Intelligent Machine Design Laboratory

EEL5666.6515

Instructors: Dr. A. Antonio Arroyo

Dr. Eric M. Schwartz

TAs: Mike Pridgen

Thomas Vermeer

Patrick Connaway

Lawntonomous Maximus

Abstract

Lawntonomous Maximus is an autonomous lawn mower. He is capable of mowing a typical well maintained Floridian lawn. The robot is much smaller than a standard gas powered push mower, so it has limited mowing capabilities. Lawntonomous uses ultrasonic sensors for collision avoidance and a wireless dog fence to help define the cutting area.

Executive Summary

Lawntonomous Maximus is a downsized autonomous lawnmower. Initially I wanted to use my grandmother's old lawn mower for this but due to overwhelming mass of such a task I decided to do something on a smaller scale.

I built the mechanical system for the robot from scratch. I took a part a weed wacker I purchased from Dr. Schwartz and bolted it to the mobile platform. I purchased sheet steel from Rodger's Welding and had them shear it. I welded together the components in the Machine shop on campus. All of the wooden components were made on the T-Tech. I designed them in Solidworks and had them cut by the TAs. I made the drive hubs I used in the Machine shop as well. My sister painted the robot yellow and green for me, so by media day it was looking great. I was successful in the design of my mechanical system but I want to achieve more with the electrical and software systems.

I wanted to attempt some fuzzy logic and achieve wall following but the ultrasonic sensors I used were too unreliable. The main program was incredibly simple because of this. I was pleased with how well collision avoidance was working, but would have liked to have done softer turns.

The biggest let down was the wireless dog fence I really wanted to demonstrate this sensor functioning. I was still pleased with my demo but would have been happier to see the special sensor working. Overall I was pleased with the project and look forward to continuing it this summer.

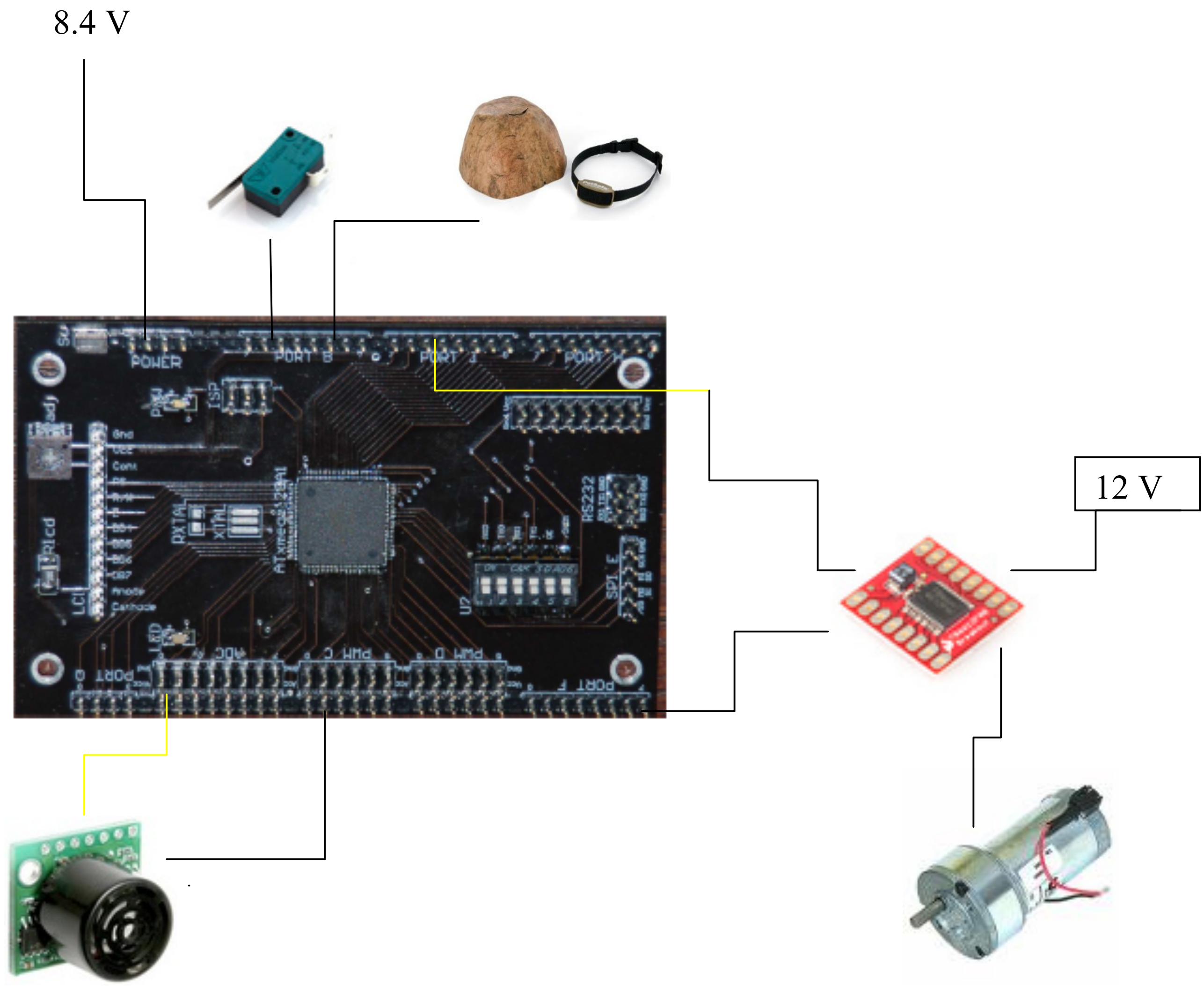
Introduction

When I was a boy, too young to get a real job, I would mow lawns. My grandmother would pay my brother and me to do a lot of yard work. She has about an acre of land, so mowing her lawn was a lucrative chore at the time. My brother and I would each use a push mower and it would take hours to finish the job. I believe this is why I came up with the concept of an autonomous lawnmower.

I conceived the idea of an autonomous lawnmower in the fall term of 2009 and then heard about this class. I felt as though it was the perfect way to materialize my ideas. I was a little disappointed when I found out an autonomous lawnmower was already on the market, but I still felt compelled to build one myself.

I built a downsized mower because of my budget and the amount of time I can invest. Initially I intend on programming the robot to be capable of obstacle avoidance and position determination. I want to program it such that it can mow the small lawn at my apartment complex.

Integrated System



The integrated system is relatively simple. The ultrasonic sensors just determine when an obstacle is in the path of the robot. The speed of the robot is a function of the analog voltage returned from the ultrasonic sensor. The bump switches and wireless fence signal basically cause the same reaction from the robot. They both just cause the robot to reverse then make a turn between 90 and 270 degrees based on the value from the front center sonar. I want to integrate my GPS sensor into the system this summer.

Mobile platform

The mobile platform is made of 16 gauge sheet steel (See figure 1). The two side pieces are welded into place. The drive motors are mounted on these components. When I originally designed the platform I didn't account for the space the procured by the welds. This caused a slight misalignment of the drive motors, so the robot would not go straight.

Figure 1

The electronics are mounted on balsa wood which is bolted to the platform. The PVR Board is mounted in the front behind a bumper equipped with three Ultrasonic Sensors and two bump switches. Originally I had an identical bumper mounted on the rear but the sensors were only used when Lawntonomous was reversing. This proved to be impractical, so I reduced this to a much simpler configuration. I just used one ultrasonic sensor mounted in the center.

Actuation

Two Merkle Korff motors are used to drive to robot. They are mounted onto the sheet steel welded on the sides I machined two identical aluminum drive hubs and milled holes in the drive wheels. The hubs transfer the torque via a set screw which rests on the face cut out of the shaft of the motor.

The motors are rated to output 18 lb in of torque at 16 rpm at peak voltage. From empirical data Lawntonomous moves about 1 ft/s at max speed. The drive wheels were purchased from a surplus distributor online they are 8" lawnmower wheels. The motor driver provided from the lab. Sean Fruct wrote the source and header files I used for the motor driver.

I took apart a weed wacker and mounted it on the platform for the mowing system. The circuitry for this system was fairly complex. When the robot is turned on the mowing system will also turn on. There was a kill switch located atop the robot for emergency situations.

Sensors

The robot implements sonar for obstacle avoidance. This is the most practical option, because the robot operates outside. Three sensors will be mounted on the front of the mower (one centered and one to the left and right). One is mounted in the rear. Initially I wanted to attempt some fuzzy logic but my

sensors were not reliable enough. In the end I used a single threshold value for all the sensors and averaged it over 5 values. The collision avoidance is very reliable. I used the MaxBotix ultrasonic sensors (Product code: RB-Max-01) (See figure 3).

Two bump switches are mounted on the front bumper. They have only been used for turns made too close to walls. Initially I wanted to use G.P.S. for the special sensor. I purchased a less expensive sensor that is only accurate within a few meters. I struggled with the software for a few weeks and then decided to use a different sensor. In hind sight I should have bribed a CSE major to help me.

I attempted to implement a wireless pet fence as my special sensor (See figure 4). Originally I wanted to use it to confine the perimeter in which Lawntonomous cut. This proved to be a more challenging task than I anticipated. The wireless transmitter was designed to keep pets out of one's garden. The transmitter would shock ones pet if it entered the area. The shock would stop after about 30 seconds which proved to be a problem. I was unable to turn off the sensor and turn it back on to regain the signal.

Figure 4

I then tried to use the transmitter to repel the robot. This was a much simpler task. I was able to get a clean signal from the shock collar without the need of a capacitor. When the robot was in range the signal would go high and I could read it through an I/O port. The system was incredibly simple and I was able to get it properly functioning the night before media day. I went to test it in the Rotunda but it was locked by that hour. The next day I was unable to get the sensor to function when the robot was moving. I could get a great signal when the robot was not moving. I am uncertain as to why this occurred. I should have spent the extra money and bought a fence meant to confine a pet as oppose to repel it.

Behaviors

The robot is programmed to do random mowing. The numbers input to the motor functions are a function of the values returned from the ultrasonic sensors. This system is incredibly simple and proved to work great for my yard. There is about a 5 to 10 degree slope my yard this proved to be no challenge for the high torque motors. Although the thick tree roots that have grown up and out from the ground proved to be invisible to the ultsonic sensors. The front drive wheels make over but the weed wacker gets caught onto them.

The bump switches just cause Lawntonomous to reverse and the turn in a random direction based on the value from the front center ultrasonic sensor. I am still working with the wireless fence if I cannot get it fully functional I am going to use it to keep my cats from getting outside.

Experimental Layout and Results

The Ultrasonic sensors I used were proved to be functional for basic collision avoidance but when I attempted more complex behaviors they were not very reliable. I attempted to employ a function for softer turns, but was unable to successfully execute it when it was integrated into the main program.

The wireless dog fence involved a lot of experimenting. I took apart the shocker collar to get the circuit board out of it. I hooked it up the oscilloscope and found the node that gave the best signal when the receiver was in range of the transmitter. When I collected the signal through an I/O port the signal would go high when in range which made this sensor seemingly too easy. I was able to use this signal to tell Lawntonomous which areas of the yard are off limits.

Unfortunately I was only successful with it once the night before media day in the IMDL lab. I went to test it in the Rotunda that night but it was locked. The next day I was unable to replicate the experiment. I could get Lawntonomous to receive the signal when he wasn't moving, but once he began translating he could not pick it up.

Conclusion

The goal of this project was to build a robot capable of maintaining a yard by mowing it a few times a week during the summer and a couple times a month during the winter. I was successful in this sense. I have a lot of work to do on Lawntonomous over the summer to make a more effective mower.

I am content with what was accomplished this semester. The mechanical design was the most successful portion of the project. The robot was aesthetically pleasing. The cutting system I designed was much more attractive and equally effective relative to the original weed wacker. There are few things I could improve with this aspect. The sensor suite is pretty lame right now. I intend on making the wireless fence fully functional and adding getting some assistance with the GPS sensor. I also have built a circuit for an "Edge of the World" sensor. It will recognize a sudden change in elevation to prevent Lawntonomous from going off a cliff. I am pleased with my progress thus far, and look forward to my future work.

Picture Sources

Figures 1: Photo by author

Figure 2:

Maxbotix LV-MaxSonar-EZ1 High Performance Sonar Module Product code : RB-Max-01

http://www.robotshop.us/maxbotix-ez1-ultrasonic-ranger-2.html

Figure 3:

http://www.amazon.com/gp/product/images/B002GQDUBW/sr=8-11/qid=1271818796/ref=dp_image_0?ie=UTF8&n=1055398&s=home-garden&qid=1271818796&sr=8-11

Appendix

Header files used

```
#ifndef  __PVR_h__
#define  __PVR_h__
#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"
#define LCD                    PORTK_OUT
#define LCDDDR          PORTK_DIR
volatile int delaycnt;
void xmegaInit(void);
void delayInit(void);
void delay_ms(int cnt);
void delay_us(int cnt);
void lcdDataWork(unsigned char c);
void lcdData(unsigned char c);
void lcdCharWork(unsigned char c);
void lcdChar(unsigned char c);
void lcdString(unsigned char ca[]);
void lcdInt(int value);
void lcdGoto(int row, int col);
void lcdInit(void);
void ServoCInit(void);
void ServoDInit(void);
void ServoC0(int value);
void ServoC1(int value);
void ServoC2(int value);
void ServoC3(int value);
void ServoC4(int value);
void ServoC5(int value);
void ServoD0(int value);
void ServoD1(int value);
void ServoD2(int value);
void ServoD3(int value);
void ServoD4(int value);
void ServoD5(int value);
void ADCAInit(void);
```

```
int ADCA0(void);
int ADCA1(void);
int ADCA2(void);
int ADCA3(void);
int ADCA4(void);
int ADCA5(void);
int ADCA6(void);
int ADCA7(void);
#endif
   #ifndef  __MotorFunctions_h__
   #define  __MotorFunctions_h__

#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

void drivemotorsinit(void);

void drivemotorA( int mode, int speed);

void drivemotorB( int mode, int speed);

void drive(int direction, int rightspeed, int leftspeed);
   #endif
```

Source Files used

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

/*********
 * Xmega *
 *********/

void xmegaInit(void)
{
        CCP = 0xD8;
        CLK_PSCTRL = 0x00;
        PORTQ_DIR = 0x01;
```

```
	//setup oscilllator
	OSC_CTRL = 0x02;						//enable 32MHz
internal clock
	while ((OSC_STATUS & 0x02) == 0);			//wait for oscillator to
be ready
	CCP = 0xD8;								//write
signature to CCP
	CLK_CTRL = 0x01;						//select internal
32MHz RC oscillator
}

/*********
 * Delay *
 *********/

void delayInit(void)
{
	TCF1_CTRLA = 0x01;						//set clock/1
	TCF1_CTRLB = 0x31;						//enable COMA
and COMB, set to FRQ
	TCF1_INTCTRLB = 0x00;					//turn off
interrupts for COMA and COMB
	SREG |= CPU_I_bm;						//enable all
interrupts
	PMIC_CTRL |= 0x01;						//enable all low
priority interrupts
}

void delay_ms(int cnt)
{
	delaycnt = 0;							//set count value
	TCF1_CCA = 32000;						//set COMA to be
1ms delay
	TCF1_CNT = 0;							//reset counter
	TCF1_INTCTRLB = 0x01;					//enable low
priority interrupt for delay
	while (cnt != delaycnt);				//delay
	TCF1_INTCTRLB = 0x00;					//disable
interrupts
}
```

```
void delay_us(int cnt)
{
	delaycnt = 0;						//set counter
	TCF1_CCA = 32;						//set COMA to be
1us delay
	TCF1_CNT = 0;						//reset counter
	TCF1_INTCTRLB = 0x01;					//enable low
priority interrupt for delay
	while (cnt != delaycnt);				//delay
	TCF1_INTCTRLB = 0x00;					//disable
interrupts
}

SIGNAL(TCF1_CCB_vect)
{
	delaycnt++;
}

SIGNAL(TCF1_CCA_vect)
{
	delaycnt++;
}

/*******
 * LCD *
 *******/

#define LCD				PORTK_OUT
#define LCDDDR			PORTK_DIR

void lcdDataWork(unsigned char c)
{
	c &= 0xF0;						//keep data bits,
clear the rest
	c |= 0x08;						//set E high
	LCD = c;						//write to LCD
	delay_ms(2);						//delay
	c ^= 0x08;						//set E low
	LCD = c;						//write to LCD
```

```
        delay_ms(2);                                        //delay
        c |= 0x08;                                          //set E high
        LCD = c;                                            //write to LCD
        delay_ms(2);                                        //delay
}

void lcdData(unsigned char c)
{
        unsigned char cHi = c & 0xF0;               //give cHi the high 4 bits of c
        unsigned char cLo = c & 0x0F;                       //give cLo the low 4
bits of c
        cLo = cLo * 0x10;                                   //shift cLo left 4
bits
        lcdDataWork(cHi);
        lcdDataWork(cLo);
}

void lcdCharWork(unsigned char c)
{
        c &= 0xF0;                                          //keep data bits,
clear the rest
        c |= 0x0A;                                          //set E and RS
high
        LCD = c;                                            //write to LCD
        delay_ms(2);                                        //delay
        c ^= 0x08;                                          //set E low
        LCD = c;                                            //write to LCD
        delay_ms(2);                                        //delay
        c |= 0x08;                                          //set E high
        LCD = c;                                            //write to LCD
        delay_ms(2);                                        //delay
}

void lcdChar(unsigned char c)
{
        unsigned char cHi = c & 0xF0;               //give cHi the high 4 bits of c
        unsigned char cLo = c & 0x0F;                       //give cLo the low 4
bits of c
        cLo = cLo * 0x10;                                   //shift cLo left 4
bits
```

```
        lcdCharWork(cHi);
        lcdCharWork(cLo);
}


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

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

        if (value<0)
        {
            lcdChar('-');
                value *= -1;
        }

    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 lcdGoto(int row, int col)
{
        unsigned char pos;
        if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
        {
                pos = col;
                if (row == 1)
                        pos += 0x40;
                else if (row == 2)
                        pos += 0x14;
                else if (row == 3)
                        pos += 0x54;
                lcdData(0x80 + pos);
        }
}

void lcdInit(void)
{
        delayInit();                            //set up the delay functions
        LCDDDR = 0xFF;                                  //set LCD port to outputs.
        delay_ms(20);                                   //wait to ensure LCD
powered up
        lcdDataWork(0x30);                              //put in 4 bit mode, part 1
        delay_ms(10);                                   //wait for lcd to finish
        lcdDataWork(0x30);                              //put in 4 bit mode, part 2
        delay_ms(2);                                    //wait for lcd to finish
        lcdData(0x32);                                  //put in 4 bit mode, part 3
        lcdData(0x2C);                                  //enable 2 line mode
        lcdData(0x0C);                                  //turn everything on
```

```
	lcdData(0x01);					//clear LCD
}

/*********
 * Servo *
 *********/

void ServoCInit(void)
{
	TCC0_CTRLA = 0x05;				//set TCC0_CLK to CLK/64
	TCC0_CTRLB = 0xF3;				//Enable OC A, B, C, and D.
Set to Single Slope PWM
							//OCnX = 1 from
Bottom to CCx and 0 from CCx to Top
	TCC0_PER = 10000;				//20ms / (1/(32MHz/64)) =
10000.  PER = Top
	TCC1_CTRLA = 0x05;				//set TCC1_CLK to CLK/64
	TCC1_CTRLB = 0x33;				//Enable OC A and B.  Set to
Single Slope PWM
							//OCnX = 1 from
Bottom to CCx and 0 from CCx to Top
	TCC1_PER = 10000;				//20ms / (1/(32MHz/64)) =
10000.  PER = Top
	PORTC_DIR = 0x3F;				//set PORTC5:0 to output
	TCC0_CCA = 0;					//PWMC0 off
	TCC0_CCB = 0;					//PWMC1 off
	TCC0_CCC = 0;					//PWMC2 off
	TCC0_CCD = 0;					//PWMC3 off
	TCC1_CCA = 0;					//PWMC4 off
	TCC1_CCB = 0;					//PWMC5 off
}

void ServoDInit(void)
{
	TCD0_CTRLA = 0x05;				//set TCC0_CLK to CLK/64
	TCD0_CTRLB = 0xF3;				//Enable OC A, B, C, and D.
Set to Single Slope PWM
							//OCnX = 1 from
Bottom to CCx and 0 from CCx to Top
	TCD0_PER = 10000;				//20ms / (1/(32MHz/64)) =
```

```
10000.  PER = Top
	TCD1_CTRLA = 0x05;			//set TCC1_CLK to CLK/64
	TCD1_CTRLB = 0x33;			//Enable OC A and B.  Set to
Single Slope PWM
						//OCnX = 1 from
Bottom to CCx and 0 from CCx to Top
	TCD1_PER = 10000;			//20ms / (1/(32MHz/64)) =
10000.  PER = Top
	PORTD_DIR = 0x3F;			//set PORTC5:0 to output
	TCD0_CCA = 0;				//PWMC0 off
	TCD0_CCB = 0;				//PWMC1 off
	TCD0_CCC = 0;				//PWMC2 off
	TCD0_CCD = 0;				//PWMC3 off
	TCD1_CCA = 0;				//PWMC4 off
	TCD1_CCB = 0;				//PWMC5 off
}

void ServoC0(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
	TCC0_CCA = (750 + value);		//Generate PWM.
}

void ServoC1(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
	TCC0_CCB = (750 + value);		//Generate PWM.
}

void ServoC2(int value)
```

```
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
	TCC0_CCC = (750 + value);	//Generate PWM.
}

void ServoC3(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
	TCC0_CCD = (750 + value);	//Generate PWM.
}

void ServoC4(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
	TCC1_CCA = (750 + value);	//Generate PWM.
}

void ServoC5(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			//  -100 => 1ms
	else if (value < -100)		//  0    => 1.5ms
		value = -100;			//  100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				//  new range +/- 250
```

```
	TCC1_CCB = (750 + value);		//Generate PWM.
}

void ServoD0(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			// -100 => 1ms
	else if (value < -100)		// 0    => 1.5ms
		value = -100;			// 100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				// new range +/- 250
	TCD0_CCA = (750 + value);		//Generate PWM.
}

void ServoD1(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			// -100 => 1ms
	else if (value < -100)		// 0    => 1.5ms
		value = -100;			// 100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				// new range +/- 250
	TCD0_CCB = (750 + value);		//Generate PWM.
}

void ServoD2(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			// -100 => 1ms
	else if (value < -100)		// 0    => 1.5ms
		value = -100;			// 100  => 2ms
	value *= 5;				//multiply value by 2.5
	value /= 2;				// new range +/- 250
	TCD0_CCC = (750 + value);		//Generate PWM.
}

void ServoD3(int value)
{
	if (value > 100)			//cap at +/- 100
		value = 100;			// -100 => 1ms
```

```
	else if (value < -100)			// 0	=> 1.5ms
		value = -100;				// 100	=> 2ms
	value *= 5;					//multiply value by 2.5
	value /= 2;					// new range +/- 250
	TCD0_CCD = (750 + value);		//Generate PWM.
}

void ServoD4(int value)
{
	if (value > 100)				//cap at +/- 100
		value = 100;				// -100 => 1ms
	else if (value < -100)			// 0	=> 1.5ms
		value = -100;				// 100	=> 2ms
	value *= 5;					//multiply value by 2.5
	value /= 2;					// new range +/- 250
	TCD1_CCA = (750 + value);		//Generate PWM.
}

void ServoD5(int value)
{
	if (value > 100)				//cap at +/- 100
		value = 100;				// -100 => 1ms
	else if (value < -100)			// 0	=> 1.5ms
		value = -100;				// 100	=> 2ms
	value *= 5;					//multiply value by 2.5
	value /= 2;					// new range +/- 250
	TCD1_CCB = (750 + value);		//Generate PWM.
}

/********
 * ADCA *
 ********/

void ADCAInit(void)
{
	ADCA_CTRLB = 0x00;				//12bit, right adjusted
	ADCA_REFCTRL = 0x20;			//set to Vref = Vcc/1.6 =
2.0V (approx)
	ADCA_CH0_CTRL = 0x01;			//set to single-ended
	ADCA_CH0_INTCTRL = 0x00;		//set flag at conversion complete.
```

```
Disable interrupt
	ADCA_CH0_MUXCTRL = 0x08;			//set to Channel 1
	ADCA_CTRLA |= 0x01;				//Enable ADCA
}
int ADCA0(void)
{
	ADCA_CH0_MUXCTRL = 0x00;			//Set to Pin 0
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}
int ADCA1(void)
{
	ADCA_CH0_MUXCTRL = 0x08;			//Set to Pin 1
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}
int ADCA2(void)
{
	ADCA_CH0_MUXCTRL = 0x10;			//Set to Pin 2
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}
int ADCA3(void)
{
	ADCA_CH0_MUXCTRL = 0x18;			//Set to Pin 3
```

```
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;				//return result
}
int ADCA4(void)
{
	ADCA_CH0_MUXCTRL = 0x20;			//Set to Pin 4
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;				//return result
}
int ADCA5(void)
{
	ADCA_CH0_MUXCTRL = 0x28;			//Set to Pin 5
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;				//return result
}
int ADCA6(void)
{
	ADCA_CH0_MUXCTRL = 0x30;			//Set to Pin 6
	ADCA_CTRLA |= 0x04;				//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;				//return result
```

```
}
int ADCA7(void)
{
	ADCA_CH0_MUXCTRL = 0x38;		//Set to Pin 7
	ADCA_CTRLA |= 0x04;			//Start Conversion on ADCA
Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);//wait for conversion
to complete
	delay_ms(10);
	int value = ADCA_CH0_RES;		//grab result
	return value;				//return result
}



#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

/*==================================================
The Following three functions are used to control
the Motor Driver 1A Dual TB6612FNG sku: ROB-09457
from SparkFun Electronics.

drivemotorsinit(void) - This function initializes the
IO Pins on Port J and readies them for use with
the motor driver.

drivemotorA(int mode, int speed) - This function takes an integer from
1-5 as the mode, and a number from 0-100 for the
speed as a percentage of max speed.

drivemotorB(int mode, int speed) - This function takes an integer from
1-5 as the mode, and a number from 0-100 for the
speed as a percentage of max speed.

Wiring Directions:
Left  Side of Chip
	Plug Vm into the pin labeled PWM D5(Vcc)
	Plug Vcc into one of the Power Bus's Vcc pins
```

Plug Gnd into a Power Bus's Gnd Go through and plug all the grounds on the
chip into pins on the power bus or wire them
all together and plug in one.
Plug A01 into a plug on motor A
Plug A02 into the other plug on motor A
Plug B01 into a plug on motor B
Plug B02 into the other plug on motor B
Right Side of Chip
Plug PWMA into Port F0(Signal)
Plug AIN2 into Port J pin 0
Plug AIN1 into Port J pin 1
Plug STBY into Port J pin 2
Plug BIN1 into Port J pin 3
Plug Bin2 into Port J pin 4
Plug PWMB into Port F1(Signal)

```
=================================================*/

void drivemotorsinit(void){
	PORTJ_DIR |= ((1<<0) | (1<<1) | (1<<2) | (1<<3) | (1<<4)); // Set pins 0-4 as output (1)
	PORTJ_OUT = ((1<<0) | (1<<1) | (1<<2) | (1<<3) | (1<<4)); // Set all pins to Output Low (0), this puts both motors in standby

	TCF0_CTRLA = 0x05;			//set TCC0_CLK to CLK/64
	TCF0_CTRLB = 0xF3;			//Enable OC A, B, C, and D. Set to Single Slope PW
	//OCnX = 1 from Bottom to CCx and 0 from CCx to Top
	TCF0_PER = 10000;			//20ms / (1/(32MHz/64)) = 10000.  PER = Top
	PORTF_DIR = 0x3F;			//set PORTF:0 to output
	TCF0_CCA = 0;				//PWMF0 off
	TCF0_CCB = 0;				//PWMF1 off

}

void drive(int direction, int rightspeed, int leftspeed){
```

```
		if (leftspeed > 100){
			leftspeed = 100;
			}

		if (leftspeed < 0){
			leftspeed = 0;
			}

		if (rightspeed > 100){
			rightspeed = 100;
			}

		if (rightspeed < 0){
			rightspeed = 0;
			}

		TCF0_CCA = (1000*rightspeed); // Set the Duty Cycle
		TCF0_CCB = (1000*leftspeed); // Set the Duty Cycle

		if (direction == 1){ // Forward
			PORTJ_OUT = ((1<<0) | (0<<1) | (1<<2) | (1<<3) | (0<<4) |
(1<<2));
			}
		if (direction == 2){ // Reverse
			PORTJ_OUT = ((0<<0) | (1<<1) | (1<<2) | (0<<3) | (1<<4) |
(1<<2));
			}
		if (direction == 3){ // Right
			PORTJ_OUT = ((1<<0) | (0<<1) | (1<<2) | (0<<3) | (1<<4) |
(1<<2));
			}
		if (direction == 4){ // Left
			PORTJ_OUT = ((0<<0) | (1<<1) | (1<<2) | (1<<3) | (0<<4) |
(1<<2));;
			}


}
```

```
void drivemotorA( int mode, int speed) {

        if (speed > 100){
            speed = 100;
            }

        if (speed < 0){
            speed = 0;
            }

        TCF0_CCA = (1000*speed); // Set the Duty Cycle

        if (mode == 1){ // Standby
            PORTJ_OUT = (0<<2);
            }
        if (mode == 2){ // Stop
            PORTJ_OUT = ((0<<0) | (0<<1) | (1<<2));
            }
        if (mode == 3){ // Short Brake
            PORTJ_OUT = ((1<<0) | (1<<1) | (1<<2));
            }
        if (mode == 4){ // Clockwise
            PORTJ_OUT = ((1<<0) | (0<<1) | (1<<2));
            }
        if (mode == 5){ // Counter Clockwise
            PORTJ_OUT = ((0<<0) | (1<<1) | (1<<2));
            }
}


void drivemotorB( int mode, int speed) {
        if (speed > 100){
        speed = 100;
        }

        if (speed < 0){
        speed = 0;
        }

        TCF0_CCB = (1000*speed); // set the Duty cycle
```

```
		if (mode == 1){ // Standby
			PORTJ_OUT = (0<<2);
			}
		if (mode == 2){ // Stop
			PORTJ_OUT = ((0<<3) | (0<<4) | (1<<2));
			}
		if (mode == 3){ // Short Stop
			PORTJ_OUT = ((1<<3) | (1<<4) | (1<<2));
			}
		if (mode == 4){ // Clockwise
			PORTJ_OUT = ((1<<3) | (0<<4) | (1<<2));
			}
		if (mode == 5){ // Counter Clockwise
			PORTJ_OUT = ((0<<3) | (1<<4) | (1<<2));
			}
}
```

Main Program

```
#include <avr/io.h>
#include "PVR.h"
#include "MotorFunctions.h"

#define FC (ADCA1() + ADCA1() + ADCA1()+ ADCA1() + ADCA1()) / 5
#define FR (ADCA2() + ADCA2() + ADCA2()+ ADCA2() + ADCA2()) / 5
#define FL (ADCA3() + ADCA3() + ADCA3()+ ADCA3() + ADCA3()) / 5
```

```
#define BC (ADCA4() + ADCA4() + ADCA4()+ ADCA4() + ADCA4()) / 5

int i;
int j;
int k;

int thresh;

thresh = 500;

void main(void)
{
	xmegaInit();						//setup XMega
	delayInit();					//setup delay functions
	ADCAInit();						//setup PORTA
analong readings
	lcdInit();						//setup LCD on
PORT K
	ServoDInit();						//setup PORTD
Servos
	ServoCInit();						//setup PORTC
Servos
	drivemotorsinit();					//setup motors
	PORTQ_DIR |= 0x01;					//set Q0 (LED) as
output
	PORTB_DIR |= 0x00;			//set B0 as input
	PORTH_DIR |= 0x01;			//set H0 as output



	while(1)
	{

		PORTH_OUT = 0x01;

		i = PORTB_IN;
	i = i & 0x01;

		j = PORTB_IN;
		j = j & 0x02;
```

```
k = PORTB_IN;
k = k & 0x04;

lcdInit();

if(i == 1)
{
            drive(1,0,0);
            delay_ms(300);
            lcdString("Entered Off");
            lcdGoto(1,0);
      lcdString("Limits  Area");
            drive(2,80,80);
            delay_ms(2000);
            drive(3,80,80);
            delay_ms(3*FC^2);
}
else
{
      if(FC < thresh)
      {
            lcdInit();
            drive (1,0,0);
            delay_ms(300);

            if (FR < thresh && FL < thresh)
            {
                  lcdString("Obstacle->Front");
            lcdGoto(1,0);
            lcdString("Side");
                  drive(2,FC/5,FC/5);
                  delay_ms(5*FC^2);
                  drive(3,FC/5,FC/5);
                  delay_ms(FC^2);
            }
            else if (FR < thresh)
            {
                  drive(4,FR/5,FR/5);
                  lcdString("Obstacle->Front");
```

```
			lcdGoto(1,0);
		lcdString("Right side");
			delay_ms(FR^2);
		}
		else if (FL < thresh)
		{
			drive(3,FL/5,FL/5);
			lcdString("Obstacle -> Front");
		lcdGoto(1,0);
	lcdString("Left side");
			delay_ms(FL^2);
		}
		else
		{
		drive(2,90,90);
		delay_ms(FC^2);
		drive(3,90,90);
		delay_ms(1500);
		}
	}
	else if (FR < thresh)
	{
		drive(1,80,50);
		delay_ms(FR^2);
	}
	else if (FL < thresh)
	{
		drive(1,50,80);
		delay_ms(FL^2);
	}
	else if (FR < thresh & FL < thresh)
	{
		drive(2,80,80);
		delay_ms(FR^2);
		drive(FR/7,80,80);
		delay_ms(FL^2);
	}
	else if (j == 0 | k == 0)
	{
		drive(1,0,0);
```

```
				delay_ms(300);

					if (j == 0)
					{
						lcdString("Left Side");
					lcdGoto(1,0);
					lcdString("Collision");
						drive(2,80,80);
						delay_ms(2000);
						drive(3,80,80);
						delay_ms(2000);
					}
					else
					{
						lcdString("Right side");
					lcdGoto(1,0);
					lcdString("Collision");
						drive(2,80,80);
						delay_ms(2000);
						drive(3,80,80);
						delay_ms(2000);
					}
			}
			else
			{
				drive(1, 100, 100);
				lcdString("Lawntonomous");
				lcdGoto(1,0);
				lcdString("Maximus");
				delay_ms(500);
			}
		}
	}
}
```