Introduction:

The goal of Lab 4 was to implement I2C devices both in hardware and software to control the car. Ultimately, Lab 4 was an integration lab, testing our ability to take multiple devices we had effectively used on their own and putting them together into one functional package. In this lab we implemented the ultrasonic ranger, a compass, an LCD display, a 12 button keypad, and an RF serial transceiver, all operating on the same I2C bus. 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 both car speed and direction, allowing the car to drive in a relatively straight line, as well as course correct.

In the process of integrating the multiple existing systems, we had to adjust the initialization functions and variable initializations significantly. We ended up using 3 capture/control modules, namely 0, 2 and 3. We also moved our address variables from global to local, such that each time a device was called, it referenced an address given within that function to ensure the correct device was called. In order to avoid overlapping communication on the I2C bus, we used a series of flags which would update at predetermined intervals (one for the compass, one for the ranger, and one for printing). Whenever one of these flags tripped, that I2C communication would finish before another could begin.

Multiple gains were tested in the development of the system. At high steering gains, the car usually oversteered greatly. We were only using a proportional control, so quick steering actions weren’t countered by a derivative control. The drive gain seemed to have little effect on the system other than making the car slow down much quicker or slower as the range closed. After multiple rounds of testing we settled on 2.5 as a steering gain and 3 as a drive gain.

Conclusion:

Lab 4 demonstrated the value of careful coding in advance. Although we understood during Lab 3 that the pieces of code we would be developing would be integrated together, we didn’t necessarily consciously reflect this in our programming. Therefore, when it came time to integrate 3 separate pieces of code, each primarily written by a different group member, we faced multiple challenges. Variable names referring to different things would be the same, or visa versa. We also had three different PCA init functions to merge. This could have been easily remedied with careful and conscious coding choices in advance.

Aside from this invaluable coding lesson, this lab also showed the power of control systems. At first our car was a halting, barely moving hunk on the table. After a few gain changes, the thing was practically a rocket. Finding the correct gains for the control system was a trial and error process but one which paid out huge dividends as the lab continued.

The final aspect of this lab we struggled with was the compass. We had to calibrate the compass usually once or twice a lab session because it would slowly drift off course. It was a hardware problem which had a clear solution but was nonetheless consistently frustrating.

PSEUDOCODE

Variable/function initializations

Main

Inits

While (infinite)

While slide switch == switched

Set motors to neutral

If (first time in loop)

Get heading from user

Get range from user

Update first time in loop variable

If (heading flag)

Read the compass

Adjust servo PWM

Reset flag

If (range flag)

Read the ranger

If (light is bright enough)

Call reverse mode

Else

Call forward mode

Set motor PWM

Reset flag

If (print flag)

Clear LCD

Print to LCD

Print to serial

Reset flag

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

Forward\_mode function

Set heading to normal heading

Reverse\_mode function

Read potentiometer value

Set motors to maximum allowed reverse

Get error value

Adjust PW with negative kp to steer backwards

Check for PW exceeding PW\_MAX

Set PW

Set\_servo\_PWM function

Get error value

Adjust PW with a positive kp to steer forwards

Check for PW exceeding PW\_MAX

Set PW

Set\_drive\_PWM function

Read potentiometer value

Check for range within 3 intervals

Less than 30

Between 30 and desired\_range (usually 60)

Above 60

Adjust PW with kp

Check for PW exceeding PW\_MAX

Set PW

Pick\_heading function

Assign local variables

While (infinite)

Check for user input in 5 cases

If 1, return 0

If 2, return 900

If 3, return 180

If 4, return 270

If 0

Assign local variable

While (infinite)

Read three numbers (starting at 100’s place)

Add each one together

Return 3 digit number

Pick\_range function

Assign local variables

While (inifinite)

Read two numbers (starting at 10’s place)

Add them together

Return 2 digit number

Read\_AD\_input function

Assign local variable

SFR inits

Wait for conversion to complete

If (cast variable == 0)

Return ADC value

Else

Return ADC variable scaled to some casting gradient

PCA\_ISR function

If (CF)

Increment all four counts

If heading count > 40 ms

Update heading flag

Heading count = 0

If range count > 80 ms

Update range flag

Range count = 0

If print count > 1000 ms

Update print flag

Print count = 0

Pause function

Wait 120 ms

Wait function

Wait 1000 ms