Lab 4: Heading/Ranger Integration, Battery Voltage and LCD Display

Preparation

Reading

Lab Manual

Chapter 4 - The Silicon Labs C8051F020 and the EVB (Analog to Digital Conversion)

Chapter 5 - LCD and Keypad

Chapter 7 - Control Algorithms

LMS, Course Materials

Electric Compass Data Sheet

Ultrasound Ranger Data Sheet

LCD and Keypad Data Sheet

Wireless RF Serial Communication Link

Objectives

General

1. Integration of the compass and ranger systems developed in the previous lab to control the steering. Speed can be fixed or adjusted via the keypad or SecureCRT through the wireless serial link.
2. Use Analog to Digital Conversion as was implemented in Lab 2 to read battery voltage.
3. Display control information on LCD display screen or SecureCRT and enter variables using keypad.

Hardware

1. Wire a single protoboard with both an Ultrasonic Ranger and an Electronic Compass. Only one set of pull-up resistors is needed on the System Management Bus, SMB.
2. Add a Liquid Crystal Display and number pad. These also use the SMB interface.
3. Design a voltage divider circuit to measure the battery voltage.
4. Add a wireless RF serial transceiver.

Software

1. Integrate the C code used in Lab 3 part 3. Write a main function that calls a read_compass function and sets the PWM for the steering servo based on the present heading and a desired compass heading. The main code also calls a ranger function and adjusts the PWM for the servo motor based on the range measurement from the ultrasonic sensor to an obstacle.

2. Write C code to configure the A/D conversion in the C8051, to read the battery voltage. Use your code from Laboratory 2 as a template and choose input on Port 1.
3. Write C code to display settings on the LCD display.
4. Write C code to allow user entry of gain constants and desired heading.
5. Write code to tabulate the data from the compass heading error & servo motor PW value, and transmit it to the SecureCRT terminal for filing and later plotting.

Motivation

The *Smart Car* is being used as a test bed to develop the software for the gondola. This will involve integrating the two subsystems developed by ranger and compass pairs, as well as adding on a LCD and keypad for displaying data and adjusting variable values on the fly.

The simplest blimp control is in "hover mode", where the embedded controller causes the gondola/blimp to maintain a constant altitude and a constant heading. The electronic compass is used to read the heading. It is compared to a desired heading and the control algorithm generates a PWM signal for the tail fan. The gondola uses a speed control module to power the tail fan. That module uses PWM signals that are essentially the same as the steering servo of the car. Therefore code developed to steer the car based on the compass can be ported essentially unaltered to the gondola.

Concurrent with the need to control the heading of the gondola/blimp, is the need to control its altitude. The gondola is propelled by two thrust fans, (these are the same type of fan as the one located on the tail for steering.) The thrust fans have two control features, the angle setting and the power setting. The fans are mounted on a shaft that can be rotated by a servo, very similar to the one that steers the car. The power to the fans can also be controlled. This is done using a speed control module as is in the case of tail fan and the car.

The simplest altitude control is to set the trust fan angle so that the fans are horizontal and the thrust is vertical, and leave it there. In this situation, the altitude can be controlled just by adjusting the power to the fans. This is what will be bench tested using the car. The ultrasonic sensor is aimed up on the car; it is aimed down on the gondola/blimp. If the gondola is near the floor, the fan power should be set to maximum. The analogy with the car is that if an object is close to the car, it will run at maximum power. If the gondola is at the desired height, the thrust fans should be off or running at a relatively slow speed. So for moderately distant objects, the car should stop. If the gondola is too far from the floor, the thrust fans should be reversed, so on the car if the closest object is far away the car should travel in reverse.

The goal of this lab is to integrate the steering and speed control of the car to prepare for the transition to the gondola. It is worth noting that this test mimics a hovering gondola/blimp. If one desired to have forward motion of the gondola/blimp, then the angle of the thrust motors must be set to something other than horizontal. For that situation, the altitude control is then dependent on both the thrust power and thrust angle. This is considerably more complex than the present experiment to develop hover code, so it is left as a future project. It is wise to consider forward motion options, but it isn't part of this lab or course.

In addition to the ranger and compass integration, this lab involves monitoring the vehicle's battery voltage. Both the *Smart Car* and the Gondola are powered by rechargeable batteries that form a part of the unit. As the charge in the battery is being consumed by running the *Smart Car* or the Gondola, the output voltage of the battery slowly drops. Since the ICs in the circuit require a minimum supply voltage, they fail to function properly if the output voltage drops below the

required minimum voltage supply. When the voltage is low, the operation of the processor may become erratic as the motors turn on and off. A symptom of a very low battery voltage is that the microcontroller will shutdown and restart whenever a motor is turned on.

Reading the battery voltage is accomplished by first designing a resistor voltage divider circuit, and then using the C8051 Analog to Digital convertor and the internal reference voltage of 2.4 V to measure the battery voltage. The steps are described in the next section under Lab Description and Activities.

Finally, this lab requires the integration of an LCD screen and keypad and a separate wireless serial communication link. For this lab, the keypad is used to input the desired heading and to set the steering gain constant. The screen is used to display current heading and the ranger reading. In labs 5 and 6 the LCD and keypad will be used as a way to display and set the gain constants for both heading and altitude. This allows these values to be set on-the-fly, rather than recompiling the program each time a gain is changed. Furthermore, the LCD can be used to display useful values such as the heading, range, and battery voltage. The wireless link is used to send data to a SecureCRT terminal while the car is moving so that performance plots can be created. As the car is driven the values of the heading error based on the compass readings and the steering motor PWs based on the feedback gain vs. time will be transmitted to a laptop where the points will be saved in a file and plotted later.

Lab Description and Activities

For this lab and for the rest of the course, the ranger pair and the compass pair will merge to form a team. Each pair of students in the team may still maintain their own lab notebook, however **only a single lab notebook** from a team would be considered during the final submission. The individual pairs will still have separate focus in the lab. It is valid to refer to notes on Labs 1, 2 or 3 in the notebook of the other pair in a team. However for Labs 4, 5 & 6 all the new notes should be kept in a **single notebook with all four team member names on the cover**.

The hardware and software from Lab 3 pairs shall be merged. Only one protoboard shall be used, which requires moving components from one board to the other. It is up to the team to determine which board to use.

The team must be careful when merging the software to make sure that initialization functions and variable usage is consistent with the task. The initialization of the PCA is one area where changes may be required. Both sensors use the SMBus, which is valid. Many slaves can be on one SMBus, and both sensors functions as slaves.

The integrated software and hardware will result in a car that can follow a compass heading but have the steering modified by the distance to an obstacle in front of the car. The integrated software shall poll the run/stop switch(es) connected to P3.6 (and/or P3.7) to start and stop any one or both control functions. There will be similar switches on the *Gondola*. The details are left up to the team.

The LCD/keypad should mount on the back of the car as shown in the lecture slides. However, if this can't be made to work there is an option to add Velcro to the protoboard to hold the LCD/keypad. The board has a 4 wire cable that connects power, ground, SDA and SCL. The LCD board must be returned each class and not kept with your protoboard. There are not enough for each team to keep their own. Also, remember to keep the 4-pin adapter with the connector on the ribbon cable and **NOT IN YOUR** protoboard when returning it at the end of class or open shop.

Hardware

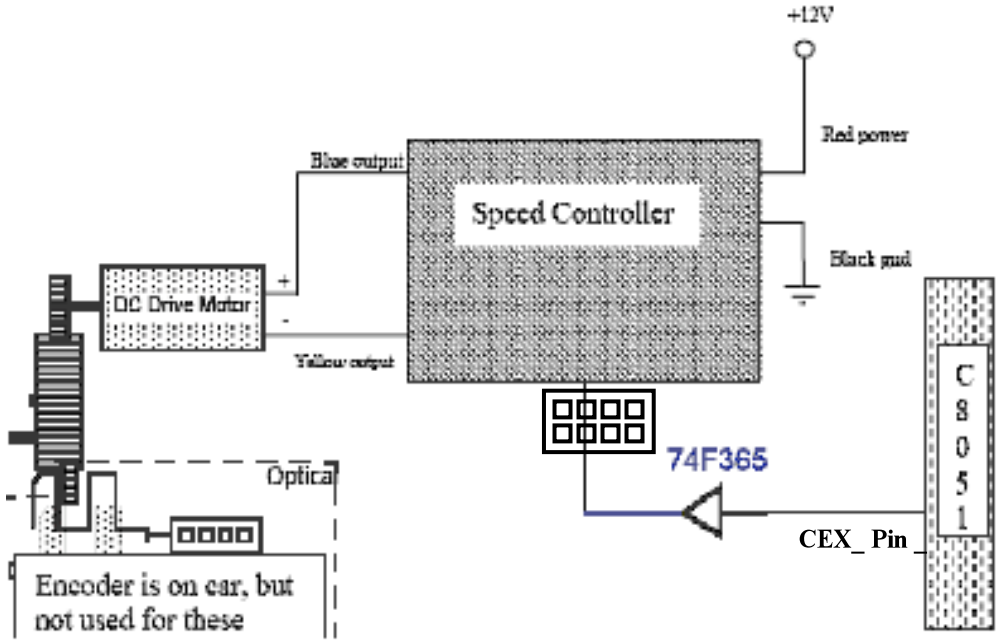

Figure 4.1 - Car drive-motor circuitry from Lab 3 (part 1)

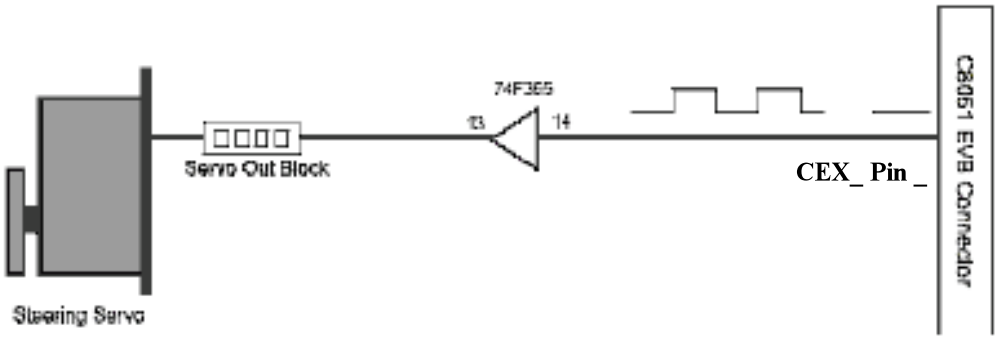

Figure 4.2 - Car steering servo motor control circuitry from Lab 3 (part 1)

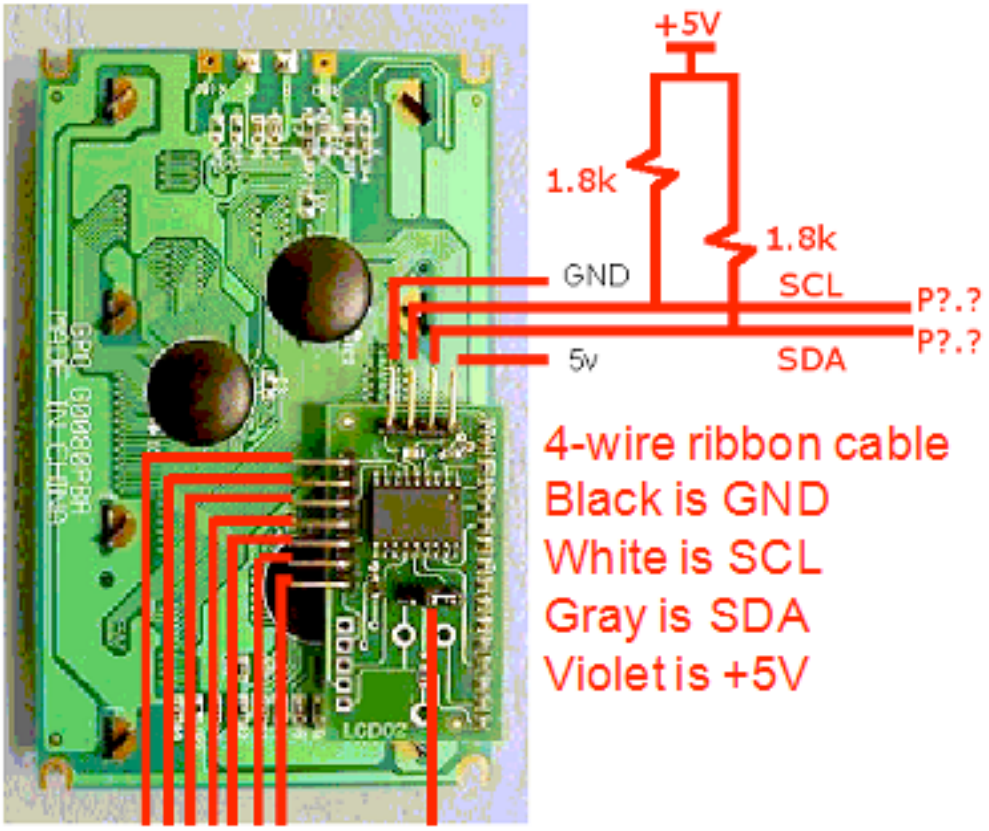

Figure 4.3 - Circuitry for LCD panel

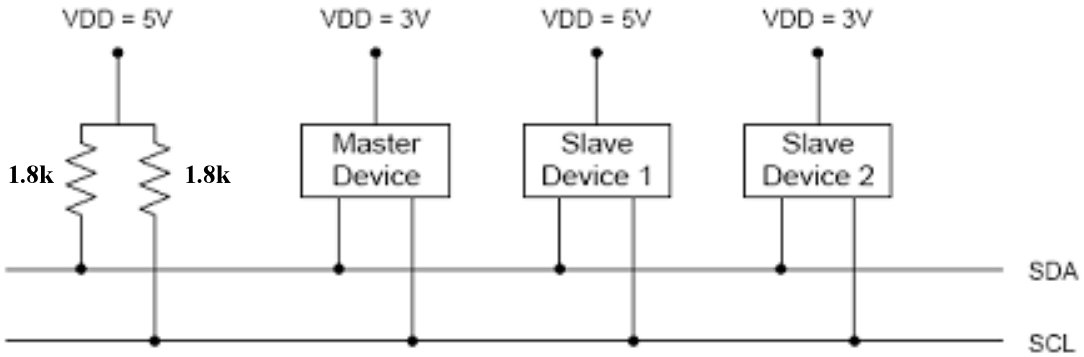

Figure 4.4 - Configuring multiple slaves on a single master

Note: Additional hardware – a run/stop slide switch connected to any unused I/O pin of the team's choice still be used. We recommend either P3.6 or P3.7, as these pins will also have switches on the gondolas. This allows disabling the drive motor on the car so that adjustments can be made with the program running without requiring that the car be placed on a foam block.

In order to monitor the battery voltage, a resistor voltage divider circuit must be designed so that the output voltage is not more than 2.4 V, when the input voltage is 15 V. Note that we only have standard resistor values, so you will then have to adjust to the available resistors.

1. Use the following schematic diagram to decide the values of R1 and R2 in the voltage divider circuit.

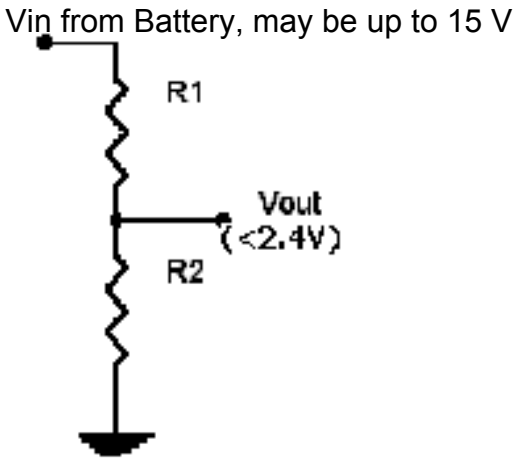

Figure 4.5 - Power supply voltage divider schematic

We need $v_{out} < 2.4$ V when $v_{in} = 15$ V.

We also want current to be 0.5 mA < i < 2 mA.

Therefore, pick R1 and R2 to meet these requirements.

$$Vout = R2 \cdot i = \frac{R2}{R1 + R2} Vin$$

$$(Vin - Vout) = R1 \cdot i = \frac{R1}{R1 + R2} Vin$$

Note that only standard resistor values are available so values must be adjusted to use resistors stocked in the studio. Hint: 5 k ohm < R1 < 25 k ohm, 1 k ohm < R2 < 5 k ohm

As a result of too many hardware failures from incorrect wiring to the nominal +12V battery voltage, a fixed resistor R1 = 10 k has been prewired in the Figure 4.5 divider and on the car. The only point available to students is Vout, however it is critical that R2 is added connecting Vout to ground before connecting Vout to the analog input pin.

2. Mount the resistor R2 on your protoboard with one end grounded and connect Vout from the block on the car to the other end of R2. Note that the actual voltage measured at Vin might be greater than 12 V, but we are safe because we are designing our circuit for a 15 V supply!
3. Use a voltmeter to prove that the output of the voltage divider is 2.4 V or less. Have a TA confirm this before you continue. Voltage in excess of 5 V might cause the analog converter on Port 1 to fail.

4. Only after a TA has seen a voltmeter check of the divider output voltage will you be permitted to connect the midpoint voltage to a pin of Port 1. Remember to use pins not associated with the PCA outputs, so use Port 1 pins 4, 5, 6 or 7.
5. Write code to do an ADC conversion on that pin and print the results and add this to your Lab 4 program. Print out the battery voltage about every second.

Car RF Transceiver Module

The radio frequency (RF) transceiver module on the car communicates with the matching USB RF transceiver module connected to your laptop. This is used to establish a serial connection in the same way the wired serial cable was used, which allows data to be transferred between the car and your laptop. With this, commands from your laptop are sent to the car and output from the car sent to your laptop through your SecureCRT terminal.

The car RF transceiver module requires 5 V power (pin 4, black wire) and ground (pin 2, orange wire) lines. Pin 5 (white wire) is also grounded. To send and receive data, it also requires connections to TX (pin 1, brown wire) and RX (pin 3, green wire), which connect to P0.0 and P0.1 on the EVB respectively. Make sure the 10-pin header is attached to the module as shown in the photo, with the brown wire closest to the side with the 4 jumper pins used to set the baud rate. The wired RS-232/USB and the wireless RF **CANNOT** both be used simultaneously. There will be conflicts. If the wired connection is used the connections to P0.0 and P0.1 on the EVB bus must be temporarily pulled out. If the wireless is used the RS-232 DB9 plug must be disconnected from the EVB.

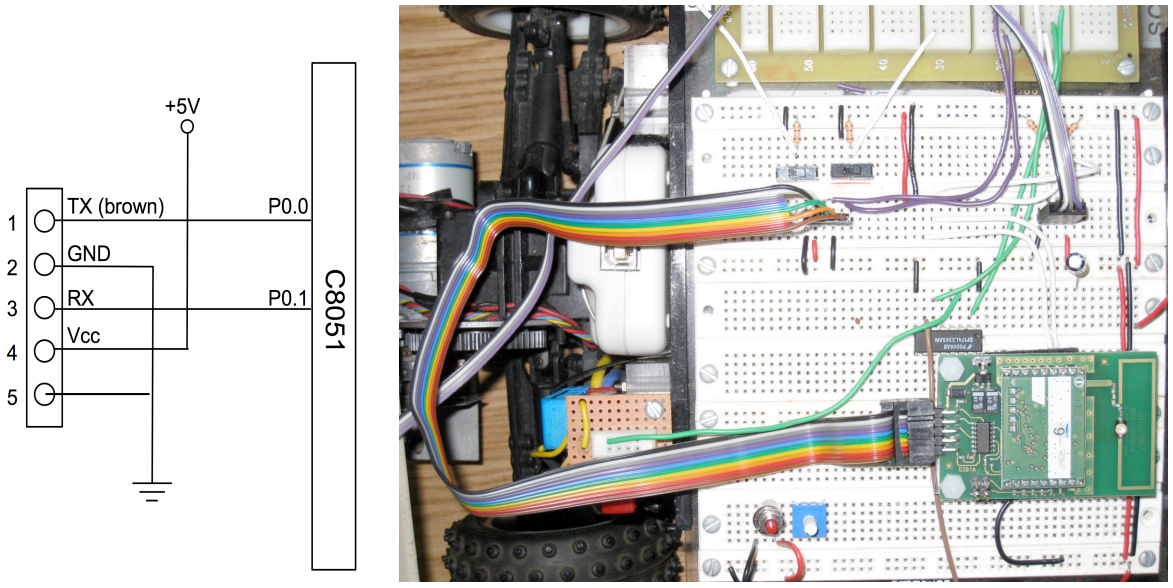

Figure 5.6 – Connections for RF transceiver and photo of module.

Software

Write code to use the LCD/keypad, SecureCRT (with printf() & putchar(), …), to do A/D conversion and print the results and control the steering & drive motors based on compass & ranger data.

Combine the code from Lab 3 with the following constraints:

1. Include a run/stop slide switch.
2. The desired heading must be selectable. This will be done by pressing keys on the keypad. The system should allow the operated to enter a specific desired angle, or pick from a predefined list of 0°, 90°, 180°, 270°. If the user selects a specific desired angle, the LCD screen should prompt for a specific value. If the user selects to choose an angle from a predefined list, the LCD should show the list along with the key needed to press to select that angle.
3. The steering gain must also be selectable. As with the heading, the entry is done using the keypad. Entry options include: 1) select from a menu, 2) increment or decrement using key strokes, or 3) key in a value. (Please only pick one option, not all.)
4. Once the desired heading is selected, the LCD should display the current heading, the current range and the battery voltage. Updating the display every 400 ms is reasonable, but not faster.

There are several other considerations:

1. The steering servo and the speed controller must be updated every 20 ms. The PCA hardware does this automatically as long as it is properly configured.
2. The Compass updates every 33.3 ms, 30 times a second. The gondola controller will work best if each compass reading is a new value, so continue to update the heading every second PCA ISR, or every 40 ms.
3. The ranger needs 65 ms to complete the ping. Continue to use the PCA ISR to set a flag to read the ranger once every 4^{th} ISR, or once every 80 ms. In some cases 100 ms works better.
4. The keypad can be queried for input when appropriate. This shouldn't be done faster than once every 40 ms. When outputting values to the LCD display, do not update them more frequently than 2 or 3 times per second. You may want to create a 400 ms global counter flag that is used to indicate when to update the display.
5. It is necessary to try several gain constants for the steering gain. The car should attempt to get to the desired heading in a short distance without jerky steering adjustments.
6. What happens if the car goes in reverse? You do not need to allow for this but you should at least recognize that there might be a problem with the steering control.
7. The condition where maximum error in the heading or ranger corresponds to a maximum pulsewidth should be reevaluated to produce a faster response in the system. This can be accomplished by increasing the gain coefficient, however, care should be taken so that the maximum and minimum allowable pulsewidths are not exceeded.
8. After an initial estimate of an appropriate gain, code can be implemented to allow the user to change the gain term upon execution rather than recompiling the software. Before entering the infinite loop, the program asks to manipulate the proportional gain of the steering. Use the following function and call it from the main function. The function below uses SecureCRT, ***but your code may use the keypad and SecureCRT***. See the following C function as an example on how to use the SecureCRT keyboard to enter values.

```
int Update_Value(int Constant, unsigned char incr, int maxval, int minval)
{
   int deflt;
   char input;

// 'c' - default, 'i' - increment, 'd' - decrement, 'u' - update and return
   deflt = Constant;
   while(1)
   {
      input = getchar();
      if (input == 'c') Constant = deflt;
      if (input == 'i')
      {
         Constant += incr;
         if (Constant > maxval) Constant = maxval;
      }
      if (input == 'd')
      {
         Constant -= incr;
         if (Constant < minval) Constant = minval;
      }
      if (input == 'u') return Constant;
   }
}
```

9. A new feature to avoid obstacles has been added. The ranger will be mounted with a right-angled bracket so that it is forward looking, and propped up to point up slightly at a ~30° angle. This prevents reflections off the ground from causing problems. In this position the ranger can be used to detect objects in the path and steer around them to the left (or right) if a correct control algorithm is used. Note: speed is **NOT** controlled by the ranger. It can be set to a fixed value or adjusted using keyboard or keypad entries, similar to what was used in Lab 3. As with the heading error (determined by the difference between the desired and actual heading) that is multiplied by a gain constant to determine how strongly or weakly the steering is adjusted, a gain on the ranger values will be used to set how sharply the car will turn depending on how close the obstacle is.

   The ranger should ignore anything more than 55 to 60 cm away since spurious reflections make measurements on anything farther away effectively useless without modifications to the ranger firmware. If the wheels are already turned to their maximum angle to the left and the ranger detects an obstacle within ~20 cm the car should halt to avoid a collision. Within ~12 cm it should halt regardless of the steering angle (required when obstacle avoidance is disabled). If halting is due to a momentary obstacle, travel should continue when the obstacle is removed. You will need to create your own code for the drive motor halt conditions. The steering controller function should have the form (`range_gain` is negative):

```
// range is the value from the ultrasonic ranger
   if (range > MAX_RANGE) range_adj = 0;   //no obstacle in range, no change
   else range_adj = (int)(range_gain * (MAX_RANGE - range));  //find adjustment
//a positive range_gain steers to the right; negative range_gain to the left
...
// compass_adj is the compass heading error multiplied by its error gain
   SERVO_PW = PW_CENTER + compass_adj + range_adj;  //use both to adjust steering
//make sure SERVO_PW is within the limits of the steering servo motor
```

Data Acquisition

When your code is functioning correctly, in addition to drawing a line on the paper, gather data to plot response curves for your steering control. You should plot heading data, range data, and steering motor pulsewidths vs. time (x-coordinate). In order to save the data, you will need to printf() the values to the SecureCRT screen with the wireless serial link and then copy the output to a plotting utility, such as Excel or MATLAB. Read the **Terminal Emulator Program** section in the **Installing_SiLabs-SDCC-Drivers** manual on LMS for more details. You need to obtain curves for 3 different gains (low, medium & high). You must find an 'optimal' setting, but you should also look at other values to verify trends. Plot the data as a scatter plot or straight-line scatter plot (in Excel). Make sure the axes show units: seconds or ms on the x-axis, and degrees (error), cm. and percent (pulsewidth) on the y-axis. Scale pulsewidth (subtract out the calibrated center PW value for your car) so that straight is 0% and normalize the values so that full left is -100%, and full right is +100%. Read the **Lab4-Results_Memo** under **Lab 4** on LMS for details of what data to collect for plotting and analysis.

Lab Check-Off: Demonstration and Verification

1. Show the calculations used to determine the resistor values, R1 and R2.
2. Demonstrate how a desired heading and the neutral thrust value are initialized. As stated the heading must be user selectable. A valid option is that the user may only select from a list.
3. Set the desired heading and steering gain using the keypad or keyboard.
4. Place car on floor. Slide switch to run. The ranger also adjusts car steering. If car is heading in desired direction in-range value will force steering to left. For no obstacle, car will turn to the desired heading and maintain desired heading. The steering shouldn't be jerky and should attempt to achieve desired heading in a short distance of travel.
5. Car will turn in the direction that results in reaching the desired heading the quickest. For example, if desired heading is 90 degrees and the car is started with an actual heading of 350, it will turn right to achieve the desired heading. If the actual heading is 190, it will turn left. Both assume no obstacle in range.
6. Use the paper on the classroom floor, placing the car at the starting point. Set the car to head as specified and trace the path the car follows using a felt pen. If an obstacle (trashcan or wall) is in the path the car should steer around it to the left and then resume its desired course as much as possible.
7. Your TA may ask you to explain how sections of the C code or circuitry you developed for this exercise works. To do this, you will need to understand the entire system.
8. Display the heading, range, servo motor %, and battery voltage on the LCD screen.

Writing Assignment – Lab Notebook

Enter full schematics of the combined circuit. Print and attach the full code. Both pairs of students are responsible for both the schematics and the code. Both pairs must keep copies of the code. In the lab notebook describe what had to be changed to allow the code to be merged. Initialization functions are of particular note. Include a discussion of how the code meets the required timing of the sensors.

Several compass and ranger gains must be tried. Make comments about the car performance with both high and low gains. Determine a usable set of gains for your car.

Show the calculations used to determine the resistor values. Show the calculation of the expected divide ratio, Vout/Vin based on the ideal resistor values. List the multimeter reading of the battery voltage and the divider voltage, and calculate the actual divide ratio based on the actual resistor values. A measurement of the battery voltage, Vin, may be obtained from the Vout connection point as long as no other connection, other than the voltmeter, is attached to that point. The R1 resistor may be ignored for this voltage reading (since the current through it and into the meter is essentially zero). **This means that the connection to the analog input on P1 and R2 must be temporarily removed during the measurement. Remember to reattach R2 and the P1 connection after measuring the battery voltage.** Calculate the percentage disagreement between the calculations and the actual experiment. Remember that we have 5% resistors, a 1kΩ will measure anywhere from 0.95kΩ to 1.05kΩ. Comment on if the actual divide ratio is consistent with calculated when the resistor tolerance is taken into account. Calculate a conversion factor for taking the A/D 8 bit result (Vout) and the equation to find the battery voltage (Vin) from the measured Vout.

Writing Assignment – Results Memo

A brief 2-page written (plus plots, pseudocode, and C program-listing) memo is required for this lab. Submitting a description here allows the final report to concentrate on the accelerometer and gondola lab details. The memo must have the five response plots for the 3 values (~0.2, ~2, & ~8) of the proportional feedback compass error gain when the ranger gain is 0 (no obstacle avoidance) and 2 values for the ranger gain (around -30 & around -60) when the compass gain is ~2 (obstacle avoidance active). Plot the compass value, the ranger value, and the steering pulsewidth on the same time axis. Values will need to be scaled to plot nicely. As a suggestion compass values should be between ±180°, ranger values clipped at 60 cm (to disregard noisy data) and steering pulswidths scaled to ±100% of their maximum allowed values (+100% for fully right, 0% for straight, and -100% for fully left). Optional plots may include the steering error and the adjustments (gain x error) resulting from the compass and ranger values. Analysis of the plots should explain what is happening and why. Significant features on the plots should also be noted and explained. Include a discussion of how the code performs the desired control by adjusting the steering to correct the heading error. Finally a complete commented listing of the C code must be attached. The program code (and **only the code**) should be printed with a **single spaced Courier font, 10 points (or less), left justified, with proper indenting. Ensure that the single line spacing has no (0) pts before or after each line.** If you don't know how to do this ask someone. **You will lose points for improper formatting**. With correct formatting you should be able to get more than 50 lines of code on a page with a 10-point font!

Grading – Preparations and Checkoff

Prior to the starting the laboratory you must complete

1) The appropriate Worksheets (Worksheet 8).

2) The Pin-out form (Port, Interrupt, XBR0, PCA, SMB initializations)

3) The Pseudocode (Revision when finished)

When you are ready to be checked off, the TAs will be looking at the following items:

4) All indicated project requirements are performed correctly (defined above in ***Lab Description & Activities***)

5) Appropriately formatted and commented source code

6) Clean and neat hardware, with appropriate use of colors for source and ground connections

Additionally, you will be asked a number of questions. The questions will cover topics such as

7) Understanding algorithms in the code, identifying locations in the software that perform specific actions, understanding the hardware components, understanding the test equipment

The final item that will be included in the Laboratory grade is your

8) Notebook.

The above 8 items each have an individual contribution to your Laboratory grade.

Sample C Code for Lab 4

The following code provides an example of the main function for combining control of both the steering and the drive motor. You should use the code as a guide when developing the merged code for Laboratory 4. Items to note are:

- Initialization routines are declared, but the routines should be taken from your previous code
- Pulse width variables are declared globally for both the steering servo and the drive motor
- Flags to read the electronic compass, the ultrasonic ranger, and the keypad are declared and set in the PCA Interrupt Service Routine
- Functions to set the PCA output pulses are called, but not defined. They should be taken from your previous code.
- Functions to read the sensors are called, but not defined. They should be taken from your previous code.

```
/* Sample code for main function to read the compass and ranger */
#include <c8051_SDCC.h>
#include <stdlib.h>// needed for abs function
#include <stdio.h>
#include <i2c.h>

//-----------------------------------------------------------------------------
// 8051 Initialization Functions
//-----------------------------------------------------------------------------
void Port_Init(void);
void PCA_Init(void);
void SMB_Init(void);
void ADC_Init(void);
void Interrupt_Init(void);
void PCA_ISR(void) __interrupt 9;
int  read_compass(void);
void set_servo_PWM(void);
int  read_ranger(void);    // new feature - read value, and then start a new ping
void set_drive_PWM(void);
int  pick_heading(void);   // function which allow operator to pick desired heading

//define global variables
unsigned int PW_CENTER = _____;
unsigned int PW_RIGHT = _____;
unsigned int PW_LEFT = _____;
unsigned int SERVO_PW = _____;
unsigned int SERVO_MAX = _____;
unsigned int SERVO_MIN = _____;

unsigned char new_heading = 0;  // flag for count of compass timing
unsigned char new_range = 0;    // flag for count of ranger timing
unsigned int  heading;
unsigned int  range;
int compass_adj = 0;            // correction value from compass
int range_adj = 0;              // correction value from ranger
unsigned char r_count;          // overflow count for range
unsigned char h_count;          // overflow count for heading

__sbit __at ____ RUN;

//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
void main(void)
{
    unsigned char run_stop;     // define local variables
    Sys_Init();                 // initialize board
    Port_Init();
    PCA_Init();
    SMB_Init
    r_count = 0;
    h_count = 0;
    while (1)
    {
        run_stop = 0;
        while (!RUN)            // make RUN an sbit for the run/stop switch
        {                       // stay in loop until switch is in run position
            if (run_stop == 0)
            {
                desired_heading = pick_heading();
                desired_range = pick_range();
                run_stop = 1:   // only try to update desired heading once
            }
        }
```

```
        if (new_heading)        // enough overflows for a new heading
        {
            heading = read_compass();
            set_servo_PWM();    // if new data, adjust servo PWM for compass & ranger
            new_heading = 0;
            h_count = 0;
        }
        if (new_range)          // enough overflow for a new range
        {
            range = read_ranger();  // get range
            // read_range must start a new ping after a read
            set_range_adj();    // if new data, set value to adjust steering PWM
            new_range = 0;
            r_count = 0;
        }
    }
}

//-----------------------------------------------------------------------------
// PCA_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for Programmable Counter Array Overflow Interrupt
//
void PCA_ISR(void) __interrupt 9
{
    if (CF)
    {
        CF = 0; // clear overflow indicator
        h_count++;
        if (h_count>=2)
        {
            new_heading=1;
            h_count = 0;
        }
        r_count++;
        if (r_count>=4)
        {
            new_range = 1;
            r_count = 0;
        }
        PCA0 = PCA_start;
    }
    // handle other PCA interrupt sources
    PCA0CN &= 0xC0;
}

//-----------------------------------------------------------------------------
// All other routines ...
//-----------------------------------------------------------------------------
//
```