Grayson Gerlich, Hannah Eisenhofer, Nathaniel Moretz

Grading TA: Zaid Bin Tariq

LITEC Lab 6:

Gondola Report

Section 2 Side B

Professor: Russell Kraft

Introduction

The goal of Lab 6 was to use the various I2C devices, servos and motors already present on the gondola to demonstrate hardware and software competencies and mimic the functionality of an oscillating floor fan. In this lab we implemented the ultrasonic ranger, a compass, an LCD display, a 12 button keypad, and an RF serial transceiver. We developed code to take user input on the keypad, and set thresholds and control gains for the control system. We also developed code to read ranger and compass outputs to control tail and thrust fans. Implementing PID control allowed the gondola to oscillate smoothly.

This lab was very similar to Lab 4 at its core, but we still had to adjust the initialization functions and variable initializations. We ended up using 4 CCMs, 0 through 3, which controlled the 3 gondola motors and the thrust servo. Unlike Lab 4, we were not worried about the I2C bus overlapping, since the ranger was only called in the initialization, and the compass was only called during oscillation. However, we still had to aggressively manage our I2C variables and calls to avoid generating issues.

Multiple gains were tested in the development of the system. At high proportional gains, the gondola overshot its target far enough to simply continuously spin in one direction. The derivative gain was added to make sure that when the gondola reached its target, oscillations were damped and it eventually settled on target.

Schematic Included as Next Page

Data Analysis

From Worksheet 11, we had some idea of the gains that would be optimal for gondola operation. After extensive testing, we settled upon a kP of 1 and a kD of 18 as the optimal gain settings. Figure 1 demonstrates “optimal” gondola operation. Figure 1 was made using data from 3 second oscillation trials. During our check off procedure, we were asked to use a 5 second oscillation time with the same gains. The results are in Figure 2. All figures are presented with heading as a raw value instead of a normalized one in order to allow direct comparison to the desired heading.

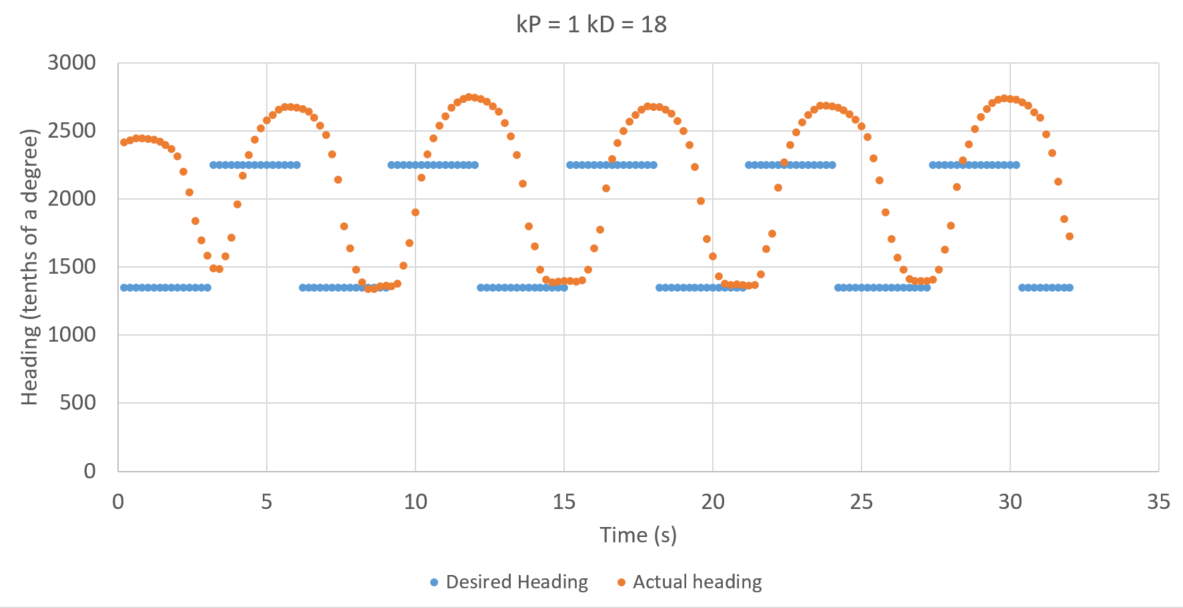


Figure 1: Gondola operation at normal gains. Only the headings are visualized here to show how clean the oscillation is. The overshoot of the highest heading is due to the motor being more powerful in the forward direction than in reverse. While hardware and software solutions are possible for this issue, we did not implement them at time of testing.

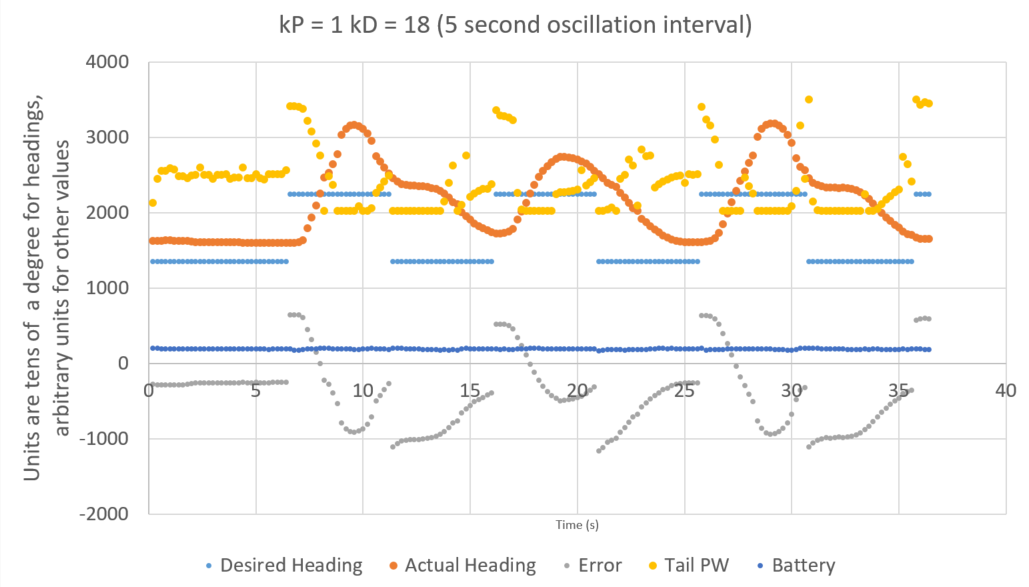


Figure 2: Normal gondola gains, but with 5 second oscillations. All data taken during normal operation is presented on the plot to provide a full image of what happens during gondola function. The battery voltage can be seen to very slightly decline over the 35 second trial. The initial state of the gondola is an interesting case. The error is relatively small, and with a small kP, the error isn’t actually enough to drive the motor in reverse with enough strength to rotate the turntable. The interesting peak structure of the actual heading curve is a result of the strength disparity between forward and reverse motor operation. The motor in forward turns fast enough for the oscillation to begin to equalize before the desired heading shifts. A different set of gains would be more appropriate for this oscillation interval. Note the tail PW following approximately along with the error value.

The two edge cases in the PD control algorithm are: low kP with high kD and visa-versa. The first case is demonstrated below in Figure 3.

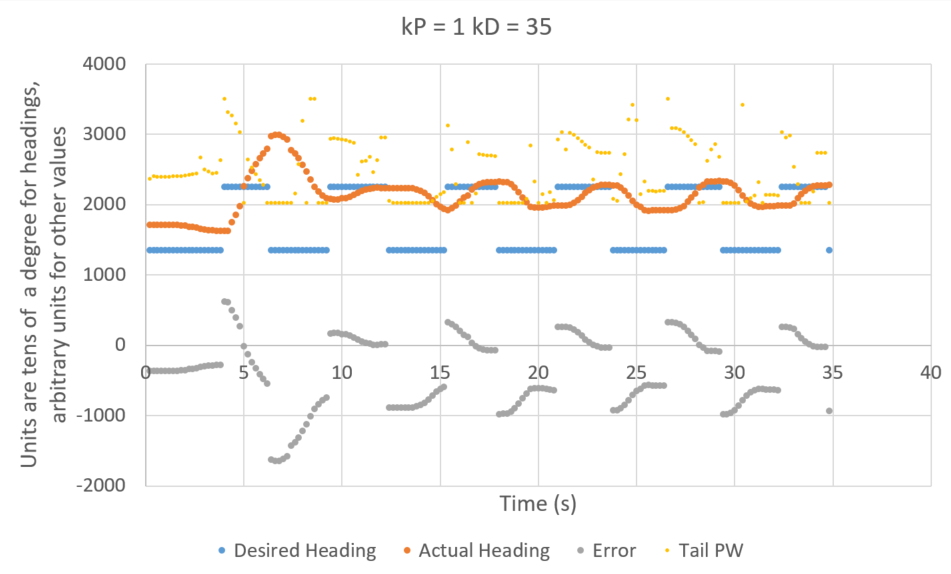


Figure 3: Gondola operation with a greatly inflated kD. As is clearly visible from the actual heading, the increased kD damps motion far too much for the kP to generate a motion in the tail fan with each change of heading. Once again, the actual heading is skewed upwards due to the greater strength of the tail fan in forward as opposed to reverse. The visual weight of the tail PW has been reduced for ease of understanding other data, but the PW curves still loosely follow the error.

The other error case, of course, is high kP and low kD, which generates the dreaded full spin. This case is shown below in Figure 4.

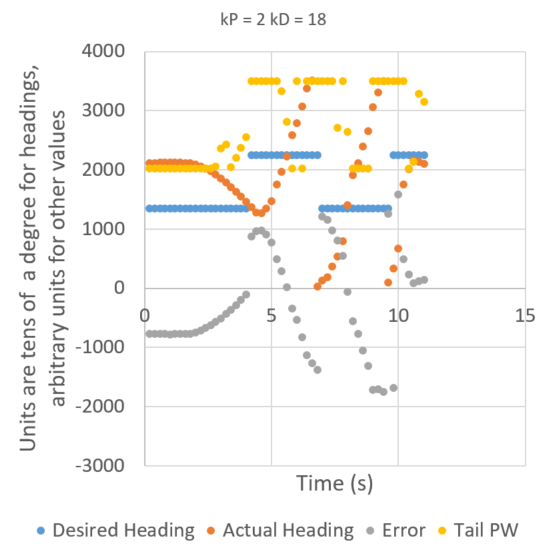


Figure 4: Gondola operation with a higher kP and lower kD. The nature of the proportional gain is such that an increase in gain of just 1 has drastic effects in gondola operation. This trial was stopped swiftly because the gondola was whipping around, but this full 360-degree spin demonstrates the danger of high kP. The drastic slopes of both the actual heading and error are a result of this deathspin.

Seeing the difference between the two edge cases, it is apparent that neither is optimal, but it took a much greater change in kD to elicit a response on the level of a small change in kP. This begs the question, can the system even out with even higher gains? Figure 5 demonstrates a failed attempt at this.

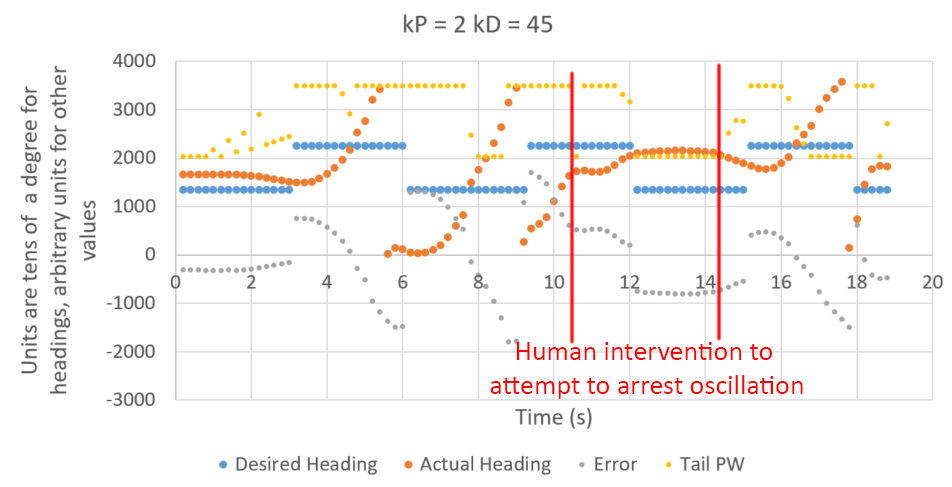


Figure 5: Gondola operation with wildly inflated kP and kD gains. The deathspin is immediately visible in the first two operations. At this point, we intervened to attempt to see if the system could stabilize, but it became immediately clear that the kP gain was simply too drastic. Given how much time Tail PW spends flatlined along the top of the plot, it would likely take a kD of close to twice the current value to reign in the oscillations.

These figures demonstrate the functionality of the gondola under most working scenarios, as well as the effects of the various gains.

Pseudocode

Variable initializations

Main

Inits

While (infinite)

While dip switch == switched

Set motors to neutral

If (first time in loop)

Get gains from user

Using LCD/keypad

Get thrust fan tilt and motor values from user

Using ranger (I2C)

Update first time in loop variable

Turn on thrust fan motors to set value

If 3 seconds have passed

If switch flag set to 0

Set desired heading to 135

Else

Set desired heading to 225

Invert switch flag

Reset 3 second count

If (heading flag)

Read the compass

Adjust servo PWM

Reset flag

If (print flag)

Battery is found using AD input

Print to serial

Reset flag

SFR Inits

Read\_compass function

Assign local variables

Use i2c read to get data

Combine the two bytes of data

Return heading

Read\_ranger function

Assign local variables

Use i2c read to get data

Combine the two bytes of data

Start new ping

Return light level

Set\_tail\_PWM function

Get error value

Adjust if greater/less than 180 deg error

Adjust PW with algorithm 6 from WS #11

Check for PW exceeding PW\_MAX/MIN

Set PW

Tilt set function

Read ranger value

Check time held at that ranger value (with margin for error/noise)

Adjust PW with linear relation between range and servo\_pw

Check for PW exceeding PW\_MAX

Set PW

Thrust set function

Read ranger value

Check time held at that ranger value (with margin for error/noise)

Adjust PW with linear relation between range and thrust\_pw

Check for PW exceeding PW\_MAX

Set PW

Set\_gains function

Assign local variables

While (infinite) (reads kP)

Read two numbers (starting at 10’s place)

Add each one together

Return 2 digit number

While (infinite) (reads kD)

Read two numbers (starting at 10’s place)

Add each one together

Return 2 digit number

Print to LCD

Read\_AD\_input function

Assign local variable

Wait for conversion to complete

Return ADC variable

PCA\_ISR function

If (CF)

Increment all three counts (LCD, switch, heading)

If heading count > 40 ms

Update heading flag

Heading count = 0

If LCD count > 200ms

Reset LCD count

Pause function

Wait 120 ms

Wait function

Wait 1000 ms

Conclusions

Lab 6 was a learning experience all around. We did Lab 6 before we did Lab 5, so the gondola was our introduction to both PD control and wireless communication with the C8051. Despite this, our algorithms worked fairly smoothly. Most of our debugging was done visually, as the gondola is very reactive and tends to show you exactly what is happening. This lab taught us a great deal about PD control and the importance of rigorously testing your gains. We have a newfound respect for controls engineers, and systems that can automatically dampen themselves appropriately are very, very cool.

Our PD control algorithm from worksheet 11 worked pretty much perfectly straight out of the gate, but one of our biggest problems was getting the control algorithm implemented. Our code beforehand was very bloated, we had lots of variables, and just didn’t have the memory space to implement a 16-byte control algorithm. The solution to this issue lead to some of our most creative (and in our opinion) most efficient coding. For example, because we only use the “range” value during the initialization/calibration step, we repurposed it for our print count while the gondola was in normal operation.

Another large challenge we faced was the 5 second wait time on the ranger for setting the thrust fans and the servo. We initially were checking for if the range stayed within +- 10 cm for 5 seconds. This seemed to work fine for middle ranges, but nothing else. We realized eventually that we were using unsigned chars, so for low and high ranges, we were hitting char overflow and the value wouldn’t remain where we expected it to. We changed this by taking absolute values between the previous range taken and the current one taken, and checking for the threshold that way.

If we were to improve this software in the future, we would pursue an integral control. Currently, we can’t damp the system enough with the derivative control to prevent forward motor motion overshoot without overdamping the system. This could be solved in two ways, either by hardcoding forward motor motion to be weaker than reverse, or by implementing an integral control. We believe the second would be a more elegant and informative solution.

Hardware implementation: N/A, GONDOLA

Software implementation:

Nathaniel: 30% Grayson: 30% Hannah: 40%

Data analysis (if relevant):

Nathaniel: 30% Grayson: 40% Hannah: 30%

Report development & editing:

Nathaniel: 40% Grayson: 30% Hannah: 30%

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_