



<http://bob.joshashby.com>  
In Conjunction with:



<http://joshashby.com>

# Project: Bouncing Off Bumpers

Joshua Ashby  
Maurice Ashby  
Linux And Sci

<http://joshashby.com>  
[joshuaashby@joshashby.com](mailto:joshuaashby@joshashby.com)

April 19, 2010

# Contents

<b>1</b>	<b>Revisions</b>	<b>3</b>
<b>2</b>	<b>Introduction and Background</b>	<b>4</b>
<b>3</b>	<b>Frame</b>	<b>4</b>
3.1	Shape, Design and Material . . . . .	4
3.2	Steering . . . . .	4
3.3	Propulsion . . . . .	5
<b>4</b>	<b>Electronics</b>	<b>6</b>
4.1	Introduction . . . . .	6
4.2	Motor controller . . . . .	6
4.3	Power Node . . . . .	6
<b>5</b>	<b>Software</b>	<b>7</b>
5.1	Libraries - Introduction . . . . .	7
5.2	Digital Functions . . . . .	7
5.3	Analog Functions . . . . .	7
5.4	PWM Functions . . . . .	7
5.5	Robotics Functions . . . . .	7
5.6	Boot Functions . . . . .	7
<b>6</b>	<b>About the builders</b>	<b>8</b>
6.1	Joshua Ashby . . . . .	8
6.2	Maurice Ashby . . . . .	8
<b>7</b>	<b>Appendix</b>	<b>9</b>
7.1	Code . . . . .	9

## List of Figures

1	Example of the steering build used . . . . .	4
2	Example of the steering motor . . . . .	5
3	Example of the drive system used . . . . .	5

## List of Listings

1	The digital function library. . . . .	9
2	The analog function library. . . . .	10
3	The PWM function library. . . . .	11
4	The robotics function library. . . . .	19
5	The boot function library. . . . .	22

## 1 Revisions

March 10th/11th/12th, 2010 - Joshua Ashby

Added more to the electronics, Motor controller node part to include the new Quad Low-side V1.5.1 through Revision 3 boards.

April 19th, 2010 - Joshua Ashby

Added even more to the new sections, cleaning it up, and overall re-writing some parts for better understanding.

Rewrote software section also.

## Abstract

Bouncing off Bumpers, or BOB is a competition robot built by Joshua Ashby and his grandfather Maurice Ashby for the April 15th, 2009 Sparkfun Autonomous Vehicle Competition. It measures approximately 3 feet long by 2 feet wide by 2 feet tall; it weighs approximately 50 pounds without the battery and electronics. This paper will go into detail about the many systems involved in the build process of BOB, and provide insight into how many of these systems were designed, and the logic behind them (Please note that this paper is always going to be changing, and the data could easily change the day after the latest publishing). BOB has also competed in the 2010 Sparkfun AVC and won the Kill Switch award.

## 2 Introduction and Background

The electronics were designed by me (Joshua Ashby), and built by me along with the aid of my grandfather Maurice Ashby. As of March 10th, 2010 BOB is running on the newly designed and completed Generation 3 electronics. These electronics include the Taco quad-motor Motor controller board with a few minor additions to the board, along with several other monor boards. Over all the design process for the electronics have taken the longest as the motor controllers must be able to meet the demand of 10A per motor, as a result several generation of electronics have gone out the window.

The mechanics of the robot, which refers to the frame and body, the steering and related mechanisms, and the propulsion system were designed and built by us over a course of approximately 5 weeks, and has not had any major problems besides the replacement of the front motor.

## 3 Frame<sup>1</sup>

### 3.1 Shape, Design and Material

In order to build the frame in both a time and cost effective way, we choose to reuse some old square steel tubing that measures and has a wall thickness of. My grandfather had just enough laying around his shop to build a frame with. By using steel square tubing, and welding the joints, we were able to build a sturdy frame capable of carrying well over 100LBS over rough terrain.

The design of a three wheeled robot came after evaluating the cost of the wheels that would be used; to keep the cost down, only three wheels would be used. Because only three wheels would be used, the frame would have to be built in a fashion that did not promote tilting when the robot turns but still allow a large amount of room to build on top of. To accomplish this, an elongated pentagon design was created. The main drive wheel would be placed at the back of the robot in a triangle shaped portion of the frame, while the middle and front of the body would be a square shape, with two wheels in the front to steer with.

### 3.2 Steering

The steering for BOB was modeled after a car style steering system, where two wheels are connected via a rod which is in turn moved left or right to turn the wheels (Figure 1).



Figure 1: Example of the steering build used

---

<sup>1</sup>Please note, the frame was made to be cheap, and need little maintainance

TODO: Fix this description The build of this was accomplished by using two pre-built wheel casters, and welding on strips of quarter inch thick steel. These strips are approximately 6 inches long, and at the end that is not welded to the caster wheels, there is a hole drilled. Then connecting the two wheels via these strips, is a second pair of steel strips that are hooked up to the steering motor.

The steering motor is a 9.6V 10A drill motor that has been mounted with a right angle drive. This allows the motor to lay flat with the robot frame and still be able to turn the steering rod (Figure 2).



Figure 2: Example of the steering motor

One problem that we did not foresee while building the steering, is the ability to drive straight. Because the steering mechanism does not have a method of straightening its self out quickly and effectively, a new task is introduced to the electronics and programming until further improvements to the steering can be made. This may be accomplished by turning the front motor into an mechanism much like a servo through the use of programming and the ability of the motor controllers, however as of right now, thing has been done to take care of this problem.

### 3.3 Propulsion

Transferring power from the drill motor to the back drive wheel was one of the greatest technical difficulties we encountered. We started off by testing the idea of a friction drive. This style of drive has the motor running parallel to the wheel, and the output shaft using friction to turn the wheel. This worked great going downhill, but as soon as the drive had a load to pull, such as on flat ground, or uphill, the drive would start to slip.

Our second attempt was based off of a bike, just instead of a chain drive, we decided to do a belt drive as my grandfather had many of the needed parts. The motor was mounted perpendicular to the rotating axis of the wheel, and had a small 1.25 inch radius belt pulley on it. The wheel then had a larger, 2 inch radius, belt pulley on it. The two pulleys were connected via 8 inch diameter cogged belt. The axle for the motor is a milled axle that is supported by a bearing block at one end. The other end is tapered down to allow it to fit in the motor chuck. This also allows the motor to be disconnected from the axle, allowing the robot to be moved around with out power to the motor. (Figure 3).



Figure 3: Example of the drive system used

This drive system worked perfectly both downhill and uphill, and as a result it is the drive system currently used. The motor is the same as the steering motor, a recycled drill motor that is rated at 9.6V and 10A.

## 4 Electronics

### 4.1 Introduction

The electronics have always been a troublesome matter for this project and as this is being typed, the electronics still are providing issues, even with new designs.

The motors, which each draw 10A, must have easy to use, and cheap to build motor controllers, along with the ability to easily replace major parts while the robot is not in the Lab or near a soldering iron. This means the motor controllers can not be store bought, as all of the quality controllers that we can find are not only expensive, but also do not have common, easy to find parts. Instead to replace a part, the whole controller must be replaced most the time. Generation 1 electronics consisted of one micro-controller, and one and a half motor controllers. These electronics shorted out at the 2009 Sparkfun AVC competition, and as a result will only be used as both a comparison, and as a resource for what not to do on future generations.

### 4.2 Motor controller

As stated above, the use of store bought motor controllers is out of the question for use on BOB. They tend to be expensive, and typically must have the whole unit replaced if something burns out. Because of this, the motor controllers have been hand designed by us.<sup>2</sup>

The first version, which is the version that was used during the 2009 Sparkfun AVC competition, was designed to be very simple, and yet still provide the power that was needed for the motors. It consisted of a TIP125 PNP transistor driving a pair of paralleled IRF540N n-channel MOSFETs. At the time of their designing and building, we both were very new to MOSFETs and as a result the knowledge of how to hook the MOSFETs up, and the voltage required to drive them was unknown. This caused many problems, such as the MOSFETs not having enough voltage and amperes for them to fully close. This caused them to over heat, and also caused a massive power spike somewhere along the lines that burnt out the MOSFETs and transistor.

The new generation 2 motor controllers were designed to avoid these problems, along with begin to merge into a modular system as talked about in section Electronics: Introduction. These new motor controllers were called the Motor Nodes, and included two motor controllers, and a micro-controller that talked to the controllers. The micro-controller for generation 2 electronics was an Atmega328p which took care of both the sensors and the motor controllers. The basic design of using the transistors to switch the logic level were used, but with a few improvements such as temperature shut off for the MOSFETs, however it was discovered in late December 2009 that this method of driving the MOSFETs was not adequate enough, and as a result this method is no longer used. For this reason the Generation 2 electronics have been ditched and new Generation 3 electronics designed and put in place.

The new board consists of the main ATmega328P running at 5V with a 16MHz crystal. The I2C headers are broken out<sup>3</sup>, and the ATmega has 2 LEDs connected to PortD pins 3 and 5, also known as OC2B and OC0B. This allows for PWM debugging of any functions. Next the Atmega is connected to two TC4424CPA chips, one of these chips being on PD1 and PD2, OR1A, OR1B for PWM speed control, and the other chip being on pins PD2, PD4 for simple digital triggering of relays.

Along with these new MOSFET drivers, the MOSFET gates have current limiting resistors, and 12V zener diodes to prevent power spikes. The board also has a 10uF and 1000uF capacitor to help smooth out power drops from the motors.

This board, which is named Taco is in fact a general purpose MOSFET board, but has been designed in conjunction with BOB for use on him, however the only problem so far has been the traces not being big enough to carry the 10A and 20A from the motors. As a result the board also has several high amperage wires for the MOSFETs and motor outputs.

The I2C header is broken out which I2C on the Atmega328p is the analog pins 4 and 5, which means that the board can also be used for analog inputs, such as was used for the ultrasounds on BOB.

### 4.3 Power Node

The Power Node, as of Generation 2 electronics is simply a dead Node as one might call it. It has no intelligence, instead it's only function for generation 2 is to regulate and distribute 5V to all the boards. It does this via Molex connectors off of old ATX power supplies from computers. This same board design was used in the Generation 3 electronics also.

---

<sup>2</sup>Also it's a matter of which is funner to build and use.

<sup>3</sup>Which also provide two analog pins for the ultrasounds on BOB

## **5 Software**

### **5.1 Libraries - Introduction**

As of January 2010, BOB runs a custom written library set that takes care of everything from PWM and digital function to analog, ultrasounds, and calibration.

### **5.2 Digital Functions**

First up is the digital function library (Page: 9). The digital function library takes care of turning a pin on or off, which seems simple enough. While this task is quite trivial in code, those few extra lines tend to make the code look messy, which I don't like as much.

### **5.3 Analog Functions**

Unlike the digital code, reading from an analog pin takes some work. First you have to setup the registers, which in my case get me going at a prescaler of 128, interrupt driven, and left aligned bits. The left aligned results mean that I don't have to do fancy code and get 10bit results, which I don't need. instead I can simply read the ADCH register and get an 8 bit result. The next step is to read the results, or change the pin that I am reading from, both which take more code. As a result of all this, placing the analog functions inside of a nice, clean library makes sense (Page: 10).

### **5.4 PWM Functions**

Like the analog functions, the PWM functions that I needed were large, and very messy pieces of code. They consisted of several functions that setup the registers for the three PWM timers, and then several more to add in functions like ramp up and down functionality. (Page: 11).

### **5.5 Robotics Functions**

The robotics library takes care of most of the things that BOB needs to run. It houses the turn functions, and the filter that takes care of the ultrasound data. This filter is a custom rolling average with a few additions which smooth the data points out really well as the analog outputs on the ultrasounds are a little jumpy (Page: 19).

### **5.6 Boot Functions**

Finally the boot library takes care of setting everything up as the robot first starts, basically it's an implementation of a low level bios for BOB (Page: 22).



## **6 About the builders**

### **6.1 Joshua Ashby**

### **6.2 Maurice Ashby**

## 7 Appendix

All code and schematics are released under the Creative Commons Attribution-Noncommercial 3.0 United States License.

The code can be found online at github: <http://github.com/JoshAshby/Robotbob/tree/experimental>

### 7.1 Code

Listing 1: The digital function library.

```
1 //-----
2 //
3 /*
4  DIGITAL.h
5  2010 - Josh Ashby
6  joshuaashby@joshashby.com
7  http://joshashby.com
8  http://github.com/JoshAshby
9  freenode/#linuxandsci - JoshAshby
10 */
11 //-----
12 #include "adc.h"
13 #include "pwm.h"
14 #include "digital.h"
15 #include "boot.h"
16 #include "global.h"
17 #include "robotfunc.h"
18 //add the ability for it to auto detect which port based on what pin number you give
19 void portB_out(int pin, int value)
20 {
21     if (value == 0)
22     {
23         PORTB &= ~(1<<pin);
24     }
25     else
26     {
27         PORTB |= (1<<pin);
28     }
29 }
30 void portD_out(int pin, int value)
31 {
32     if (value == 0)
33     {
34         PORTD &= ~(1<<pin);
35     }
36     else
37     {
38         PORTD |= (1<<pin);
39     }
40 }
41 void out(char port, int pin, int value){
42     switch (port) {
43         case 'D':
44             if(value == 1){
45                 PORTD |= (1<<pin);
46             }
47             else {
48                 PORTD &= ~(1<<pin);
```

```

49         }
50         break;
51     case 'B':
52         if (value == 1){
53             PORTB |= (1<<pin);
54         }
55         else {
56             PORTB &= ~(1<<pin);
57         }
58         break;
59     }
60 }

```

Listing 2: The analog function library.

```

1  //-----
2  /*
3  ADC.c
4  2010 - Josh Ashby
5  joshuaashby@joshashby.com
6  http://joshashby.com
7  http://github.com/JoshAshby
8  freenode/#linuxandsci - JoshAshby
9  */
10 //-----
11 #include "adc.h"
12 #include "pwm.h"
13 #include "digital.h"
14 #include "boot.h"
15 #include "global.h"
16 #include "robotfunc.h"
17 ISR(ADC_vect)
18 {
19 }
20 void adc_start(void)
21 {
22     ADCSRA |= (1 << ADPS2)
23             | (1 << ADPS1)
24             | (1 << ADPS0); // Set ADC prescaler to 128 - 125KHz sample rate @ 16MHz
25     ADMUX |= (1 << REFS0); // Set ADC reference to AVCC
26     ADMUX |= (1 << ADLAR); // Left adjust ADC result to allow easy 8 bit reading
27     ADCSRA |= (1 << ADSC);
28     ADCSRA |= (1 << ADEN); // Enable ADC
29     ADCSRA |= (1 << ADIE); // Enable ADC Interrupt
30     sei();
31     ADCSRA |= (1 << ADSC); // Start A2D Conversions
32 }
33 void adc_stop(){
34     //stop the ADC
35     ADCSRA &= ~(1 << ADSC);
36 }
37 void adc_change(int chan){
38     //stop the ADC
39     ADCSRA &= ~(1 << ADSC);
40     //and now change the ADMUX bits to fit which channel
41     //you want to use, this should probably be replaced by a switch soon
42     switch (chan) {
43     case 0:
44         ADMUX &= ~(1 << MUX0)

```

```

45         & ~(1 << MUX1)
46         & ~(1 << MUX2)
47         & ~(1 << MUX3);
48     break;
49 case 1:
50     ADMUX |= (1 << MUX0);
51     ADMUX &= ~(1 << MUX1)
52             & ~(1 << MUX2)
53             & ~(1 << MUX3);
54     break;
55 case 2:
56     ADMUX &= ~(1 << MUX0);
57     ADMUX |= (1 << MUX1);
58     ADMUX &= ~(1 << MUX2)
59             & ~(1 << MUX3);
60     break;
61 case 3:
62     ADMUX |= (1 << MUX0)
63             | (1 << MUX1);
64     ADMUX &= ~(1 << MUX2)
65             & ~(1 << MUX3);
66     break;
67 case 4:
68     ADMUX &= ~(1 << MUX0)
69             & ~(1 << MUX1);
70     ADMUX |= (1 << MUX2);
71     ADMUX &= ~(1 << MUX3);
72     break;
73 case 5:
74     ADMUX |= (1 << MUX0);
75     ADMUX &= ~(1 << MUX1);
76     ADMUX |= (1 << MUX2);
77     ADMUX &= ~(1 << MUX3);
78     break;
79 case 6:
80     ADMUX &= ~(1 << MUX0);
81     ADMUX |= (1 << MUX1)
82             | (1 << MUX2);
83     ADMUX &= ~(1 << MUX3);
84     break;
85 case 7:
86     ADMUX |= (1 << MUX0)
87             | (1 << MUX1)
88             | (1 << MUX2);
89     ADMUX &= ~(1 << MUX3);
90     break;
91 case 8:
92     ADMUX &= ~(1 << MUX0)
93             & ~(1 << MUX1)
94             & ~(1 << MUX2);
95     ADMUX |= (1 << MUX3);
96     break;
97 }
98 ADCSRA |= (1 << ADSC);
99 }

```

Listing 3: The PWM function library.

```

2  /*
3  PWM.c
4  2010 - Josh Ashby
5  joshuaashby@joshashby.com
6  http://joshashby.com
7  http://github.com/JoshAshby
8  freenode/#linuxandsci - JoshAshby
9  */
10 //-----
11 #include "adc.h"
12 #include "pwm.h"
13 #include "digital.h"
14 #include "boot.h"
15 #include "global.h"
16 #include "robotfunc.h"
17 void pwm_setup_all(void){
18     TCCR0B |= (1<<CS00)
19             | (1<<CS01);
20     TCCR0A |= (1<<WGM00);
21
22     DDRD |= (1<<5);
23     DDRD |= (1<<6);
24
25     pwm_speed0A = 0;
26     pwm_value0A = 0;
27     pwm_value_old0A = 0;
28
29     pwm_speed0B = 0;
30     pwm_value0B = 0;
31     pwm_value_old0B = 0;
32
33     TCCR1B |= (1<<CS11)
34             | (1<<CS10);
35     TCCR1A |= (1<<WGM10);
36
37     DDRB |= (1<<1);
38     DDRB |= (1<<2);
39
40     pwm_speed1A = 0;
41     pwm_value1A = 0;
42     pwm_value_old1A = 0;
43
44     pwm_speed1B = 0;
45     pwm_value1B = 0;
46     pwm_value_old1B = 0;
47
48     TCCR2B |= (1<<CS22);
49     TCCR2A |= (1<<WGM20);
50
51     DDRD |= (1<<3);
52     DDRB |= (1<<3);
53
54     pwm_speed2A = 0;
55     pwm_value2A = 0;
56     pwm_value_old2A = 0;
57
58     pwm_speed2B = 0;
59     pwm_value2B = 0;
60     pwm_value_old2B = 0;

```

```

61 }
62 void pwm_setup0(void)
63 {
64     TCCR0B |= (1<<CS00)
65             | (1<<CS01);
66     TCCR0A |= (1<<WGM00);
67
68     DDRD |= (1<<5);
69     DDRD |= (1<<6);
70
71     pwm_speed0A = 0;
72     pwm_value0A = 0;
73     pwm_value_old0A = 0;
74
75     pwm_speed0B = 0;
76     pwm_value0B = 0;
77     pwm_value_old0B = 0;
78 }
79 void pwm0A(unsigned int value)//set the duty cycle on the PWM
80 {
81     TCCR0A |= (1<<COM0A1);
82     OCR0A = value;
83 }
84 void pwm0B(unsigned int value)//set the duty cycle on the PWM
85 {
86     TCCR0A |= (1<<COM0B1);
87     OCR0B = value;
88 }
89 //calling any of these will stop the processor for a short amount of time due to the delay
90 void pwm_ramp0A(unsigned int value, unsigned int speed)
91 {
92     if (value == 0) { //safe guard to prevent i from overflowing
93         pwm0A(0);
94     }
95     else {
96         if (value > pwm_value_old0A) { //determine if it should ramp up or down
97             TCCR0A |= (1<<COM0A1);
98             unsigned int i = pwm_value_old0A;
99             while (i <= value) { //ramp up
100                 OCR0A = i;
101                 i++;
102                 _delay_ms(speed);
103             }
104             pwm_value_old0A = value; //store the old pwm for autoramping
105         } else {
106             TCCR0A |= (1<<COM0A1);
107             unsigned int i = pwm_value_old0A;
108             while (i >= value) { //ramp down
109                 OCR0A = i;
110                 i--;
111                 _delay_ms(speed);
112             }
113         }
114         pwm_value_old0A = value; //store the old pwm for autoramping
115     }
116 }
117 void pwm_rampUp0A(unsigned int value, unsigned int speed)
118 {
119     TCCR0A |= (1<<COM0A1);

```

```

120     unsigned int i = pwm_value_old0A;
121     while (i<=value) {//ramp up
122         OCR0A=i;
123         i++;
124         _delay_ms(speed);
125     }
126     pwm_value_old0A = value;//store the old pwm for autoramping
127 }
128 void pwm_rampDown0A(unsigned int value , unsigned int speed)
129 {
130     if (value == 0) {//safe gaurd to prevent i from over flowing
131         pwm0A(0);
132     }
133     else {
134         TCCR0A |= (1<<COM0A1);
135         unsigned int i = pwm_value_old0A;
136         while (i>=value) {//ramp down
137             OCR0A=i;
138             i--;
139             _delay_ms(speed);
140         }
141     }
142     pwm_value_old0A = value;//store the old pwm for autoramping
143 }
144
145
146 void pwm_ramp0B(unsigned int value , unsigned int speed)
147 {
148     if (value == 0) {//safe gaurd to prevent i from over flowing
149         pwm0B(0);
150     }
151     if (value > pwm_value_old0B){//determine if it should ramp up or down
152         TCCR0A |= (1<<COM0B1);
153         unsigned int i = pwm_value_old0B;
154         while (i<=value) {//ramp up
155             OCR0B=i;
156             i++;
157             _delay_ms(speed);
158         }
159         pwm_value_old0B = value;//store the old pwm for autoramping
160     } else {
161         TCCR0A |= (1<<COM0B1);
162         unsigned int i = pwm_value_old0B;
163         while (i>=value) {//ramp down
164             OCR0B=i;
165             i--;
166             _delay_ms(speed);
167         }
168         pwm_value_old0B = value;//store the old pwm for autoramping
169     }
170 }
171 void pwm_rampUp0B(unsigned int value , unsigned int speed)
172 {
173     TCCR0A |= (1<<COM0B1);
174     unsigned int i = pwm_value_old0B;
175     while (i<=value) {//ramp up
176         OCR0B=i;
177         i++;
178         _delay_ms(speed);

```

```

179     }
180     pwm_value_old0B = value; //store the old pwm for autoramping
181 }
182 void pwm_rampDown0B(unsigned int value, unsigned int speed)
183 {
184     if (value == 0) {//safe gaurd to prevent i from over flowing
185         pwm0B(0);
186     }
187     TCCR0A |= (1<<COM0B1);
188     unsigned int i = pwm_value_old0B;
189     while (i>=value) {//ramp down
190         OCR1B=i;
191         i--;
192         _delay_ms(speed);
193     }
194     pwm_value_old0B = value; //store the old pwm for autoramping
195 }
196
197 //-----
198 void pwm_setup1(void)
199 {
200     TCCR1B |= (1<<CS11)
201             | (1<<CS10);
202     TCCR1A |= (1<<WGM10);
203
204     DDRB |= (1<<1);
205     DDRB |= (1<<2);
206
207     pwm_speed1A = 0;
208     pwm_value1A = 0;
209     pwm_value_old1A = 0;
210
211     pwm_speed1B = 0;
212     pwm_value1B = 0;
213     pwm_value_old1B = 0;
214 }
215 void pwm1A(int value)//set the duty cycle on the PWM
216 {
217     TCCR1A |= (1<<COM1A1);
218     OCR1A = value;
219 }
220 void pwm1B(unsigned int value)//set the duty cycle on the PWM
221 {
222     TCCR1A |= (1<<COM1B1);
223     OCR1B = value;
224 }
225 //calling any of these will stop the processor for a short amonnt of time due to the delay
226 void pwm_ramp1A(int value, int speed)
227 {
228     if (value == 0) {//safe gaurd to prevent i from over flowing
229         pwm1A(0);
230     }
231     else {
232         if (value > pwm_value_old1A){//determine if it should ramp up or down
233             TCCR1A |= (1<<COM1A1);
234             unsigned int i = pwm_value_old1A;
235             while (i<=value) {//ramp up
236                 OCR1A=i;
237                 i++;

```



```

238         _delay_ms(speed);
239     }
240     pwm_value_old1A = value; //store the old pwm for autoramping
241 } else {
242     TCCR1A |= (1<<COM1A1);
243     unsigned int i = pwm_value_old1A;
244     while (i>=value) //ramp down
245         OCR1A=i;
246         i--;
247         _delay_ms(speed);
248     }
249 }
250     pwm_value_old1A = value; //store the old pwm for autoramping
251 }
252 }
253 void pwm_rampUp1A(unsigned int value, unsigned int speed)
254 {
255     TCCR1A |= (1<<COM1A1);
256     unsigned int i = pwm_value_old1A;
257     while (i<=value) //ramp up
258         OCR1A=i;
259         i++;
260         _delay_ms(speed);
261     }
262     pwm_value_old1A = value; //store the old pwm for autoramping
263 }
264 void pwm_rampDown1A(unsigned int value, unsigned int speed)
265 {
266     if (value == 0) //safe gaurd to prevent i from over flowing
267         pwm1A(0);
268     }
269     else {
270         TCCR1A |= (1<<COM1A1);
271         unsigned int i = pwm_value_old1A;
272         while (i>=value) //ramp down
273             OCR1A=i;
274             i--;
275             _delay_ms(speed);
276         }
277     }
278     pwm_value_old1A = value; //store the old pwm for autoramping
279 }
280
281
282 void pwm_ramp1B(unsigned int value, unsigned int speed)
283 {
284     if (value == 0) //safe gaurd to prevent i from over flowing
285         pwm1B(0);
286     }
287     if (value > pwm_value_old1B) //determine if it should ramp up or down
288         TCCR1A |= (1<<COM1B1);
289         unsigned int i = pwm_value_old1B;
290         while (i<=value) //ramp up
291             OCR1B=i;
292             i++;
293             _delay_ms(speed);
294         }
295     pwm_value_old1B = value; //store the old pwm for autoramping
296 } else {

```

```

297     TCCR1A |= (1<<COM1B1);
298     unsigned int i = pwm_value_old1B;
299     while (i>=value) {//ramp down
300         OCR1B=i;
301         i--;
302         _delay_ms(speed);
303     }
304     pwm_value_old1B = value;//store the old pwm for autoramping
305 }
306 }
307 void pwm_rampUp1B(unsigned int value , unsigned int speed)
308 {
309     TCCR1A |= (1<<COM1B1);
310     unsigned int i = pwm_value_old1B;
311     while (i<=value) {//ramp up
312         OCR1B=i;
313         i++;
314         _delay_ms(speed);
315     }
316     pwm_value_old1B = value;//store the old pwm for autoramping
317 }
318 void pwm_rampDown1B(unsigned int value , unsigned int speed)
319 {
320     if (value == 0) {//safe gaurd to prevent i from over flowing
321         pwm1B(0);
322     }
323     TCCR1A |= (1<<COM1B1);
324     unsigned int i = pwm_value_old1B;
325     while (i>=value) {//ramp down
326         OCR1B=i;
327         i--;
328         _delay_ms(speed);
329     }
330     pwm_value_old1B = value;//store the old pwm for autoramping
331 }
332 //-----
333 void pwm_setup2(void)
334 {
335     TCCR2B |= (1<<CS22);
336     TCCR2A |= (1<<WGM20);
337
338     DDRD |= (1<<3);
339     DDRB |= (1<<3);
340
341     pwm_speed2A = 0;
342     pwm_value2A = 0;
343     pwm_value_old2A = 0;
344
345     pwm_speed2B = 0;
346     pwm_value2B = 0;
347     pwm_value_old2B = 0;
348 }
349 void pwm2A(unsigned int value)//set the duty cycle on the PWM
350 {
351     TCCR2A |= (1<<COM2A1);
352     OCR2A = value;
353 }
354 void pwm2B(unsigned int value)//set the duty cycle on the PWM
355 {

```

```

356     TCCR2A |= (1<<COM2B1);
357     OCR2B = value;
358 }
359 //calling any of these will stop the processor for a short amount of time due to the delay
360 void pwm_ramp2A(unsigned int value, unsigned int speed)
361 {
362     if (value == 0) {//safe guard to prevent i from over flowing
363         pwm2A(0);
364     }
365     else {
366         if (value > pwm_value_old2A){//determine if it should ramp up or down
367             TCCR2A |= (1<<COM2A1);
368             unsigned int i = pwm_value_old2A;
369             while (i<=value) {//ramp up
370                 OCR2A=i;
371                 i++;
372                 _delay_ms(speed);
373             }
374             pwm_value_old2A = value;//store the old pwm for autoramping
375         } else {
376             TCCR2A |= (1<<COM2A1);
377             unsigned int i = pwm_value_old2A;
378             while (i>=value) {//ramp down
379                 OCR2A=i;
380                 i--;
381                 _delay_ms(speed);
382             }
383         }
384         pwm_value_old2A = value;//store the old pwm for autoramping
385     }
386 }
387 void pwm_rampUp2A(unsigned int value, unsigned int speed)
388 {
389     TCCR2A |= (1<<COM2A1);
390     unsigned int i = pwm_value_old2A;
391     while (i<=value) {//ramp up
392         OCR2A=i;
393         i++;
394         _delay_ms(speed);
395     }
396     pwm_value_old2A = value;//store the old pwm for autoramping
397 }
398 void pwm_rampDown2A(unsigned int value, unsigned int speed)
399 {
400     if (value == 0) {//safe guard to prevent i from over flowing
401         pwm2A(0);
402     }
403     else {
404         TCCR2A |= (1<<COM2A1);
405         unsigned int i = pwm_value_old2A;
406         while (i>=value) {//ramp down
407             OCR2A=i;
408             i--;
409             _delay_ms(speed);
410         }
411     }
412     pwm_value_old2A = value;//store the old pwm for autoramping
413 }
414

```

```

415
416 void pwm_ramp2B(unsigned int value , unsigned int speed)
417 {
418     if (value == 0) {//safe gaurd to prevent i from over flowing
419         pwm2B(0);
420     }
421     if (value > pwm_value_old2B){//determine if it should ramp up or down
422         TCCR2A |= (1<<COM2B1);
423         unsigned int i = pwm_value_old2B;
424         while (i<=value) {//ramp up
425             OCR2B=i;
426             i++;
427             _delay_ms(speed);
428         }
429         pwm_value_old2B = value;//store the old pwm for autoramping
430     } else {
431         TCCR2A |= (1<<COM2B1);
432         unsigned int i = pwm_value_old2B;
433         while (i>=value) {//ramp down
434             OCR2B=i;
435             i--;
436             _delay_ms(speed);
437         }
438         pwm_value_old2B = value;//store the old pwm for autoramping
439     }
440 }
441 void pwm_rampUp2B(unsigned int value , unsigned int speed)
442 {
443     TCCR2A |= (1<<COM2B1);
444     unsigned int i = pwm_value_old2B;
445     while (i<=value) {//ramp up
446         OCR2B=i;
447         i++;
448         _delay_ms(speed);
449     }
450     pwm_value_old2B = value;//store the old pwm for autoramping
451 }
452 void pwm_rampDown2B(unsigned int value , unsigned int speed)
453 {
454     if (value == 0) {//safe gaurd to prevent i from over flowing
455         pwm2B(0);
456     }
457     TCCR2A |= (1<<COM2B1);
458     unsigned int i = pwm_value_old2B;
459     while (i>=value) {//ramp down
460         OCR2B=i;
461         i--;
462         _delay_ms(speed);
463     }
464     pwm_value_old2B = value;//store the old pwm for autoramping
465 }

```

---

Listing 4: The robotics function library.

---

```

1 #include "adc.h"
2 #include "pwm.h"
3 #include "digital.h"
4 #include "boot.h"
5 #include "global.h"

```

```

6 #include "robotfunc.h"
7 #include <util/delay.h>
8 //-----
9 /*
10 robotfunc.c
11 2010 - Josh Ashby
12 joshuaashby@joshashby.com
13 http://joshashby.com
14 http://github.com/JoshAshby
15 freenode/#linuxandsci - JoshAshby
16 */
17 //-----
18
19 void turn_left(void){
20     out('D', 4, 1);
21     _delay_ms(5);
22     pwm1B(255);
23     _delay_ms(200);
24     pwm1B(0);
25     out('D', 4, 0);
26 }
27 void turn_right(void){
28     pwm1B(255);
29     _delay_ms(150);
30     pwm1B(0);
31 }
32 void stop(void){
33     pwm0A(0);
34     pwm0B(0);
35     pwm1A(0);
36     pwm1B(0);
37     pwm2A(0);
38     pwm2B(0);
39     out('D', 2, 1);
40     error(1);
41     out('D', 4, 0);
42     out('D', 5, 0);
43 }
44 void calibrate(void){
45     adc_change(5);
46     _delay_ms(20);
47     adc = ADCH;
48     for (j = 0; j <= 20; j++){
49         if (ADCH > average + 100)
50         {
51             adc = (ADCH/2) + (average/2);
52         }
53         if (ADCH < average - 100){
54             adc = (ADCH/2) + (average/2);
55         }
56         rollAverage[j] = adc;
57     }
58     for (j = 0; j <= 20; j++){
59         average += rollAverage[j];
60     }
61     average = average/18;
62     base = average;
63     for (j = 0; j <= 20; j++){
64         rollAverage[j] = 0;

```

```

65     }
66 }
67 int ultrasound_filter(int pin){
68     /*simple filter that works quite well, it simply
69     smooths out the ADC data from the ultrasounds
70     if the ADCH data is out of range, it will divide
71     it by two, and then add the average divided by two*/
72     adc_change(pin);
73     _delay_ms(20);
74     adc = ADCH;
75     for (j = 0; j <= 30; j++){
76         if (ADCH > average + 100)
77         {
78             adc = (ADCH/2) + (average/2);
79         }
80         if (ADCH < average - 100){
81             adc = (ADCH/2) + (average/2);
82         }
83         rollAverage[j] = adc;
84     }
85     for (j = 0; j <= 30; j++){
86         average += rollAverage[j];
87     }
88     average = average/30;
89     return average;
90 }
91 void ultrasound_test(void){
92     if (ultrasound_filter(4) >= base) {
93         out('D', 2, 0);
94         pwm2B(ultrasound_filter(5));
95     } else {
96         out('D', 2, 1);
97         pwm2B(ultrasound_filter(4));
98     }
99 }
100 void test_turn(void){
101     out('B', 2, 1);
102     _delay_ms(200);
103     out('B', 2, 0);
104     _delay_ms(200);
105     out('D', 4, 1);
106     _delay_ms(500);
107     out('B', 2, 1);
108     _delay_ms(200);
109     out('B', 2, 0);
110     _delay_ms(500);
111     out('D', 4, 0);
112     _delay_ms(500);
113 }
114 void test_motor(void){
115     pwm_ramp1A(255, 10);
116     _delay_ms(2000);
117     pwm_ramp1A(1, 0);
118     pwmlA(0);
119     _delay_ms(500);
120     out('D', 5, 1);
121     _delay_ms(500);
122     pwm_ramp1A(255, 10);
123     _delay_ms(2000);

```

```

124     pwm_ramp1A(1, 0);
125     pwm1A(0);
126     _delay_ms(500);
127     out('D', 5, 0);
128 }

```

---

Listing 5: The boot function library.

```

1  #include "adc.h"
2  #include "pwm.h"
3  #include "digital.h"
4  #include "boot.h"
5  #include "global.h"
6  #include "robotfunc.h"
7  #include <util/delay.h>
8
9  //-----
10 /*
11  Boot.c
12  2010 – Josh Ashby
13  joshuaashby@joshashby.com
14  http://joshashby.com
15  http://github.com/JoshAshby
16  freenode/#linuxandsci – JoshAshby
17  */
18  //-----
19  //add a basic bios that will take, start the ADC
20  //calibrate the sensors to what value they should try to stay at
21  //also go through and make sure everything is working from what it
22  //can tell if there is an error then it will blink the status led
23  void all_good(){//turn the status led on
24      out('D', 3, 1);
25  }
26  void oh_crap(){//status led off
27      out('D', 3, 0);
28  }
29  void error(int type){//blink the status led if there is an error
30      switch (type) {
31          case 0:
32              out('D', 3, 1);
33              _delay_ms(500);
34              out('D', 3, 0);
35              _delay_ms(500);
36              break;
37          case 1:
38              pwm_ramp2B(255, 10);
39              pwm_ramp2B(1, 10);
40              break;
41          case 2:
42              pwm_ramp2B(255, 50);
43              pwm_ramp2B(0, 10);
44              break;
45      }
46  }
47  void bios(){
48      DDRD |= (1<<2);//LED power
49      DDRD |= (1<<3);//LED Status
50      DDRD |= (1<<4);//relay back
51      DDRD |= (1<<5);//relay front

```

```
52 | out('D', 2, 1); //CPU power LED
53 | pwm_setup_all();
54 | adc_start(); //because we're using interrupts ADCH will auto update
55 | calibrate();
56 | all_good();
57 | }
```

---