AC 2009-2010: SOPHOMORE-LEVEL PROGRAMMING CONCEPTS AND METHODOLOGY COURSE IN COMPUTER ENGINEERING, COVERING BOTH HARDWARE AND SOFTWARE

Haluk Ozemek, San Jose State University

Preetpal Kang, San Jose State University

Albert Khanh Nguyen, San Jose State University

pradeep badhan, San Jose State University
Undergraduate student in Computer Engineering

A Sophomore-level Programming Concepts and Methodology course in Computer Engineering covering both hardware and software.

Abstract

One of the retention problems in an engineering discipline is that the students are not introduced to their field early enough to have a good understanding of their profession. Usually, the first two years of the engineering undergraduate program are spent in completing general education and science requirements. With this in mind, the authors have developed a sophomore-level Programming Concepts and Methodology lecture/lab course in the Computer Engineering Department at San Jose State University, which emphasizes the key elements of computer engineering. The main objective of this course is to illustrate how a high-level program, in this case C language, interacts with peripheral hardware. Throughout the semester, students are introduced to C language to be used on a microcontroller-based board to interface with sensors and transducers. A complete set of lab exercises enable students to build robots as a class project without requiring any background in electronics or programming. The course also introduces the fundamentals of embedded systems and hardware/software co-design to sophomore students.

INTRODUCTION

Computer Engineers must have proficient knowledge of both computer hardware and software which has produced the fundamentals of this course. In this course, a sophomore becomes knowledgeable on how software can interact with hardware, and how real world problems are solved by employing both hardware and software. As a result, students establish strong educational foundation which eliminates the difficulties on an actual project that they encounter in their professional life.

Before this course was introduced, the Computer Engineering Department at San Jose State University lost about 80% of its students to other non-engineering majors because the previous course focused only on the C, C++ . Consequently, students did not understand how software can control hardware. With this new course, the retention rate has completely reversed; 80% of students stayed in computer engineering after completing this course[1].

COURSE OVERVIEW

The course is divided into two phases. In the first phase, students learn the basic C programming language and basic computer organization. Since many students do not have any prior programming experience, the programming experiments start with very basic concepts and structures but rapidly become complex at later stages of the course. The C programming section covers the following topics: Program Control, Functions, Arrays, Pointers, Characters and Strings, Formatted I/O, Structures. The computer organization part concentrates on the Central Processing Unit (CPU), Arithmetic Logic Unit (ALU), Memory, Input/Output Unit (I/O) and their relationship to one another. Throughout the lectures, a correlation between human anatomy and computer organization is explained for better understanding the course concepts. This excites students once they correlate the relationships between a human body and a computer. As

an example, the clock can be correlated with heart; one synchronizes the events in a computer and the other regulates the events in a human body. Execution of an instruction generates necessary control signals, which are transmitted on the control bus to perform the necessary register transfers. This can be associated to the brain executing a command such as "get up!", which in return, generates electrical signals to the leg muscles via nerves and contract them to establish the getting up action. Hardware is fixed in both the humans and the computer which is difficult to change, but software applications can be changed easily. A C language can be correlated to a human language. Different programming languages may be run on the same hardware as the human being can learn different languages. These discussions during the lecture keep the students interest high and trigger their curiosity. In the laboratory, students first use Visual Studio to develop simple programs and learn program debugging methods.

In the second phase of the course students not only continue to improve and acquire new programming skills, but they also learn how to use a microcontroller. Understanding basic computer organization makes the microcontroller operation easily understandable. Furthermore, students conduct research on how to interface different peripheral hardware, such as sensors and actuators. In the laboratory, students develop projects using microcontroller development kit developed by SJValley Engineering (SJVE)[2].

The laboratory experiments for this course are designed to cover the lecture topics and elements to provide self-motivation to students. When students have "hands on" experience and learn how easily they can interface various sensors and actuators, they develop self-confidence and interest that help them throughout their educational and professional career.

Experiment 1: Simple Input/Output: The first experiment allows students to become familiar with Visual Studio and programming language C. Students learn how to create, write, compile, and debug programs in Visual Studio.

Experiment 2: Conditional Operations: The second experiment introduces conditional logic. Students write a program that contains different types of conditional operands.

Experiment 3: Simple Loops and Arrays: The third assignment summarizes effective and efficient use of loops. Students learn how to reduce the number of lines of code by using various types of repetition. Furthermore, arrays are introduced in this assignment.

Experiment 4: Programming the Atmega 324 microcontroller and blinking the LEDs: In this experiment, students start using the microcontroller kit. The purpose of this experiment is to get students familiar with the microcontroller by writing and compiling a simple program in AVR Studio and downloading the program on the chip.

Experiment 5: Function Call and Detecting Switches: Functional calls are purposely not introduced until this assignment. In this experiment, students start to utilize the libraries, which results in function calls.

Experiment 6: Serial I/O: In this assignment, students learn how to use a UART for input and output. The skills gained in this assignment are very useful when debugging the projects.

Experiment 7: Sensors and Servo Motor Controller*:* In this assignment, students learn how to interface infrared distance sensors with the microcontroller. Furthermore, the assignment shows how to control servo motors.

STUDENT PROJECTS

Right from the time when the course was introduced at San Jose State University, there have been many creative student projects as the course is improved over time. One of the most successful student projects is presented in this paper. The project was completed by two students, who developed a security robot that can identify a human presence. Once a human is detected, it required an RFID (Radio Frequency Identification) code to be scanned. Afterwards, it would compare the scanned key code with key codes stored on the chip. The robot also has the ability to follow motion. In addition, students interfaced a LCD to the robot to display useful information.

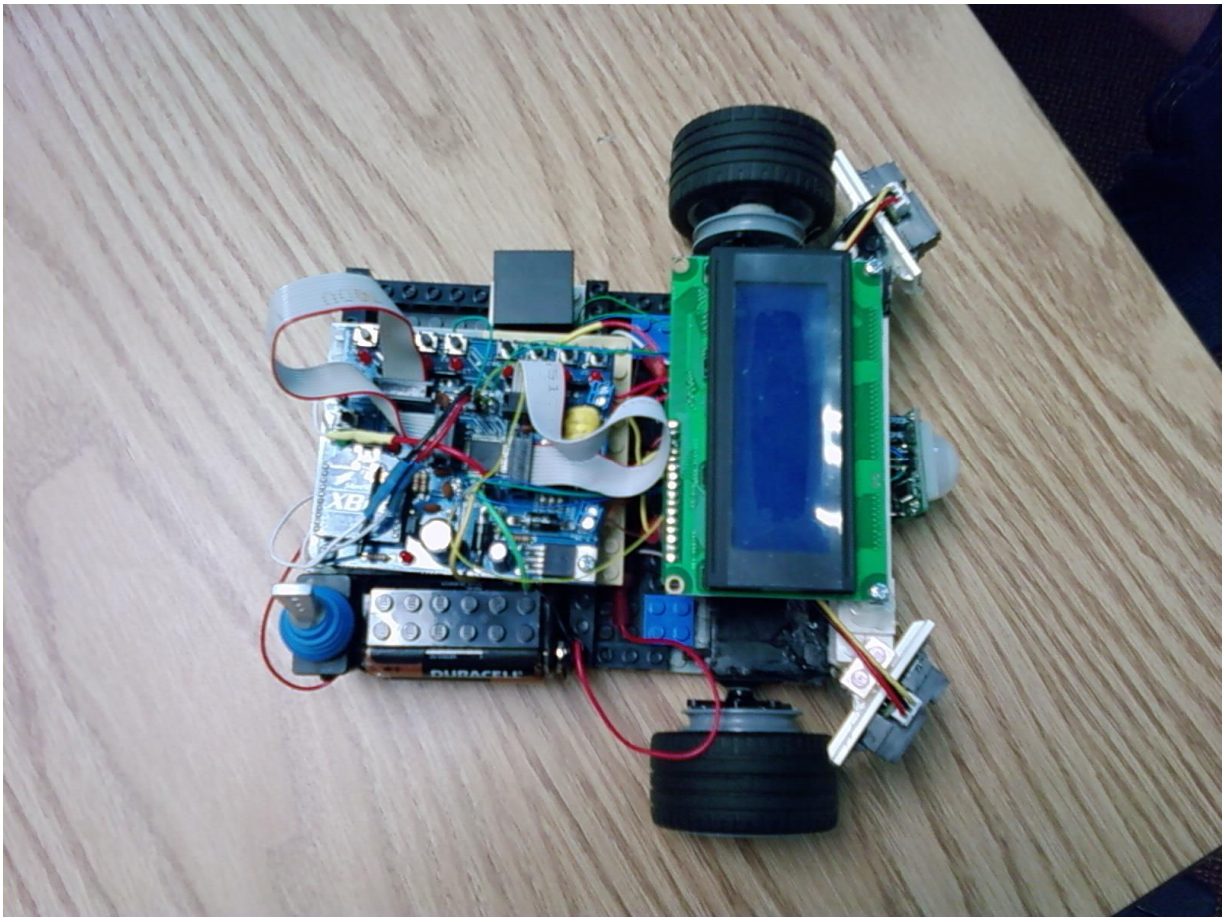
CONCLUSION

This is a sophomore level introductory course that combines computer hardware and a high level software language, and enables students to understand the essentials of computer engineering field. When a sophomore requires hands-on experience on how easily sensors and control servo-motors are controlled, they immediately acquire self-motivation to learn more on computer engineering topics and create very innovative projects. With a strong foundation in microcontrollers, students not only learn the true value of computer engineering, but they also form a career path to do better in their career. Besides, this course has also created a big impact on the retention rate of Computer Engineering students at San Jose State University.

REFERENCES

1. Departmental observation and Office of Institutional Research at SJSU, "University Student Enrollment by Class Level, College & Major (2009, Feb. 1). [Online]. Available: http://www.oir.sjsu.edu/cognos8/cgi-bin/cognos.cgi
2. Atmega324 Ultimate Kit (2009, Feb. 1). [Online]. Available: http://sjvalleyengineering.com/index.php?option=com_content&view=article&id=76&Itemid=34

SECURITYBOT 9000
BY BEN TANKERSLEY AND EVAN HOLLOWAY



***Abstract*—this lab report describes the processes and actions that were taken to complete our final project of creating a security robot.**

I. INTRODUCTION

The purpose of this lab was to create a security robot which implements and utilizes the capabilities of distance, motion and heat sensors along with servo motors for mobility.

II. DESIGN METHODLOGY

In order to complete this lab, several things had to be done first to sync the PCB with the computer and execute the desired program.

1. First attach the PCB to the power supply
   a. Assure that the hot and ground wires are attached correctly, with one other member verifying a proper connection to avoid destroying the PCB.
   b. Turn on the power supply and set slightly above 9V.
   c. Check for power LCD on PCB if present, good. If no power LCD turn off immediately.

2. The next step is to open an AVR Studio 4 project.
   a. Select and open AVR Studio.
   b. Once opened click the "New Project" button to open a new project.
      i. Next select the "AVR GCC" and name your project.
      ii. Select next, select the AVR simulator, finally on the right select ATmega324p
      iii. The compiler is now ready to accept code.

3. Begin programming in the AVR Complier.
   a. First include the necessary libraries.
      i. Add the libraries (#include):
         "AVRcommon.h"
         <stdio.h>
         "serial.h"
         "serial2.h"
         "taskScheduler.h"
         "adc.h"
      ii. Need to add the source libraries for the included libraries. To do this right click on the source folder in the top left of the compiler, select " add existing source files" and continue to select all the previous mentioned libraries.

4. Now to program the robot with our code.

5. Load the completed code onto the PCB.
   a. The boot loader is pre-loaded onto the PCB so there is no worry with this.
   b. The next step is to connect the wireless UART transmitter.

      i. Connect the UART into any usable USB port.

      ii. Open the device manager:

1. Windows Pound + Pause Break → Hardware → Device Manager

iii. Find the COM port assigned to the UART.

c. Now open the Megaload.exe

i. We must set up Megaload by selecting the COMM port we recently determined and setting the speed to 38400bps, as this is the speed in which the boot loader can speak to the PCB

d. Select the file which the code is saved to. It will be in the same folder as the common AVR files.

e. Simply reset the PCB to load the program

6. Parts List:

- Blue PCB
- Power supply
- Ribbon cables
- AVR Studio 4
- Megaload
- Hercules
- 3 Infrared Distance Sensors
- Motion Sensor
- RFID Sensor
- LCD Display.
- Battery Packs
- Servo Motors
- Lego's
- Power Switch

III. SCHEMATICS

Before wiring the components in, be sure to add in the proper initializations in the coding, as such:

PCB Initialization:

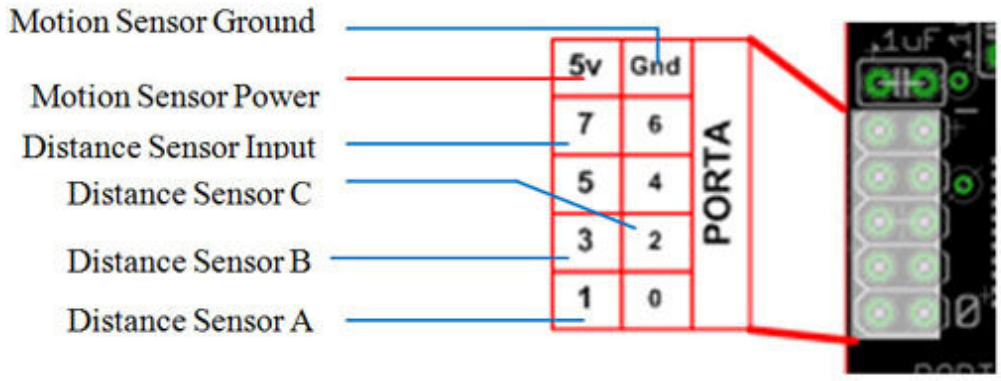
```
initializeSerialBaudRate(38400); // PCB talkes at 38400 bps
initializeSerialRecieve(true);
initializeSerialTransmit(false);

RFID Tag sensor:
initializeSerialBaudRate2(9600); // RFID talks at 9600bps
initializeSerialRecieve2(true);
initializeSerialTransmit2(false);
ENABLE_INTERRUPTS;


Motor Controller:
//Initialize_Motor_Controller();        //starts the motor control
initializeADC();
```

Wiring:

There are four major wiring circuits: PORTA, PORTB, PORTC, PORTD. Each one wires into a different function, as follows.

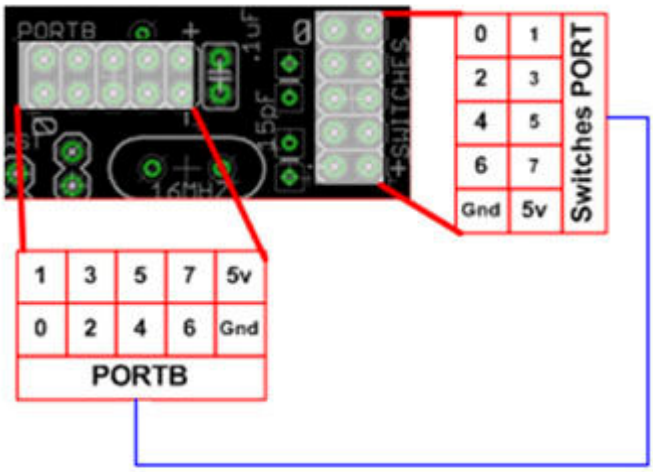
There are four major wiring circuits: PORTA, PORTB, PORTC, PORTD. Each one wires into a different function, as follows.

PORTA is divided up amongst the motion and distance sensors:

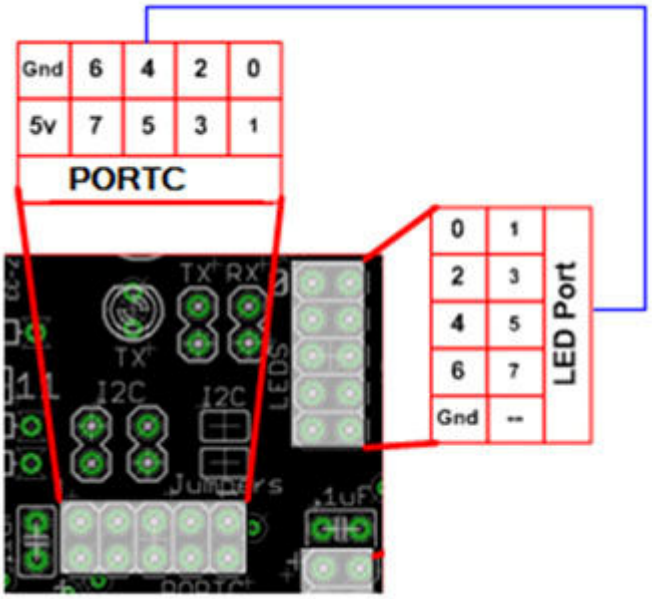


As seen here, each pin has a specific use. The three distance sensors and the motion sensor run off of PORTA. Though it's configured to be a PIN actually, and receive inputs from each sensor.

PORTB is simply a ribbon cable wired to the Switches, as such:

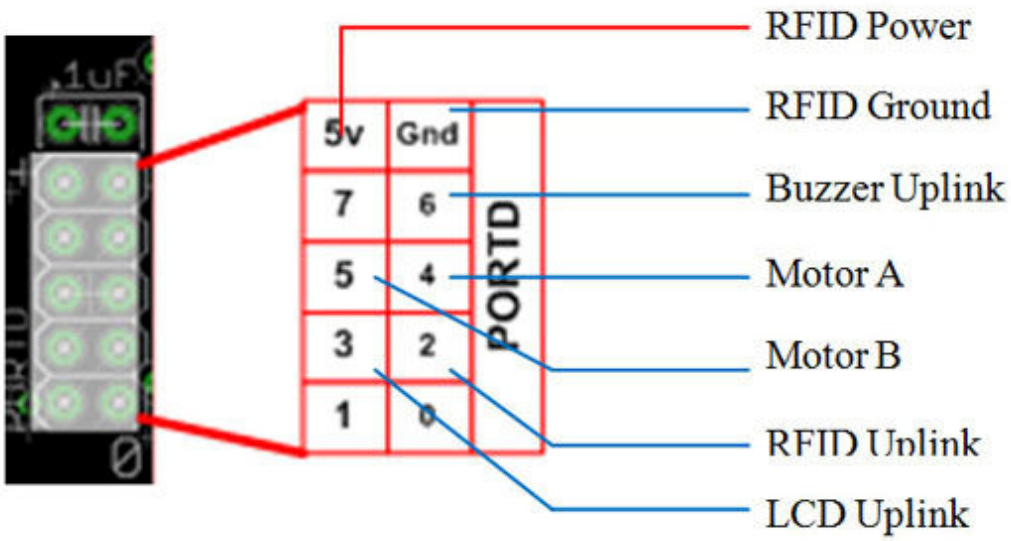

PORTB
.1uF
15pF
16MHz
+SWITCHES
0 1
2 3
4 5
6 7
Gnd 5v
Switches PORT
1 3 5 7 5v
0 2 4 6 Gnd
PORTB

PORTC is again a ribbon cable, this time to the LED port, as follows:



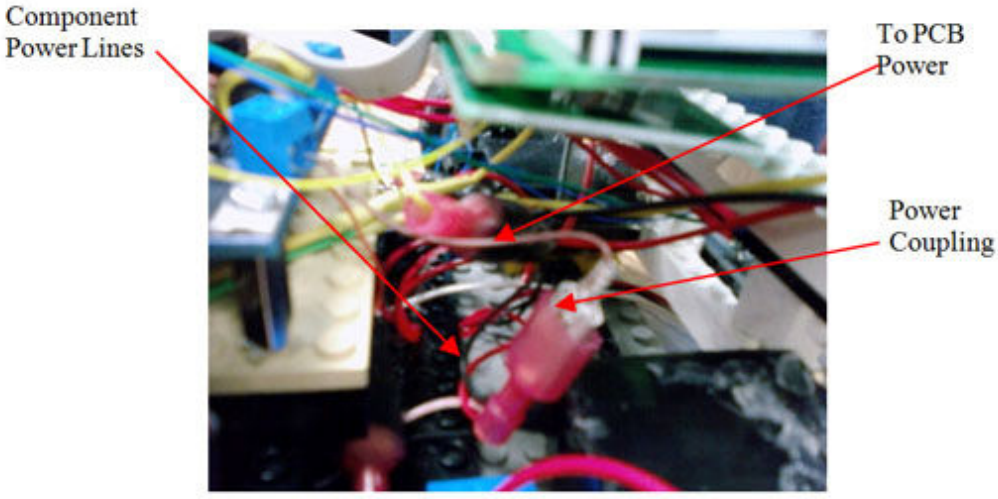
Align the power sides of the ribbon cable, and attach.

PORTD is wired to the RFID sensor, the LCD, the buzzer, and the two motors:

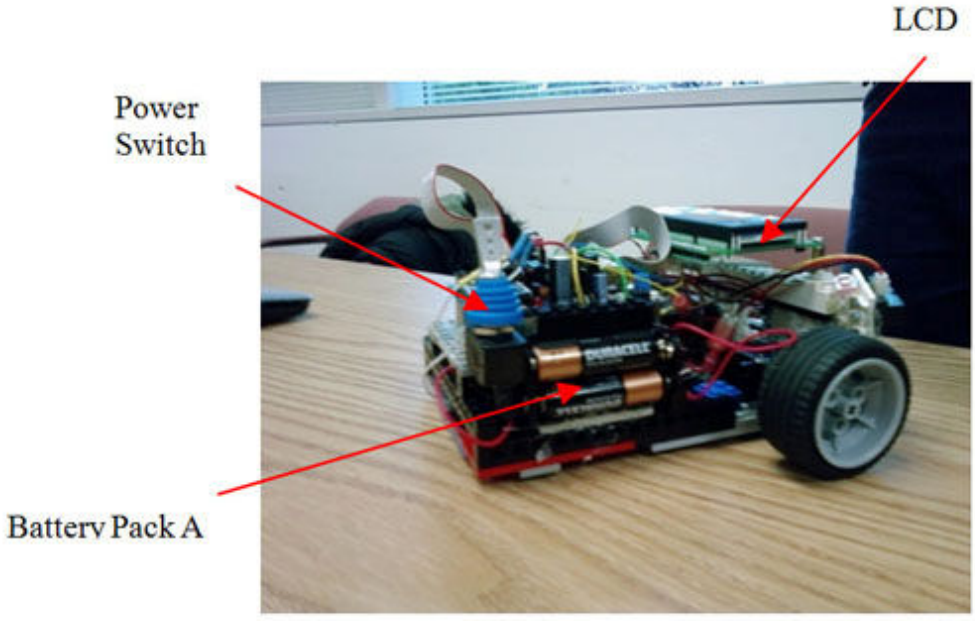


As shown, the RFID runs off the power and ground, with its uplink to the PCB in PD.2. The LCD Uplink is wired into PD.3. The Motors run off PD.4 and PD.5 for uplinks. The buzzer uplinks to PD.6. Power and ground for everything besides the RFID are run off the board itself, not PORTD.

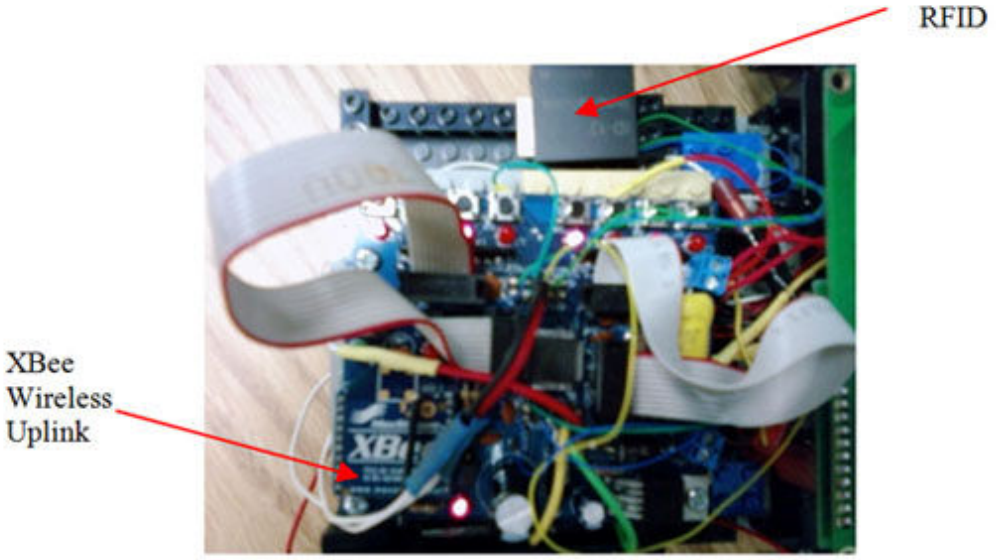
Aside from the PORT connections, the power and ground wiring ran through the PCB, including the power to the PCB from the battery packs. The battery packs are wired into a series, allowing the two packs to combine and use less power to run the robot. The battery packs are wired directly into the PCB, where as the auxiliaries (sensors, motors, RFID) are run off the PCB outgoing power, linked together with power couplings as so:



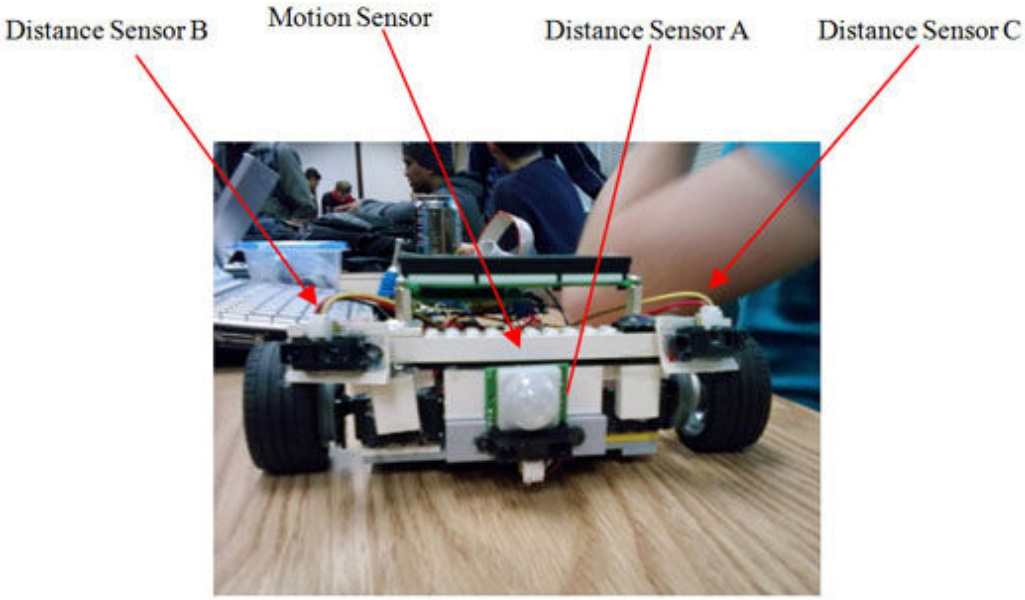
The battery packs (pack B is hidden under the PCB), LCD, and power switch:



The overall PCB wiring ended up something like so, though it's difficult to get a solid picture.



Our sensors are all applied as so (front view):



IV. TESTING PROCEDURES

1. After loading your program onto the PCB the testing procedure is very simple.
2. Open the Hercules.exe from the desktop.
    - At the top go to the serial tab.
    - Under this tab, change the port to the same port found earlier
    - Also, change the Baud to “38400”.
    - Be sure to close the port in the Megaload program, then open the port in Hercules.
3. Now, once again, press the reset button the PCB and the “hello world” program should initialize in the window.
4. Right click in the window and check the “local echo” option as on.
5. You should now be able to communicate back to your device and get the desired outcome.
6. Upload the PCB with the robot code as built. Reset, and check to see if the robot runs smoothly.
    - Take notes while the robot runs, trying to figure out what is working and what isn’t.
    - Adjust coding accordingly.

V. TESTING RESULTS

The robot had some major problems. The code wasn’t producing the correct functions, and the robot mainly just twittered around in circles. Once the motor code was fixed, work began on the infrared distance sensors, which would allow the robot to dodge walls and obstacles, as well as determine how far away a person or object was. At first, it didn’t work. Many frustrating hours later, the simple fix of speeding up the infrared sensor’s input speed proved to be the solution. Finally the RFID sensor was attached, and coding for it applied to the master code. Several code corrections, missing libraries, and finally meshing the two codes together, the robot could distinguish between a good RFID tag and a bad one. With a working, moving, stable robot, housings for the auxiliary pieces (batteries, sensors, wires, PCB) could begin. Many Lego’s, some hot glue, and some sauder produced an excellent quality, highly manageable and serviceable robot.

One major problem encountered was the motors. After finding the working stop-speed, the robot could be turned on at a dead stop. The problem lied with attempting to return the robot to stop-speed after it had begun moving. Even with knowing the proper settings and coding, the motors still turned, albeit slightly. However, the solution arose to tie the robots motors directly into the state-machine. Should the robot need to come to a complete halt, it returns itself to a state where it can still search out obstacles and movement, but without the motors on. A simple fix.

The motion sensor proved to be a horrible little thing. It produces a conical area of reception, and thus is hard to mount. Any variation in height or angle could completely de-align the sensor, making it useless. At the same time, it wasn’t actually following motion. It was just locating it, and sitting there. We recoded the motors and distance sensors to work much more specifically with the motion sensor. Finally the motion sensor clicked into place, and the robot chased any movement that it found.

VI. CONCLUSION

This project was designed to build a security robot. While the flamethrower idea was a little to much, and the buzzer idea was implemented too late, the security robot turned out quite well. It is quite capable of searching out something in motion, following it, while avoiding walls and obstacles. It can ask for identification, and sound an alarm if the identification is improper. For a first semester project in programming, it came out remarkably well.

One thing learned was the amount of time a project like this can take. Programming alone easily broke 25+ hours to do, not to mention actual testing. The chassis took over 10 hours to build, and rebuild as components were added, removed, moved around, and more. All together, including final testing, this project cleared 60+ hours easily.

The presentation part turned out to be the easiest portion of the entire thing. Once the presentation was rolling, it came naturally to point out what the robot could do, how it accomplished each task, what design methodology went into said task, and what problems were encountered in each step of the project. Overall, it was agreed the project turned out extremely well, as did the presentation.