Project Progress Report #1

Ethan Thompson, Jared Porter, and Mary Thompson

Literature Review:

The ball-and-beam system is a widely studied example for understanding feedback control principles. This system's primary challenge is maintaining the ball's position on a pivoted beam, which requires precise measurement and control of its motion. Ultrasonic sensors and PID controllers have been extensively utilized in this context to enhance accuracy and responsiveness.

Găşpăresc (2014) examined the effectiveness of Parallax ping ultrasonic sensors that will be used in this project finding an accuracy rating of 0.3 cm and ±0.5 cm. Considering the beam in this project is about 48 cm long, this accuracy rating should suffice.

Saad (2017) as well as many, many other groups have researched the design and Implementation of a ball-beam controller using PID algorithms, therefore this project is not novel in that respect (although novel to us in our efforts to create it), what will be novel is the use of spools and thread paired with servo motor to control the torque of the beam.

Project Timeline:

- We have successfully coded the process simulation
- We have modeled and 3D printed the beam of the system
- We have found the the equations surrounding the angles

Currently we are designing the base of the system which will hold the thread controlling the torque of the rotation. An uncertainty is simply how the spools of thread will be integrated into the base of the system. We also have yet to see if the servo motor can offer the torque necessary to adequately rotate the beam.

Simulation

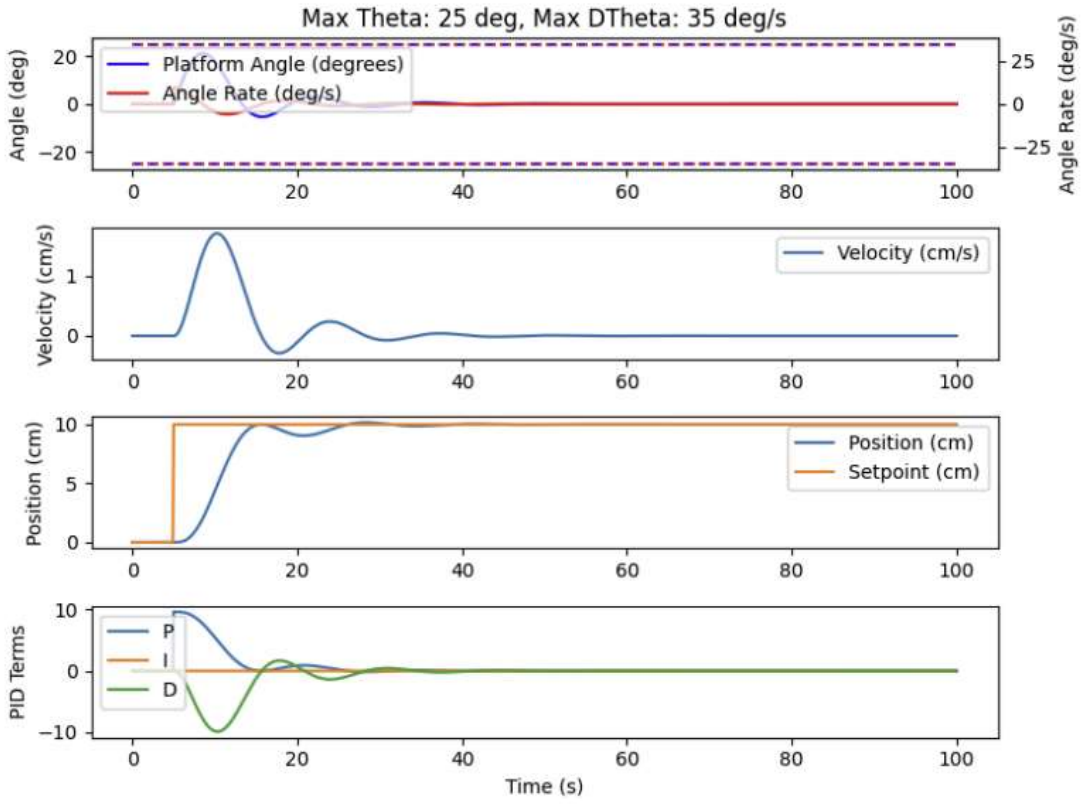
PID Optimization

```
# Process Variable Arrays
times = np.linspace(0, maxTime, nPts)
dTheta = np.zeros(len(times)) # deg/s
theta = np.zeros(len(times))  # deg
pos = np.zeros(len(times))
vel = np.zeros(len(times))
SP = np.zeros(len(times))
Ps = np.zeros(len(times))
Is = np.zeros(len(times))
Ds = np.zeros(len(times))

# Setpoint changes
SP[50:] = 10/100
#SP[100:] = 0

# Lever differential equation
def lever(state, t):
    thetaVal, dThetaVal = state
    return dThetaVal, 0

# Ball differential equation
def ball(state, t, thetaVal):
    x, dx = state
    return dx, m*g*np.sin(thetaVal) - b*dx/m
```

```
def simulate(Kc, KI, KD):
    # Configure PID controller
    pid = pc.PID(Kc, KI, KD, dt=times[1]-times[0], outputLimits=[-maxDTheta, maxDTheta
    # Reset process variables
    theta[:] = 0
    pos[:] = 0
    vel[:] = 0
    # Simulate
    for i in range(len(times)-1):
        tArr = [times[i], times[i+1]]
        # Set the setpoint and get PID output
        pid.setpoint = SP[i]
        if theta[i] > maxTheta or theta[i] < -maxTheta:
            Iterm = False
        else:
            Iterm = True
        dTheta[i], Ps[i], Is[i], Ds[i] = pid(pos[i], Iterm)
        # First, integrate for new lever position
        init = [theta[i], dTheta[i]]
        theta[i+1] = max(min(maxTheta, odeint(lever, init, tArr)[-1][0]), -maxTheta)
        # Next, integrate for ball position
        init = [pos[i], vel[i]]
        result = odeint(ball, init, tArr, args=(theta[i]/180*np.pi,))
        pos[i+1], vel[i+1] = result[-1]
    return theta, dTheta, pos, vel, SP, Ps, Is, Ds

if input("Run optimization? (y/n): ") == 'y':
    def optimize(args):
        Kc, KI, KD = args
        theta, dTheta, pos, vel, SP, Ps, Is, Ds = simulate(Kc, KI, KD)
        return np.sum(np.abs(SP - pos)) + np.sum(np.abs(theta))/500

    # Optimize parameters
    result = minimize(optimize, [60, 0.0, 487.1], method='Nelder-Mead')
    print(result)
    Kc, KI, KD = result.x

    # Get data
    theta, dTheta, pos, vel, SP, Ps, Is, Ds = simulate(Kc, KI, KD)
else:
    print("Cancelled optimization")

# Plot results
plt.figure(figsize=(8, 6))
plt.subplot(4, 1, 1)
plt.title(f"Max Theta: {maxTheta} deg, Max DTheta: {maxDTheta} deg/s")
line1, = plt.gca().plot(times, theta, label='Platform Angle (degrees)', color = 'b')
plt.plot([times[0], times[-1]], [maxTheta, maxTheta], 'b--')
plt.plot([times[0], times[-1]], [-maxTheta, -maxTheta], 'b--')
plt.ylabel('Angle (deg)')
ax = plt.gca().twinx()
line2, = ax.plot(times, dTheta, label ='Angle Rate (deg/s)', color='r')
ax.plot([times[0], times[-1]], [maxDTheta, maxDTheta], 'r:')
ax.plot([times[0], times[-1]], [-maxDTheta, -maxDTheta], 'r:')
plt.ylabel("Angle Rate (deg/s)")
lines = [line1, line2]
labels = [line.get_label() for line in lines]
```

```
ax.legend(lines, labels)
plt.subplot(4, 1, 2)
plt.plot(times, vel*100, label='Velocity (cm/s)')
plt.ylabel('Velocity (cm/s)')
plt.legend()
plt.subplot(4, 1, 3)
plt.plot(times, pos*100, label='Position (cm)')
plt.plot(times, SP*100, label='Setpoint (cm)')
plt.ylabel('Position (cm)')
plt.legend()
plt.subplot(4, 1, 4)
plt.plot(times, Ps, label='P')
plt.plot(times, Is, label='I')
plt.plot(times, Ds, label='D')
plt.ylabel('PID Terms')
plt.xlabel('Time (s)')
plt.legend()
plt.tight_layout()
plt.show();
```

```
       message: Optimization terminated successfully.
       success: True
        status: 0
           fun: 9.15327866464746
             x: [ 9.615e+01  1.015e-04  5.793e+02]
           nit: 139
          nfev: 255
 final_simplex: (array([[ 9.615e+01,  1.015e-04,  5.793e+02],
                        [ 9.615e+01,  1.015e-04,  5.793e+02],
                        [ 9.615e+01,  1.014e-04,  5.793e+02],
                        [ 9.615e+01,  1.023e-04,  5.793e+02]]), array([ 9.153e+00,  9.
153e+00,  9.153e+00,  9.153e+00]))
```



Model Description: