Inverted Pendulum Project

Andrew Mascillaro, Nolan Flynn

Base Model

The base model for Rocky is an inverted pendulum. It is approximated to a long rod of an effective length l_{eff}. Also, it has two wheels and an axle at the bottom, which are connected to a motor. The motor is controlled by a control system, which takes in the angle (displacement) of the system and outputs a rotational force on the axle, preferably one that helps the inverted pendulum stabilize itself.

We have two main measures for determining the success of our control system.

1. Rocky should not fall if given a disturbance of 30° or less. This disturbance would have a magnitude of $< 30^\circ$ degrees and last about 0.1 seconds. We believe this is an achievable goal and should protect Rocky from most common disturbances in the environment.
2. Rocky should stabilize to an oscillation no greater than 1° 2 seconds after a disturbance. This would prove that Rocky can effectively reach a steady state in a short period of time.

Parameter Specifications

Gyroscope Test

A gyroscope test was conducted to determine some physical characteristics of the rocky system. The "Rocky" was hung from its top and swung as a (non-inverted) pendulum to observe its oscillation frequency and effective length.

Natural Frequency

A standard pendulum oscillates with a natural frequency of $\sqrt{\frac{g}{l}}$, where g is acceleration due to gravity and l is the length of the pendulum. Our team determined the period of oscillation of Rocky as if it were a standard pendulum by observing how much time elapsed between each oscillation, represented by the elapsed time from the local maxima on a graph of angle versus time.

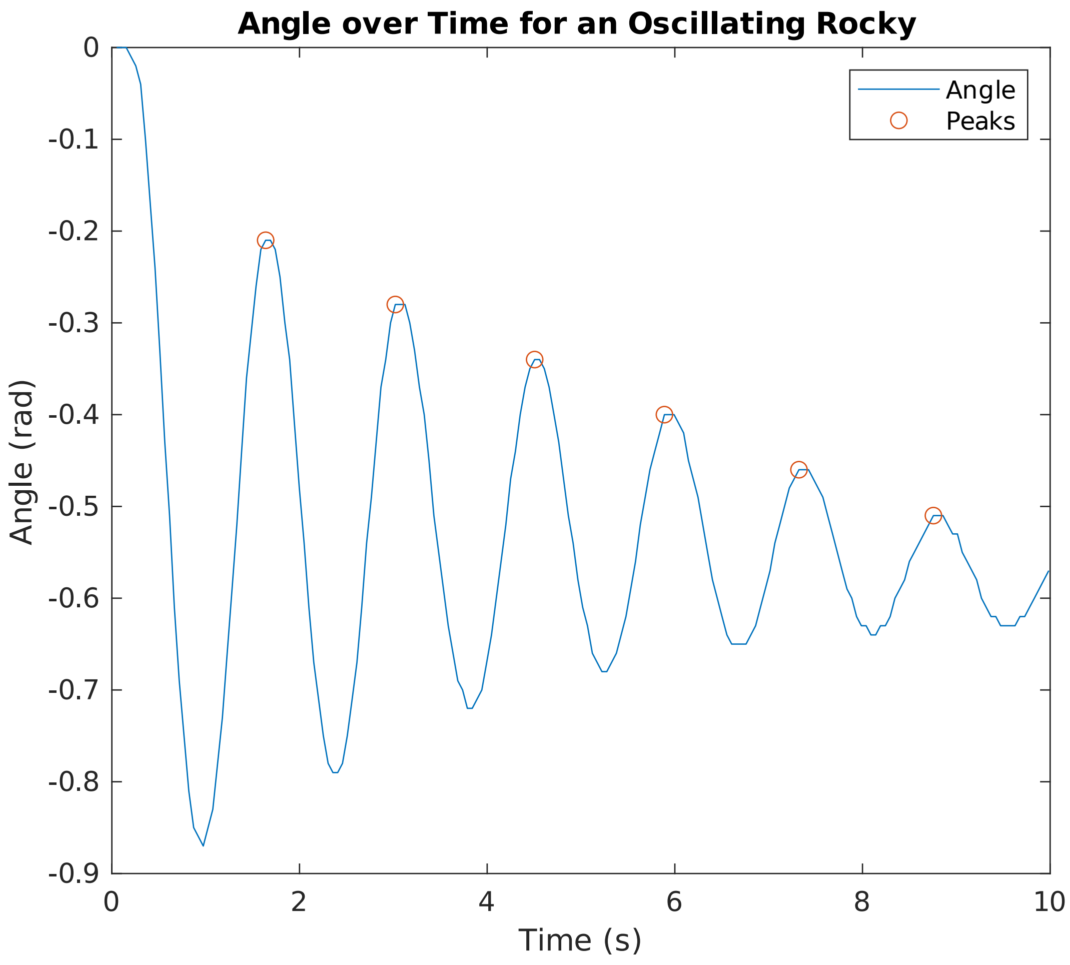

Figure 1: Angle over Time for an Oscillating Rocky

The value of ω_n, or natural frequency, is the mean of the elapsed times between each peak, or $4.4142\ \mathrm{rad/s}$

This is the natural frequency at which Rocky oscillates as a normal pendulum.

Effective Length

Now that we have the natural frequency of the pendulum, the length can be calculated using $\omega_n = \sqrt{\frac{g}{l_{eff}}}$. Since we know $\omega_n = 4.4142\ \mathrm{rad/s}$, simplifying the equation reveals that $l_{eff} = 0.5035\ \mathrm{m}$, which is almost 20"

Motor Test

The motor test is designed to determine the characteristics of the motor and inverted pendulum system. This specific test is a step response for Rocky comparing an input motor signal to the output velocity. The characteristics of this motion help determine the motor gain K and time constant τ.

Motor Gain

The motor's gain is the ratio of an input motor signal to an output velocity response. In this test, the input signal has a magnitude of 200 and we must solve for the steady state response.

The input signal is constant, and similar to charging a capacitor, the velocity over time exponentially approaches the steady state response as shown: $v(t) = -200K(1 - e^{-t/\tau})$. Note that since K is the ratio of the output and input signals, the input signal multiplied by K is the magnitude of the output signal, or velocity. An exponential fit in this form determines the velocity as a function of time to be about $-200 \cdot (0.0017) \cdot (1 - e^{-\frac{t}{0.0611}})$, shown in Figure 2.

This means that the magnitude of the gain, K is about 0.0017.

Time Constant

Using the same data as above, we can determine how fast the inverted pendulum approaches the steady state by extracting τ from the line of best fit. In this case, $\tau = 0.0611\,\mathrm{s}$.

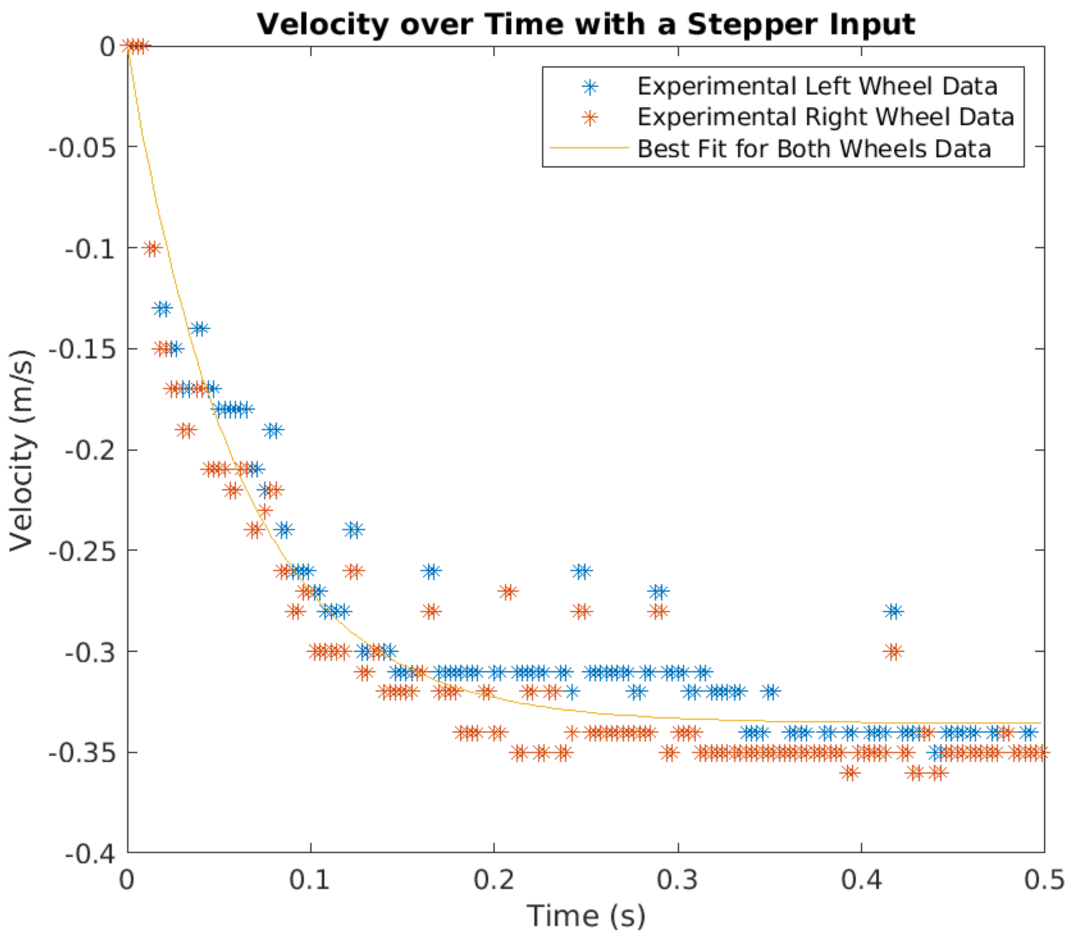

Figure 2: Velocity over Time with a Stepper Input

PI Control

To control the feedback loop, we implemented a proportional integral controller. To find the two gain values, we first chose the desired poles of our system. The four chosen poles were $-1 \pm \omega_n i$ and two poles at $-\omega_n$. The first two poles were chosen to quickly damp oscillations at the robot's natural frequency, which is a frequency likely to be present. The second poles were chosen to have the robot damp all oscillations more quickly while letting the first two dominate the response. To calculate the gain values, the system's transfer function was calculated with K_p and K_i represented symbolically, the coefficients of the transfer function's denominator are found using the poles we chose, and this equation is solved to calculate K_p and K_i. The resulting values are $K_p = 4383$ and $K_i = 13155$.

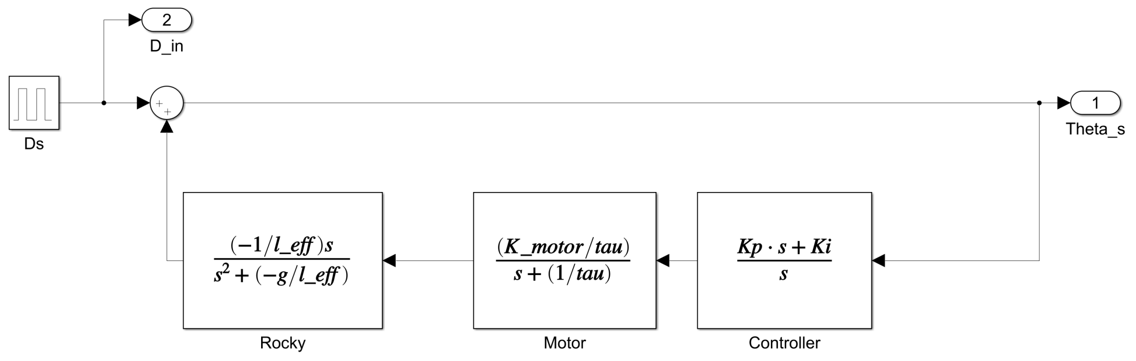
2
D_in
Ds
1
Theta_s
$\frac{(-1/l\_eff)s}{s^2+(-g/l\_eff)}$
Rocky
$\frac{(K\_motor/tau)}{s+(1/tau)}$
Motor
$\frac{Kp \cdot s + Ki}{s}$
Controller

Figure 3: Base System Diagram

Enhanced Model

For our enhanced model, we decided to control the rotation of the robot by controlling its desired angle. We wanted the robot to oscillate back and forth without losing its ability to remain upright. For our oscillations, we chose a formula of $\theta_d = 0.1\sin(2\pi \cdot t)\sin(20 \cdot 2\pi \cdot t)$, although any reasonably sized input could be used. To ensure the robot would not lose any stability, we implemented a switch to set the desired angle to zero if the measured angle of the robot was larger than $0.25\,\mathrm{rad}$. This allows the robot to only do the desired oscillations when it's approximately upright.

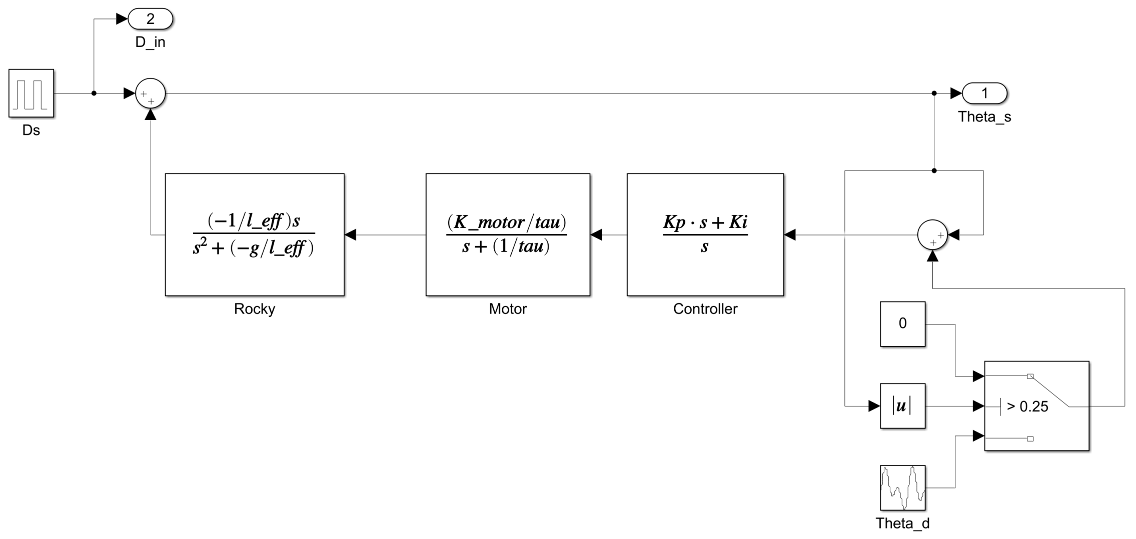
2
D_in
Ds
1
Theta_s
$\frac{(-1/l\_eff)s}{s^2+(-g/l\_eff)}$
Rocky
$\frac{(K\_motor/tau)}{s+(1/tau)}$
Motor
$\frac{Kp \cdot s + Ki}{s}$
Controller
0
|u|
> 0.25
Theta_d

Figure 4: Enhanced System Diagram

Code Appendix

Main File

This file calls other methods for parameter calculation

```
clc
close all
clear all
format compact

[wn, l_eff] = gyroscope_parameters();
[K_motor, tau] = motor_parameters();

[Kp, Ki] = control_parameters(K_motor, tau, wn, l_eff)
```

Gyroscope Test

This file calculates ω_n and l_{eff} for the system.

```
function [wn, l_eff] = gyroscope_parameters(varargin)

% GYROSCOPE TEST!!!
load("mats/gyrotest.mat");

% find exponential points
[pks,locs] = findpeaks(angle_rad, t);

% get natural frequency
% as difference between peaks
mean_period = mean(diff(locs));
wn = 2*pi/mean_period;

% length as a function of frequency
l_eff = 9.81/(wn^2);

if nargin == 1
    % plot
    hold off
    plot(t, angle_rad, 'DisplayName', 'Angle');
    hold on;
    plot(locs, pks, 'o', 'DisplayName', 'Peaks');
    xlabel("Time (s)");
    ylabel("Angle (rad)");
    title("Angle over Time for an Oscillating Rocky");
    legend;
    savefig("figs/gyroscope_params.fig");
    saveas(gcf, "figs/gyroscope_params.png");
```

```
end

end
```

Motor Parameters

This file calculated K_{motor} and τ for Rocky.

```
function [K, tau] = motor_parameters(varargin)

% load stepper data
steppy = load("mats/steptest.mat");

% create custom exponential fit
custom_exp_func = @(K, tau, x) K*(1-exp(-x./tau));
custom_exp_fit = fittype(custom_exp_func);

% create an exponential line of best fit
good_xs = [steppy.t(steppy.t < 0.5); steppy.t(steppy.t < 0.5)];
good_ys = [steppy.outputL(steppy.t < 0.5); steppy.outputR(steppy.t <
    0.5)];
fit_params = fit(good_xs, good_ys, custom_exp_fit, 'Start', [1,1]);

% extract K and tau
K = fit_params.K / mean(steppy.input);
disp(mean(steppy.input))
tau = fit_params.tau;

if nargin == 1
    % plot
    hold off
    plot(steppy.t(steppy.t < 0.5), steppy.outputL(steppy.t < 0.5), ...
        '*', 'DisplayName', 'Experimental Left Wheel Data')
    hold on
    plot(steppy.t(steppy.t < 0.5), steppy.outputR(steppy.t < 0.5), ...
        '*', 'DisplayName', 'Experimental Right Wheel Data')
    plot(steppy.t(steppy.t < 0.5), ...
        custom_exp_func(fit_params.K, tau, steppy.t(steppy.t < 0.5)),
            ...
        'DisplayName', 'Best Fit for Both Wheels Data')
    xlabel("Time (s)");
    ylabel("Velocity (m/s)");
    title("Velocity over Time with a Stepper Input");
    legend;
    savefig("figs/motor_params.fig");
    saveas(gcf, "figs/motor_params.png");
end

end
```

Control Parameters

Solve for K_p and K_i given desired poles and system constants.

```
% Rocky_closed_loop_poles.m
%
% 1) Symbolically calculates closed loop transfer function of PI
   disturbannce
% rejection control system for Rocky.
% Currently no motor model (M =1).Placeholder for motor model (1st
   order TF)
%
% 2) Specify location of (target)poles based on desired reponse. The
   number of
% poles = denominator polynomial of closed loop TF
%
% 3) Extract the closed loop denomiator poly and set = polynomial of
   target
% poles
%
% 4) Solve for Ki and Kp to match coefficients of polynomials. In
   general,
% this will be underdefined and will not be able to place poles in
   exact
% locations.
%
% 5) Plot impulse response to see closed-loop behavior.
%
% based on code by SG. last modified 3/12/21 CL

function [Kp, Ki] = control_parameters(K_motor, tau, wn, l_eff)
% clear all;
% clear all;

syms s a b l g Kp Ki Jp Ji Ci    % define symbolic variables

Hvtheta = -s/l/(s^2-g/l);        % TF from velocity to angle of pendulum

K = Kp + Ki/s;                   % TF of the PI angle controller
M = a*b/(s+a)                  % TF of motor
% M = 1;                           % TF without motor
%
%closed loop transfer function from disturbance d(t)totheta(t)
Hcloop = 1/(1-Hvtheta*M*K)


pretty(simplify(Hcloop))        % to display the total transfer function

% Substitute parameters and solve
% system parameters
% [wn, l_eff] = gyroscope_parameters();
% [K_motor, tau] = motor_parameters();

g = 9.81;
l = l_eff;  %effective length
a = 1/tau;           %nomical motor parameters
b = K_motor;        %nomical motor parameters

Hcloop_sub = subs(Hcloop) % sub parameter values into Hcloop

% specify locations of the target poles,
% choose # based on order of Htot denominator
% e.g., want some oscillations, want fast decay, etc.
p1 = -1 + wn*i
p2 = -1 - wn*i
p3 = -wn
p4 = -wn


% target characteristic polynomial
% if motor model (TF) is added, order of polynomial will increases
tgt_char_poly = (s-p1)*(s-p2)*(s-p3)*(s-p4)

% get the denominator from Hcloop_sub
[n d] = numden(Hcloop_sub)

% find the coefficients of the denominator polynomial TF
coeffs_denom = coeffs(d, s)

% divide though the coefficient of the highest power term
coeffs_denom = coeffs(d, s)/(coeffs_denom(end))

% find coefficients of the target charecteristic polynomial
coeffs_tgt = coeffs(tgt_char_poly, s)

% solve the system of equations setting the coefficients of the
% polynomial in the target to the actual polynomials
solutions = solve(coeffs_denom(1:2) == coeffs_tgt(1:2),  Kp, Ki)

% display the solutions as double precision numbers
Kp = real(double(solutions.Kp))
Ki = real(double(solutions.Ki))

% Location of the poles of the closed-loop TF.
% NOTE there are only 2 unknowns but 3 polynomial coefficients so
% the problem is underdetermined and the closed loop poles don't exact
% match the target poles.
% use trial-and-error to tune response
closed_loop_poles = vpa (roots(subs(coeffs_denom)), 4)

% Plot impulse response of closed-loop system
    TFstring = char(subs(Hcloop));
```

```matlab
    % Define 's' as transfer function variable
    s = tf('s');
    % Evaluate the expression
    eval(['TFH = ',TFstring]);
%     figure (1)
%     impulse(TFH, 2);   %plot the impulse reponse
end
```