Robolympics: Control System for Inverted Pendulum Robot

Quinn Kelley and Mark Goldwater

November 8, 2018

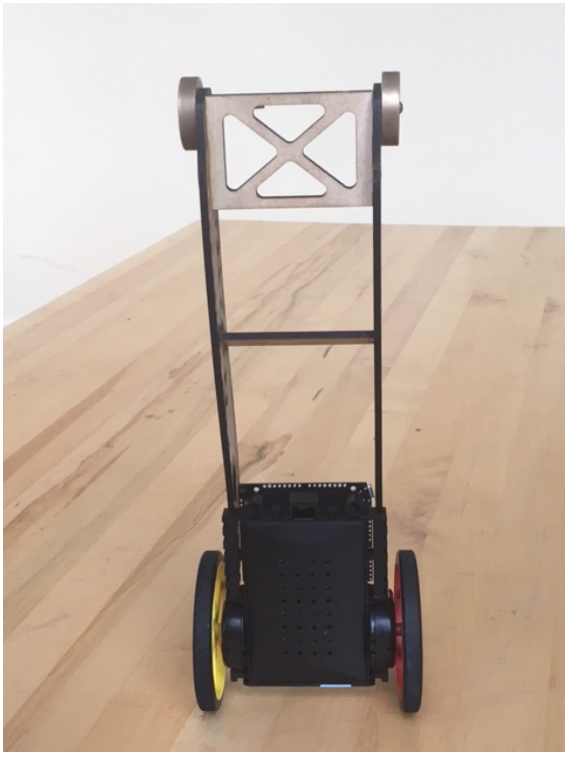

Figure 1: This figure features Rocky the inverted pendulum balancing upright.

1 Athlete Demographics: System Characterization

In this project we implemented a control system on the inverted pendulum robot featured in Figure 1. The following is analysis done on the system that we used to get the robot to stand up straight, as well as to translate it. Figure 1 shows the provided robot. The system was abstracted to an inverted pendulum model as shown in Figure 2, with l being the length from the wheelbase to the robot's center of mass m, and θ being the angle from the vertical.

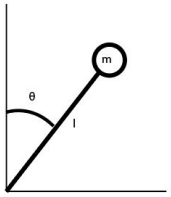

Figure 2: The inverted pendulum model used to represent the robot.

Assuming that the pendulum has a massless rod, and that the robot's wheels have a linear velocity v, the governing ordinary differential equation of the system is:

$$ml^2\frac{d^2\theta}{dt^2} - mgl\theta = -ml\frac{dv}{dt} \tag{1}$$

This governing equation was used to find the transfer function with the velocity of the wheels as an input and the resulting angle of the inverted pendulum as the output. This transfer function is as follows:

$$\frac{\theta(s)}{V(s)} = \frac{-s}{ls^2 - g} \tag{2}$$

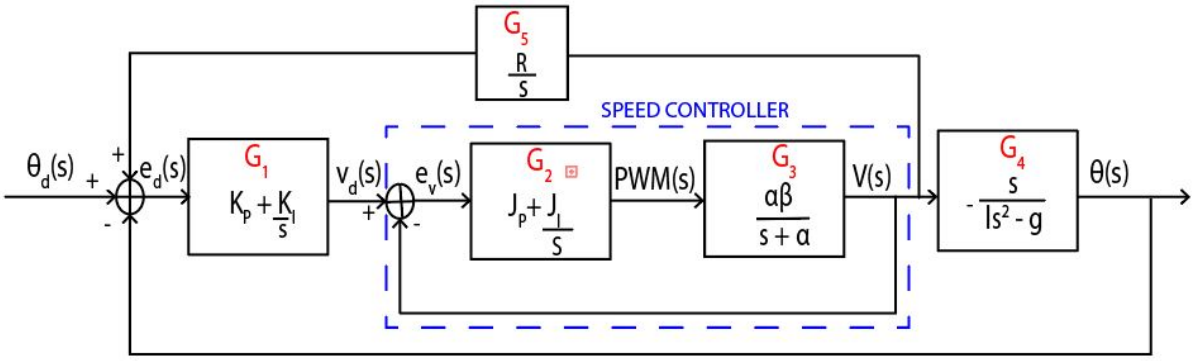

Figure 3: Block diagram for the standing inverted pendulum.

Figure 3, contains the block diagram that we used to control our pendulum system to stand upright. The first error block in the diagram calculates the main error of our system. It takes a desired angle (θ_d), subtracts the actual angle (θ) of the system, and adds an integrated velocity multiplied by a tuned constant **R** in order to take displacement of the robot into account. This way, the displacement needs to be zero in order for this first error function to decay to zero. The error calculation resulting from this block then feeds into an angle PI controller (G1).

A PI controller has two constants, a proportional constant and an integral constant (typically referred to as K_P and K_I), that are used to operate on the error and its integral over time in order

to correct the current state of the system. In the time domain this looks like the following equation:

$$K_P(error) + K_I \int_0^t error \tag{3}$$

For each controlled variable, the higher the constants are the more that respective term is taken into account for the correction of the system's response.

The result of the angle controller (desired velocity) then feeds into a control loop for the motors (G2-G3). This nested loop consists of an error block which takes V_d and subtracts from it an actual velocity. This error feeds into the motor PI controller which controls the system to the desired velocity by adjusting the PWM signal to minimize the velocity error block. This PWM signal feeds into the motor block which converts it to an output velocity. Lastly, this output velocity is taken in by the block that represents the mechanical system of the inverted pendulum (G4). This block takes this velocity and outputs a resulting angle which, as previously mentioned, is fed back into the first error function of the system.

After establishing that this was the block diagram we wanted to use to control our system, we began our calculations for the transfer function of the entire system.

We started out by establishing transfer functions across each of the individual blocks in our system.

$$G_1(s) = K_p + \frac{K_I}{s} \tag{4}$$

$$G_2(s) = J_p + \frac{J_I}{s} \tag{5}$$

$$G_3(s) = \frac{\alpha\beta}{s+\alpha} \tag{6}$$

$$G_4(s) = -\frac{s}{ls^2 - g} \tag{7}$$

$$G_5(s) = \frac{R}{s} \tag{8}$$

G_1, and G_2 are standard PI controller blocks. In our system, G_1 is a PI controller that guides the system in achieving our desired angle (0 degrees) and G_2 is a PI controller that guides the system in achieving a desired velocity.

The transfer function for the motor, G_3, was derived from a first order differential equation model of the motor reaching a steady state. The transfer function G_4 was derived from the governing equation discussed at the beginning of this section. Lastly, the transfer function G_5 is an integrator block that calculates displacement based on the robot's velocity over time.

From here we defined larger groups of blocks in our system in terms of the basic transfer functions above.

$$\theta(s) = e_v(s)G_4(s)G_3(s)G_2(s) \tag{9}$$

$$V(s) = e_v(s)G_3(s)G_2(s) \tag{10}$$

$$e_v(s) = e_\theta(s)G_1(s) \tag{11}$$

$$e_\theta(s) = \theta_d(s) - \theta(s) + V(s)G_5(s) \tag{12}$$

Next, to calculate the transfer function of the entire system, we manipulated equations 9-12 using Wofram Mathematica.

$$\frac{\theta(s)}{\theta_d(s)} = \frac{G_1 G_3 G_2 G_4}{-G_1 G_3 G_2 G_5 + G_1 G_3 G_2 G_4 + 1} \tag{13}$$

The poles of the transfer function can be used to characterize the response of the system. These are found by determining the roots of the polynomial in the denominator of the transfer function. Figure 3 shows the poles graphed on the complex plane.

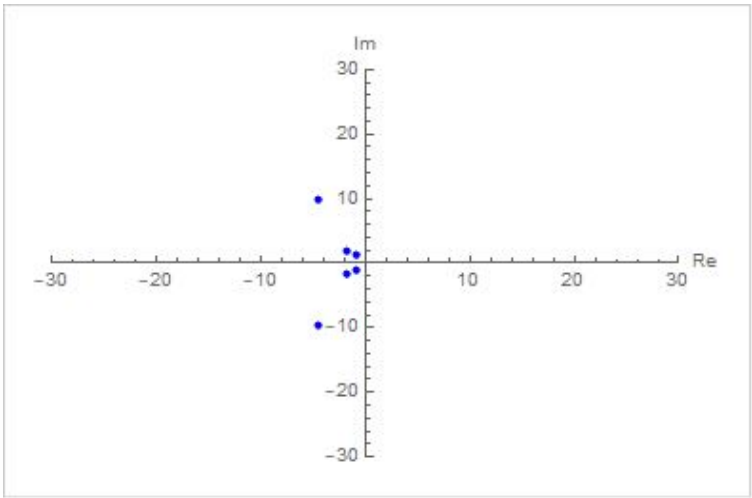

Figure 4: This graph shows the positioning for the poles of the parameters we chose to make our system balance (discussed in the next section) on the complex plane.

2 Athlete Performance Information: Tuning Control Parameters

In order to control our robot to make it stand up straight, we use the following parameters: K_p = -78.6, K_i = -62.24, $J_p = 10$, $J_i = 65$, R = -0.1. As mentioned in the Athlete Demographics Section, each of these constants determines how much a given proportional or integral term of a controller influences the correction of the system. The proportional controller corrects based off of the current value of the error, and the integral term corrects based off of the accumulated error over the course of the time that the system has been running. If just a P term is used in a controller, there is some error in the desired steady state value. The integral term helps in eliminating this error.

We characterized the response of the motor block (determining the associated parameter values) experimentally by inputting a PWM into the system, and fitting a curve of the output velocity as it reached steady state. The parameters we found for the motor block were $\alpha = 14$ and $\beta = 1/400$. The parameter $L = 1/10$ was found by approximating the center of mass of the robot. We shifted the position of the center of mass upwards by adding two 50g weights to the top of the robot, thus lengthening the moment arm of the pendulum, in order to slow down the rate at which the system responded. The reasoning as to why this works can be found in the solution to the governing equation of the system below

$$\theta(t) = \theta(0)\left(\frac{1}{2}e^{\sqrt{-\frac{g}{l}}t} + \frac{1}{2}e^{\sqrt{\frac{g}{l}}t}\right) \tag{14}$$

As the value of l increases, the exponential terms in this solution would decay more slowly which means that by moving up the center of mass, we can slow down Rocky's reaction time to a point where we can easily manipulate the robot with the motors. This is necessary because if the motors need to react faster than they are able to in order to maintain a stable system, then balancing Rocky would not be possible.

Poles of the transfer function concentrated towards the left side of the complex plane stabilize the system because it makes sure all the exponential terms are raised to negative powers creating a function that, when disturbed by an impulse will converge to a stable point (a pendulum angle of zero degrees in our system).We found the unknown angle controller parameters (K_p and K_i) by minimizing the maximum real value of the poles. This was achieved using an objective function that was defined in terms of K_p and K_i, with previously tuned motor controller parameters and all system constants predetermined. Resulting from this minimization were ideal values of K_p and K_i.

Another consideration that had to be made was ensuring that the input velocity did not exceed the speed that the motors were able to achieve (about 0.75 m/s). Once we found poles that we thought would result in the desired system behavior, we plotted the expected response for a small angular disturbance to the system, and confirmed that the robot's resulting velocity stayed within the possible physical range. The predicted velocity plot for our parameters is shown in figure 4.

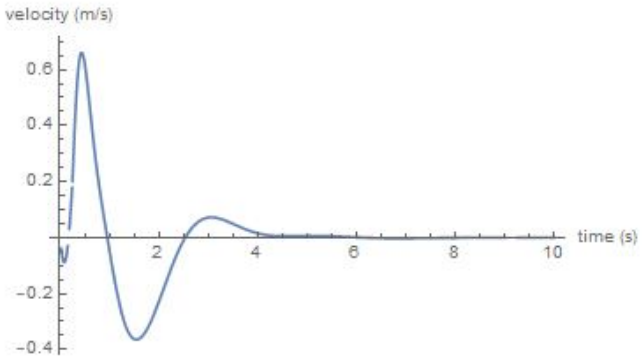

Figure 5: This graph shows the anticipated linear velocity of our pendulum following a small disturbance. Note that the velocity stays below the system limit of 0.75 m/s

3 Training Session Description

3.1 Implementation of the Inverted Pendulum Control

At the end of day 10 in class, we were at the point where the robot would remain stable at vertical for a short period of time as it moved forward at an increasing speed until it collapsed onto the floor. In our implementation of Rocky we have extended our analysis by removing its "runaway" tendencies. This feat was accomplished mainly with the added feature of the integration block. This factored the integrated velocity (displacement) into the first error function. Because our system wants to drive this error to zero in order to reach its stable state, the drift in our robot went away as a result of the addition of this block. Moreover, we found that adding a velocity feedback into the leftmost error block of our diagram (with a proportional constant of -0.05) slightly reduced the

drift of the robot even further. This is because now that the velocity of each wheel is taken into consideration by the error function our system wants to drive that to zero as well.

In terms of the implementation of our system in Arduino code (see Section 5) we followed the calculations prescribed by our block diagram in order to derive PWM values for each wheel whenever our main update function was called.

3.2 Translating the Inverted Pendulum

In order to send the robot forwards at a fairly consistent velocity, we subtracted a constant from the integrated displacement. This constant was increased at an experimentally determined rate, producing a smooth forward movement, as a result of our ever-changing desired displacement.

One issue we ran into was that the pendulum would arc to one side or another as it traveled. This was because our robot had two different types of wheels, likely with distinctly different motor characteristics. We resolved this issue by scaling the PWM values for each wheel independently, so that their resulting velocities would be roughly the same, demonstrated by a roughly straight path. This had the unintended benefit of removing the effect of the "dead zone": the behavior arising when the velocity that needs to be given to the wheels is too low to overcome the friction in the gearbox.

4 Video

Footage of the inverted pendulum implementation can be found here.
Footage of the translating inverted pendulum can be found here.

5 Source Code

The source code can be found on GitHub here.