

# Embedded Control

## Lab 6: Gondola Report

*Alseny Sylla, Nathan Pritchard, Deborah Lark*

***Course Instructor:*** Jeff Braunstein

***Grading TA:*** Rui Zhao

***Lab Section:*** 1B

***Date and Time:*** MR 10:00 AM-12:50PM

***Due date:*** 05/11/2016 at 5:00pm

<a href="#">Introduction</a> .....	2
<a href="#">Purpose and objective</a> .....	2
<a href="#">Overview of gondola feedback control</a> .....	3
<a href="#">Result, Analysis, conclusions</a> .....	5
<a href="#">Performance plot</a> .....	8
<a href="#">References</a> .....	16
<a href="#">Division labor</a> .....	16
<a href="#">Appendix</a> .....	18
<a href="#">Circuit schematic</a> .....	18
<a href="#">Pseudo code</a> .....	19
<a href="#">Flowchart</a> .....	21

# Introduction

## Purpose and objective

This lab is a transition from the controlling car to the blimp Gondola. Its main purpose is to apply the knowledge we gained in implementing all the functionality for the smart car to the gondola. Just like in lab five, we controlled the speed and heading of the Gondola when knocked off course by implementing an ultrasonic ranger, an electronic compass, an LCD with keypad, and a Wireless RF Serial Communication Link. The compass is used to control the heading the Gondola should point to, and the ranger to keep track of the distance from an obstacle. The LCD with keypad is used to compute the values for the desired heading and the proportional and derivative gains while the RF is used as communication center between the Gondola and the user's computer. Once the code is downloaded in the Gondola, it asks the user to enter a desired heading which will be its target. The tail fan will spin either clockwise or counterclockwise, fast or slow until the gondola points to the desired heading. But this will only happen if the ranger doesn't detect an obstacle within 10 to 90 cm range. For this lab there is a 50 cm nominal distance. When there is an obstacle below the nominal distance, the desired heading will be adjusted up to negative 180 degree or up to positive 180 degree if the obstacle is greater than the nominal distance but below 90 cm. We used proportional gain ( $k_p$ ) to control the system. With smaller  $k_p$ , the gondola came close the desired heading with no overshoot while

with higher gain the gondola would overshoot the chosen heading because of the high fan speed. However the use of derivative gain constant ( $k_d$ ) enabled us to use higher gains with no overshoot.

## Overview of gondola feedback control

As it was mentioned above, the proportional and derivative control otherwise known as PD control are used to avoid some natural damping the gondola experiences while performing its error correction. We were provided an algorithm to control the PD control.

$$U(t) = k_p * e(t) + k_d * (d/dt) e(t)$$

$u(t)$ = signal sent

$k_p$ = proportional gain constant

$k_d$ = the derivative gain constant

$e(t)$  = error.

Thanks to this algorithm, we were able to create a system that can adjust the heading even when knocked far off course.

Beside from this equation, we also needed to calculate the derivative changing error. The implementation for this equation was based on the previous and current error. We considered the previous error as an error that happened a certain time ago, like 40 millisecond. This equation looks like this:

$$u(t) = k_p * e(t) + k_p (previous\_error - current\_error).$$

In addition, it's also important to mention that the servo motor was used to handle the rotary movement which controlled using a pulsewidth modulated system. This is done thanks to some essential components of the Gondola such as the potentiometer, a comparator dge that should be applied in order to apply a speed relative to the heading.

The circuitry of the gondola is pretty similar to the circuit we build for lab4. It has two pull-up resistors that allow two way transmission. The SCL (serial clock line) connection allows the master and the slave devices to keep track of the timing while the SDA (serial data line) permit data to be sent and received.

Even though we have explained the functionalities of the compass, ranger, LCD with keypad and the wireless RF serial link modules, it will be good to still give a little bit of details on how they are significant for the Gondola. Again, the compass is used to control the steering angle of the Gondola by giving it output values that range from 0 to 3600. Thanks the magnetic field sensor inside of it, it can read the current direction with respect to magnetic north. It is connected to both power and ground, and utilized wiring the SCL and SDA lines. On the side, we have the ranger that detects object in a specific scope. This device sends out high frequency in order to determine the distance at which the object is from it. By counting the time it takes for the signal to come back, it can figure out the distance. In this lab, we used it to adjust the heading by either positive or negative 180 degree if there an object at certain range of nominal distance. I mentioned above how the LCD with keypad allows us to compute values. But how does it work? Thanks to certain functions such as `kpd_input()` from the `i2c` header file, the user can compute a value up to `0xFFFF`. It is also requires power and ground to make it work. In previous lab, we

have been using wire to connect the computer to microcontroller. But thanks to the wireless RF serial link, we can connect them without the use of wire so that the Gondola can spin freely.

## Result, Analysis, conclusion

After many days of debugging and stress, we were finally able to meet all the requirements for this lab. We had the gondola working exactly as it was supposed to. It successfully steers to the desired heading, and oscillates back and forth until it points to where it is supposed to then comes to a stop. It was beautiful watching this, because it was capable of increasing or decreasing the speed of the tail fan, either clockwise or counterclockwise, depending on how far it is from the entered heading until it reaches its destination. The final code was fully functioning and was proven by one the TA.

Before the TA even checked it, we tried different inputs and exposed the gondola to different scenarios just to make sure that the lab works well. One of the verifications that we did was to manually spin the Gondola clockwise really fast to check if it would automatically slow down and point to the desired heading. After doing three to four times, no matter how fast we spin it, the gondola will still slow down by itself and adjust its direction, most of the time by rotating in the opposite direction until points to the desired heading. We tried the other way as well by spinning counterclockwise and everything worked as expected.

Since the ultrasonic ranger was used to adjust the desired heading when it detects a range within certain scope, we placed an object within to check our work. As it was specified in the laboratory instructions, the desired heading was supposed to be adjusted up to negative 180

degree when the ranger detects an object from 10 to 50 cm away and up to positive 180 degree when it detects an object from 50 to 90 cm. To confirm this, we placed objects in front of the sensor at varying distances during different stages of the Gondola's rotation. The Gondola consistently reached the desired heading and adapted to the changes that were introduced, proving that the system is quite, stable and reliable. Beside from spinning and placing objects, we analyzed the plots that were made using different proportional and derivative gain constant values. Normally, larger gains will more likely exhibit underdamped responses while smaller gains will more like exhibit overdamped responses. Knowing that, we tested small gain values and remarked that the gondola would hardly budge, and would take a long time to approach the desired heading.

A normal balance of  $k_p$  and  $k_d$  would show a good run while an overdamped system reaches its destination in a faster time. Further analysis of the plots is provided in "performance plots" section.

By doing this lab, we learned some significant concepts. This lab introduced us to the PD control for the first time. We have been working with the smart car the whole time, which, unlike the Gondola, has a natural damping system to allow for a quick response. Therefore we had to learn how the derivative gain works since it applies this damping, permitting a large proportional gains in order to get a quick response. One of the main concepts that we were glad to learn was how to stop the Gondola from continuously spinning when it was spun by high velocity. We realized that when the error was above 180 degree, the gondola would tried to get to the desired heading the long way instead of taking the shorter and faster path. Therefore that would cause it to overshoot the desired heading, which eventually to lead to a bigger error again. We fixed that

by setting the pulse width to be neutral if the heading has changed for more than 60 degree in within 40 milliseconds. We also learned that the PD control has a variety of real work applications like the Mars rover landing.

Over the completion of this lab, one the most dominant difficulty was the availability of the Gondola. With limited amount of Gondola and many groups, we were obliged to wait for a long period before the code could be tested. Another problem was the tail fan would spin faster in one direction than the other. But this didn't affect much the performance of the gondola. This is something that could have been fixed if we had more time and more Gondolas.

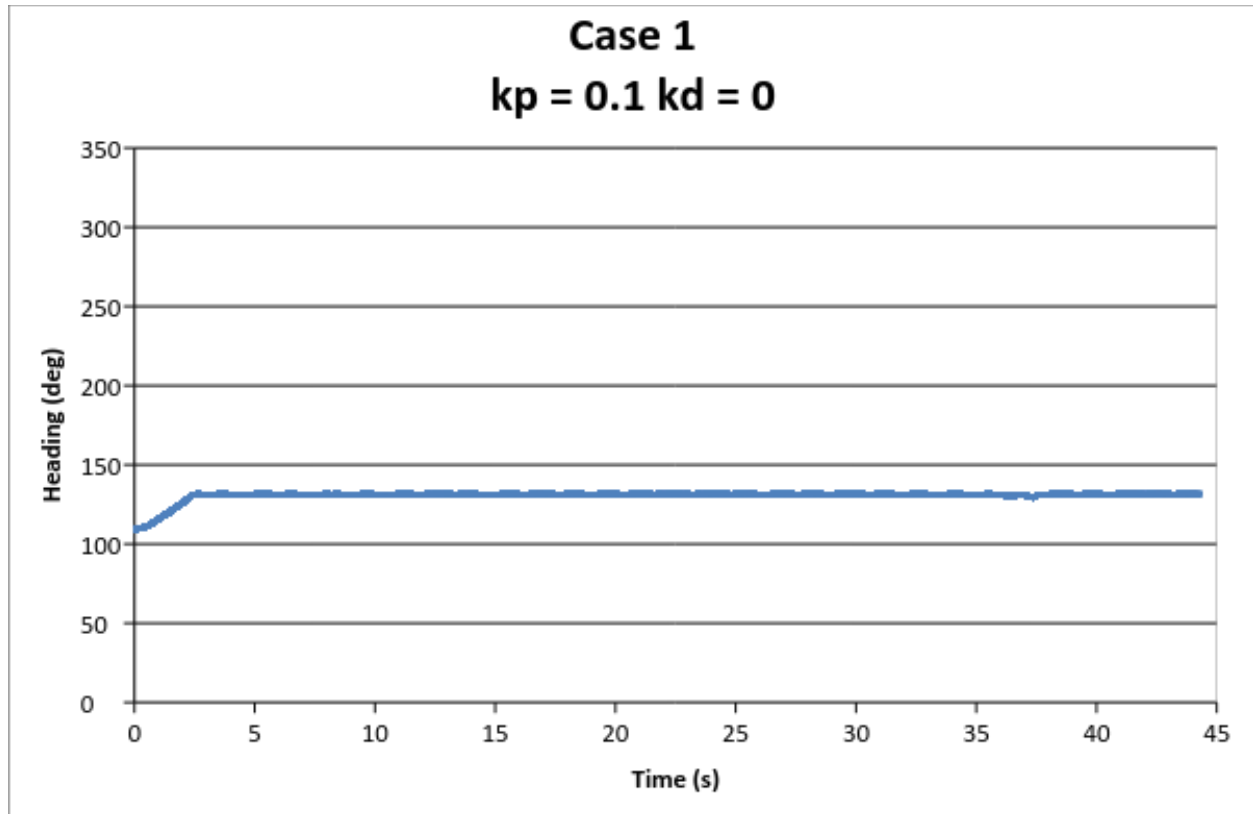
Since we wrote two codes for this lab, one the problems one the teammates encountered were mostly bugs in the code. When calculating the pulsewidth needed to be sent to the tail fan, pw\_center was not added since he accidentally set its initial value to 0. This was easily solved by going over the code and setting its value to 2760. Timing was also one the problems. Since this lab is at the end of the semester, students usually exams and final projects, we couldn't all meet up at one time. Beside from these problems mentioned above, everything else was fine.

In conclusion, this lab was completed with success and we were all happy to work together.

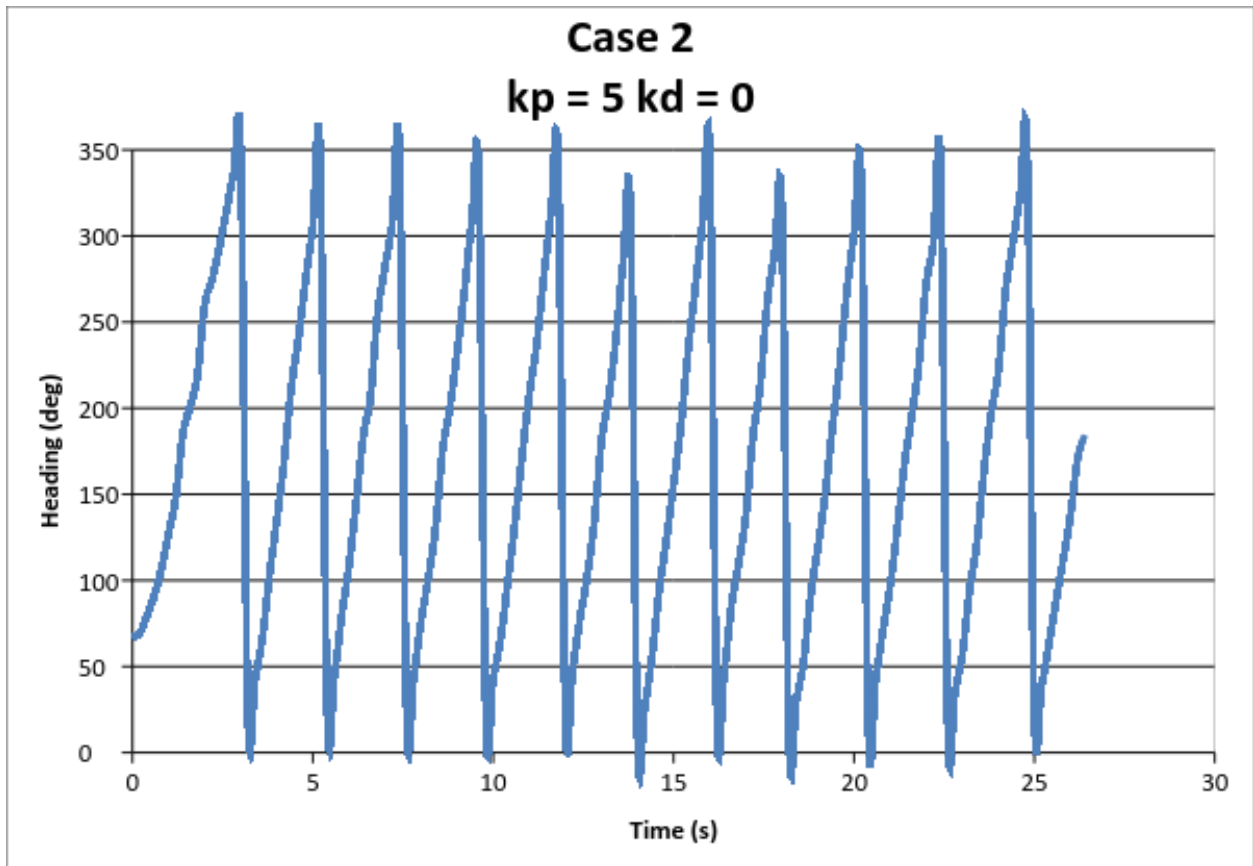
Hopefully we will apply the knowledge that we gained in this lab in real life situations. Also all the TAs were pretty helpful and friendly.



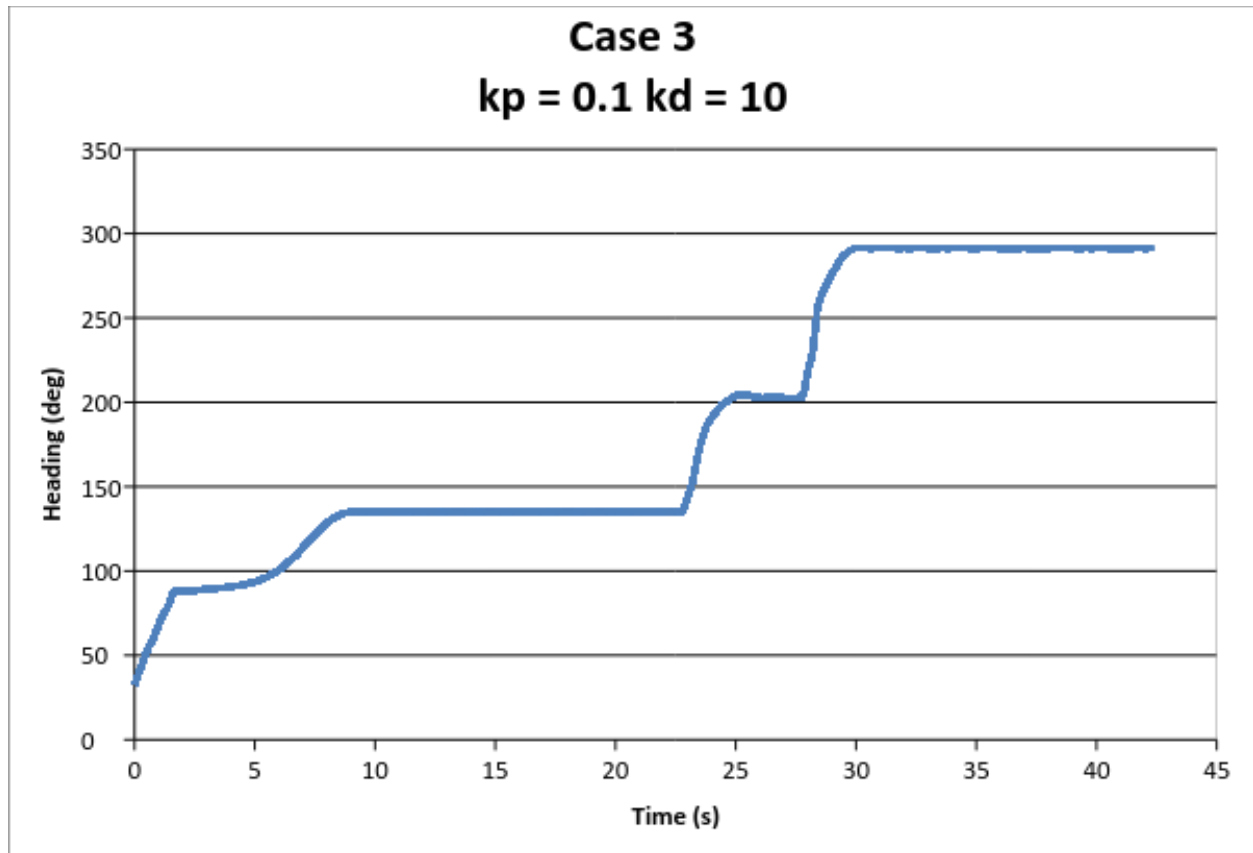
## Performance plot



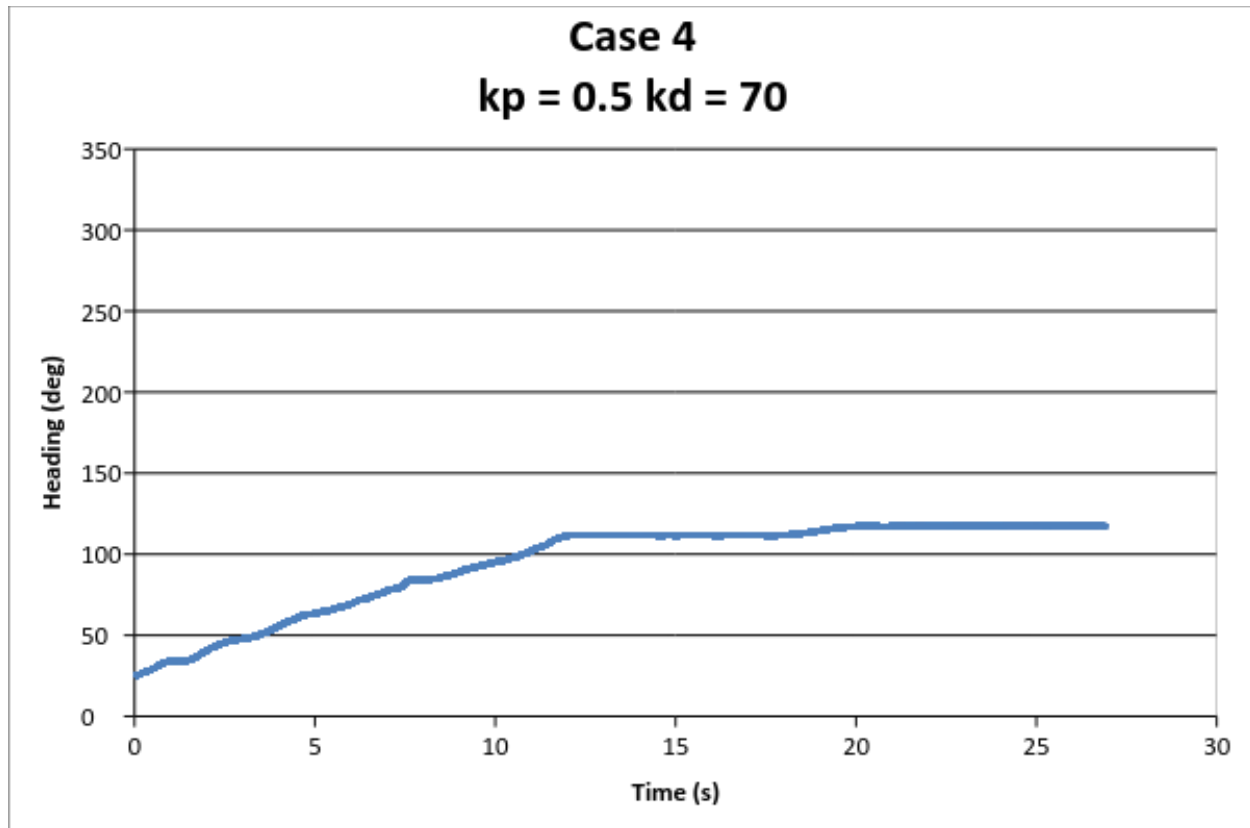
With a  $k_p$  like this, the gondola was barely able to move, because dividing the difference of the heading and desired heading the value was really small compared to the  $PW\_center$ . the pulse width calculated was close to the  $pw\_neutral$  which cause the tail fan to barely move. This straight line shows that heading barely changes.



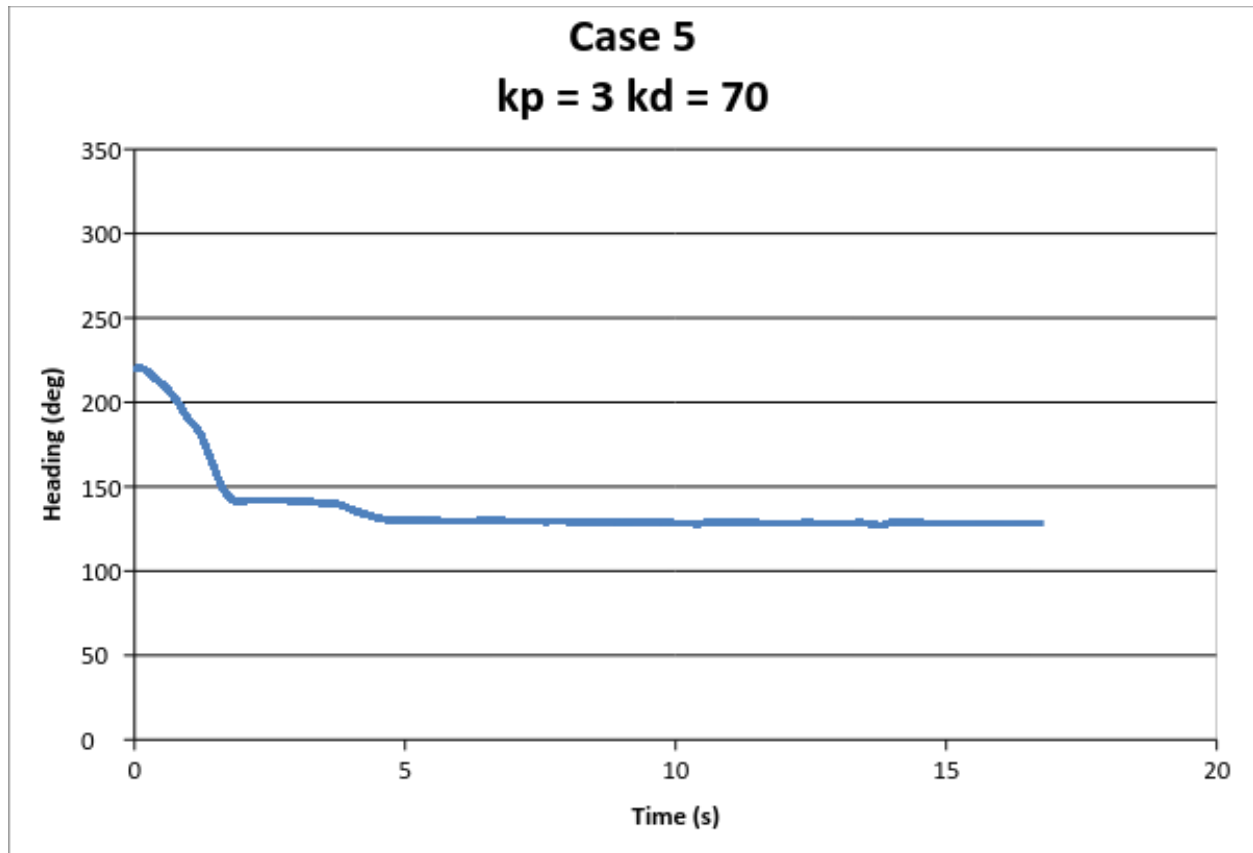
Increasing the  $k_p$  to 5 and setting the  $k_d$  to 0, the gondola was moving in circle. This means that there was no damping and therefore we expect the gondola to rotate in circle. Looking this graph, we can see that there is a move from +180 degree to - 180 degree. That's why we have straight lines down.



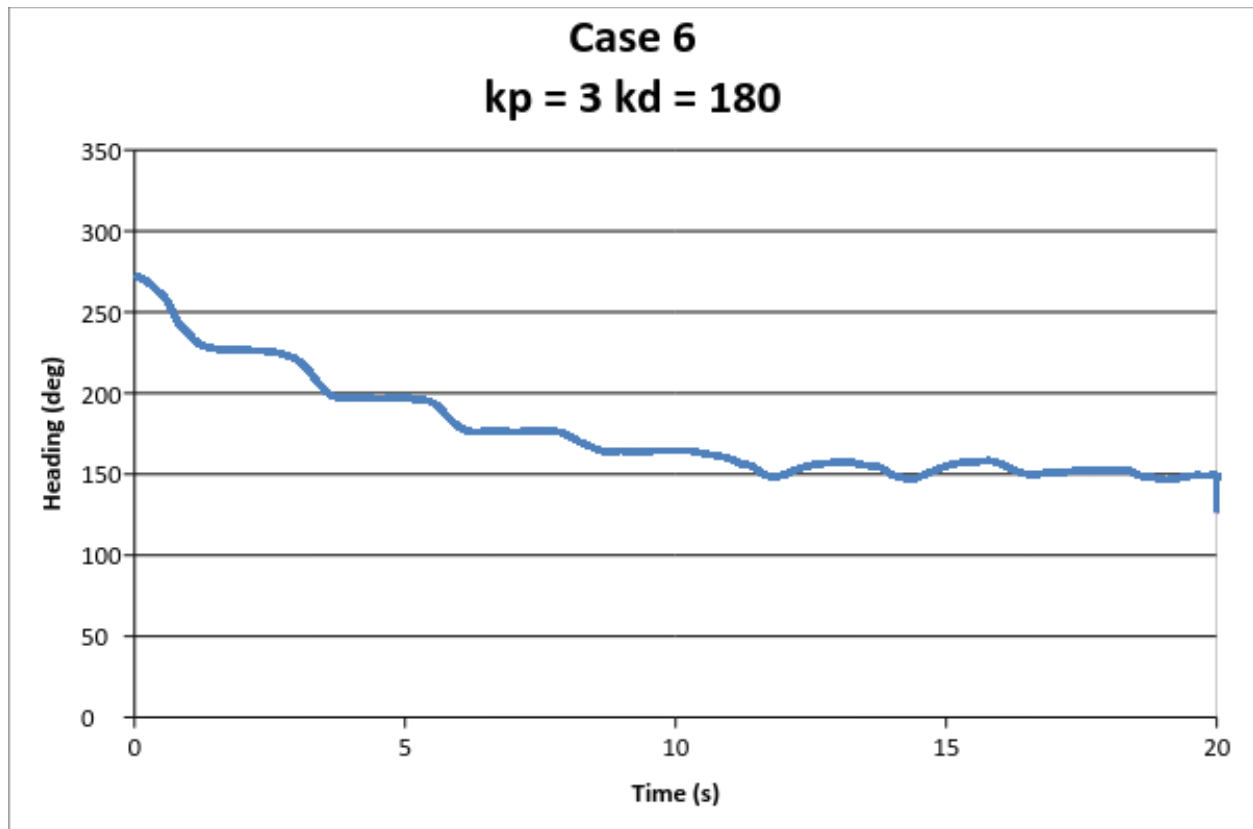
Having the value of the  $k_d$  to 10 and the  $k_p$  to .1, we produce an overdamped response. That's why we have some steady slope of the heading toward to the desired heading. In this case, the tail fan slows down as it approaches the desired value.



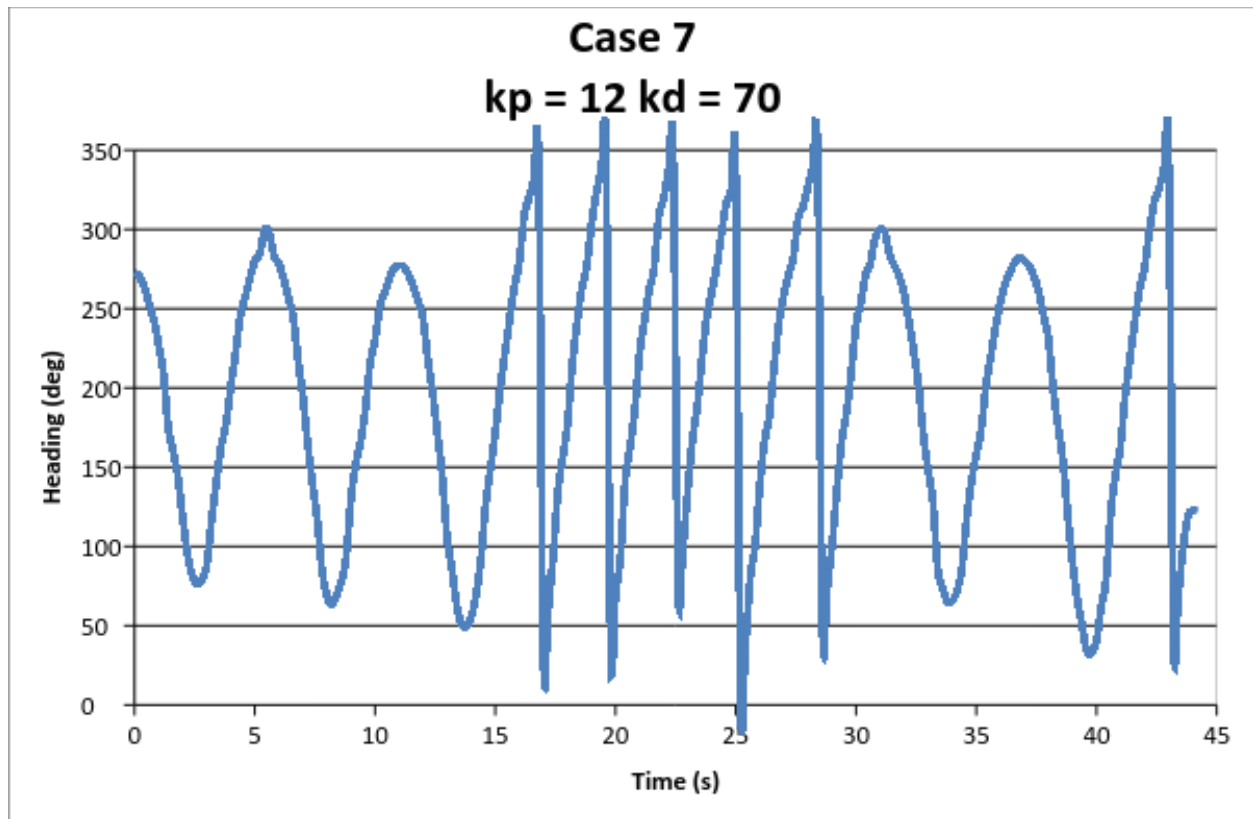
$K_p=0.5$  and  $k_d = 70$ . In this case, the system exhibits an overdamped response as well just like the previous case. We can see how the gondola is creeping to the desired heading value. This is exactly what we expected, no overshooting. Also we can see that the response happens relatively slow.



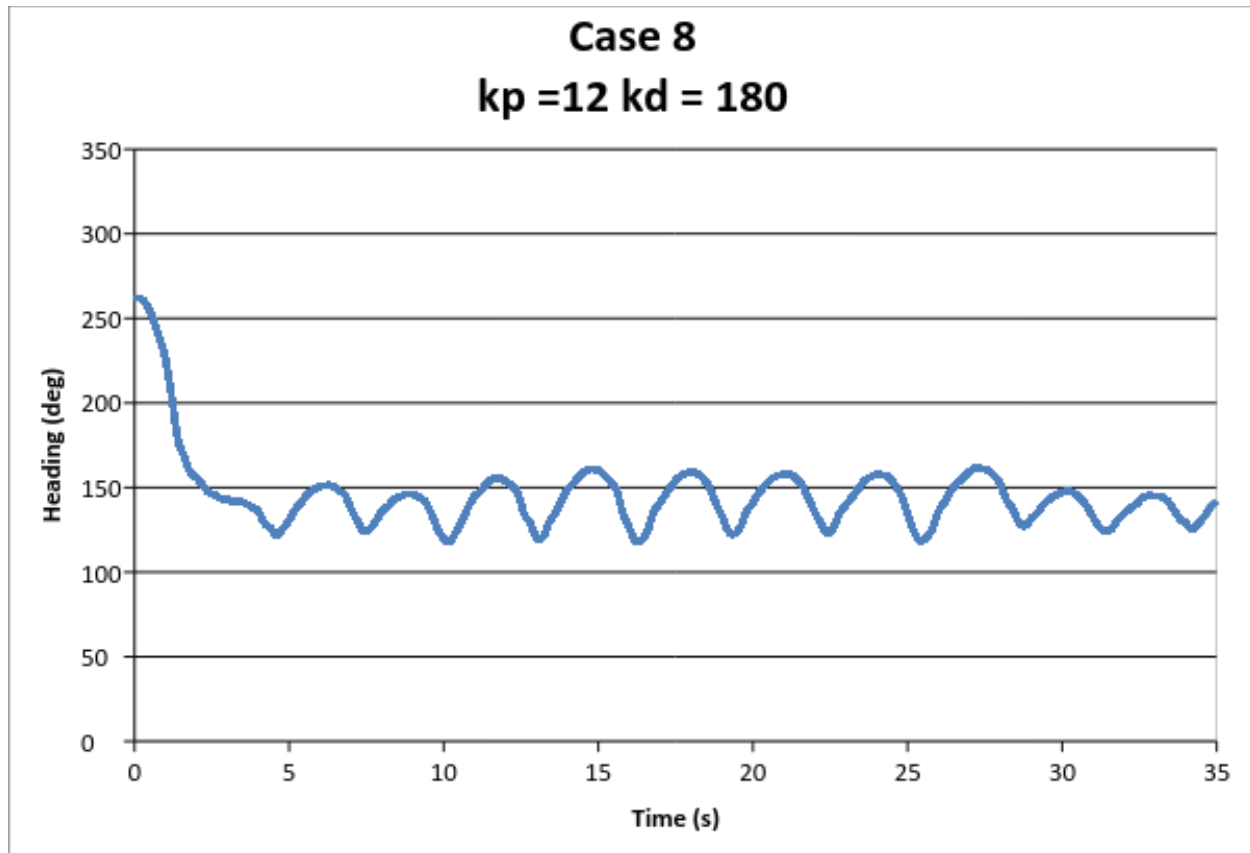
Now we have increase the  $k_p$  to 3 and still leave the  $k_d$  to 70. With a higher proportional gain constant, the system will overdamped the desired heading. What will happen is that, it starts to overshoot and correct itself. That after the 0.3 second, the gondola comes to the desired heading without any problem.



Having the same proportional gain constant and increasing the  $k_d$  to 180, the gondola will continuously oscillate up and down by trying to stabilize itself at the desired heading. That's because we have an overdamped response in this case. We don't have much of a difference between the previous case and this one. We can therefore conclude that increasing the  $k_d$  value a little bit wouldn't make much of a difference.



Increasing the  $k_p$  gain to 12 and leaving the  $k_d$  to 70 (which is normal) we expect an underdamped system. We can see from the graph that the motion of the fan is jerkier. This case is not really good case the test the work as we need a far lower value to get a better result. Even though the result is not good enough, the  $k_d$  value helps us get around the desired heading.



Keeping the  $k_p$  value to 12 and increasing the  $k_d$  to 180 causes an overdamped system with a little bit of high amplitude. We can see that the amplitude decreases a little bit and increases as well with every oscillation. The graph still tries to get to the desired heading.

Overall, the data analysis is really significant because it shows us what values are good enough to get a successful response. It shows both the worst and the best case. From this graph we see that a very low  $k_d$  and  $k_p$  value will lead to a very slow motion and sometime jerky, while high values will lead to fast motions. From here we now know how to get a critically damped, overdamped or underdamped.



# References

Litec lab manual.

## Division labor

### Academic Integrity Certification

All the undersigned hereby acknowledge that all parts of this laboratory exercise and report, other than what was supplied by the course through handouts, code templates and web-based media, have been developed, written, drawn, etc. by the team. The guidelines in the Embedded Control Lab..Manual regarding plagiarism and academic integrity have been read, understood, and followed. This applies to all pseudo-code, actual C code, data acquired by the software submitted as part of this report, all plots and tables generated from the data, and any descriptions documenting the work required by the lab procedure. It is understood that any misrepresentations of this policy will result in a failing grade for the course.

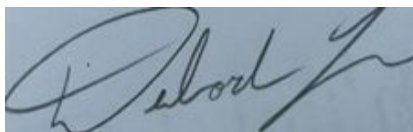
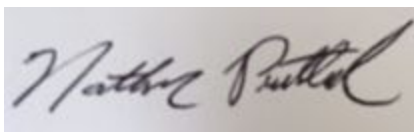
**Hardware Implementation:**      **Nate (N/A)    Deb (N/A)    Alseny (N/A)**

**Software Implementation:**      **Nate (40%)    Deb (30%)    Alseny (30%)**

**Data Analysis (if relevant):**      **Nate (30%)    Deb (30%)    Alseny (40%)**

**Report Development & Editing:**      **Nate (25%)    Deb (25%)    Alseny (50%)**

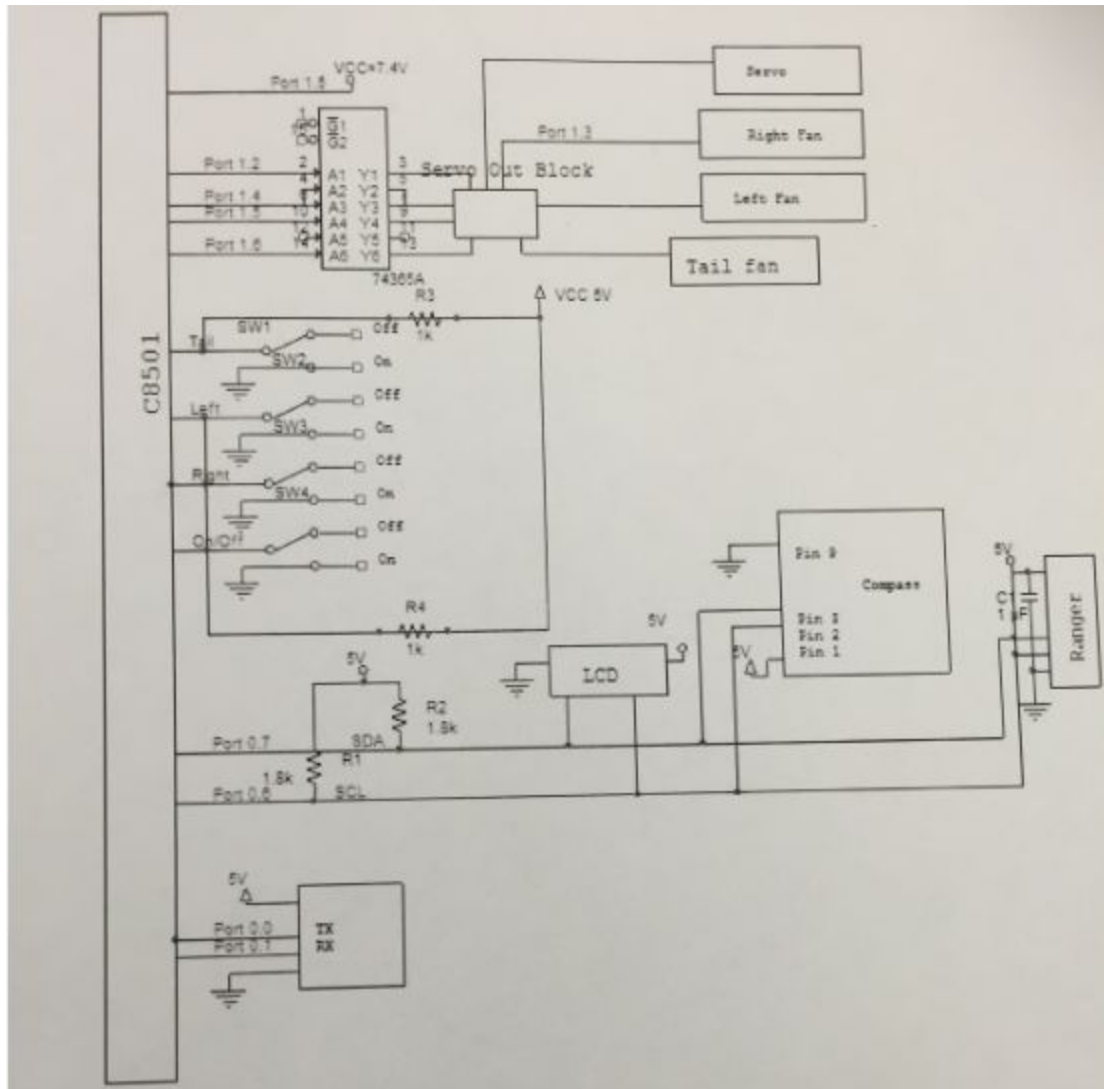
The following signatures indicate awareness that the above statements are understood and accurate:

A handwritten signature in black ink on a light blue background, appearing to read "Nate".A handwritten signature in black ink on a light blue background, appearing to read "Deb".A handwritten signature in black ink on a light blue background, appearing to read "Alseny".



# Appendix

## Circuit schematic



## Pseudo code

### Lab 6 Pseudocode

#### //Function Prototypes

```
void Port_Init(void), void PCA_Init (void), void XBR0_Init(), void SMB_Init(void), void  
ADC_Init(void), unsigned char read_AD_input(unsigned char n), void PCA_ISR (void)  
__interrupt 9,  
void tiltConfig(void), unsigned int RangerRead(void), void pause(unsigned char m), void  
ErrorandHeading(void), void Setup(void), unsigned int CompassRead(void), void  
updateFans(long fanPW);
```

#### //Global Variables

```
unsigned int __xdata PCA_start = 28672, signed int __xdata PW_FAN_C = 2750, signed int  
__xdata PW_FAN_R = 2000, signed int __xdata PW_FAN_F = 3500 signed int __xdata  
NOM_DIST = 75,  
signed int __xdata NOM_SPEED = 100, char new_range = 0, char new_compass = 0, char  
RangerCount = 0, char CompassCount = 0, signed int Kp = 0, signed int Kd = 0, signed int  
desired,  
signed int new_desired, char input, int counts = 0, char display_count = 0, int mult, int heading,  
int old_heading, unsigned int PW, long tmp_pw, signed int new_error = 0,  
signed int old_error = 0, signed int speed, signed int range = 50, int volt
```

#### //Main Function

```
call Inits (Sys Init, putchar, Port Init, XBRO Init, PCA Init, SMB Init, ADC Init)  
pause(50) //long pause for things to warm up  
call "setup" function to set gains and desired heading  
define the values to be printed  
while(1)  
    if(new_range)  
        range = ranger value  
        new_range = 0
```

```

    if(new_compass)
        heading = compass value
        call "ErrorandHeading" function to calculate error and change desired heading
based off range
        tmp_pw = motor neutral + kp*error/10 + kd*change in error
        make sure tmp_pw within range
        set motor pulsewidth
        reset motor pulsewidth based off rotational speed if need be
        new_compass = 0
    if(display_count >=10)
        calculate voltage
        printf necessary values
        lcd print heading, range, and voltage
        display_count = 0

```

## Flowchart

