Dueling Dragsters
Final Report
Michael Pridgen
Thomas Vermeer
April 24, 2007
IMDL

Table of Content

Abstract

The purpose of this project was to build two fully autonomous robots that would race each other on a custom-built track. The robots would simulate a National Hotrod Association Top-Fuel Dragster race in every way possible. The dragster would pull themselves up to the prestage area, then inch up to the stage area, then wait for the green light. On the green light, the dragsters would hit full throttle and attempt to win the race. The dragsters would be responsible for keeping themselves straight and stopping at the end of the course. All communication between the track and the dragsters would be done with lasers and photo resistors. The track would know when the laser was broken and report this as breaking the prestage and stage laser. The prestage laser would serve two functions, as both the signal to the track of the dragster's location and telling the dragsters when to go. While in motion, the dragsters use a single side-mounted sonar to determine their distance from the center line. They then use this distance to extrapolate how they need to turn in order to remain on a straight path. They have a single servo mounted in front to control their steering.

Executive Summary

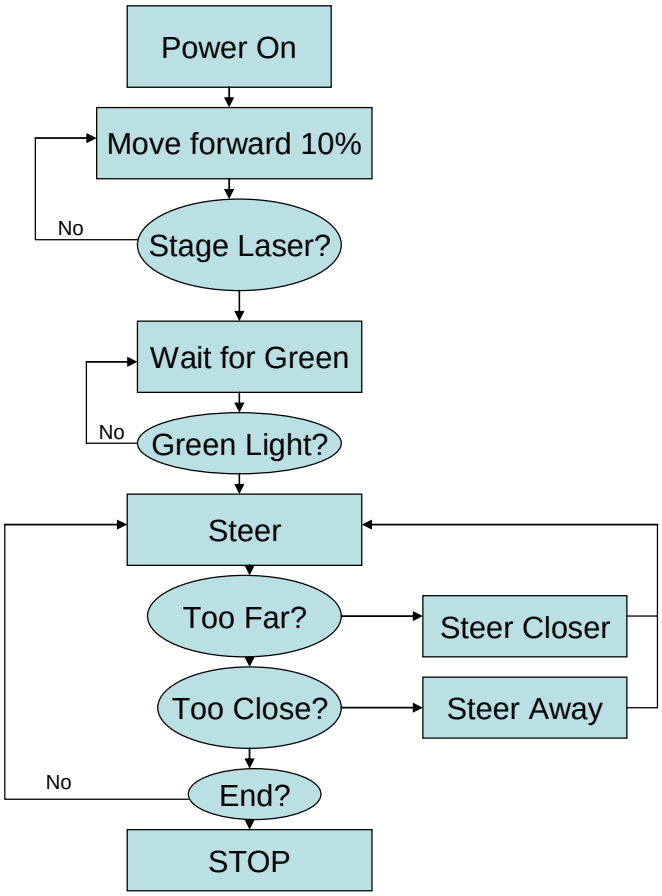
At the beginning of this semester, we set out to create the fastest land-based IMDL project ever. Mike Pridgen and Thomas Vermeer decided to work together to make robotic dragsters that would simulate a National Hotrod Association dragster race, such as Gator Nationals. We started by purchasing a variety of parts for custom remote controlled racing car parts. We ordered batteries, wheels, axils, gears, and two racing motors. After some testing with wooden frames, we decided that the platforms of our dragsters should be made of fiberglass. We bought metal and fiberglass from Lowe's and began construction. In the mean time, we purchased 3 mavric IIB boards from bdmicro.com and wrote the code to make them work with both servos and sonar. This proved very time consuming due to the fact I am not familiar with C programming or the architecture of the Atmel AtMega128 microcontroller. Once we had the servos working, we mounted a Lego gear onto the servo head and used a flat Lego gear to interface with it. We made a rack and pinion steering column using these two gears and it worked rather well. The wheels could easily steer left and right. We then mounted the sonar onto the side of the car. It was very difficult to write the proper code to follow a wall, the steering had the tendency to extremely overshoot the turning and crash itself. We compensated for this by making sure the robot was extremely well aligned and that it only needed to steer a very small amount for it to not hit the wall. I also build a function that would kill the motor if the sonar returned a value higher than expected, i.e. when the course was over and the 2X4 gone. Once we had to dragsters steering, we needed a way to make them go. After borrowing a remote control motor driver from a friend, we decided that this was definitely the way to go. We purchased Intellispeed motor drivers from HobbyTown USA on Archer. It was very easy to get these to work, the function to control them is almost identical to the function for servo actuation. Once we had the cars driving and steering on their own, we began construction of the track. The track has a 'tree' which has sets of lights for each dragster. For each dragster is has a prestage light, stage light, set lights, and go lights. We set up a series of lasers and photo resistors on the track and attached them to the A/D system of the Mavric. If the photo resistor has no light on it, it has very high resistance, and if ot has the laser on it, the resistance is almost zero. We had the laser turn the lasers on and off and compare the voltages on the A/D channels so that the track would know precisely which lasers were broken. When the cars first turn on, they inch forward. The track illuminates the prestage lights when the first laser is broken, and stage lights when the second laser is broken. When the second laser is broken, the stage lights come on and the prestage laser becomes aligned with a photo resistor on the dragster. The dragster then waits for the prestage laser to turn off, coinciding with the green light turning on. When this signal goes true, the cars immediately move the motor to full throttle and begin to steer. When the sonar reports that the 2X4 is no longer present, the motor is put into full reverse, causing the motor to become a brake and stop the car. The track will then report the reaction time to the 4 line LCD mounted on it. The reaction time is the time between when the light turns green and when the car begins to move.

Introduction

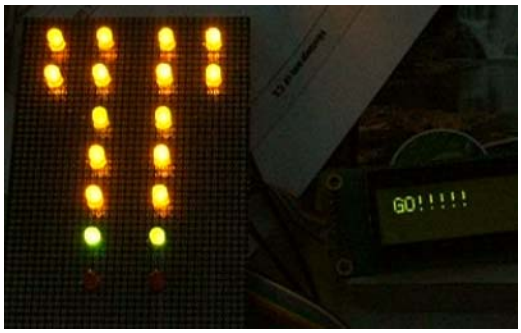
In the beginning of the semester, Mike Pridgen and Thomas Vermeer set out to make the fastest ground robots ever in IMDL. The dragsters are based on real dragsters such as the ones that race in Gator Nationals. We planned to have the robots powered by AA batteries for the Mavric and NiMH R/C car battery packs for the motor. The cars would use a servo for steering and a R/C car motor driver for speed. The track would make sure both cars started from the same location and started the race at the same time. This paper will explain the design of our system.

Integrated Systems

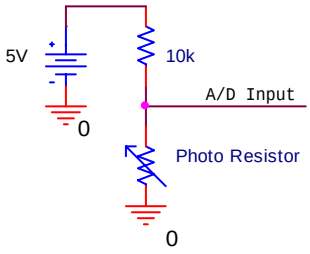
The system is based on C code programmed onto an Atmel Atmega128 board. The following flow chart explains the logical flow of the dragster.



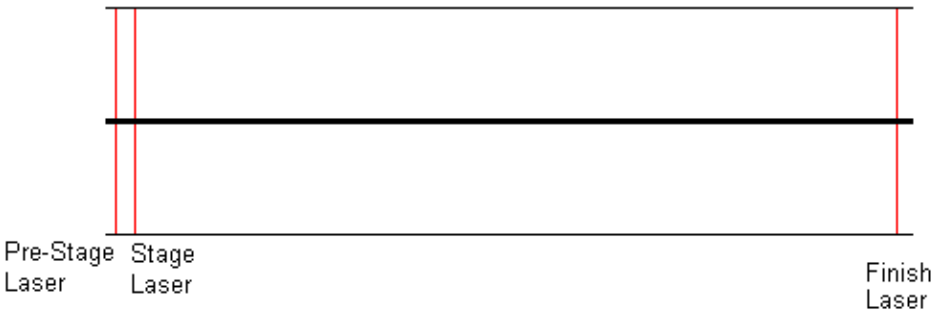
Track



Each track will be comprised of three lasers and photo resistors. The schematic for the lasers is shown in Figure 1.

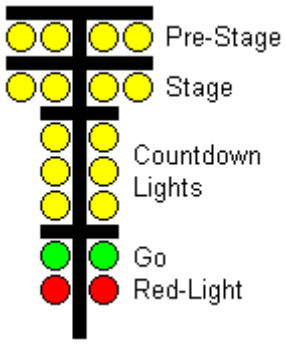


When no laser is shining on the photo resistor, the A/D input voltage is around 3.5 V. When the laser shines on the photo resistor, the intense light of the laser drops the resistance to 0Ω. This creates a short to ground, which forces the A/D input voltage to be 0V. Each laser will be connected to a separate A/D channel. The channels will be continuously sampled to determine if the laser is shining on the photo resistor or not. The lasers will be lain out on the track as shown in Figure 2.



The car will drive into the track from behind the staging lasers. Initially, the car will break the pre-stage laser, turning on the pre-stage lights of the tree. See Figure 3 for a diagram of the tree. The car will continue forwards slowly until the stage laser breaks. At this point, the pre-stage laser will be broken again by the car, by a photo resistor on the underside of the car. When the car detects the laser on the photo resistor, it will stop.

When the stage laser is broken, the track turns on the stage lights. After both cars are successfully staged, all four stage and pre-stage lasers should be broken. After a short period of time, the countdown lights will turn on and the pre-stage laser will pulse, which is the "Go" signal. As soon as the car moves forward, the pre-stage beam will no longer be blocked by the car's photo resistor. The delay between the "go" signal and the pre-stage laser being restored will determine the reaction time of the dragster. The car will drive straight down the track, and eventually break the finish line. The delay between the restoration of the pre-stage and the breaking of the finish line will determine the elapsed time for the run.



The rear of the car will have a bump sensor built into a wheelie bar as shown in Figure 4. During the run, if the car does a wheelie, the bump sensor will be activated, and the LCD will display an acknowledgement that a wheelie occurred. The secondary feature of the bump sensor is when the car is backing up slowly, if it runs into something, the switch is depressed telling the car to stop.

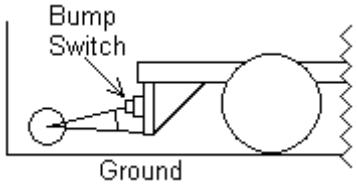


Figure 4

The schematic for the bump switch is shown in Figure 5. The output is connected to an input on the Mavric IIB board. When the switch is depressed, the output is shorted to ground, forcing the input to 0. Otherwise, the output is connected to 5V through a pull-up resistor, giving a high input.

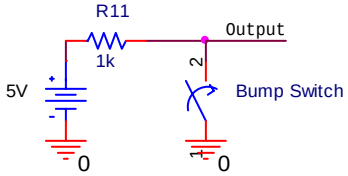
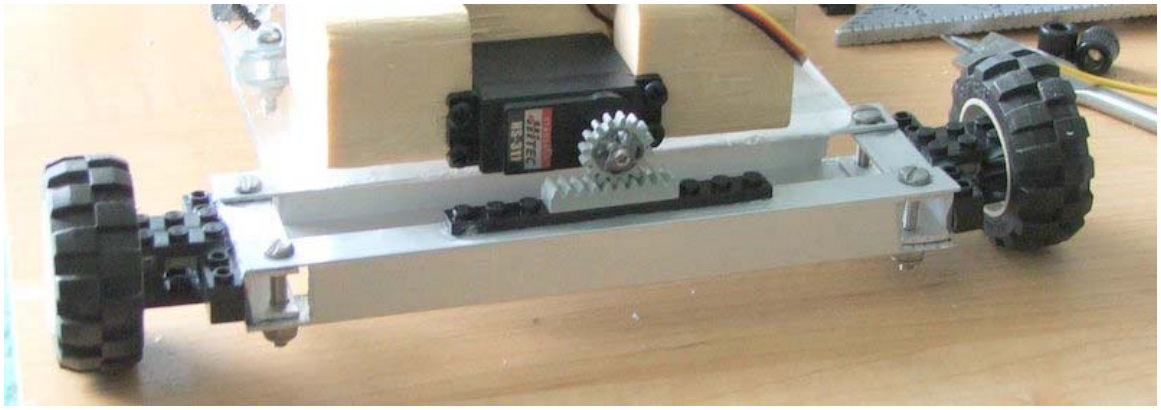


Figure 5

Actuation

The steering of our dragster was accomplished with a single HiTec Servo on a rack and pinion design. When the servo turns, the car steers left or right.

HiTec HS-311



This is the steering column of the dragster.

The movement of the car was done with an Intellispeed motor driver and a R/C car racing motor.

This is the motor driver. It will support 128 Amps continuous and 660 amps peak.

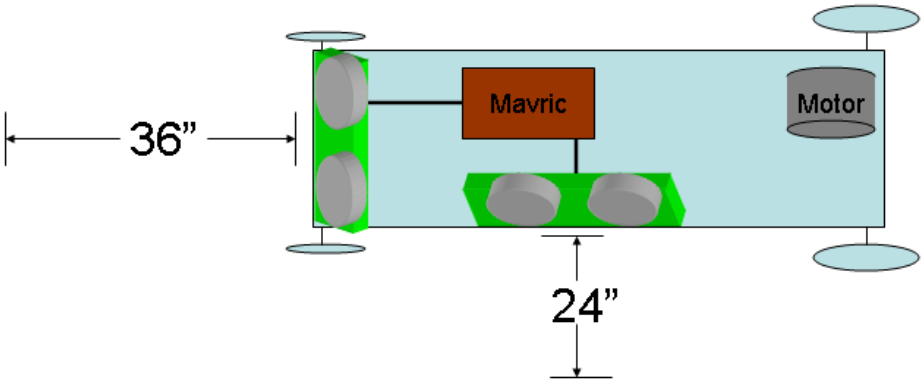
This is the back of Mike's car. You can see the block holding the motor, the motor touching the gears, and the motor driver attached to the motor. We simply drilled a hole in the 2X4 with a hole saw to make the motor fit snugly.

The motors are 25 turn and have a stall current of around 60 amps. The motors were actually the most reliable part of our car. They always worked and gave us no trouble.

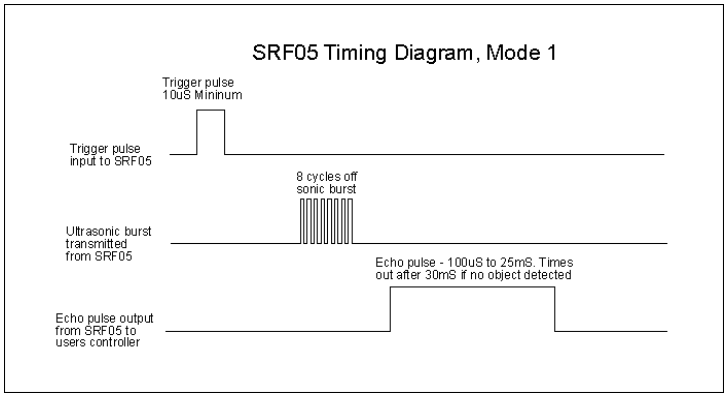
Sensors

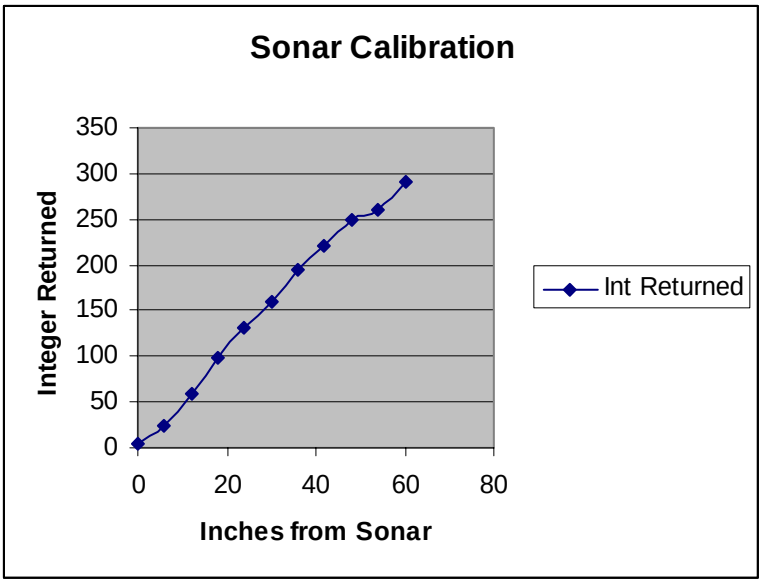
Sonar

For our sonar we used two Devantech SRF05 sonic range finders we purchased off acroname.com Two sonar systems will be mounted on the dragster. In our programming, we will have a function called update steering which will run approximately 30 times per second. It will take several sonar readings, take the average, and determine where the wheels need to be in order to correct the steering to keep the car going straight and approximately 24” from the center wall.



This picture shows how will mount the sonar on our dragster. This is the timing diagram.



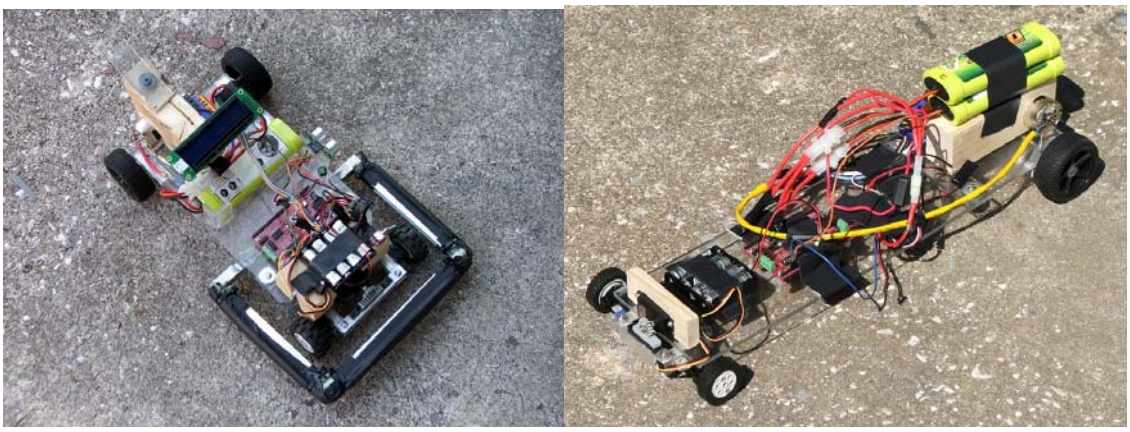


This graph shows the number of ticks on the Mavric between when the pulse is sent from the sonar until it returns. It is linearly proportional to the distance.

Behavior

The dragsters have very simple behavior. They will inch forward towards the stage lights. When they break both the prestage and stage lights, they will wait for the green light. The will know the light is green because the laser turns off. When the lasers turn off, the dragsters will race full speed down the track and steer. They will steer by getting sonar values and seeing if they are too close or too far from the wall. If they care close, they will steer away. If they are far, they will steer closer. When the dragster detects a much larger than normal sonar pulse, it will know that it has gone past all of the 2X4s and the race is over. It will initialize the ABS braking system built into the motor driver and make the car come to a quick stop.

Experimental Layout and Results



These are the pictures of our final two dragsters. Thomas's dragster is on the left and Mike's dragster is on the right.

This is the final layout of our race. As you can see, the Tree is in the middle section with the cars on either side of it. There are 2 sets of lasers for each car. Just as they were programmed, the cars moved forward until they broke the stage laser. Once broken, they waited for the green signal. Once the lights turned green, the cars took off at full speed towards the finish line.

Conclusion

After all has been said and done, this was a very successful semester. The dragsters performed almost to perfection as explained in pervious sections. One of the biggest problems with the cars was their propensity to break. Thomas crashed his cars a total of 5 times, with almost complete destruction. My advice to future students would be to start early and don't be afraid to take risks. I was too afraid to try anything too far outside the box because I was worried about breaking something I could not easily repair. In the future, we would like to add an elapsed time sensor to the end of the track so we could definitely know who won each race and exactly by how much. If I could start the project over, I would do almost everything the same, only faster. I kept a surprising large portion of my original design, I just designed very slowly.

Documentation

Kyle Carithers, Final Report
http://www.mil.ufl.edu/courses/eel5666/papers/default.htm, 2006

Natthapol Prakongpan, Final Report
http://www.mil.ufl.edu/courses/eel5666/papers/default.htm, 2006

Appendices

Main Program Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "ThomasMotors.h"
#include "mikeLCD.h"
#include "mikeSonar.h"


int main(void)
{
        int laserTHSH=150;
        int sonarTHSH=150;
        int ADLOW=0;
        int ADHIGH=0;
        int i=0;
        int photo;
        DDRG = 0b000;
        initLCD();
        lcdString("Program Started");
        initServo();
        initADC();
        initSonar();
        initMotor();
        moveServo(0);
        ms_sleep(1000);
        lcdClear();
        i=cleanSonar();
        if (i>sonarTHSH){kill();}
        lcdString("Searching for Laser");
        moveMotor(6);
        while (1)
    {
          photo = adcZero();
          if (photo < laserTHSH)
                break;

    }
        moveMotor(0);
        lcdClear();
        lcdString("Laser Found");
        ms_sleep(200);
        lcdClear();
        lcdString("Waiting for GREEN");

        i=cleanSonar();
        if (i>sonarTHSH){kill();}
    while (1)
     {
          photo = adcZero();
```

```
            if (photo > laserTHSH)
             break;
                                        lcdGoto(1,0);
                                        lcdClear();
                                        lcdInt(photo);


        }
        ms_sleep(400);



            i=cleanSonar();
            if (i>sonarTHSH){kill();}

            moveMotor(100);

            int time=90;
            while(time>0)
            {
                    int i=cleanSonar();
                    if (i>180){
                    brake();}
                    int x=newServoPOS(i);
                    moveServo(x);
                    time--;
            }

            moveMotor(0);

            while(1){
                    int i=cleanSonar();
                    int x=newServoPOS(i);
                    moveServo(x);
                    moveMotor(0);
                    lcdClear();
        lcdString("Sonar=");
                    lcdInt(i);
        lcdString("  Servo=");
                    lcdInt(100+x);
                    ms_sleep(100);

            }

            return 0;
}


int cleanSonar(void)
{
int x1=0;
int x2=0;
int x3=0;
int dist=0;

x1=sonarDist();
```

```
if (x1>400){
return 400;
}

x2=sonarDist();
x3=sonarDist();

dist=(x1+x2+x3)/3;

return dist;

}

int newServoPOS(int i){

int myServoPOS=0;
if ((i>50)){
myServoPOS=-5;
}
if ((i<20)){
myServoPOS=5;
}
return myServoPOS;
}

void kill(void){
        while(1){
                moveMotor(0);
                moveServo(3);
                lcdClear();
                lcdString("Motor KILLED");
                ms_sleep(50000);
        }
}

void brake(void){
                int i=0;
                moveServo(3);
                moveMotor(-100);
                ms_sleep(2000);
                kill();
}
```

Analog to Digital Header File

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "mikeADC.h"

void initADC(void)
{
//init port F for AD
        DDRF = 0x00;     //set as input
        PORTF = 0;
//set to use internal ARef, and put value in ADCH
        ADMUX = 0b01100000;
        ms_sleep(100);    //wait for power up
//set to on, free running at 1/2 times CLK
        ADCSRA = ~(_BV(ADIE));
        ms_sleep(100);    //wait for initialization
}

int adcZero(void)
{
        ADMUX = 0b01100000;
        ms_sleep(1);
        return ADCH;
}

int adcOne(void)
{
        ADMUX = 0b01100001;
        ms_sleep(1);
        return ADCH;

}

int adcTwo(void)
{
        ADMUX = 0b01100010;
        return ADCH;
}

int adcThree(void)
{
        ADMUX = 0b01100011;
        return ADCH;
}

int adcFour(void)
{
        ADMUX = 0b01100100;
        return ADCH;
}
```

LCD Header File

```
/*          Pinout
**          P7 - No connect (NC)
**          P6 - Enable (E)
**          P5 - Read/Write_L (R/W_L)
**          P4 - Register Select (RS)
**          P3 - Data 7 (DB7)
**          P2 - Data 6 (DB6)
**          P1 - Data 5 (DB5)
**          P0 - Data 4 (DB4)
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "mikeLCD.h"


volatile uint16_t ms_count;


/*
 * ms_sleep() - delay for specified number of milliseconds
 */
void ms_sleep(uint16_t ms)
{
  TCNT2  = 0;
  ms_count = 0;
  while (ms_count != ms)
    ;
}


/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE2)
{
  ms_count++;
}


/*
 * initialize timer 0 to generate an interrupt every millisecond.
 */
void init_timer(void)
{
  /*
   * Initialize timer0 to generate an output compare interrupt, and
   * set the output compare register so that we get that interrupt
   * every millisecond.
   */
  TIFR  |= _BV(OCIE2);
```

```
 TCCR2  = _BV(WGM01)|_BV(CS02)|_BV(CS00); /* CTC, prescale = 128 */
 TCNT2  = 0;
 TIMSK |= _BV(OCIE2);    /* enable output compare interrupt */
 OCR2   = 12;         /* match in 1 ms */
}

void lcdDataWork(unsigned char c)
{
        c = c & 0x0F;           //keep low four as data
        c = c | 0x40;           //set E hi
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
        c = c ^ 0x40;           //set E low
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
        c = c | 0x40;           //set E hi
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
}

void lcdData(unsigned char c)
{
        unsigned char c1 = c & 0xF0;            //give c1 hi 4 bits of data
        unsigned char c2 = c & 0x0F;            //give c2 low 4 bits of data
        c1 = c1 / 0x10;         //shift c1 right by 4
        lcdDataWork(c1);
        lcdDataWork(c2);
}

void lcdCharWork(unsigned char c)
{
        c = c & 0x0F;           //keep low four as data
        c = c | 0x50;           //set E and RS hi
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
        c = c ^ 0x40;           //set E low
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
        c = c | 0x40;           //set E hi
        LCD = c;                        //write to LCD
        ms_sleep(1);            //wait 10 ms
}

void lcdChar(unsigned char c)
{
        unsigned char c1 = c & 0xF0;            //give c1 hi 4 bits of data
        unsigned char c2 = c & 0x0F;            //give c2 low 4 bits of data
        c1 = c1 / 0x10;         //shift c1 right by 4
        lcdCharWork(c1);
        lcdCharWork(c2);
}

void lcdString(unsigned char ca[])
{
        int i = 0;
        while (ca[i] != '\0')
```

```
        {
                lcdChar(ca[i++]);

        }
}

void lcdGoto(int row, int col)
{
        unsigned char xval = col;
        unsigned char yval;
        xval = xval & 0x1F;
        if (row == 1)
                yval = 0xC0;
        else
                yval = 0x80;
        lcdData(yval + xval);
}

void lcdClear(void)
{
        lcdData(0x01);
}

void lcdInt(int value)
{
    int temp_val;
    int x = 10000;
         int leftZeros=5;

        if (value<0){
        lcdChar('-');
        }

   while (value / x == 0)
    {
      x/=10;
                leftZeros--;
    }


    while ((value > 0) || (leftZeros>0))
    {
        temp_val = value / x;
        value -= temp_val * x;
        lcdChar(temp_val+ 0x30);
         x /= 10;
                  leftZeros--;
   }

        while (leftZeros>0){
        lcdChar(0+ 0x30);
        leftZeros--;
        }

    return;
```

```
}

void initLCD(void)
{
	init_timer();

	/* enable interrupts */
	sei();

	LCDDDR = 0xFF;			// set as output for LCD
	//put in 4bit mode
	lcdData(0x33);
	lcdData(0x32);
	//enable 2 line mode
	lcdData(0x2C);
	//turn everything on
	lcdData(0x0C);
	//Clear
	lcdData(0x01);
}
```

Sonar Header File

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "mikeSonar.h"

void sendPulse(void)
{
	SONAR = 0x02;  //send burst
	int i = 0;
	while (i < 100)
		i++;
	SONAR = 0x00;  //end burst
}

int waitPulse(void)
{
	int i = 0;
	while ((SONARIN & 0x01) == 0x00);
	while ((SONARIN & 0x01) == 0x01)
	{
		i++;
	}
	return i/20;
}

int sonarDist(void)
{
	int last0, last1, last2, last3, last4, last;
	sendPulse();
	last0 = waitPulse();
	sendPulse();
	last1 = waitPulse();
	sendPulse();
	last2 = waitPulse();
	sendPulse();
	last3 = waitPulse();
	sendPulse();
	last4 = waitPulse();
	last = (last0+last1+last2+last3+last4)/5;
	return (last);
}

void initSonar(void)
{

	SONARDDR = 0x02;	//set Sonar(1) to out, Sonar(0) to input
}
```

Motor and Servo Header File

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "ThomasMotors.h"

void initMotor(void)
{	ICR1 = 20000;
	TCCR1A = 0xFC;
	TCCR1B = 0x12;
	TCNT1 = 0x0000;
	DDRB = 0xFF;
	ms_sleep(1000);
	moveMotor(0);}

void moveMotor(int amnt) //-100 for far left, 100 for far right
{
	int tempPOS=M_CENTER;
	float LeftAMT=(-amnt*0.01);
	float RightAMT=(amnt*0.01);
	if (amnt>0)  {tempPOS=tempPOS-RightAMT*farRight;	}
	else{tempPOS=tempPOS+LeftAMT*farLeft;			}
	MOTOR=tempPOS;
}

void centerMotor(void)
{MOTOR = M_CENTER;}

void initServo(void)
{
	ICR1 = 20000;
	TCCR1A = 0xFC;
	TCCR1B = 0x12;
	TCNT1 = 0x0000;
	DDRB = 0xE1;
	servoPOS = S_CENTER;
}

void moveServo(int amnt) //-100 for far left, 100 for far right
{
	int tempPOS=S_CENTER;
	float LeftAMT=(-amnt*0.01);
	float RightAMT=(amnt*0.01);
	if (amnt>0)  {tempPOS=tempPOS-RightAMT*farRight;	}
	else{tempPOS=tempPOS+LeftAMT*farLeft;			}
	servoPOS=tempPOS;
}

void centerServo(void)
{
	servoPOS = S_CENTER;
}
```