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 and driving. SecureCRT through the wireless serial link will be used to collect data and set some options in the program.
2. Use Analog to Digital Conversion as was implemented in Lab 2 to set the steering proportional feedback gain (optional: add a voltage divider to measure battery voltage).
3. Display control information on LCD display screen or SecureCRT and enter variables using either the keypad or the terminal keyboard.

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 SMB (System Management Bus).
2. Add a Liquid Crystal Display and number keypad. These also use the SMB interface.
3. Design a voltage divider circuit to measure a potentiometer input for adjusting the proportional steering feedback gain from 0.0 to 10.2 (ADC Result/25.0).
4. Add a wireless RF serial transceiver.
5. Add a slide switch (P3.7) that can disable (set to neutral) the drive motor and the steering servomotor so these can be turned off while making adjustments to the car while the program is still running.

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 desired heading and/or drive motor based on SecureCRT inputs when the measurement from the ultrasonic sensor to an obstacle is less than a set value.
2. Write C code to configure the A/D converter in the C8051 to read a potentiometer voltage (and optionally the battery voltage through a voltage divider). Use your code from Laboratory 2 as a template and choose inputs on Port 1.
3. Write C code to display relevant values or messages on the LCD display.
4. Write C code to allow user to enter initial speed, desired heading, and other parameters.
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 for the ranger and compass, as well as adding on an LCD and keypad for displaying data and adjusting variable values on the fly.

Historically the simplest gondola control is "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 drive motor of the car. Therefore code developed to steer the car based on the compass can be ported largely unaltered to the gondola.

Instead of developing 2 separate control loops that traditionally were used on the gondolas, the emphasis will now be to use the combined sensor information to control both the steering and stopping of the car. Here, the goal is to drive the car in the direction of the desired heading until an obstacle is detected. Once detected, the car should stop before the obstacle and wait for input from the terminal. An 'L' or 'R' (also 'l' or 'r') will indicate that the car should turn hard left or right respectively and start driving forward again. While driving around the obstacle the keyboard input will be checked. When a space ' ' is received the car will resume following its desired heading until another obstacle is detected. The car should stop within a few cm of the second obstacle without touching it. Once stopped at the second obstacle, the car should remain stopped until the program is restarted (power switch toggled).

As in the Game Lab, a potentiometer will be used to set a value. Here the value will be the K_p proportional feedback gain for the steering control loop. As the potentiometer goes from a minimum to maximum voltage the value of the gain should go from 0.0 to 10.0 as a float variable. Simply dividing the ADC Result by 25.0 will set the gain properly. The potentiometer must be set up as it was in the Lab 2 hardware with a series resistor between +5V and the top lead of the potentiometer, and the bottom lead of the potentiometer connected to ground. The wiper is again connected to the analog input channel.

First build the potentiometer circuit, and then use the C8051 Analog to Digital convertor and the internal reference voltage of 2.4 V to determine the voltage. The steps are described in the next section under ***Lab Description and Activities***. K_p needs to be set only once at the beginning.

Optionally, in addition to the potentiometer circuit, you may try to monitor 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 shut down and restart whenever a motor is turned on.

Finally, this lab requires the integration of an LCD screen and keypad and a separate wireless serial communication link. For this lab, the keypad or SecureCRT keyboard are used to input the desired heading and initial speed. 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 various gains and constants (i.e. heading and range sensitivity). This allows these values to be set on-the-fly, rather than recompiling the program each time a gain is changed. Furthermore, the LCD should 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 data file and plotted later (using Excel or MATLAB).

Lab Description and Activities

For this lab and for the rest of the course, the team of 3 students is expected to do some parallel development of the required functions needed to complete the labs using the 2 assigned protoboards with simple test circuits. Each team still maintains their lab notebook for submission as usual. The individual members will have separate assigned tasks for the lab. It is valid to refer to notes on Labs 1, 2 or 3 in the notebook.

The hardware and software from Lab 3 should be merged if not done so already. Only one protoboard can be used in the final lab check-off, which may require moving components from one board to the other. It is up to the team to determine which board to use.

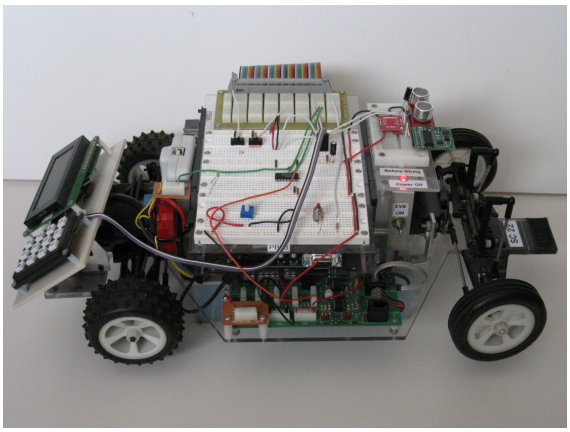
The team must be careful when merging the software to make sure that initialization functions and variable usage is consistent with the task. The initialization of the PCA is one area where changes may be required. Both sensors use the SMBus, which is valid. Many slaves can be on one SMBus, and both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

The integrated software and hardware will result in a car that will follow a compass heading (west or whatever is marked on the paper) at a fixed initial speed until it detects an obstacle. The car should stop approximately 30 to 60 cm before the obstacle and wait for a keyboard entry. A SecureCRT terminal should be connected to the car through a wireless serial transceiver. An 'L' or 'R' (also 'l' or 'r') input will be used to turn the car left or right and start driving around the obstacle (temporarily ignoring the ranger and compass data. While steering around the obstacle the program should check the keyboard input using the function `getchar_nw()`, which returns a value of 0xFF if no key has been pressed. If a space (' ') is received, the car should then resume following its

previous desired heading using its fixed speed until another obstacle is found, at which point the car should stop a few cm from this obstacle and shut down (until the power switch is toggled).

While running, the integrated software must also continuously poll the run/stop slide switch connected to P3.7 to turn on/off (set to neutral) both the drive and servomotors. There will be similar switches on the *Gondola*. The details are left up to the team. The drive speed should be set to a constant while not interacting with obstacles. You may choose to have some values entered by the keypad or SecureCRT keyboard, but the obstacle avoidance maneuver must be done with SecureCRT.

The LCD/keypad should mount on the back of the car as shown below. 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 left on the car or returned each class and not kept with your protoboard. There are not enough for each team to keep their own.



Hardware

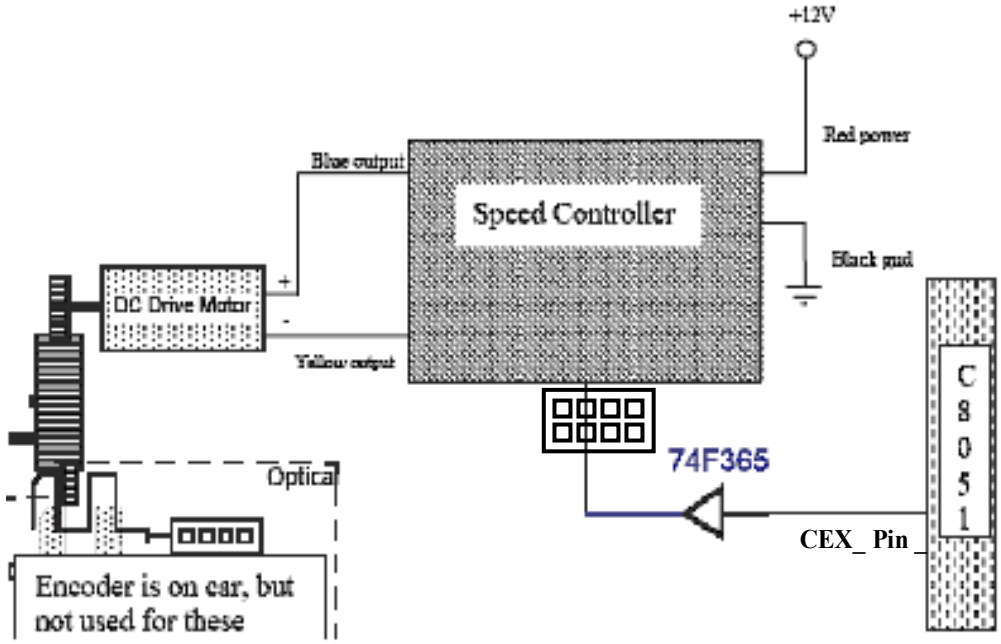


Figure 4.1 - Car drive-motor circuitry from Lab 3 (part 1)

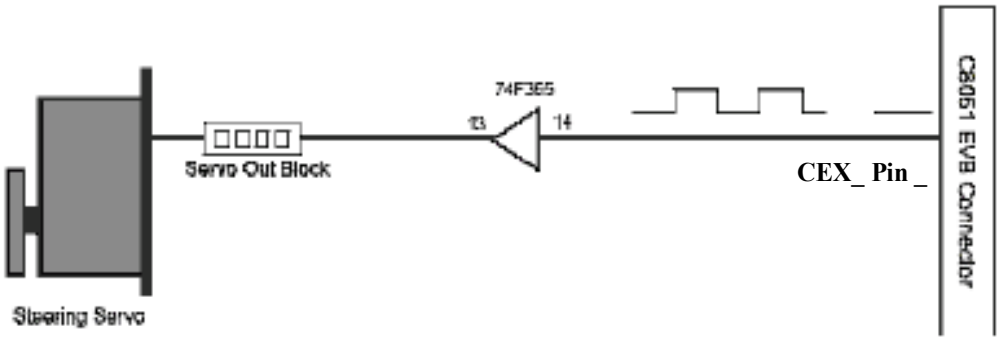


Figure 4.2 - Car steering servomotor control circuitry from Lab 3 (part 1)

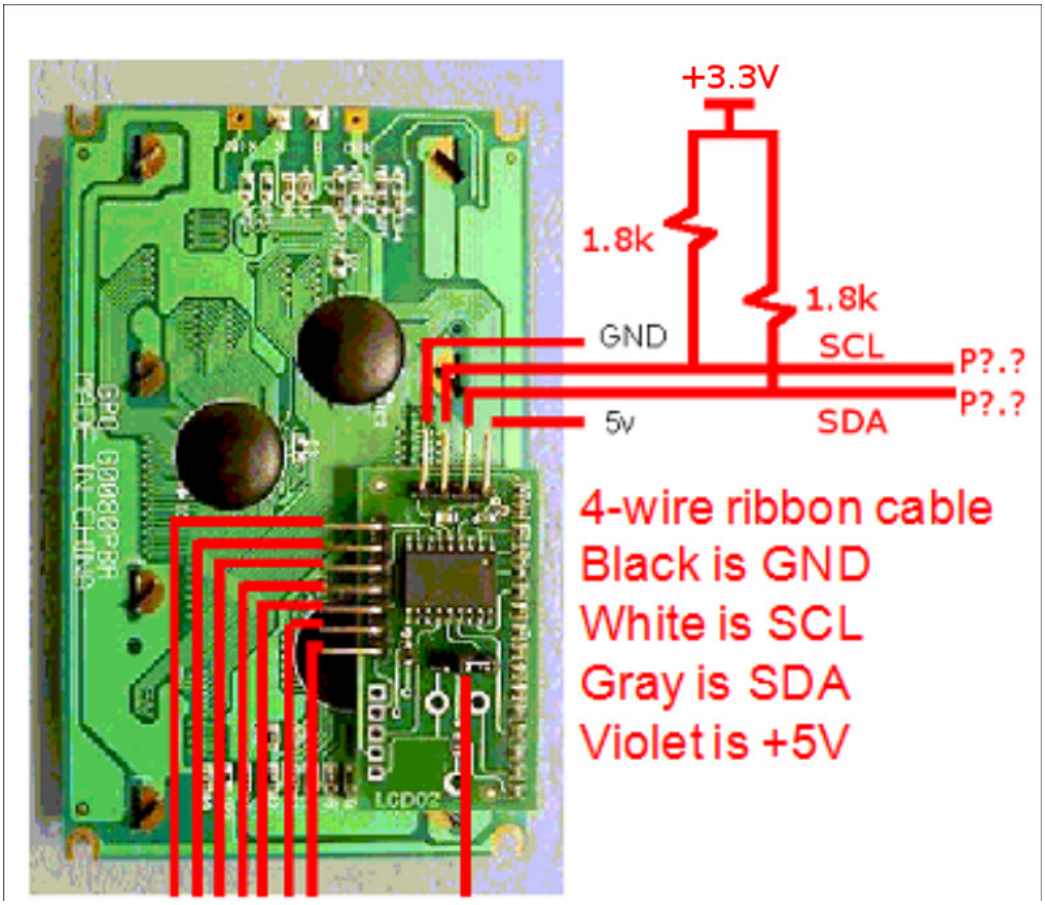


Figure 4.3 - Circuitry for LCD panel (Do NOT include 1.8k resistors if already present for compass)

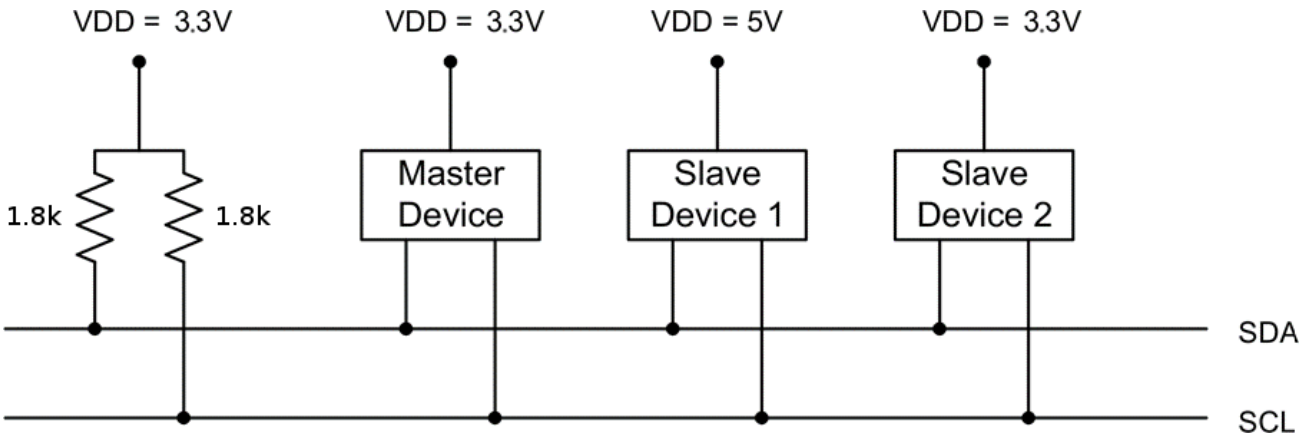


Figure 4.4 - Configuring multiple slaves on a single master

Note: Additional hardware – a run/stop slide switch connected to any unused I/O pin of the team's choice should still be used. We recommend P3.7, as this pin 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.

Optionally, 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 R2 in the voltage divider circuit.

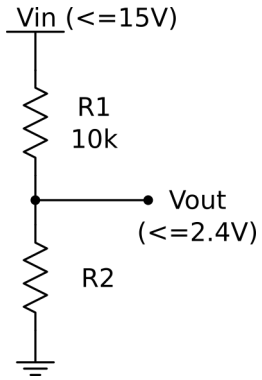


Figure 4.5 - Power supply voltage divider schematic

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 R2 to meet these requirements.

Note that only standard resistor values are available so values must be adjusted to use resistors stocked in the studio. Hint: 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 and verified with a voltmeter reading to be less than 2.4V before connecting Vout to the analog input pin on P1.

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. A 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 few seconds. The voltage may be displayed in mV, i.e. 11.5V = 11500mV to avoid floating point numbers.

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 gray plug must be disconnected from the EVB.

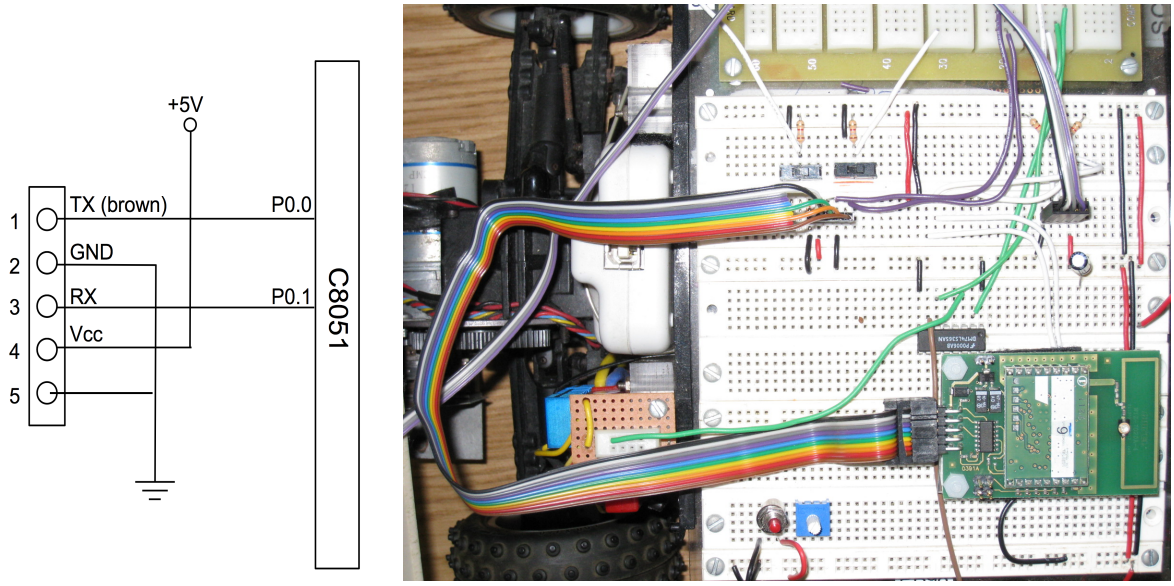


Figure 5.6 – Connections for RF transceiver and photo of module.

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 that sets the drive and servomotors to neutral.
2. The desired heading must be selectable. This can be done by pressing keys on the keyboard or keypad. The system should allow the operator to enter a specific desired angle, or pick from a predefined list of 0°, 90°, 180°, 270°. If the user is to enter a desired angle, the SecureCRT or LCD screen should indicate so with a prompt. If the user is to choose an angle from a predefined list, the screen should show the list along with the key press needed to select that angle.
3. The steering gain and drive gain (only for obstacle tracking, not used here) must also be selectable. As mentioned previously, the potentiometer is used to select the gain. This is set once with the other initializations. The final value must be displayed for the user to see, and allowing the user to make adjustments until a desired value is set is a nice feature.
4. Once the desired heading and gains are selected, the LCD should display the current heading, the current range and optionally the battery voltage. Updating the display every 400 ms or longer is reasonable. Updating more frequently is not needed and should be avoided.

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 interupt, 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 4th interrupt, 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? Except when backing up to avoid an obstacle, you do not need to allow for this but you should at least recognize that there might be a problem with the steering control based on the compass and desired heading error.
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 slight modification will allow it to use the keypad. See the following C function as an example on how to use the SecureCRT keyboard to enter values. Since you are required to set the gain by adjusting the potentiometer and performing an analog conversion, your value of `input` (must be declared as a `float`) in the function below will be the ADC result/25.0. This value should be printed using `printf_fast_f()`. The `while(1)` loop should continue to read the ADC and print the value until `getchar_nw()` returns a value other than 0xFF when a key is pressed on the terminal or `read_keypad()` returns a non-zero value from the keypad.

```
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. The objective to avoid obstacles has been modified. 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. The ranger should ignore anything more than 100 cm away since spurious reflections make measurements on anything farther away effectively useless without modifications to the ranger firmware. Once the first obstacle is detected (e.g., range value of ~50 cm) the car should stop and wait for input from the terminal ('L' or 'R'). On input the car should turn appropriately and then drive at the previous speed while waiting for `getchar_nw()` to return a space character ' '. At that point the car should revert back to following its previous desired heading. Now if the ranger detects a 2nd obstacle, once the desired separation distance is reached (a few cm), the car should halt and remain halted.

10. Summarizing, values users should adjust to improve performance are: steering gain (possible range of 0.0 – 10.2), ranger distances for obstacle sensing, and drive speed. NOTE: declaring and using the steering gain as a `float` may cause some unforeseen problems. It is acceptable to declare it to be an `int` in the range from 0 to 102 and then do a final divide by 10 in the statement where it is used to determine the steering pulsewidth. As always, to prevent integer truncation, make sure all multiplication operations are performed before any divide operations.

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 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 **GradingLab4Rpt-student.pdf** 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 R2 with R1 = 10k.
2. Demonstrate how a desired heading and the neutral drive 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, keyboard and/or pot.
5. Place car on floor. Slide switch to run. The steering shouldn't be jerky and should attempt to achieve the desired heading in a short distance of travel. Car will turn in the direction that results in reaching the desired heading the quickest. For example, if desired heading is 90° 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.
5. The car will stop at the first obstacle and then wait for a terminal input. The car should then follow the original heading once around the obstacle.
5. The car will stop at the second obstacle and not react to any changes once halted.
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 with the new heading, or then halt if not able to avoid the obstacle.
7. Your TA may ask you to explain how sections of the C code or circuitry you developed for this exercise work. 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. Teams are responsible for both the schematics and the code. All members 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 gains and ranger thresholds are to be tried. Make comments about the car's performance with both high and low gains. Determine a usable set of gains and thresholds for your car.

If implemented, 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 actual battery voltage, Vin, using a voltmeter 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 so 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 – Lab 4 Report

A brief written memo (see **GradingLab4Rpt-student.pdf** under **Lab 4** on LMS) plus plots and pseudocode is required for this lab (a C program-listing should **NOT** be included but it must be uploaded to LMS and **NO** formal schematic is required). The memo must have five response plots: 3 for the different values (~0.2, ~2, & ~8) of the proportional feedback compass error gain and 2 for different values (your choice) for the speed setting and ranger distance for stopping when an obstacle is detected. You should use the best steering gain for the 2 other trials. You may set the fixed speed setpoint to any value that works well. Plot the compass value, the ranger value, and the steering pulsewidth on the same time axis. An indication of when the drive motor halts may be helpful for understanding events. Values will need to be scaled to plot nicely. As a suggestion compass values should be between ±180°, ranger values clipped at 100 cm (to ignore noisy data) and steering pulsewidths 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 values and ranger heading change. Analysis of the plots should explain what is happening and why. Note: **not all plots need to work**. Explain why certain values prevented the car from meeting the desired results. 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. Although you don't need to show justification,

you should say whether a high or low drive speed performed better while controller gains were being optimized.

Finally a complete commented listing of the C code must be uploaded to LMS.

Grading – Preparations and Check-off

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. The main function *should* not be considered complete. Items to note are:

- Initialization routines are declared, but the routines should be taken from your previous code
- Pulsewidth variables are declared globally for 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 char print_flag = 0;   // flag for count of printing
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
unsigned char  print_count;     // overflow count for printing

__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;
}
        if (new_range)          // enough overflow for a new range
        {
            range = read_ranger();  // get range
            // read_range() must start a new ping after a read
            if (range < obstacle_threshold) // if an obstacle is detected
            {
                obstacle_tracking();  // Move into obstacle tracking routine
                set_range_adj();  // if new data, set value to adjust steering PWM
            }
            new_range = 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;
        }
        if (print_count>=25)
        {
            print_flag = 1;
            print_count = 0;
        }
        PCA0 = PCA_start;
    }
    // handle other PCA interrupt sources
    PCA0CN &= 0xC0;
}

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