Coach's Notes: Bullwinkle

Kyle "Micky" Emmi, Jonathan "Goldmill" Zerez

November 2018

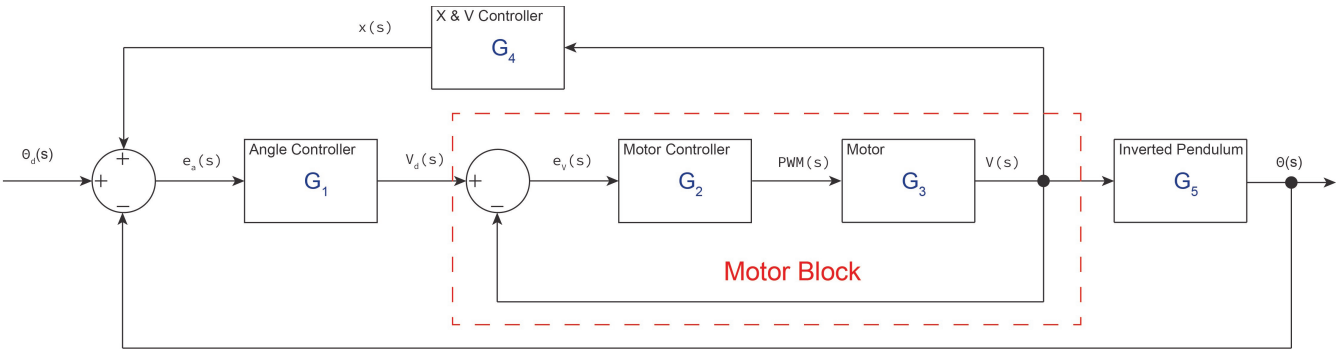

Figure 1: Block Diagram for Survivor

1 Introduction

Bullwinkle is the lesser known sidekick and counterpart to the world renown boxer, Rocky. He hails from a small Indiana farm. Being the youngest of 5 older brothers, he always had to fight tooth and nail to get his fair share of the food come supper time. He's ready to bring home the gold for his folks back home in the sprint competition of the Rocky Olympics. Here is his best sprint, and here is his best attempt at the survivor event.

2 Athlete Demographics

2.1 Individual Transfer Functions

Our block diagram for the survivor event has five different components in it. It has three controllers, an angle controller, a motor controller, and a position/velocity controller, and two plants, the motor and the rocky itself. Each subsection below will detail the transfer functions of each component of the overall system.

2.1.1 Transfer Function for the angle controller

The angle controller is a proportional-integral controller that takes an angle as an input, and outputs a desired velocity in meters per second. The transfer function for this block is represented by G_1. Mathematically, G_1 is:

$$G_1(s) = K_p + \frac{K_i}{s} \tag{1}$$

Where K_p and K_i are user tuned constants that impact the system's performance. The incoming error in angle is multiplied by the K_p term, and the integral of angle error over time is multiplied by the K_i term. The form of this transfer function is standard for a simple proportional-integral controller in S-space. This controller is the supposed "main" controller for the system, and ensures that the angle of the robot stays as close to vertical as possible.

2.1.2 Transfer Function for the motor controller

The motor controller is a proportional-integral controller that takes the difference between the desired velocity and actual velocity as an input, and outputs a PWM signal. The transfer function for this block is represented by G_2. Mathematically, G_2 is:

$$G_2(s) = J_p + \frac{J_i}{s} \tag{2}$$

Where J_p and J_i are user tuned constants that impact the system's performance. The incoming error in velocity is multiplied by the J_p term, and the integral of velocity error over time is multiplied by the J_i term. The form of this transfer function is standard for a simple proportional-integral controller in S-space. This controller is a more secondary controller in the system that ensures that the velocity of the robot is as close to what the output of G_1 dictates it to be.

2.1.3 Transfer Function for the motor plant

The motor plant is a model of how the velocity of the motor varies as a function of input PWM. It takes PWM

as an input, and outputs velocity. The transfer function for this block is represented by G_3. Mathematically, G_3 is:

$$G_3(s) = \frac{\alpha\beta}{s+\alpha} \tag{3}$$

Where α and β are experimentally determined constants based on the motor's performance. To understand how we got to this function, it will be useful to see its representation in the time domain. If we take the inverse Laplace transform of this transfer function, we find that its time domain representation is:

$$g_3(t) = \alpha\beta e^{-\alpha t} \tag{4}$$

When testing the motors, we found that the velocity response to a step in PWM was this in this form, that is, an exponential that asymptotically approaches a steady state condition. By testing the step response of the motor and measuring velocity, we were able to define α and β, which are related to the time constant of the response, and the steady state velocity. G_3 gives us more accuracy when predicting how the robot will respond to PWM commands, and thus increases stability.

2.1.4 Transfer Function for the position/velocity controller

The position and velocity controller is a proportional-integral controller that takes velocity and position of the robot as an input. The transfer function for this block is represented by G_4. Mathematically, G_4 is:

$$G_4 = D_p + \frac{D_i}{s} \tag{5}$$

Where D_p and D_i are user tuned constants that impact the system's performance. The incoming error in velocity is multiplied by the D_p term, and the integral of velocity error over time (ie displacement) is multiplied by the D_i term. The form of this transfer function is standard for a simple proportional-integral controller in S-space. This controller is a more secondary controller in the system that ensures that the overall velocity and displacement of the robot

2.1.5 Transfer Function for the position/velocity controller

The inverted pendulum plant is a simplified model of a inverted pendulum with a translating pivot point. It takes velocity as an input, and outputs an angle.The transfer function for this block is represented by G_5. Mathematically, G_5 is:

$$G_5 = \frac{s}{g - ls^2} \tag{6}$$

Where l is the heigh of center of mass of the robot, and g is the gravitational constant. This simplified model is based on the original equation of motion for a translating inverted pendulum:

$$ml^2\theta''(t) - mgl\sin\theta = -mlv'(t)\cos\theta \tag{7}$$

Where m is the mass of the entire system. To simplify the model, we used the small angle approximation where $\sin\theta = \theta$ and $\cos\theta = 1$. These approximations are valid to make because the robot will always balancing at small angles around vertical. They simplify the equation to be:

$$ml^2\theta''(t) - mgl\theta = -mlv'(t) \tag{8}$$

Which can be taken into the S-domain with the help of Mathematica

2.2 The Overall Transfer Function

Putting all of these equations together gives us Bullwinkle's overall transfer function. This function is just a few simple algebra operations away and begins with Equation 9 below. This is the definition of the error in angle and is the result of the first circle in the Block Diagram. The overall transfer function will have $\theta(s)$ over θ_d so we need to replace $e_a(s)$ and $x(s)$. To do this we use Equations 10 and 11. Solving Equation 10 for $e_a(s)$ and substituting that into Equations9 and 11 leaves us with $x(s)$ as the only remaining unwanted variable and is easily removed with Equation 11. This leaves us with only $\theta(s)$ and θ_d thus solving for $\frac{\theta(s)}{\theta_d}$ gives us our final transfer function in Equation 12.

$$e_a(s) = \theta_d + x(s) - \theta(s) \tag{9}$$

$$\theta(s) = e_a(s) * G_1 * \frac{G_2G_3}{1 + G_2G_3} * G_5 \tag{10}$$

$$x(s) = e_a(s) * G_1 * \frac{G_2G_3}{1 + G_2G_3} * G_4 \tag{11}$$

$$H(s) = \frac{\theta(s)}{\theta_d} = \frac{1}{\frac{1+G_2G_3}{G_1G_2G_3G_5} - \frac{G_4}{G_5} + 1} \tag{12}$$

By plugging in the individual transfer functions for G_1, G_2, G_3, G_4, and G_5 as detailed earlier into Equation 12, we were able to produce a theoretical model of how the system would behave. Placing this system into Mathematica, we found for the poles of the final transfer function by solving for the zeros of the denominator (in terms of s). Evaluating poles is useful as they allow one to quickly determine the end behavior of a system. Poles with negative real values will tend towards stability, while poles with positive real values will tend towards instability. Poles with zero imaginary components will not oscillate, and poles with non-zero imaginary components correspond to oscillatory behavior. By looking at a pole plot, we were able to quickly evaluate the theoretical behavior of our system without having to physically test anything. By choosing controller constants Kp Ki, Jp, Ji, Dp, and Di that corresponded to poles with exclusively negative real components, and relatively small imaginary components, we were able to start testing with a system that

performed well enough to qualitatively tune to perfection based on physical testing. Figure 2 below is the pole plot for the entire system using our final controller constants.

3 Athlete Performance Information

Below is a table of the different parameters that we used in the survivor event.

Parameter	Value	Units
K_p	-2.7	$\frac{m/s}{rad}$
K_i	-14	$\frac{m/s}{rad*t}$
J_p	500	$\frac{PWM}{m/s}$
J_i	5000	$\frac{PWM}{m}$
D_p	-0.1	$\frac{rad}{m/s}$
D_i	-0.15	$\frac{rad}{m}$
l	0.09	m
g	9.8	m/s^2
α	10	$1/s$
β	$\frac{1}{400}$	m
m	0.47	kg

Physical constants (l, α, β, m) were determined experimentally. Specifically, l and m were determined by simply measuring the mass of the robot, as well as the height of the center of mass. α and β were determined by examining the step response of the motors on Rocky to a PWM signal, and fitting an exponential curve to the response using MATLAB's Cftool. As for the theoretical control constants, they were determined largely by looking at the Pole Plot generated in Mathematica. Using Mathematica's "Manipulate" function, we were able to dynamically test differing values for control constants that would produce poles with exclusively negative components. We set the range for values to play with based on the units of each control constant, and the physical constraints of our system (ex: max motor velocity). Once we found a combination of control constants, we would test it on the physical system. Once we got a controller that worked relatively well, we tended to qualitatively tune the constants based on observations of performance.

4 Training Session Description

After developing/discovering the methodology for tuning control constants as described above, progress was made very quickly and efficiently. That process allowed us to be confident that control constants would not only give us a stable system, but also would respect the physical constraints of our system before spending time on testing.

In terms of developing the architecture of the entire system, we followed a process of incremental development. We first started with the simplest controller–just an angle controller, and developed it until it performed reasonably well. Then, based on observations and suggestions, we would add new controllers and increase the complexity of the entire control system. After the angle controller was complete, we integrated the transfer function of the motor plant, and then the motor controller. These additions were due to observations that the motors were unable to achieve requested velocities. After those blocks were added to the overall transfer function, we added a position controller to try to ensure that the robot would try to return to the origin, because the robot had a tendency to drift away. Finally, we added a velocity controller in addition to the position controller to ensure that the velocity of the robot at the origin was as close to zero as possible. We added this because the velocity of the robot at the origin would be non-zero, and cause increasing oscillations about the origin.

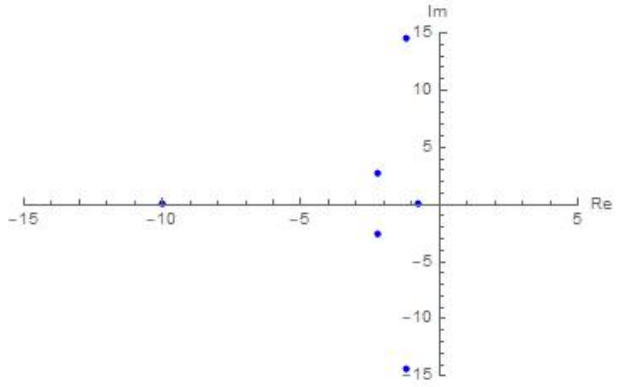

Figure 2: Pole diagram of Survivor control scheme