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));
    % 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
```