Time (Tube Angular Position). Position Setpoint = 100mm. Outer Kp=0.001, Ki=0.0001, Kd=0.001; Inner Kp=4, Ki=0.01, Kd=6.*

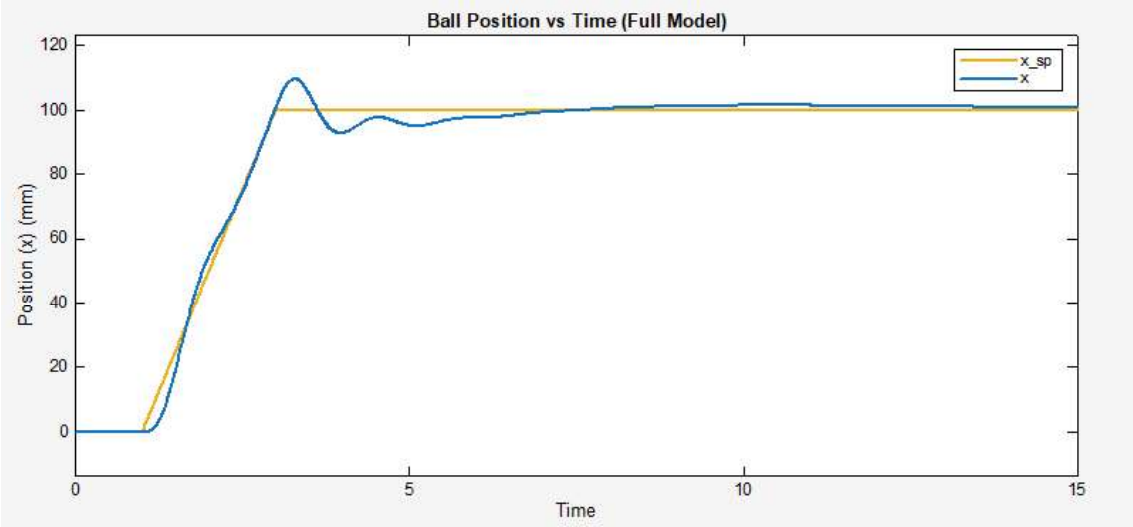


Figure 18. *Simulated plot of Ball Position vs Time (Ball X-Position). Position Setpoint = 230mm. Outer Kp=0.001, Ki=0.0001, Kd=0.001; Inner Kp=4, Ki=0.01, Kd=6.*

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

System Response Behavior – Experimental Data

For the following plots of experimental data, we tested our system with the tube angle starting around 0° and the ball starting in a position around the center of the tube (150-300mm). The position setpoint was then changed by +50mm, +100mm, +150mm and the response was measured for both the Tube Angle and the Ball Position. We also plotted the control efforts from the P, I, and D terms over time. The gains selected were as follows: Position: Kp = 0.25, Ki = 0.0005, Kd = 0.5; Angular: Kp = 10, Ki = 0.1, Kd = 0.5.

Table 2. *Summary of step response performance.*

Setpoint Delta	**50mm**	**100mm**	**150mm**
Typ. overshoot	0%	50%	30%
Typ. steady-state error	10%	3%	5%
Typ. settling time	6 sec	10 sec	10 sec

***Step Response Data*:** Setpoint Change +50mm

Figure 19. *Setpoint Δ = +50mm. (Top Left) Tube Angle vs Time. (Top Right) Tube Angle control efforts. (Bot Left) Ball Position vs Time. (Bot Right) Ball Position control efforts.*

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

***Step Response Data*: Setpoint Change +100mm**

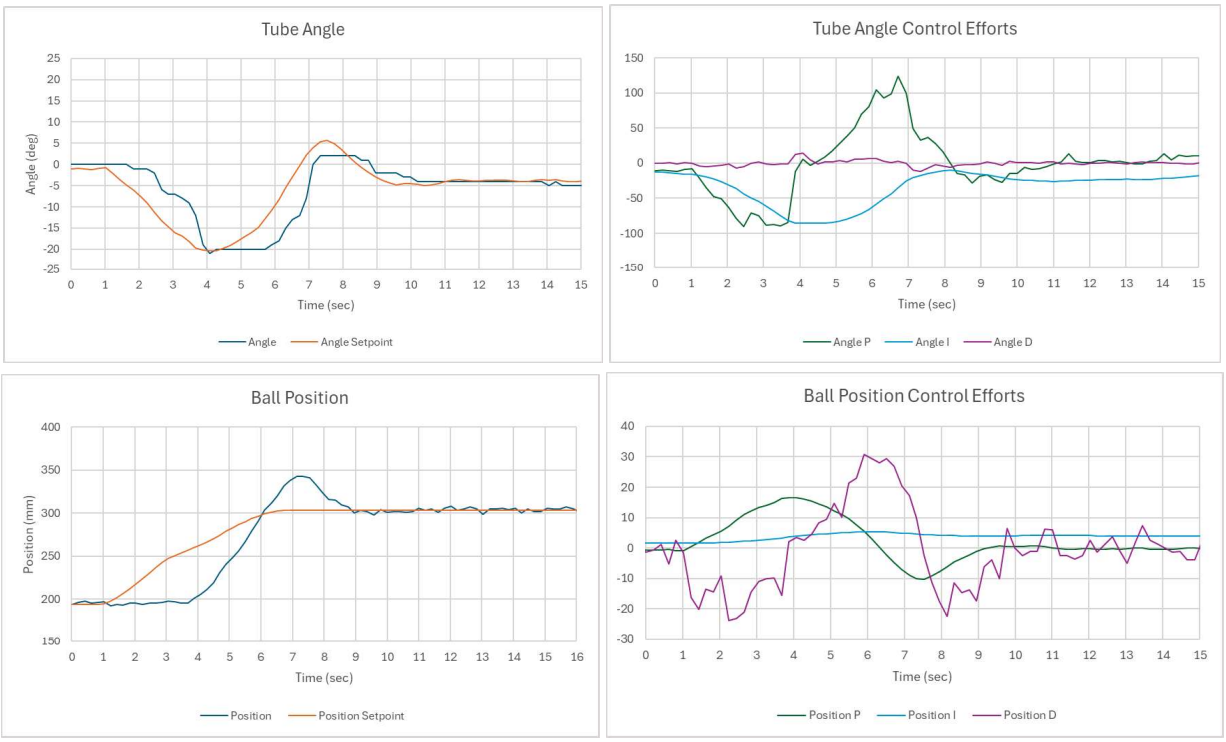


Figure 20. *Setpoint Δ = +100mm. (Top Left) Tube Angle vs Time. (Top Right) Tube Angle control efforts. (Bot Left) Ball Position vs Time. (Bot Right) Ball Position control efforts.*

***Step Response Data*: Setpoint Change +150mm**

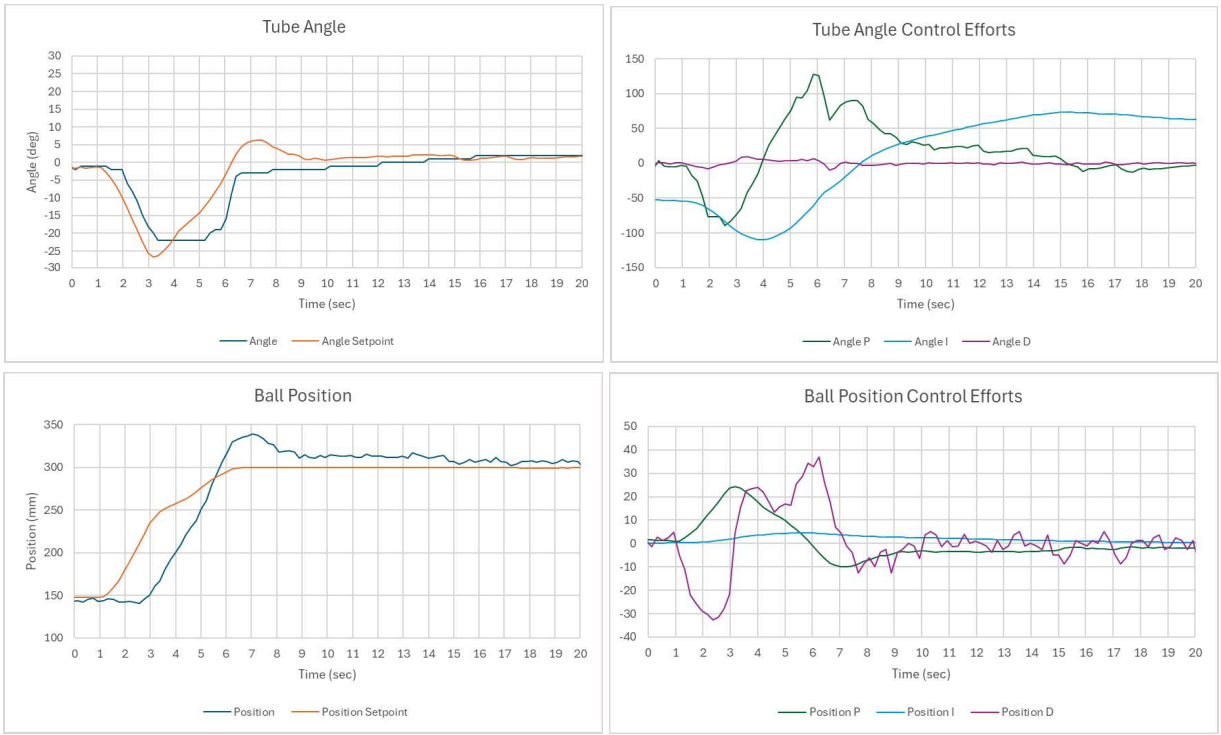


Figure 21. *Setpoint Δ = +100mm. (Top Left) Tube Angle vs Time. (Top Right) Tube Angle control efforts. (Bot Left) Ball Position vs Time. (Bot Right) Ball Position control efforts.*

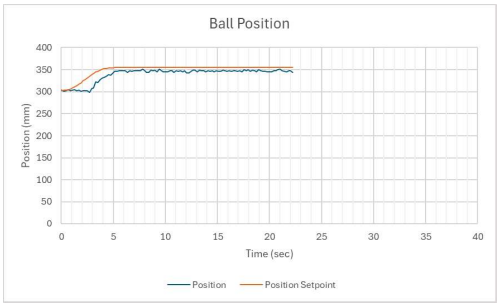
Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Setpoint Restriction Near Pivot

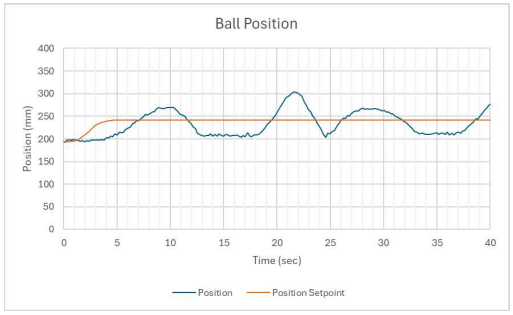
As expected of this less-than-precision system, the performance varied across the range of motion due to a number of factors, including but not limited to the roundness of the ball, the flatness of the tube surface, the inconsistent friction between the tube pivot shaft and the 3D-printed frame, and the asymmetric force from the wire harness of the distance sensor attached at the 0mm end of the tube. We chose PID gains to optimize across the variations as much as possible.

However, we anticipated that performance near the pivot (230mm) would be different from performance elsewhere due to the geometry of the system, and testing confirmed. The pivot, although offset from the axis of the linear ball motion, still resembled a kinematic singularity, where no applied tilt can generate any force on the ball. The offset prevents this case exactly, but the dynamics were different enough from the rest of the system that we were not able to find a compromise on PID terms. See below for the resulting instability around 230mm. For this reason, we recommend any future user of this system restrict their commanded setpoints to 0-200mm or 260-450mm.

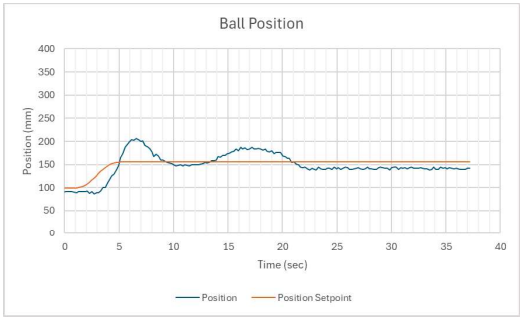
50mm Step, Initial Position 300mm



50mm Step, Initial Position 200mm



50mm Step, Initial Position 100mm

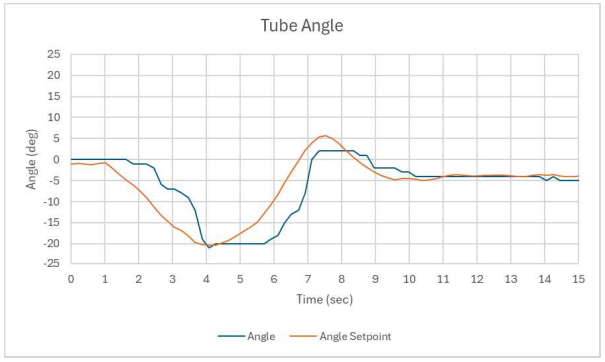
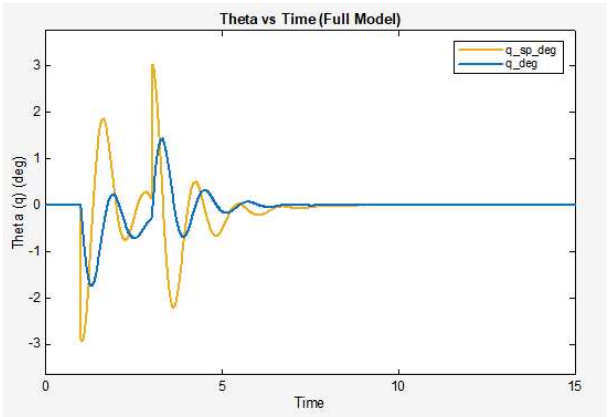
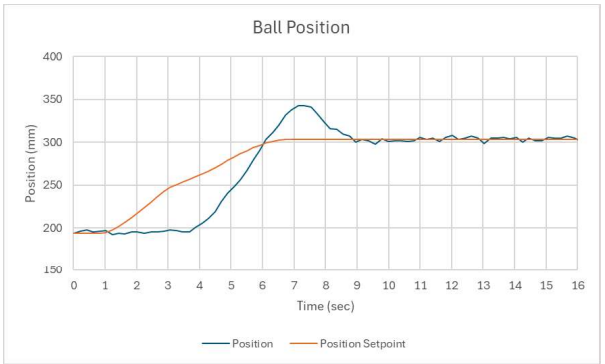
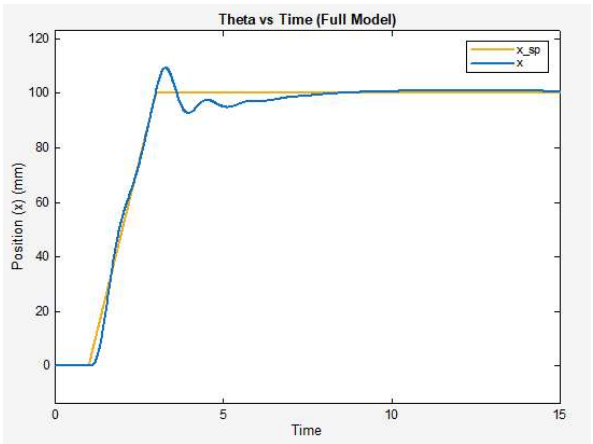


Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

System Response Behavior – Simulated vs Experimental Data

Below is a head-to-head comparison of the predicted and experimental results. I am comparing the experimental "Step Response data: +100mm" to the simulink model.

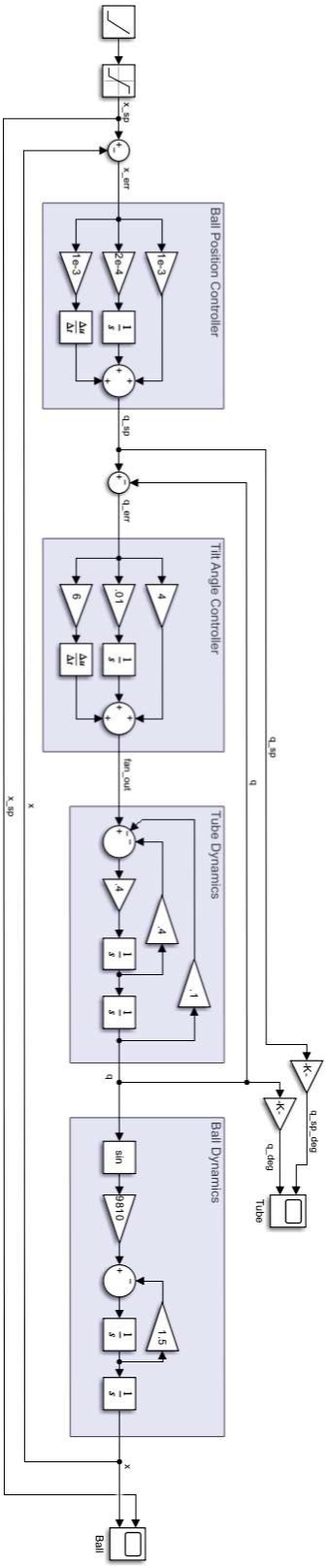
Table 3. *Comparison of the Experimental Results to the Simulated Results of our system behavior*

Experimental Results	**Simulated Results**
- Initial Angle: ~0° - Initial Position: 200mm - Position Setpoint: 300mm - Position: Kp = 0.25, Ki = 0.0005, Kd = 0.5 - Angular: Kp = 10, Ki = 0.1, Kd = 0.5 *Position:* - Overshoot: ~25% - S.S. = 10 seconds - S.S. Offset +2%	- Initial Angle: ~0° - Initial Position: 0mm - Position Setpoint: 100mm - Position: Kp = 0.001, Ki = 0.0001, Kd = 0.001 - Angular: Kp = 6, Ki = 0.01, Kd = 6 *Position:* - Overshoot: 10% - S.S. = 10 seconds - S.S. Offset: ~1%
	
	

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

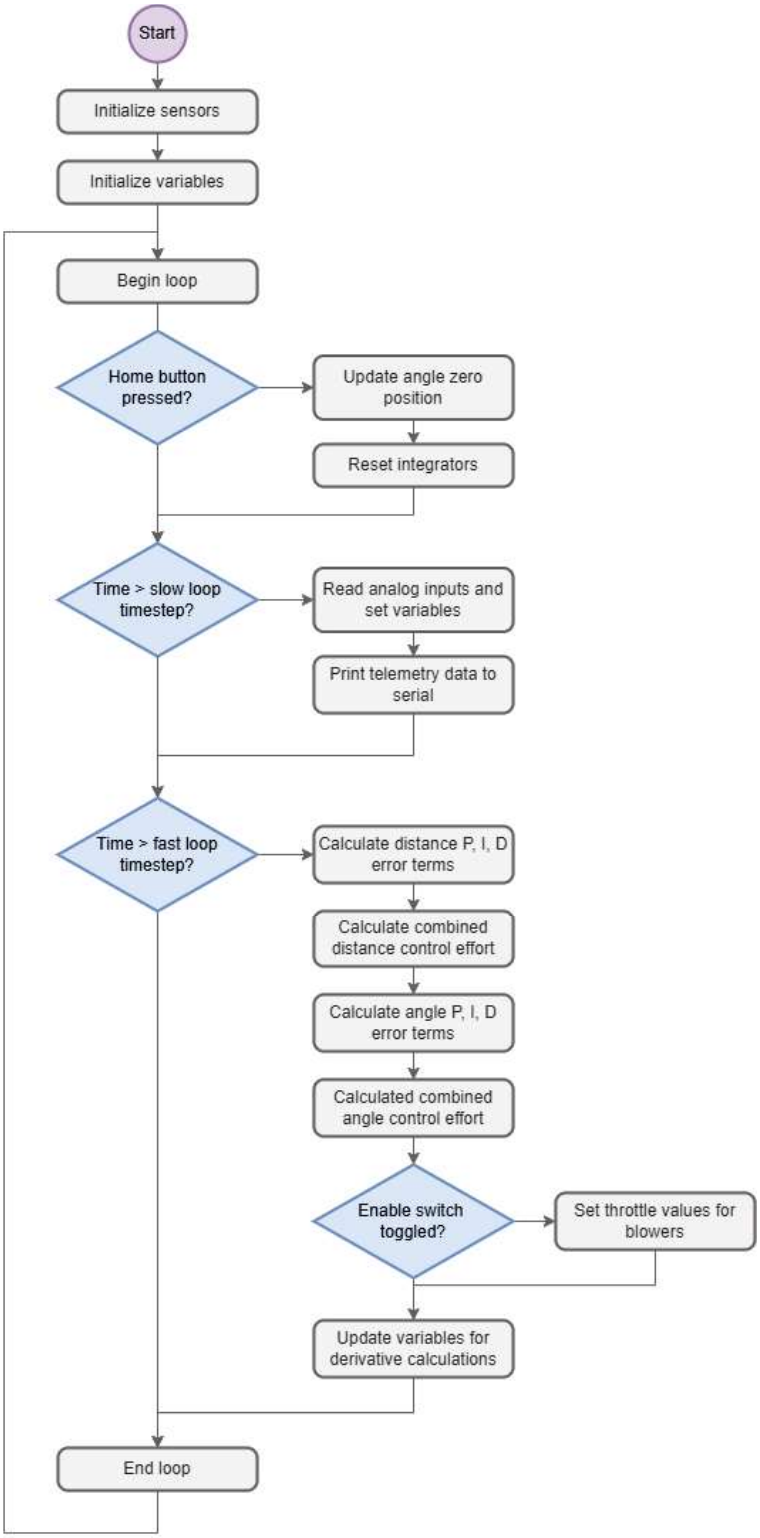
Appendix:

Simulink Model



Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Embedded Software Flow Chart



Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Embedded System Text (Python)

```
import time
from collections import deque
import board
from analogio import AnalogIn
from digitalio import DigitalInOut, Direction, Pull
import vl53l4cx
import rotaryio
from adafruit_motorkit import MotorKit
import usb_cdc

# SIMULATION
SIMULATION = False

# DIO / AIO
home_switch = DigitalInOut(board.D13)
home_switch.direction = Direction.INPUT
home_switch.pull = Pull.UP

enable_switch = DigitalInOut(board.D12)
enable_switch.direction = Direction.INPUT
enable_switch.pull = Pull.UP

sp_dial = AnalogIn(board.A0)
dist_kp_dial = AnalogIn(board.A3)
dist_ki_dial = AnalogIn(board.A4)
ang_kp_dial = AnalogIn(board.A1)
ang_ki_dial = AnalogIn(board.A2)

# Serial
serial = usb_cdc.console

# Motor Shield
kit = MotorKit()
kit = MotorKit(i2c=board.I2C())
kit.motor3.throttle = None
kit.motor4.throttle = None
motor3_sign = -1
motor4_sign = -1
motor_offset = -0.65

# Encoder
home_offset = 0
if not SIMULATION:
    encoder = rotaryio.IncrementalEncoder(board.D10, board.D9, divisor=1)

# Distance Sensor
last_distance_measurement = 0
if not SIMULATION:
    i2c = board.I2C()  # uses board.SCL and board.SDA
    # i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
    vl53 = vl53l4cx.VL53L4CX(i2c)
    vl53.distance_mode = 1
    vl53.timing_budget = 20
    model_id, module_type, mask_rev = vl53.model_info
    vl53.start_ranging()

# Control Parameters
ang_kp = 10
ang_kd = 0.5
ang_ki = 0.1

dist_kp = 0.25
dist_kd = 0.5
dist_ki = 0.0005
```

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

```
# Control Variables
def_ang_kp = 10
def_ang_kd = 0.5
def_ang_ki = 0.1

def_dist_kp = 0.25
def_dist_kd = 0.5
def_dist_ki = 0.0005

ang_err_p = 0
ang_err_p_prev = 0
ang_err_pf = 0
ang_err_pf_prev = 0
ang_err_d = 0
ang_err_i = 0
ang_err_i_max = 10000

dist_err_p = 0
dist_err_p_prev = 0
dist_err_pf = 0
dist_err_pf_prev = 0
dist_err_d = 0
dist_err_i = 0
dist_err_i_max = 100000

ang_out_p = 0
ang_out_i = 0
ang_out_d = 0
dist_out_p = 0
dist_out_i = 0

ang_buffer = deque((), 20)
pos_buffer = deque((), 20)
pos_sp_buffer = deque((), 80)

last_update = time.monotonic()
update_interval = 0.02

last_slow_update = time.monotonic()
slow_update_interval = 0.2

def GetAngle():
    if not SIMULATION:
        ang = encoder.position - home_offset
    else:
        ang = 0
    return ang

def UpdateAngleQueue(ang):
    global ang_buffer
    ang_buffer.append(ang)
    return

def GetFiltAngle():
    return sum(ang_buffer) / len(ang_buffer)

# def GetAngleSetpoint():
#     sp = sp_dial.value / 65536 * 100 - 50
#     return sp

def GetPosition():
    if not SIMULATION:
        pos = last_distance_measurement
    else:
        pos = 200
    return pos

def UpdatePositionQueue(pos):
    global pos_buffer
```

```
    pos_buffer.append(pos)
    return

def GetFiltPosition():
    return sum(pos_buffer) / len(pos_buffer)


def GetPositionSetpoint():
    sp = sp_dial.value / 65536 * 400
    return sp

def UpdatePositionSetpointQueue(pos):
    global pos_sp_buffer
    pos_sp_buffer.append(pos)
    return

def GetFiltPositionSetpoint():
    return sum(pos_sp_buffer) / len(pos_sp_buffer)

def GetDistKp():
    # range from 0 to 2x default
    kp = (dist_kp_dial.value / 65536) * def_dist_kp * 2
    return kp

def GetDistKi():
    # range from 0 to 2x default
    ki = (dist_ki_dial.value / 65536) * def_dist_ki * 2
    return ki

def GetAngKp():
    # range from 0 to 2x default
    kp = (ang_kp_dial.value / 65536) * def_ang_kp * 2
    return kp

def GetAngKi():
    # range from 0 to 2x default
    ki = (ang_ki_dial.value / 65536) * def_ang_ki * 2
    return ki

def ParseAndExecuteCommand(command):
    print(f"parsing command {command}")
    global ang_kp, ang_ki, dist_kp, dist_ki
    if '=' in command:
        var_name, var_value = command.split('=', 1)
        var_name = var_name.strip()
        var_value = var_value.strip()

        if var_name == "ang_kp":
            ang_kp = float(var_value)

        elif var_name == "ang_ki":
            ang_ki = float(var_value)

        elif var_name == "dist_kp":
            dist_kp = float(var_value)

        elif var_name == "dist_ki":
            dist_ki = float(var_value)

        else:
            print(f"Error: {var_name} not recognized")
    else:
        print("Error: Invalid command format. No '=' found")



while True:
    now = time.monotonic()
```

```
    # Update position data
    if not SIMULATION:
        if vl53.data_ready:
            last_distance_measurement = vl53.distance
            vl53.clear_interrupt()

    # Look for serial data
    if serial.in_waiting > 0:
        incoming = serial.readline()
        try:
            command = incoming.decode().strip()
            ParseAndExecuteCommand(command)

        except Exception as e:
            print(f"Error: {str(e)}")

    # Check home button for reset
    if not home_switch.value:
        if not SIMULATION:
            home_offset = encoder.position
        ang_err_i = 0
        dist_err_i = 0



    # Run slow loop for serial output and gain input
    if (now - last_slow_update > slow_update_interval):
        # global dist_kp, dist_ki, ang_kp, ang_ki
        dist_kp = GetDistKp()
        dist_ki = GetDistKi()
        ang_kp = GetAngKp()
        ang_ki = GetAngKi()
        print(f"{GetPosition()} {GetFiltPositionSetpoint()} {dist_out_p} {dist_out_i} {GetAngle()}
{ang_sp} {ang_out_p} {ang_out_i} {dist_kp} {dist_ki} {ang_kp} {ang_ki} {dist_out_d} {ang_out_d}")
        last_slow_update = now



    # Run fast loop for control
    if (now - last_update > update_interval):

        # Update queues/buffers
        UpdateAngleQueue(GetAngle())
        UpdatePositionQueue(GetPosition())
        UpdatePositionSetpointQueue(GetPositionSetpoint())

        # Position
        dist_err_p = GetFiltPositionSetpoint() - GetPosition()
        dist_err_pf = GetFiltPositionSetpoint() - GetFiltPosition()

        dist_err_d = (dist_err_pf_prev - dist_err_pf) / update_interval
        dist_err_i += dist_err_p

        dist_err_i = max(-dist_err_i_max, min(dist_err_i_max, dist_err_i)) # cap integrator

        dist_out_p = dist_kp*dist_err_pf
        dist_out_i = dist_ki*dist_err_i
        dist_out_d = dist_kd*dist_err_d
        ang_sp = -1 * (dist_out_p + dist_out_i)

        # Angle
        ang_err_p = ang_sp - GetAngle()
        ang_err_pf = ang_sp - GetFiltAngle()

        ang_err_d = (ang_err_pf - ang_err_pf_prev) / update_interval
        ang_err_i += ang_err_p

        ang_err_i = max(-ang_err_i_max, min(ang_err_i_max, ang_err_i)) # cap integrator
```

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

```
ang_out_p = ang_kp*ang_err_p
ang_out_i = ang_ki*ang_err_i
ang_out_d = ang_kd*ang_err_d
T = (ang_out_p + ang_out_d + ang_out_i) / 1000

m3 = motor3_sign * (motor_offset - T)
m4 = motor4_sign * (motor_offset + T)

# print(f"{m3} {m4}")

if (enable_switch.value):
    kit.motor3.throttle = max(0, min(1, m3))
    kit.motor4.throttle = max(0, min(1, m4))
else:
    kit.motor3.throttle = None
    kit.motor4.throttle = None

ang_err_pf_prev = ang_err_pf
dist_err_pf_prev = dist_err_pf
last_update = now
```

Chris Szikszai & Alex Reed
MECH 208
Due: 3/18/2026

Custom Windows GUI Screenshots