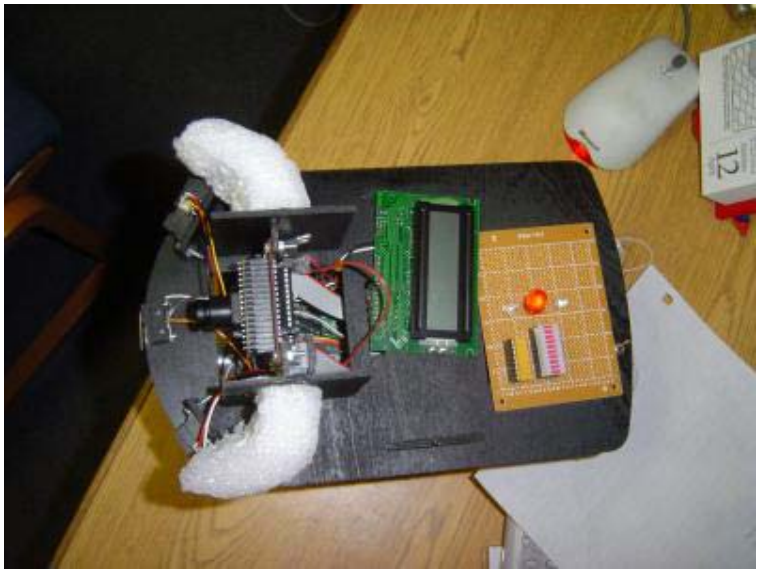
University of Florida
EEL 5666
Intelligent Machine Design Lab

Anthony Huereca
Spring 2004

# Sbob
# (Señor Bob)



# Table of Contents

## Abstract

This paper will describe and lay out the development of Sbob, an intelligent autonomous robot. Sbob is a bull robot that is designed to ram red objects placed in front of him. He includes feedback as to what mode he is currently in, as well as distance measurement with sonar to track how close he gets to the object before it is moved out of the away. This paper will explain the various sensors used to implement these behaviors and the process of building the robot over the course of the semester.

## Executive Summary

Sbob is a robot that mimics the actions and behaviors of a real bull used for bull fighting. It wanders around at random avoiding obstacles until a red object is placed in its path. It then tries to ram into it, while also keeping track of the closest distance that he achieved before the red object was pulled away. He can then determine how good the user was at bull fighting. For color recognition I decided to use the CMUcam as it has been used in a previous robots to determine colors and object tracking. For distance measurement I went with a sonar sensor, as they are the most accurate. Sbob uses the sonar sensor to discover that an object is in its path, and then uses the CMUcam to determine if the object is red. If it is, then Sbob begins tracking how far away it is and moves toward the object. This will continue until Sbob either hits the object and turns in a circle to celebrate its victory, or else the object is lifted up or moved out of the way and Sbob will have to try again. It detects successful attacks through a bump switch located at the front of his nose. Sbob also changes its attacking behavior to make it harder on the matador based on the number of misses. Obstacle avoidance is achieved via the sonar sensor located at the front and two angled IR sensors also at the front.

## Introduction

Sbob grew out of the idea of a ramming robot. Louis, a TA for this semester, suggested color analysis for use as the special sensor, and hence make it into a bull that only attempts to run into red objects. I continued on this idea with a distance measuring system to see how close the robot could travel to the object before the "matador" pulled it away.

Sbob could be used in a variety of ways in real world applications. It can be used as a toy for children who watch bull fighting on TV but whose parents understandably do not want them to try it out on a real bull. A much larger version of this robot could even be used as practice bulls so that matadors could safely practice their bull fighting skills, as well as make it much more humane since the sport of bullfighting kills the bull to end the competition. It would also be interesting to replace the annual "Running of the Bulls" in San Fermin, Spain with a few dozen of these robotic bulls, and have the robots chase people who are wearing red through the streets. This might conflict with some of Asimov's Laws of Robotics however.

## Integrated System

The processor that I chose was the Atmel ATMega128 because it came highly recommended by the TA's. It is a good chip with over 50 I/O pins, including two UART devices, two 16-bit counters, 8 interrupt pins, and 8 Analog-to-Digital pins. I then used the STK501 development board, which fits on top of the STK500 development board as an add-on piece of hardware. I originally had planned on buying only the STK500, but the STK501 is required to use the ATMega128 chip.

Figure 1: STK501 on top of STK500 with LCD Attached

The STK500 and STK501 boards are very user friendly from a hardware standpoint. They include port headers, two RS232 serial ports, LED lights for debugging, and plenty of ground and power pins for attaching devices. The large disadvantage however is that the development boards are quite expensive, and if I had to do it over again I would have gone with a cheaper board.

The programming aspect was done with AVR-GCC. Using C as the programming language made programming behaviors much easier, and AVR-GCC is a powerful compiler with support for strings and some advanced mathematics. I wrote the code in Programmer's Notepad, and then used AVR Studio to download the compiled hex file onto the microprocessor. The STK500 uses a serial cable to connect the computer to the processor, so no extra hardware was needed. Another advantage of using AVR-GCC is there are many coding examples and helpful websites for it. Most of my code was written from scratch, but having examples was useful in looking up how to call various functions and in just getting started initially.

The code was written in sections, and each device was tested and programmed separately before integrating them together. This helped with debugging, which is a crucial skill for this course.

Power was an issue in the beginning since the board requires at least 10 volts of power, but was accomplished by putting an eight pack and two pack of AA batteries in series with each other and then hot gluing the two packs together. This produced a voltage of about 12.5 volts using 10 NiMH batteries. A toggle switch was then added to turn the robot on and off, with leads going to the board power, servo voltage regulator, and CMUcam.

A LCD was also required for everyone's robot, which helped immensely with debugging problems. I was able to output the various values of registers and variables I had created to the LCD to ensure that I was getting the data I was expecting. It also came in useful to output where exactly I was in the program at any particular moment, and could then be used to figure out the problem areas where it might be getting stuck or behaving badly. After some initial coding work, writing to it was as easy as a regular C print statement.

## Mobile Platform

The platform was created out of the balsa wood that IMDL provides and then cut out on the T-Tech machine. It was designed in AutoCAD 2000, with which I had no previous experience with. A crash course lesson was given in class however, and I was able to pick up quite easily.

The hardest part of the platform design is to think about all the issues that will come up as sensors and devices are added. It has to be designed to have enough room to put the devices in the places that need it. Measuring the exact dimensions of a device and looking at datasheet diagrams is crucial in this part of construction, and

having rough sketches of what it should look like also help.

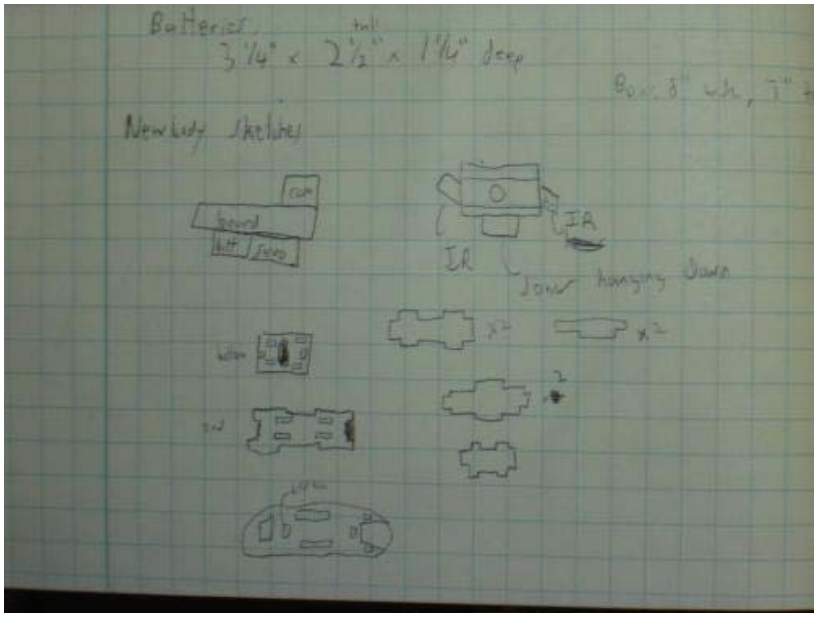



Figure 2: Initial Body Design Sketches

My design was fairly straightforward, consisting of a lower level for the batteries and servo motors, and then an upper box to hold the development board. The AutoCad drawings for the top and sides can be seen below:

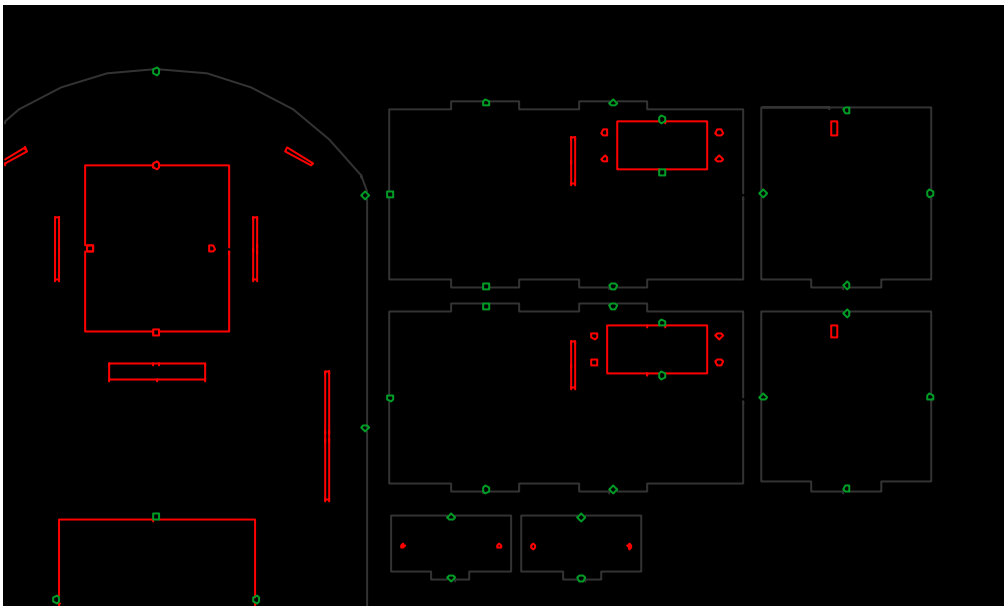


Figure 3: AutoCad Drawings

It took me two revisions to get the platform I wanted, as I had to add some holes for the LED cable and power switch, and make the bottom holes larger for the servo's. I wanted to make it look like at least somewhat like a bull, and thus created a

“head” for the camera and created two horns that I carved out of Styrofoam. It is far better to make a platform too large than too small to give some leeway for wires and any miscalculations.

I used a simple furniture glide piece for the third contact point, which was held with hot glue, and used Velcro to hold the batteries in place. The sonar was glued into place in the front with wood glue, as was the bump sensor. Everything else was bolted in with screws and then fit snuggly into slots in the wood. Wood glue was used to keep the platform together. Finally I put a coat of black spray paint on the wood to give it a more bull like look.

## Actuation

I used two hacked servos for movement of the robot. They were BP148N standard ball bearing servos with 47 oz.in of torque that were very easy to hack to get continuous motion out of them. I used the Fast PWM signal on Timer0 to generate the PWM signal for the servos, which came out of Port B, pins 6 & 7. I had some initial problems because I assumed that my chip was running at a different clock speed than it actually was, which affected the register values I needed. I then ran into a problem with my board crashing and resetting itself when I changed directions, which I found out was because I needed the power voltage to the servo to be regulated. So I attached a five volt regulator with a 470 capacitor on the input and a 22uF capacitor on the output, and finally had a moving robot.

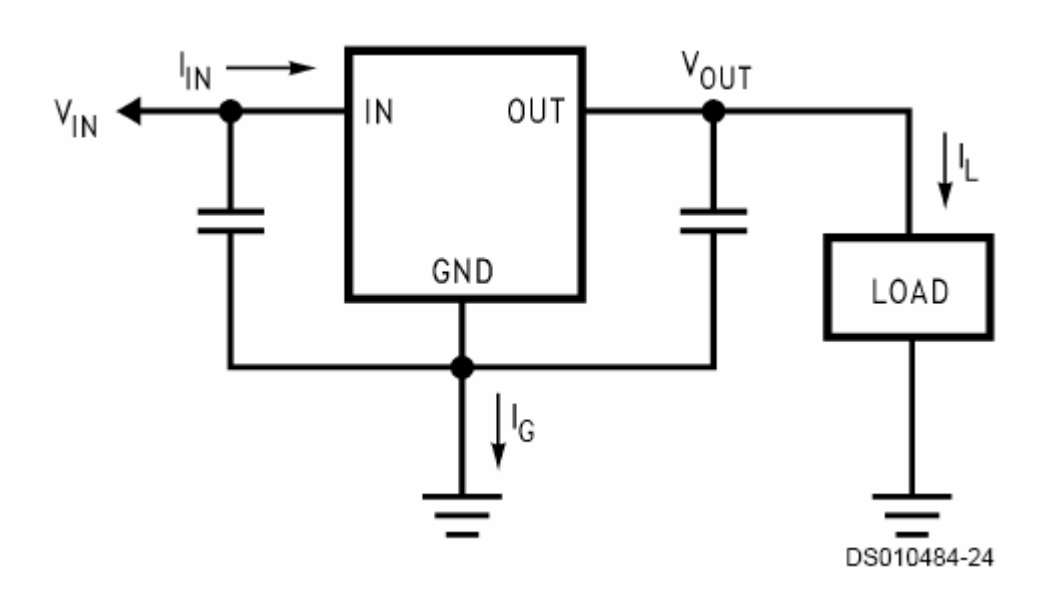



Figure 4: LM7805 Voltage Regulator circuit

In hindsight I might have wanted to use two motors instead of servo's to get more speed, in order to make it more challenging to try to avoid it hitting the red object.

For wheels I used two 2.55" diameter plastic wheels that attached directly to the servo and used rubber bands for traction. They performed very well.

## Sensors

I used three different types of sensors for obstacle avoiding. The first are two IR sensors, which were used to detect objects off to the side, so that Sbob could avoid them before it brushed along side it. A sonar system was then used to detect objects in front of the robot to avoid, as well as for measuring distances. Finally a simple lever switch was used to detect hits and accidental collisions at the front of Sbob.

The bump switch is a lever switch purchased from RadioShack that is pulled high normally and goes to ground when it is pushed in. It is then tied to External Interrupt 0 on Port E Pin 1. It was debounced in software to prevent extraneous interrupts.

I used the Sharp GP2D12 IR sensors, which were then connected to Port F, Pins 0 & 1 to convert the analog voltage to a digital value using the Analog to Digital

Converter. As an object becomes closer, the voltage rises in an exponential fashion. I was then able to use this value to compare against an experimentally found hard coded value to determine if the robot needed to turn to avoid an object.

The sonar device I used was the Devantech SRF04 ultrasonic rangefinder. It is fairly simple to operate, and works by calculating the time between a pulse being sent out to the sonar device, and then a rising pulse read in by the processor. I used the 16-bit Timer3 to send out a 15-uSec pulse, and then used the input capture unit to record the timer value when the output pulse was received and issue an interrupt. I used a running average of (3*old_average+new_value)/4 to help maintain a steady value that isn't affected by random jumps in readings, and also compared each new value against the previous value and threw out any readings that suddenly jumped. I found that the device is remarkably stable, and will give the same readings at the same distance consistently. It is also very accurate, which came in useful for calculating the distances to the object.

The closer an object is, the less time that passes by before the input capture unit is activated. The table and chart below show the very linear operation of this device, which was useful then in being able to convert the timer values into inches.

| Inches | Timer Value |
|---|---|
| 1 | 313 |
| 2 | 361 |
| 3 | 443 |
| 4 | 514 |
| 5 | 593 |
| 6 | 650 |
| 7 | 735 |
| 8 | 795 |
| 9 | 885 |
| 10 | 950 |
| 11 | 1042 |
| 12 | 1136 |
| 13 | 1201 |
| 14 | 1275 |
| 15 | 1346 |
| 16 | 1444 |
| 17 | 1496 |

| | |
|---|---|
| 18 | 1565 |
| 19 | 1652 |
| 20 | 1728 |
| 21 | 1813 |
| 22 | 1889 |
| 23 | 1975 |
| 24 | 2001 |
| 27 | 2257 |
| 30 | 2488 |
| 33 | 2706 |
| 36 | 2974 |

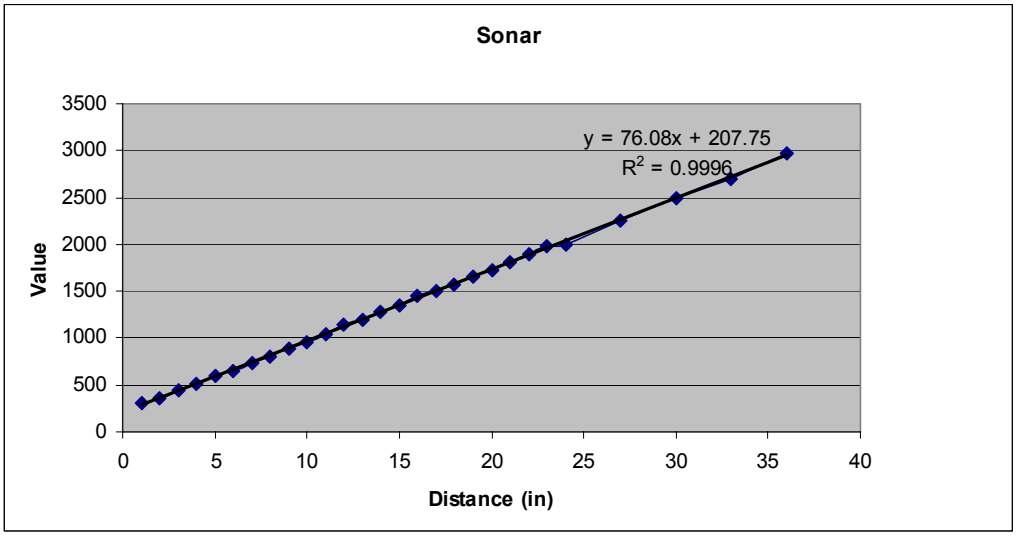



Figure 5: Sonar Readings

The special sensor is the CMUcam, which is used to determine the color of the object placed in front of Sbob. The special sensor report fully expands on the operation and theory behind this device, but it works via the serial port on the board and has a command set to get back data from the camera. I found that it works fairly well, especially in bright lighting conditions. Thanks to new some new lens filters, the red saturation that previous reports commented on is almost a complete non-issue now. It does have trouble correctly recording some colors such as blue however, which looked more like a very spotty white.

## Behaviors

My goal with this robot was to make it as "bull-like" as possible and to have some fun with it as well. As a computer engineer out of the CISE department this was

my favorite part of the entire process, since once the hardware is done, the robots behaviors are entirely left up to the imagination to make as simple or as complicated as you want.

At start up, several initialization functions are called to set up the LCD, servos, IR sensors, sonar, and camera. If an object is held within 6” of Sbob at start up, it attempts to dynamically calculate the camera values, time lag, and maximum viewing distance of the camera. Otherwise it uses the default values configured which work very well in the IMDL lab but not as well in other places, such as my dorm room. There is more detail about this in the CMUcam special sensor report.

Once the initialization is done, then it sits and waits for the front bump switch is pressed to begin moving. This is a very good idea since it allows you to not have to worry about the robot running away while programming it with a cord attached to the computer.

Sbob then begins basic obstacle avoidance. If the IR sensors sense something above their threshold, then it will turn to avoid it until it no longer senses anything. If the bump switch is hit while not in attack mode, it will treat it as an accidental collision and reverse and rotate to avoid the object while a white LED is lit to symbolize it surrendering. If the sonar detects an object below its threshold, then it will slow down or stop (depending on how far away the object is) and if it is not red, then will rotate for a time and then continue forward again.

Red (or the color being looked for) is detected by calling the track color command (TC) and looking at the confidence value. If it is above 30, then the object is determined to be red, and it begins the charge routine that varies based on the number of misses. If there are no misses, then it simply speeds up a little bit and continues on a straight line until it either hits the object, or it is pulled out of the way. It detects a miss by using the confidence value from the TC command and finding

when it falls below 20.

After the first miss, it slows down before until just before it hits the object, at which point it speeds back up in an effort to fake out the matador.

The third time it will begin tracking the red object and try to follow it as it moves around by looking at the medium X values from the TC commands.

Finally the fourth time it varies it's speed while also trying to follow the red object.

If it misses again, then it goes into a "sleep" mode since I figure a real bull would be tired of chasing all that red. It gets out of it by waving an object in front of it and then it begins fresh again.

## Experimental Layout and Results

Many "mini" experiments were performed during the creation process to make the development process run smoothly. The code for each sensor and device was written separately on its own and tested thoroughly before combining it with other parts of the program. This ensured that that the two major demos of the course, obstacle avoidance and the final demo, were programmed smoothly and I could focus on the grand overall behavior scheme instead of fixing buggy sensor code.

In the beginning I had planned on using the mean color value of the camera image to detect when a red object was placed in front of Sbob by taking the mean color value and seeing if the red channel value rose as an object was placed in front of it. This did not work out very well as the object had to be held very close to the camera and many non-red objects would also cause the value to rise. I then played around with the TC command more and found that it worked very well and the confidence value was a very good indicator of detecting color.

## Conclusions

This has been the most time consuming, frustrating, expensive, and stressful class I have ever taken. It has also been by far the most rewarding, cool, and interesting class that I will probably ever take. I had been looking forward to this course ever since hearing Dr. Schwartz talk about it in Digital Logic 3701 and it has lived up to its expectations of being a fun but very challenging course. I was able to accomplish all the goals I had set out for my robot to do, and I am very happy in how it turned out. All things red fear the presence of this robotic bull. I learned a lot and it has been the best possible kind of experience, one in which is hands on and teaches that the real world does not behave like theory says it should. Louis, William, and Max were awesome TA's and I'd also like to thank Dr. Arroyo and Dr. Schwartz for their expertise and advice.

Future things I would like to implement include the dynamic color configuration as explained in the CMUcam sensor report. I would also like to add an "avoid" color that if Sbob saw it, he would go in reverse and essentially try to stay away from it.

Things I would do differently if I had to start over include using motors instead of servos to make it more challenging for the matador.

I learned many useful things in the course of building Sbob that can't be found in any textbook. At the first sign of something acting up, be sure to check the battery level. Or if the board will not turn on, make sure that the batteries are not low. Also don't solder battery leads while they're attached to the battery pack. I melted two metal connector springs on two different packs this way, including one that started smoking quite heavily. I finally learned my lesson on the third one though. Everything also takes three times as long to do as it would seem it should. A person cannot afford to procrastinate in this course and with that in mind, do not take this class with any

other time intensive course. Debugging skills are incredibly important. Make sure all the code is developed separately and that each part works before putting them together.

Finally, I created a website to provide more pictures and details about the creation of Sbob, which can be found at http://binaryfusion.net/sbob/

## Documentation


Atmel ATMega128 Documentation
http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

AVR STK501 Documentation
http://www.atmel.com/dyn/resources/prod_documents/DOC2491.PDF

AVR STK500 Documentation
http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf

AVRFreaks
http://www.avrfreaks.com/

AVRCalc
http://www.avrfreaks.com/Tools/showtools.php?ToolID=292

AVR GCC Library Reference
http://www.avrfreaks.net/AVRGCC/libcref.php

AVR Bugs
http://claymore.engineer.gvsu.edu/egr326/AtmelAVR

Sharp GP2D12 IR Reference Sheet
http://www.junun.org/MarkIII/datasheets/GP2D12_15.pdf

Devantech SRF04 Ultrasonic Range Finder Reference Sheet
http://www.robot-electronics.co.uk/htm/srf04tech.htm

Circuit Diagram for Voltage Regulator:
http://www.national.com/ds/LM/LM341.pdf

Servo PWM Control
http://handyboard.com/faq/display.php?key=dcservo

Kristen Allen's Wait, LCD and A/D Conversion Code

# Appendix

**Parts:**

**Some comments:**

While the STK500 and STK501 are very nice boards, in the end it is not worth paying almost double what you can get with the Mavric II board.

Acroname is more expensive, but all the parts I ordered from them came with very nice custom made manuals and were the most impressive of all the websites. It's not really worth an extra $3-$4, but it is impressive.

Also I originally ordered the CMUcam from Acroname, but it comes as a "kit" which means it comes in a lot of little pieces that need to be put together. I returned it and bought a pre-assembled and tested one from Seattle Robotics.

**Board: STK500 (Digikey ASTK500-ND) - $79**
**STK501 (Digikey ASTK501-ND) includes ATMega128 chip - $79**
DigiKey http://www.digikey.com

**Camera: CMUcam - $109**
Seattle Robotics http://www.seattlerobotics.com

**Servos: Balsa Products BP148N Standard Torque - $10.50 each**
Mark III Robot Store http://www.junun.org/MarkIII/Store.jsp

**Wheels: Injection Mold - $6.00 pair**
Mark III Robot Store http://www.junun.org/MarkIII/Store.jsp

**Sonar: Davantech SRF04 - $34.50**
Acroname http://www.acroname.com

**IR: Sharp GP2D12 - $11.50 each**
Acroname http://www.acroname.com

**Bump Switch: SPDT Lever Switch - $3**
RadioShack

**LCD: Microprocessor Lab LCD Screen - Free**

**Bright Red, Green, and White LEDs: $10 (est)**
RadioShack

**Protoboard: Radio Shack Large Squares - $4 each**
RadioShack

**Nuts and bolts: $4**
Home Depot

**Furniture Glide Piece: $2**
Home Depot

**Red Fabric and Poster Board for target and Styrofoam for Horns: $4**

Walmart

**Black Spray Paint: Free**
Home

**Wires/Solder/Wood/Toggle Switch/Lots of female headers/5V Regulator/Hot Glue: Free**
IMDL Lab

**Code:**

```
/************************
*
*       Sbob
*       Anthony Huereca
*       IMDL: Spring 2004
*
*       Autonomous bull robot
*       ATMega128 @ 4MHz
*
************************/

#include<avr/signal.h>
#include<avr/interrupt.h>
#include<avr/io.h>
#include <inttypes.h>
#include<stdlib.h>
#include<string.h>

#define LCD_PORT PORTA

#define IR_RIGHT 0
#define IR_LEFT 1

//servo register definitions
#define SR3r 0x03A0
#define SR2r 0x0368
#define SR1r 0x0320
#define SR0 0x02EE
#define SR1 0x02B3
#define SR2 0x0255
#define SR3 0x0200

#define SL3r 0x0200
#define SL2r 0x0255
#define SL1r 0x02B3
```

```
#define SL0 0x02EE
#define SL1 0x0321
#define SL2 0x036B
#define SL3 0x03A0

/*******************************
*
*        Global Variables
*
*******************************/

//gets responses from camera
volatile unsigned char uart_recv[1000];
//position to put into recieve array
volatile int recpos;
//tells when screen dump is done
volatile char done;
//used for bump interrupt
volatile char see_red, hit;

//used for debouncing external interrupt
volatile char exint;
//only update LCD when needed, so doesn't flicker
char change;

//number of misses
unsigned char misses;

//best distance
volatile int oldrecord, record;

//behavior variables
int distance, time, xpos;

//color tracking variables
volatile unsigned char confidence;
```

```
volatile unsigned char x;

//sonar variables
volatile int prev, sonar_value;

/*********************
* INTERRUPT ROUTINES
**********************/

//UART recieve interrupt
SIGNAL(SIG_UART0_RECV)
{
        //get data
        unsigned char foo=UDR0;
        uart_recv[recpos++]=foo;
        //this part for screen dump finish signal
        if(foo==0x03)
                done=0x01;
}

//bump switch interrupt, PD0
SIGNAL(SIG_INTERRUPT0)
{
        //used for debouncing after pressed
        if(!exint)
        {
                exint=0x01;
                //if hit after attacking red object, celebrate
                if(see_red)
                {
                        //reset variables
                        hit=0x01;
                        misses=0x00;
                        record=0x0F00;
                        PORTF=PORTF&0x0F;
                        see_red=0x00;
```

```
                clear_lcd();
                lcd_puts("You just got pwnd");
                celebrate();
            }
            //accidental hit
            else
            {
                change=0x01;
                //control LED lights on back
                cbi(PORTD,4);
                cbi(PORTD,5);
                sbi(PORTD,7);
                clear_lcd();
                lcd_puts("I'm sorry!");
                avoid();
                cbi(PORTD,7);
            }
        }
        else
        {
            wait(100);
            exint=0x00;
        }

}

//sonar interrupt
SIGNAL(SIG_INPUT_CAPTURE3)
{
  int high, low, temp1, temp2, prevhigh;
  low=ICR3L;
  high=ICR3H;
  high=high<<8;
  temp1=low+high;
  prevhigh=prev+0x0600;
  //protect against random missed echo's
```

```
  if(temp1<prevhigh)
        {
              //take average
          temp2=sonar_value*3+temp1;
          sonar_value=temp2>>2;
          if(sonar_value<record && see_red)
                record=sonar_value;
        }
        prev=temp1;
}

/************************
*
*       SENSOR INIT FUNCTIONS
*
*************************/

//PB6 is Right Servo
//PB7 is Left Servo
void servo_init()
{
 outp(0xC0,DDRB);   //enable output pins
 outp(0x27,OCR1AH); //top for Fast PWM
 outp(0x10,OCR1AL);
 outp(0x02,OCR1BH); //toggle PB6 low when TCNT is here. 1.5ms
 outp(0xee,OCR1BL);
 outp(0x02,OCR1CH); //toggle PB7 low when TCNT is here 1.5ms
 outp(0xee,OCR1CL);

 outp(0x00,TCNT1H); //clear counter
 outp(0x00,TCNT1L);
 outp(0x2B,TCCR1A); //clear on compare match, set pin high at top, for output
B and C, TOP set in OCR1A
 outp(0x1A,TCCR1B); //second part of TOP set in OCR1A and scale clock by 8
}
```

```
//PE0 connected to Transmit pin on CMUcam
//PE1 connected to Recieve pin on CMUcam
void camera_init()
{
        //Set up board UART
        //set baud to 19200 @ 4 Mhz
        outp(0x0C,UBRR0L);
        //enable transmitter, reciever, and receive interrupts
        outp(0x98,UCSR0B);
        //8 data bits
        UCSR0C=0x06;

        //configure camera
        //set up camera for poll and raw mode
        uart_puts("PM 1\r");
        wait(100);
        uart_puts("RM 3\r");
        wait(100);
        //get TC parameters
        get_color_values();
        //make sure middle mass mode on
        wait(100);
        uart_puts("MM 1\r");
        wait(100);
}

//PE7 connected to echo pin
//PE4 connected to trigger pin
void sonar_init(void)
{
  outp(0x10,DDRE);
  outp(0x44,OCR3AH); //top, 35 ms
  outp(0x5C,OCR3AL);
  outp(0x00,OCR3BH); //toggle PB6 when TCNT is here
  outp(0x10,OCR3BL);
```

```
  outp(0x00,TCNT3H); //clear counter
  outp(0x00,TCNT3L);
  outp(0x23,TCCR3A); //clear on compare match, set at top, for output B, TOP
set in OCR3A
  outp(0x9A,TCCR3B); //enable noise canceler, second part of TOP set in
OCR1A and scale by 8

  outp(0x20,ETIMSK); //enable input capture iterrupt
}

//PF1 and PF0 connected to IR
/*From Kristen Allen*/
 //Initialize the A/D converter
 void ir_init(void)
 {
        outp((1<<ADEN) | (1<<ADPS2) | (ADPS1), ADCSRA); //Initialize to use
8bit resolution for all channels
 }

void lcd_init()
{
 //set LCD_PORT output
  outp(0xFF,DDRA);

  //enable 4-bit mode
  outp(0x00,LCD_PORT);
  wait(15);
  outp(0x03,LCD_PORT);
  latch_led();

  wait(5);
  outp(0x03,LCD_PORT);
  latch_led();

  //wait 100 us
  wait(1);
  outp(0x03,LCD_PORT);
```

```
 latch_led();

 //wait 4.1 ms
 wait(5);
 outp(0x02,LCD_PORT);
 latch_led();

 //four bit mode enabled, now configure LCD
 //set up 2 line mode
 wait(2);
 outp(0x02,LCD_PORT);
 latch_led();
 outp(0x0C,LCD_PORT);
 latch_led();
 //set up cursor and blink
 wait(2);
 outp(0x00,LCD_PORT);
 latch_led();
 outp(0x0C,LCD_PORT);
 latch_led();

 //clear home
 wait(2);
 outp(0x00,LCD_PORT);
 latch_led();
 outp(0x01,LCD_PORT);
 latch_led();
 wait(2);
}

/*****************************
*
*        SENSOR FUNCTIONS
*
*****************************/
```

```
/*****
* UART
*****/
//basically LCD code
void uart_puts(const char *s)
{
  register char c;
  while ( (c = *s++) ) {
    uart_write(c);
  }
}

//taken from Atmel User Manual
void uart_write(unsigned char data)
{
                while ( !( UCSR0A & (1<<UDRE0)) )
                        ;
                UDR0 = data;
}

/*****
* LCD
*****/
/* LCD code taken from
Source: Kristen Allen
Author: Chad Sylvester
Original Author: Peter Fleury <pfleury@gmx.ch> http://jump.to/fleury
*/
void latch_led()
{
 sbi(LCD_PORT, 4);
 cbi(LCD_PORT, 4);
}

void clear_lcd()
{
```

```
  //clear home
  wait(2);
  outp(0x00,LCD_PORT);
  latch_led();
  outp(0x01,LCD_PORT);
  latch_led();
  wait(2);
}

//writes a string to the LCD
void lcd_puts(const char *s)
{
   register char c;
   while ( (c = *s++) ) {
     lcd_write(c);
   }
}

//writes a specific character to the LCD
void lcd_write(unsigned char data)
{
  outp( ((data>>4)&0x0F)|(1<<6),LCD_PORT );
  latch_led();
  outp( (data&0x0F)|(1<<6), LCD_PORT);
  latch_led();
  wait(1);
}


//From Kristen Allen
//Function to read a specific channel with the desired reference voltage
int check_ir(int channel)
{
        int first, second, avg;

        //set 5V as reference, left justified, and read pin 0
```

```
        if(channel==IR_LEFT)
          outp(0x60,ADMUX);
   else
          outp(0x61,ADMUX);

        sbi(ADCSRA, ADSC);

        loop_until_bit_is_clear(ADCSRA, ADSC);          //wait till
conversion is complete
        first = inp(ADCH);

        if(channel==IR_LEFT)
          outp(0x60,ADMUX);
   else
          outp(0x61,ADMUX);

        sbi(ADCSRA, ADSC);

        loop_until_bit_is_clear(ADCSRA, ADSC);          //wait till
conversion is complete
        second = inp(ADCH);
        avg=(first+second)>>1;
        return avg;
}

//get sonar values without interrupts
int check_sonar()
{
        int high, temp, low, temp1, temp2;

        low=ICR3L;
        high=ICR3H;
   high=high<<8;
   temp1=low+high;
        wait(40);
        low=ICR3L;
        high=ICR3H;
```

```
   high=high<<8;
   temp2=low+high;
        temp=(temp1+temp2)>>1;
   return temp;
}


void display_sonar()
{
   char string_output[4];
        itoa(sonar_value, string_output, 16);
        clear_lcd();
        lcd_puts(string_output);
}

//takes arguements outlined in definitions. For example, servo(SL2,SR2)
//Then smooths it out by a factor of four
void servo(int left, int right)
{
        int temp, templ, tempr, rr, lr, leftreg, rightreg;
        leftreg=OCR1BL;
        temp=OCR1BH;
        temp=temp<<8;
        leftreg=temp+leftreg;
        rightreg=OCR1CL;
        temp=OCR1CH;
        temp=temp<<8;
        rightreg=temp+rightreg;
        //calculate 1/4 of the difference, and add each during each loop
        int diffr, diffl;
        if(leftreg>left)
                diffl=-((leftreg-left)>>2);
        else if(leftreg<left)
                diffl=((left-leftreg)>>2);
        else
                diffl=0x00;
```

```
if(rightreg>right)
        diffr=-((rightreg-right)>>2);
else if(rightreg<right)
        diffr=((right-rightreg)>>2);
else
        diffr=0x00;
//if there is a difference in register values then run algorithm
if(diffl||diffr)
{
        char i=0;
        while(i<3)
        {
                leftreg=leftreg+diffl;
                rightreg=rightreg+diffr;
                /*clear_lcd();
                lcd_puts("reg L: ");
                itoa(leftreg, string_output, 16);
                lcd_puts(string_output);
                lcd_puts("                    reg R: ");
                itoa(rightreg, string_output, 16);
                lcd_puts(string_output);
                */
                templ=leftreg>>8;
                tempr=rightreg>>8;
                OCR1BH=templ;
                OCR1BL=leftreg;
                OCR1CH=tempr;
                OCR1CL=rightreg;
                i++;
                wait(25);
        }
        templ=left>>8;
        tempr=right>>8;
        OCR1BH=templ;
        OCR1BL=left;
        OCR1CH=tempr;
```

```
                OCR1CL=right;
                wait(25);
        }
}

//get camera confidence value from TC command
void get_new_confidence_value()
{
        recpos=0;
        uart_puts("TC\r");
        while(recpos<10 && hit==0x00);
        confidence=uart_recv[8];
        x=uart_recv[2];
        wait(50);
}

void get_color_values()
{
        int redmax, redmin, bluemax, bluemin, greenmin, greenmax;
        char redmaxs[5], redmins[5], bluemaxs[5], bluemins[5], greenmaxs[5],
greenmins[5];
        int dist=check_sonar();
        //if nothing in front, then go with default parameters for red. Else get
them interactively
        if(dist<0x0400)
        {
                int redcomp, greencomp, bluecomp;
                //make windows size smaller so don't have huge amounts of data
                uart_puts("SW 25 50 55 90\r");
                wait(500);
                done=0x00;
                recpos=0;
                //enable interrupts so can get data back
                sei();
                recpos=0;
                clear_lcd();
                lcd_puts("Analyzing Object");
```

```
//get frame
uart_puts("DF\r");
while(!done);
clear_lcd();
//attempting to calculate TC color parameters
redmin=(int)uart_recv[3];
redmax=redmin;
greenmin=(int)uart_recv[4];
greenmax=greenmin;
bluemin=(int)uart_recv[5];
bluemax=bluemin;
int i=6;

while(i<recpos)
{
        if(uart_recv[i]==0x02)
                i++;
        redcomp=uart_recv[i++];
        greencomp=uart_recv[i++];
        bluecomp=uart_recv[i++];
        if(redcomp<(redmax+5) && redcomp>redmax)
                redmax=redcomp;
        else if(redcomp>(redmin-5) && redcomp<redmin)
                redmin=redcomp;
        if(greencomp<(greenmax+5) && greencomp>greenmax)
                greenmax=greencomp;
        else if(greencomp>(greenmin-5) && greencomp<greenmin)
                greenmin=greencomp;
        if(bluecomp<(bluemax+5) && bluecomp>bluemax)
                bluemax=bluecomp;
        else if(bluecomp>(bluemin-5) && bluecomp<bluemin)
                bluemin=bluecomp;
}

if(redmax<30)
        redmax=30;
```

```
        if(greenmax<30)
                greenmax=30;
        if(bluemax<30)
                bluemax=30;

        //set window back to full size
        uart_puts("SW\r");
}
else
{
        redmin=100;
        redmax=240;
        greenmin=16;
        greenmax=50;
        bluemin=16;
        bluemax=50;

}
clear_lcd();
lcd_puts("R:");
itoa(redmin, redmins, 10);
lcd_puts(redmins);
lcd_puts(" ");
itoa(redmax, redmaxs, 10);
lcd_puts(redmaxs);
lcd_puts(" G:");
itoa(greenmin, greenmins, 10);
lcd_puts(greenmins);
lcd_puts(" ");
itoa(greenmax, greenmaxs, 10);
lcd_puts(greenmaxs);
lcd_puts("                    B: ");
itoa(bluemin, bluemins, 10);
lcd_puts(bluemins);
lcd_puts(" ");
itoa(bluemax, bluemaxs, 10);
```

```
lcd_puts(bluemaxs);
wait(3000);
//call once so can just call TC later on
recpos=0;
uart_puts("TC ");
uart_puts(redmins);
uart_puts(" ");
uart_puts(redmaxs);
uart_puts(" ");
uart_puts(greenmins);
uart_puts(" ");
uart_puts(greenmaxs);
uart_puts(" ");
uart_puts(bluemins);
uart_puts(" ");
uart_puts(bluemaxs);
uart_puts("\r");
wait(300);
if(dist<0x400)
{
        //calculate time lag and max recognizable distance
        sei();
        wait(200);
        char foo2[5];
        clear_lcd();
        lcd_puts("Lift object out of the way");
        while(sonar_value<0x800)
                ;
        wait(200);
        clear_lcd();
        lcd_puts("Put object in front");
        int conf=0x00;
        while(sonar_value>0x800)
                ;
        clear_lcd();
        lcd_puts("Looking for recognition");
```

```
                time=1;
                char sout[5];
                get_new_confidence_value();
                while(confidence<200)
                {
                        time++;
                        get_new_confidence_value();
                }
                time=time*75;
                get_new_confidence_value();
                xpos=x;
                get_new_confidence_value();
                xpos=(xpos+x)>>1;
                clear_lcd();
                lcd_puts("Move away");
                get_new_confidence_value();
                while(confidence>100)
                {
                        distance=sonar_value;
                        get_new_confidence_value();
                }
                //allow some error room
                distance=distance-0x100;
        }
        //if no object in front, use default values
        else
        {
                time=500;
                distance=0x0700;
                xpos=37;
        }
        cli();
}

/************************
*
```

```
*       BEHAVIORS (the fun stuff)
*
************************/

//From Kristen Allen
//approx 1ms @ 4Mhz
void wait(int delaytime)
{
  while (delaytime)
  {
    delaytime--;
    int i;
    for(i=850;i;i--)asm("nop");
  }
}


void avoid()
{
        //back up first if close to object
        if(sonar_value<0x250)
        {
                servo(SL2r,SR2r);
                wait(300);
        }
        int irr=check_ir(IR_RIGHT);
        int irl=check_ir(IR_LEFT);
        //do random turn if neither IR sensor reports anything close
        if(irr<0x15 && irl<0x15)
        {
                if(TCNT1L&0x01)
                {
                        servo(SL2,SR2r);
                }
                else
                {
```

```
                servo(SL2r,SR2);
            }
        }
        else
        {
            if(irr>irl)
            {
                servo(SL2r,SR2);
            }
            else
            {
                servo(SL2,SR2r);
            }
        }
        wait(500);
        servo(SL2,SR2);
}




void blink_leds()
{
        //PORTF4-7=LED Bank
        //PORTD4=red
        //PORTD5=green
        //PORTD7=white
        //PORTC=STK500 leds
        unsigned char foo=0x10;
        PORTF=PORTF&0x0F;
        PORTD=PORTD&0x0F;
        PORTC=0x00;
        uart_puts("L1 0\r");
        PORTF=PORTF&0x0F;
        PORTD=PORTD&0x0F;
```

```
        PORTC=0x00;
        wait(100);
        foo=0x10;
        PORTF=PORTF|foo;
        PORTD=PORTD|foo;
        wait(200);
        foo=foo<<1;
        PORTF=PORTF|foo;
        PORTD=PORTD|foo;
        wait(200);
        foo=foo<<1;
        PORTF=PORTF|foo;
        PORTC=~PORTC;
        uart_puts("L1 1\r");
        wait(200);
        foo=foo<<1;
        PORTF=PORTF|foo;
        PORTD=PORTD|foo;
        wait(200);
        PORTF=PORTF&0x0F;
        PORTD=PORTD&0x0F;
}

void celebrate()
{
        //reverse
        servo(SL2r,SR2r);
        wait(300);
        //spin
        servo(SL2,SR2r);
        int i=0;
        while(i<4)
        {
                blink_leds();
                i++;
        }
```

```
        uart_puts("L1 0\r");
        PORTF=PORTF&0x0F;
        PORTD=PORTD&0x0F;
        PORTC=0x00;
        servo(SL2,SR2);
}

//most important behavior
void ram()
{
        int conf;
        char zeros, past;
        unsigned char uno, dos;
        char right=xpos+7;
        char left=xpos-7;
        char string_output[5];
        clear_lcd();
        lcd_puts("CHARGE!!!!!11unounouno");
        blink_leds();
        //make motors stutter
        servo(SL1r,SR1r);
        wait(50);
        servo(SL1,SR1);
        wait(50);
        servo(SL1r,SR1r);
        wait(50);
        sbi(PORTD,4);
        PORTF=PORTF|misses;
        //begin moving forward
        servo(SL2,SR2);
        int sonar_temp=sonar_value;
        switch(misses)
        {
                case 0:
                        lcd_puts("               1: What's this?");
                        //increment miss counter for next time, and speed up
```

```
        misses=0x10;
        servo(SL3,SR3);
        conf=0x50;
        //continue until lose red tracking
        while(conf>20 && hit==0x00)
        {
                get_new_confidence_value();
                conf=confidence;
                get_new_confidence_value();
                conf=(conf+confidence)>>1;
                wait(10);
        }
        break;
case 0x10:
        lcd_puts("              2: Sorta Mad");
        misses=0x30;
        past=0x00;
        conf=0x50;
        while(conf>20 && hit==0x00)
        {
                get_new_confidence_value();
                conf=confidence;
                get_new_confidence_value();
                conf=(conf+confidence)>>1;

                sonar_temp=sonar_value;
                if(sonar_temp>0x400 && !past)
                {
                        servo(SL3,SR3);
                }
                else if(sonar_temp>0x200 && past<0x02)
                {
                        servo(SL1,SR1);
                        past=0x01;
                }
                else
```

```
                {
                        servo(SL2,SR2);
                        past=0x02;
                }
                wait(10);
        }
        break;
case 0x30:
        lcd_puts("              3: Blowing my top");
        misses=0x70;
        zeros=0;
        //allow two missed readings
        while(zeros<2 && hit==0x00)
        {
                get_new_confidence_value();
                uno=x;
                get_new_confidence_value();
                dos=x;
                if(confidence>20)
                {
                        zeros=0;
                        if(uno>right && dos>right)
                                right_track();
                        else if(uno<left && dos<left)
                                left_track();
                        else
                                straight_track();
                }
                else
                        zeros++;
                wait(50);
        }
        break;
case 0x70:
        lcd_puts("              4: You're FINISHED!");
        misses=0xF0;
```

```
past=0x00;
zeros=0;
while(zeros<2 && hit==0x00)
{
        get_new_confidence_value();
        uno=x;
        get_new_confidence_value();
        dos=x;
        if(confidence>20)
        {
                zeros=0;
                wait(50);
                if(uno>right && dos>right)
                        right_track();
                else if(uno<left && dos<left)
                        left_track();
                else
                {
                        //clear_lcd();
                        //lcd_puts("straight");
                        sonar_temp=sonar_value;
                        if(sonar_temp>0x400 && !past)
                        {
                                //lcd_puts("fast");
                                servo(SL3,SR3);
                        }
                        else if(sonar_temp>0x200 &&
past<0x02)
                        {
                                //lcd_puts("slow");
                                servo(SL1,SR1);
                                past=0x01;
                        }
                        else
                        {
                                //lcd_puts("regular");
```

```
                                        servo(SL2,SR2);
                                        past=0x02;
                                }
                        }
                }
                else
                        zeros++;
                wait(50);
        }
        break;
default:
        lcd_puts("       If I see this, something bad happened");
        wait(1000);
}
//if didn't hit object
if(!hit)
{
        clear_lcd();
        servo(SL0,SR0);
        PORTF=PORTF|misses;
        see_red=0x00;
        int dist;
        //calculate distance in inches
        dist=record-0x0D0;
        int temp2=0x4C;
        div_t foo=div(dist,temp2);
        dist=foo.quot;
        int remain=foo.rem;
        itoa(dist, string_output, 10);
        //new record
        if(oldrecord>record)
        {
                oldrecord=record;
                clear_lcd();
                lcd_puts("You set a new record!                    ");
                lcd_puts(string_output);
```

```
            if(remain>=0x26)
                    lcd_puts(".5");
            lcd_puts(" inches");
            wait(2000);
            avoid();
    }
    else
    {
            clear_lcd();
            lcd_puts("Crap, I missed! NOOOOO!!!");
            wait(2000);
            avoid();
    }
    //sleep mode
    if(misses==0xF0)
    {
            servo(SL0,SR0);
            clear_lcd();
            lcd_puts("I'm tired now, so                going to sleep");
            //turn off LED's
            PORTF=PORTF&0x0F;
            PORTD=PORTD&0x0F;
            sbi(PORTD,5);
            wait(2000);
            clear_lcd();
            lcd_puts("Zzzzzzzzzzzzz                Record:");
            lcd_puts(string_output);
            if(remain>=0x26)
                    lcd_puts(".5");
            lcd_puts(" inches");
            record=0x0F00;
            misses=0x00;
            //turn off external interrupts
            cbi(EIMSK,0);
            int irl=check_ir(IR_LEFT);
            int irr=check_ir(IR_RIGHT);
```

```
                        char bump=0x01;
                        //scans for movement in front of it
                        while(bump && irl<0x75 && irr<0x75 &&
sonar_temp>0x250)
                        {
                                bump=inp(PIND);
                                bump=bump&0x01;
                                irl=check_ir(IR_LEFT);
                                irr=check_ir(IR_RIGHT);
                                sonar_temp=sonar_value;
                                wait(40);
                        }
                        sbi(EIMSK,0);
                        cbi(PORTD,5);
                        clear_lcd();
                        lcd_puts("You woke me up!");
                        wait(2000);
                        servo(SL2,SR2);
                }
        }
}

void left_track()
{
        //clear_lcd();
        //lcd_puts("going to the left");
        servo(SL0,SR2);
}

void right_track()
{
        //clear_lcd();
        //lcd_puts("going to the right");
        servo(SL2,SR0);
}
```

```
void straight_track()
{
        //clear_lcd();
        //lcd_puts("going forward");
        servo(SL2,SR2);
}

void main(void)
{
        //set record to high value initially
        record=0x0F00;
        oldrecord=record;
        char bump=0x01;
        change=0x01;
        misses=0x00;
        see_red=0x00;
        hit=0x00;
        //debug output
        outp(0xFF,DDRC);
        //LED lights
        outp(0xF0,DDRD);
        outp(0xF0,DDRF);
        //turn on front lights
   outp(0x00,PORTC);


        lcd_init();
        lcd_puts("SBOB is getting ready");
        sonar_init();
        camera_init();
        ir_init();
        servo_init();
        //get initial value for sonar interrupt
        sonar_value=check_sonar();
        prev=sonar_value;
```

```
        clear_lcd();
    lcd_puts("Welcome to SBOB                    ");
        char sout[5];
        int dist2=distance-0x0D0;
        int temp2=0x4C;
        div_t foo=div(dist2,temp2);
        dist2=foo.quot;
        itoa(dist2, sout, 10);
        lcd_puts("D: ");
        lcd_puts(sout);
        lcd_puts(" in");
        itoa(time,sout,10);
        lcd_puts(" T: ");
        lcd_puts(sout);
        lcd_puts(" ms");
        //wait for bump switch to be hit
        while(bump)
    {
      bump=inp(PIND);
           bump=bump&0x01;
    }
        //bump interrupt init
        sbi(EICRA,1);
        sbi(EIMSK,0);
        exint=0x01;
        wait(300);
        //start interrupts and GO!
        sei();
        while(1)
        {
                //turns off any stray LED's
                PORTD=PORTD&0x0F;
                int irr=check_ir(IR_RIGHT);
                if(irr>0x45)
                {
```

```
        //clear_lcd();
        //lcd_puts("go left");
        while(irr>0x45)
        {
                irr=check_ir(IR_RIGHT);
                servo(SL0,SR2);
        }
        wait(400);
        servo(SL2,SR2);
}
int irl=check_ir(IR_LEFT);
if(irl>0x45)
{
        //clear_lcd();
        //lcd_puts("go right");
        while(irl>0x45)
        {
                irl=check_ir(IR_LEFT);
                servo(SL2,SR0);
        }
        wait(400);
        servo(SL2,SR2);
}
//if no object in front, display best distance
if(sonar_value>distance)
{
        servo(SL2,SR2);
        cbi(PORTD,4);
        if(record==0x0F00)
        {
                if(change)
                {
                        //Until see red object, display this.
                        clear_lcd();
                        lcd_puts("I am Sbob the bull.");
                        change=0;
```

```
                        }
                }
                else
                {
                        if(change)
                        {
                                char records[5];
                                //calculate inches
                                int dist=record-0x0D0;
                                int temp2=0x4C;
                                div_t foo=div(dist,temp2);
                                dist=foo.quot;
                                int remain=foo.rem;
                                itoa(dist, records, 10);
                                clear_lcd();
                                lcd_puts("The current record is
");
                                lcd_puts(records);
                                if(remain>=0x26)
                                        lcd_puts(".5");
                                lcd_puts(" inches");
                                change=0;
                        }
                }
        }
        else
        {
                change=0x01;
                //slow down if far away and wait for camera to update.
Takes a while
                if(sonar_value>0x600 && time<700)
                        servo(SL1,SR1);
                //if object is close, then stop
                else
                        servo(SL0,SR0);
                clear_lcd();
                lcd_puts("Scanning");
```

```
                    wait(time);
                    recpos=0;
                    get_new_confidence_value();
                    clear_lcd();
                    if(confidence>30)
                    {
                            sbi(PORTD,4);
                            see_red=0x01;
                            hit=0x00;
                            ram();
                    }
                    else
                    {
                            clear_lcd();
                            lcd_puts("That color is BORING");
                            cbi(PORTD,4);
                            avoid();

                    }
            }
        }
}
```