Formal Report
‘BUZZ’

Long Vong
9363-6111

December 7, 2010

EEL4665 Intelligent Machine Design Lab

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

Teaching Assistants
Mike Pridgen
Tim Martin
Ryan Stevens
Devin Hughes
Thomas Vermeer

Table of Contents

Abstract

This paper will go through the proposal, development, and results of a robot nicknamed ‘Buzz’ as it is developed for the EEL4665/5666 Intelligent Machines Design Laboratory. The purpose of this robot is to assist a person to wake up and get out of bed at a set time. It will run away from the user forcing the person to get up and chase it. This robot will use a variety of sensors and provide feedback to the user.

Executive Summary

My goal with Buzz was to create a highly mobile robot to interact with people while serving an important function. I thought about one of the big problems I had; waking up in the morning to be on time for class. To achieve this, I have 3 alarm placed in different spots across the room so I’d have to get up at least 3 times t snooze. So I designed Buzz to help people like me wake up in the morning. At the end, Buzz was successful in meeting the goals that I set out.

Being from the electrical engineering department, I didn’t feel comfortable with a complicated mechanical design. I chose a simple two tiered circular platform for my robot. I equipped it with two DC motors and 2 ball casters for stability along with a motor driver for control of the motors. The main advantage of this is that Buzz can turn in place without hitting anything. Conversely, Buzz does not move forward and turn at the same time. While it is possible, this movement is much harder to control and the robot would have to move much slower to properly avoid obstacles. Also, towards the end, I was running out of space on my robot so adding a third platform for user interface was easy. This has taught me that the simpler designs can be the best ones.

Buzz is equipped with 10 sensors to help it navigate and perform its goals. For obstacle detecting Buzz has 3 IR sensors that lie on the mid platform and include 1 long range IR sensors facing forward and 2 midrange sensors facing 45 degrees off-center. The analog IR sensors worked extremely well detecting objects. With only 1 sensors, it could navigate an empty room effectively. The only problem the IR sensors had was it had trouble detecting table legs and chair legs. On the bottom platform there are 5 analog bump sensors to detect collisions in the event that the IR sensors fail. The IR sensors worked well so the bump sensors were hardly used except when buzz hit something under 3 inches because the IR sensors were too high for it. For detecting humans, there is a pyroelectric sensor on the back and a CMUcam1 on the front of the middle platform. The pyroelectric sensor didn’t work as well as I wanted it to due to a very noisy signal that resulted in false positives. I removed it from the final design because it worked better without it.

The CMUcam1 is the special sensor and it allows for color tracking. Many weeks were spent trying to interface the CMUcam1 to the robot serially. When implemented, the CMUcam1 worked well with tracking colors but would vary depending on the lighting. To improve the accuracy, I associated a red LED on the human to track for demonstration. The CMUcam1 would return an X-Y representation of the color blob and Buzz would use this information to turn left, right, go straight or reverse.

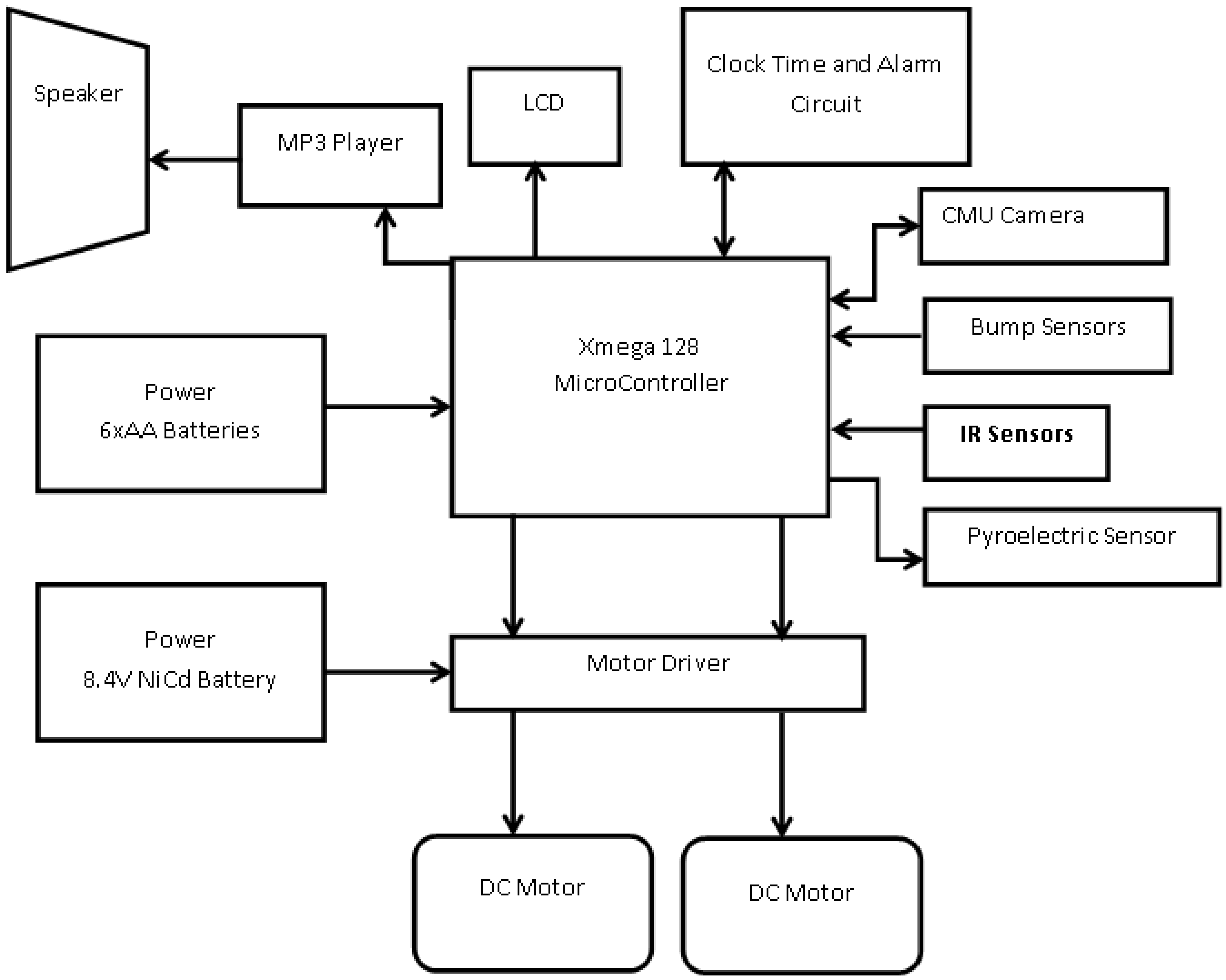
Finally I had to implement the alarm clock features into my robot. I built a small independent alarm clock circuit that would display time and keep track of the alarm. It would blink LEDs and sound a buzzer when it was time. It also set a pin on the PVR board letting know to activate Buzz. It would keep sounding until a button was pressed on Buzz. This circuit worked very well. During the last week of development, I added sound to my robot through an mp3 trigger which allowed it to play music and feedback to the user.

Introduction

Some of the simplest daily tasks can become difficult every now and then even for the best of us. How many times how you hit the snooze button on your alarm to get those extra minutes of sleep, only to oversleep and end up running late in the morning? This is where the idea for Buzz comes in. A robot that helps a person get out of bed and stay out of bed. Instead of the alarm being situated right next to bed, the alarm will be on Buzz who will run away from you so you have to get up and chase it. Then it follows you around playing music to make sure you stay out of bed while you perform your morning routine.

Integrated System

The robot uses the Pridgen Vermeer Robotics Xmega 128A1 microcontroller to control the robot. This board was built specifically for robotics has plenty of I/O pins, A/D converters for sensors, and output to an LCD screen. A system level diagram is shown below:

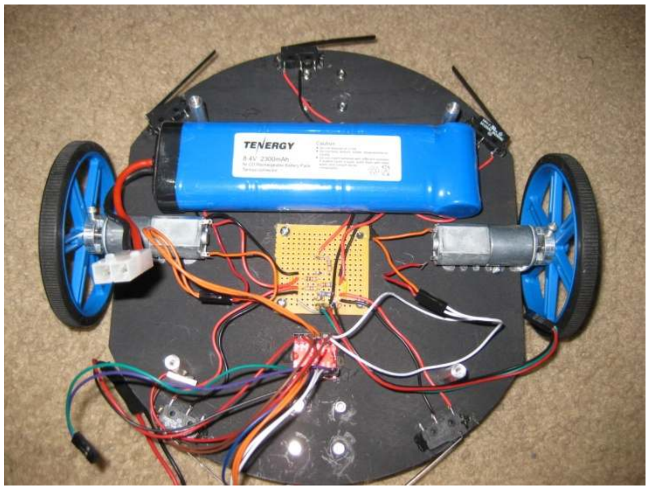


Mobile Platform

The mobile platform for my robot is made out of wood and consists of three tiers. The bottom layer houses all the actuation components, while the mid layer houses most the sensors and the main processing board. The top layer contains the user interface relaying data such as time and other information to the user.

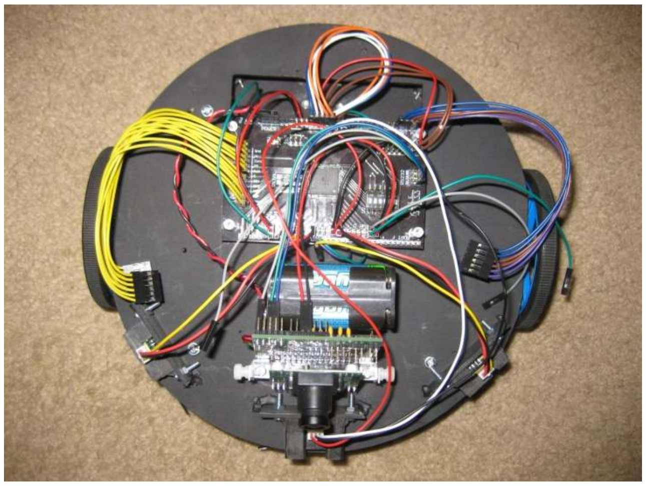
Bottom platform

- 2 67:1 Gearmotors
- 2 90cm diameter wheels
- 2 ball casters on bottom
- 2 back bump sensors
- 3 from bump sensors
- 1 dual motor driver
- 1 8.4V NiCd Battery Pack for motors



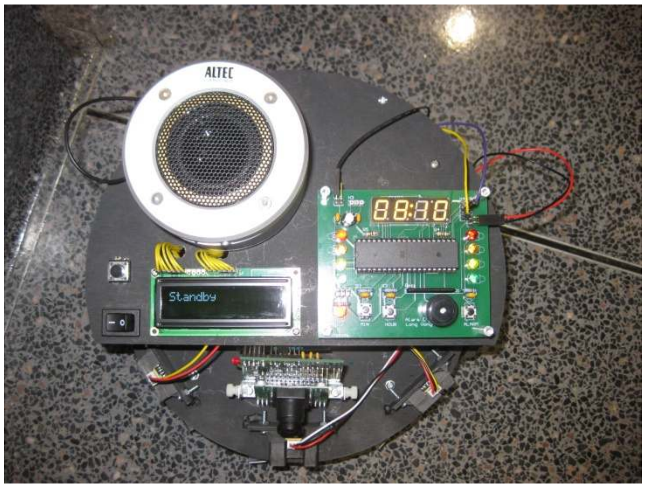
Middle Platform

- PVR board with Atxmega128A1 processor
- 1 long range Sharp GP2Y0A02 range finder
- 2 mid range Sharp GP2Y0A21range finder
- CMUcam1
- 6xAA battery pack



Top Platform

- Alarm clock circuit
- 16x2 LCD backlit display
- Altec Lansing Portable Speaker
- MP3 Trigger V2 on bottom
- Power Switch
- Alarm off button



Actuation

The robot uses two DC motors for actuation. Along with the motors, two ball casters are used for stability (pictured on the right). These were purchased from pololu.com.

The two motors have a built in gearbox with a 67:1 gear ratio (shown on right). It is rated at 210RPM at 6V with a 250mA free run current and 57oz-in of torque. These were the second set of motors used. The first could not provide adequate torque at lower speed. These have a 4mm D-shaped output shaft. These motors were attached to 90mm diameter wheels.

This design allowed Buzz to turn in place so it wouldn't hit anything while turning. When buzz detected an obstacle, it would turn in place until the area in from was clear, then it would continue going forward. This algorithm is pretty simple but worked very well for obstacle avoidance. While it is possible for Buzz to go straight and turn at the same time, this motion was found to be less effective in avoiding obstacles at the desired speed. Having such a small turning radius allowed Buzz to be highly mobile. This is one of the goals of the project, because the user would have to catch Buzz to turn off the alarm. The speed of Buzz was limited due to the speed of the sensors. If the robot was moving too fast and it detected an obstacle, it wouldn't have enough time to react and stop. At most, the motors were running on a 60% duty cycle to allow for optimal operation.

Sensors

Sharp GP2Y0A02 IR Sensor

The Sharp IR Sensor is an analog device that is capable of measuring distance. This is done using a position sensitive detector and an IR LED and the signal is processed on the sensor. It outputs an analog voltage that corresponds to distance. This unit is rated at 8 – 60in (20-150cm), however tests done were not able to achieve these distances. The figure below is from the sensor datasheet and approximates the distance to voltage relationship.

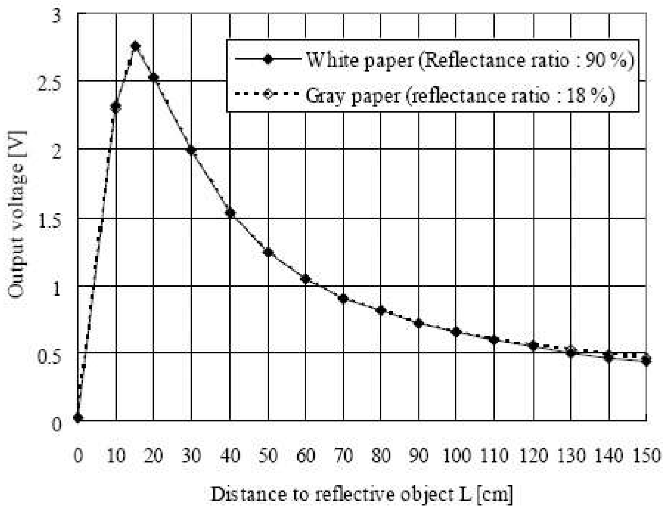


Figure 1. Sensor Output Voltage vs. Distance

Actual tests of the sensor yielded values a peak voltage of around 3.1V at a distance around 5-6 inches. The minimum useful distance yielded by this sensor is a distance around 3-4 feet with a voltage around 0.7V. This voltage is read by the A/D Converter on the Xmega128A1 and yields values from 0-4095 corresponding to 0-3.3V. The actual distance-voltage relationship is close enough to the graph above.

Sharp GP2Y0A21 IR Sensor

This sharp sensor is comparable in operation to the GP2Y0A02 IR sensor. The main difference is the distance that it can detect which is rated for 4– 32in (10-80cm). Test done yielded a maximum output of 3.2V at a distance around 3-4 inches. The maximum useful distance is around 2-3 feet with a voltage around 1.1V. It has a similar distance-voltage relationship as the other sharp sensor. Two of these are used along with the longer range IR Sensor to do obstacle avoidance.

Bump Sensor

There are a total of 5 bump sensors on Buzz. The sensor consists of a simple tactile switch that is assembled in a pull-up resistor type circuit with a voltage divider. The output is connected to the A/D converter on the Xmega128A1. The figure below shows how all the switches are connected.

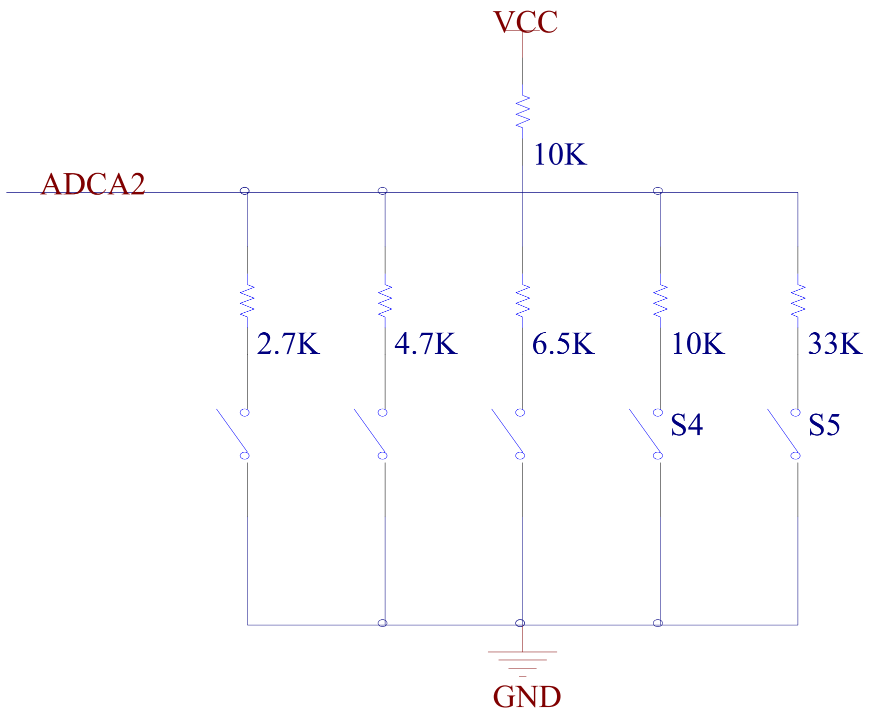


Figure 2. Bump Sensor Circuitry

What this circuit is output different voltage levels depending on which switch are pressed. When no switch is pressed, ADCA2 is held high because VCC goes directly to the input pin. This is possible because the pins are high impedance, so there is no current flow. When a switch is pressed, it completes the circuit and current flows through the corresponding resistor and switch and the circuit becomes a voltage divider and the ADC reads that voltage. Since all the resistors are different on each switch, it will yield a different voltage divider. Thus, by using only one ADC, 5 different bump switches can be detected.

SE-10 Pyroelectric Sensor

This is a motion detecting sensor capable of detecting 'moving heat'. During power up, the sensor needs to remain still so it can adjust to the settings. Then when it detects movement, the alarm pin goes low. The alarm pin is an open collector so it needs a pull-up resistor circuitry. The alarm pin is connected to PORTF on the microcontroller.

Special Sensor CMUCAM1

One behavior that I want Buzz to show is the ability to follow the user and engage them. I chose the CMUCAM to perform this function. The CMUCAM is a small camera system consisting of a SX28 microcontroller and an OV6620 Omnivision CMOS camera. It can communicate with a computer via a RS232 serial port or a microcontroller using TTL serial communications. I used a 115200 baud rate with 8 bits, 1 start and 1 stop along with no parity. I spent many weeks trying to interface the camera with the PVR board. Most was spent on experimentation and debugging. I had to connect the camera a computer to control it, and then I connected the PVR board to a computer to see how the PVR board reacts. Using this methodical process I eventually configured the PVR board to work with the CMUcam1.

The camera has an 80x143 resolution and can display the output to a computer. It is useful to be able to see what the robot sees so you can adjust parameters accordingly. There are things that you have to adjust for such as light levels and white balance. How the camera works is that you give it a range of RGB values and it will search for that color. Once it finds that color, it will send data through serial giving the offset pixels from center. This way, the Xmega128A1 can adjust accordingly to keep the color in the center of the frame. An example command(Track Color) is shown below.

```
TC [Rmin Rmax Gmin Gmax Bmin Bmax]\r
```

You give it the RGB parameters that you want and it will return:

```
M mx my x1 y1 x2 y2 pixels confidence\r
```

The x and y will give the position of the centroid of the color blob and the pixel confidence tells you how many pixels are counted versus all pixels. The main benefit of the CMUcam is the easy command set allows for good color tracking. I had to set the CMUcam to "raw mode" so that it would send out bytes instead of ASCII, this allowed me to use the data directly without extracting the data from the ASCII string. I did run into a problem with the packets arriving out of order to the PVR board in "raw mode". I never could fix this problem, but adjusted for it so that it still worked.

For the purposes of demonstration, I confined the color red to only the user. So Buzz searches until it finds a red target, it will identify this red with the user and perform actions to follow it. This worked well, but with different lighting yielded different results. One way to account for this was to widen the color parameter, however, this led to some false positives. The best solution was to use LEDs for the camera to track. Since the LEDs output their own light, it would lead to more consistent readings.

Behaviors

The first behavior Buzz will do is Standby Mode. Buzz will sit there waiting for the alarm to go off. When it does, it enters Wake Up mode. Buzz will run around the room randomly avoiding obstacles.

The IR distance sensors and bump sensors are used for obstacle avoidance. The long range IR sensor is the main sensor for obstacle avoidance and is used more for object detection that distance. It is pointed frontward on Buzz. When the voltage reaches a certain value Buzz will do one of two things. If the object is far enough away, it will do a slight turn. This means that it will continue going forward while turning. If an object is too close, it will stop and do a full turn, meaning it will turn in place.

This is where the two GP2Y0A21 sensors come in. They are pointed 45 degrees offset from the main IR sensor in both directions. They are mainly there to help Buzz decide which direction to turn. Since the IR sensors are fairly focused, it is hard for one sensor to have a wide field of view. The two side sensors make sure that Buzz is not turning into an object and also decide which direction to turn.

The bump sensors are the back-up sensors for obstacle avoidance. Thus it has higher priority for control of Buzz. Once the sensors are hit, it will back-up or move forward to get away from the obstacle and resume scanning with the IR. Also, the bump sensors are on the bottom platform, so this allows multi-level obstacle detection.

The second behavior that Buzz will have is interaction with a human. The pyroelectric sensor helps Buzz to 'see' a human. One thing Buzz tries to do is run away from the user. The PIR sensor is mounted on the back of Buzz and will detect if a person is behind Buzz. If there is, Buzz will change course and try to evade the human. Buzz will then follow the user after they have caught it. The CMUcam gives Buzz the ability to follow a human by following a defined color such as red. Buzz will turn around slowly until it sees the color red. Once it has it in the center it will move forward towards the target until a set distance.

Conclusion

While this project and class required a lot of work, it was extremely satisfying in the end. I've always enjoyed building projects, but they were always electrically based. I've never been comfortable working with motors or servos but I did want to learn. This is where the adage, 'the best way to learn is by doing it' holds true. I have learned so much practical and useful knowledge in this class. I set out to build a mobile autonomous robot and Buzz satisfied the key objectives of that goal. Although I did not accomplish everything I wanted to with Buzz, I feel it works very well. I'm surprised how much I learned about mechanical designs mainly by running into mechanical problems and having to find solutions. This class takes you through all the phases of building a complete product. Every step and component affects the final product. There are limitless options each with their own benefit/cost. While one component is a lot cheaper than the other, you also have to consider the extra time you must put in for it to work. Placing components may make the robot unbalanced and inadvertently create tendency to turn. These lessons come from many hours of experimentation, talking to peers, and research. I'm glad I put in the time to build a successful alarm clock robot and would recommend this class to anyone who has an interest.

Appendix

```
//NAME:              buzzrobot.c
//
//AUTHOR:            Long Vong
//
//DESCRIPTION:       The main program code for my robot Buzz.  Buzz wakes a person
//up in the morning and runs around.  The goal is to get the person to get out
//of bed in order to catch it. Then the robot will track the user to make sure
//he stays awake.

#include "global.h"
#include "PVR.h"
#include "motor.h"
#include "usart.h"
#include "clock.h"
#include <stdio.h>
#include <string.h>

#define SPEED 50
#define DEBUG 1
#define TCDELAY 1
#define color "TC 200 245 1 40 1 40\r"
#define BUFFER_LENGTH  40

int speed = SPEED;
int buffer[BUFFER_LENGTH];
int left_IR;
int right_IR;
int lcd_count = 0;
int obstacle;
int buzz = 1;

void TrackColor(char *command)
{
   memset( buffer, 0, sizeof(int)*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');
```

```
		//delay_ms(5);
	i = 0;
		//buffer[0] = USARTE0_DATA;
		for(i = 0; i < 11;
			while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));		// wait for RXCIF to be set
			buffer[i] = USARTE0_DATA;
		}
}

void lcd_update(int value)
{
		int tempi;
		float tempf;
		lcdGoto(1,0);
		lcdString("                ");
		lcdGoto(1,0);
		lcdString("Val: ");
		lcdInt(value);
		tempf = value;
		lcdGoto(1,9);
		lcdString("V:");
		tempi = 330*(tempf/4096);
		lcdInt(tempi);
		delay_ms(10);
}

void check_bump(void)
{
		int bump;
		bump = ADCA1();

		if (bump < 4000)
		{
		PORTH_OUT = TRACK005;

		//Front
		if (bump < 2600 && bump > 2400)
			{
				PORTQ_OUT = 1;
				while(bump < 2600 && bump > 2400)
					{
						motion(REVERSE);
						bump = ADCA1();
```

```
                    lcd_update(bump);
                    delay_ms(20);
               }
     }
//Left
else if (bump < 3600 && bump > 3200)
     {
          PORTQ_OUT = 1;
          while(bump < 3600 && bump > 3200)
               {
                    motion(REVERSE);
                    bump = ADCA1();
                    lcd_update(bump);
                    delay_ms(20);
               }
     }

//Right
else if (bump < 3000 && bump > 2700)
     {
          PORTQ_OUT = 1;
          while(bump < 3000 && bump > 2700)
               {
                    motion(REVERSE);
                    bump = ADCA1();
                    lcd_update(bump);
                    delay_ms(20);
               }
     }

//BackLeft
else if (bump < 2300 && bump > 2100)
     {
          PORTQ_OUT = 1;
          while(bump < 2300 && bump > 2100)
               {
                    motion(FORWARD);
                    bump = ADCA1();
                    lcd_update(bump);
                    delay_ms(20);
               }
     }
//BackRight
```

```
	else if (bump < 1700 && bump > 1400)
		{
			PORTQ_OUT = 1;
			while(bump < 1700 && bump > 1400)
				{
					motion(FORWARD);
					bump = ADCA1();
					lcd_update(bump);
					delay_ms(20);
				}
		}
	}
	PORTH_OUT = CLEAR;
}

void IRsensors( void )
{
	int x;
	PORTQ_OUT ^= 0x01;
	obstacle = ADCA2();					//Read in analog value from front IR
	left_IR = ADCA7();					//Read in analog value from left IR
	right_IR = ADCA6();					//Read in analog value from right IR
	check_bump();
	x = speed;
	if (speed < 20)
	{
		speed = 20;
	}

	else if (speed > 60)
	{
		speed = 60;
	}

	//If obstacle is true
	if (obstacle > 2500 || left_IR > 3500 || right_IR > 3500)
	{
		//PORTQ_OUT = 0x01;					//Turn on DEBUG LED
		PORTH_OUT = TRACK001;
		motion(STOP);						//Stop the motors
		left_IR = ADCA7();
		delay_ms(5);
		right_IR = ADCA6();
```

```
delay_ms(5);

//turn left
if (right_IR < left_IR)
{
	LMotorC0(speed/2);
	RMotorC1(speed/2);
	motion(LEFT);
	while(obstacle > 2500 || left_IR > 3500 || right_IR > 3500)
	{
		check_bump();
		obstacle = ADCA2();
		left_IR = ADCA7();
		right_IR = ADCA6();
		delay_ms(5);
		if (lcd_count > 10)
		{
			lcd_update(obstacle);
			lcd_count = 0;
		}
		lcd_count++;
	}
}
//turn right
else
{
	LMotorC0(speed/2);
	RMotorC1(speed/2);
	motion(RIGHT);
	while(obstacle > 2500 || left_IR > 3500 || right_IR > 3500)
	{
		check_bump();
		obstacle = ADCA2();
		left_IR = ADCA7();
		right_IR = ADCA6();
		delay_ms(5);
		if (lcd_count > 10)
		{
			lcd_update(obstacle);
			lcd_count = 0;
		}
		lcd_count++;
	}
```

```
                }
        }
        speed = x;
        LMotorC0(speed);
        RMotorC1(speed);
        PORTH_OUT = CLEAR;
}

void WakeUp(void)
{
        //Initialize Local Variables
        int alarm                = 0;
        int human                = 0;
        int tempspeed  = 0;
        int count                = 0;
        int test           = 0;
        int holdcount  = 0;
        int start                = 0;

        //Read in Input from Alarm clock
        PORTJ_OUT = 0x00;
        delay_ms(5);
        start = PORTJ_IN;
        start &= 0x02;
        if (start == 0x02)
        {
                lcdData(0x00);
                alarm = 0x01;
        }
        //display_time();
        if (alarm == 0x01)
        {
                speed = SPEED;
                LMotorC0(speed);
                RMotorC1(speed);
                lcdGoto(0,0);
                lcdString("WAKE UP!");
                PORTH_OUT = TRACK003;
                delay_ms(3000);
                while (alarm == 0x01)
                {
                        //display_time();
                        motion(FORWARD);
```

```
				check_bump();
				IRsensors();
				test = PORTJ_IN;
				test &= 0x40;
				if(test == 0x00)
				{
					holdcount++;
					if (holdcount > 5)
					{
						PORTJ_OUT = 0x01;
						alarm = 0;
					}

				}
			}

			buzz = 0;
			motion(STOP);
			PORTH_OUT = CLEAR;
			PORTJ_OUT = 0x01;
		}
		else
		{
			motion(STOP);
		}
}


void main(void)
{

		///////////////////////////////////
		//Initialize Robot
		///////////////////////////////////
		xmegaInit();							//setup XMega
		delayInit();							//setup delay functions
		ADCAInit();								//setup PORTA analong readings
		lcdInit();								//setup LCD on PORTK
		motorInit();
		USARTInit();
		//clockInit();

		///////////////////////////////////
```

```
	//Initialize Variables
	/////////////////////////////////
	static char *temp;
	int countDown = 5;

	/////////////////////////////////
	//Start Up Countdown Delay
	/////////////////////////////////
	PORTH_OUT = TRACK007;
	lcdGoto(0,0);						//move LCD cursor to the second line (Line
1) of LCD
	lcdString("Starting Buzz");
	for(int i = 0; i < countDown; i ++)		//countDown second countdown startup delay
	{
		lcdGoto(1,0);
		lcdInt(countDown-i);
		delay_ms(1000);
	}
	lcdData(0x00);							//Clear LCD
	delay_ms(1000);
	PORTQ_OUT = 0x00;
	PORTH_OUT = CLEAR;

	/////////////////////////////////
	//Standby Mode - Wait for Alarm
	/////////////////////////////////
	lcdString("Standby");
	while(buzz){WakeUp();}


	/////////////////////////////////
	//Initialize CMUCam1
	/////////////////////////////////
	USARTsend("RS\r");				//Reset the CMUcam1
	temp = USARTreceive();			//Receive ACK
	delay_ms(1000);
	USARTsend("\r");				//Reset the CMUcam1
	temp = USARTreceive();			//Receive ACK
	lcdString("Reset");
	delay_ms(1000);

	delay_ms(5);
	USARTsend("PM 1\r");		// Set CMUcam to Poll Mode
	temp = USARTreceive();				//Receive ACK
	delay_ms(5);
	lcdData(0x00);					//Clear LCD
	delay_ms(100);

	count2 = 0;	// ******* //
	//TrackColor("TC 170 252 1 61 1 61\r");

	//////////////////////////////////
	//Initialize Tracking Mode
	//////////////////////////////////
	speed = 15;
	LMotorC0(speed);
	RMotorC1(speed);
	PORTH_OUT = TRACK006;

	USARTsend("RM 3\r");
	delay_ms(5);
	lcdString("Tracking Mode");
	for(int i = 0; i < countDown; i ++)				//countDown second countdown
startup delay
	{
		lcdGoto(1,0);
		lcdInt(countDown-i);
		delay_ms(1000);
	}
	lcdData(0x00);						//Clead LCD
	lcdString("Tracking Mode");
	PORTH_OUT = CLEAR;
	delay_ms(400);
	PORTH_OUT = TRACK004;


	//////////////////////////////////
	//Main Loop - Tracking Mode
	//////////////////////////////////
	while(1)
	{
		PORTQ_OUT ^= 0x01;				//Toggle Debug LED
		lcdData(0x00);					//Clear LCD
		lcdGoto(1,0);
		TrackColor(color);
		lcdString("MY: ");
```

```
lcdInt(MY);
lcdString(" MX: ");
lcdInt(MX);

if(CONFIDENCE > 50 )
{
motion(STOP);
lcdGoto(0,0);
lcdString("HIT");

        if( ((MX > 30 ) && ( MX < 50 )) && (CONFIDENCE > 50))
        {
                speed = 30;
                LMotorC0(speed);
                RMotorC1(speed);

                while( ((MX > 30 ) && ( MX < 50 )) && (CONFIDENCE > 50))
                {

                        //High MY means that it is close to the ground
                        //Low MY means that high off the ground

                        if ((MY > 30))
                        {
                                motion(FORWARD);
                                check_bump();
                                IRsensors();
                                delay_ms(5);
                        }
                        else if (MY < 10)
                        {
                                motion(REVERSE);
                                check_bump();
                                delay_ms(5);
                        }


                        else if((MY < 30) && (MY > 10))
                        {
                                PORTH_OUT = TRACK002;
                                motion(STOP);
                                check_bump();
                                delay_ms(5);
                        PORTH_OUT = CLEAR;
                    }

                    TrackColor(color);
                    delay_ms(4);
                }
            }

            else
            {
                speed = 12;
                LMotorC0(speed);
                RMotorC1(speed);
                TrackColor(color);
                delay_ms(1);
                if ((MX < 30) && (CONFIDENCE > 20) && (MX != 0) )
                {
                    while((MX < 30) && (CONFIDENCE > 20) && (MX != 0) )
                    {
                        //delay_ms(TCDELAY);
                        motion(RIGHT);
                        IRsensors();
                        TrackColor(color);
                    }
                    motion(STOP);
                }


                else if ((MX > 50) && (CONFIDENCE > 20) && (MX != 0))
                {
                    while((MX > 50) && (CONFIDENCE > 20) && (MX != 0))
                    {
                        //delay_ms(TCDELAY);
                        motion(LEFT);
                        IRsensors();
                        TrackColor(color);
                    }
                    motion(STOP);
                }


            }
        }
```

```
                else if (MX == 0 || CONFIDENCE == 0)
                        {
                        speed = 12;
                        LMotorC0(speed);
                        RMotorC1(speed);
                        while(MX == 0 && MY == 0)
                        {
                                motion(LEFT);
                                lcdGoto(0,0);
                                lcdString("NFOUND");
                                lcdGoto(1,0);
                                lcdString("MY: ");
                                lcdInt(MY);
                                lcdString(" MX: ");
                                lcdInt(MX);
                                TrackColor(color);
                        }
                }
        }

}
```

```
/*  clock.c

        Long Vong

        A simple alarm clock circuit running on an ATMEGA32
*/

#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define ZERO    0b11000000
#define ONE             0b11111001
#define TWO     0b10100100
#define THREE           0b10110000
#define FOUR    0b10011001
#define FIVE    0b10010010
#define SIX             0b10000010
#define SEVEN           0b11111000
#define EIGHT           0b10000000
#define NINE    0b10010000

#define DIG1    0b00010001
#define DIG2    0b00010010
#define DIG3    0b00010100
#define DIG4    0b00011000


#define ALARM_HOUR                      12
#define ALARM_MINUTE    10

#define TRIGGER 0

//GLOBAL VARIABLES
static int number[10] ={ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE};
int alarm_hour;
int alarm_minute;
int temp_hour;
int temp_minute;
int hour;
int minute;
int second;
int thousand, hundred, ten, one;
int set = 0;


//PORTC PIN7 PIN6 is output
//PORTD PIN1 is switch
//
void display_time( void )
{


        PORTA = number[thousand];
        PORTC = DIG4;
        _delay_ms(2);


        PORTA = number[hundred];
        PORTC = DIG3;
        _delay_ms(2);

        PORTA = number[ten];
        PORTC = DIG2;
        _delay_ms(2);

        PORTA = number[one];
        PORTC = DIG1;
        _delay_ms(2);

        PORTC = 0;
}



void update_time(void)
{

                if (second >= 59)
                {
                        second = 0;
                        minute++;
                }
                else if (minute >= 60)
                {
                        minute = 0;
                        hour++;
                }
                else if (hour >= 25)
                {
                        hour = 1;
                }
                ten = minute/10;
                one = minute%10;
                thousand = hour/10;
                hundred = hour%10;

}

/*
timer 1 counter
Sets the timer to update counter every second( 15625 at F_CPU/64)
*/
void timer1Init(void)
{
        // Initialize Timer 1 to CTC mode, rising edge triggered, clock prescaler at 64

        TCCR1B = ( 1 << WGM12 ) | ( 1 << CS11 ) | ( 1 << CS10 );
        OCR1A = 15625;                                  // Set compare match value to 15625 counts
        TIMSK |= ( 1 << OCIE1A );               // Enable Timer 1A Output Compare Match Interrupt
                                                        // Start Timer 1 at F_CPU/64.  15625
counts at F_CPU = 1 MHz is 1 sec.
}

void InterruptInit(void)
{
        MCUCR = ( 1 << ISC11 ) | ( 1 << ISC10 ) | ( 1 << ISC01 ) | ( 1 << ISC00 );
        GICR = ( 1 << INT1 ) | ( 1 << INT0 );
}

void setalarm(void)
{
        int test = 0;
        int debounce_count = 0;
        temp_hour = hour;
        temp_minute = minute;
```

```
hour = alarm_hour;
minute = alarm_minute;



test = PIND;
test &= 0x02;
while(test == 00)
{
        update_time();
        display_time();
        test = PIND;
        test &= 0x02;
}

while(set == 1)
{
        update_time();
        display_time();
        test = PIND;
        test &= 0x02;
        if (test == 0x00)
        {
                debounce_count++;
                if (debounce_count > 10)
                {
                        set = 0;
                        debounce_count = 0;
                }
        }

        else
        {
                debounce_count = 0;
        }
}

alarm_hour = hour;
alarm_minute = minute;

hour = temp_hour;
minute = temp_minute;
```

```
}

void alarm(void)
{
        int timer = 0;
        int i = 0;
        int t = 0;
        int temp;
        int alarm = 1;

        PORTB |= 0xff;
        PORTC |= 0x40;
        PORTD |= 0x80;

        alarm = 0x01;
        _delay_ms(100);
        temp = PINC;
        temp &= 0x80;

        while( alarm == 0x01)
        {

                if(i == 25)
                {
                        PORTB ^= 0xff;          //Toggle LEDs
                        i = 0;
                        update_time();
                }
                if(t == 150)
                {
                        display_time();
                        t = 0;
                }

                PORTD |= 0x81;                  //b.0 = 1
                _delay_ms(2);
                PORTD &= 0xfe;                  //b.0 = 0
                _delay_ms(2);

                timer++;
                i++;
                t++;
                temp = PINC;
```

```
		temp &= 0x80;
		if (temp == 0x80)
		{
			alarm = 0x00;
			PORTB = 0x00;
			PORTD = 0x00;
		}

	}

}




void main ( void )
{

	DDRA |= 0xff; //Set porta as an output for 7 SEG DISPLAY
	DDRB |= 0xff; //Set portb as an output for LEDS
	DDRC |= 0x7f; //Set portc as an output for DIGIT SELECT/LED
	DDRD |= 0xf1; //Set portd as an output to LED and input to interrupt

	//PORTC PIN 8 is an input
	//PORTD PIN 8 is an output

	int count=0;
	hour = 12;
	minute = 0;
	InterruptInit();		//Initialize External Interrupts
	timer1Init();		//Initialize 16bit Timer 1
	sei();				//enable interrupts
	int test;
	alarm_hour = ALARM_HOUR;
	alarm_minute = ALARM_MINUTE;
	while(1)
	{
		PORTD = 0x00;
		PORTC &= 0xbf;
		update_time();
		display_time();
		test = PIND;
		test &= 0x02;
```

```
                if (test == 0x00)
                {
                        count++;
                        if ( count > 10 )
                        {
                                set = 1;
                                setalarm();
                                count = 0;
                        }

                }
                else
                {
                        count = 0;
                }
                if( hour == alarm_hour && minute == alarm_minute )
                {
                                alarm();

                }
        }
}


ISR( TIMER1_COMPA_vect )
{
        second++;
}

ISR( INT0_vect )
{
        minute++;
}

ISR( INT1_vect )
{
        hour++;
}
```