Final Report

S.A.R.A.H.
Search and Recovery Autonomous Hovercraft

Daniel Collotte
EEL 4665/5666
Intelligent Machines Design Laboratory

TAs: Ryan Stevens
Tim Martin
Josh Weaver

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Table of Contents

Abstract

Given a straightforward task of building an autonomous robot, I decided to complicate things by building one that hovers! The result of this was SARAH, an autonomous metal detecting hovercraft. SARAH uses a gyroscope and two sonars for navigation, and I specify how these sensors integrate. In addition, I had to implement several novel solutions to problems revolving around the key requirement to carrying out any inexact science: adjustability. In this report I specify my design for a skirt attachment gasket, metal detector mount, and balancing weights- all adjustable.

Executive Summary

SARAH (Search and Recovery Autonomous Hovercraft) is a fully autonomous metal detecting robot. Once activated, SARAH will randomly roam an area searching for metal, and upon finding it notifies the user by flashing her lights.

SARAH is based on a 1" thick, 24" diameter circular wood platform. A "bag" type hovercraft skirt and lift ring, the aforementioned platform, a skirt gasket, a motor mount, and electronics harness are sandwiched together to form her. Attached to this platform by a hinge is a parallel mechanism with a modified metal detector attached to it. Also attached to the front of the gasket is two LV-EZ2 sonars, which are used for object avoidance.

SARAH is coded to perform in a pseudo multithreaded manner, employing various "simultaneously" executing processes as a part of one unified control system, with each process having a clear and distinct purpose from the others.

In order to notify the user when metal is detected, I developed a custom RGB LED driver based on 74HC'595 shift registers. This, along with custom driver functions allows me to independently control the lights, and consequentially what data the user is provided with through color and light codes.

Introduction

SARAH is an autonomous metal searching hovercraft- she searches an area in a random search pattern, using sonar and a gyroscope to avoid objects and control it's movement. Her creation was inspired by the timeless image of the beachfront metal detector junkie; the type of person who might spend hours walking around, metal detector in hand, hoping for the find of their life. Now, perhaps, our fanatic may sit back and enjoy a tasty beverage while a robot does all the work. However, SARAH is not all fun and games. During her creation, it became clear she might also find a more noble cause- landmines injure and kill many people every year [1], but if scaled up, SARAH could help fight that. Even one or two car-sized SARAHs would be able to assist in finding and clearing out a considerable number of mines in an affected area. Because SARAH floats, she would not trigger any of them.

Integrated System

SARAH is designed to rapidly react to sensor data in order to scan an area for metal while avoiding obstacles, and notify the user when it discovers some. The main components of the system are three Rule 240 bilge blowers, a Sabertooth 1.0 motor driver, an Epiphany DIY board (Xmega64A1), an IDG500 gyroscope, a custom LED driver, and a CEN-TECH 97245 metal detector.

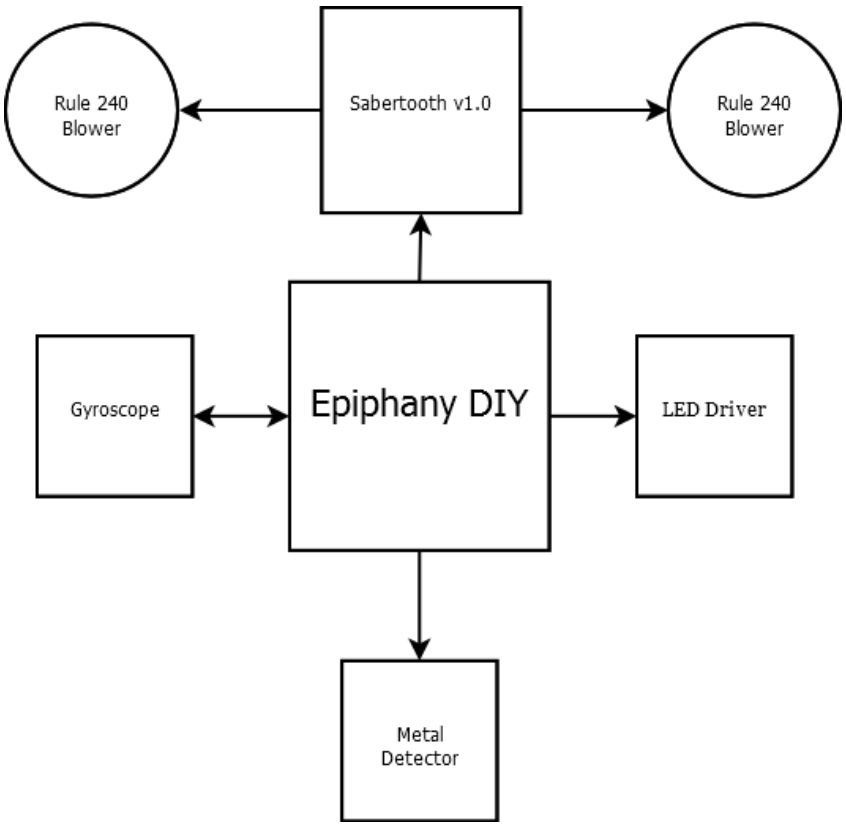

Figure 1 – Block diagram of SARAH's components.

SARAH has four processes that run in a pseudo-multithreaded manner; (1) the sensor process, which reads in the latest value from the sensors (sonar and gyroscope), and calculates averaged values which other processes may use, (2) the main process, which constantly checks these sensor values and adjusts the motors accordingly, (3) the metal detection process, which assumes control whenever metal is detected, and (4) the velocity minimization process, which puffs air in reverse whenever SARAH gains too much speed.

Mobile Platform

SARAH is constructed on a circular hovering platform 24” in diameter, inspired by the common “science project” bag skirt hovercraft [2]. All components of SARAH are mounted in an approximate balance around the center of lift. In addition, two Velcro strips are placed at the rear of the robot so that lead fishing weights (Figure 2) may be fixed as needed for balance.

Figure 2 - 8oz lead fishing weights used to balance SARAH.

The platform’s hover skirt (Figure 3) is constructed from windproof ripstop nylon used in the construction of kites. The skirt is attached to the platform by means of a custom-designed circular retainer.(Figure 4) The retainer’s underside has standard household rubber weather-strip, which helps seal the skirt. Normally skirts on hovercraft of this design are attached by staples, but this design was chosen so that the skirt could be “tightened” and pulled in. I was unsure of the performance of various skirt heights, and this method allowed me to quickly and non-irreversibly attach the skirt to find the optimal height.

Figure 3 - Construction of the bag skirt.

Figure 4 - Skirt gasket without rubber seal.

The hovering design allows SARAH to cover a flat long area, even low density ground, with relative ease and speed compared to a human searching for similar objects with a handheld metal detector.

Figure 5 - SARAH

The CEN-TECH metal detecting module is mounted on a parallel mechanism on the front of the robot, allowing for the height to be adjusted relative to the inflated height of the robot. In addition, the parallel mechanism is attached by a hinge to the main robot platform, allowing it to flex when the skirt is deflated. The hinge is also important because it provides lee-way when striking uneven terrain (or a wall) and the robot will less likely suffer damage.

Power is provided by a high continuous current discharge Lithium Polymer battery, rated at 11.1V 2200mAh 25C.

Actuation

Lift is provided by a powerful 4” diameter Rule 240 bilge blower, which provides 235 CFM when running at 12V @ 4.3A. Thrust is provided by two of the same blowers, mounted along the diameter of the platform on opposing sides. When metal is detected SARAH will stop over it and flash her lights to notify you there is metal underneath it, and then resume her search.

The lift blower is permanently wired to the power, whereas the thrust blowers are controlled through the Sabertooth motor driver, which was necessary because the Epiphany DIY’s onboard motor drivers cannot provide enough start-up current, despite technically being capable of the necessary continuous current. The Sabertooth is controlled via serial communication. (*Appendix B: Motor Driver*)

In the interest of not dealing with wireless communication, SARAH provides feedback on all her actions using four RGB (Red, Green, Blue) Light Emitting Diodes, driven by custom software (*Appendix C: LED Driver*) and hardware. (Figure 6)

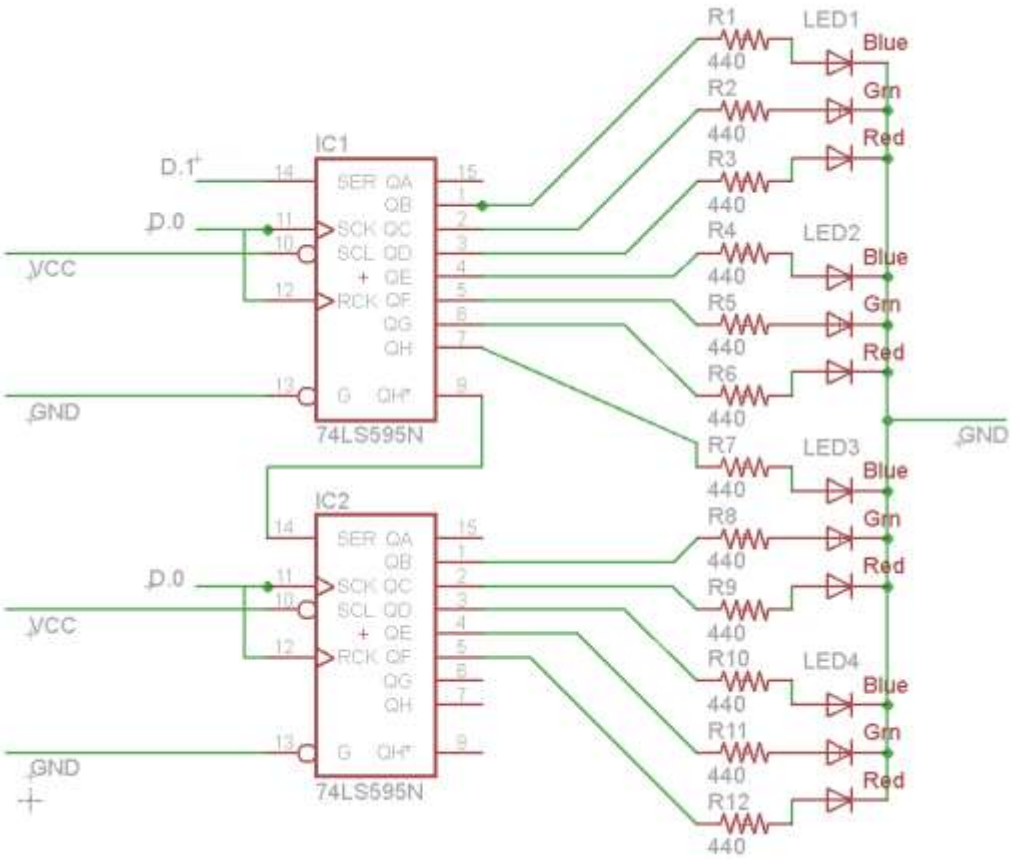

Figure 6 - LED Driver Schematic.

Interesting Note: The LED driver is based on a design originally intended by me for use in a disco dance floor!

Sensors

List of Sensors

1. IDG500 (2 axis gyroscope)
2. Maxbotix LV-EZ2 (ultrasonic rangefinder)
3. CEN-TECH 97245 (metal detector)

Gyroscope – IDG500

Application
SARAH uses the yaw information from the gyroscope to prevent wild spins, or what I like to call "chaotic conditions." When a chaotic condition is encountered (ex: Something is grazed or snags the parallel mechanism, sending SARAH into a wild spin), the yaw information allows SARAH to minimize spin, attempt to recover, and continue scanning.

Theory
The gyroscope uses a lithographically constructed MicroElectroMechanical System (MEMS) which vibrates. Similar to a Foucault pendulum, the vibrating MEMS structure appears to move relative to the reference frame of detector MEMS apparatus, allowing rotation in a plane to be detected.

Software
See *Appendix D: Sensor Filtering and Update*

Performance
We can see below in *Experimental Layout and Results, Figure 10* that although this gyroscope provides fairly noisy data, a simple running average filter dramatically increases its usefulness. (Also note the poor performance of a MEMS accelerometer, which was removed from SARAH in a later design revision.)

References

- Vendor: InvenSense (http://invensense.com/mems/gyro/idg500.html)
- Part #: IDG-500
- Phone: (408)-988-7339

Ultrasonic Rangefinder - LV-EZ2

Application
The LV-EZ2 ultrasonic rangefinder offers a relatively wide field of detection (nearly the width of SARAH). Two of these in a "daisy chain" configuration [3] are used to provide SARAH with object avoidance capabilities.

Theory
The LV-EZ2 calculates the range of an object in front of it by sending out an ultrasonic "click" and then immediately listening for its echo. Bats use the same principle, echolocation, to supplement their poor eyesight. The sensor does this many times per second and provides an analog voltage corresponding to the distance. The claimed scaling factor is *Vcc/512* which would give a resolution of ~9.8mV/in on a 5V supply. [4]

Software
See *Appendix D: Sensor Filtering and Update*

Performance
We can see below (Figure 7) that the LV-EZ2 provides a linear response within its operational range of twenty feet. Filtering this data removes any erroneous values.

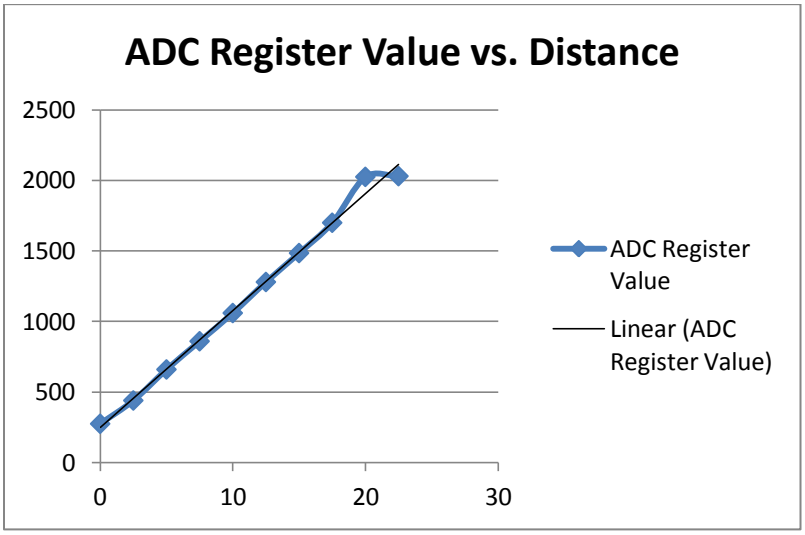

Figure 7 - Performance of the LV-EZ2 ultrasonic rangefinder.

References

- Vendor: Maxbotix (http://www.maxbotix.com/products/LV.htm)
- Part #: LV-EZ2
- Phone: (218)-454-0766

Metal Detector – CEN-TECH 97245

Application
This sensor will be used to find metal objects. When scaled up, this would even enable me to search for treasure or land-mines.

Theory
The metal detector I modified operates on the theory of beat frequency oscillation. It utilizes two coils tuned to oscillate near the same frequency: a search coil and a reference coil. These two

oscillators are summed and passed through a low pass filter. Normally, the output has no signal, but when a metal object disturbs the search oscillator, a signal is produced.

Software

No special software is needed, as the metal detector has been modified to trigger at TTL voltage levels. The buzzer (labeled "BZ" in Figure 8 below) was removed and replaced with a simple resistive divider, providing a TTL low level when triggered. (Normally, the circuit is pulled high.)

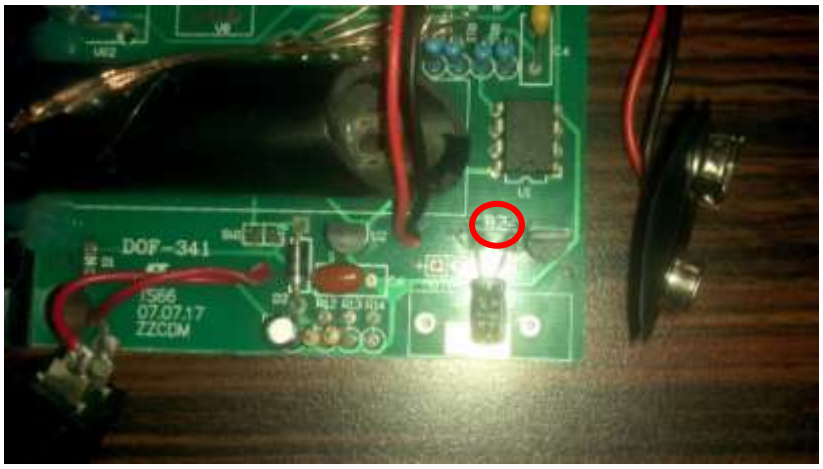

Figure 8 - Internals of the CEN-TECH 97247

Performance

The metal detector has an effective detection distance of ~2" from its search and reference coils. This was tested using 1/32" copper sheet, and 1/16" stainless steel sheet.

Behaviors

SARAH uses a random search pattern, created through stochastic interactions with the environment, to scan a wide flat area for metal objects, and notify the user where they are. When an obstacle is encountered, SARAH makes her best effort to turn and hover away from the obstacle. Once a proximity threshold is passed, SARAH's turn response is inversely proportional to the distance from the obstacle (*Appendix E: Proportional Drive*).

Experimental Layout and Results

Ultrasonic Rangefinder Testing

Objective

The objective of this experiment was to determine what ADC register values are created by various distances detected by the ultrasonic rangefinder.

Procedure

A measuring tape was used to mark increments of 2.5' from the ultrasonic rangefinder. Next, a piece of foam board (Figure 9) was held at each distance, and the ADC register value was noted. These values (Table 1) were reported using the code in *Appendix A*.

Figure 9 – The range finding experimental setup.

Table 1 – Reported ADC register values for various distances.

Distance (feet)	ADC Register Value
0	275
2.5	440
5	660
7.5	860
10	1060
12.5	1280
15	1485
17.5	1700
20	2025
22.5	2030

Conclusion

As noted in the sensor section (Figure 7), the rangefinder has a linear response within its operational range, and as a result, the actual distance of any object is easy to determine.

Gyroscope Data Acquisition

Objective

The objective of this experiment was to gain a subjective "feel" for the sensitivity of the gyroscope and determine how to put the data it provides to use.

Procedure

A small momentary push button was connected a long wire, and that wire to the Epiphany DIY. When the button is pushed, SARAH records the eight seconds of data from the gyroscope, and turns on an LED to indicate that it is indeed recording. When SARAH is done recording, she is attached via USB to a PC running serial terminal software. The button is then pressed again, dumping all eight seconds of raw data to the PC. (Appendix F: Sensor Data Acquisition) This experiment was performed for various torques and moments, providing a satisfactory "feel" of the gyroscope's sensitivity.

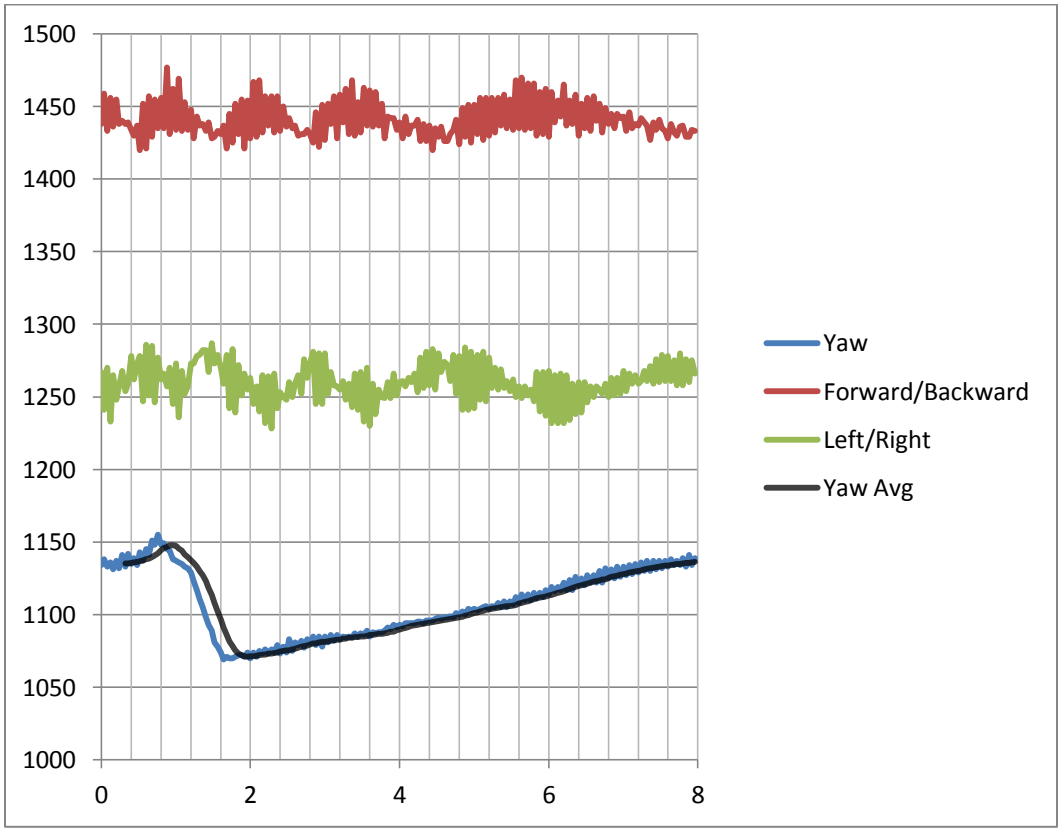

Figure 10 - Gyroscope and accelerometer data collected from SARAH.

Once this data was acquired, it was imported into Micrsoft Excel and subject to various filters, of which, the eight-point running average filter provided the most usable data.

Conclusion

Unfortunately, as *Figure 10* shows, the gyroscope is in actuality an angular accelerometer, and cannot be used as I originally wished to use it. I wanted to use it to give SARAH a sense of which direction to travel, or a reference "straight direction," so to speak. However it does provide adequate data, especially after filtering, in order to help SARAH minimize chaotic conditions.

Conclusion

SARAH has achieved the primary goal I set out to achieve- she searches for metal while the user relaxes. Using sonar and a gyroscope, SARAH is able to navigate randomly, but safely around a flat area. In such an area, movement is so easy for SARAH that I actually had to write code to slow her down!

A first prototype always has limitations, and SARAH is no exception. If a surface undulates too much, SARAH's skirt will not make a seal with the ground and she will not be able to move. This can be improved in future versions through the use of a segmented or "finger" skirt design. Another issue SARAH has is her complete unawareness of velocity, only acceleration. In a future revision I would try modifying an optical or laser mouse and mounting it on the bottom of SARAH in order to provide accurate velocity and possibly even displacement data, which would allow SARAH to move much more gracefully. Finally, although the LEDs are bright, the user must still be watching the robot to find out when metal is detected. I'd like to improve this by having SARAH both emit an audible beep and leave some sort of marker at the location of the metal.

For future students, I'd recommend avoiding MEMS inertial devices, both gyroscopes and accelerometers. For the purposes of a robot like this, these devices are a waste of resources and are almost completely useless. I was luck to glean some usefulness from my gyroscope, but I feel it was far outweighed by the time I spent learning the device.

Documentation

[1] M. Newman, "Landmine Casualties," 2006. [Online]. Available: http://www.worldmapper.org/posters/worldmapper_map290_ver5.pdf. [Accessed 23 April 2012].

[2] W. J. Beaty, "HOVERCRAFT SCIENCE FAIR PROJECT," 1991. [Online]. Available: http://amasci.com/amateur/hovercft.html. [Accessed 23 April 2012].

[3] MaxBotix Inc., "Chaining MaxSonar Sensors," 27 August 2008. [Online]. Available: http://www.maxbotix.com/documents/LV_Chaining_Constantly_Looping_AN_Out.pdf. [Accessed 23 April 2012].

[4] MaxBotix Inc., "MaxBotix Ultrasonic Sensors," 2011. [Online]. Available: http://www.maxbotix.com/documents/MB1020_Datasheet.pdf. [Accessed 23 April 2012].

Appendices

Appendix A: ADC Echo

```
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>

#include "uart.h"
#include "ADC.h"


int main (void)
{
	board_init();
	uartInit(&USARTC0,115200);//USB UART Init
	ADCsInits();
	sei();
	int i;
	while (1)
	{

		for(i=0;i<8;i++)
		{
			fprintf(&USB_str,"ADC channel %d = %d\r\n\r\n",i,analogRead_ADCA(i));
		}

		_delay_ms(100);//delay 100 milliseconds
	}
}
```

Appendix B: Motor Driver

myMotor.h

```
#ifndef MYMOTOR_H_
#define MYMOTOR_H_

#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>


#define LEFT                    1
#define RIGHT                   0
#define FORWARD                 1
#define BACKWARD                0
#define STOP                    2
#define waitTxD1()              while(!(USARTD1_STATUS & USART_DREIF_bm))

void motorSet(char chooseMotor,int speed, char direction);
```

myMotor.c

```
#include "myMotor.h"

void motorSet(char chooseMotor,int speed, char direction)
 {
        waitTxD1(); //Wait until Tx buffer is empty
        //Lower 128 bits are left motor, Upper are Right motor
        //1, 128 are forward full
         switch (chooseMotor)
         {
                 if (speed > 64)speed=64;
                case LEFT:
                        if (direction == FORWARD)
                        {
                                USARTD1.DATA = (65-speed);
                        }
                        else
                        {
                                USARTD1.DATA = (63+speed);
                        }
                        break;
                case RIGHT:

                        if (direction == FORWARD)
                        {
                                USARTD1.DATA = (193-speed);
                        }
                        else
                        {
                                USARTD1.DATA = (191+speed);
                        }
                        break;
                case STOP:
                        USARTD1.DATA = 0;
                        break;
                default:
                        USARTD1.DATA = 0;
                        break;

         }
 }
```

Appendix C: LED Driver

LEDs_595.h

```
#ifndef LEDS_595_H_
#define LEDS_595_H_

#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>

#define RED             0x4
#define GREEN   0x2
#define BLUE    0x1
#define BLACK   0
#define WHITE   0x7
#define YELLOW  0x6
#define MAGENTA 0x5
#define CYAN    0x3


//Will be used in the future to set single LEDs, instead of all 4
volatile unsigned int LED_FRONT;
volatile unsigned int LED_BACK;
volatile unsigned int LED_LEFT;
volatile unsigned int LED_RIGHT;

void led_clk(void); //Clock the shift+storage registers on the 74hc595
void clr_leds(void); //Shifts 0 to all the bits of the driver to blank the LEDs.
void set_leds(unsigned int front_led, unsigned int back_led, unsigned int left_led, unsigned int
right_led); //Used to set the 4 LEDs to different values.
void init_leds(void);  //initializes the pins for the LED driver
void update_leds(void); //Update LEDS from global vars.


#endif /* LEDS_595_H_ */
```

LEDs_595.c

```
#include "LEDs_595.h"

void led_clk()
{
                PORTD.OUTSET = PIN0_bm;
                _delay_us(1);
                PORTD.OUTCLR = PIN0_bm;
                _delay_us(1);
}

void update_leds(void)
{
        unsigned int output = 0;
        char bm = 0x01;
        int i;
        //output = (front_led<<11) | ((back_led&0x6)<<8) | ((back_led&0x1)<<8) | (left_led<<5) |
(right_led<<1);
        output = (LED_FRONT<<10) | (LED_BACK<<7) | ((LED_LEFT&0x4)<<4) | ((LED_LEFT&0x3)<<3) | LED_RIGHT;
        //printf("%x", output);
        for(i=0;i<13;i++)
        {
                if ((output & bm) == 0)
                {
                        PORTD.OUTCLR = PIN1_bm;
                }
                else
                {
```

```
                        PORTD.OUTSET = PIN1_bm;
                }

                output = output>>1;
                _delay_us(1);
                led_clk();
        }
        PORTD.OUTCLR = PIN1_bm;
        led_clk();
        led_clk();
}


void set_leds(unsigned int front_led, unsigned int back_led, unsigned int left_led, unsigned int
right_led)
{
        unsigned int output = 0;
        char bm = 0x01;
        int i;
        //output = (front_led<<11) | ((back_led&0x6)<<8) | ((back_led&0x1)<<8) | (left_led<<5) |
(right_led<<1);
        output = (front_led<<10) | (back_led<<7) | ((left_led&0x4)<<4) | ((left_led&0x3)<<3) | right_led;
        //printf("%x", output);
        for(i=0;i<13;i++)
        {
                if ((output & bm) == 0)
                {
                        PORTD.OUTCLR = PIN1_bm;
                }
                else
                {
                        PORTD.OUTSET = PIN1_bm;
                }

                output = output>>1;
                _delay_us(1);
                led_clk();
        }
        PORTD.OUTCLR = PIN1_bm;
        led_clk();
        led_clk();
}

void clr_leds()
{
        int i;
        PORTD.OUTCLR = PIN1_bm;
        for (i=0;i<17;i++)
        {
                led_clk();
        }
}

void init_leds()
{
        PORTD.DIRSET = PIN0_bm | PIN1_bm;
}
```

Appendix D: Sensor Filtering and Update

Note: This code is placed in a 40ms timer interrupt.

```
printf("Interrupt FIRED!\n\r");
printf("RAW L/R = %d|%d\r\n", analogRead_ADCA(1), analogRead_ADCA(2));
int i, temp;
rangeDataL[rangeIndexL] = analogRead_ADCA(1);
rangeIndexL++;
if (rangeIndexL==8) rangeIndexL = 0;
temp = 0;
for (i=0;i<8;i++)
{
        temp += rangeDataL[i];
}
rangeAvgL = temp>>3;

rangeDataR[rangeIndexR] = analogRead_ADCA(2);
rangeIndexR++;
if (rangeIndexR==8) rangeIndexR = 0;
temp = 0;
for (i=0;i<8;i++)
{
        temp += rangeDataR[i];
}
rangeAvgR = temp>>3;

YawData[YawIndex] = analogRead_ADCA(4);
YawIndex++;
if (YawIndex==8) YawIndex = 0;
temp = 0;
for (i=0;i<8;i++)
{
        temp += YawData[i];
}
YawAvg = temp>>3;

metalSense = analogRead_ADCA(0);
printf("L = %d, R = %d, Y = %d, Metal = %d\r\n", rangeAvgL, rangeAvgR, YawAvg, metalSense);
```

Appendix E: Proportional Drive

Note: This code is executes a maximum of once every 10 ms, and is meant to be placed in an infinite main loop.

```
//Movement code
if ((rangeAvgR <= Range_t) || (rangeAvgL <= Range_t))
{
        TCD1.CTRLA = TC_CLKSEL_OFF_gc;
        TCD1.CNT = 0;

        //LEDs
        if (rangeAvgL < 280) //488 = 3 ft
        {
                LED_LEFT= RED;
        }
        else if(rangeAvgL < Range_t)
        {
                LED_LEFT= YELLOW;
        }

        if (rangeAvgR < 280) //488 = 3 ft
        {
                LED_RIGHT       = RED;
        }
        else if(rangeAvgR < Range_t)
        {
                LED_RIGHT       = YELLOW;
        }
        //end LEDs

        if(rangeAvgL > rangeAvgR)
        {
                motorSet(LEFT, 45+(Turn_power-((rangeAvgR-250)/(Range_t/Turn_power))),
BACKWARD );
                motorSet(RIGHT, 45+(Turn_power-((rangeAvgR-250)/(Range_t/Turn_power))),
FORWARD );
        }
        else
        {
                motorSet(RIGHT, 45+(Turn_power-((rangeAvgL-250)/(Range_t/Turn_power))),
BACKWARD );
                motorSet(LEFT, 45+(Turn_power-((rangeAvgL-250)/(Range_t/Turn_power))),
FORWARD );
        }
}
else
{

        LED_FRONT       = GREEN;
        LED_BACK= GREEN;
        LED_LEFT= BLACK;
        LED_RIGHT       = BLACK;

        motorSet(LEFT, 48, FORWARD);
        motorSet(RIGHT, 48, FORWARD);

        TCD1.CTRLA = TC_CLKSEL_DIV1024_gc;
}
update_leds();
LED_FRONT       = GREEN;
LED_BACK= GREEN;
LED_LEFT= BLACK;
LED_RIGHT       = BLACK;
_delay_ms(10);
```

Appendix F: Sensor Data Acquisition

```
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>

#include "uart.h"
#include "ADC.h"
#include "myMotor.h"

#define DbLedOn()               (PORTR.OUTCLR = 0x02)           //Turns the debug led on.  The led is
connected with inverted logic
#define DbLedOff()              (PORTR.OUTSET = 0x02)           //Turns the debug led off.  The led is
connected with inverted logic
#define DbLedToggle()   (PORTR.OUTTGL = 0x02)           //Toggles the debug led off.  The led is
connected with inverted logic

#define SAMPLES         400
#define SAMPLERATE      5 //in ms, total time of samples*samplerate


#define MOTORTEST //If defined, powers the motors during the test



int main (void)
{
        board_init();
        DbLedOn();
        uartInit(&USARTC0,115200);
        uartInit(&USARTD1, 19200);
        ADCsInits();
        stdout = &uartC0_str;
        sei();
        int i;
        int j=3;
        char check = 0;
        int sensorData[SAMPLES][3] = {0};
        DbLedOff();
        PORTD.DIRCLR = 0x01;
        PORTD.PIN0CTRL = 0x10;
        PORTD.DIRSET = PIN0_bm;

        motorSet(STOP, 1 , 1);

        while (1)
        {
                if(PORTD.IN & PIN0_bm)
                {
                        #ifdef MOTORTEST
                        motorSet(LEFT, 50, FORWARD);  //Included only in forward/backward push tests
                        motorSet(RIGHT, 50, FORWARD); //Included only in forward/backward push tests
                        #endif


                        DbLedToggle();
                        for (i=0;i<SAMPLES;i++)
                        {
                                #ifdef MOTORTEST
                                if (i==(SAMPLES/2))                                             //Included only
in forward/backward push tests
                                {                                                               //
                                        motorSet(STOP, 1 , 1);   //
                                }                                                               //
```

```
                                #endif

                                sensorData[i][0] = analogRead_ADCA(4); //Yaw norm: ~1130
                                sensorData[i][1] = analogRead_ADCA(6); //Forward/back
                                sensorData[i][2] = analogRead_ADCA(7); //Left/right
                                _delay_ms(SAMPLERATE);
                                //Acquire 200 points of data, each channel
                        }
                        check = 1;
                        DbLedToggle();
                        while (check != 0)
                        {
                                //wait for another button press, then print the data serially.
                                if (PORTD.IN & PIN0_bm)
                                {
                                        DbLedToggle();
                                        fprintf(&USB_str,"TIME(ms) ,  YAW  ,  FB  ,  LR  \r\n");
                                        for (i=0;i<SAMPLES;i++)
                                        {
                                                fprintf(&USB_str,"%d , %d , %d ,
%d\r\n",i*SAMPLERATE,sensorData[i][0], sensorData[i][1], sensorData[i][2]);
                                        }
                                        check = 0;
                                        j++;
                                        DbLedToggle();
                                }
                        }
                }

        }


}
```