Lab 4 Report

By Danny Alegria, Ankur Singh and Martin Xu

Embedded Control Summer 2019

Section 1

Litec Side B

Introduction

Purpose/Objectives

The purpose of this lab and report was for the team to learn how to automate some mechanical device (in this case the “smart car”) through C code. The team’s goal was to have the car drive and steer independently, using user input and readings from on-board sensors to decide how to travel.

The team gained experience in learning how to make the C8051 microcontroller work in tangent with different sensors, the SMBus, the different mechanical devices on the car itself that control the steering and car velocity, and other parts connected to the car through a circuit board, such as a potentiometer. The team learned how to use pulsewidth modulation (PWM) to set the speed of the driver motor and the steering wheels of the servo motor. The team also learned how to properly use the concept of error in order to properly linearly map the ranger and compass values to specific velocities from the drive motor and different orientation of the wheels from the servo meter.

Overview of Steering & Drive Control

The car in this lab is partially autonomous; after the user sets the parameters for desired heading, steering gain, and motor speed, the car runs without intervention.

The steering is controlled by a compass; after the user inputs a desired heading with a keypad (in 1/10s of a degree, with north being 0 degrees), the car will turn to align itself with the desired direction—the wheels will turn towards the direction if the car is going forwards, and away if the car is going backwards. The gain set by the keypad determines how aggressively the car turns—the higher the gain, the tighter the car turns.

Motor speed is dictated by two inputs—the gain set by the keypad, and the voltage determined by the potentiometer. Both serve as multipliers for the speed of the car. For example, if the gain is set to 1, and the potentiometer is set to half way, the car will travel at half of its maximum speed. If the gain is set to 2, and the potentiometer is set to half way, the car will travel at full speed.

The ultrasonic ranger determines if the car goes forward or backwards. The first time the car is run, it will go forwards; after that, it will check to see if there is anything within 20 centimeters in front of it. If there is, it will switch directions—forwards if it was going backwards, and vice versa.

Results, Plots, Analysis of Plots, & Conclusions

Verification

The required specifications are as follows:

When the switch is off, the user can

> Input a desired heading (limited to 600 to 1200 degrees, which is east +/- 30°**).**
> Input a desired gain for steering, and a desired gain for the drive motor.

When the switch is then turned on

> The car will first go forward, turning the BiLED to green and turning in the inputted direction, while outputting to the computer the relevant data.
> Every three seconds after, the ranger will check to see if there is anything within 20 centimeters of it. If there is, it will switch the car's travel direction. If the car is going backwards, the BiLED will be red.

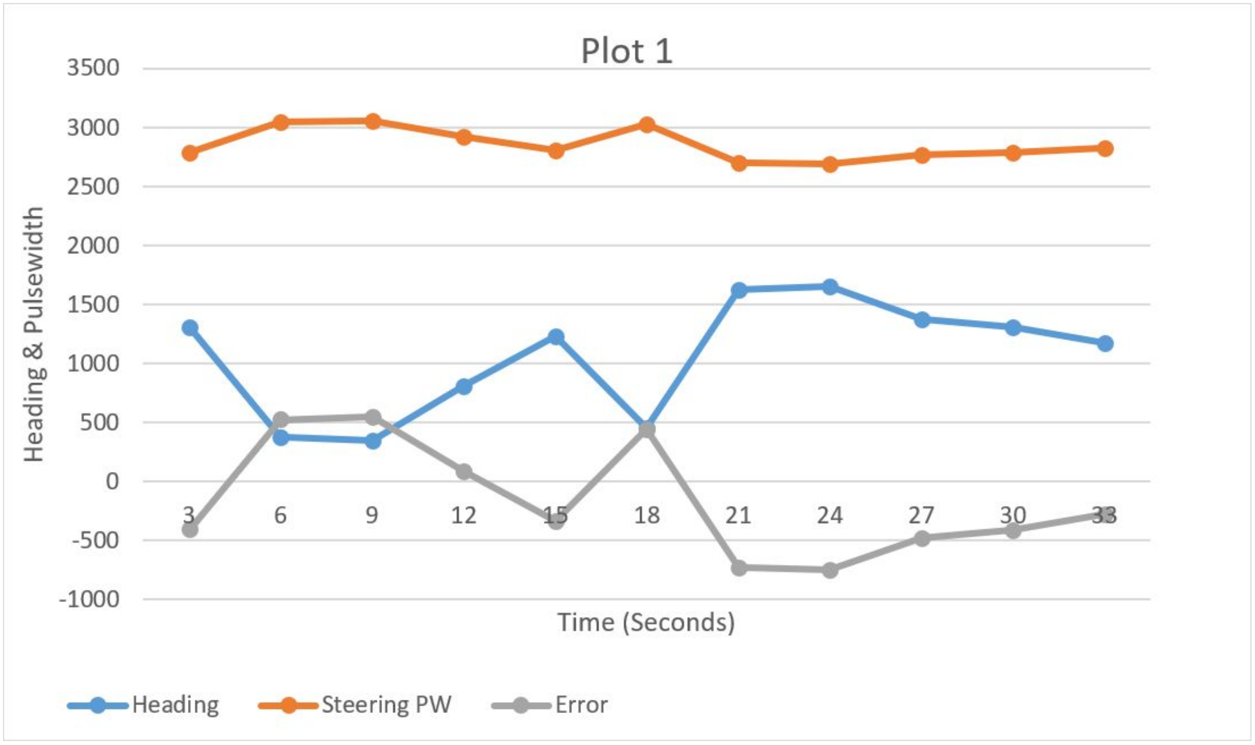
In testing, the team put the car on the ground with the switch off. After successfully inputting a drive gain of 1, a steering gain of 1, and a desired heading of 900 with the keypad, and setting the potentiometer to full way, the team turned the switch on.

The car turned the BiLED green, and drove forward at maximum speed, with the front wheels turning. When the car straightened out, the direction the car was facing was about due east. The car was picked up and turned another direction, and it turned back east. When an object was placed in front of the ranger, within 20 centimeters, the car set the BiLED to red, and drove backwards at full speed, still facing due east.

Subsequent tests set the drive gain to 1, and the potentiometer to half way; this resulted in the car driving at half speed. Setting the gain to 2 and leaving the potentiometer in half way made the car drive at full speed.

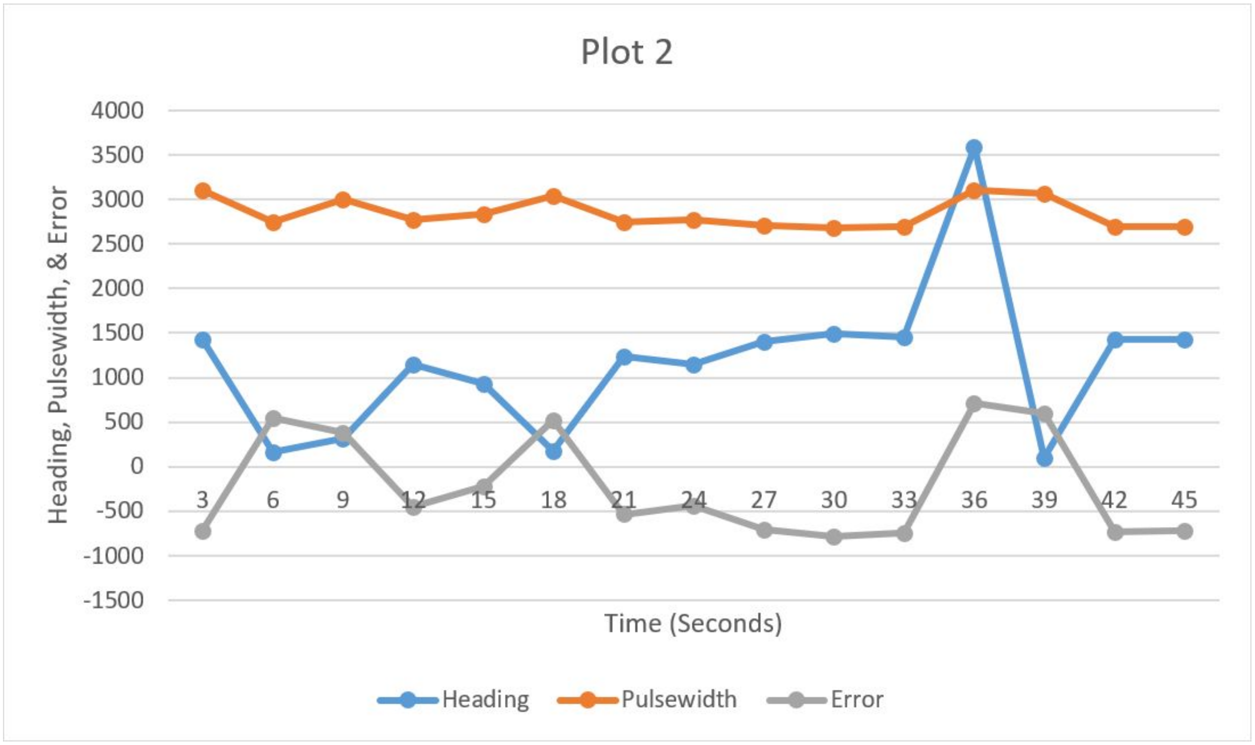
Setting the heading to anything outside of 600 to 1200 prompted the user to enter a number again. Setting the steering gain to 2 made the car turn tighter.

Analysis of plots from tabulated terminal data with explanations



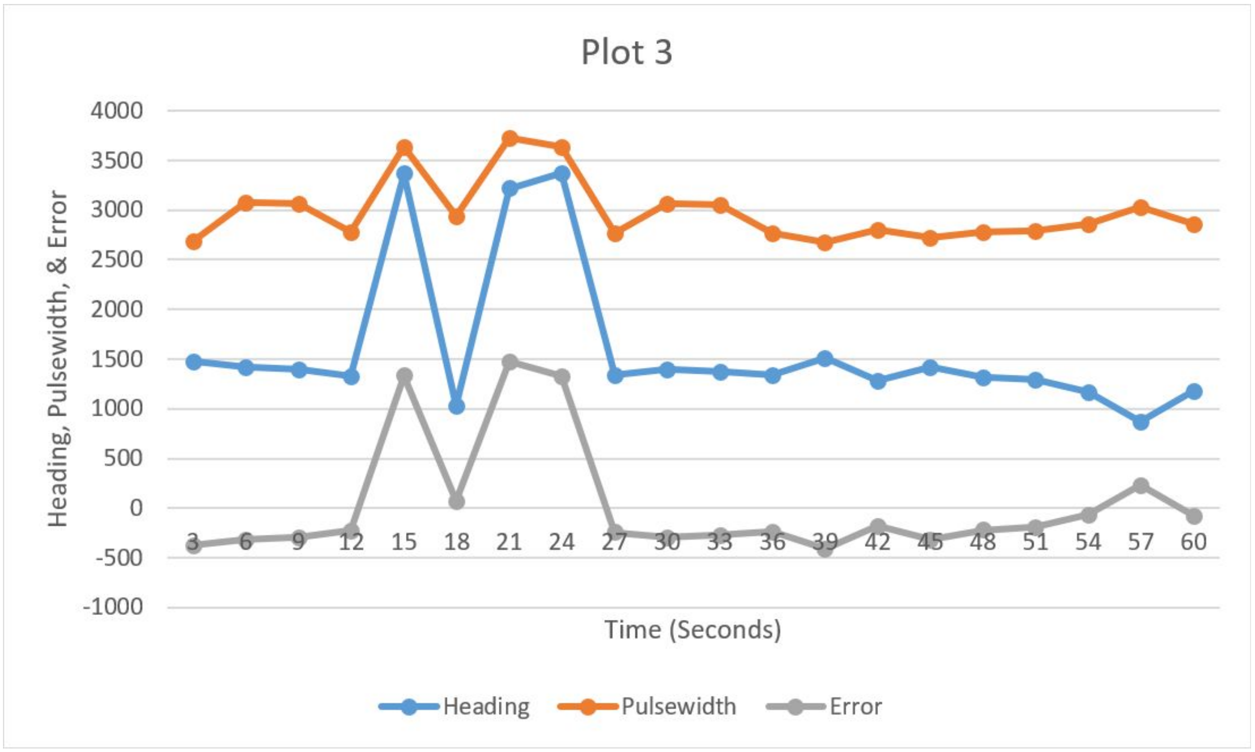
Plot 1: Heading & Steering Pulsewidth vs Time, Steering Gain 1, Drive Gain 1, 50% ADC

In the setup for Plot 1, the steering gain is 1, the drive gain is 1, and the ADC value is approximately 128, or at half setting. The desired input direction was 900, so on the graph when the error is 0 the heading is 900, though due to the system not being perfect the car keeps turning due to system error. In theory, the car keeps changing direction until the heading is 900, and it keeps adjusting the steering direction until it approaches an error of 0. The heading and error graphs are mirror reflections of each other in regards to the x-axis because in the code error is equal to the difference between desired heading and actual heading (desired heading - actual heading). So when the heading is above 900, which is the desired heading, the error will decrease in value and vice versa, which corresponds in the graph to a reflection over the x-axis. The farther the heading is from 900, the greater the error will be. Since the wheels are constaing moving to get into the proper orientation, the pulsewidth value is changing constantly too.



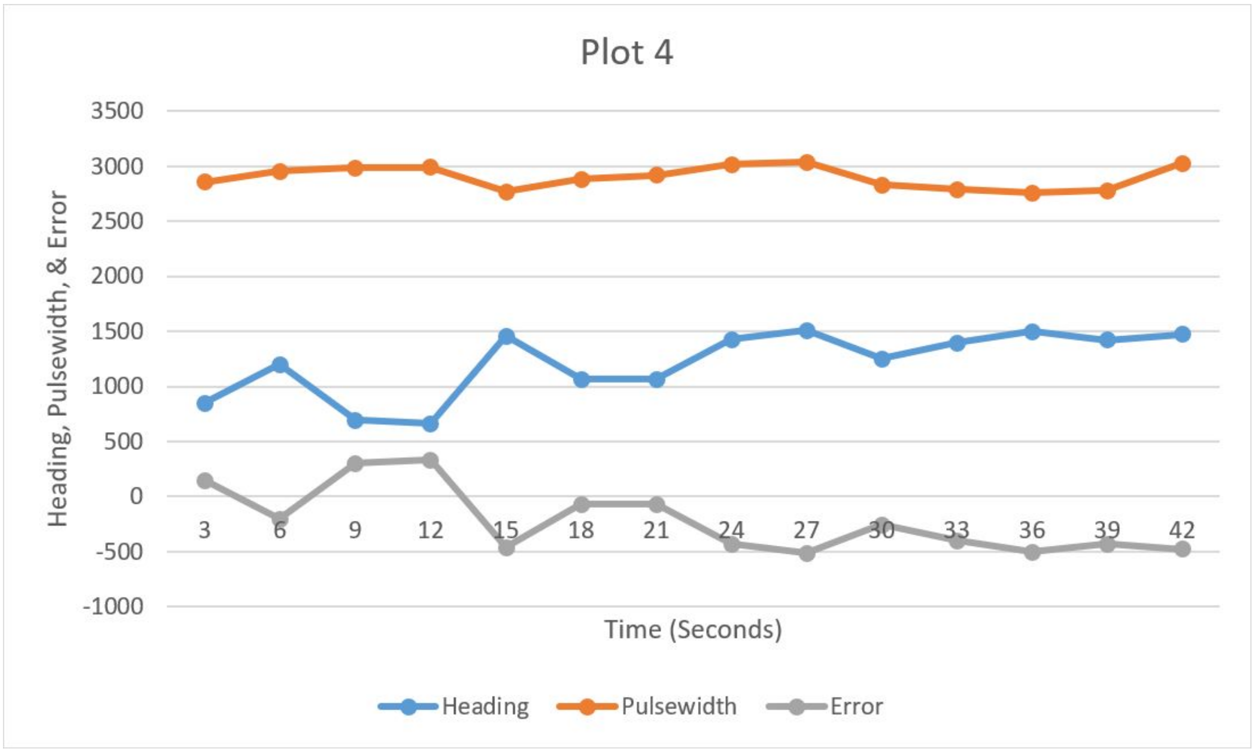
Plot 2: Heading, Steering Pulsewidth & Error vs Time, Steering Gain 1, Drive Gain 1, 100% ADC

Plot 2 is similar to Plot 1 except the target heading is 700 and the Analog to Digital Conversion value increases, from 128 to 255 (max value). The speed is much faster, but the steering does not match the intensity of the drive motor so the car has less control than the previous graph, evident by the sharp peaks in the time interval. The dramatic change in heading at 36 seconds is an example of the car hitting a barrier and needing to be moved, such as when there is another team testing nearby. Overall the performance in this poor compared to the other tests; however, it is notably very different than Plot 4, which it should be similar to.



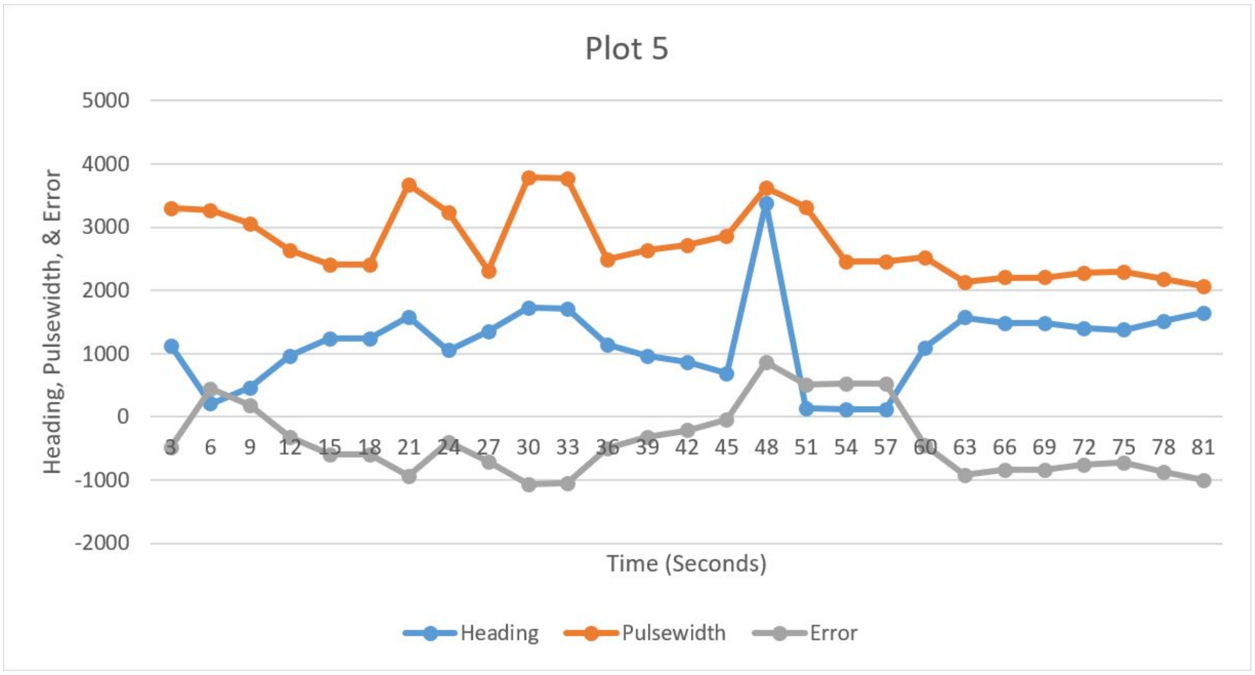
Plot 3: Heading & Steering Pulsewidth vs Time, Steering Gain 2, Drive Gain 1, 100% ADC

Plot 3 describes the same input setup as Plot 2, but with less steering precision by increasing the steering gain. This plot is one of the more successful tests performed because despite two large peaks in heading and error (due to the car impacting an object and being moved by hand), the overall trend in heading approaches 1100, the desired heading. It is possible that the higher gain gave the car more response when the error was large, leading to the accurate trend much faster than a car with a smaller steering gain.



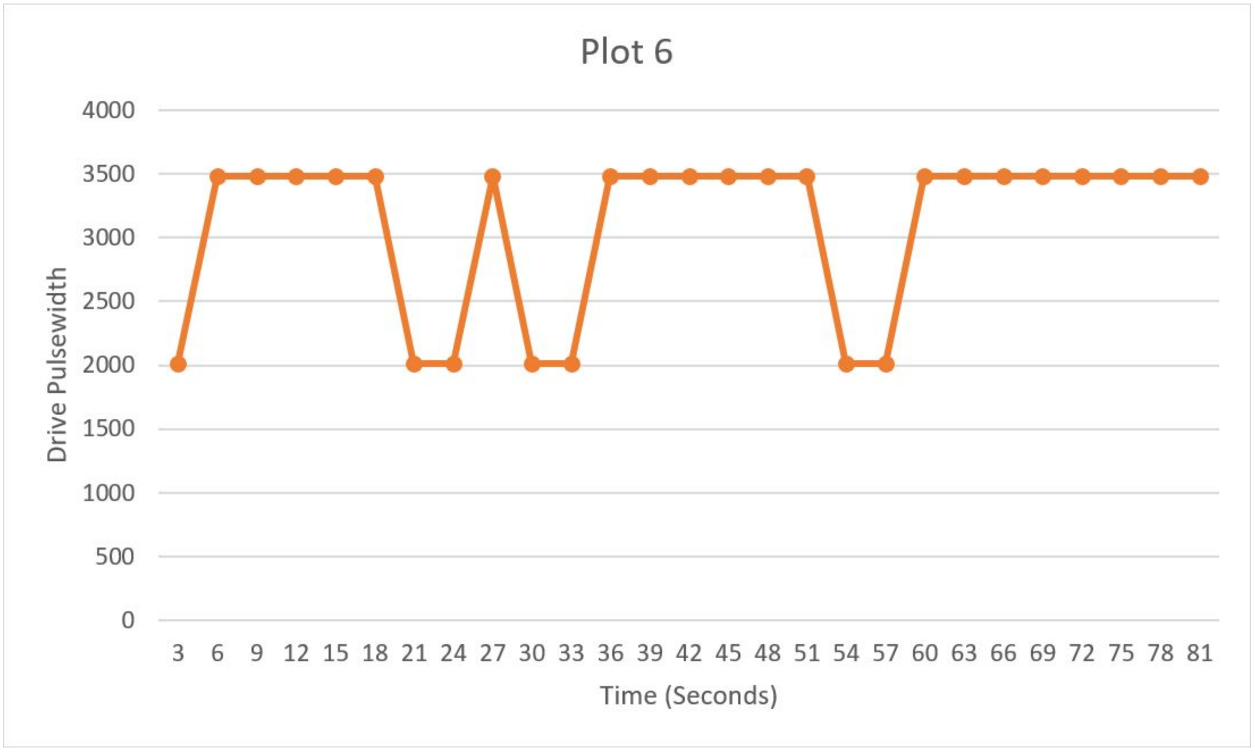
Plot 4: Heading & Steering Pulsewidth vs Time, Steering Gain 1, Drive Gain 2, 50% ADC

Plot 4 is driven by a mostly standard setup; however the drive gain is set to 2 and the potentiometer to 50%, which should be analogous to Plot 2, in which the drive gain is 1 but the potentiometer is 100%. The pulsewidth is relatively consistent throughout the time interval, with the heading leveling out and error decreasing. That said, the heading the car begins to converge to is approximately 1500, which is substantially different than the goal of 1000. This is perhaps due to moving around the room and encountering different magnetic fields, minute fluctuations in steering pulsewidth, or an inaccurate k value. This could also be due to the steering not being quick enough to approach the target heading before running into an obstacle.



Plot 5: Heading & Steering Pulsewidth vs Time, Steering Gain 3, Drive Gain 3, 75% ADC

In the setup for Plot 5, the drive gain and steering gain are set at 3, with a desired heading of 900. Between 3 and 45 seconds, the trend in the heading and pulsewidth is as expected, tending to 900 at ~40 seconds. However, due to the speed of the car in this setup, the car was running into more obstacles than at slower and more predictable gain values, and was more susceptible to the varying magnetic fields of the room. The spikes are due to the car being picked up and put in a more open area to drive, so there are large jumps in error and pulsewidth in these intervals. At the end is another strange convergence that happens outside the target heading, and this could likely be due to any of the previously mentioned reasons, or because the gains are set so high the car cannot properly address the readings.



Plot 6: Drive Pulsewidth, Steering Gain 3, Drive Gain 3, 75% ADC

Plot 6 shows how drive pulsewidth (in cycles) changes over time. Every time the pulsewidth changes, the car is changing direction. The neutral pulsewidth is the average value of 2750; when the car is going forward, the pulsewidth is at the maximum of 3487, and at the minimum of 2013 when backwards. Even though the potentiometer is set to 75%, the drive gain of 3 multiplies the pulsewidth past the maximum or minimum value, and the software limits it to be the maximum or minimum pulsewidth. Thus, the car travels at maximum speed in either direction.

Problems Encountered & Solutions

Many problems were encountered by the team when doing this lab, though by using the resources and people available to them (such as the lab manual, teaching assistants and the professor), they were able to come up with solutions to these problems. While the problems proved to be frustrating at first, the team view these problems as a good experience as they served as good teaching experiences.

One of the first problems the team first encountered is that when the code from Lab 3 was merged, the variables took up more space than the 128 bytes allowed in internal memory, not allowing the code to run properly. The team's variables also had to share space with the variables in the included header files, such as "i2c.h". The team solved this problem by referring to Silicon Laboratories reference sheets, and learned how they could designate some variables with "__xdata" and store the variables in external memory space.

Another problem the team had was coming up with the formula to linearly map the analog to digital conversion value to the drive motor pulsewidth. The difficult part was that the A/D value from the potentiometer controls how fast the car drives, but this applies to both forward and reverse speeds. This means that while an A/D value of 0 maps to a neutral drive pulsewidth of 2750, an A/D value of 255 has to map to both 3487 (forward maximum) and 2013 (backwards maximum), depending on direction. The team solved this problem by having two formulas for drive pulsewidth—one for forwards, and one for backwards.

One unexpected problem that the team encountered was that the steering servo didn't work. The software and hardware connections were checked multiple times, and it was only when a logic probe was used to test if the pulsewidth was being sent to the servo was it found that the buffer chip was burned out. A new chip fixed the problem.

The most time-consuming problem that the team had was keeping track of the ports and pins for each device. The LCD/keypad, the ultrasonic ranger, and the compass all used an SDA pin, and SCL pin, and a 3.3 V pin. In addition, the potentiometer, the slide switch, the BiLED, the drive motor, and the steering servo all had their own pins. The team made errors in initializing the P0MDOUT, P1MDOUT, P1MDIN, P3MDOUT, and crossbar settings, and had to redo the bit masking multiple times, cross-checking the software initializations with the hardware.

Suggested improvements to HW & SW

While the code accomplished all primary goals and functioned adequately, there were several areas in which the implementation could be made more efficient. During the code writing process, there were several changes made in order to make the code more efficient, such as deleting variables or having certain variables be dual purpose. That said, one of the problems encountered later in the lab was outputting to the LCD screen. There were several function calls that, though the syntax was correct, had no effect on the LCD, likely due to hardware limitations of the screen. A more efficient option would have been to create a function that accepts an input, prints the input onto the LCD screen, and clears it, all while waiting for an appropriate amount of time for the screen to accept the input. Another problem area that could have been improved upon is that if the slide switch was left off after the user had gone through all of the input prompts, the LCD would loop back to the first prompt, and would not exit until an input is put in again, even if the switch is then turned on. The way to get around that was to flip the switch on before entering the final input. Instead of calling the LCD initialization code in the main method sequentially, the initialization code could have been put in a different function that would only run the code once per switch, making it easier to use.

Because the hardware was less in our control versus the software, there were fewer things to be done by the team to make it run faster. The protoboard had relatively straight-forward wiring and therefore none of our errors were hardware specific. One interesting problem that occurred was a delay between the ranger reading a distance and a reaction from the drive motor. In software, the delay was coded to be 3 seconds long, however in practice delays sometimes approached 10 seconds long, and consecutive readings would process much faster and yield inconsistent results. While this issue was never ultimately addressed, the hardware functioned well enough for the lab to be completed.

Another hardware problem that the team noticed was that the drive motor had very low torque at low speeds. At speeds under 50% of maximum speed, the motor was barely enough to get the car to move, yet above 50%, the car moves really fast. The gearing in the car's drivetrain itself would need to be changed to fix this problem, or a different motor used.

***Academic Integrity Certification** (this part is required exactly as stated)*

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.

***Participation** (this is only a template; make changes as appropriate or necessary)*

The following individual members of the team were responsible for (give percentages of involvement)

Hardware implementation: (wiring & pin-out sheet)	Daniel Alegria	50 %
	Martin Yu	50 %
Software implementation: (pseudo-code & code)	Ankur Singh	20%
	Daniel Alegria	40%
	Martin Yu	40%
Data analysis (if relevant):	Ankur Singh	30%
	Daniel Alegria	35%
	Martin Yu	35%
Report development & editing*: (schematic, diagrams & plots)	Ankur Singh	33%
	Daniel Alegria	34%
	Martin Yu	33%

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

Ankur Singh

[illegible]

[illegible]

*Note, report development/formatting does not constitute an engineering contribution toward successful laboratory completion.