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

Turn BILED off

If (first time in loop)

Get input from user to confirm calibration

Calibrate

Call set\_gains()

Get direction (right or left) from user

Update first time in loop variable

If motor = forward

Biled green

If motor = reverse

Biled red

If motor = stopped

Biled off

If (accel flag)

If gy within +- 10 for 5 cycles

Levelflag = 0

If levelflag != 0

Set steering in user selected direction

Full reverse

If levelflag = 0

Call set\_servo\_PWM

Call set\_drive\_PWM

Reset flag

Reset count

If (print flag)

Clear LCD

Print to LCD

Print to serial

Reset flag

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 both counts

If accel count > 80

Update accel flag

accel count = 0

If print count > 200 ms

Update print flag

Print count = 0

PCA0 offset

Pause function

Wait 120 ms

Wait function

Wait 1000 ms

Calibration function

Declare local variables

Reset averages

For 64 counts

I2c read

Wait for confirmation

Read all 4 bytes

Add data

Average data

Read\_accels function

Declare local variables

Reset averages

For 8 counts

I2c read

Wait for confirmation

Read all 4 bytes

Add data

Average data

Subtract calibration data

Set\_servo\_PWM function

Get error value

Adjust PW with ks and gx

Check for PW exceeding PW\_MAX

Set PW

Set\_drive\_PWM function

Read potentiometer value

Scale PW value according to gy

Scale PW value according to gx value (sideways tilt = forward drive)

Check for PW exceeding PW\_MAX

Set PW

Set\_gains function

Declare local variables

Get user input in 5 cases (using standard infinite while and read\_keypad):

1 = ks of 1

2 = ks of 2

3 = ks of 3

4 = ks of 4

5 = custom ks

Get user custom ks

Get user input in 5 cases (using standard infinite while and read\_keypad):

1 = kdx of 3

2 = kdx of 6

3 = kdx of 9

4 = kdx of 12

5 = custom ks

Get user custom ks