Rocky: The Self-Balancing Robot

Engineering Systems Analysis Final

Efe Gulcu, Natsuki Sacks, Madison Tong

March 2023

Closed-Loop Feedback System

Overview of the System

As given in Rocky instructions, the block diagram for this closed-loop feedback system is as follows.

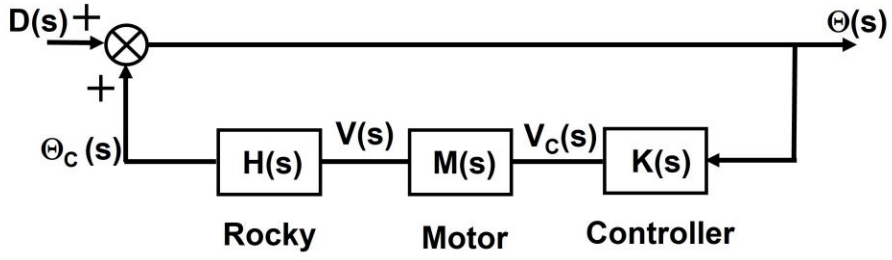

Figure 1: Block diagram of the closed-loop system that balances Rocky.

As given in the instructions, $M(s)$ and $K(s)$ are as follows.

$$M(s) = \frac{V(s)}{V_c(s)} = \frac{\frac{K}{\tau}}{s + \frac{1}{\tau}} \tag{1}$$

$$K(s) = K_p + \frac{K_i}{s} \tag{2}$$

To find our $H(s)$, we referenced the equation that we found from Day 10 of the classwork.

$$H(s) = \frac{\theta(s)}{V(s)} = \frac{\frac{-1}{l}s}{s^2 - \frac{g}{l}} \tag{3}$$

Determine K and τ

To find K and τ, we first needed to find output values of the motor model. We ran the `Rocky_Motor_Test_Calibration.ino` file to receive speeds over three seconds for both the left and right wheels.

In order to find the K value, we took the Final Value Theorem of equation 1 as follows:

$$\lim_{s\to 0} s * \frac{V_{step}}{s} * M(s) = V_{step}\frac{\frac{K}{\tau}}{\frac{1}{\tau}} = V_{step}K = 300K \tag{4}$$

After plotting our speed vs. time graph of the right wheel on MATLAB, we used the curve fitting tool to graph a line of best fit over the data. we reached a steady state value of $0.82m/s$.

Using this, we set equation 4 equal to $0.82m/s$ and found that $K = 0.00273$.

Given the information that τ is the time it takes for the system to reach approximately 63% of the steady state value, we multiplied 0.82 by 0.63, giving us 0.5166, and mapped that value to a corresponding time value on the x-axis of our graph on MATLAB to give us an approximate value of $\tau = 0.0874$. This is visualized by Figure 2 below.

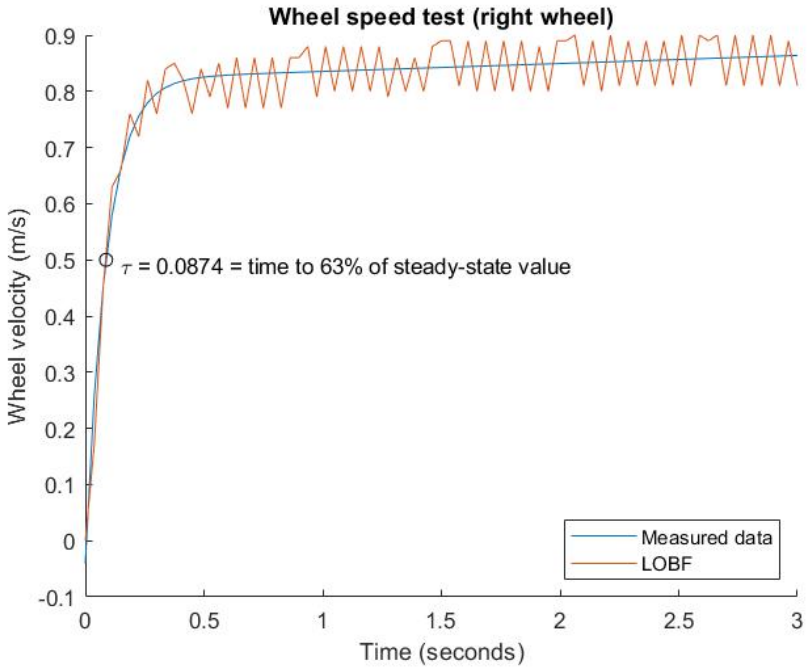

Figure 2: Visualization of how we found τ.

Measuring the Natural Frequency and Effective Length

We measured the natural frequency by letting Rocky freely oscillate as it was held from the shafts of the wheels.

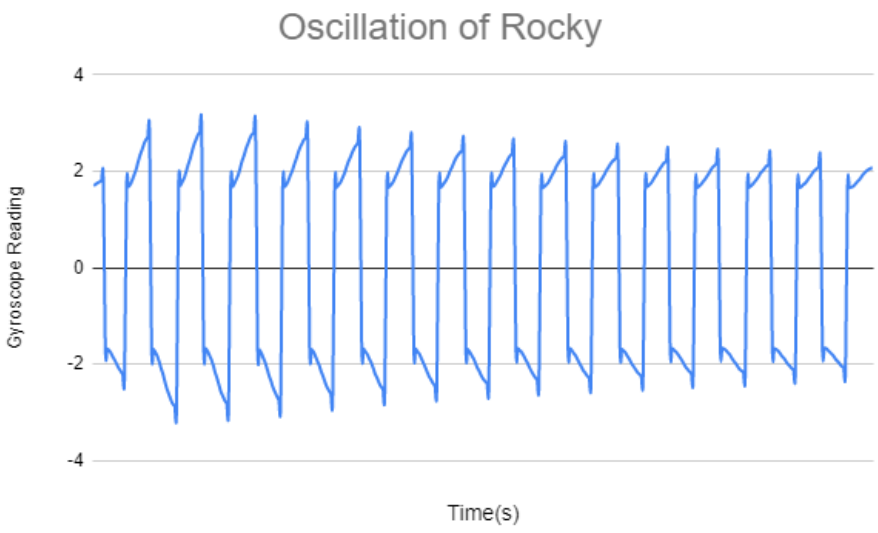

Figure 3: The experimental graph of the oscillatory behavior of Rocky.

We calculated the average time between consecutive peaks on Figure 3 and we found that the natural frequency of our Rocky is 0.738 Hz. Given that we are assuming Rocky is a system that resembles a pendulum with a mass-less rod and a point mass at the end of the rod, we calculate the effective length of Rocky with the given equation.

$$\omega_n = \sqrt{\frac{g}{l_eff}}$$

Using this equation, we found that the effective length of Rocky is 0.456 m.

Performance Specifications

Since the natural frequency W_n of Rocky is 0.738 Hz, we wanted the frequency of oscillations after a disturbance to be greater than W_n. We decided to declare this to be 2 Hz. We also said the decay rate should be around 0.7 seconds for a smoothly stabilized Rocky.

Optimizing System Performance

There are several factors which affect the behavior of the system that can be taken into account and manipulated when choosing poles: rise time, overshoot, and settling time. The following graph depicts each factor in relation to a system output graph:

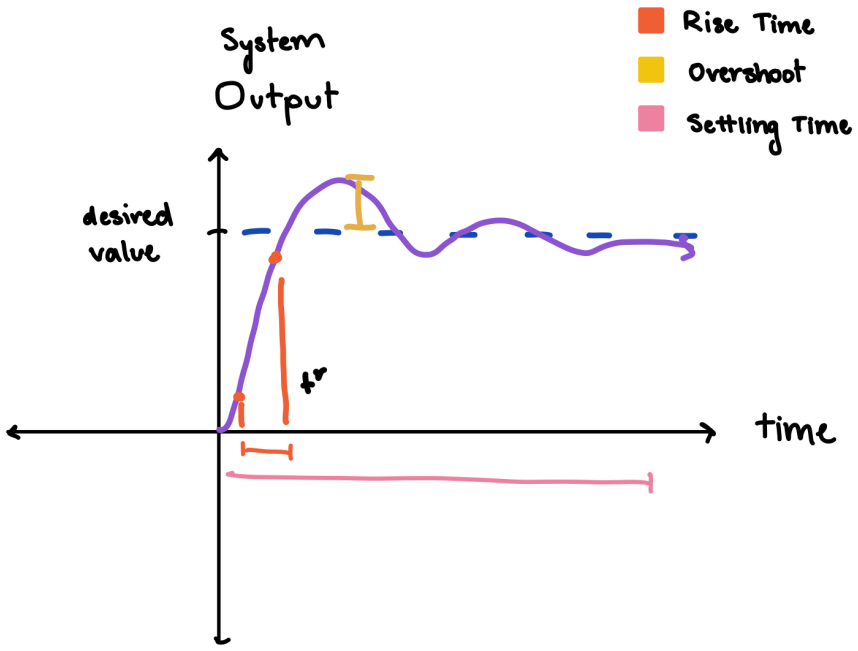

Figure 4: Depiction of a generalized System Output v.Time graph including indicators for rise time, overshoot, and settling time.

First, there is rise time which, in our case, would be considered the time it takes for the system to get from 10% to 90% of its desired angular velocity. It is known that having a faster time constant would increase rise time, and therefore optimize our system. Therefore, we want to increase the real part of our poles.

Another factor is overshoot, which is the maximum peak of the system's output above the desired value. Optimizing overshoot means decreasing the amount that the system goes over the desired value. In order to do this, we want to decrease oscillation and increase damping, which means our poles should have a smaller imaginary part.

Finally, there is settling time which is the time it takes until the system output is within 5% of the desired value. Optimizing settling time means that it should be decreased so that the system settles as fast as possible. In order to decrease settling time, we want to increase the value of the real part of our poles.

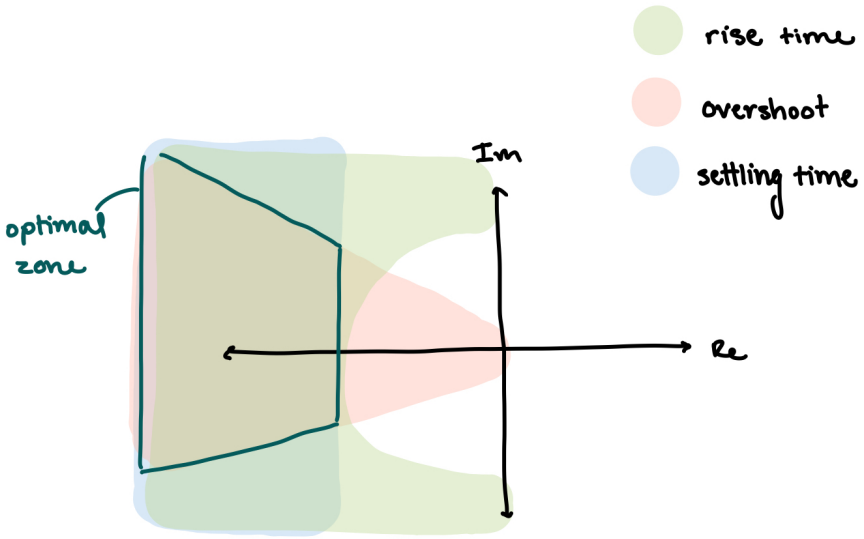

Figure 5: Optimal zones for poles according to rise time, overshoot, and settling time on the Imaginary-Real graph.

Simulating our Initial System using MATLAB's Simulink

We started off by simulating the system on Simulink with only the PI controller. The transfer function of this closed-loop system is as follows.

$$\frac{K_s(K_p + \frac{K_i}{s})}{l\tau(\frac{g}{l} - s^2)(s + \frac{1}{\tau})}$$

We used the poles $p1, p2 = -1 \pm 2i$ and $p3 = -24$ to test the Simulink. We used the MATLAB script titled `Rocky_closed_loop_poles_23.m` to calculate the following control constants:

$$Ki = 1.6581e + 04 \tag{5}$$
$$Kp = 1.4330e + 03 \tag{6}$$

The following figure is the Simulink model we used.

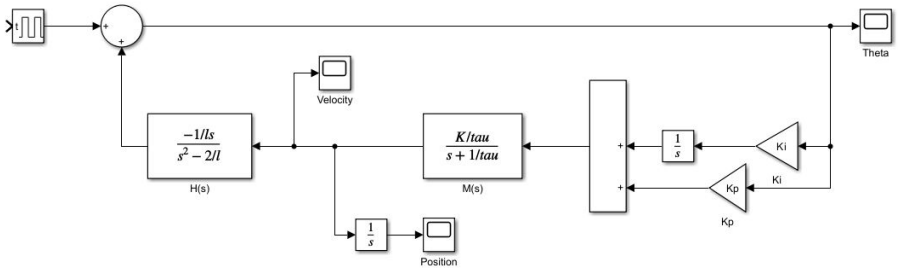

Figure 6: The nested feedback loop Simulink model with only the PI θ controller. Simulink version of the block diagram in Figure 1.

We included several scopes in the Simulink model in order to see the outputs of angle of body, velocity, and position of the system given a disturbance force as input.

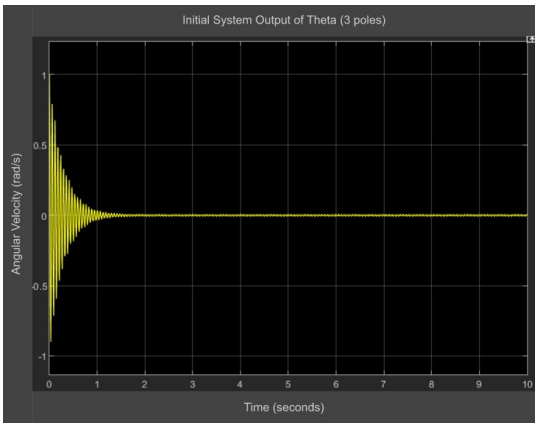

Figure 7: Theta output we get from the initial simulation

As shown in Figure 7, the θ value for our initial simulation successfully reaches a steady state of 0° per second. This is as expected, as our initial system is created only to balance Rocky's angle; it does not include the nested loops that provide feedback for the velocity and position.
Since we were not improving upon velocity and position, when testing Rocky, it would oscillate incredibly quickly and fall over almost immediately. As expected, Rocky wasn't able to effectively stabilize itself, since the wheel velocities could not receive feedback relating to their target velocity.

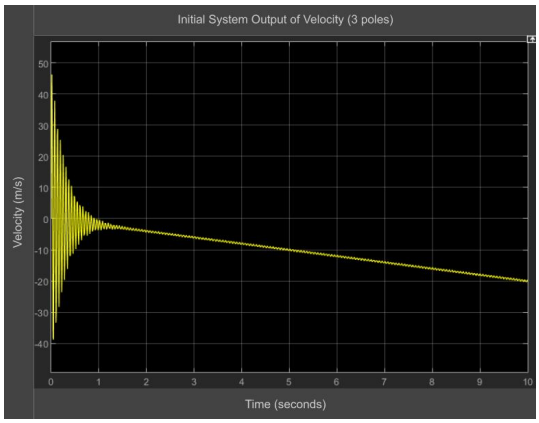

Figure 8: Velocity output we get from the initial simulation.

As observed in Figure 8, the velocity never reaches steady-state and alternatively, continues to skew further from 0, meaning that Rocky would continue to increase speed in one direction. This makes sense because there is an absence of a feedback loop, which means it never tries to balance its velocity.

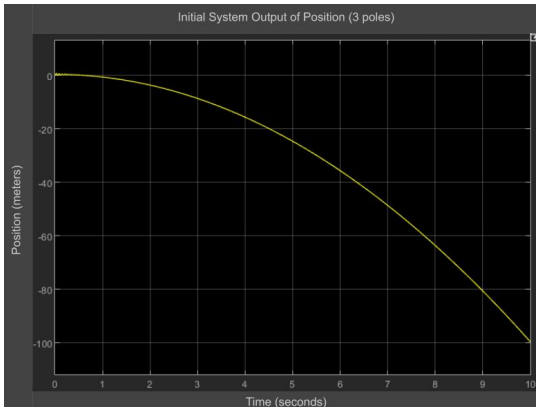

Figure 9: Position output we get from the initial simulation.

As seen in Figure 9, the position also never reaches steady-state and exponentially moves further from 0, meaning that Rocky would continue to move further away from its starting point.

Similar to velocity, the position has no feedback loop which means Rocky will never try to move back to its original position.

The behavior of the position graph is as expected, when considering that the position is the integral of velocity, and the steady-state of the velocity graph is linearly decreasing.

Stationary System

Finding the poles and constants

In order to actually balance Rocky, we needed first to calculate our poles to optimize the specifications of the system. To do this, we found the closed-loop transfer function of the entire system given in Figure 4 using Black's Formula.

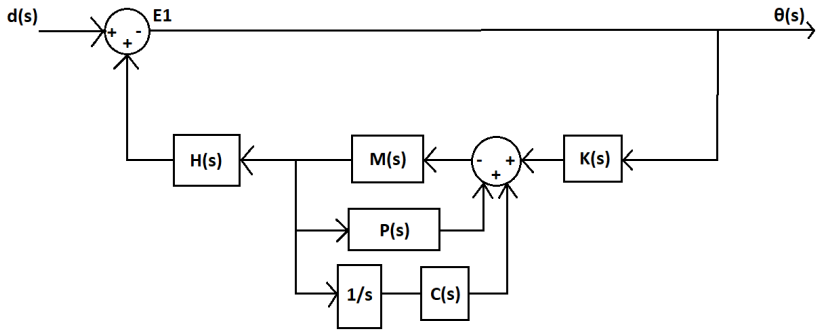

Figure 10: Block diagram of the improved closed-loop system that balances Rocky.

To find ideal poles, we utilized the given MATLAB script titled `Rocky_5_closed_loop_poles.m`. We simply altered the length l slightly and ran the script to provide us with constants Kp, Ki, Jp, Ki, Ci. To get a general sense of our system, we first calculated the constants by using the poles already defined. These were as follows.

$$p1 = -1 + 2i \tag{7}$$
$$p2 = -1 - 2i \tag{8}$$
$$p3 = -6 \tag{9}$$
$$p4 = -42 \tag{10}$$
$$p5 = -24 \tag{11}$$

When placing the generated constants into our MATLAB Simulink closed-loop feedback system, we found that the simulated output was greatly off. The system wasn't dampening fast enough and there were too many oscillations. When we ran these values on Rocky as well, we noticed that the system generally spun pretty fast, and would immediately fall to the ground.

To combat this, we increased the complex conjugates to be $p1, p2 = -8 \pm 5i$. Increasing the imaginary part would increase the reaction time, as well as overshoot consequentially. Increasing the real part would increase dampening, and therefore stabilize faster, while also counteracting the increase in overshoot from the imaginary part.

With these poles, the MATLAB script calculated the following constants.

$$Ki = 7.9110e + 05 \tag{12}$$
$$Kp = 1.6506e + 05 \tag{13}$$
$$Ji = -2.9060e + 05 \tag{14}$$
$$Jp = 2.1143e + 03 \tag{15}$$
$$Ci = -7.0547e + 05 \tag{16}$$

Originally, some of these constants were complex and therefore consisted of an imaginary portion. The imaginary coefficients were so small that they were negligible.

Simulating our stationary balancing system using MATLAB's Simulink

Before we started testing our values with Rocky, we used Simulink to simulate the system for us. This saved us a lot of trial-and-error testing time.

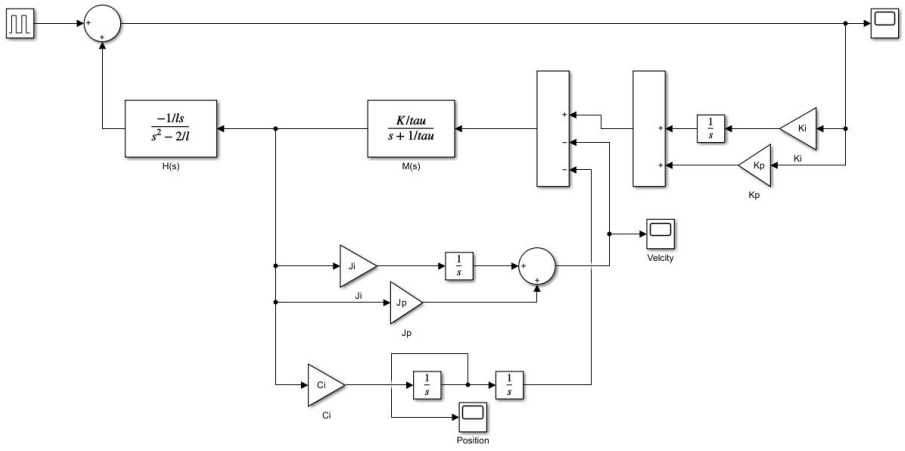

Figure 11: The nested feedback loop Simulink model. Three scopes are included to view the output of each nested loop.

The transfer functions of $H(s)$ and $M(s)$ are displayed in the model. The large feedback loop (without the nested loops) controls the angle. The first nested loop (constants J_p and J_i) control the wheel speeds. The second nest loop controls the position of Rocky to ensure that it returns to it's origin.

V_c is the measurement between the PI controller (using K_p and K_i) and M_s, as labeled on Figure 4. This tells us the desired velocity of the left and right wheels at a given angle. The general equation for this is as follows.

$$V_{c_l} = (K_p + \frac{K_i}{s}\theta - [(J_p + \frac{J_i}{s})v + \frac{C_i}{s^2}v] \tag{17}$$

$$V_{c_l} = (K_p + \frac{K_i}{s}\theta - (J_p + \frac{J_i}{s} + \frac{C_i}{s^2})v \tag{18}$$

$$V_{c_l} = K_p\theta + K_i \int \theta - (J_p v + J_i x + C_i \int x) \tag{19}$$

Equation 19 is the final equation we used. To convert from the s-domain to the time-domain, we utilized the fact that $\frac{1}{s}$ is an integrator. For example, $v * \frac{1}{s}$ is position x.
When putting our final generated values into our Simulink model, we received the following output.

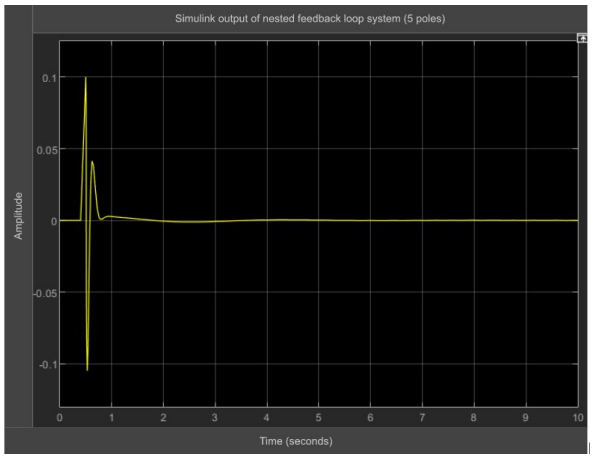

Figure 12: Output of the largest nested loop controlling the angle.

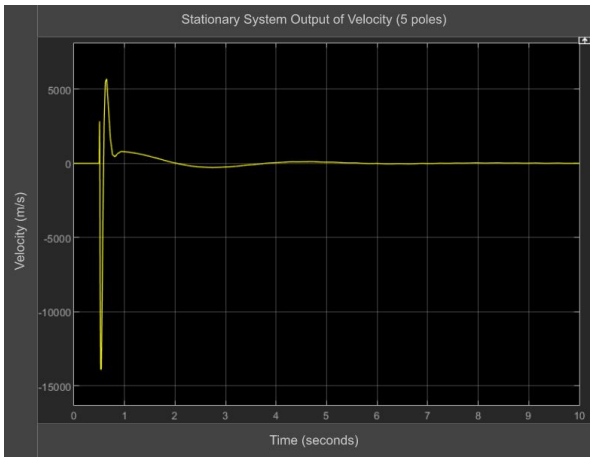

Figure 13: Output of Simulink for the $J(s)$ feedback loop controlling the wheel velocities.

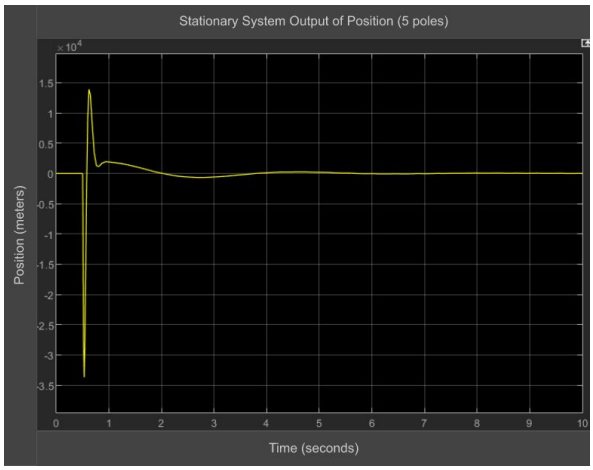

Figure 14: Output of Simulink for the $C(s)$ feedback loop controlling the position of Rocky.

Figures 12 through 14 look great, as they all reach a steady-state of zero. There is minimal oscillation and the system stabilizes quickly.

To actually run Rocky, we adjusted the Arduino code titled `Rocky_Balance_Starter_Code_23.ino` with the proper constants and velocity equations. We found that switching the distance left/right variables (i.e. $distLeft$ with V_{cR}) made Rocky work perfectly, while the opposite failed quickly.

IT WORKS!

Here is the result!