University of Florida

Department of Electrical and Computer Engineering

EEL 5666

Intelligent Machines Design Laboratory

Final Report

The Bullfighting Autonomous Robot

Robot Name: BULL

Student Name: Aminatu Oyebanjo

Instructors: Dr. A Arroyo and Dr. Schwartz

TAs: Thomas Vermeer and Mike Pridgen

Date: 04/20/2010

Website: https://sites.google.com/site/bullautonomousrobot/

Table of Contents

List of Figures

1 Abstract

BULL is an autonomous robot that is always looking for a red flag in order to attack the flag. BULL is going to accomplish this task with the help of a CMUcam. The CMUcam will be used to search for the red flag. As soon as BULL sees this red flag, a red LED on the autonomous robot will be turned on to show that a red flag has been spotted. The autonomous robot will then attack the red flag.

An ATxmega128A1 microcontroller will be the brain of this autonomous robot. The microcontroller will control all the components associated with BULL like motors, servos, LEDs, LCD, sensors e.t.c.

2 Introduction

Bullfighting is a popular game in Portugal, Spain, some cities in South France, and several Latin American Countries. The game involves matador (bull fighters), who execute different techniques to subdue the bulls. The main purpose of this autonomous robot is to perform part of the Spanish style bullfighting.

The part that will be performed by the robot will be the part where bulls are tested for charges by the banderilleros (flagmen) with the use of a pink cape called capote while the matador looks on. For the design of BULL a red flag will be used instead of pink cape.

In order to achieve the above task, the autonomous robot will need a CMUcam to detect a red flag. After the robot finds the red flag, BULL will try to find the center of the flag with the help of the CMUcam1 camera. After BULL finds the center of the red flag, BULL will charge towards the red flag.

Three infrared radiation sensors are used for obstacle avoidance. Two of the IR sensors are place in front of BULL. One IR sensor is placed on the right, and the other on the left. One of the IR sensors is placed between the eyes of the robot.

3 Integrated System

The components that will be used to achieve behavior of BULL are:

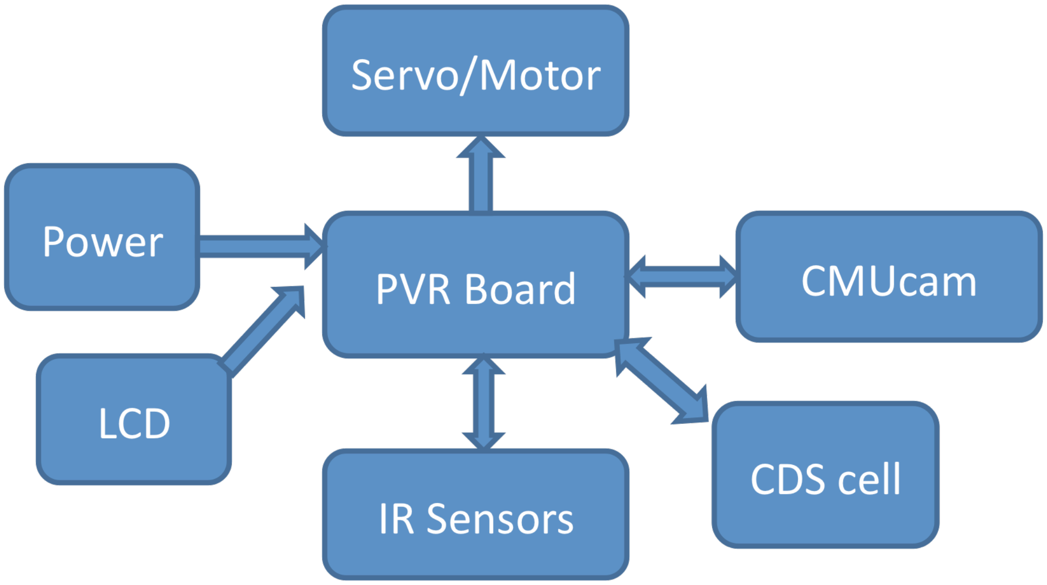

Figure 3-1 Block Diagram of Integrated System

PVR board: This board has the ATxmega128A1 microcontroller that will be used with the help of code written to it to control other components of the robot. This board has PWM port that produces the signal that is used to control the direction and speed of a motor or servo, ADC port, power I/O pins, other I/O pins, and other peripherals.

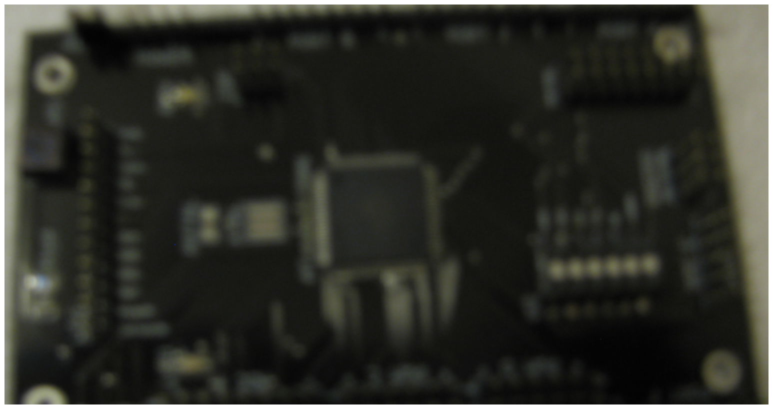

Figure 3-2 PVR Board

Infrared radiation sensors: Three of these sensors are used to detect obstacles. Two of the sensors are placed in the front while one is placed between the eyes of BULL.

Figure 3-3 Sharp GP2Y0A21YK IR Sensor

Wheels and Motor: The wheels and motor that will be used for this autonomous robot will be hacked from an already made RC car. The motor and wheels that comes with this RC car will be used to move BULL around. These wheels were chosen for speed.

LCD: This provides feedback of the activities of the robot.

Figure 3-4 16X2 LCD

Servo Motor: This is used to control the steering of the front wheels.

Figure 3-5 Full rotation servo

Chargeable Batteries: Six nickel-metal hydride is used to power the robot.

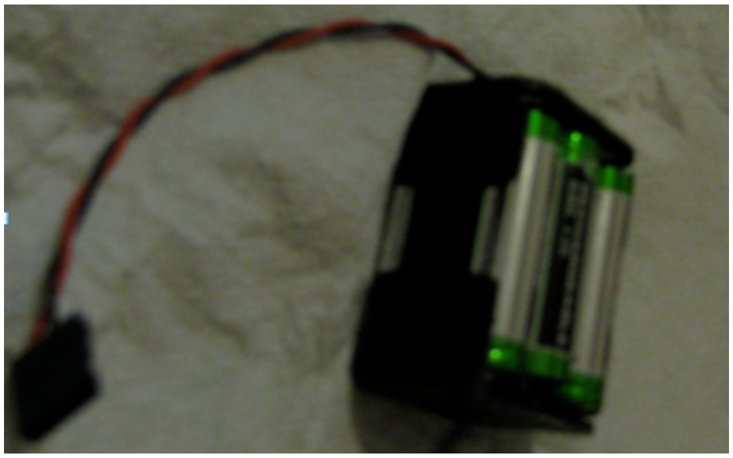

Figure 3-6 Picture of Six-pack NiMH batteries

CMUcam1: This camera is used to search for red flag.

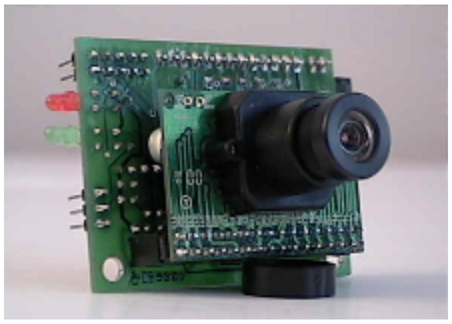

Figure 3-7 Picture of CMUcam1

4 Mobile Platform

BULL's platform was designed so that it will look like a bull. The Bull look-alike was placed on the box-like platform used to house the servo motor for the front wheel, and the motor for the back wheel. The CMUcam was placed in the front to search for the red flag. Two red LEDs used as the eyes of the robot. The red LEDs turn on when the robot sees red flag before it attacks. The movement of BULL is control by wheels instead of legs. All the components needed to make the robot work are inside the robot except the sensors. The two IR sensors are placed in front of the robot while one IR sensor is placed between the eyes of the robot for obstacle avoidance.

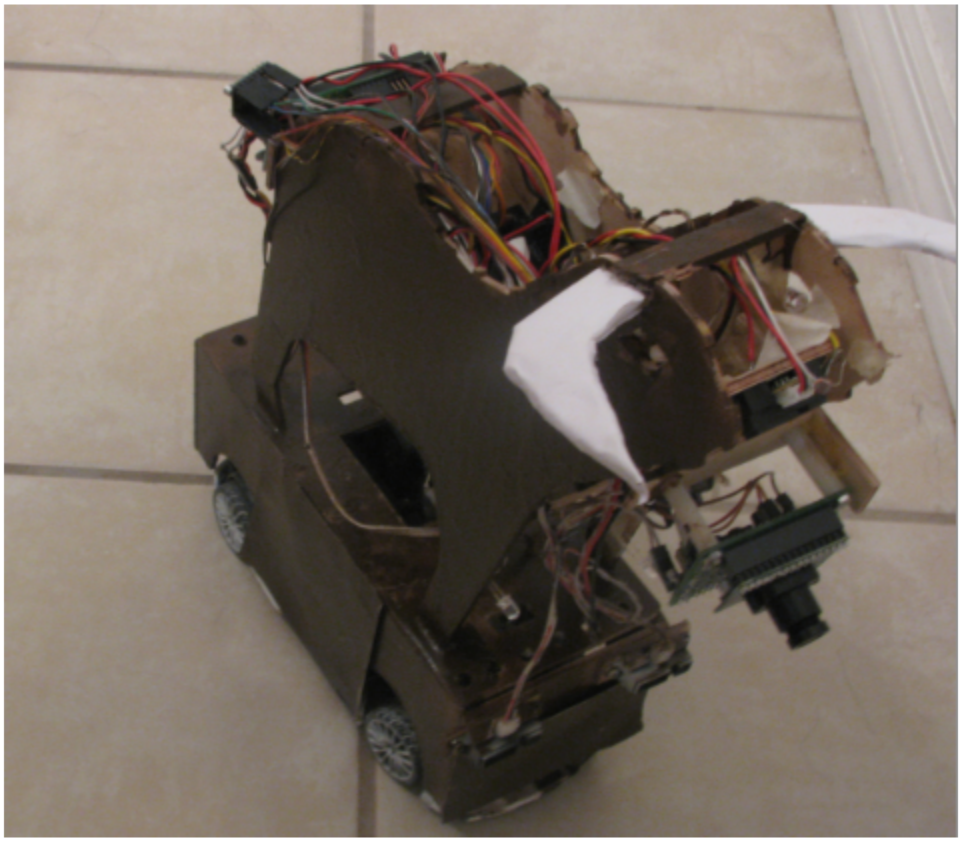
Figure 4-1 Platform

5 Sensors

In order for BULL to perform its tasks, BULL is going to use three different sensors. The sensors include: CMUcam, two infrared radiation sensors, and contact sensor.

5.1 CMUcam1

As previously explained, **CMUcam** will be used to detect the red flag and it will be placed in front of the robot. When CMUcam detects the red flag, it will let the microprocessor know how far off center it is. BULL will keep turning until its look at the center of the red flag. As soon as BULL is right in the front of the center of the red flag, it will charge forward.

5.2 Infrared Radiation

Three IR sensors are used in this design. One was place between the eyes of the robot. Two were placed in the front of the robot. The main function of the IR sensors is to avoid obstacle. When the IR sensor on the right sees an obstacle, it will turn left. When the IR sensor on the left sees an obstacle, it will turn right. If the IR sensor between the eyes of the robot sees an obstacle, the robot will reverse.

5.3 CDS cell

CDS cell also known as photoresistor is used by the robot while it is chasing the red flag. The resistance CDS cell is dependent on light. When the intensity of light reduces, the resistance of the CDS cell increases, verse versa. When the resistance increases, the voltage across the CDS cell would also increase.

The function of the CDS cell in the design of this robot is to let the robot know during the chasing of red flag period of the robot when it reaches the robot.

6 Behaviors

As previously discussed, BULL with will search for red flag. When it finds the red flag, BULL will charge towards it. At the same time, the robot will still avoid obstacles with the help of IR sensors.

Searching For Red Flag

When the power button on the autonomous robot is pressed, the motor in the rear wheel will move the robot at a slow speed while the servo will steer the robot left and right. At the same time, The CMUcam placed in from of the robot with keep searching for red color. As soon as red flag is spotted, the red LEDs that act as the eyes of the robot will turn on.

After Finding Red Flag

After the robot finds the red flag, BULL will center itself. After centering itself it will charge forward until the CDS cell does not see light anymore. After hitting the red flag, BULL will go back into obstacle avoidance mode and starts looking for another red flag.

7 Conclusion

The next time I design a robot, I will make sure not to use a motor that come with a toy car. It came to my attention that it is difficult to control the speed of the toy car motor. Also, the car would not move until it reaches 90% of its maximum speed, which is too fast. The reason I needed the wheels to be slow is so that the IR sensors would have enough time to observe their environment in order to avoid obstacle. I was told by the professors to start with the maximum speed for about a quarter of a second, and then reduce the speed.

Apart from the issue with controlling the speed, BULL worked fine. When it sees red it charges towards it. When it does not see red, it goes back to its obstacle avoidance mode.

References

"Basic 16x2 Character LCD - Black on Green 5V." Photo. Sparkfun. 24 March 2010. < http://www.sparkfun.com/commerce/product_info.php?products_id=255>.

"CMUcam." Photo. Seattle Robotics. 24 March 2010. < http://www.seattlerobotics.com /cmucam.htm>.

"IMDL – Guidelines for Written Reports." 24 March 2010. <http://mil.ufl.edu/5666/handouts /Fall09/written_reports.pdf>.

Kim, Seon. "CAT – Crawling Autonomous Robot." 31 March 2010. <http://sites.google.com /site/projectcrawlbot/>.

"Origin and History of Bullfighting." *Don Quijote*. Donquijote.org, n.p, n.d. Web. 27 January 2010. < http://www.donquijote.org/culture/spain/bullfight/>.

"Sharp GP2D120 IR Sensors." Photo. Terk. 24 March 2010. < http://www.terk.ri.cmu.edu /recipes/flower/7/ >.

Appendix A

Code: Bull.c

```
/*********************************************************************************************/
/*********************************Created by Aminatu Oyebanjo*********************************/
/********************************  With Help From Jason Mormoson*******************************/
/******************************* BULL.C [BULL AUTONOMUOS ROBOT]*****************************/


#include <avr/io.h>
#include "PVR.h"
#include "global.h"
#include "usart.h"

 void main(void)
{
        xmegaInit();                                            //setup XMega
        delayInit();                                            //setup delay functions

        ADCAInit();                                     //setup PORTA analong readings
        lcdInit();                                              //setup LCD on PORTK
        USARTInit();                                                    //Initialize the camera
        PORTQ_DIR |= 0x01;                                      //set Q0 (LED) as output
        PORTH_DIR |= 0x88;                                      //set red LED as output
        PORTF_DIR |= 0xE0;                                      //Motor Driver bits 7,6,5
        PORTB_DIR |= 0x03;                              // set white LED has output
        ServoCInit();                                   //Intialize PMW
        int LF=0;                                               //Left front IR sensor
        int RF=0;                                               //Right Front IR sensor
        int LB=0;                                               //Left back IR sensor
        int CENTRE=0;                                   //Center IR sensor
        int nl=90;                                              //Speed Control
        int n=90;                                               //Servo Direction Control
        int CDScell=0;                                          //CDS cells values
        //int del= 500;
        int i=500;
        int i2=100;
        int de=100;
        PORTB_OUT |= 0x03;                                      //turn on white LEDs

        lcdGoto(0,0);
        lcdString("BULL AUTONOMOUS");
        delay_ms(10);
        LF= ADCA7();
        RF= ADCA5();
        //LB =ADCA0();
        CENTRE = ADCA1();
        CDScell= ADCA3();
        lcdData(0x01);
        lcdGoto(0,0);
        lcdString("Calibrating");
         delay_ms(50);
        lcdData(0x01);
        lcdGoto(0,0);
        lcdString("LF");
        lcdInt(LF);
        lcdString("    RF");
        lcdInt(RF);
        lcdGoto(1,0);
        lcdString("CENTER");
        lcdInt(CENTRE);
        lcdGoto(0,0);
        delay_ms(5);
        lcdString("CDScell Calibration");
         delay_ms(5000);
        while(1)
        {
HERE:
        lcdData(0x01);
        TrackColor("TC 146 176 1 31 1 31\r");   //Tracking red color
         lcdGoto(0,0);
        lcdInt(buffer[2]);                          //reading MX
        lcdString("Tracking red fag...");
        delay_ms(2);

  if(buffer[2]<79 && buffer[2]>1)
                {
                  PORTH_OUT |=0x88;
                  lcdString("Found red flag");
                        lcdGoto(1, 0);
                        lcdString("Trying to center");
                        //lcdInt(buffer[2]);
                        delay_ms(10);
                        ServoC5(-30);               //Servo is centered
                        delay_ms(50);
                        Drive();
                        delay_ms(100);
                        if(buffer[2]>42 && buffer[2]<79)
                                {  lcdData(0x01);
                                        lcdGoto(0,0);
                                  lcdString("Red flag is on the Left");
                                        lcdGoto(1,0);
                                  lcdString("Turn left");
                                        LeftTurn(100);
                                        delay_ms(500);
                                        Drive();
                                        delay_ms(100);
                           }

                                else if(buffer[2]>0 && buffer[2]<18)
                                {
                                  lcdData(0x01);
                                        lcdGoto(0,0);
                                  lcdString("Red flag is on the Right");
                                        lcdGoto(1,0);
```

```
                              lcdString("Turn right");
                                    RightTurn(i2);
                                    delay_ms(500);
                                    Drive();
                                    delay_ms(100);
        }

                        else if(buffer[2]>18 && buffer[2]<42)
                          {  Stop();
                                    lcdData(0x01);
                                    lcdGoto(0,0);
                              lcdString("Attack!!!");
                                    ServoC1(-30);
                                    delay_ms(100);
                                    Run();
                                    Forward();
                                    ServoC1(100);
                                    delay_ms(100);

                              //    while(buffer[2]>30 && buffer[2]<46)
                                    if(ADCA3()<2000)
                              {     PORTH_OUT |=0x88;
                                          Run();
                                          Forward();
                                          ServoC1(100);
                                          ServoC1(100);
                                    //    delay_ms(1000);
                                lcdData(0x01);
                                          lcdGoto(0,0);
                                      lcdString("I am stuck");
                                          lcdInt(ADCA3());
                                    }
                              }
                  }
                  else
                  {
                    lcdData(0x01);
                        lcdGoto(0,0);
                        lcdString("Searching for red flag...");
                        PORTH_OUT &=0x00;
                        //Stop();
                        ServoC1(70);
                        delay_ms(500);
                        ServoC5(-30);
                        delay_ms(100);
                        Drive();

                  /*    LF= ADCA7();
                        RF= ADCA5();
                  CENTRE = ADCA1(); */
                       delay_ms(100);
                        lcdGoto(0,0);
                        lcdString("No more red, Now avoiding Obtacle");
                    lcdInt(LF);
                    lcdString(" RF:");
          lcdString(" CENTRE:");
                    lcdInt(ADCA1());
                    if(CENTRE>2000 && RF>1500)
     {
                              Stop();
                              delay_ms(20);
                              RightTurn(i2);
                              delay_ms(i);
                              Reverse();                                        //turn random
                              ServoC1(100);
                              delay_ms(1000);
                    }

                    else if(CENTRE>2000 && LF()>1200)
                    {
                              Stop();
                              delay_ms(20);
                              LeftTurn(i2);

                              delay_ms(4*i);
                              Reverse();                                        //turn random
                              ServoC1(100);
                              delay_ms(1000);
                              ServoC1(-30);
                              delay_ms(i2);
                    }

                    else if(LF> 1200)              // && LF <=1700)
                    {                                                           /* Slow down, then turn randon*/
                              Stop();
                              delay_ms(20);
                              RightTurn(i2);
                              delay_ms(4*i);
                              Drive();
                              delay_ms(i);
                              ServoC5(-30);          //Servo is center
                              delay_ms(i);
                              Drive();
                              delay_ms(de);
                    }

                    else if (RF>1700)
                    {
                              Stop();
                              delay_ms(20);
                              LeftTurn(i2);
                              delay_ms(i*4);
                              Drive();
                              delay_ms(i);
                              ServoC5(-30);
                              delay_ms(i);
```

```
                        Drive();
                        delay_ms(de);
                        goto HERE;
                    }

                    else if (RF>1700 && LF()>1200)
                    {
                        Stop();
                        delay_ms(20);
                        //RightTurn(i2);
                        ServoC5(-30);
                        delay_ms(i);
                        Reverse();                          //turn random
                        ServoC1(100);
                        delay_ms(1000);

                    }
                }
            }
}
```

```
/******************************************************************************************/
/****************************Created by Mike Pridgen & Thomas Vermeer **************************/
/****************************** Edited by Aminatu Oyebanjo*************************************/
/**************************************** PVR.C**********************************************/

#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.
}
/**Edited by Aminatu Oyebanjo to get frequency between 1KHz and 10KHz***/
void ServoC1(int value)
{
	if (value > 100)				//cap at +/- 100
		value = 100;
	else if (value < 0)
		value =1;

	TCC0_CCB = (100*value);			//Generate PWM of frequency iKHz to 10KHz
}

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 = 0x10;                    //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
        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
	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
	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
	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
	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
	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
```

```
        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
        int value = ADCA_CH0_RES;                   //grab result
        return value;                                       //return result
}


///new code added by Aminatu Oyebanjo

void Speed (int value)
{
        value *=5;
        value /=2;
        TCD1_CCB = (750+value);
}

void Run (void)
{
        PORTF_OUT |=0x20;    //set Port , standby high
}

void Standby(void)
{
        PORTF_OUT &=0xDF;
}

void Forward(void)
{
        PORTF_OUT &=0xBF;                           //IN1=0 port 6
        PORTF_OUT |=0x80;                           //IN2=1 port 7
}

void Reverse(void)
{
        PORTF_OUT |=0x40;                           //IN1=1
        PORTF_OUT &=0x7F;                           //IN2=0
}

void Brake(void)
{
        PORTF_OUT |=0x40;                           //IN1=1
        PORTF_OUT |=0x80;                           //IN1=0
}

void Stop(void)
{
```

```
            PORTF_OUT &=0xB0;           //IN1=0
   PORTF_OUT &=0x7F;           //IN2=0
}
void LeftTurn(int speed)    //left turn
            {
            ServoC5(speed);
            }
void RightTurn(int speed)    //right turn
            {
             ServoC5(-1*speed);
            }
void Drive(void)
{
Run();
Forward();
ServoC1(100);  //start fast
delay_ms(250);
ServoC1(60);  //slow down
}
```

```
/**************************************************************************************************/
/************************************Created by Seon Kim****************************************/
/********************************Edited by Aminatu Oyebanjo************************************/
/**************************** *********global.c*************************************************/
#include "global.h"

int buffer[BUFFER_LENGTH];
```

```
/**************************************************************************************************/
/************************************Created by Seon Kim****************************************/
/**************************** *********usart.c*************************************************/


#include <avr/io.h>
#include "usart.h"
#include "global.h"

void USARTInit(void)
{
   PORTE.DIRSET = PIN3_bm;                                      // PIN3 (TXD0) as output
   PORTE.DIRCLR = PIN2_bm;                                      // PC2 (RXD0) as input

            USARTE0_CTRLC = 0x03;                               // Set 8N1

   USART_Baudrate_Set(&USART, 2094, -7); //Set baud rate (2094,-7) for 115,200 bps, (3317,-4) for 9600 bps
            //USART_Baudrate_Set(&USARTE0, 51 , 0);

   USART_Rx_Enable(&USART);                            // Enable Rx
   USART_Tx_Enable(&USART);                            // Enable Tx
```

```
  SendCommand("RS\r");                                    // Reset CMUcam
        SendCommand("RM 3\r");                                   // Set to Raw Mode with no ACK/NCK
        SendCommand("PM 1\r");            // Set CMUcam to Poll Mode
        //SendCommand("L1 1\r");
}

void SendCommand(char *command){                          // Send command string and return reply string
        memset( buffer, 0, BUFFER_LENGTH );
        int i = 0;
        lcdData(0x01);
        do{
                while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));        // check if data register is empty

                USARTE0_DATA = command[i];                                              // store
data in tx_char
                while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));
  }while(command[i++]!='\r');
        i = 0;
        do{
                while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));        // wait for RXCIF to be set
                buffer[i] = USARTE0_DATA;
        }while(buffer[i++]!=':');                                       // Might wanna change this to !=
'\r' for continuous mode
}

void TrackColor(char *command){
  memset( buffer, 0, BUFFER_LENGTH );

  int i = 0;
  do{
                while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));    // check if data register is empty
    USARTE0_DATA = command[i];                    // store data in tx_char
    while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));
  }while(command[i++]!='\r');

  i = 0;
        //buffer[0] = USARTE0_DATA; //6
        for(i = 0; i < 11; i++){
                while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));        // wait for RXCIF to be set
                buffer[i] = USARTE0_DATA;
        }
}
```