University of Florida

# BRaD: Bomb Recovery and Disposal Robot

EEL 5666: Intelligent Machines Design Laboratory
1st Written Report
Spring 2010

Student Name: Jose R. Vento

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

TAs : Mike Pridgen
Thomas Vermeer

Date: 4/20/10

# I. Table of Contents

## Contents

# II. Abstract

Because of the increasing hazards troops face in present urban combat environments more autonomous systems are needed to decrease the exposure that soldiers face. The Bomb Recovery and Disposal robot is aimed at detecting, acquiring and disposing of hazardous objects in the form of colored balls. The vehicle is built on a tracked propulsion system for increased maneuverability and accessibility in varying terrain conditions. The robot will pick up the balls and place them in an onboard chute and eject them into a bin in a predetermined location. The system will use a camera to distinguish between the targets and a combination of sonar and IR to navigate around objects.

# III. Executive Summary

This project had as its target the creation of an autonomous robot that would retrieve an item by searching randomly and place it in a container without prior knowledge of their respective locations. BRaD uses CMU1 CMOS camera to detect a color in a specific range and track it continuously until it is able to retrieve it with the arm. The camera is interfaced via the MAVRIC IIB board which is effectively the brain of the system; controlling all inputs and outputs, the LCD screen, the motors, the servos, the sonars, and the IRs. The sonars and the IRs are polled constantly to determine whether there is an object in the proximity of the robot and try to avoid it by increasing distances. The IRs provide immediate object detection since they have a limited range of approximately 12 inches, the sonars on the other hand provide a much wider range of vision and allow for maneuvering in a greater spatial environment. The camera turned out was the main sensor used in this project, there were several hurdles mainly with how the camera behaved on changing environments with respect to light. To solve this problem a large external light was added and it was found that performance increased greatly with respect to the area where it was being operated, mainly the ability of the floor to reflect light.

# IV. Introduction

In the last hundred years humanity has seen conflicts in all types of environments and settings. From trench warfare in World War I and II, to guerilla warfare in Korea and Vietnam to the urban intensive environments of both Gulf wars. BRaD is a robot aimed to reducing the hazards of urban confrontations for the soldiers and shifting it to equipment with low operational overhead, i.e. an autonomous robot. The Bomb Recovery and Disposal robot is a comparatively low cost low maintenance high performing system designed to detect “hazardous devices,” store them and dispose of them in a safe manner; while eliminating the exposure of personnel.

Through this exposition we will see the different components that make up BRaD and the logic process the system applies to differentiate between a potentially harmful device and a device that presents no danger to friendly troops. BRaD is a tracked system that will roam any room that it is placed in and scout for balls of different colors, store them in its confines and eject them into a bin at a set location. The recovery system is tentatively done with a three axis arm

that picks the ball and places it into a chute to be ejected via a solenoid into the bin. We will also see the logic process for the behavior of the system and how it is achieved through the merging of analog sensors and digital processing.

# V. Integrated System

The Bomb Recovery and Disposal robot is built as a highly modular system, from an integration point of view the robot has several important components and some redundant systems for the more complicated functions that it needs to achieve, i.e. obstacle avoidance. Figure 1 shows an overall block diagram of how the system converges to achieve its tasks. For navigation in theory but mainly obstacle avoidance the robot will rely on sonars and IRs to detect objects in its cope of movement. These two sensors are purposely redundant, first because IR may not work on varying light conditions or in the case sonar, outside interference by overlaying signals; and second because the robot always needs to stay aware of its environment for survival purposes. In the case of the target detection and acquirement system the robot will rely on a camera to distinguish the different colored objects to decide whether to pick up the object and store it or let it be. For this specific robot, the camera is its largest limitation; it is the most cost intensive item and it limits us to simple objects like balls.

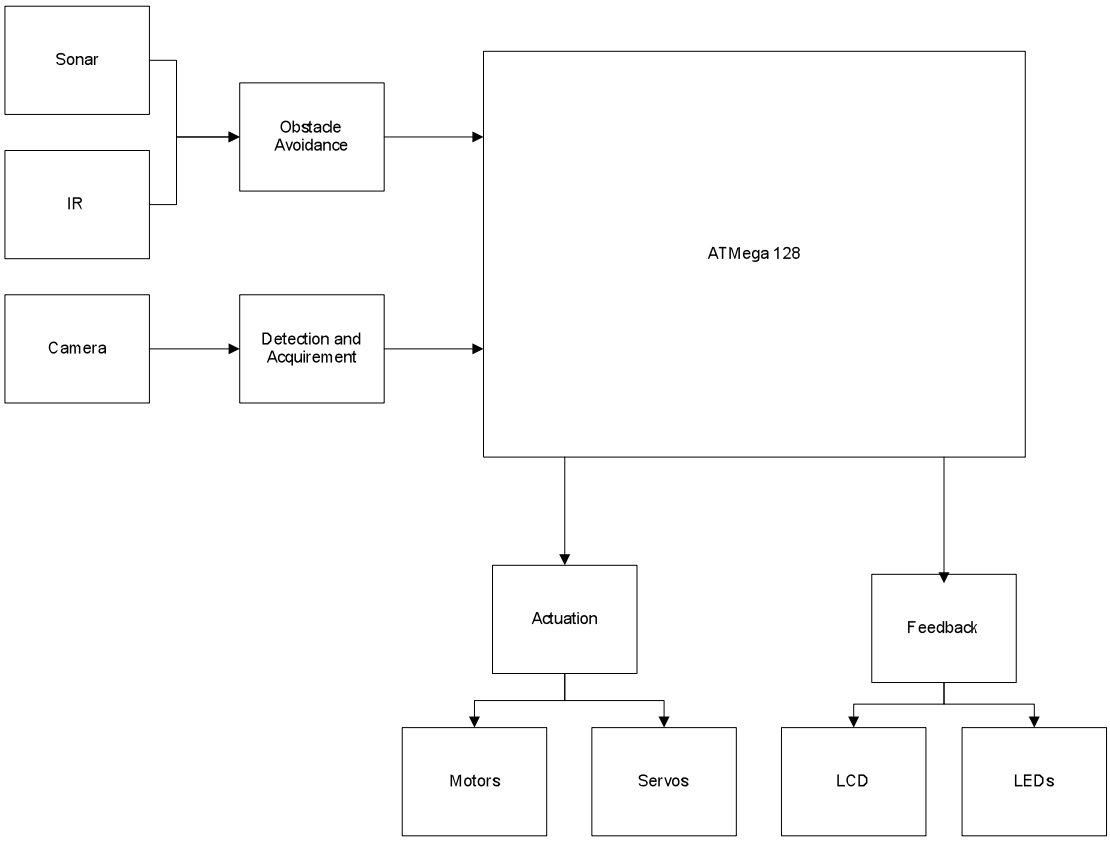


Figure 1: Hardware Integration

The system has two types of output, the feedback interface and the actuation. The system will be able to tell the users what its current function is and what the sensor values at that time, the LCD is the main source since it is much more interactive and more information can be displayed. We also use LED to provide certain type of specific information (i.e. is the robot in active mode or wait mode). Certainly the most important part of the robot is the actuation portion. This robot hast two propulsion motors and depending on how the arm is achieved from 3 to 4 servos.

Behind all the systems is an Atmel ATMega 128 microcontroller on a BDMICRO MavricII-B board. This development board offers all the range needed for this project. From dual UARTs 6 PWM channels for the two motors and the servos; 8 analog to digital converters and up to 53 input/output pins.

# VI. Mobile Platform

Table 1: Dimensions

| Width (in) | Length (in) | Height (in) |
|---|---|---|
| 13.25 | 11.00 | ~8.00 |

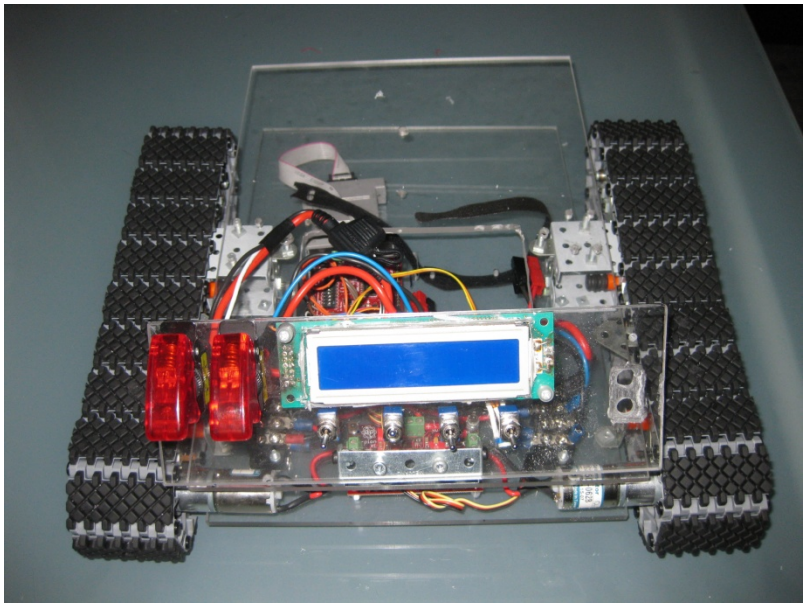

Figure 2: BRaD at construction phase.

The robot is built on a pre-manufactured polyurethane on plastic track system from Lynxmotion, with a modified Erector® set frame attached to a Plexiglas body. BRaD has a deck distribution, several decks allow for the separation of components by its placement priority. The

electronics and propulsion systems are set on the bottom deck to allow for the navigation and detection system to be placed on the more accessible portions.

# VII. Actuation

The dual track system calls for two motors to actuate each track and enable steering comparable to that of a tank. Depending on the direction of turn one of the tracks may be stopped or reversed. The PK22 (shown in Figure 3) motors run at 12 V with a nominal speed of 64 RPM. These motors were selected because they offer a high amount of torque at relatively decent speed, for this application the speed is just right but with respect to other packages offered this gearhead design is a tad slow.

Figure 3:PK22 Motor, 12 V 64 RPM

To control these motors, a Scorpion dual motor controller from robot power is used (Figure 4). This controller offers a wide range of input voltage and a continuous current rating of 12.5 Amps, much more that that needed for the motors which have a stall current of approximately 3.5 Amps. Although this might seem as overkill for this specific application, the modular nature of the robot might call for future expansion and outfitting of new parts and the remaining parts must be able to handle such growth. Providing power for this design are two Lithium Polymer 11.1 V 4000 mAh BlueLipo batteries (Figure 5); one is designated for the electronics and servos and the other is completely dedicated to propulsion. The dual battery design not only offers extended usable life of the system but protects the electronics from the residual spikes of the motors.


Figure 4: Scorpion Motor Controller


Figure 5: BlueLipo Batteries, LiPo 11.1 V 4000mAh

# VIII. Sensors

Due to the objection that the robot will encounter different environments, there was a need to have more than one ranging system. The system will optimally use 4 sonars and 5 IR placed strategically. Since the obstacle avoidance portion of this robot is not yet built, it is subject to change and will be discussed further. For detection the most likely candidate will be a CMU camera to detect bright colors. The placement of the ranging sensors is shown in figure 8.The placement for the IR's was a result mainly of wanting to create an immediate "perimeter" around the robot so it know when there are things in its immediate proximity. Sonars, however, are set for navigation; with the wider cone at long ranges we can create an all around closeness map to navigate around the objects before they are too close.

Figure 6: IR Sensors

Figure 7: SRF05

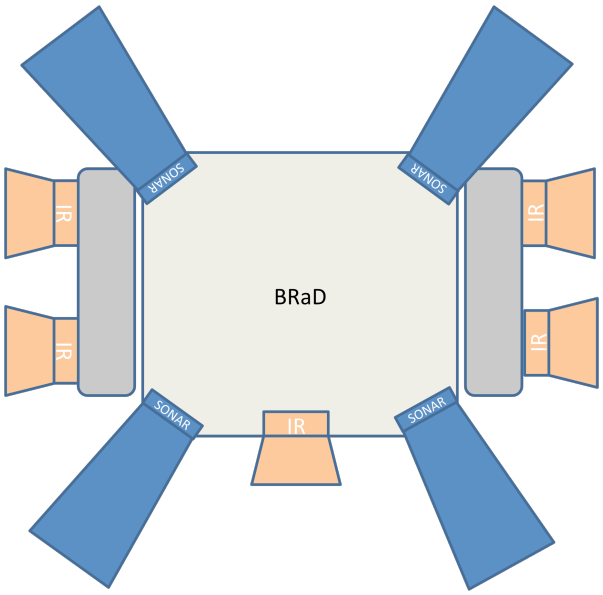


Figure 8: Sensor Placement.

The CMU1 Camera used in this project offers a flexibility to the project needed for the the different settings in which the robot would operate. The camera has the following specifications:

- Track user defined color blobs at 17 Frames Per Second
- Find the centroid of the blob
- Gather mean color and variance data
- Transfer a real-time binary bitmap of the tracked pixels in an image
- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image
- 80x143 Resolution
- 115,200 / 38,400 / 19,200 / 9600 baud serial communication
- Slave parallel image processing mode off a single camera bus
- Automatically detect a color and drive a servo to track an object upon startup
- Ability to control 1 servo or have 1 digital I/O pin

These capabilities allow us to assign different colors to each one of the targets (i.e. the placement bin, the targets, and the dummy targets).

# IX. Behaviors

The navigation algorithm will be an average of what the sonars see at medium to "long" range and what the IR see at short range. Since there will be two IR for each of the sides any discrepancy in their values will invalidate them for real use and the value used would be that of the sonar, and vice-versa.

# X. Experimental Layout and Results

Through testing we were able to determine the real values under different condition on which the IRs would work. This was critical in order to establish an operating zone for the robots sensors, this way we can know that even under environment changes the IRs will perform. The result from these experiments is shown in Table 2. Figure 7 shows a small exponential trend, this is the result of ambient noise, the closer the target is to the sensor the more accurate the data. When the target is too close the sensor saturates.

Table 2: IR Range Table

| Distance (in.) | Digital Read |
|---|---|
| 12 | 176-185 |
| 10 | 205-218 |
| 8 | 264-276 |
| 6 | 350-365 |
| 4 | 488-500 |
| 2 | 510-630 |
| 1 | 70-90 (Saturation) |

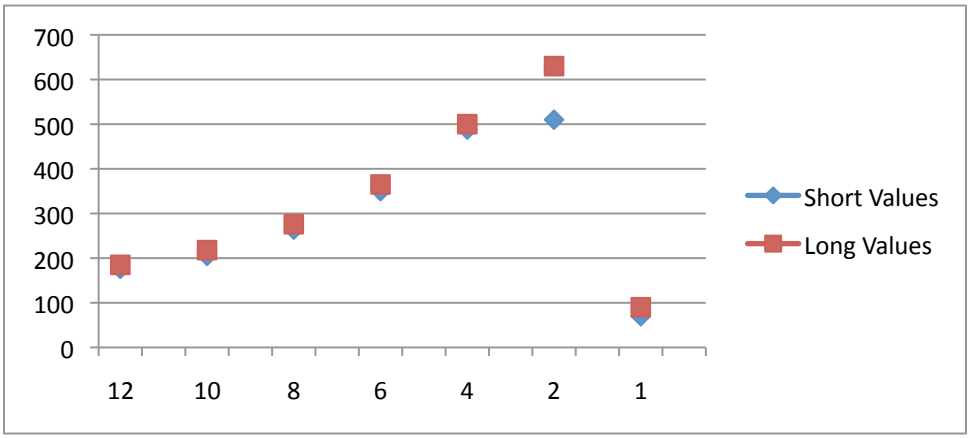


Figure 9: IR trend

# XI. Conclusion

Although at this stage of the project not much has been completed one can say that realistically that BRaD has yet to give its first steps. However, much progress has been done with respect to propulsion, since there has been a first, second, and third test runs the last being the only successful one since the others ended because of some form of failure.

The idea of having an autonomous machine running around with an explosive device might be a bit unnerving but with the right application of technology and proofs of concept like this robot it might just be possible.

# XII. Appendices

Here we can see all the portions of my code.

```
//////////MAIN CODE
#define F_CPU 16000000UL  // 16 MHz
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <avr/pgmspace.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>



#include "LCD.h"
#include "PWM.h"
#include "ir.h"
#include "inout.h"

//#include "USART.h"
//#include "CMU1.h"

//CMU FUNCTIONS
/*
void cmu_init(void);
void CMU_GM(void);
void CMU_TC(int ,int ,int ,int ,int ,int );
int binary2int(unsigned char);
```

```
//USART FUNCTIONS

void uart0_init(void);
void UART0_TX(char );
unsigned char UART0_RX(void);*/
#define SONAR_PORT PORTD
#define SONAR_DDRX DDRD
#define SONAR_PINX PIND


volatile uint16_t ms_count;
volatile int MAX_MSG_SIZE = 30;
volatile unsigned char CMUBuffer[15];
unsigned char CMUreturn[15];
char front[10];
unsigned int RED;
unsigned int GREEN;
unsigned int BLUE;
unsigned int Xpos;
unsigned int Ypos;
unsigned int conf;

int igotthetarget = 0;
int stopinthenameoflove = 0;
int complete = 0;
int begin = 0;
int done = 0;
int beginbox = 0;
int donebox = 0;
int rad = 0;
uint16_t rise = 180;
uint16_t risebox = 50;

int LF;   //1
int LR;
int RF;   //2
int RR;
int F;    //0

int sonarFL = 0; //2
int sonarFR = 0; //3
int sonarRL = 0; //0
int sonarRR = 0; //1

int hello = 0 ;
```

```
////////////////////////////////////////////////////////////////////
//////////////////////CMU CODE ////////////////////////////////////
//CODE WRITTEN BY LUIS VEGA AND MODIFIED TO SUIT THIS APPLICATION//////////
////////////////////////////////////////////////////////////////////
void uart0_init(void)
{
	UBRR0H = 0x00;
	UBRR0L = 16; //16 is for 115.2k if U2X=1, 0x33 = 51 for 38.4k baud
	UCSR0A = 0x02;
	UCSR0B = 0x18; //Bit 4 is Rx enable, Bit 3 is Tx enable
	UCSR0B |= (1<<RXEN)|(1<<TXEN);
	UCSR0C = 0x06; //Bit 6 = Mode (0=Ascync, 1=Sync),
}

/*Transmit a message over UART0 in the form of a character array*/
void UART0_TX(char message[MAX_MSG_SIZE])
{
	int t = 0;
	while ((t < (MAX_MSG_SIZE + 1)) & (message[t] != 0x00))
	{
		/*Wait for an empty transmit buffer*/
		while (!( UCSR0A & (1<<UDRE0)));

		UDR0 = message[t];
		t++;
	}
}

/*Receive a message*/
unsigned char UART0_RX(void)
{

	while(!(UCSR0A & (1<<RXC0)));
	return UDR0;
}
////////////////////////////////////
// FUNCTIONS FOR CMU/////////////////


/*CMUCam functions*/
void cmu_init(void)
{
	/*Reset*/
	UART0_TX("RS\r");
	_delay_ms(200);
	/*Poll Mode*/
```

```
        UART0_TX("PM 1\r");
        _delay_ms(200);
        /*Raw Output*/
        UART0_TX("RM 3\r");
        _delay_ms(200);
        /*Middle Mass On*/
        UART0_TX("MM 1\r");
        //_delay_ms(200);
   /*Track with the full window*/
        UART0_TX("SW 1 1 80 143 \r");
        //_delay_ms(200);


}

/*
** Get the mean (average) values of red, green, and blue (R, G, B)
** and store them in the global "CMUBuffer"
*/
void CMU_GM(void)
{
        //lcd_clear_screen();
        //lcd_out_string("1*");

        int i = 0;
        char tempChar;
        UART0_TX("GM\r");
        //lcd_out_string("2*");

        /*Read and discard the first 255 framing byte*/
        tempChar = UART0_RX(); /// 0058
        //tempChar = UART0_RX(); /// 0058

        //lcd_out_string("3*");
        /*Read 7 byte long "type S" packet*/
        for(i=0;i<7;i++)
        {
                CMUBuffer[i] = UART0_RX();
        }
        //lcd_out_string("4*");

        /*Trash the last 255 framing byte*/
        while(tempChar != ':')
        {
                tempChar = UART0_RX();
        }
```

```
	//lcd_out_string("5*");

	CMUBuffer[i] = '\0';
	//lcd_out_string("6*");

}

/*Tracks a color*/
void CMU_TC(int Rmin, int Rmax, int Gmin, int Gmax, int Bmin, int Bmax)
{
	int i = 0;
	char tempChar, tempMessage[30];
	sprintf(tempMessage,"TC %i %i %i %i %i %i\r", Rmin, Rmax, Gmin, Gmax, Bmin,
Bmax);
	UART0_TX(tempMessage);
	tempChar = UART0_RX();
	/*Return a 9 byte long "type M" packet*/
	for(i=0;i<9;i++)
	{
		CMUBuffer[i] = UART0_RX();
	}
	while(tempChar!= ':')
	{
		tempChar = UART0_RX();
	}
	CMUBuffer[i] = '\0';
}



void display_mean(void){
   alloff();

      CMU_GM(); // Dummy read
	red();

	_delay_ms(1000); // Wait for transients to die out (Very important)
	yellow();
	LCD_com(0x01);
	LCD_string("      RED:  GREEN:  BLUE:");

while(1){
   alloff();
	CMU_GM(); // Get means (15 min, 240 max)

	_delay_ms(20);
```

```
   RED = CMUBuffer[1]; // Red mean
        GREEN = CMUBuffer[2]; // Green mean
        BLUE = CMUBuffer[3]; // Blue mean

      blue();
        _delay_ms(10);
        LCD_com(0xC0);
        LCD_int(RED);
        LCD_string("   ");
        LCD_com(0xC7);
        LCD_int(GREEN);
        LCD_string("   ");
        LCD_com(0xCE);
        LCD_int(BLUE);
        LCD_string("   ");
        _delay_ms(10);

}
}

//////////////////////////SONAR
///
int sonar(int sonar_number)

        {
                int done = 0;
                int counter=0;
                int distance=0;
           // 0 RL
                                // 1 RR


                 unsigned int sonar_unsigned=0b00000001;
           ///Shifts to the desired pin///
                sonar_unsigned = sonar_unsigned<<(sonar_number);
                ///Set pin to output///
                DDRD |= sonar_unsigned;
               ///Set pin high, this is the trigger///
                                        PORTD &= ~sonar_unsigned;
                                        _delay_us(10);
                PORTD |= sonar_unsigned;
             ///Wait for atleast 10 microseconds///
                 _delay_us(15);
          ///Set pin low///
                PORTD &= ~sonar_unsigned;
```

```
            ///Set the pin to input///
                  DDRD &= 0x00;
          ///Wait for the high signal from the sonar output///
                   while((PIND & sonar_unsigned)==0){}
               ///Wait for the end of the high signal///
                  while((counter>=0) & (done == 0))
                  {
                           ///increment counter///
                            counter++;
                            ///check for end of high signal and break if low///
                            if((PIND & sonar_unsigned)==0){ done = 1;}
                   }

distance= counter/74 ;



 return distance;
}
//////////////////SONARRRRR


void recogete(void){
motors(225,225);
_delay_ms(2000);
motors(0,0);
_delay_ms(1000);
servo(-175,rise,-400,-30);
_delay_ms(1000);
servo(-175,0,-400,-30);
LCD_com(0x01);
LCD_string("      I GOT THE TARGET");
_delay_ms(20);
igotthetarget = 1;
}
void sueltame(void){
LCD_com(0x01);
LCD_string("      PUT IT IN");
_delay_ms(20);

F = IR_value(1, 5);
_delay_ms(30);

servo(-175,risebox,-100,-50);
_delay_ms(2000);
motors(0,0);
```

```
motors(220,-220);
_delay_ms(250);;

while(F <= 400){
motors(200,200);
F = IR_value(0, 5);

}

motors(0,0);
servo(-175,risebox,-400,-50);
_delay_ms(2000);
servo(-175,risebox,-400,-400);
_delay_ms(2000);
servo(-175,0,-100,-400);
motors(-225,-225);
_delay_ms(1000);
servo(-175,0,-400,-400);
motors(-225,-225);
_delay_ms(500);
LCD_com(0x01);
LCD_string("      DROPPED TARGET");
_delay_ms(20);
complete  = 1;
}


void track(void){
int doit = 0;
while((rise<= 450) & (done == 0)){


        if(begin == 0){rise=0;
                                        begin = 1;}
servo(-175,rise,-400,-350);

memset(CMUBuffer, 0, 15);

CMU_TC(210, 240, 30, 120, 13, 17);//LAB
//CMU_TC(210, 240, 210, 240, 10, 30); //HOUSE

motors(0, 0);

Xpos= CMUBuffer[1]; // Red mean
Ypos = CMUBuffer[2]; // Green mean
conf = CMUBuffer[8]; // Green mean
```

```
        if ((Xpos > 50) & (conf >40)){
                        motors(-225, 225);
                        //LCD_com(0x01);
                        //LCD_string("CENT: TURN LEFT");
                }
        else if ((Xpos< 30)& (conf >40)){
                        motors(225, -225);
                //      LCD_com(0x01);
                //      LCD_string("CENT: TURN RIGHT");
                }
        else if (((Xpos >= 30) | (Xpos <= 50)) & (conf > 40)){
                        if(Ypos <40){
                        motors(225,225);
                //      LCD_com(0x01);
                //      LCD_string("CENTERED!!! MOVING FORWARD");
                        }
                        else{
                        motors(0,0);
                        red();
                                if(rise <= 450){
                                yellow_only();
                                rise = rise + 50;
                                                                }
           else{done = 1;}

                        }
                }
  else{ doit =1 ;
                        LCD_com(0x01);
                        LCD_string("     CRAZY :-)");
                }



}
if(doit == 1){
recogete();}

}

void track_box(void){
int doitbox = 0;
while((risebox<= 145) & (donebox == 0)){
igotthetarget = 0;
```

```
        if(beginbox == 0){risebox = 0;
                                                beginbox = 1;}
servo(-175,risebox,-400,-50);

memset(CMUBuffer, 0, 15);
CMU_TC(30, 75, 210, 240, 90, 150);
//CMU_TC(220, 240, 220, 240, 220, 240);//LAB
//CMU_TC(210, 240, 210, 240, 10, 30); //HOUSE

motors(0, 0);

Xpos= CMUBuffer[1]; // Red mean
Ypos = CMUBuffer[2]; // Green mean
conf = CMUBuffer[8]; // Green mean


        if ((Xpos > 50) & (conf >40)){
                        motors(-225, 225);
                        LCD_com(0x01);
                        LCD_string("CENT: TURN LEFT");
                }
        else if ((Xpos< 30)& (conf >40)){
                        motors(225, -225);
                        LCD_com(0x01);
                        LCD_string("CENT: TURN RIGHT");
                }
        else if (((Xpos >= 30) | (Xpos <= 50)) & (conf > 40)){
                        if(Ypos <50){
        motors(225,225);
                LCD_com(0x01);
                        LCD_string("CENTERED!!! MOVING FORWARD");
                        }
                        else{
                        motors(0,0);
                        red();
                                if(risebox <= 150){
                                yellow_only();
                                risebox = risebox + 45;
                                                        }
           else{donebox = 1;}

                        }
                }
  else{ doitbox =1 ;
                        LCD_com(0x01);
```

```
                        LCD_string("      STOPPED ON THE BOX");
                }


}
if(doitbox == 1){
sueltame();
done = 0;
donebox = 0;
igotthetarget = 0;}

}



//////////////////////////////////////////////////////////
//////////////////////////END CMU/////////////////////////////
//////////////////////////////////////////////////////////

//INTERRRUPTTTTTTOOOO
/*
void pardonme(void){


        TCCR0 |= (1<<WGM01) |(1<<CS02) | (1<<CS01) | (1<<CS00);
   OCR0 = 16;
        TIMSK |= (1<<OCIE0);
   sei();
}*/




int main(void)
   {
// INITIALIZATIONS
  servo_init();
  init_LCD();
  adc_init();/* initialize A/D Converter */
  uart0_init();
  cmu_init();
// END INITIALIZATIONS

//TURN ON BOARD LED
```

```
  DDRB |= 0x01;   //Enable PORTB0 as output
  PORTB ^= 1; //toggle LED
//END BOARD LED

//PANNEL LED TEST ROUTINE
  red_only();
  _delay_ms(250);
  blue_only();
  _delay_ms(250);
  white_only();
  _delay_ms(250);
  yellow_only();
  _delay_ms(250);
//END TEST ROUTINE

//CAMERA LED TEST
  UART0_TX("L1 1\r"); // CAMERA LED TESTING for FUNCTIONALITY
  _delay_ms(1000);
  UART0_TX("L1 2\r"); // RETURN LED TO DEFAULT(TRACKING)
  _delay_ms(1000);
//END CAMERA TEST
  servo(-175,0,-400,-400);

//// BEGIN CODE////

CMU_TC(210, 240, 210, 240, 10, 30);//HOUSW
_delay_ms(10);


while(1){

memset(CMUBuffer, 0, 15);

CMU_TC(210, 240, 30, 120, 13, 17);//LAB
conf = CMUBuffer[8];
_delay_ms(10);
F = IR_value(0, 5);
LF = IR_value(1, 5);
RF = IR_value(2, 5);

                    if(conf<40){

                                                //if(F>=300){
                                                                /*
                                                                motors(-225,-225);
                                                                }
```

```
                                        else if(RF>=300){
                                        motors(-225,225);
                                        }
                                        else if(LF>=300){
                                        motors(225,-225);
                                        }
                                        else{*/
                                        motors(-225,225); // go left
                                        //}

                                }
else{
                                alloff();
                                CMU_TC(200, 240, 90, 100, 11, 18);//TRACK THE BALL
                                _delay_ms(20);
                                red();
                                yellow();

                                igotthetarget = 0;
                                complete = 0;
                                begin = 0;
                                done = 0;
                                beginbox = 0;
                                donebox = 0;
                                rad = 0;
                                rise = 0;
                                risebox = 0;



                                track();
                        while(igotthetarget == 1){
                                        F = IR_value(0, 5);
                                        LF = IR_value(1, 5);
                                        RF = IR_value(2, 5);
                                CMU_TC(30, 75, 210, 240, 90, 150);//LAB
                                conf = CMUBuffer[8]; // Green mean
                                _delay_ms(10);
                                if(conf<40){
                                        /*
                                        if(F>=300){
                                        motors(-225,-225);
                                        }
                                        else if(RF>=300){
                                        motors(-225,225);
                                        }
```

```
                                        else if(LF>=300){
                                        motors(225,-225);
                                        }
                                        else{*/
                                        motors(225,-225); // go left
                                        //}

                                }
                                else{track_box();}

                        }

                        motors(-225,225);
                        _delay_ms(1000);

                        } // end big else



 }//end while


}
/*

ISR(TIMER0_COMP_vect){

CMU_TC(210, 240, 30, 120, 13, 17);//LAB
conf = CMUBuffer[8]; // Green mean
_delay_ms(10);
if(conf>40){
 LCD_com(0x01);
 LCD_string("       I GOT IT ");
 stopinthenameoflove = 1;
}
else{
LCD_com(0x01);
 LCD_string("       I DONT GOT IT ");
}

          sonarFL; //2
//  sonarFR; //3
//  sonarRL; //0
//  sonarRR; //1
```

```
//sonarFL = sonar(2);
//sonarFR = sonar(3);
//sonarRL = sonar(0);
//sonarRR = sonar(1);

LCD_com(0x01);
LCD_string("     FL   FR   RL   RR");

LCD_com(0xC0);
LCD_int(sonarFL);
LCD_string(" ");
LCD_int(sonarFR);
LCD_string(" ");
LCD_int(sonarRL);
LCD_string(" ");
LCD_int(sonarRR);
LCD_string(" ");

}*/

//////////LCD CODE

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>


void LCD_com(int command)
{

                                                //7    6    5    4   3  2  1   0
                                                //DB7  DB6  DB5  DB4   E  RW  RS


unsigned int UpperNibble = 0xF0 & command;
unsigned int LowerNibble = 0x0F & command;
LowerNibble = LowerNibble << 4;
LowerNibble = LowerNibble & 0xF0;
```

```
PORTC = 0x04 | UpperNibble;
_delay_ms(1);
PORTC = 0x00 | UpperNibble;
_delay_ms(1);

PORTC = 0x04 | LowerNibble;
_delay_ms(1);
PORTC = 0x00 | LowerNibble;
_delay_ms(1);
}



void LCD_char(int character)
{

unsigned int UpperNibble = 0xF0 & character;
unsigned int LowerNibble = 0x0F & character;
LowerNibble = LowerNibble << 4;
LowerNibble = LowerNibble & 0xF0;

PORTC = 0x05 | UpperNibble;
_delay_ms(1);
PORTC = 0x01 | UpperNibble;
_delay_ms(1);

PORTC = 0x05 | LowerNibble;
_delay_ms(1);
PORTC = 0x01 | LowerNibble;
_delay_ms(1);



}

void LCD_string(char stringy[])
{
        int i = 0;
        while (stringy[i] != '\0')
        {
                LCD_char(stringy[i]);
                i++;
        }

}
```

```
void init_LCD(void)
{

DDRC = 0xFF;          // LCD Output

_delay_ms(15);
LCD_com(0x33);
_delay_ms(1);
LCD_com(0x32);
LCD_com(0x2C);
LCD_com(0x0F);
LCD_com(0x01);


}

void LCD_int(uint16_t integer)
{
        /*
        ** Break down the original number into the thousands, hundreds,
        tens,
        ** and ones places and then immediately write that value to the
        LCD
        */
        uint8_t thousands = integer / 1000;
        LCD_char(thousands + 0x30); // 0x30 = zero in hexidecimal          format = 0b00110000
(in binary format)
        uint8_t hundreds = (integer - thousands*1000) / 100;
        LCD_char(hundreds + 0x30);
        uint8_t tens = (integer - thousands*1000 - hundreds*100 ) / 10;
        LCD_char(tens + 0x30);
        uint8_t ones = (integer - thousands*1000 - hundreds*100 -  tens*10);
        LCD_char(ones + 0x30);
}

///////////////IR CODE
/*
 * ATmega128 A/D Converter utility routines
 */

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


/*
 * adc_init() - initialize A/D converter
```

```
 *
 * Initialize A/D converter to free running, start conversion, use
 * internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming
 * 16 MHz MCU clock)
 */
void adc_init(void)
{
  /* configure ADC port (PORTF) as input */
  DDRF  = 0x00;
  PORTF = 0x00;

  ADMUX = _BV(REFS0);
  ADCSR = _BV(ADEN)|_BV(ADSC)|_BV(ADFR) |
_BV(ADPS2)|_BV(ADPS1)|_BV(ADPS0);
}


/*
 * adc_chsel() - A/D Channel Select
 *
 * Select the specified A/D channel for the next conversion
 */
void adc_chsel(uint8_t channel)
{
  /* select channel */
  ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}


/*
 * adc_wait() - A/D Wait for conversion
 *
 * Wait for conversion complete.
 */
void adc_wait(void)
{
  /* wait for last conversion to complete */
  while ((ADCSR & _BV(ADIF)) == 0)
    ;
}


/*
 * adc_start() - A/D start conversion
 *
 * Start an A/D conversion on the selected channel
```

```
 */
void adc_start(void)
{
  /* clear conversion, start another conversion */
  ADCSR |= _BV(ADIF);
}


/*
 * adc_read() - A/D Converter - read channel
 *
 * Read the currently selected A/D Converter channel.
 */
uint16_t adc_read(void)
{
  return ADC;
}


/*
 * adc_readn() - A/D Converter, read multiple times and average
 *
 * Read the specified A/D channel 'n' times and return the average of
 * the samples
 */
uint16_t adc_readn(uint8_t channel, uint8_t n)
{
  uint16_t t;
  uint8_t i;

  adc_chsel(channel);
  adc_start();
  adc_wait();

  adc_start();

  /* sample selected channel n times, take the average */
  t = 0;
  for (i=0; i<n; i++) {
    adc_wait();
    t += adc_read();
    adc_start();
  }

  /* return the average of n samples */
  return t / n;
```

```
}


uint16_t IR_value(uint8_t ir_select , uint8_t sample_rate)
{
//      DDRD |= 0x0F;
        /* CHANNEL NAMES                                   ADDRESS on PORT D
        2       RF -- RIGHT SIDE FRONT IR     0x02
        3       RR -- RIGHT SIDE REAR IR          0x03

        4       F  -- FRONT  IR                   0x04

        0       LF -- LEFT SIDE FRONT IR  0x00
        1       LR -- LEFT SIDE REAR IR       0x01

                */

  if(ir_select == 0 ){
        //      PORTD |= 0x00;
                return adc_readn(0, sample_rate); // the first value is always 0
                                                  // because it is PIN 0 on PORT F
                                                  // the second value is the number of
                                                  // samples passed to the function
                }
  else if(ir_select == 1 ){
        //      PORTD |= 0x01;
                return adc_readn(1, sample_rate); // the first value is always 0
                                                  // because it is PIN 0 on PORT F
                                                  // the second value is the number of
                                                  // samples passed to the function
                }
  else if(ir_select == 2 ){
        //      PORTD |= 0x02;
                return adc_readn(2, sample_rate); // the first value is always 0
                                                  // because it is PIN 0 on PORT F
                                                  // the second value is the number of
                                                  // samples passed to the function
                }
 /* else if(ir_select == 3 ){
        //      PORTD |= 0x03;
                return adc_readn(0, sample_rate); // the first value is always 0
                                                  // because it is PIN 0 on PORT F
                                                  // the second value is the number of
                                                  // samples passed to the function
                }
  else if(ir_select == 4 ){
```

```
	//	PORTD |= 0x05;
		return adc_readn(0, sample_rate); // the first value is always 0
                                                  // because it is PIN 0 on PORT F
                                                  // the second value is the number of
                                                  // samples passed to the function
		}*/

}

//////////////////////INPUT OUTPUT MANIPULATION
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

//Enable PORTB0 as output
                                  // bit 0 RED write 0x01
                                  // bit 1 WHITE write 0x02
                                  // bit 2 BLUE write 0x04
                                  // bit 3 Yellow write 0x08


void red_only(void){
	DDRA |= 0x0F;   //Enable PORTA as output
	PORTA &= 0x00;
	PORTA |= 0x01;
		}
void white_only(void){
	DDRA |= 0x0F;   //Enable PORTA as output
	PORTA &= 0x00;
	PORTA |= 0x02;
		}
void blue_only(void){
	DDRA |= 0x0F;   //Enable PORTA as output
	PORTA &= 0x00;
	PORTA |= 0x04;
		}
void yellow_only(void){
	DDRA |= 0x0F;   //Enable PORTA as output
	PORTA &= 0x00;
	PORTA |= 0x08;
```

```
		}
void red(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		//PORTA &= 0x00;
		PORTA |= 0x01;
			}
void white(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		//PORTA &= 0x00;
		PORTA |= 0x02;
			}
void blue(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		//PORTA &= 0x00;
		PORTA |= 0x04;
			}
void yellow(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		// PORTA &= 0x00;
		PORTA |= 0x08;
			}
void allcolors(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		PORTA &= 0x00;
		PORTA |= 0x0F;
			}
void alloff(void){
		DDRA |= 0x0F;   //Enable PORTA as output
		PORTA &= 0x00;
		PORTA |= 0x00;
			}


//POINTLESSSS CODEEEEEEEE

 /*
  // LIGHT SHOW ---------
  DDRA |= 0x0F;   //Enable PORTB0 as output
  PORTA |= 0x08;  // bit 0 RED write 0x01
                                  // bit 1 WHITE write 0x02
                                  // bit 2 BLUE write 0x04
                                  // bit 3 Yellow write 0x08
  _delay_ms(1000);
  PORTA |= 0x04;
  _delay_ms(1000);
  PORTA |= 0x02;
```

```
  _delay_ms(1000);
  PORTA |= 0x01;



  _delay_ms(500);
  PORTA ^= 0x0f;
  _delay_ms(500);
  PORTA ^= 0x0f;
  _delay_ms(500);
  PORTA ^= 0x0f;


  _delay_ms(1000);
  PORTA |= 0x01;
  _delay_ms(1000);
  PORTA |= 0x02;
  _delay_ms(1000);
  PORTA |= 0x04;
  _delay_ms(1000);
  PORTA |= 0x08;
*/
```