REU & EEL 5666 IMDL

Summer 2005

Final Report

Instructor: Dr. Arroyo, Dr. Schwartz

Student: Quentin Lindsey

Date:8/4/05

Table of Content

Introduction

Fire fighting is inherently a dangerous occupation. If the dangerous portion of firefighting is taken out of human hands then this advancement would be great improvement over present standards. Presently there are robots used in military and police applications that can be controlled remotely to deactivate or detonate explosives. If this could be applied to fire-fighting applications, the lower causality level of bomb squads could be enjoyed by fire-fighters.

This project will seek to create autonomous agent capable of finding and extinguishing a fire on multi-level buildings as well as on the floor. Using intelligent aiming it will use a safer controlled method of extinguishing fires instead of the all encompassing methods found in other examples of autonomous firefighting.

This paper will focus on the key mechanical, electrical and programming components of the agent.

Integrated System

The integrated system will incorporate the following key components into the agent:

- Line Follower – Four OPB745 infrared photodiode and phototransistor pair are used to follow the lines which simulate the other grid of a city.
- Forward sensor array- The array consist of a GPD120 proximity detector and Eltec pyroelectric sensor mounted on a 180 deg range servo.

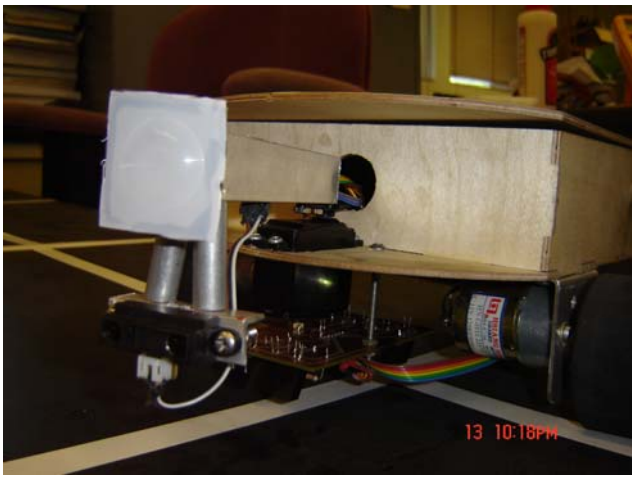


Figure 1 Forward sensor array

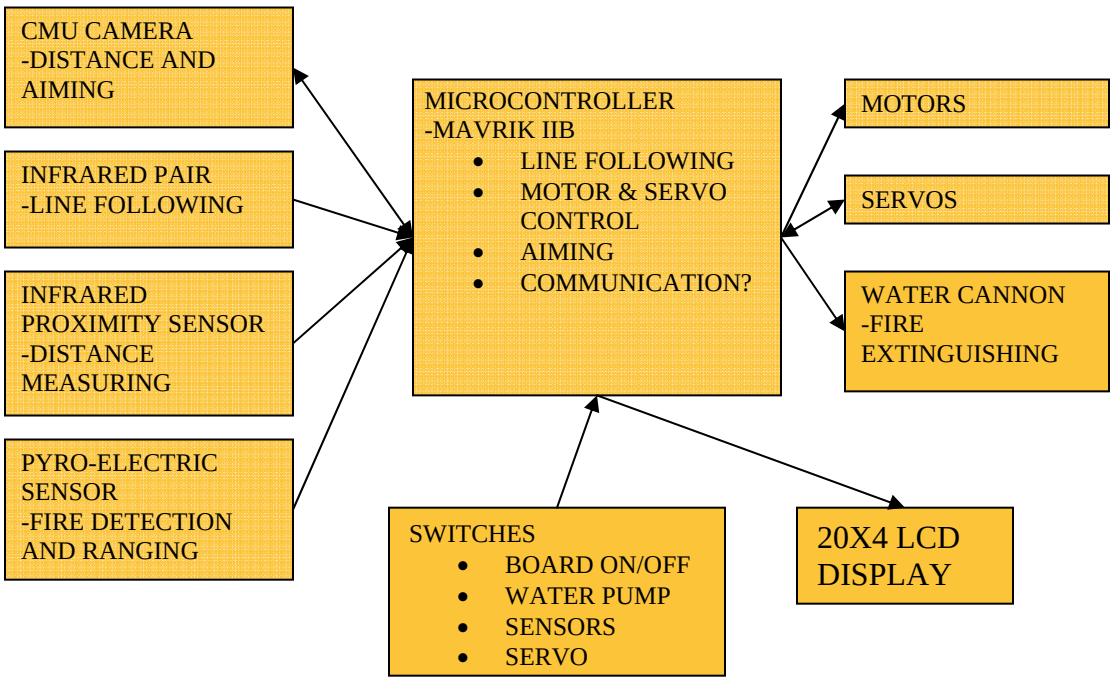
- CMU Camera with Dual Axle Freedom- This camera is used for precisely finding the flame and aiming the water cannon. It is mounted on two servos for pan/tilt capability. It is the data processing is done on board because of the simplicity of the vision algorithm.
- Water Cannon with Dual Axle Freedom-The water cannon is constructed of a 12V water pump which is mounted in the upper platform along with a small water tank. The cannon is mounted on

two servos also allowing it to have pan/tilt capability. On this same pan/tilt servo assembly a GPD120 is mounted to detect the distance of the water cannon from a possible fire.

- Input and Output-A four on/off switches and 20X4 LCD display with LED backlight enabled for output. The LCD display is used in 4-bit mode to conserve pins since system that requires speeds only achieved in 8 bit mode would be unnecessary or even perhaps destructive.

STRUCTURE OF THE INTEGRATED SYSTEMS

Error!



Mobile Platform

The mobility platform is constructed of two components: a mobility (bottom) platform and actuation (upper) platform. There are also multiple side sections supporting each platform. The mobility platform will house the motors and power supply as well as most of the PCB boards including the Mavrik IIB, power board and dual driver board. The bottom section houses the electronics because the upper platform has a higher chance of water damage.

The actuation platform will support the servos for both camera and cannon as well as the water pump. The camera and cannon will be on opposing sides of the vehicle with the pump center between the castor and two wheels.

For practical and aesthetic purposes, the entire mobile platform will be covered by a hinged cover with an acrylic plastic layer to reduce the likelihood of water damage.

ELECTRONICS

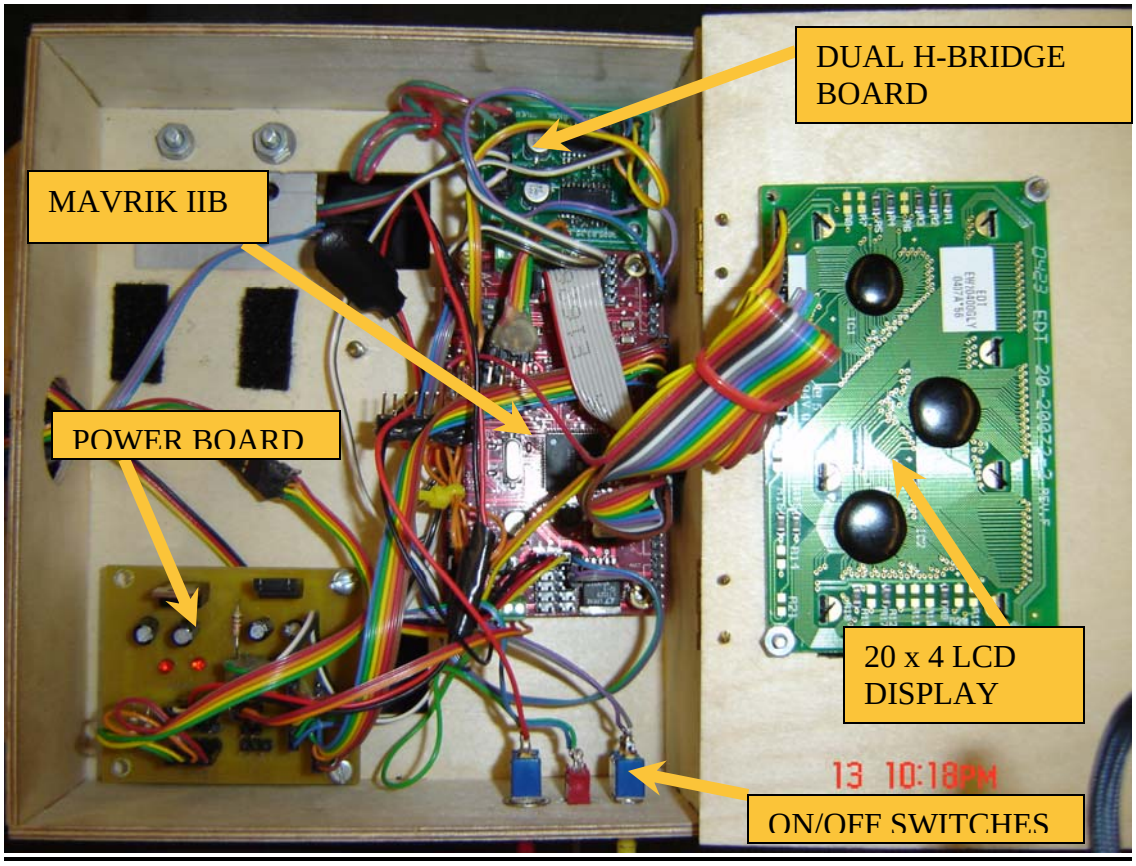


Figure 2: Electronics of Red Bulldog

LINE FOLLOWING BOARD

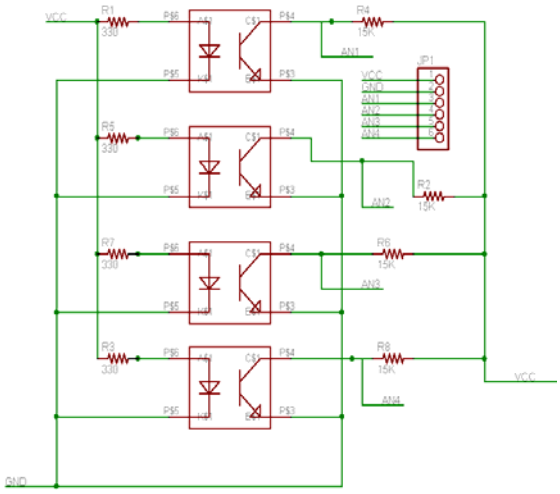


Figure 3: Schematic for Line Following PCB

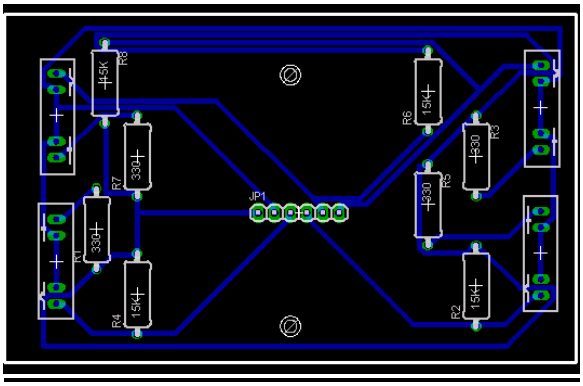


Figure 4: Board for Line Following PCB

Figure 5: Line Following Array attached

POWER BOARD

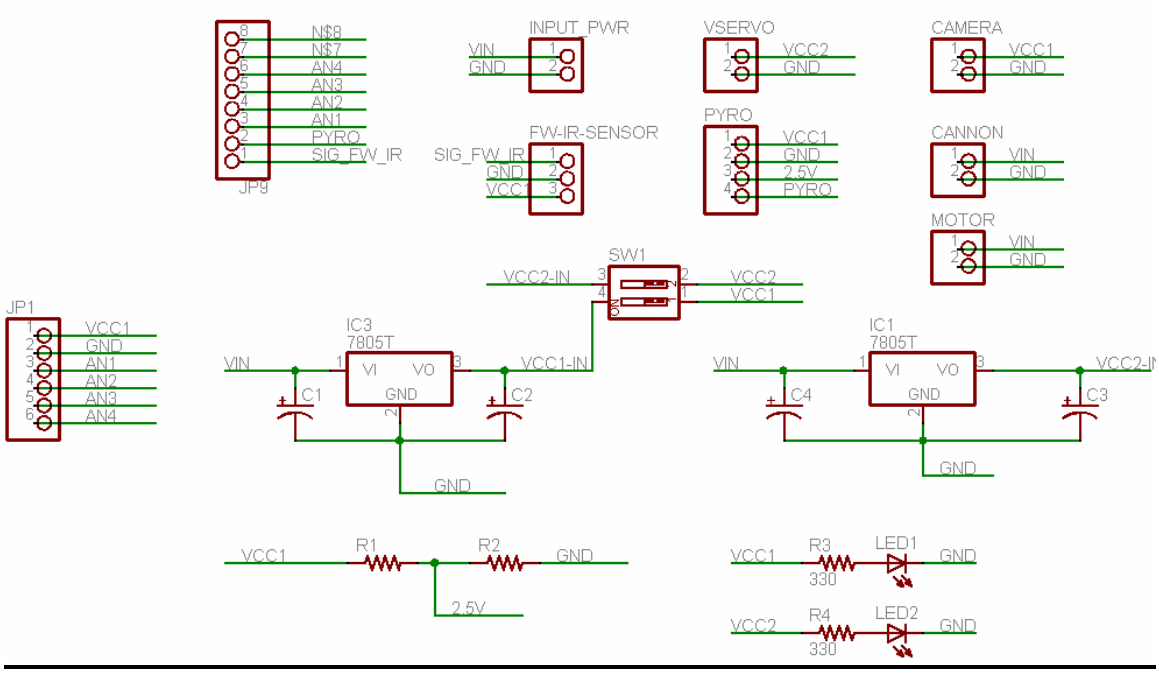


Figure 6: Schematic for Power Board

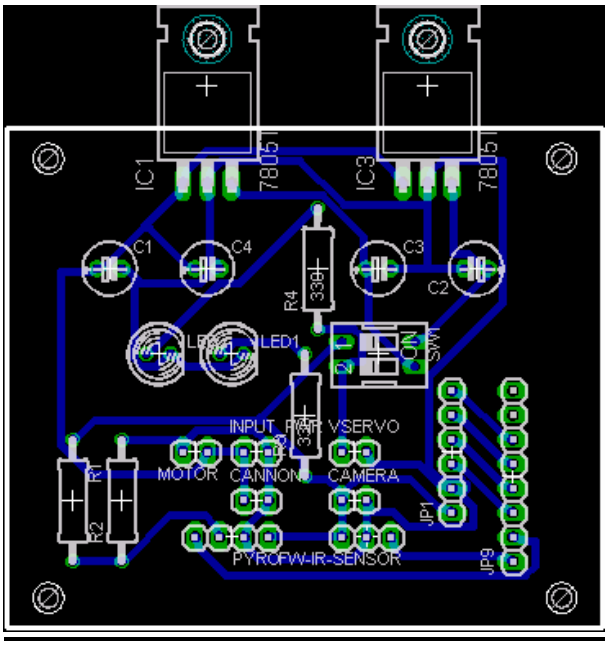


Figure 7: Board for Power Board

Actuation

For locomotion two spur motors propel the agent. These particular motors had a favorable torque to speed ratio leaning toward the higher torque necessary to propel the mobile platform. They are driven by a 3 amp dual driver board designed by William Dubel. They are attached via Aluminum hubs and 2.75 in diameter wheels.

Item	Voltage	Torque	Velocity	Shaft Dia.	Out. Diam
Gear Head Motor	12V	123.20 oz-in	120 rpm	6 mm	37 mm

The water cannon is a combination of a nozzle, wiper pump, and a small water tank. The nozzle is attached to a pan/tilt servo pair allowing it to have greater maneuverability. The wiper pump is a 12V pump, whose purpose is to pump water from the tank for a very limited time because of the high power consumption. It is a centripetal pump only capable of pumping water that is held above it. It is capable of producing a stream that can reach upward 20 ft with a constant 12V such that it would produce a water stream sufficient enough with a voltage around 10-12V.

Sensors

Pyroelectric Sensor

The pyroelectric sensor is a rather difficult sensor to use. It is composed of two parallel IR sensors which use the difference of the analog IR outputs for the detection of infrared radiation such as the radiation from a burning candle or human. As previously mentioned, humans can cause the same effect as the candle such that care must be taken to form a perimeter around the test field. The key in the correct usage of the pyroelectric sensor is that either the source must move or the sensor must move in order for a correct reading to be obtained.

CMU CAM

The CMU camera offers a low cost and power solution to viable computer vision techniques. Since time was an issue, the track color module, which was already created, was used. After receiving min and max for each color channel—red, blue and green, the camera returns the middle of mass x and y, the window enclosing the mass and the confidence of the mass. This data is returned to the microcontroller where the appropriate actions are taken.

IR Proximity Detector

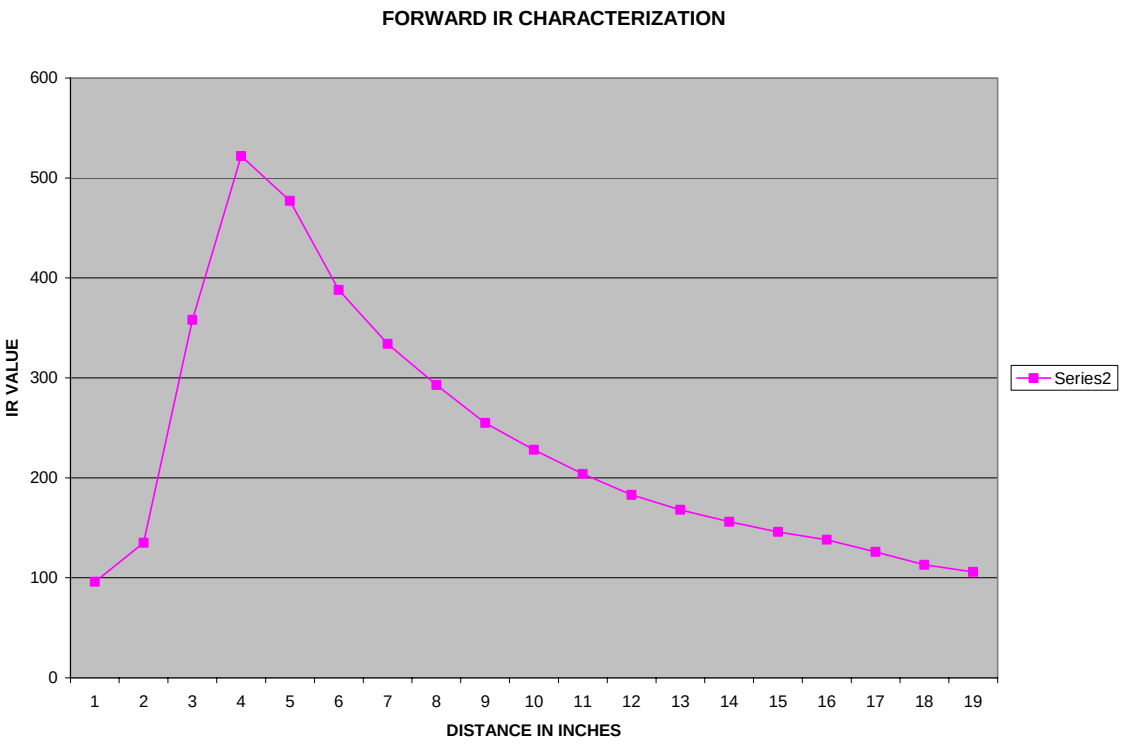


Figure 8: IR Characterization of the forward GPD120 proximity detector

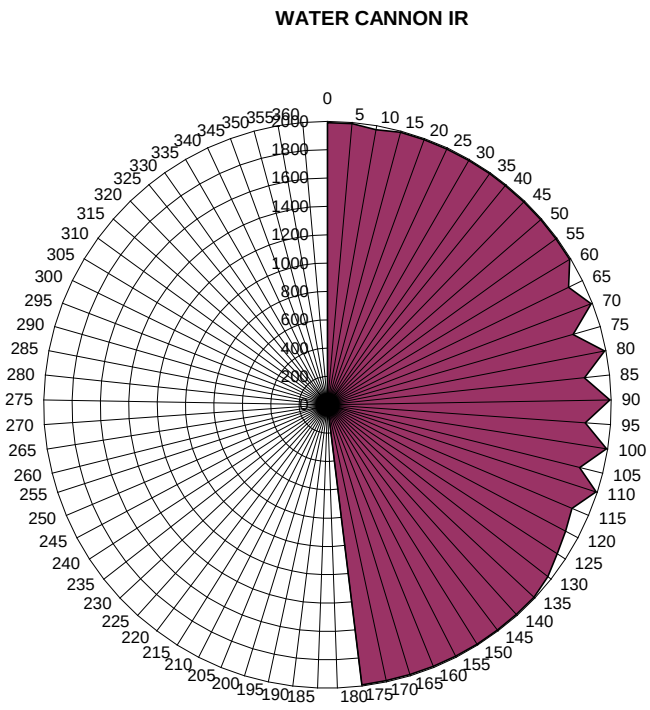


Figure 9: The tilt minimum of the GPD120 on the pan/tilt array. This minimum is required to ensure that the GPD120 does not detect the edges of the actual agent.

Behaviors

Line Following

The line following behavior is a rather simple technique. For this sensor the two forward OPB745 of the bottom sensor array are used. The OPB745s are analog sensor such that analog values range between 0 to 1024. The sensors on average gave approximately 100 over white and 500 over black. Since the two OPB745s are mounted such that if they are placed over the white tape with each sensor seeing the same amount of white and black, they should ideally have the same value. With this known, any

variation from this will tell the microcontroller everything that it needs to adjust the motors to compensate. By comparing the values of both sensors, in this case (Left_value-Right_value), both the side of the line and the error of how far it is off the line such that the microcontroller can adjust the speed of each motor proportionally to the distance off the line. This allows it handle various degrees of curves at higher speeds because it does not depend on discrete motor speeds.

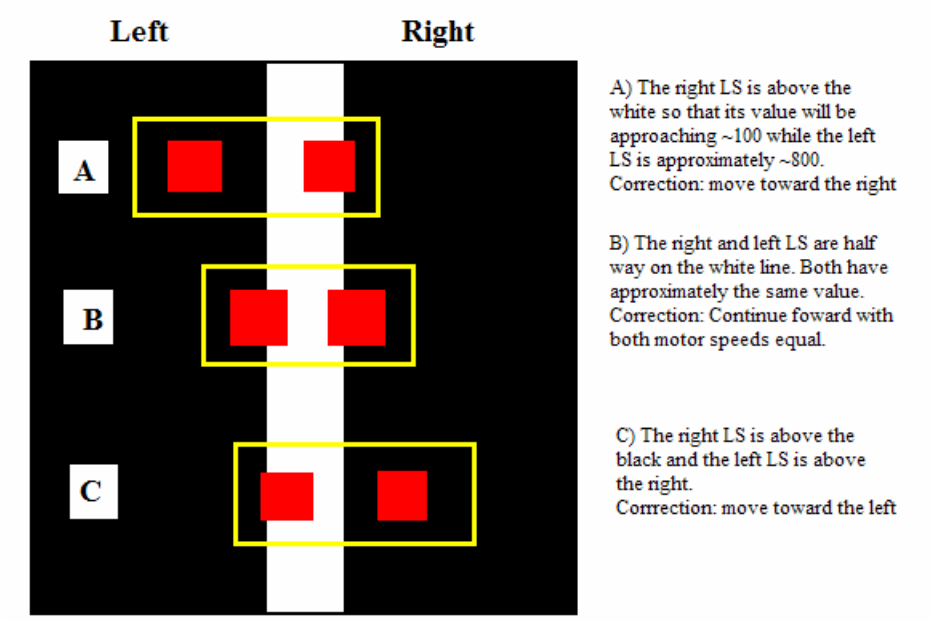


Figure 10: Illustrations of the routine for Line Following

This behavior presented a serious of problems. The problems seemed to stem from the hardware, essentially either the OPB745 or the PCB. These apparent problems were later found not to be the culprit. Essentially the mistake was in the testing apparatus. Black electrical tape was used on white poster board (later on white electrical tape and black painted plywood). Soon

it was found that the black tape reflected as much IR as the white poster board. This was later remedied in the final testing apparatus by painting the plywood with flat black paint and later sanding it such that a rough texture was remained. This remedy resulted in the line following circuitry and routines never causing additional problems.

Intersection Detection

Intersection detection is nothing more that a thresholding and transfer behavior. At the beginning of the test, the robot is position so that the back pair of OPB745s is position over the white tape parallel to the axis of the wheels. During the configuration routine that is run before anything is done, the value of both these are averaged and saved as the threshold value. This threshold value is constantly being checked until the threshold value of both back sensors meet the same condition. When this threshold is found, control is transferred to the Tracking Fire behavior.

Obstacle Avoidance

The obstacle avoidance is implemented by the GPD120 on the forward pointing sensor array. At the beginning of a run, the GPD120 is pointed at a building directly diagonal to it. That value is used as a threshold value which will be used to determine the likelihood of an obstacle. During the pan of the pyro-sensor, the microcontroller determines whether there is

an obstacle to the far left , center, and far right. This information is returned to the arbitrator, which is used to determine the next planned move.

Tracking the fire

Determining the direction of a possible fire is the job of the pyroelectric sensor. At an intersection the forward array is panned across the left to right until the pyro-sensor registers neutral which is 2.5V or approximately 500. At this point the pyro-sensor is slowly panned from left to right registering the max and min value of the entire sweep as well as their servo position. Approximately at the center of the min and max is the location of the fire. This value converted into a likely direction for which the robot should take.

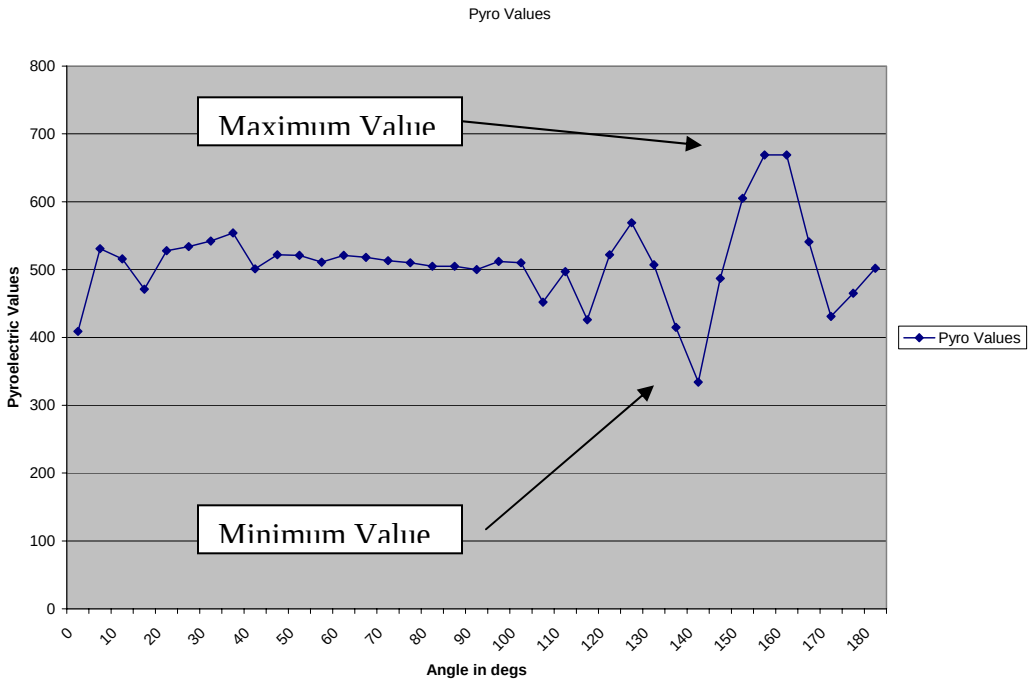
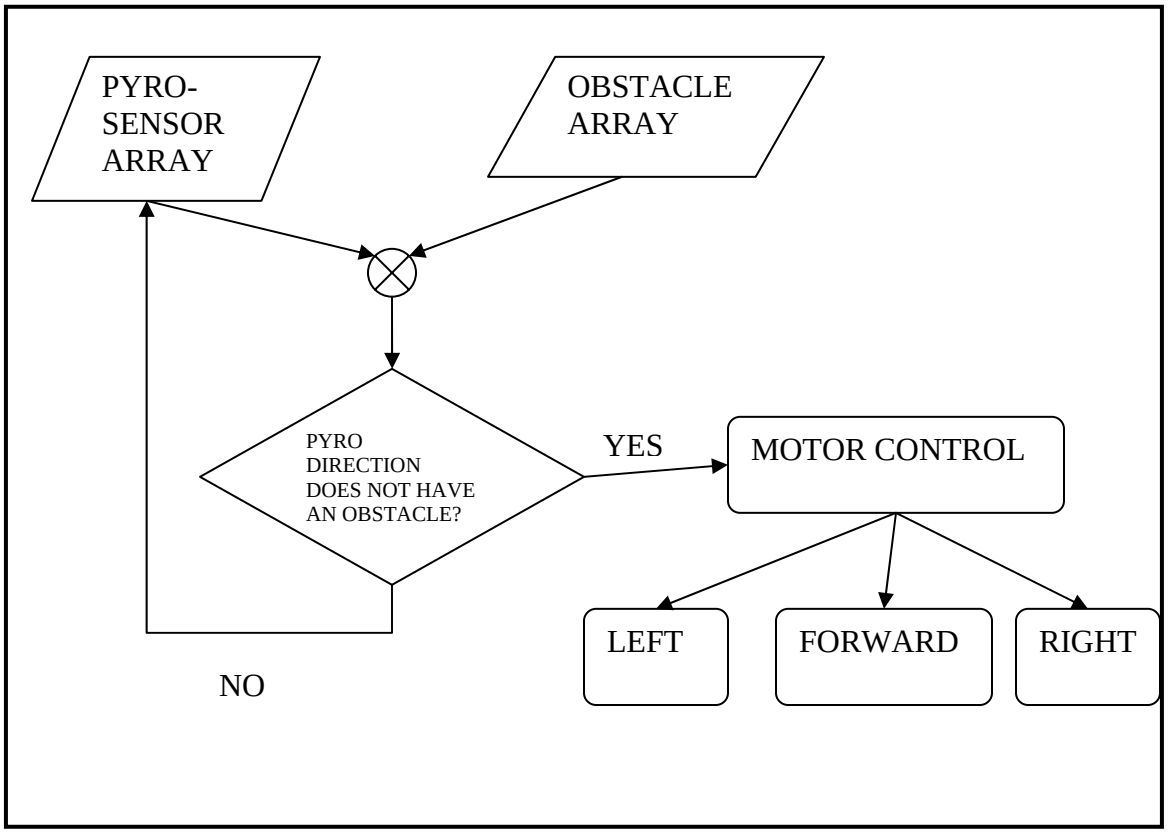


Figure 11: Pyro Characteristics

Arbitration

The arbitrator of the robot is located in the main program. It takes three factors into consideration—1) the direction determined by the pyro-sensor, 2) the obstacle avoidance information, 3) and the x/y edge of the world coordinates. The pyro-sensor actually gives much more information than one simple direction but it is an array of possible direction in order of increased likelihood. After all sensors report their information, the arbitrator first considers the first choice of the pyro-sensor. It determines whether an obstacle was located in that direction. If one was found, it considers the next viable pyro-sensor direction. If not, it checks the direction and x/y coordinates of the robot in order to ensure that the robot does not go

out of the test field. After determining the direction to turn, the arbitrator sends one of three minor motor behaviors: forward, right or left.



Targeting the flame

After determining that the flame is in the square adjacent to the robot, the microcontroller then polls the CMU camera with track color commands. It uses the middle mass information to line up the center of the camera's viewing window with the center of the middle mass. Upon centering, the camera is offseted slightly to account for the location of the nozzle with respect to the center of the camera's window.

Extinguishing

The extinguishing behavior is the simplest of the entire behavior array. It consists of pulling the pin 0 on PORTD high for two seconds. This change actives the npn transistor in line with the 12 volt battery pack. Future work might include placing the pump on a PWM to allow for variable speed and force streams of water.

Conclusion

This project offered varying obstacles including hardware and software, but the biggest of all was time. Time established the pace at which most of my mini-projects were finished. Nonetheless, the effort and time placed in this project in robot which surpassed some of my expectations, especially those of them made around three hours before DEMO day. On its demonstration, it actually accomplished everything that it was supposed to do short of sending a stream of water toward the fire. As a result of human error (low batteries) it could only supply a tinkling of water; however, this tinkling of water proved that the concept of autonomous firefighting can be realized in a controlled environment.

Special thanks to:

- Prof. Arroyo and Schwartz for giving me the opportunity here at the MIL and their encourage
- All the other REU participants of 2005 at the MIL
- Clint Doriot for troubleshooting and the LCD_SET_POSITION function
- Sara Keen, Steven Pickles and Julio Suarez for their UART interfacing code.
- Adam Barnett for his financial support for CMUcam1

APPENDIX

SOURCE CODE

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: VERSION_1.C
* DESCRIPTION:
*
* THIS IS THE MAIN PROGRAM FOR THE RED BULLDOG. IT CONTAINS THE ENTIRE MAIN ROUTINE
* IN WHICH ALL THE FUNCTIONS ARE INITIALLY CALLED.
*****************************************************************************************/

//Header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <inttypes.h>
#include "adc.h"
#include "LCD.h"
#include "UART.h"
#include "SENSOR.h"
#include "MOTOR.h"
#include "SERVO.h"
#include "CMU.h"
#include <avr/iom128.h>

//Define Statements

#define OCR_1MS 250

//Variable Declarations
int SERVO1_DIR=0;//1 to the right, 0 to the left
volatile uint16_t ms_count;
const int maxtext = 40;//K=0;

int STAT0,i=1500,FIRE_DIR[4],
X_COOR=1,Y_COOR=1,CURRENT_DIR=0,FW_IR_OBSTACLE[3],COMPLETION=0,FIRE_DIRECTION;
volatile uint8_t   UART0_ReceivedChar;
uint8_t   UART0_RxChar;
long OLDRMSPEED,OLDLMSPEED;
uint16_t FORWARD_IR_SENSOR,PYRO_SENSOR,LEFT_LINE_SENSOR, RIGHT_LINE_SENSOR,LINE_ERROR;
uint16_t BK_LEFT_LINE_SENSOR, BK_RIGHT_LINE_SENSOR,INTERSECTION_THRESHOLD;
uint16_t NORM_BK_LEFT_LINE,NORM_BK_RIGHT_LINE,OBSTACLE_THRESHOLD,WATER_IR;

int GM_ARRAY[9],TC_ARRAY[9];

//Function Prototypes
void delay(uint16_t);
void DELAY(void);
void DELAY2(void);
void INIT_TIMER(void);
void CONFIGURE_SENSORS(void);
void ALL_DEVICE_INIT(void);
int ARBITRATOR(void);
void AIMING(int);
void TRACK_FLAME(void);
void CANNON_FIRE(void);

//Function
/*
 * delay() - delay for specified number of milliseconds
 */
void delay(uint16_t ms)
{
  TCNT0  = 0;
  ms_count = 0;
  while (ms_count != ms);
}
```

```
/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{

  ms_count++;
}


/*
 * Initialize timer0 to use the main crystal clock and the output
 * compare interrupt feature to generate an interrupt approximately
 * once per millisecond to use as a general purpose time base.
 */
void INIT_TIMER(void)
{
  TCCR0 = 0;
  TIFR  |= _BV(OCIE0)|_BV(TOIE0);
  TIMSK |= _BV(TOIE0)|_BV(OCIE0);        /* enable output compare interrupt */
  TCCR0  = _BV(WGM01)|_BV(CS02)|_BV(CS00); /* CTC, prescale = 128 */
  TCNT0  = 0;
  OCR0   = OCR_1MS;                    /* match in aprox 1 ms */
}
void INIT_TIMER2(void)
{
                TIFR |= 0x80; // clear flag
                TCCR2 = 0b00001011; // timer 2 output compare with 256 prescaler *CTC
                TCNT2  = 0;
                TIMSK |= 0x80; // timer2 output compare intterupt enable
                OCR2 = 255;
}


void DELAY(void)
{
  volatile uint16_t time1;
  for(time1 = 0; time1 < 2000; time1++);
}


void DELAY2(void)
{
  int u;
  volatile uint16_t time1;
  for(u=0;u < 50; u++)
  {
    for(time1 = 0; time1 < 20000; time1++);
  }
}


void ALL_DEVICE_INIT()
{
/*********************************************************************
 * Initialization function dedicated to initializing all devices   *
 * that have separate initialization functions                      *
 *********************************************************************/
                //Timer Initialization
                INIT_TIMER();
                //INIT_TIMER2();

                //LED initialization
                DDRB = 0x01;  /* enable PORTB 1 as an output */
                DDRD = 0x01;  /* enable PORTB 1 as an output */
                PORTD=0X00;
                //LCD INTIALIZATION
                LCD_INIT();
```

```
        //MOTOR & SERVO initialization
        SERVO_INIT();
        MOTOR_INIT();

        //UART Initalization
        USART0_INIT();
        //AD INITIALIZATION
        adc_init();
        //CMU CAMERA INITIALIZATION

       CONFIGURE_SENSORS();
        //SET ALL DEVICE POSITION
       SERVO_CONTROL(1,1500);
       SERVO_CONTROL(2,1400);
       SERVO_CONTROL(3,1550);
       SET_TABLE();
}


int ARBITRATOR(void)
{


/*
*CURRENT DIRECTION
*DEFAULT 0
*0-NORTH
*1-EAST
*2-SOUTH
*3-WEST
*
*
*/


for (int j = 0;j < 3;j++)
{

        if (FW_IR_OBSTACLE[FIRE_DIR[j]] ==0)
        {
                /*********************************************************************/
                //CURRENT DIRECTION IS NORTH AND PROJECTED DIRECTION IS LEFT
                if ((FIRE_DIR[j] == 0) && (CURRENT_DIR == 0))
                {

                        if (((X_COOR-1) > 0)&&((X_COOR-1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
                        {
                                X_COOR--;
                                FULL_TURN(0);
                                CURRENT_DIR = 3;
                                LCD_SEND_COMMAND(0x01);
                                LCD_WRITE_DATA("LEFT");
                                return 0;
                        }
                }

                //CURRENT DIRECTION IS EAST AND PROJECTED DIRECTION IS LEFT
                if ((FIRE_DIR[j] == 0) && (CURRENT_DIR == 1))
                {

                        if ((X_COOR > 0)&&(X_COOR < 4)&& ((Y_COOR+1) > 0)&&((Y_COOR+1) < 4))
                        {
                                Y_COOR++;
                                FULL_TURN(0);
                                CURRENT_DIR = 0;
                                LCD_SEND_COMMAND(0x01);
                                LCD_WRITE_DATA("LEFT");
                                return 0;
```

```
		}
}

//CURRENT DIRECTION IS SOUTH AND PROJECTED DIRECTION IS LEFT
if ((FIRE_DIR[j] == 0)&&(CURRENT_DIR == 2))
{

		if (((X_COOR+1) > 0)&&((X_COOR+1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
		{
			X_COOR++;
			FULL_TURN(0);
			CURRENT_DIR = 1;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("LEFT");
			return 0;
		}
}


//CURRENT DIRECTION IS WEST AND PROJECTED DIRECTION IS LEFT
if ((FIRE_DIR[j] == 0)&&(CURRENT_DIR == 3))
{

		if ((X_COOR > 0)&&(X_COOR < 4)&&((Y_COOR-1) > 0)&&((Y_COOR-1) < 4))
		{
			Y_COOR--;
			FULL_TURN(0);
			CURRENT_DIR = 2;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("LEFT");
			return 0;
		}
}

/*********************************************************************/
//CURRENT DIRECTION IS NORTH AND PROJECTED DIRECTION IS FORWARD
if ((FIRE_DIR[j] == 2)&&(CURRENT_DIR == 0))
{

		if ((X_COOR > 0)&&(X_COOR < 4)&& ((Y_COOR+1) > 0)&&((Y_COOR+1) < 4))
		{
			Y_COOR++;
			FULL_TURN(2);
			CURRENT_DIR = 0;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("FORWARD");
			return 0;
		}
}

//CURRENT DIRECTION IS EAST AND PROJECTED DIRECTION IS FORWARD
if ((FIRE_DIR[j] == 2)&&(CURRENT_DIR == 1))
{

		if (((X_COOR+1) > 0)&&((X_COOR+1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
		{
			X_COOR++;
			FULL_TURN(2);
			CURRENT_DIR = 1;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("FORWARD");
			return 0;
		}
}

//CURRENT DIRECTION IS SOUTH AND PROJECTED DIRECTION IS FORWARD
if ((FIRE_DIR[j] == 2)&&(CURRENT_DIR == 2))
{

		if ((X_COOR > 0)&&(X_COOR < 4)&& ((Y_COOR-1) > 0)&&((Y_COOR-1) < 4))
```

```
		{
			Y_COOR--;
			FULL_TURN(2);
			CURRENT_DIR = 2;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("FORWARD");
			return 0;
		}
}


//CURRENT DIRECTION IS WEST AND PROJECTED DIRECTION IS FORWARD
if ((FIRE_DIR[j] == 2)&&(CURRENT_DIR == 3))
{

		if (((X_COOR-1) > 0)&&((X_COOR-1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
		{
			X_COOR--;
			FULL_TURN(2);
			CURRENT_DIR = 3;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("FORWARD");
			return 0;
		}
}

/*********************************************************************/
//CURRENT DIRECTION IS NORTH AND PROJECTED DIRECTION IS RIGHT
if ((FIRE_DIR[j] == 1)&&(CURRENT_DIR == 0))
{

		if (((X_COOR+1) > 0)&&((X_COOR+1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
		{
			X_COOR++;
			FULL_TURN(1);
			CURRENT_DIR = 1;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("RIGHT");
			return 0;
		}
}

//CURRENT DIRECTION IS EAST AND PROJECTED DIRECTION IS RIGHT
if ((FIRE_DIR[j] == 1)&&(CURRENT_DIR == 1))
{

		if ((X_COOR > 0)&&(X_COOR > 3)&& ((Y_COOR-1) > 0)&&((Y_COOR-1) < 4))
		{
			Y_COOR--;
			FULL_TURN(1);
			CURRENT_DIR = 2;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("RIGHT");
			return 0;
		}
}
//CURRENT DIRECTION IS SOUTH AND PROJECTED DIRECTION IS RIGHT
if ((FIRE_DIR[j] == 1)&&(CURRENT_DIR == 2))
{

		if (((X_COOR+1) > 0)&&((X_COOR+1) < 4)&& (Y_COOR > 0)&&(Y_COOR < 4))
		{
			X_COOR++;
			FULL_TURN(1);
			CURRENT_DIR = 3;
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("RIGHT");
			return 0;
		}
}
```

```
                        //CURRENT DIRECTION IS WEST AND PROJECTED DIRECTION IS RIGHT
                        if ((FIRE_DIR[j] == 1)&&(CURRENT_DIR == 3))
                        {

                                if ((X_COOR > 0) &&(X_COOR < 4)&& ((Y_COOR+1) > 0)&&((Y_COOR+1) < 4))
                                {
                                        Y_COOR++;
                                        FULL_TURN(1);
                                        CURRENT_DIR = 0;
                                        LCD_SEND_COMMAND(0x01);
                                        LCD_WRITE_DATA("RIGHT");
                                        return 0;
                                }
                        }


                        }
                        }
                        return 0;

                }

int ABS(int num)
{
if (num >=0)
{
        return num;
}
else
{
        return num*-1;
}
return num;
}
void AIMING(int DIR)
{
int BETA,LIMIT;
SERVO_CONTROL(1,DIR);
DIR = (DIR - 600) / 10;


if ((DIR >=0) && (DIR < 90))
{
        BETA = (90-((90 - DIR) / 2));
}
else if((DIR > 90) && (DIR <= 180))
{
        BETA = (DIR - ((DIR - 90) / 2));
}
else //(DIR==90)
{
        BETA=90;
}

BETA = 180 - BETA;

BETA = (10 * BETA) + 550;
SERVO_CONTROL(2,BETA);
LCD_SEND_COMMAND(0x01);
LCD_WRITE_DATA("CORRECT DIR");



if (((BETA >= 0)&&(BETA < 50))||((BETA > 130)&&(BETA < 180)))
{
        LIMIT=1990;
}
else
{
```

```
		LIMIT=1800;
}


TRACK_FLAME();

}

void TRACK_FLAME(void)
{

		//CENTERING THE X PLANE (UP AND DOWN)
		//
		uint8_t X_CENTER = 40;
		uint8_t Y_CENTER = 72;
		int Y_CORRECTION;
		int X_CORRECTION;
		int f = SERVO2-200;
		SERVO_CONTROL(2,f);
		int j=1850;

		LCD_SEND_COMMAND(0x01);
		LCD_WRITE_DATA("LIFTING");
		do
		{

			SERVO_CONTROL(3,j);
			j=j-20;
			i=f;

			do
			{
				SERVO_CONTROL(2,i);
				CMU_TRACK_COLOR();
				i=i+20;
				LCD_SET_POSITION(1,0);
				LCD_SEND_INT(j);
				LCD_SET_POSITION(1,8);
				LCD_SEND_INT(i);
				LCD_SET_POSITION(2,0);
				LCD_SEND_INT(TC_ARRAY[8]);

			}
			while((TC_ARRAY[8]<100) && (i< f+400));

		}
		while((TC_ARRAY[8]<100) && (j>1500));

		LCD_SEND_COMMAND(0x01);
		LCD_WRITE_DATA("TARGETING");

		do
		{
			CMU_TRACK_COLOR();

			X_CORRECTION = TC_ARRAY[1] - X_CENTER;
			if (X_CORRECTION >= 0)
			{

				LCD_SET_POSITION(1,0);
				LCD_WRITE_DATA("RIGHT");
				LCD_SET_POSITION(1,7);
				LCD_SEND_INT(X_CORRECTION);
				LCD_SET_POSITION(1,14);
				LCD_SEND_INT(TC_ARRAY[1]);
				SERVO_CONTROL(2, SERVO2 - 5);
			}
			else
			{
				LCD_SET_POSITION(1,0);
```

```
                        LCD_WRITE_DATA("LEFT ");
                        LCD_SET_POSITION(1,7);
                        LCD_SEND_INT(X_CORRECTION);
                        LCD_SET_POSITION(1,14);
                        LCD_SEND_INT(TC_ARRAY[1]);
                        SERVO_CONTROL(2, SERVO2 + 5);
                }

                //CENTERING THE Y PLANE (RIGHT AND LEFT)
                CMU_TRACK_COLOR();
                Y_CORRECTION = TC_ARRAY[2] - Y_CENTER;
                if (Y_CORRECTION >= 0)
                {
                        LCD_SET_POSITION(2,0);
                        LCD_WRITE_DATA("UP  ");
                        LCD_SET_POSITION(2,7);
                        LCD_SEND_INT(Y_CORRECTION);
                        LCD_SET_POSITION(2,14);
                        LCD_SEND_INT(TC_ARRAY[2]);
                        SERVO_CONTROL(3, SERVO3 - 5);
                }
                else
                {
                        LCD_SET_POSITION(2,0);
                        LCD_WRITE_DATA("DOWN");
                        LCD_SET_POSITION(2,7);
                        LCD_SEND_INT(Y_CORRECTION);
                        LCD_SET_POSITION(2,14);
                        LCD_SEND_INT(TC_ARRAY[2]);
                        SERVO_CONTROL(3, SERVO3 + 5);
                }

        }while((ABS(X_CORRECTION) >= 10)||(ABS(Y_CORRECTION) >= 10));
        //ALIGNS THE NOZZLE WITH THE FIRE
        SERVO_CONTROL(3,SERVO3-200);
        CANNON_FIRE();

}
void CONFIGURE_SENSORS(void)
{
/*****************************************************************************************
 *Configuring sensors involve
  -line up the line following sensor such that the front sensors are at an intersection and
   the back sensors are centered on the line behind.this gives two valuable thresholds
                        -at the intersections the threshold for how dark it should be is set
                        -
 *
 */
  // First take several samples of the front left and right line_sensors

  LCD_SEND_COMMAND(0x01);
  LCD_WRITE_DATA("CONFIGURING");


  NORM_BK_LEFT_LINE = adc_readn(2,40);
  NORM_BK_RIGHT_LINE = adc_readn(4,40);
  SERVO_CONTROL(1,2050);
  DELAY();
  OBSTACLE_THRESHOLD = adc_readn(0,40)-250;
  CMU_INIT();
  LCD_SEND_COMMAND(0x01);
  LCD_WRITE_DATA("COMPLETE");

  //This value is taken as intersection threshold
  INTERSECTION_THRESHOLD= ((NORM_BK_LEFT_LINE+NORM_BK_RIGHT_LINE) / 2)+150;

}


void CANNON_FIRE(void)
```

```
{
LCD_SEND_COMMAND(0X01);
LCD_WRITE_DATA("FIRE FIRE FIRE");
//DDRD=0X01;//ENABLE PORTD.0 AS AN OUTPUT
PORTD=0X01;

DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();
DELAY2();

PORTD=0x00;
COMPLETION=1;
}

int main(void)
{
DELAY2();
DELAY2();
ALL_DEVICE_INIT();
sei(); // enable interrupts

LCD_SEND_COMMAND(0X01);//Clear home


INTERSECTION_DETECTION();


 while (COMPLETION!=1)
{
        LINE_FOLLOWER();
        INTERSECTION_DETECTION();

}

LCD_SEND_COMMAND(0x01);
LCD_WRITE_DATA("FIRE OUT");
RMSPEED=0;
LMSPEED=0;

while(1)
{
}


}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: CMU.C
* DESCRIPTION:
*
* THIS IS THE PROGRAM WHICH CONTAINS THE CARNEGIE MELLON UNIVERSITY CAMERA SUBROUTINES
* INCLUDING THE INITIALIZATION AND TRACK COLOR FUNCTION
*****************************************************************************************/
 //Header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <inttypes.h>
#include "CMU.h"
#include "SENSOR.h"
#include "LCD.h"
#include "SERVO.h"
#include "UART.h"

void CMU_INIT(void);
void CMU_RETURN_GM(void);
void CMU_TRACK_COLOR(void);

//VARIABLE DECLARATION
int GM_ARRAY[9],TC_ARRAY[9];

//INITIALIZATION OF CMU CAMERA
void CMU_INIT(void)
{
//RESETING THE CAMERA
USART0_TRANSMIT("RS\r");
DELAY2();
DELAY2();

//SETTING TO POLL MODE
USART0_TRANSMIT("PM 1\r");
DELAY2();
DELAY2();

//SETTING TO OUTPUT IN HEX AND NO ACK/NCK
USART0_TRANSMIT("RM 3\r");
DELAY2();
DELAY2();

//SETTING GREEN LED TO ON WHEN TRACKING
USART0_TRANSMIT("L1 2\r");
DELAY2();
DELAY2();

}

//SENDS THE GET MEAN COMMAND AND RECEIVES THE DATA
void CMU_RETURN_GM(void)
{

USART0_TRANSMIT("GM\r");
//DELAY();

do
{
        GM_ARRAY[0]=USART0_RECEIVE();
}
while (GM_ARRAY[0] != 0xFE);

for (int i=1;i<8;i++)
```

```
{

        GM_ARRAY[i]=USART0_RECEIVE();
}

}

//SENDS THE TRACK COLOR COMMAND AND RECEIVES THE DATA
void CMU_TRACK_COLOR(void)
{
//SENDING THE TRACK COLOR COMMAND WITH THE RGB OF THE CANDLE

USART0_TRANSMIT("TC 220 240 200 240 200 240\r");

//WAITING UNTIL THE BEGINNING OF THE PACKET IS RECEIVED

while(USART0_RECEIVE() != 255);


int j=0;

//PLACING EACH PIECE OF DATA INTO AN ARRAY
do
{
        TC_ARRAY[j] = USART0_RECEIVE();
        j++;

}while(j != 9);

}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: LCD.C
* DESCRIPTION:
*
* THIS IS THE LCD HEADER FILE WHICH CONTAINS ALL THE SUBROUTINES THAT INVOLVE THE LCD
* DISPLAY
*****************************************************************************************/
//Header files
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/signal.h>

//Define Statements
#define EN 0x10
#define RS 0x40

//Variable Declarations
volatile uint16_t ms_count;
const int maxtextLCD = 40;
const unsigned char Cursor_Set = 0x80;
const unsigned char LINE1 = 0x00;
const unsigned char LINE2 = 0x40;
const unsigned char LINE3 = 0x14;
const unsigned char LINE4 = 0x54;
char text(maxtextLCD);

//Function Prototypes
void delay(uint16_t);
void LCD_SEND_COMMAND(unsigned char);
void LCD_SET_POSITION(int,int);
void LCD_INIT(void);
void LCD_SEND_DATA(unsigned char);
void LCD_WRITE_DATA(char[]);
void LCD_SEND_INT(int);

//Functions

void LCD_INIT(void)
{
  DDRA = 0xFF;  /* enable PORTA as an output */
  DELAY();//delay(15); //15 ms delay
  LCD_SEND_COMMAND(0x33);//4 bit mode enable
  LCD_SEND_COMMAND(0x32);//4 bit mode enable
  LCD_SEND_COMMAND(0X28);//Enable 2-line mode
  LCD_SEND_COMMAND(0X0F);//Display, cursor, blink
  LCD_SEND_COMMAND(0X01);//Clear home
}


void LCD_SET_POSITION(int row, int col)
{
//ROWS 1-4
//COL 0-19
//Create basic cursor position command that will be used to set the position
unsigned char Cursor_Pos_Cmd =0x00;

//Change the command to account for the specified row
if (row == 1) Cursor_Pos_Cmd = LINE1;
else if (row == 2) Cursor_Pos_Cmd = LINE2;
else if (row == 3) Cursor_Pos_Cmd = LINE3;
else Cursor_Pos_Cmd = LINE4;

//Change the command to account for the column by offsetting the address
Cursor_Pos_Cmd += col;
```

```
//Change the command to be write cursor position command by
Cursor_Pos_Cmd |= Cursor_Set;

//Change the cursor
LCD_SEND_COMMAND(Cursor_Pos_Cmd);
}
void LCD_SEND_COMMAND(unsigned char b)
{
  PORTA=(b>>4)|EN;
  DELAY();//delay(5);  // 4.1 ms

  PORTA=(b>>4);
  DELAY();//delay(5); // 4.1 ms

  PORTA=(b & 0x0F)|EN;
  DELAY();//delay(5); // 4.1 ms

  PORTA=(b & 0x0F);
  DELAY();//delay(5); //4.1 ms
}


void LCD_SEND_DATA(unsigned char b)
{
PORTA=(b>>4)|EN|RS;
DELAY();//delay(5);  // 4.1 ms

PORTA=(b>>4)|RS;
DELAY();//delay(5); // 4.1 ms

PORTA=(b & 0x0F)|EN|RS;
DELAY();//delay(5); // 4.1 ms

PORTA=(b & 0x0F)|RS;
DELAY();//delay(5); //4.1 ms
}

//SENDS TEXTS TO LCD DISPLAY
void LCD_WRITE_DATA(char text[maxtextLCD])
{
        int i=0;
        char b;
        while ((i < (maxtextLCD+1)) & (text[i] != 0x00))
        {
                b=text[i];
                LCD_SEND_DATA(b);
                i++;
        }
}

//SENDS NUMBERS OR VARIABLES TO LCD DISPLAY
void LCD_SEND_INT(int val)
{
if (val < 0)
{
        LCD_WRITE_DATA("-");
        val = -1 * val;
}
else
{
        LCD_WRITE_DATA(" ");
}

LCD_SEND_DATA(val/10000 + 0x30);
LCD_SEND_DATA((val%10000)/1000 + 0x30);
LCD_SEND_DATA((val%1000)/100+0x30);
LCD_SEND_DATA((val%100)/10+0x30);
LCD_SEND_DATA(val%10+0x30);
}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: MOTOR.C
* DESCRIPTION:
*
* THIS IS THE MOTOR PROGRAM WHICH CONTAINS THE INITIALIZATION AND MOTOR CONTROL SUBROUTINE
*****************************************************************************************/
 //Header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <inttypes.h>
#include "SENSOR.h"
#include "LCD.h"
#include "adc.h"

//Define Statements

#define LMSPEED OCR3A
#define RMSPEED OCR3B
#define LMDIR 1
#define RMDIR 2

//Variable Declarations
long OLDRMSPEED=0,OLDLMSPEED=0;
const int K=0;
uint16_t FORWARD_IR_SENSOR,PYRO_SENSOR, LEFT_LINE_SENSOR, RIGHT_LINE_SENSOR,LINE_ERROR;
uint16_t BK_LEFT_LINE_SENSOR, BK_RIGHT_LINE_SENSOR,INTERSECTION_THRESHOLD;

//Function Prototypes
void DELAY(void);
void DELAY2(void);
void INIT_TIMER(void);
void MOTOR_CONTROL(int,int);
void MOTOR_INIT(void);
void FULL_TURN(int);

//Functions

void MOTOR_CONTROL(int m,int SPEED)
{
/*************************************************************************
 *after receiving which motor, its speed, it sends the
 *corresponding values to the pwm channels on timer 3 A and B
 *************************************************************************/

 if(m==1)//MOTOR 1 IS LEFT MOTOR
            {
            long a=((OLDLMSPEED*K)/(K+1));
            long b=(SPEED/(K+1));
            OLDLMSPEED = a + b;

                        if (OLDLMSPEED < 0)
                        {
                                    PORTB &=~_BV(LMDIR);
                                    LMSPEED=-1*OLDLMSPEED;
                        }
                        else
                        {
                                    PORTB|=_BV(LMDIR);
                                    LMSPEED=OLDLMSPEED;
                        }
   }
 else //MOTOR 2 IS RIGHT MOTOR
            {
```

```
		long a=((OLDRMSPEED*K)/(K+1));
		long b=(SPEED/(K+1));
		OLDRMSPEED = a + b;

			if (OLDRMSPEED < 0)
			{
				PORTB|=_BV(RMDIR);
				RMSPEED = -1*OLDRMSPEED;
			}
			else
			{
				PORTB &=~_BV(RMDIR);
				RMSPEED = OLDRMSPEED;
			}
		}
}


void MOTOR_INIT()
{
/*********************************************************************
 *Initializes the motor max to between 0-20000 or 0-100% duty cycle*
 *********************************************************************/
 DDRE=0xFE; //E4,E5,E6 ARE THE PORTS USED FOR PWM ON TIMER 3
 TCCR3A=0xA8;
 TCCR3B=0x12;
 ICR3=20000;
}

void FULL_TURN(int dir)
{
/***************************************
 *CCW/left 0
 *CW/right 1
 *FORWARD 2
 *FULL 180 3
 ***************************************/
 if (dir == 0) // LEFT
 {
		//SETS THE MOTORS TO A SLOW COUNTERCLOCKWISE ROTATION
		 RMSPEED=5000;
		 LMSPEED=5000;
		 PORTB &=~_BV(LMDIR);
		 PORTB &=~_BV(RMDIR);
		 DELAY2();
		 DELAY2();
		 DELAY2();


		 //WAITS UNTIL THE INTERSECTION IS DETECTED
		 while ((BK_LEFT_LINE_SENSOR >=INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
>=INTERSECTION_THRESHOLD))
		 {
			BK_LEFT_LINE_SENSOR =adc_readn(2,10);
			BK_RIGHT_LINE_SENSOR =adc_readn(4,10);
		 }
		 RMSPEED=0;
		 LMSPEED=0;

		 //LINE FOLLOWS UNTIL THE BACK SENSORS NO LONGER DETECT THE INTERSECTION
		 while ((BK_LEFT_LINE_SENSOR <INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
<INTERSECTION_THRESHOLD))
		{
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("LEAVING");
			LINE_FOLLOWER();
		}
 }
 else if (dir == 1) // RIGHT
 {
```

```
		//SETS THE MOTORS TO A SLOW CLOCKWISE ROTATION
		RMSPEED=5000;
		LMSPEED=5000;
		PORTB|=_BV(RMDIR);
		PORTB|=_BV(LMDIR);
		DELAY2();
		DELAY2();
		DELAY2();


		 //WAITS UNTIL THE INTERSECTION IS DETECTED
		while ((BK_LEFT_LINE_SENSOR >=INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
>=INTERSECTION_THRESHOLD))
		{
			BK_LEFT_LINE_SENSOR =adc_readn(2,10);
			BK_RIGHT_LINE_SENSOR =adc_readn(4,10);
		}
		RMSPEED=0;
		LMSPEED=0;

		 //LINE FOLLOWS UNTIL THE BACK SENSORS NO LONGER DETECT THE INTERSECTION
		while ((BK_LEFT_LINE_SENSOR <INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
<INTERSECTION_THRESHOLD))
		{
			LCD_SEND_COMMAND(0x01);
			LCD_WRITE_DATA("LEAVING");
			LINE_FOLLOWER();
		}
 }
else if (dir == 2)	//STRAIGHT FORWARD
{
		//LINE FOLLOWS UNTIL THE BACK SENSORS NO LONGER DETECT THE INTERSECTION
		while ((BK_LEFT_LINE_SENSOR <INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
<INTERSECTION_THRESHOLD))
			{
				LCD_SEND_COMMAND(0x01);
				LCD_WRITE_DATA("LEAVING");
				RMSPEED=5000;
				LMSPEED=5000;
				PORTB &=~_BV(RMDIR);
				PORTB|=_BV(LMDIR);
				BK_LEFT_LINE_SENSOR =adc_readn(2,10);
				BK_RIGHT_LINE_SENSOR =adc_readn(4,10);
			}
}
else if (dir == 3)	//FULL 180
{
		RMSPEED=5000;
		LMSPEED=5000;
		PORTB|=_BV(RMDIR);
		PORTB|=_BV(LMDIR);
		DELAY2();
		DELAY2();
		DELAY2();
		DELAY2();

		while (RIGHT_LINE_SENSOR >=INTERSECTION_THRESHOLD)
		{
			RIGHT_LINE_SENSOR = adc_readn(5,10);
		}
		RMSPEED=0;
		LMSPEED=0;
		DELAY2();
		DELAY2();
		DELAY2();
		RMSPEED=5000;
		LMSPEED=5000;
		DELAY2();
		DELAY2();
```

```
	while (RIGHT_LINE_SENSOR >=INTERSECTION_THRESHOLD)
	{
		RIGHT_LINE_SENSOR = adc_readn(5,10);
		//RIGHT_LINE_SENSOR = adc_readn(5,10);
		//BK_LEFT_LINE_SENSOR =adc_readn(2,10);
		//BK_RIGHT_LINE_SENSOR =adc_readn(4,10);
	}
	/*
	RMSPEED=0;
	LMSPEED=0;
	*/
	while ((BK_LEFT_LINE_SENSOR <INTERSECTION_THRESHOLD) && (BK_RIGHT_LINE_SENSOR
<INTERSECTION_THRESHOLD))
	{
		LCD_SEND_COMMAND(0x01);
		LCD_WRITE_DATA("LEAVING");
		LINE_FOLLOWER();
	}
}

}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: SENSOR.C
* DESCRIPTION:
*
* THIS IS THE SENSOR HEADER FILE WHICH CONTAINS ALL THE SENSOR ROUTINE
*
*
*
*
*
*****************************************************************************************/
 //Header files
#include <avr/io.h>
#include <inttypes.h>
#include "adc.h"
#include "SERVO.h"
#include "LCD.h"
#include "MOTOR.h"
#include "CMU.h"

void FORWARD_IR(long);
void LINE_FOLLOWER(void);
int FOWARD_PYRO(void);
void INTERSECTION_DETECTION(void);
void SET_TABLE(void);
int IR_DISTANCE(int);


int
SERVO1_DIR,HIGH_SERVO_POSITION,LOW_SERVO_POSITION,FIRE_DIRECTION,FIRE_DIR[4],CURRENT_DIR,FW_IR_
OBSTACLE[3],CURRENT_DIR,COMPLETION;
int X_COOR, Y_COOR,IR_TABLE[25];
uint16_t FORWARD_IR_SENSOR,PYRO_SENSOR, LEFT_LINE_SENSOR,
RIGHT_LINE_SENSOR,LINE_ERROR,INTERSECTION_THRESHOLD,OBSTACLE_THRESHOLD;
uint16_t BK_LEFT_LINE_SENSOR, BK_RIGHT_LINE_SENSOR,LINE_MEMORY_BITS[5];
uint16_t
NORM_BK_LEFT_LINE,NORM_BK_RIGHT_LINE,HIGH_PYRO,LOW_PYRO,NEUTRAL_PYRO,WATER_IR,PYRO_THRES
HOLD=550;



//DETERMINES WHETHER THERE ARE OBSTACLES LEFT, RIGHT OR FORWARD OF THE INTERSECTION
void FORWARD_IR(long i)
{
	FORWARD_IR_SENSOR = adc_readn(0, 10);

	//DETERMINES WHETHER THERE IS AN OBSTACLE TO THE LEFT OF THE INTERSECTION
	if (i == 600)
	{
		if (FORWARD_IR_SENSOR >= OBSTACLE_THRESHOLD)
		{
			FW_IR_OBSTACLE[0]=1;
		}
		else
		{
			FW_IR_OBSTACLE[0]=0;
		}

	}
	//DETERMINES WHETHER THERE IS AN OBSTACLE TO THE FORWARD OF THE INTERSECTION
	else if (i == 1500)
	{
		if (FORWARD_IR_SENSOR >= OBSTACLE_THRESHOLD)
		{
			FW_IR_OBSTACLE[2]=1;
```

```
                }
                else
                {
                        FW_IR_OBSTACLE[2]=0;
                }

        }
        //DETERMINES WHETHER THERE IS AN OBSTACLE TO THE RIGHT OF THE INTERSECTION
        else if (i == 2350)
        {
                if (FORWARD_IR_SENSOR >= OBSTACLE_THRESHOLD)
                {
                        FW_IR_OBSTACLE[1]=1;
                }
                else
                {
                        FW_IR_OBSTACLE[1]=0;
                }

        }


}
void LINE_FOLLOWER(void)
{

   LEFT_LINE_SENSOR = adc_readn(3,10);
        RIGHT_LINE_SENSOR = adc_readn(5,10);
        BK_LEFT_LINE_SENSOR =adc_readn(2,10);
        BK_RIGHT_LINE_SENSOR =adc_readn(4,10);
        LINE_ERROR=LEFT_LINE_SENSOR-RIGHT_LINE_SENSOR;;
        MOTOR_CONTROL(1,(7000+(15*LINE_ERROR)));
        MOTOR_CONTROL(2,(7000-(15*LINE_ERROR)));

}

int FORWARD_PYRO(void)
{

/************************************************************
*the colors on the pyro sensor are
*green - ground
*red-v+
*yellow- 2.5 v reference
*orange - output
HIGH_PYRO,LOW_PYRO,NEUTRAL_PYRO
************************************************************/
FW_IR_OBSTACLE[0]=0;
FW_IR_OBSTACLE[1]=0;
FW_IR_OBSTACLE[2]=0;
 //INITIALIZE SERVO TO LEFT POSITION

 //WAIT TILL PYROSENSOR IS AT NEUTRAL POSITION ABOUT 500
 int i=600;
 do
{
         SERVO_CONTROL(1,i);
         DELAY2();
        PYRO_SENSOR = adc_readn(1, 10);
        LCD_SEND_COMMAND(0x01);
        LCD_WRITE_DATA("WAITING FOR NEUTRAL");
        LCD_SET_POSITION(2,0);
        LCD_SEND_INT(PYRO_SENSOR);
        i=i+50;
        if (i>2350) i=600;
}
while ((PYRO_SENSOR<400) || (PYRO_SENSOR>600));
int r=0;
do
{
```

```
		PYRO_SENSOR = adc_readn(1, 10);
		if ((PYRO_SENSOR>400) && (PYRO_SENSOR<600))
		{
			r++;
		}
		else
		{
			r=0;
		}
}
while (r<6);

 DELAY2();

//PLACE PYRO VALUES INTO HIGH AND LOW PYRO
HIGH_PYRO=adc_readn(1, 10);
LOW_PYRO=adc_readn(1, 10);
HIGH_SERVO_POSITION = 600;
LOW_SERVO_POSITION = 600;
//ROTATE SERVO BY ABOUT 6 DEG PAUSING IN BETWEEN
 SERVO1_DIR=0;

 for ( int i=600;i<2400;)
 {

		FORWARD_IR(i);
		LCD_SEND_COMMAND(0x01);
		LCD_SET_POSITION(1,0);
		LCD_SEND_INT(HIGH_SERVO_POSITION);
		LCD_SET_POSITION(1,8);
		LCD_SEND_INT(HIGH_PYRO);
		LCD_SET_POSITION(2,0);
		LCD_SEND_INT(LOW_SERVO_POSITION);
		LCD_SET_POSITION(2,8);
		LCD_SEND_INT(LOW_PYRO);
		LCD_SET_POSITION(3,0);
		LCD_SEND_INT(PYRO_SENSOR);
		LCD_SET_POSITION(4,14);
		LCD_SEND_INT(FW_IR_OBSTACLE[0]);
		LCD_SET_POSITION(4,8);
		LCD_SEND_INT(FW_IR_OBSTACLE[2]);
		LCD_SET_POSITION(4,0);
		LCD_SEND_INT(FW_IR_OBSTACLE[1]);


			if ((SERVO1_DIR == 0) & (i<2400))
			{
				SERVO_CONTROL(1,i);
				i=i+50;
			}

			else if ((SERVO1_DIR == 1) & (i>600))
			{
				SERVO_CONTROL(1,i);
				i=i-50;
			}

			else
			{
			if (SERVO1_DIR == 1)
				{
					SERVO1_DIR=0;
				}
			else
				{
					SERVO1_DIR=1;
				}
			}
```

```
                //SAMPLE PYRO AND PLACE INTO HIGH OR LOW BINS
                PYRO_SENSOR = adc_readn(1, 10);
                if (PYRO_SENSOR >= HIGH_PYRO)
                {
                        HIGH_PYRO = PYRO_SENSOR;
                        HIGH_SERVO_POSITION = i;
                }

                if (PYRO_SENSOR < LOW_PYRO)
                {
                        LOW_PYRO = PYRO_SENSOR;
                        LOW_SERVO_POSITION = i;
                }

        //      unsigned int DELAY_TIME;
                //if ((PYRO_SENSOR > 650)||(PYRO_SENSOR < 450))
                //{
                //      DELAY_TIME = 2;
                //}
                //else
                //{
                //      DELAY_TIME = 2;
                //}


                for(int y=0; y <8;y++)
                {
                        DELAY();
                }
                if (PYRO_SENSOR > 1005) i=2500;
}
//DETERMINES THE POSITION BASED ON THE HIGH AND LOW SERVO POSITION

        FIRE_DIRECTION=(LOW_SERVO_POSITION+HIGH_SERVO_POSITION)/2;

        //DETERMINES WHICH HEADING TO SEND BACK : FORWARD, LEFT, RIGHT

        //FIRE TO THE LEFT
        if (FIRE_DIRECTION < 1100)
        {
        FIRE_DIR[0]=0;
        FIRE_DIR[1]=2;
        FIRE_DIR[2]=1;

        }

        //FIRE TO THE FORWARD
        if ((FIRE_DIRECTION >= 1100) && (FIRE_DIRECTION < 1900))
        {
                FIRE_DIR[0]=2;
                if (FIRE_DIRECTION >= 1500)
                {
                        FIRE_DIR[1]=1;
                        FIRE_DIR[2]=0;
                }
                else
                {
                        FIRE_DIR[1]=0;
                        FIRE_DIR[2]=1;
                }

        }

        //FIRE TO THE RIGHT
        if (FIRE_DIRECTION >= 1900)
        {
        FIRE_DIR[0]=1;
        FIRE_DIR[1]=2;
        FIRE_DIR[2]=0;
        }
```

```
		//DETERMINES IF HEAT SOURCE IS IN THE SQUARE ADJACENT TO ITS LOCATION
		if ((HIGH_PYRO - LOW_PYRO) > PYRO_THRESHOLD)
		{
		//SENDS THE AIMING PROTOCOL THE DIRECTION THAT THE HEAT SOURCE IS IN
		AIMING(FIRE_DIRECTION);
		return 0;
		}

return 0;
}



void INTERSECTION_DETECTION(void)
{

if ((BK_LEFT_LINE_SENSOR < INTERSECTION_THRESHOLD) &&(BK_RIGHT_LINE_SENSOR <
INTERSECTION_THRESHOLD))
{

		//STOPS BOTH MOTORS
		LMSPEED=0;
		RMSPEED=0;

		LCD_SEND_COMMAND(0x01);
		LCD_WRITE_DATA("TURNING");

		//POLL PYRO SENSOR FOR CORRECT DIRECTION
		FORWARD_PYRO();
		if (COMPLETION == 0)
		{
			ARBITRATOR();
		}

}
}

void SET_TABLE(void)
{
IR_TABLE[0]=477;
IR_TABLE[1]=388;
IR_TABLE[2]=334;
IR_TABLE[3]=293;
IR_TABLE[4]=255;
IR_TABLE[5]=228;
IR_TABLE[6]=204;
IR_TABLE[7]=183;
IR_TABLE[8]=168;
IR_TABLE[9]=156;
IR_TABLE[10]=146;
IR_TABLE[11]=138;
IR_TABLE[12]=126;
IR_TABLE[13]=113;
IR_TABLE[14]=106;
IR_TABLE[15]=96;
IR_TABLE[16]=86;
IR_TABLE[17]=76;
IR_TABLE[18]=66;
IR_TABLE[19]=56;
IR_TABLE[20]=46;
IR_TABLE[21]=36;
IR_TABLE[22]=26;
IR_TABLE[23]=16;
IR_TABLE[24]=6;


}

int IR_DISTANCE(int DIS)
```

```
{
if (DIS > 477) return 0;
if (DIS < 6) return 100;

for (int i=0; i<25;i++)
{
        if ((IR_TABLE[i] >= DIS)&& (DIS >= IR_TABLE[i+1]))
        {
                if ((IR_TABLE[i] - DIS) >= (IR_TABLE[i+1]-DIS))
                {
                        return (i+5);
                }
                else
                {
                        return (i+4);
                }

        }
}
return 0;
}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: SERVO.C
* DESCRIPTION:
*
* THIS IS THE SERVO PROGREAM WHICH CONTAINS ALL THE INITIALIZATION AND SERVO CONTROL
*****************************************************************************************/
 //Header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <inttypes.h>

//Define Statements
#define SERVO1 OCR1A
#define SERVO2 OCR1B
#define SERVO3 OCR1C
#define SERVO4 OCR3C


//Functions
void SERVO_CONTROL(int num,int val)
{
 if(num==1)
        SERVO1=val; //SERVO 1 FORWARD SERVO 600(LEFT)-1500(CENTER)-2350(RIGHT)
 else if (num==2)
        SERVO2=val; //SERVO 2 PAN SERVO 550(RIGHT)-1400(CENTER)-2250(LEFT)
 else if (num==3)
        SERVO3=val; //SERVO 3 TILT SERVO 650(UP) -1550(CENTER) -2000(DOWN)
 else
        SERVO4=val; //SERVO 4
 }


void SERVO_INIT(void)
{
/*************************************************************************
 *after receiving which servo, its position, it sends the                *
 *corresponding values to the pwm channels on timer 1 A and B, and timer 3 C*
 *************************************************************************/

 DDRB=0xFF;//B4,B5,B6 ARE THE PORTS USED FOR PWM ON TIMER 3
 TCCR1A=0xA8;
 TCCR1B=0x12;
 ICR1=20000;
}
```

```
/*****************************************************************************************
* QUENTIN LINDSEY
* YALE UNIVERSITY
* REU SUMMER 2005
* MACHINE INTELLIGENCE LABORATORY @ UNIVERSITY OF FLORIDA
* ROBOT'S NAME: RED BULLDOG -- FIRE FIGHTING ROBOT
* PROGRAM NAME: UART.C
* DESCRIPTION:
*
* COMPLIMENTS OF STEVEN PICKLES AND SARA KEEN.
*
*
*
*
*
*****************************************************************************************/ //Header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <inttypes.h>
#include "LCD.h"

//Define Statements

#define BAUD 38400L   /* use 9600 baud for UART0 connection 9600L*/
#define BAUD_RR ((CPU_FREQ/(16L*BAUD) - 1)) /* baud rate register value for desired baud rate */
#define CPU_FREQ 16000000L  /* set to clock frequency in Hz */
#define OCR_1MS 250

//Variable Declarations

int STAT0;
volatile uint8_t   UART0_ReceivedChar;
const int maxtextUART = 40;
uint8_t   UART0_RxChar;
char new_data;
char old_data;


//Function Definition
int USART0_INIT(void)
{
                        /* enable UART0 */

        UBRR1H = 0x00;  //set up UART for 38.4k baud
        UBRR1L = 0x33;
        UCSR1A |= 0x02;
        UCSR1C = 0x06;
        UCSR1B = 0x18;
        return 0;
}

void USART0_TRANSMIT(char data[maxtextUART])
{
        int t=0;
        while ((t < (maxtextUART+1)) & (data[t] != 0x00))
        {
                /* Wait for empty transmit buffer */
                while (!(UCSR1A & (1<<UDRE1)))
                ;
                UDR1 = data[t];
                t++;
        }
}



unsigned char USART0_RECEIVE(void)
{
```

```
        //unsigned int m = 32000;
        /* Wait for data to be received */
        while ( !(UCSR1A & (1<<RXC1)) )
        {
        //m--;
        //if (m <=0) break;
        }

        /* Get and return received data from buffer */

        return UDR1;
}
```