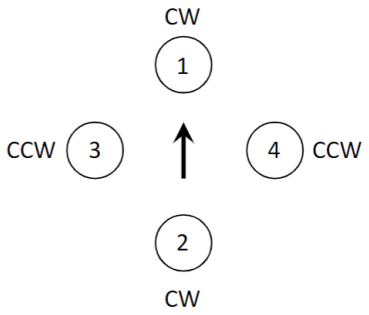
ME 457 – Mechatronic System Design
CHECKPOINT #2: Due in labs the week of November 1st, 2021

Checkpoint Synopsis:
The objectives of Checkpoint #2 is to create a short How-To manual that describes how to implement and tune a **roll rate** stabilization controller, demonstrate the stabilization for roll rate, **AND** describe how the cascade controller works along with the stabilized roll angle test data. You will need to describe how a PID controller works, how to implement the controllers, and how to perform **closed loop** gain tuning. The manual should be written such that you could read the manual in several years and be capable of implementing and tuning a cascade PID controller.

1) Describe in detail how a PID controller works **conceptually** in the context of stabilizing the quadcopter roll **rate**. What do each of the gains do?

The x component(north) data from the gyroscope must be ran through a second order Butterworth filter. The datasheet for the gyroscope will provide the scalar value that the filtered signal can be divided by to provide accurate roll rate data. The PID rate controller will have three components, proportional, integral, and derivative, and use the filtered roll rate data and a desired roll rate variable. The error is the difference between the desired roll rate and actual filtered roll rate. The error will be used in the controller. To roll a plus configuration quadcopter motor commands must be sent to motors 3(west) and motor 4(east).



Motor Commands:

$$Motor\ 3 = throttle + (P\ controller + I\ controller + D\ controller)_{Roll}$$
$$Motor\ 4 = throttle - (P\ controller + I\ controller + D\ controller)_{Roll}$$

Each controller component has a scalar component, K_P, K_I, and K_D (gains). The proportional controller is the product of K_P and the error. The integral controller is the product of K_I and the integral of the error. The derivative controller is the product of K_D and the derivative of the error. The purpose of the proportional controller is to increase/decrease the rise time. Increasing K_P will increase % overshoot and decreasing K_P will decrease % overshoot. The purpose of the integral controller is to reduce steady state error. Increasing K_I will increase % overshoot and decreasing K_I will decrease % overshoot. The purpose of the derivative controller is to reduce %

overshoot. Increasing K_D will decrease rise time and decreasing K_D will increase rise time.

2) Describe in detail how to experimentally tune the roll **rate** stabilization controllers using closed loop gain tuning.

Once the controllers are built a gain tuning method (Ziegler-Nichols) must be implemented to establish starting K_P, K_I, and K_D values. To perform the Ziegler-Nichols method set the desired roll rate to 25 $\frac{deg}{sec}$ and using only the proportional controller scale up or down K_P until the drone oscillates without making full rotations. The K_P value used to produce the oscillations is known as K_U. The period between the oscillations is known as P_U. K_U and P_U are used in a Ziegler-Nichols table to provide starting K_P, K_I, and K_D values.
Ziegler-Nichols Table With Equations:

	K_P	T_I	T_D
P	$.5K_u$	-	-
PI	$0.45K_u$	$P_u/1.2$	-
PID	$.6K_u$	$P_u/2$	$P_u/8$
PD	$0.8K_u$	-	$P_u/8$

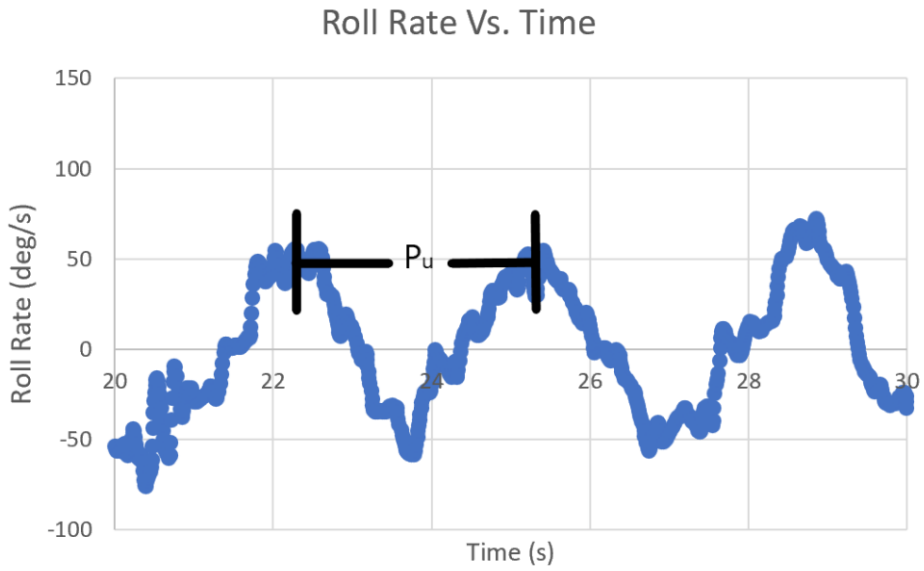
$$K_I = \frac{K_P}{T_I}$$

$$K_D = K_P \times T_D$$

From here K_P, K_I, and K_D need to be increased/decreased according to the K_P, K_I, and K_D relations stated above until the following target parameters are met.
Parameters:

- Percent Overshoot $<$ 15%
- Settling Time $<$ 1 second
- Rise Time $<$ .4 seconds

3) Show the <u>closed loop</u> Z-N test with the roll oscillations. Show your values/calculations for the PID gains.
Using $K_P = .007$:



Therefore $K_U = .007$ & $P_U = 25.38 - 22.23 = 3.15\ seconds$

	K_P	T_I	T_D	K_U	P_U
P	0.0035	-	-	0.007	3.15
PI	0.00315	2.625	-		
PID	0.0042	1.575	0.39375		
PD	0.0056	-	0.39375		

P controller:

K_P
0.0035

PI controller:

K_P	K_I
0.00315	0.0012

PID controller:

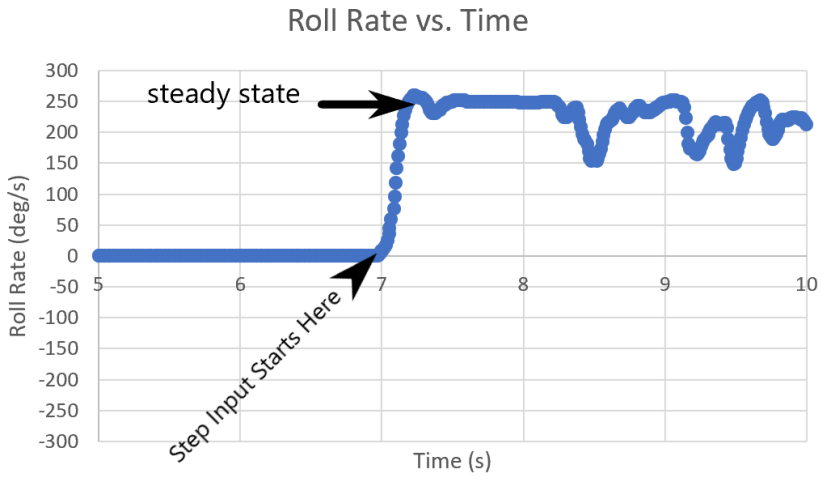
K_P	K_I	K_D
0.0042	0.002667	0.001654

PD controller:

K_P	K_D
0.0056	0.002205

4) Conduct a step response **using your Z-N gains** where you the quadcopter goes from a desired rate of zero to 30 degrees/s. Show the results (**clearly indicate where the step input is started)** and estimate the percent overshoot, settling time, and rise time (time to go from 0 degrees to 30 degrees/s). Comment on your results.

Using $K_P = .0042 \quad K_I = .02667 \quad K_D = .001654$ and $Step\ Input = 30\frac{Deg}{s}$:



$$Percent\ Overshoot = \frac{259 - 250}{250}(100\%) = 3.6\,\%$$

It appears that the steady state value is approximately 250 deg/sec. If this is true, then the % overshoot is overperforming. Based on this assumption and the fairly controlled oscillations, K_D appears to be performing decently. Increasing K_I would reduce the large steady state error and increase the small % overshoot. It appears K_I needs to be increased.

$$Settling\ Time = 7.66 - 6.97 = .69\ seconds$$

(The roll rate will never settle due to the unbalanced system. This is a predicted estimate of what the settling time might be if the system was balanced.)

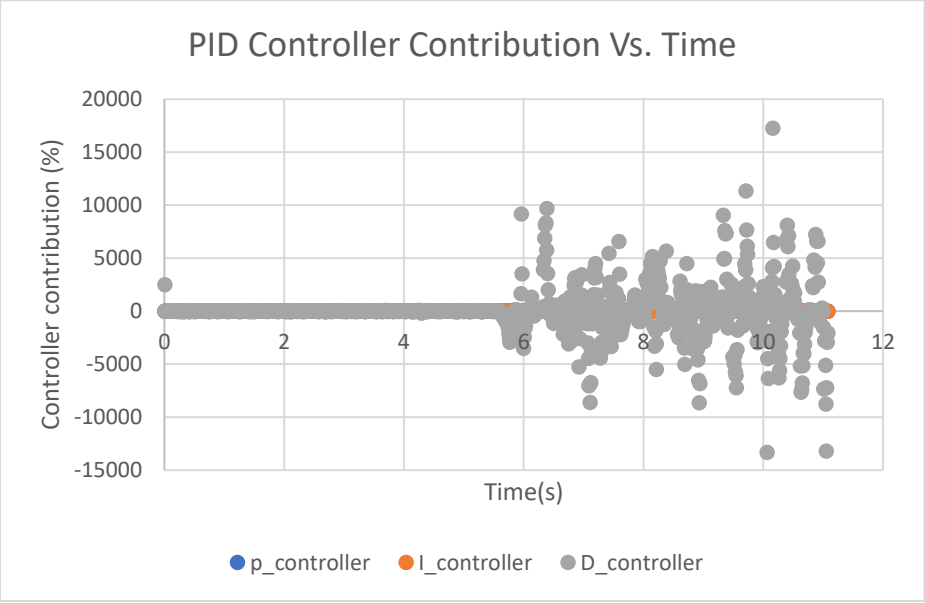
The estimated settling time is overperforming. This also points to K_D performing decently.

$$Rise\ Time = (7.19 - 6.97) \times .9 = .198\ seconds$$

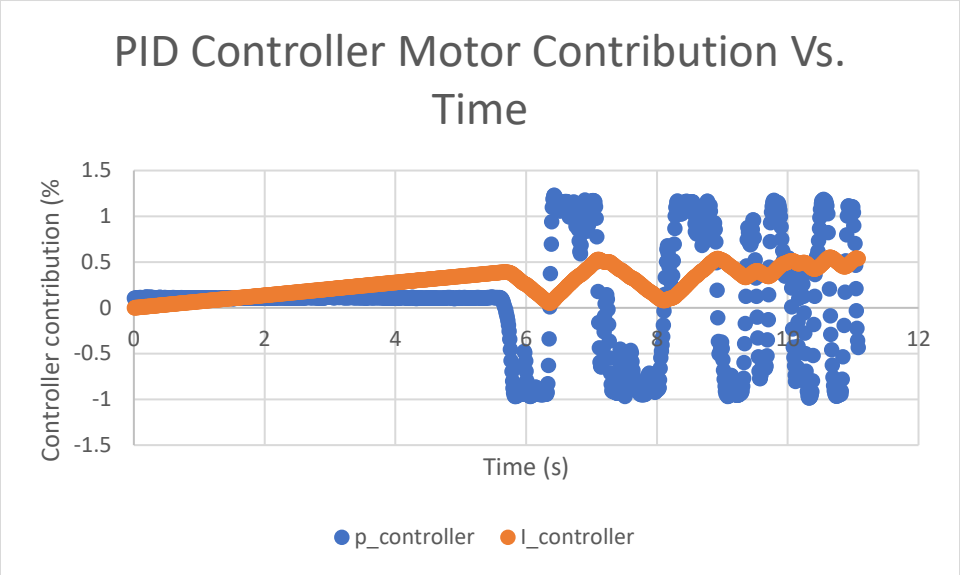
Although the rise time is below the acceptable threshold, the steady state error is way to large. The gain is currently too large. It appears that K_P is dominant so it will be decreased during the initial tuning trials. It is suspected that K_I will need to increase but during the initial trials it will remain constant.

5) In order to safely fly a quadcopter, your rate stabilization controller needs to be fast and accurate. Modify your gains until you have met the required performance for a 30-degree/s step response. **Percent overshoot < 15%, settling time < 1.0s,** and **rise time < 0.4s.** Show the results for **PID controller**, with the step input time clearly indicated. Make sure to provide your estimated percent overshoot, settling time, and rise time. Remember, you want your quadcopter to be as stable as possible (you will be performing free-flight testing in several weeks and the more stable your controller is the easier it will be to fly your quadcopter).

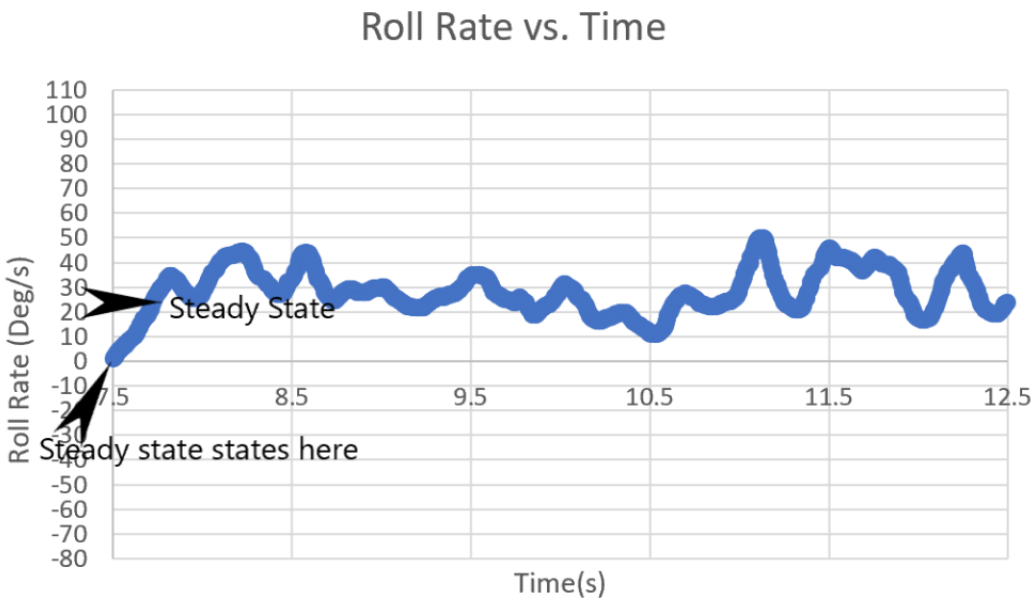
PID Ziegler-Nichols: $K_P = .0042 \quad K_I = .02667 \quad K_D = .001654$



Noise from the derivative controller is too large to see proportional and integral controller so below is a plot of just the proportional and integral controller.
PID Ziegler-Nichols: $K_P = .0042 \quad K_I = .02667 \quad K_D = .001654$



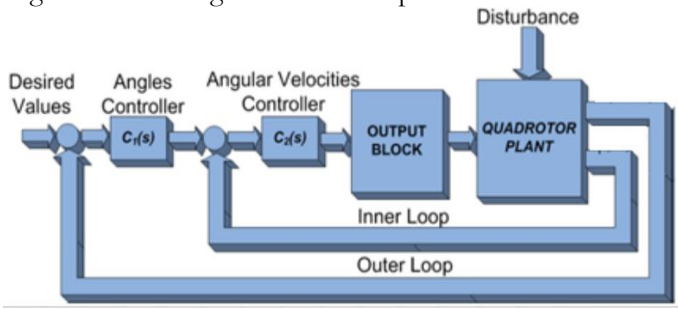
Plotting the PID controller components individually revealed that the derivative controller was noisy and the integral controller is subject to drift. The integral and derivative controller filters only successfully filtered the derivative controller so the tuning trials were done with a PD controller starting with Ziegler-Nichols values produced the following results
PD: $K_P = .003 \quad K_D = .002$



It appears after the max peak the predicted steady state, $25\ \frac{deg}{s}$, is about the peak oscillations. Rise time is approximately .25 seconds and the signal never settles. Percent Overshoot is difficult to determine because the max peak is a the second peak. The percent overshoot is approximately 50%-70%. These gains provide the current best performance but don't meet the required parameters.

6) Describe the cascade roll **angle** control architecture. What does the outer loop do? Why do we need the outer loop?

A rate controller is useful but difficult to control therefore it is not at all practical. This is why it must be integrated with a angle controller to produce a cascade controller.



The figure above is a block diagram of the cascade controller. The inner loop is rate controller. In a cascade controller the desired rate is the output from the angle controller. The input to the angle controller is the angle error. The angle error is the difference between the desired input from the user and the actual angle.

7) Experimentally tune your roll **angle** control system (cascaded inner and outer loop controller) with a 20 degree step input in the desired **angle**. Your controller must be capable of satisfying the following performance metrics for the 20 degree step input: **Percent overshoot < 15%, settling time < 1.2s,** and **rise time < 0.4s**. Show your results in a figure, provide your final gains for the full controller, and describe what you found.

I could not get my roll controller to function correctly. The erratic behavior of the drone is so much so that a roll vs. time graph would not provide any useful information.
Below is the cascade controller code:

```
# initilized variables outside the loop
roll_rate_error_old = 0
error_i = 0
error_i1 = 0
error_i2 = 0
f_error_i1 = 0
f_error_i2 = 0
error_d1 = 0
error_d2 = 0
f_error_d1 = 0
f_error_d2 = 0
throttle = 50
#desired_roll_rate = 25
desired_roll = 45
kp = 0.003
ki = 0
kd = 0.002
kp_angle = 1
    # cascade controller
    desired_angle_error = desired_roll - filtered_acc_roll
    desired_roll_rate = kp_angle * desired_angle_error
```

```
roll_rate_error = desired_roll_rate - filtered_gx

# proportional roll controller
proportional_controller = kp * roll_rate_error

# roll error integral
error_i_new = ((roll_rate_error + roll_rate_error_old)/2)*(now-then)
error_i = error_i + error_i_new

# roll error integral  filter
filtered_error_i = (b0he*error_i) + (b1he*error_i1) + (b2he*error_i2) -
(a1e*f_error_i1) - (a2e*f_error_i2)
error_i2 = error_i1
error_i1 = error_i
f_error_i2 = f_error_i1
f_error_i1 = filtered_error_i

# integral roll controller
integration_controller = ki*error_i

# roll error differentiation for Kd
error_d = (roll_rate_error-roll_rate_error_old)/(now-then)
roll_rate_error_old = roll_rate_error

# roll error derivative filter
filtered_error_d = (b0l*error_d) + (b1l*error_d1) + (b2l*error_d2) - (a1*f_error_d1) -
(a2*f_error_d2)
error_d2 = error_d1
error_d1 = error_d
f_error_d2 = f_error_d1
f_error_d1 = filtered_error_d

# derivative controller
differentiation_controller = kp*filtered_error_d

cmd.channel[3] = throttle - proportional_controller- integration_controller -
differentiation_controller
cmd.channel[2] = throttle + proportional_controller+ integration_controller +
differentiation_controller
```

References

Hema, K. “Aircraft Flight Control Simulation Using Parallel Cascade Control.” *International Journal of Instrumentation Science*, Scientific & Academic Publishing, http://article.sapub.org/10.5923.j.instrument.20130202.02.html.