Lab 5: Accelerometer Integration, Slope-directed Steering (2019S)

Preparation

Reading

Lab Manual

Chapter 5 - LCD & Keypad and Analog Conversion (review)

Chapter 7 - Control Algorithms

LMS, Course Materials

Accelerometer Data Sheet

LCD and Keypad Data Sheet (review)

Objectives

General

1. Keep your working project and code from Lab 4 someplace safe, UNALTERED. A modified version of it will be used later in Lab 6 to control a gondola on a turntable. For this lab a 2-axis Accelerometer will provide the control feedback signals instead of the Ranger and Compass.
2. A modified control strategy will use the data from the accelerometer to drive the car up or down a slope and stop when it reaches the peak, valley or other specified condition.
3. The LCD will be used to display relevant data from the program to assist in troubleshooting.

Hardware

1. Wire a single protoboard with the Accelerometer added to the Ranger and Compass on the SMB. Remember, only one set of pull-up resistors is needed on the SMB. **Although the Ranger and Compass will no longer be used, please leave them on your car.**
2. Keep the Liquid Crystal Display and number pad on the SMB interface.
3. (Optional) Add battery monitoring hardware and software if not done in Lab 4. This is a little more critical since a low battery voltage can be problematic when driving on the ramp.
4. A radio frequency (RF) link will be added again to allow for communication between the car and your laptop. Use this to receive telemetry information sent from the autonomous car to your laptop and also to aid in debugging your code.
5. Keep the potentiometer from Lab 4, which can be used to adjust one of the feedback gains.
6. Reinstall the BiLED from Labs 1 and 2 with the series resistor and control it with a digital output that passes through a 74365 buffer.
7. Other devices that may be used include slide switches, pushbuttons and a BiLED.

Software

1. Revise the C code used in Lab 4. Write a program that first calls the **accelerometer initialization routine** `Accel_Init_C()`, then calls your `read_accel()` function and then

sets the PWM for the steering servo based on the pitch & roll tilt of the car so that it drives or turns in the direction of the desired slope. Both the side-to-side tilt and the front-to-back tilt will be used to determine a PWM for the drive motor. The main code will also need to average 4 to 8 samples from the accelerometer each time, since the signal is noisy.

2. Continue to use routines to output parameter and settings on both the LCD display and the terminal to aid in troubleshooting.
3. As in Lab 4, there will be parameters to adjust affecting the behavior of the system. There will be a few different adjustable proportional gain feedback constants that must be optimized to give the car the best performance. Write C code to allow user entry of gain constants using either the keypad or terminal.

Motivation

This lab provides a chance to interface the *Smart Car* to another sensor system. Accelerometers are being added to many electronic devices to provide safety as well as convenience features. Being able to detect a free-fall condition may allow a laptop to park a hard drive and prevent a head crash that would render the drive completely useless. Similarly, 3-axis accelerometers can detect orientation of cameras and tablets so that text and graphics are automatically rotated to allow proper viewing.

The module used here is an incredibly sophisticated device that can detect accelerations along all three axes and generate interrupts based on maximum and minimum values in any direction with durations of more than or less than a predetermined period. It also includes a 3-axis magnetometer that measures magnetic field strength in each direction as a compass. For this lab, only a subset of all the features will be used. Here the gravitational acceleration in the x and y directions are all that is needed to determine the tilt and direction of a slope. As such, only the 16-bit values associated with the x-axis (oriented as the side-to-side direction, +x to the left) and the y-axis (oriented as the front-to-back direction, +y to the front) when the module is inserted in the protoboard in the ascribed fashion shown below in Figure 5.1. As a signed integer the acceleration values go from -32768 (-2g) to +32767 (+2g) where g is 9.8m/s^2. Dividing the output integer by 16 then yields ±1g = ±1024.

The goal of this lab is to control the steering and speed of the car as another example of how sensor information can be incorporated in a closed-loop feedback system. It is worth noting that aspects of this test mimic altitude control of a hovering blimp. For that situation, the altitude control is then dependent on both the thrust power and thrust angle.

(Optional) In addition to the accelerometer 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 motors, 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. The battery monitoring circuit developed in Lab 4 may prove to be more important here. Driving up a slope significantly increases the load on the drive motor compared to driving on a horizontal surface. A given control algorithms will behave very differently when a battery is low and unable to deliver the full amount of current the drive motor is expecting under a specific set of conditions. Keep your previous monitoring circuit and software to assist in determining if your car's battery needs to be recharged. Otherwise you will need to add a voltage divider to reduce the ~13V battery down to the 2.4V range of the ADC1 reference voltage.

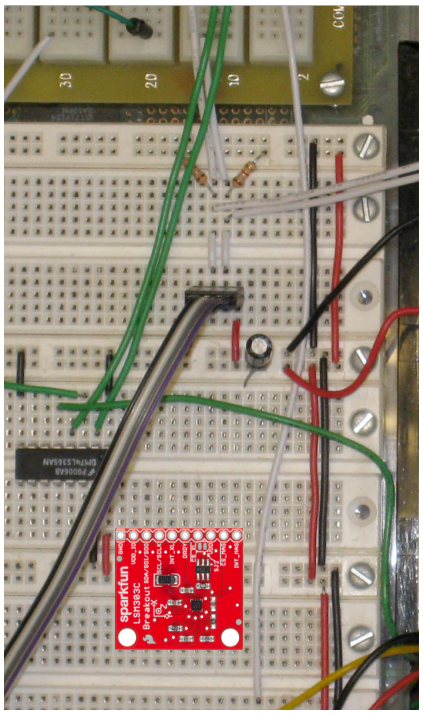

Figure 5.1 - Correct orientation of the Acceleration sensor (red module on <u>your</u> protoboard) on the car. +Y is toward the front of the car (towards the right) and +X is toward the left side (towards up).

Finally, this lab still requires the integration of an LCD screen and keypad. As before, the keypad may be used to input the desired heading (up or down the slope) and to set gain constants. The screen was 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 x- and y-acceleration and set the gain constants as needed. This allows these values to be set on-the-fly, rather than recompiling the program each time a gain is changed.

Again, a radio frequency (RF) link will be added to the car to permit communication between the car and your laptop. This is used to record telemetry data to be sent from the autonomous car to your laptop so that system response data can be plotted. The RF link appears as another laptop serial port, but requires a new driver. The Gondola Info link on LMS provides the details as does the **Drivers for RF link** section in the **Installing_SiLabs-SDCC-Drivers** manual also on the LMS course front page.

With both the RF link and the LCD/keypad panel working, teams will have more flexibility in determining how they will enter parameters and update values. Either system can be used to provide menu-driven instructions for the user.

Lab Description and Activities

As with all the previous labs, for the rest of the course the 3-student team is expected to develop various parts of the software, each assigned to a member, in parallel. Each team must still submit a report.

The hardware and software from Lab 4 will be the starting point for this lab. Only a single protoboard from Lab 4 on should be used for final check-offs.

The integrated software and hardware will result in a car that can detect the direction of the ramp tilt and have the car drive in some direction based on the slope until it levels out at the bottom or top or meets some other conditions. The integrated software should poll the run/stop switch(es) connected to port pin(s) specified by you 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 or as specified for a given semester.

Velcro has been added to the car to hold the LCD and keypad board. 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. NOTE: when unplugging the LCD board ribbon cable from your protoboard, **make sure you pull the ribbon cable by the header plug on the ribbon cable and NOT by the wire. The cable wires will be pulled off the connector pins.** It is extremely annoying when an LCD board can't be used because a broken wire prevents it from working or causes it to work intermittently.

Use one combined `printf()` statement that includes x- & y-accelerations (tilt values), the current gains, drive and steering pulse widths, (and battery voltage). This should be printed in columns (separated by commas) to allow processing and plotting using Excel or MATLAB.

Hardware

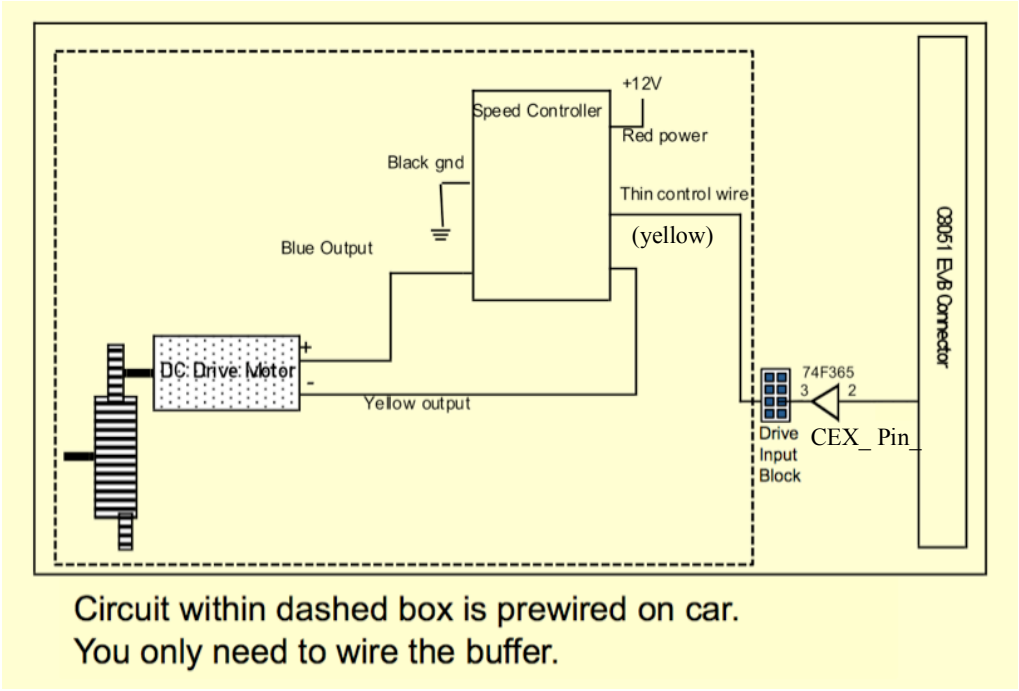

Figure 5.2 - Car drive-motor circuitry from Lab 3 (part 1).

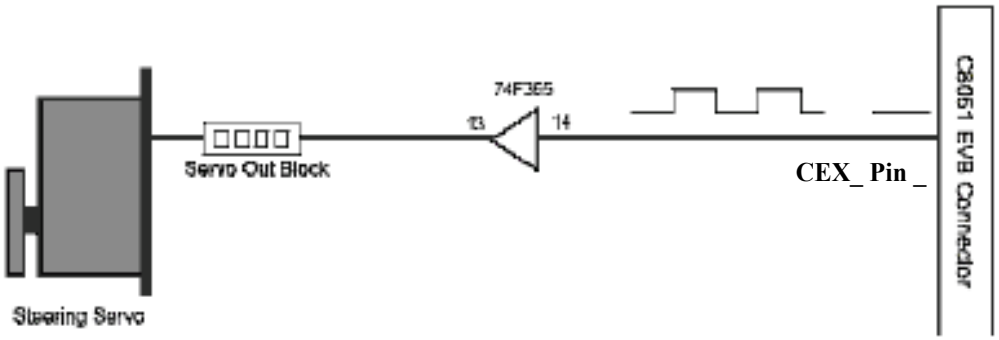

Figure 5.3 - Car steering servo motor control circuitry from Lab 3 (part 1).

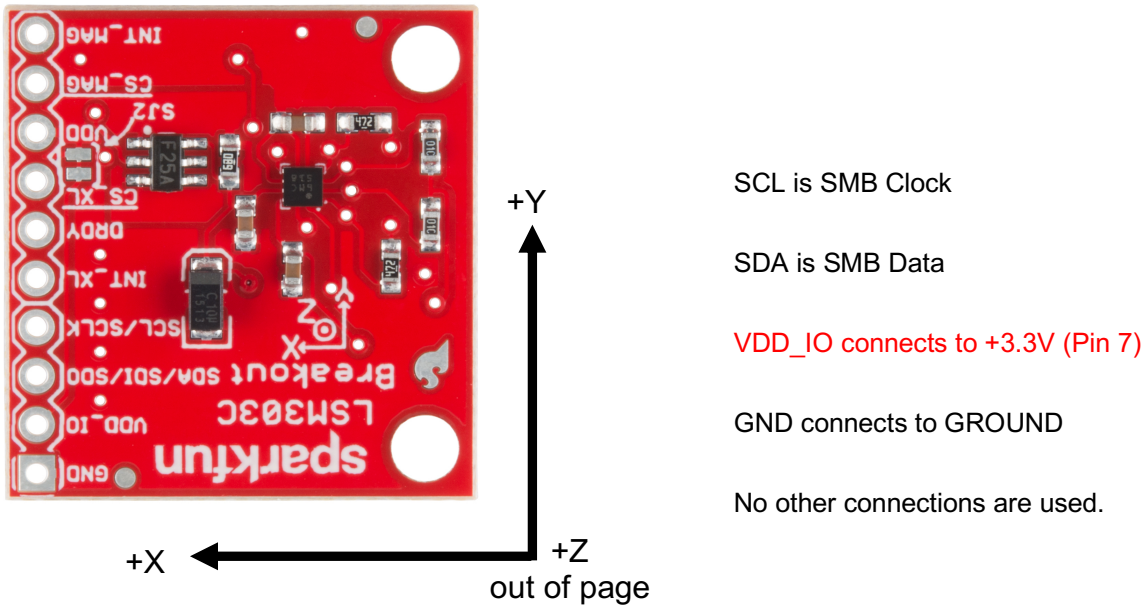

Figure 5.4 – Orientation and labeled connections for the Acceleration module.

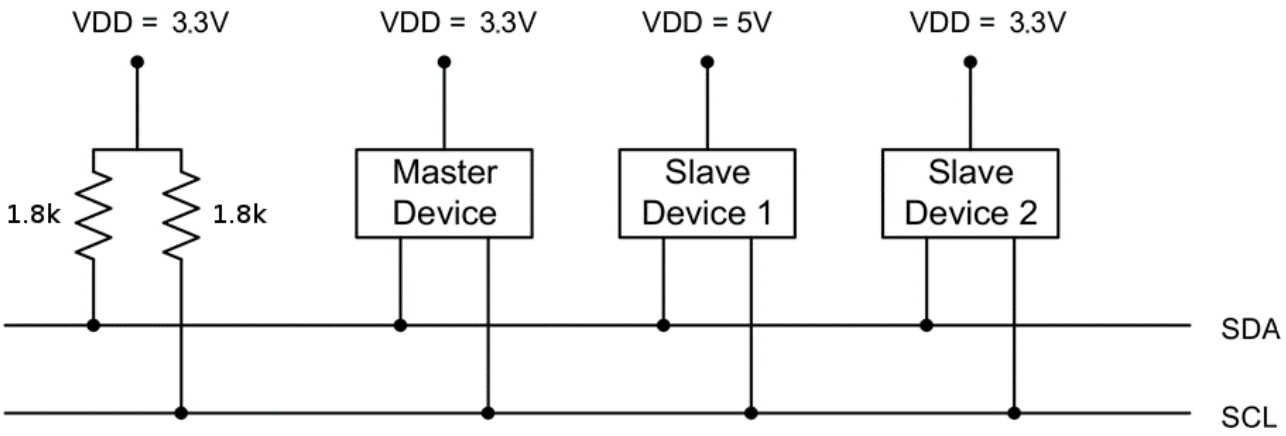

Figure 5.5 – Configuring multiple SMB slaves on a single master.

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 on your terminal.

The car RF transceiver module requires 5V 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.

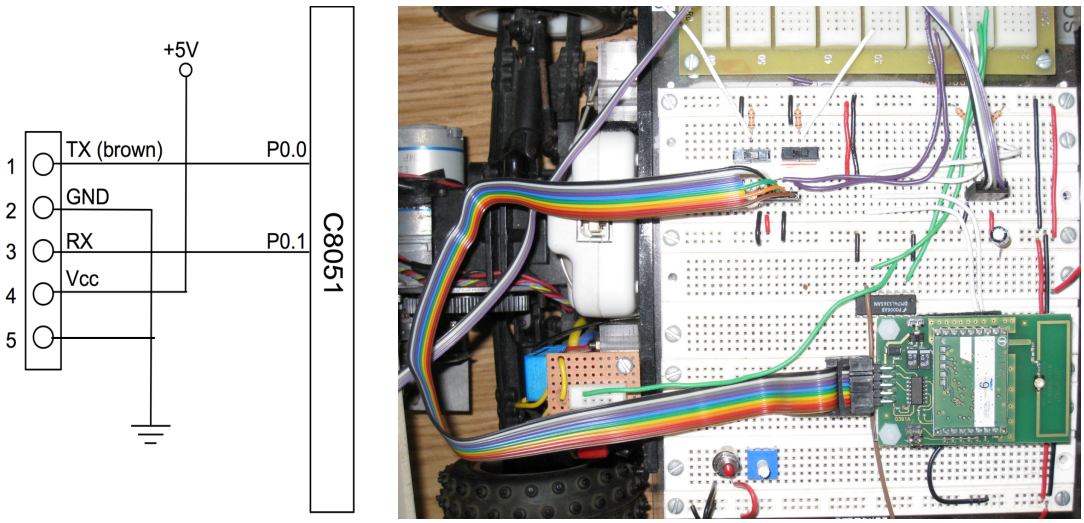

Figure 5.6 – Connections for RF transceiver and photo of module.

Note: It is suggested that the previously used additional hardware – a run/stop slide switch connected to any unused I/O pin of the team's choice – still be used to disable 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. Use whatever was done in Lab 4.

Software

Write code to use the LCD and keypad and/or terminal to print the results.

Continue developing the code from Lab 4 with the following constraints:

1. Include a run/stop slide switch.
2. Of the 2 drive feedback gains, the drive gain for the front-back pitch must be selectable via the pot setting used in Lab 2 or 4. Adjusting the pot from min to max should create a gain that varies from 1 to 50 (a pure integer value is fine) that may be updated on the fly as the car drives. The other gain is to be set by pressing keys on the keypad or the terminal and is fixed for a run. See item 3. below. Monitoring the battery voltage is again optional, but when using

the pot value on a separate analog input from the battery voltage monitor, it is necessary to wait ~2 ms after changing the analog MUX channel selector before starting a conversion, otherwise the converted value will be wrong.

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 everything has been specified, the LCD and/or terminal should display the current x & y accelerations, the current gains and the motor pulsewidths. Updating the display every 400 ms is reasonable (but not more frequently).

There are several other considerations:

1. The steering servo and the speed controller must be updated every 20 ms. The PCA hardware does this as long as it is properly configured.
2. The Accelerometer updates every 20 ms (50 times a second). The controller will work best if each reading is a new value, so continue to update the heading after a PCA ISR.
3. The keypad can be queried for input when appropriate. This shouldn't be done more often than once every few ms. Alternatively, the laptop keyboard may be queried using the function `getchar_nw(void)`, the version or `getchar()` that immediately returns a value of 0xFF if no key has been pressed, but otherwise returns the normal ASCII code for the character.
4. It is necessary to try several different gain values for the feedback. The car should attempt to get to the bottom or top of the ramp quickly in a short distance but the noisy acceleration values may cause the steering adjustments to be jerky.
5. For this semester the car will start at the top of the ramp toward one side or the other, heading downward. A second slide switch (your choice of port pin) determines which way the car will turn when it detects that the slope has exceeded a given threshold. Once the run begins, the car should start down the ramp while monitoring the front-to-back (pitch) tilt. Once it detects that the slope threshold has been exceeded the car should make a 90° turn left or right, based on the second slide switch position, and drive until the front-to-back tilt levels out. Once sideways on the ramp with front-to-back tilt ~0°, the car should halt. The BiLED should be green when driving forward and red when stopped.

   Note that while attempting to level out, the side-to-side tilt will determine whether the car needs to steer to the left or right. This may depend on the direction of the turn set by the slide switch. Also, it is expected that this control algorithm will be implemented in sequential steps: the 1st to start down the hill while monitoring the slope, and 2nd to level out after the 90° turn.
6. The condition where maximum error in the x or y tilt 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 allowed motor pulsewidths are not exceeded.
7. After an initial estimate of an appropriate gain, code can be implemented to allow the user to change the pitch (front-to-back tilt) gain during execution rather than only at the start of the program. While in the infinite loop, the program may ask to manipulate the proportional gain

of the drive motor. Use the following function and call it from the main function. The function below on p. 9 uses the terminal keyboard, but your code may use the keypad instead.

Specifically, here are some things to remember:

1. The i2c.h header function that should already be downloaded from the LMS course page and put in the correct folder so that SDCC has the required accelerometer initialization function, `void Accel_Init_C(void)`. **Make sure your code calls this** with all the other initializations.
2. The Accelerometer values are read from registers in the module. The SMB ID address of the Accelerometer is **0x3A**. The status register is 0x27. The 2 least significant bits 0 and 1 in the status register go high when the x-axis and y-axis acceleration values are ready to be read. At that point registers 0x28 and 0x29 contain the 16-bit x-axis acceleration and registers 0x2A and 0x2B the y-axis acceleration, with the low byte in the lower register number. Since the acceleration values are extremely noisy, the low byte may be discarded (effectively set to zero, leaving only the 8 most significant bits from the high byte) for both the x and y values. After sign-extending the high byte to get a 16-bit signed integer (logical shift with << 8), the equivalent 16-bit value (8 bits of real data and 8 bits of zeros) is in the 8 high order bits. The value needs to be shifted down into the 12 low order bits (logical shift with >> 4). Since the values are still noisy, 4 reading should be made consecutively and all averaged together to give the program something that is practical to use. This is a common problem with many sensors where digital filtering (averaging) is used to remove noise from signals. Once the averages for both the x and y data have been calculated, the accelerometer read routine should set global variable for the control calculation to use. Make sure the array (Data[4]) is still declared as **unsigned char**. The pseudocode for the read routine is given below.

```
Clear the averages
    avg_gx = 0;     //declared as signed int
    avg_gy = 0;     //declared as signed int
For 4 iterations (or maybe 8)
    (A delay is no longer required: Wait one 20ms cycle to avoid hitting the SMB too frequently and locking it up)
    Read status_reg_a into Data[0] (register 0x27, status_reg_a, indicates when data is ready)
    If the 2 LSbits are high: (Data[0] & 0x03) == 0x03, then continue, otherwise reread the status
    Read 4 registers starting with 0x28. NOTE: this SMB device follows a modified protocol. To
    read multiple registers the MSbit of the first register value must be high:
        i2c_read_data(addr_accel, 0x28|0x80, Data, 4);     //assert MSB to read mult. Bytes
        Discard the low byte, and extend the high byte sign to form a 16-bit acceleration
          value and then shift value to the low 12 bits of the 16-bit integer.
          Clear sums: avg_gx = 0, avg_gy = 0
          Accumulate sum for averaging.
                avg_gx += ((Data[1] << 8) >> 4);      //a simple "<< 4" WILL NOT WORK;
                avg_gy += ((Data[3] << 8) >> 4);      //it will not set the sign bit correctly
          Or to attempt to acquire 12 bits of accuracy with more required averaging, use:
                avg_gx += ((Data[1] << 8 | Data[0]) >> 4);
                avg_gy += ((Data[3] << 8 | Data[2]) >> 4);
          Done with 4 iterations (or maybe 8)
Finish calculating averages.
    avg_gx = avg_gx/4 (or use >> 2 for faster execution, >> 3 for 8 iterations)
    avg_gy = avg_gy/4 (or use >> 2 for faster execution, >> 3 for 8 iterations)
Set global variables and remove constant offset, if known.
```

gx = avg_gx (or gx = avg_gx - x0 if nominal gx offset is known)
gy = avg_gy (or gy = avg_gy - y0 if nominal gy offset is known)

3. The control statement for the steering servomotor and drive motor and will be similar to:

```
steering_pw = steering_pw_center - ks * gx      (ks is the steering feedback gain)
drive_pw = drive_pw_neutral + kdy * gy          (kdy is the y-axis drive feedback gain)
Add correction for side-to-side tilt, forcing a forward movement to turn the car.
drive_pw += kdx * abs(gx)                       (kdx is the x-axis drive feedback gain)
```

One additional issue that may be addressed is the asymmetrical strength of the drive motors between forward and reverse. This may result in inaccuracies with the car stopping at the peak or in a valley. You may choose to modify your code to adjust for this by incorporating different feedback gains, depending on the direction of travel. The solution to this is left open-ended. Introduction of an integral term in the feedback control alongside the proportional term will provide a mechanism that will increase the value on the drive motor over time until it is large enough to actually move the car. The drive motor forcing function modifications would be something like:

```
drive_pw += kdx * abs(gx) + ki * error_sum      //ki is the integral gain
error_sum += gy + abs(gx)
```

NOTE: Each accelerometer must be calibrated for a correct zero point so that the x & y readings return a ~0 when the car is on flat ground. This should be done EVERYTIME at the start whenever your program runs. Tilting the car 90° in any direction will change the zero point significantly so that the car may try to move even while on a flat surface. New values must be found if the car has been jostled or the accelerometer has been changed. It is suggested that 64 readings be averaged for both the x & y values and those numbers be saved and subtracted from every reading. Make sure your car is on a flat surface and undisturbed when the program begins its calibration to insure accurate offset values are found.

The following function can be used to adjust the value of the feedback gains on the car without having to recompile. If the keypad is used to enter characters rather than the terminal keyboard, the gains may be adjusted without having to carry a laptop around. This function allows the user to bump a value up or down with key presses. Alternatively, a function could be written to enter a gain from scratch.

```
void Update_Value(unsigned char incr, int maxval, int minval)
{
// Constant is the global int variable to be adjusted
    int deflt = 20;  //Fill in desired default value
    char input;

    // 'c' - default, 'i' - increment, 'd' - decrement
    // This can easily be changed to use the keypad instead of the terminal
    input = getchar_nw();
    if (input == 0xFF) return;
    else if (input == 'c')     Constant = deflt;
    else if (input == 'i')
    {
        Constant += incr;
        if (Constant > maxval) Constant = maxval;
    }
    if (input == 'd')
    {
```

```
            Constant -= incr;
            if (Constant < minval) Constant = minval;
        }
        printf(" Constant = %d ", Constant);
        return;
    }
```

Data Acquisition

When your code is functioning correctly, gather data to plot response curves for your steering control. You should plot x & y accelerations and both motor pulsewidths (y-coordinate) vs. time (x-coordinate). In order to save the data, you will need to print the x- & y-accelerations to the terminal screen 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 a few curves for different gain combinations. You must find an 'optimal' setting, but you should also look at other combinations 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 mg on the y-axis (where $1g = 9.8m/s^2$ so $1mg = 9.8mm/s^2$).

Lab Check-Off: Demonstration and Verification

1. Demonstrate how the heading and drive direction & speed are determined by the tilt orientation. Show by lifting the car and changing the pitch and roll (look up these terms if you aren't sure of their definition) that the steering and drive will compensate for tilt error.
2. Start the program with an accelerometer auto-calibration on the flat floor then set the desired feedback gains using the potentiometer and keypad or terminal.
3. Place car at the top of the ramp (see current semester specs), pointing down the slope. Set the slide switch to let it know in which direction turn when the slope is steep then wait for the other slide switch to start. Car will head down the slope and maintain a heading that maximizes the slope as determined by the accelerometer tilt values until the slope exceeds the preset threshold. At that point the car will turn left or right and drive until the pitch is ~0°. The steering may be jerky, but should attempt to level out in a short distance of travel. Start from both the left and right edges and turn appropriately. BiLED indicates driving (green) and stopped (red).
4. Car will stop on the slope when the pitch is level (or starts to point uphill). Car must be perpendicular to the direction of maximum slope when it stops.
5. While driving, the front-to-back tilt gain **may** be adjusted via the pot so program flow continues without waiting for a keyboard entry. Keyboard input using the function `getchar_nw()` returns a value of 0xFF if no key has been pressed. The new value should be displayed when any gain is adjusted.
6. 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.
7. Display the gains, motor pulsewidths, and x & y acceleration values on the LCD screen.
8. Print the output to the terminal screen in the following format:

```
X accel. - Y accel. - Drive PW - Steering PW
  xxxx,      xxxx,      xxxx,      xxxx
  xxxx,      xxxx,      xxxx,      xxxx
  ....,      ....,      ....,      ....
```

   Capture the screen output and save it to a file on your laptop PC.

NOTE: A number of cars have half-inch black circles on them to indicate they have mechanical issues that may prevent them from driving **UP** the ramp due to weak drive motors or worn transmissions. Those cars will only be able to drive down the ramp successfully. If required, an attempt to drive up should be demonstrated showing the car stalls no matter how large the gain is on the drive motor. In those cases, a "user assist" option may be implemented where a team member may push the car to get it to move up to the top. A verified stall will not prevent a team from being checked off for Lab 5.

Check-off and Lab Report Writing Assignment

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

Justify the algorithm provided to adjust the steering servomotor and drive motor based on the x-axis and y-axis acceleration values. Explain how the pitch and roll tilts affect these measurements and how the measurements are used to change the steering angle and drive speed & direction.

Sample C Code for Lab 5

The following code provides an example of the main function for combining control of both the steering and the drive motors. Your debugging skills developed through experience in deciding which values to observe on the terminal and LCD display while troubleshooting will speed up the development process:

- 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 accelerometer 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 modified versions of similar functions in your previous code.

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

//-----------------------------------------------------------------------------
// 8051 Initialization Functions
//-----------------------------------------------------------------------------
void Port_Init(void);
void PCA_Init (void);
void SMB_Init (void);
void Interrupt_Init(void);
void PCA_ISR ( void ) __interrupt 9;
void read_accel (void);    //Sets global variables gx & gy
void set_servo_PWM (void);
void set_drive_PWM(void);
void updateLCD(void);
void set_gains(void);  // function which allow operator to set feedback gains

//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_accel = 0;  // flag for count of accel timing
unsigned char new_lcd = 0;    // flag for count of LCD timing
unsigned int  range;
unsigned char a_count;        // overflow count for acceleration
unsigned char lcd_count;      // overflow count for LCD updates

//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
void main(void)
{
   unsigned char run_stop;    // define local variables
   Sys_Init();                // initialize board
   putchar(' ');
   Port_Init();
   PCA_Init();
   SMB_Init();
   Interrupt_Init();
   Accel_Init_C();

   a_count = 0;
   lcd_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)
         {
            set_gains();      // function adjusting feedback gains
            run_stop = 1:     // only try to update once
         }
      }
```

```
        if (new_accels)         // enough overflows for a new reading
        {
            read_accels();
            set_servo_PWM();    // set the servo PWM
            set_drive_PWM();    // set drive PWM
            new_accels = 0;
            a_count = 0;
        }

        if (new_lcd)     // enough overflow to write to LCD
        {
            updateLCD(); // display values
            new_lcd = 0;
            lcd_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
        a_count++;
        if(a_count>=____)
        {
            new_accel=1;
            a_count = 0;
        }
        lcd_count++;
        if (lcd_count>=____)
        {
            new_lcd = 1;
            lcd_count = 0;
        }
        PCA0 = PCA_start;
    }
    // handle other PCA interrupt sources
    PCA0CN &= 0xC0;
}
```

Lab 5 Accelerometer Report

Group names: _____ _____ _____

The following list is the material that would logically be included in the **brief** report. It is not intended as a direct guide. You have done the experiments, so you have the best idea of what you needed to complete those projects and what you learned during the process. This means that the following information may be incomplete based on your experience with the project. If you feel part of what you did during the laboratory is pertinent, include that information.

(suggested # of pages of text (not including plots), but you may go higher)

Introduction	(<1pg)	**20**	
Purpose/Objectives	_____	_____	
Overview of accelerometer feedback control	_____	_____	*Sum:* _____
Results, Plots, Analysis of Plots, & Conclusions	(4-5pg)	**60**	
Verification (how was performance to specifications tested)	_____	_____	
Logical layout of data & Presentation of Plots	_____	_____	
Labeled axes with units	_____	_____	
Analysis of plots from tabulated terminal data w/ explanations	_____	_____	
Problems Encountered & Solutions	_____	_____	
Suggested improvements to HW & SW	_____	_____	*Sum:* _____
Code			
NOTE: no code listing is requested here, but it must be included in the Lab Notebook and you must upload your **Fully Commented** .c file to LMS under **Assignments > Lab 5, only 1 per team**			
Formatting & Neatness	**20**		
Cover Sheet (names, section # & side, grading TA)	_____	_____	
Spelling & Grammar	_____	_____	
Required: Academic Integrity and Division of Labor page - **signed** (See the provided template form)	_____	_____	*Sum:* _____
Lateness (unexcused)			
-20% per School Day	-20% x _____	_____	*Sum:* _____
Total		**100**	***Total Points:*_____**

NOTES: No reports will be accepted if the team has not been checked off for Lab 5. No report grades will be given without uploading softcopies of the .c file to LMS for archival purposes in addition to the signed hardcopy of the report. Use last initial of members in the .c file name (ex. 2B_HHO_lab5.c for a team in section 2, side B with last names Hamlet, Othello, and Shakespeare). Only one team member should upload the file but it must contain the names of all 3 members in the header comments. Everyone on the team must sign the hardcopy of the report using the Academic Integrity form given below.

Be sure to read "(Bad) LITEC_Report_examp_graphs" under this rubric on LMS to avoid common mistakes when plotting data. Axes must be scaled and units specified. These plots are from Lab 6 data but they are being used to show bad plotting formats (independent of the data).

LITEC Accelerometer Report Guidelines (revised, 2019 Spring)

The Lab 5 report for LITEC documents the plotted data obtained from the car on the ramp. This rubric (GradingAccelRpt-student) on LMS in the Laboratories & Worksheets section under Course Materials, lists most of the items to be included, but the list is not necessarily exhaustive. Most of the written portion deals with describing and analyzing the plots.

The report should include detailed descriptions of the final goal: the feedback system on the car involving the accelerometer & driving the car to the top or bottom of the hill. Discussions should explain how the PWM pulse-width calculations are made based on the tilt errors and feedback gains (proportional, derivative, and integral, if appropriate). With respect to response plots (described below), analyze the various plots and justify their characterstics for the sets of gains used.

2019 Spring: Although for check-off you must demonstrate that the car stops sideways on the ramp, halfway up, when started in either side (left or right). Four response plots are required for the report. One should be a good response with "optimal" gains that starts on either side, turns left (or right), and stops in the middle. It must be demonstrated that the opposite side and turn direction also work although it isn't necessary to plot that run too. Other response plots do not necessarily have to successfully meet the specifications. They may be anything, including runs that may fall off the edge, continue down to the bottom of the ramp, continue to turn left and right without coming to a complete stop, or anything else. Be sure to annotate each plot to indicate key points and explain what is happening.

Reports must contain:

1) Introduction.
2) Analysis of plots from acquired data.
3) The program listing (.c file) for the gondola (Lab 6) program should be uploaded on LMS. The report should reference parts of it while explaining how the feedback control works for the grader while they read reports.
   Program listings **must be formatted as follows**:
   Make sure proper indenting is used consistently throughout
   Include an appropriate prolog (programmer names, section & side, date, brief description, etc.)
   Line comments and block comments should be used liberally with detailed description of what the code is doing or how it works
4) Clearly labeled and captioned plots for data acquired during lab, with scaled axes & units
   Normalized drive motor pulse-width (-100% to +100%) and pitch (front-to-back tilt) from accelerometer, both vs. time as the car drives up the incline.
   Time plots from the car showing heading angle as it corrects itself for several different values of P gains as the car responds to the accelerometr data. (Follow the cases given in the lab procedure.)
   {Optional} Any other plots for which you may have acquired data.
5) Do not forget to include a cover page with the team member names, section, side, Grading TA name, and report name. Also you are required to include the following integrity & division of labor sheet at the end that is signed by all team members.

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) _____
_____

Software implementation: _____
(pseudo-code & code) _____
_____

Data analysis (if relevant): _____
_____
_____

Report development & editing*: _____
(schematic, diagrams & plots) _____
_____

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

_____

_____

_____

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