Lab 4: Heading/Ranger Integration and Reading Battery Voltage

Preparation

Reading

Lab Manual

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

ADC Data Sheet

Chapter 7 - Control Algorithms

LMS, Course Materials

Electric Compass Data Sheet

Ultrasound Ranger Data Sheet

Objectives

General

1. Integration of the heading (steering) and altitude (speed) control systems developed in the previous lab, which will be used in the *Blimp*.
2. Use Analog to Digital Conversion to read battery voltage, as was implemented in Laboratory 2.

Hardware

1. Wire a single protoboard with both the Ultrasonic Ranger and the Electronic Compass. Only one set of pullup resistors is needed on the System Bus.
2. Configuration of an interface between the C8051 and a DC drive motor using a buffer and a speed controller module.
3. Configuration of an interface between the C8051 and the servo motor using a buffer.
4. Design a voltage divider circuit to measure the battery voltage.

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 sets the PWM for the drive motor based on the range from the ultrasonic sensor to an object.

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.

Motivation

The *Smart Car* is being used as a test bed to develop the software for the blimp. This will involve integrating the two subsystems developed by ranger and compass pairs.

The simplest blimp control is in “hover mode”, where the embedded controller causes the 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 blimp uses a speed control module to amplify power to 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 blimp.

Concurrent with the need to control the heading of the blimp, is the need to control its altitude. The blimp 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 blimp. If the blimp 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 blimp 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 blimp 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 blimp. It is worth noting that this test mimics a hovering blimp. If one desired to have forward motion of the 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.

In addition to the ranger and compass integration, this lab involves monitoring the vehicle’s battery voltage. Both the Smart car and the Blimp are powered by rechargeable batteries, that forms a part of the unit. As the charge in the battery is being consumed by running the SmartCar or the Blimp, 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. It becomes important for us to keep note of the battery voltage while we are running the car or the blimp.

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

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 in the notebook of the other pair in a team.

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 and have speed control based on the distance of an object above the car. The integrated software shall poll the run/stop switches connected to P3.6 and P3.7 to start and stop any one or both control functions. There will be similar switches on the *Blimp*. The details are left up to the team.

Hardware

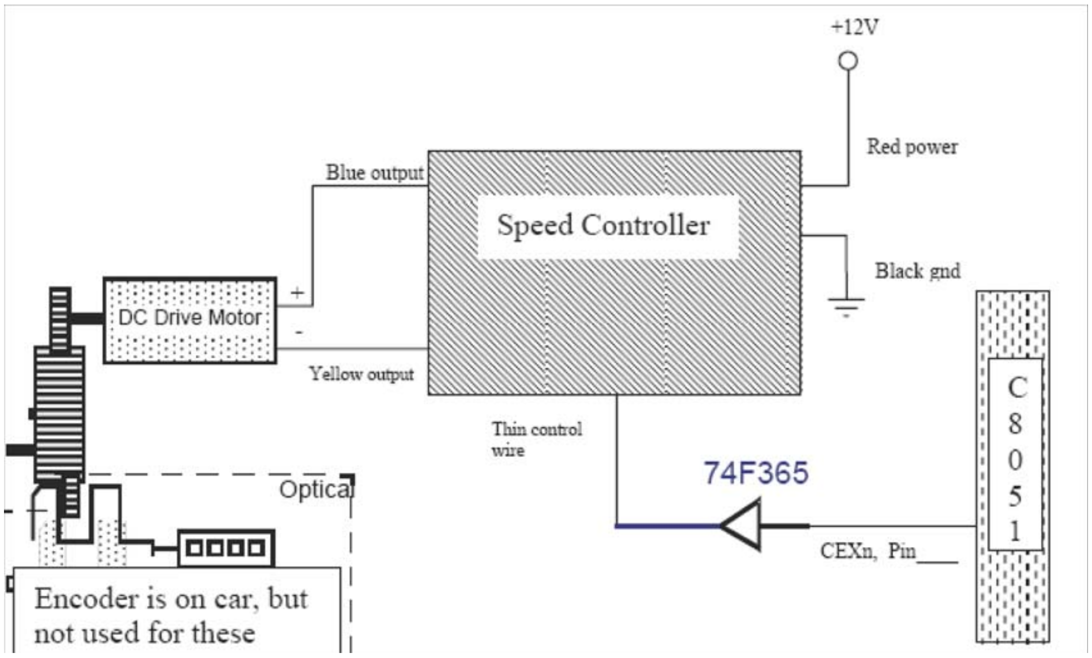


Figure C.3 - Car drive-motor circuitry from Lab 3 (part 1)

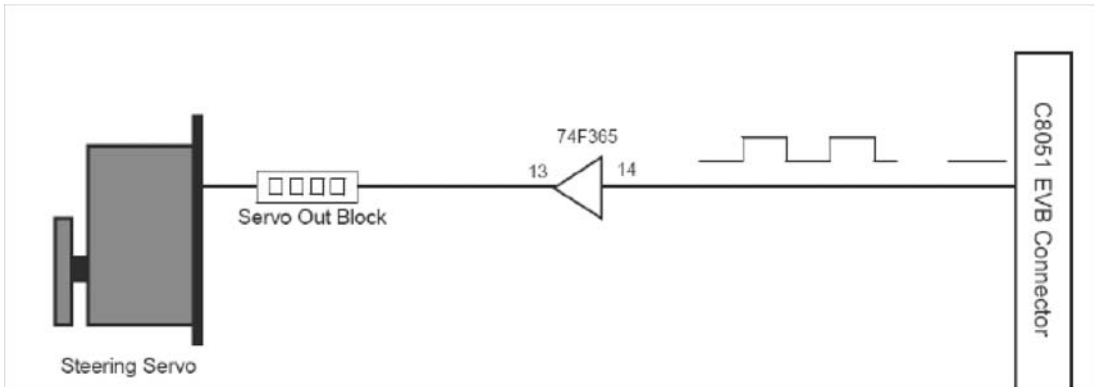


Figure C.4 - Car steering servo motor control circuitry from Lab 3 (part 1)

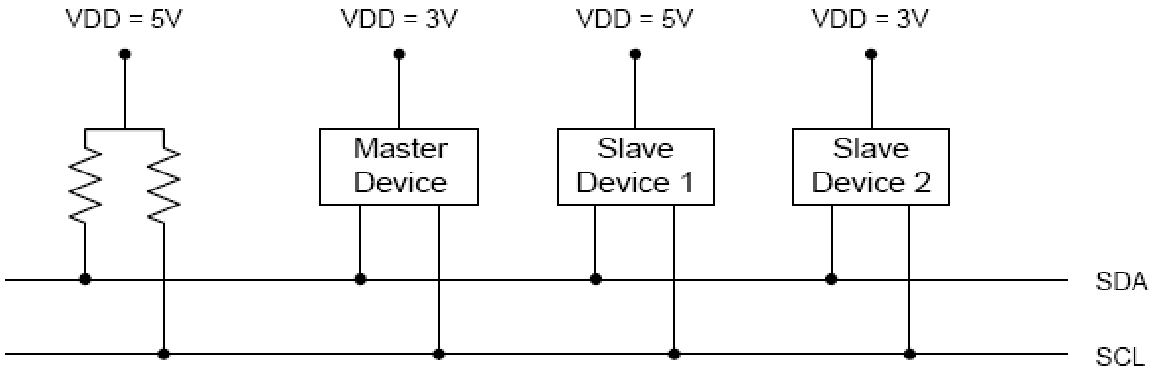
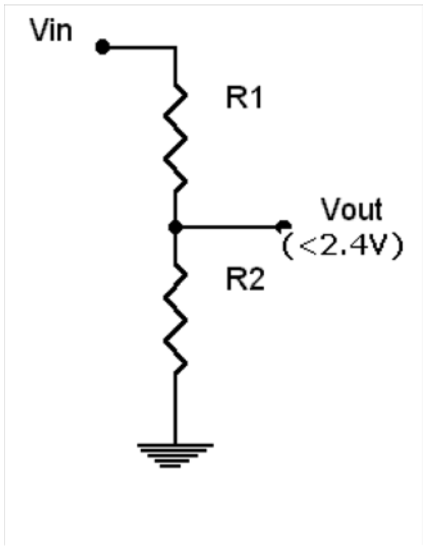


Figure C.5 - Configuring multiple slaves on a single master

Note: Additional hardware – a run/stop slide switch connected to any unused I/O pin of the team choice. We recommend either P3.6 or P3.7, as these pins will also have switches on the blimps.

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

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



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

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

Therefore, pick R1 and R2 to meet these requirements.

$$Vout = R_2 i = \frac{R_2}{R_1 + R_2} Vin$$

$$(Vin - Vout) = R_1 i = \frac{R_1}{R_1 + R_2} Vin$$

Hint: 5 Kohm< R1 < 25Kohm, 1Kohm<R2<5Kohm

2. Mount the resistors on your protoboard and connect them to 12V block and ground. Note that the actual voltage measured at the block might be greater than 12V, but we are safe because we are designing our circuit for 15V supply!
3. Use a voltmeter to prove that the output of the voltage divider is 2.4V or less. Have a TA confirm this before you continue. Voltage in excess of 5V might cause the input on port 1 to fail.
4. Only after a TA has seen a voltmeter check of the divider voltage, then connect the midpoint voltage to a pin of Port 1. Remember to use pins not associated with the PCA outputs, so use Port pins 4, 5, 6 or 7.

Software

Write code to do A/D conversion on the chosen port pin in the previous step, and print the results. The code should print the battery voltage about every second.

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 could be a one time option when the code is started. Or the better option: the operator has the option to set a new desired heading every time the run/stop slide is put into the stop position. The system may either allow the operator to enter and desired angle or may allow the operator pick from a list, 0, 90, 180, or 270 degrees.
3. The ranger reading that results in zero thrust power must be selectable in the range 30-80 [cm]. It is expected that the code will query the user during initialization, but your team has the option to use a different method.

There are several other considerations:

1. The steering servo and the speed controller must be updated at least once every 80ms and no more often than once every 40ms. The PCA ISR is set to generate the pulses every 20ms, so don’t change the PCA ISR timing.
2. The Compass updates every 33.3 ms, 30 times a second. The blimp controller will work best if each compass reading is a new value, so continue to update the heading every second PCA ISR, or every 40ms.
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 ISR, or once every 80ms.
4. 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.
5. What happens if the car goes in reverse?
6. 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 pulsewidths are not exceeded.
7. 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.

```
void 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;
    }
}
```

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 range are initialized. As stated above these must be user selectable. A valid option is that the user may only select from a list.
3. Place car on floor. Slide switch to run. Car speed is controlled by ranger. 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.
4. Car will turn in the direction which 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.
5. Use the paper on the class room floor, placing the car near the step. Set the car to head east and trace the path the car follows using a felt pen.
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. Write the value of gain that you choose for the steering.
8. Print the output in the in the Hyperterminal screen in the following format
```
Actual heading   Distance   Battery Volt
    xxxx            xxx         xxx
    xxxx            xxx         xxx
    ...             ..          ..
   xxxx            xxx         xxx
```
9. Capture and print the screen and attach it to your lab notebook. Note that "Distance" in the above table is the reading of the Ultrasonic ranger indicating the height of the object above the car.

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

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 and the ultrasonic ranger 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>

//-----------------------------------------------------------------------------
// 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;
unsigned char r_count;// overflow count for range
unsigned char h_count;// overflow count for heading

//-------------------------------------------------------------------------------
// 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, set the servo PWM
         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_drive_PWM ();// if new data, set drive 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;
        }

        PCA0L = PCA_start;
        PCA0H = PCA_start >> 8;
    }

    // handle other PCA interrupt sources
    PCA0CN &= 0xC0;
}
```