

CHROMATIC TRAILBLAZER

FINAL REPORT

9th December, 2008

Intelligent Machine Design Lab (EEL 5666)

Fall 2008

University of Florida

Department of Electrical & Computer Engineering

Advisors:

Dr. A. Antonio Arroyo

Dr. Eric M. Schwartz

T.A.:

Mike Pridgen

Thomas Vermeer

Submitted By:

Vivek Anand

ECE Department

University of Florida

Table of Contents

| | |
|-------------------|----|
| Abstract | 3 |
| Executive Summary | 4 |
| Introduction | 5 |
| Integrated System | 7 |
| Mobile Platform | 9 |
| Actuation System | 10 |
| Sensors | 11 |
| Behaviors | 15 |
| References | 19 |
| Conclusion | 20 |
| Appendices | 22 |

Abstract

Chromatic Trailblazer is being done as a part of the curriculum of Intelligent Machine Design Lab. It is an integration of two aspects on a single platform – Machine Intelligence and Image Processing. Its purpose is to track moving objects and perform the task of obstacle avoidance. The two programs run parallel to each other with request and grant based approach. The robot takes input from the user for the object to be tracked.

Executive Summary

Chromatic Trailblazer is an autonomous robot which is capable of following moving objects. It amalgamates different branches of engineering together. The robot is suitable to work in any kind of environment and works in collaboration with a remote computer. It consists of a wireless camera which is capable of transmitting video back to a remote computer. The remote computer has a wireless receiver and an audio video usb adapter for converting the analog video received from the camera to digital form.

This video becomes the input to the object tracking program which is running on the remote computer. The program then displays the video on the computer screen and asks the input from the user. The user selects the object to be tracked and thus the object tracking algorithm starts.

On the other hand, we have the microcontroller running on the robot and performing obstacle avoidance behavior. For the object tracking behavior it transmits command to the remote computer for providing the location of the object. As it receives the location of the object then it tracks as per the data received.

The connection between the object tracking program and the microcontroller is through a bluetooth network. The bluetooth reading and writing programs on the remote computer is done through c code for serial programs which is running along with the object tracking program. On the microcontroller side, it is connected to the RS 232 ports available on the microcontroller.

This is what the Chromatic Trailblazer does.

Introduction

Robots are technological innovations that are intended to ease the human work. Its main purpose is assisting humans in performing duties which humans can't perform because of the inaccessibility of the region or risks involved in the particular work. Most of the robots manufactured nowadays performs the above basic task and thus find vast applications in defense and space industry. In the recent times, there has also been a trend of using robots to ease the jobs of humans. Maybe someday, we can just doze off in the bed and let the robot perform all our duties.

The Chromatic Trailblazer is a robot which intends to follow chromatic objects. In this robot, I will try to incorporate functions which can help the robot to detect moving objects on the ground. The robot is an autonomous robot which can detect the shape and color of a particular object and track it. The object to be tracked is selected by the user.

The approach is in this manner. First all the ports on the microcontroller board get initialized. Then the robot tells the user on the remote computer to select the object which it wants to track. As soon as the object is selected on the screen, the location of the object is transferred back to the robot via the bluetooth network and the robot performs object tracking.



Fig: Full Robot Image

Integrated System

Chromatic Trailblazer is an autonomous which can detect and follow the motion of a soccer ball. The robot uses an Atmega 128 Board for the processing to be done on the robot itself. It has a wireless camera on the robot which captures the images of the surrounding and transfers it to the laptop. There is a wireless receiver connected to the laptop for receiving images.

The laptop processes the images using Open CV codes and finds out the location of the object in the image. Then it decides the direction of motion and transfers the commands to the robot using a bluetooth device. The robot has a serial port bluetooth receiver for receiving these commands. These commands are then processed by the microprocessor and conveyed to the motors for following the direction.

The robot is also equipped with the system of obstacle avoidance. It uses SONAR for detecting obstacles and has an interrupt method for preventing the robot to collide with obstacles. The robot is also provided with bump switches to bounce back the robot in case it collides with a wall. The main objective of the robot is to trace the object and maintain a fixed distance from the object.

The working principle of the robot is described below – the robot when started first tries to find the object. If it does not find the object, then the robot tries to start rotating in the right hand direction until it finds the position of the object. It rotates until the object is in the center of image and then the robot starts to move towards the object. If it loses track the object, then it predicts the path of object on the basis of previous images obtained from the wireless camera.

The motor can receive two types of commands – one from the obstacle avoidance algorithm and the image processing algorithm. The priority of the commands received from obstacle avoidance algorithm hold more priority in comparison to the commands from image processing algorithm.

A schematic diagram of Chromatic Trailblazer is shown on next page:

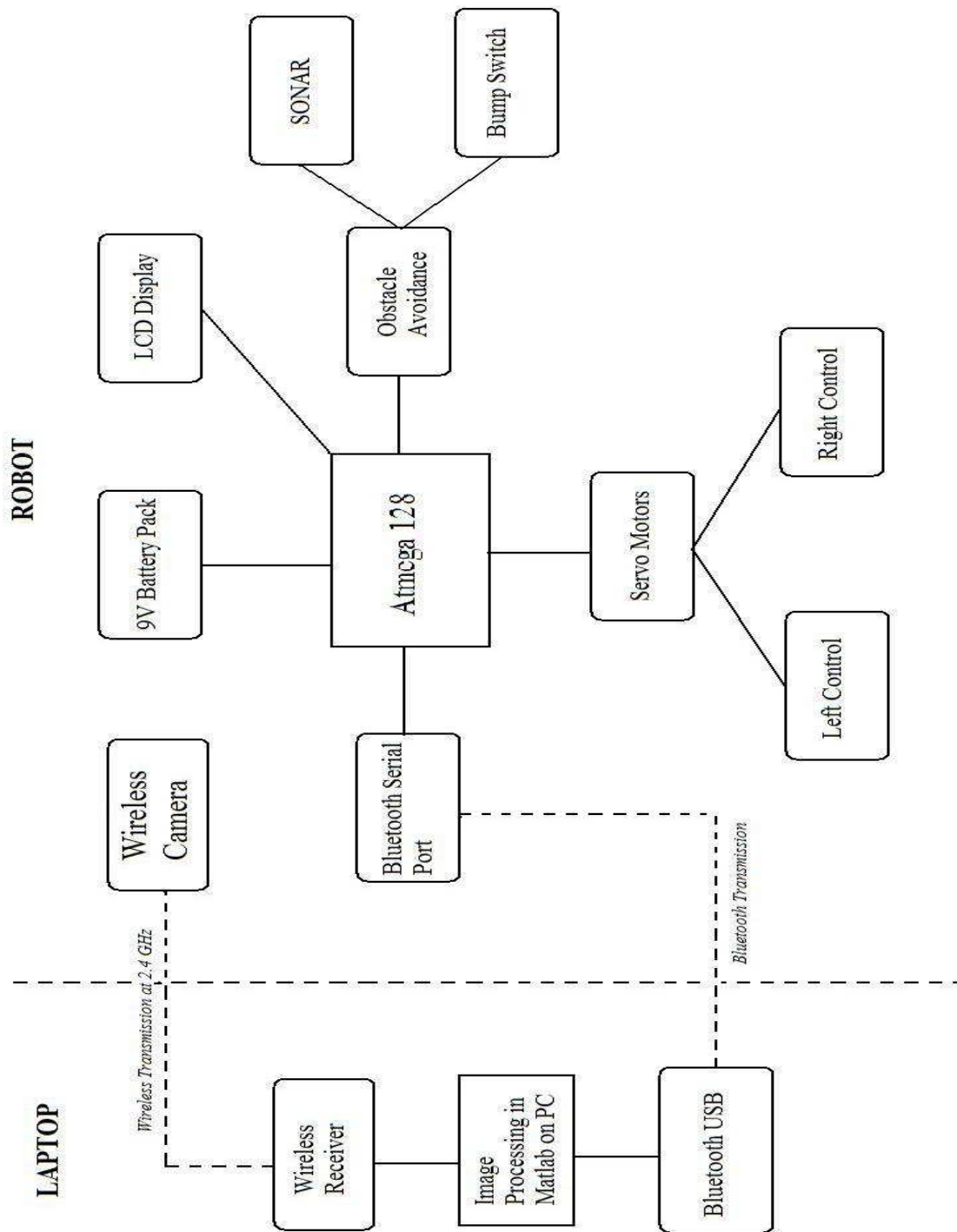


Fig: A Schematic Block Diagram of Chromatic Trailblazer

Mobile Platform

The Chromatic Trailblazer uses a rectangular base as its platform. It consists of 2 wheels on the rear side and one freewheel on the front side. The wireless camera is mounted on the front side of the robot.

The mobile platform also consists of the Atmega 128 Board with which bluetooth device, lcd, sonars, bump switches and servo motors are connected. An approximate diagram of the top view of the robot is shown below.



Fig: Top view of the Chromatic Trailblazer

Actuation System

Chromatic Trailblazer uses a differential drive system. In a differential drive system the right-wheel and the left wheel are controlled by 2 different set of motors independent of each other. I will also be using a castor wheel in the front to balance the weight of wireless camera as the platform is a rectangular one.

The motors used in Chromatic Trailblazer are DC Motors because of their robustness and speed. The servo motors are inherently slow and not suitable for high loads. The motor driver circuit is the L298 Motor Driver capable of running on 12-24 volts. The setup also required an optoisolator set for keeping the microprocessor board and the driver board optically isolated.



Fig 1: DC Motor



Fig 2: L298 Motor Driver

Sensors

Sensors mainly help the robot in identifying its surrounding. The sensors used in Chromatic Trailblazer serve two important purpose. They are:

- Obstacle Avoidance
- Image Sensing

Obstacle Avoidance

The robot is equipped with the power of avoiding obstacles in its path. To achieve this target, it uses Ultrasonic Sensors.

Ultrasonic Sensors or simply SONARs sent out an audio signal and receives the reflected sound signal. In this way, it determines the distance and location of the obstacles. There are two SONARs used in the robot, both on the extreme ends in the front side.

The SONARs used are SRF-05 Ultrasonic SONARs from Devantech. The specification of SONAR is shown below:

| Specifications | |
|----------------|--|
| Frequency | 40kHz |
| Max Range | 4 meters |
| Min Range | 3 centimeters |
| Input Trigger | 10uSec minimum, TTL level pulse |
| Echo Pulse | Positive TTL level signal, proportional to range |



Fig: SONAR (SRF-05)

Image Sensing (SPECIAL SENSOR)

Image Sensing is done by the wireless camera mounted on the top of the robot. The choice of the wireless camera was because of the fact that it can run without being dependent on the on-board power supply. It has its own set of cells to perform the operation. The camera comes along with a wireless receiver which can be connected to the laptop to receive the images being taken from the wireless camera. Website link is as follows http://www.servocity.com/html/2_4ghz_color_video_system.html. The type of camera and the specifications are shown below.

Camera Specifications

Output Level: 90db microvolts / meter @ 3 meter
Transmitting Frequency: 4CH 2,400 to 2,483 MHz
Modulation: FM
Antenna: 50 ohm SMA
Receiving Sensitivity: -85dbm
Video Input Level: 1.0 VP-P @ 75 ohm
Audio Input Level: 1.0 VP-P @ 600 ohm
Image Sensor: 1/3" C-MOS sensor
Number of Pixels: 380 TV lines
Scanning System: 525 lines, 60 fields
Sync System: Internal Sync
Minimum Illumination: 1.5 Lux/F1.5 & IR LED on:
0 Lux
SN Ratio: More than 45db
Gamma Characteristics: 0.45
Electronic Shutter Speed: 1/60 to 1,500 sec.
Lens: 78 degree wide angle lens
Microphone: Condenser
Lithium Batter: 500mah
Charging Time: 2 hours
Working Time: 5 hours
Operating Temperature: 14 degree F to 122 degree F
Dimensions: 87x43x90 mm
Weight: 235g (5.8 oz.)



Fig: Wireless Camera

Wireless Receiver Specifications

Receiving Frequency: 4CH 2,400 to 2,483 MHz

Demodulation: FM

Antenna: 50 ohm SMA

Receiving Sensitivity: -85dbm

Video Output: 1 VP-P @ 75 ohm

Audio Output: 1 VP-P @ 600 ohm

Power Supply: DC 8V

Current Consumption: 180ma

Dimensions: 115x20.5x99 mm

Weight: 248



Fig: Wireless Receiver

The output from the wireless receiver is a Composite Video which cannot be processed on computer. A device known as *Audio Video USB Adapter* (<http://www.usbvideoadapter.com/>) was used for this purpose. It takes its input either a composite video or audio-video and converts them to digitized form which can be seen on the computer and can be processed.



Fig: Audio Video USB Adapter

The image processing is done with the help of Open CV libraries. Open CV are libraries developed by Intel for the purpose of Computer Vision. The libraries are written in C language. The best part of this program is that it takes the input from the user on deciding which object to track rather than depending on the optical flow. The process mainly involves 3 steps.

1. Capture from Camera – This uses open cv functions such as `cvcapturefromdevice()`.
2. Object Tracking – The tracking algorithm is based on background subtraction.
3. Location – The video frame is of 320X240 pixels size. This has been considered as a grid and the position of the object in this virtual grid is calculated and transferred via Bluetooth. The device always connects to the COM port of the computer and simple C code has been written for this purpose.

Another advantage of using Open CV libraries is that its processing speed is much higher compared to Matlab. For a real time video, the processing time per frame is approximately 1 second while in Open CV its around 0.5 milliseconds.

The processed commands are then fed back to the robot via the Bluetooth network. A Bluetooth USB is connected to the laptop to transmit commands to the robot. The robot is equipped with a Bluetooth serial port for receiving the images. The Bluetooth serial port can be connected to anyone of the serial ports available on the Atmega 128 board. The Bluetooth USB and the Bluetooth serial port (http://www.sparkfun.com/commerce/product_info.php?products_id=8332) which is being used are shown below:-

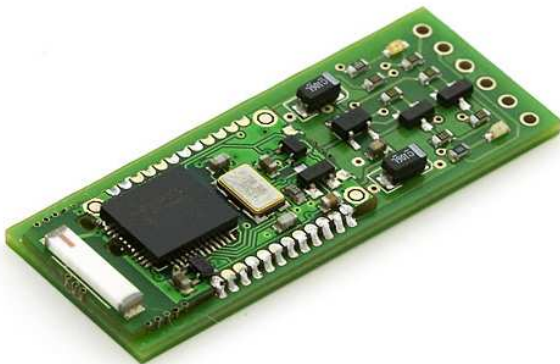


Fig 2: Bluetooth Serial Port

Behaviors

Machine Intelligence Behavior

Machine Intelligence behavior consists of 2 parts in case of Chromatic Trailblazer. In the first part we will talk about its obstacle avoidance behavior. The machine intelligence behavior is based upon Fuzzy logic. It decides its motion based on the value of the ultrasonic sensors which have been termed over here in the table as Sonar Values L for the Left sonar and Sonar Values R for the right sonar. The table then generates a set of Boolean expressions which decide the movement of the robot. Here HR stands for Hard Right, HL for Hard Left, SR for Soft Right, SL for Soft Left, Slow for Straight at slow speed, Medium is straight at Medium speed and Fast for moving straight at fast speed.

| Sonar Values L | 0-20 | 20-50 | 50-100 | 100-150 | Above 150 |
|----------------|------|-------|--------|---------|-----------|
| Sonar Values R | | | | | |
| 0-20 | HR | HL | HL | HL | HL |
| 20-50 | HR | SR | SL | SL | SL |
| 50-100 | HR | SR | Slow | Slow | Slow |
| 100-150 | HR | SR | Slow | Medium | Medium |
| Above 150 | HR | SR | Slow | Medium | Fast |

The other part where Machine Intelligence was required was the decision based on the values of the data received by the bluetooth and the sonar values. The logic was that the program will take the input from the bluetooth which can be 'l' for left, 'r' for right and 'c' for telling the location of the object. At the same time it also keeps on checking the sonar values so that it does not reaches too close to the object and does not hit objects. When it gets the character 'l' it moves

left slightly and on 'r' it moves right. On receiving the character 'c' it moves straight. When it sees that either of its sensor values are falling below 20 then it tries to get away from the obstacle by either turning left or right depending on which sensor value is smaller. The program regularly sends a request to the obstacle avoidance algorithm for the location of the object.

Object Tracking Behavior

Object tracking algorithm has been written in C language using the Intel Open CV libraries. The libraries were quite helpful and made the program easy. The first part of the program consists of collecting data from the wireless camera. The inbuilt functions can be found in the highgui library. Then the object to be tracked is taken as input from the user using the mouse helper functions. This selected object is then backtracked on each frame of the real time video and the location of the object is stored in a structure. The structure consists of the location and the size of the object. These variables are used to determine the location of the object. The size of the frame is 680 X 480. So the location is calculated as follows:

If $(x_{cod} + (width/2)) < 680$ then $x_{loc} = x_{cod} + (width/2)$

else $x_{loc} = x_{cod} - (width/2)$

Now, the decision on whether the object is on the right, left or center is taken on the basis of this x_{cod} . If x_{cod} is less than 250 then it's left, if its greater than 300 then it is right otherwise it's in center.

Serial Port Behavior

Serial port behavior is initializing the com port on which the bluetooth is connected and writing commands on bluetooth to be send and receiving commands which are sent to the bluetooth device. The program is written in C language and has been put together with the Object tracking algorithm to form a complete project. The program uses the following functions to make it working:-

1. CreateFile function for initializing the COM port 5 (where bluetooth has been connected).
2. WriteFile for sending the data
3. ReadFile for reading the data.

Note: - Never trust any online code. One should always cross check with the documentation available and do the calculations before initializing. Open CV was quite interesting to learn but it is tough to initialize and install. These link can help anyone who wants to use Open CV libraries <http://opencv.willowgarage.com/wiki/VisualC%2B%2B>

REFERENCES

1. Documentation of Atmega 128
2. C Programming for AVR
3. 6.270 Robot Builder's Guide
4. Learning Open CV – Gary and Adrain
5. Digital Image Processing – Gonzalez and Woods
6. http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_BasicFuncs.htm

Conclusion

Chromatic Trailblazer was a great learning experience for me. This provided me a platform where I was able to amalgamate my previous robotics knowledge with the Machine Intelligence concepts and Image Processing concepts. It helped me learn more about microcontrollers and interrupt handling.

I enjoyed the class at each stage. Sometime it was so stressful handling the pressure when a small error takes hours to debug. But at last I loved it. I enjoyed every moment of it.

Chromatic Trailblazer was able to do part of what I had planned to do at the start of the semester. It can easily detect objects and track it based on the shaped and the color of the object. The biggest success for me was displaying the video on the laptop and able to command it to select which object to track. Though it took a lot of time, for learning Open CV in just 2 weeks flat and writing the code but at last I enjoyed the success.

The project runs in 2 modes. For some particular instance of time, it performs the obstacle avoidance character. In the second mode, it performs the job of object tracking by asking the computer for the location of the object.

The other hurdle was that of the bluetooth connection. I had no clue at the start of semester how to do that but gradually I was able to figure it out from the net and write my own codes for writing and reading data from COM ports.

Chromatic Trailblazer won't have been at this stage if I would have not been helped by my friends and classmates. The TAs need special credit for their outstanding support and help. Their codes helped a lot in understanding the concept of AVR programming. I would specially like to thank the following people:-

1. .Dr Arroyo – His knowledge and experience is so helpful. I wish to work with him and learn more about his experience.
2. Dr Schwartz – He motivated and kept our spirits always high. I wish to work for him and gain his insight about robotics.

3. Mike Pridgen – He helped me a lot with the RS 232 ports and the timer
4. Thomas Vermeer – He was always there motivating and providing great ideas
5. Ryan Chilton – He helped me in correcting the bluetooth set up.
6. Premchand Krishna Prasad – He was a friend who was always there supporting me and motivating me in my efforts, day in and day out
7. Subrat Nayak – A friend who helped me at each stage whether it was a small or a big problem with my robot and with programming.
8. Ron – For letting me use his workshop.
9. Alok Whig – My friend who helped me with C programming.
10. Taruja Borker – My project partner for Obstacle Detection Algorithm.

At the start of the semester I had set 4 targets for my project. They were:-

1. Locate the ball and move towards that location.
2. Maintain a fixed distance from the ball.
3. Predict the path of ball if the ball is not being seen.
4. Provide horizontal and vertical movement to the wireless camera.

I am happy that I was able to meet 3 out of 4 of the targets I had set for myself. I am pretty much satisfied with it but in future I would like to make it work more properly. Sometimes while performing object tracking, it gets confused with the background or the surroundings. So I would like to remove those glitches. Other thing I would like to improve is providing the camera a vertical and horizontal motion which I was not able to complete this semester.

Appendices

*****CODES FOR MICROCONTROLLER BOARD FOR BASIC PURPOSE*****

*****LCD.H*****

```
/*    Pinout (PORT C)
**    P7 - No connect (NC)
**    P6 - Enable (E)
**    P5 - Read/Write_L (R/W_L)
**    P4 - Register Select (RS)
**    P3 - Data 7 (DB7)
**    P2 - Data 6 (DB6)
**    P1 - Data 5 (DB5)
**    P0 - Data 4 (DB4)
*/
```

```
#ifndef __LCD_h__
```

```
#define __LCD_h__
```

```
#define LCD          PORTC
```

```
#define LCDDDR      DDRC
```

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

```
#include <avr/interrupt.h>
```

```
#include <avr/signal.h>
```

```
#include "LCD.h"
```

```
void ms_sleep(uint16_t ms);
```

```
SIGNAL(SIG_OUTPUT_COMPARE0);
```

```
void init_timer(void);
```

```
void lcdDelay();
```

```
void lcdDataWork(unsigned char c);
```

```
void lcdData(unsigned char c);  
void lcdCharWork(unsigned char c);  
void lcdChar(unsigned char c);  
void lcdString(unsigned char ca[]);  
void lcdInt(int value);  
void lcdGoto(int row, int col);  
void lcdInit(void);  
#endif
```

```
*****LCD.C*****
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
volatile uint16_t mscount;

//ms_sleep() - delay for specified number of milliseconds
void ms_sleep(uint16_t ms)
{
    TCNT2 = 0;
    mscount = 0;
    while(mscount != ms);
}
//millisecond counter interrupt vector
SIGNAL(SIG_OUTPUT_COMPARE2)
{
    mscount++;
}
// initialize timer 0 to generate an interrupt every millisecond.
void init_timer(void)
{
    TIFR |= _BV(OCIE2);
    TCCR2 = _BV(WGM01)|_BV(CS02)|_BV(CS00); /* CTC, prescale = 128 */
    TCNT2 = 0;
    TIMSK |= _BV(OCIE2); /* enable output compare interrupt */
    OCR2 = 12; /* match in 1 ms */
}
void lcdDelay()
{
    long int ms_count = 0;
    while (ms_count < 350)
        ms_count++;
}
void lcdDataWork(unsigned char c)
{
    c &= 0xF0; //keep data bits, clear the rest
    c |= 0x08; //set E high
    LCD = c; //write to LCD
    lcdDelay(); //delay
    c ^= 0x08; //set E low
    LCD = c; //write to LCD
    lcdDelay(); //delay
    c |= 0x08; //set E high
```

```

        LCD = c;                //write to LCD
        lcdDelay();            //delay
    }
void lcdData(unsigned char c)
{
    unsigned char cHi = c & 0xF0;    //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;    //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                //shift cLo left 4 bits
    lcdDataWork(cHi);
    lcdDataWork(cLo);
}
void lcdCharWork(unsigned char c)
{
    c &= 0xF0;                    //keep data bits, clear the rest
    c |= 0x0A;                    //set E and RS high
    LCD = c;                      //write to LCD
    lcdDelay();                  //delay
    c ^= 0x08;                   //set E low
    LCD = c;                      //write to LCD
    lcdDelay();                  //delay
    c |= 0x08;                   //set E high
    LCD = c;                      //write to LCD
    lcdDelay();                  //delay
}
void lcdChar(unsigned char c)
{
    unsigned char cHi = c & 0xF0;    //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;    //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                //shift cLo left 4 bits
    lcdCharWork(cHi);
    lcdCharWork(cLo);
}
void lcdString(unsigned char ca[])
{
    int i = 0;
    while (ca[i] != '\0')
    {
        lcdChar(ca[i++]);
    }
}
void lcdInt(int value)
{
    int temp_val;
    int x = 10000;
    int leftZeros=5;
    if (value<0)

```

```
        lcdChar('-');
while (value / x == 0)
{
    x/=10;
    leftZeros--;
}
while ((value > 0) || (leftZeros>0))
{
    temp_val = value / x;
    value -= temp_val * x;
    lcdChar(temp_val+ 0x30);
    x /= 10;
    leftZeros--;
}
while (leftZeros>0)
{
    lcdChar(0+ 0x30);
    leftZeros--;
}
return;
}
void lcdGoto(int row, int col)
{
    unsigned char pos;
    if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
    {
        pos = col;
        if (row == 1)
            pos += 0x40;
        else if (row == 2)
            pos += 0x14;
        else if (row == 3)
            pos += 0x54;
        lcdData(0x80 + pos);
    }
}

void lcdInit(void)
{
    LCDDDR = 0xFF; //set LCD port to outputs.
    lcdData(0x33); //put in 4 bit mode part 1
    lcdData(0x32); //put in 4 bit mode part 2
    lcdData(0x2C); //enable 2 line mode
    lcdData(0x0C); //turn everything on
    lcdData(0x01); //clear LCD
}
```

```
*****SONAR.H*****
```

```
/* Pinout (PORT E)
** P7
** P6
** P5 – Left Out
** P4 – Left In
** P3
** P2
** P1 – Right Out
** P0 – Right In
*/
```

```
#ifndef __Sonar_h__
#define __Sonar_h__
```

```
#define SONAR    PORTE
#define SONARDDR DDRE
#define SONARIN  PINE
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
```

```
void sendRPulse(void);
```

```
int waitRPulse(void);
```

```
int sonarRDist(void);
```

```
void sendLPulse(void);
```

```
int waitLPulse(void);
```

```
int sonarLDist(void);
```

```
int smooth(int last);
```

```
void initSonar(void);
```

```
#endif
```

```
*****SONAR.H*****
```

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

/*RIGHT SONAR*/
void sendRPulse(void)
{
    SONAR = 0x02;    //send burst
    int i = 0;
    while (i < 100)
        i++;
    SONAR = 0x00;    //end burst
}
int waitRPulse(void)
{
    int i = 0;
    while ((SONARIN & 0x01) == 0x00);
    while ((SONARIN & 0x01) == 0x01)
    {
        i++;
    }
    return i/20;
}
int sonarRDist(void)
{
    int i, temp, last=0;
    for(i=0;i<20;i++)
    {
        sendRPulse();
        temp = waitRPulse();
        last = last+temp;
    }
    last = last/20;
    return (last);
}

/*LEFT SONAR*/
void sendLPulse(void)
{
    SONAR = 0x20;    //send burst
    int i = 0;
    while (i < 100)
```

```
        i++;
        SONAR = 0x00;    //end burst
    }
int waitLPulse(void)
{
    int i = 0;
    while ((SONARIN & 0x10) == 0x00);
    while ((SONARIN & 0x10) == 0x10)
    {
        i++;
    }
    return i/20;
}
int sonarLDist(void)
{
    int i, temp, last=0;
    for(i=0; i<20; i++)
    {
        sendLPulse();
        temp = waitLPulse();
        last = last + temp;
    }

    last = last/20;

    return (last);
}

int smooth(int last)
{
    if(last<20 || last>=1200)
        last = 0;
    if(last>=20 && last<50)
        last = 40;
    if(last>=50 && last<100)
        last = 80;
    if(last >=100 && last<140)
        last = 120;
    if(last >=140)
        last = 200;

    return last;
}
```

```
void initSonar(void)
{
    SONARDDR = 0x22;
}
```

```
*****MOTOR.H*****

/*  Pinout (PORT A)
**  P3 – Right Motor
**  P2 – Right Motor
**  P1 – Left Motor
**  P0 – Left Motor
*/
/*  Pinout (PORT B)
**  P6 – Left Motor
**  P5 – Right Motor
*/

#ifndef __MOTOR_H__
#define __MOTOR_H__

void MOTOR_Init_Ports(void); // TO SET THE PORT PINS TO OUTPUT MODE

void MOTOR_Init_Timer1_PWM(void); // TO SET THE PWM SIGNAL AT 10KHZ

void MOTOR_Motor1_Set_PWMduty(int pwmduty1); //TO SET THE TIMER 1

void MOTOR_Motor2_Set_PWMduty(int pwmduty2); //TO SET THE TIMER 2

void ROBOT_Backward(void); // TO RUN ROBOT backwards

void ROBOT_Forward(void); // TO RUN ROBOT forwards

void ROBOT_Left_Centrepoint(void); // Turn sharp left about centre point

void ROBOT_Left_Leftpoint(void); // Turn slow left about left point

void ROBOT_Right_Centrepoint(void); // Turn sharp right about centre point

void ROBOT_Right_Rightpoint(void); // Turn slow right about right point

void ROBOT_Stop(void); // to make robot stop

#endif
```

```
*****MOTOR.C*****
```

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

#define MOTOR_PWM_DDR DDRB
#define MOTOR_PWM_PORT PORTB
#define MOTOR_DIR_DDR DDRA
#define MOTOR_DIR_PORT PORTA

void MOTOR_Init_Ports(void)
{
MOTOR_PWM_DDR = 0b00110000;
MOTOR_PWM_PORT = 0b00000000;
MOTOR_DIR_DDR = 0b00001111;
MOTOR_DIR_PORT = 0b00000000;
}

//FUNCTION TO SET THE PWM SIGNAL AT 10KHZ
void MOTOR_Init_Timer1_PWM(void)
{
ICR1 = 6000;
TCCR1A = 0xA8;
TCCR1B = 0x12;
TCCR1C = 0x00;
DDRB = 0xFF;
}

//FUNCTION TO Set speed of MOTOR1
void MOTOR_Motor1_Set_PWMduty(int pwmduty1)
{
OCR1A = (ICR1 / 100) * (pwmduty1);
}

//FUNCTION TO Set speed of MOTOR2
void MOTOR_Motor2_Set_PWMduty(int pwmduty2)
{
OCR1B = (ICR1 / 100) * (pwmduty2);
}

//FUNCTION TO STOP ROBOT
void ROBOT_Stop(void)
{
```

```
MOTOR_DIR_PORT = 0b00000000;  
}
```

FUNCTION TO RUN ROBOT backwards

```
void ROBOT_Backward(void)  
{  
MOTOR_DIR_PORT = 0b00000000;  
MOTOR_DIR_PORT = 0b00001010;  
}
```

//FUNCTION TO RUN ROBOT forwards

```
void ROBOT_Forward(void)  
{  
MOTOR_DIR_PORT = 0b00000000;  
ms_sleep(100);  
MOTOR_DIR_PORT = 0b00000101;  
}
```

//FUNCTION TO make ROBOT turn sharp left about centre point

```
void ROBOT_Left_Centrepoint(void)  
{  
MOTOR_DIR_PORT = 0b00000000;  
ms_sleep(100);  
MOTOR_DIR_PORT = 0b00000110;  
}
```

//FUNCTION TO make ROBOT turn slow left about left point

```
void ROBOT_Left_Leftpoint(void)  
{  
  
MOTOR_DIR_PORT = 0b00000000;  
ms_sleep(100);  
MOTOR_DIR_PORT = 0b00000010;  
}
```

//FUNCTION TO make ROBOT turn sharp right about centre point

```
void ROBOT_Right_Centrepoint(void)  
{  
  
MOTOR_DIR_PORT = 0b00000000;  
ms_sleep(100);  
MOTOR_DIR_PORT = 0b00001001;  
}
```

//FUNCTION TO make ROBOT turn slow right about right point

```
void ROBOT_Right_Rightpoint(void)
{

MOTOR_DIR_PORT = 0b00000000;
ms_sleep(100);
MOTOR_DIR_PORT = 0b00001000;
}
```

```
*****UART.H*****

#ifndef UART_H
#define UART_H
#if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler !"
#endif

#define UART_BAUD_SELECT(baudRate,xtalCpu) ((xtalCpu)/((baudRate)*161)-1)
#define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate,xtalCpu)
(((xtalCpu)/((baudRate)*81)-1)|0x8000)
#ifndef UART_RX_BUFFER_SIZE
#define UART_RX_BUFFER_SIZE 32
#endif
#ifndef UART_TX_BUFFER_SIZE
#define UART_TX_BUFFER_SIZE 32
#endif
#if ( (UART_RX_BUFFER_SIZE+UART_TX_BUFFER_SIZE) >= (RAMEND-0x60) )
#error "size of UART_RX_BUFFER_SIZE + UART_TX_BUFFER_SIZE larger than size of
SRAM"
#endif
#define UART_FRAME_ERROR    0x0800    // Framing Error by UART
#define UART_OVERRUN_ERROR  0x0400    // Overrun condition byUART
#define UART_BUFFER_OVERFLOW 0x0200    // receive ringbuf overflow
#define UART_NO_DATA        0x0100    // no data available */

extern void uart_init(unsigned int baudrate);

extern unsigned int uart_getc(void);

extern void uart_putc(unsigned char data);

extern void uart_puts(const char *s);

extern void uart_puts_p(const char *s);

#define uart_puts_P(__s)    uart_puts_p(PSTR(__s))

extern void uart1_init(unsigned int baudrate);

extern unsigned int uart1_getc(void);

extern void uart1_putc(unsigned char data);

extern void uart1_puts(const char *s);
```

```
extern void uart1_puts_p(const char *s );  
  
#define uart1_puts_P(__s)    uart1_puts_p(PSTR(__s))  
  
int uartrec(void);  
  
#endif // UART_H
```

```
*****UART.C*****
```

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

#define UART_RX_BUFFER_MASK ( UART_RX_BUFFER_SIZE - 1)
#define UART_TX_BUFFER_MASK ( UART_TX_BUFFER_SIZE - 1)

#if ( UART_RX_BUFFER_SIZE & UART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( UART_TX_BUFFER_SIZE & UART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif

#if defined(__AVR_ATmega128__)
/* ATmega with two USART */
#define ATMEGA_USART0
#define ATMEGA_USART1
#define UART0_RECEIVE_INTERRUPT SIG_UART0_RECV
#define UART1_RECEIVE_INTERRUPT SIG_UART1_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_UART0_DATA
#define UART1_TRANSMIT_INTERRUPT SIG_UART1_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#define UART1_STATUS UCSR1A
#define UART1_CONTROL UCSR1B
#define UART1_DATA UDR1
#define UART1_UDRIE UDRIE1
#endif

static volatile unsigned char UART_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART_TxHead;
static volatile unsigned char UART_TxTail;
static volatile unsigned char UART_RxHead;
static volatile unsigned char UART_RxTail;
static volatile unsigned char UART_LastRxError;

#if defined( ATMEGA_USART1 )
static volatile unsigned char UART1_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART1_RxBuf[UART_RX_BUFFER_SIZE];
```

```
static volatile unsigned char UART1_TxHead;
static volatile unsigned char UART1_TxTail;
static volatile unsigned char UART1_RxHead;
static volatile unsigned char UART1_RxTail;
static volatile unsigned char UART1_LastRxError;
#endif
```

```
SIGNAL(UART0_RECEIVE_INTERRUPT)
```

```
{
    unsigned char tmphead;
    unsigned char data;
    unsigned char usr;
    unsigned char lastRxError;

    usr = UART0_STATUS;
    data = UART0_DATA;

#ifdef AT90_UART
    lastRxError = (usr & (_BV(FE)|_BV(DOR)));
#elif defined( ATMEGA_USART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)));
#elif defined( ATMEGA_USART0 )
    lastRxError = (usr & (_BV(FE0)|_BV(DOR0)));
#elif defined ( ATMEGA_UART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)));
#endif
```

```
    tmphead = ( UART_RxHead + 1) & UART_RX_BUFFER_MASK;
```

```
    if ( tmphead == UART_RxTail ) {
        /* error: receive buffer overflow */
        lastRxError = UART_BUFFER_OVERFLOW >> 8;
    }else{
        /* store new index */
        UART_RxHead = tmphead;
        /* store received data in buffer */
        UART_RxBuf[tmphead] = data;
    }
    UART_LastRxError = lastRxError;
}
```

```
SIGNAL(UART0_TRANSMIT_INTERRUPT)
```

```
{
    unsigned char tmptail;

    if ( UART_TxHead != UART_TxTail) {
        /* calculate and store new buffer index */
        tmptail = (UART_TxTail + 1) & UART_TX_BUFFER_MASK;
        UART_TxTail = tmptail;
        /* get one byte from buffer and write it to UART */
        UART0_DATA = UART_TxBuf[tmptail]; /* start transmission */
    }else{
        /* tx buffer empty, disable UDRE interrupt */
        UART0_CONTROL &= ~_BV(UART0_UDRIE);
    }
}

void uart_init(unsigned int baudrate)
{
    UART_TxHead = 0;
    UART_TxTail = 0;
    UART_RxHead = 0;
    UART_RxTail = 0;

#ifdef AT90_UART )
    /* set baud rate */
    UBRR = (unsigned char)baudrate;

    /* enable UART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);

#elif defined (ATMEGA_USART)
    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART0_STATUS = (1<<U2X); //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|(1<<RXEN)|(1<<TXEN);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
    #ifndef URSEL
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
```

```
#else
UCSRC = (3<<UCSZ0);
#endif

#elif defined (ATMEGA_USART0 )
/* Set baud rate */
if ( baudrate & 0x8000 )
{
    USART0_STATUS = (1<<U2X0); //Enable 2x speed
    baudrate &= ~0x8000;
}
UBRR0H = (unsigned char)(baudrate>>8);
UBRR0L = (unsigned char) baudrate;

/* Enable USART receiver and transmitter and receive complete interrupt */
USART0_CONTROL = _BV(RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

/* Set frame format: asynchronous, 8data, no parity, 1stop bit */
#ifdef URSEL0
UCSR0C = (1<<URSEL0)|(3<<UCSZ00);
#else
UCSR0C = (3<<UCSZ00);
#endif

#elif defined ( ATMEGA_UART )
/* set baud rate */
if ( baudrate & 0x8000 )
{
    USART0_STATUS = (1<<U2X); //Enable 2x speed
    baudrate &= ~0x8000;
}
UBRRHI = (unsigned char)(baudrate>>8);
UBRR = (unsigned char) baudrate;

/* Enable UART receiver and transmitter and receive complete interrupt */
USART0_CONTROL = _BV(RXCIE)|(1<<RXEN)|(1<<TXEN);

#endif

}/* uart_init */

unsigned int uart_getc(void)
{
    unsigned char tmptail;
    unsigned char data;
```

```
if ( UART_RxHead == UART_RxTail ) {
    return UART_NO_DATA; /* no data available */
}

/* calculate /store buffer index */
tmptail = (UART_RxTail + 1) & UART_RX_BUFFER_MASK;
UART_RxTail = tmptail;

/* get data from receive buffer */
data = UART_RxBuf[tmptail];

return (UART_LastRxError << 8) + data;
}/* uart_getc */

void uart_putc(unsigned char data)
{
    unsigned char tmphead;

    tmphead = (UART_TxHead + 1) & UART_TX_BUFFER_MASK;

    while ( tmphead == UART_TxTail ){
        /* wait for free space in buffer */
    }

    UART_TxBuf[tmphead] = data;
    UART_TxHead = tmphead;

    /* enable UDRE interrupt */
    UART0_CONTROL |= _BV(UART0_UDRIE);
}/* uart_putc */

void uart_puts(const char *s )
{
    while (*s)
        uart_putc(*s++);
}/* uart_puts */

void uart_puts_p(const char *progmem_s )
```

```
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) )
        uart_putc(c);

}/* uart_puts_p */

#if defined( ATMEGA_USART1 )

SIGNAL(UART1_RECEIVE_INTERRUPT)
{
    unsigned char tmphead;
    unsigned char data;
    unsigned char usr;
    unsigned char lastRxError;

    /* read UART status register and UART data register */
    usr = UART1_STATUS;
    data = UART1_DATA;

    /* */
    lastRxError = (usr & (_BV(FE1)|_BV(DOR1)) );

    /* calculate buffer index */
    tmphead = ( UART1_RxHead + 1) & UART_RX_BUFFER_MASK;

    if ( tmphead == UART1_RxTail ) {
        /* error: receive buffer overflow */
        lastRxError = UART_BUFFER_OVERFLOW >> 8;
    }else{
        /* store new index */
        UART1_RxHead = tmphead;
        /* store received data in buffer */
        UART1_RxBuf[tmphead] = data;
    }
    UART1_LastRxError = lastRxError;
}

SIGNAL(UART1_TRANSMIT_INTERRUPT)
{
    unsigned char tmptail;
```

```
if ( UART1_TxHead != UART1_TxTail) {
    /* calculate and store new buffer index */
    tmptail = (UART1_TxTail + 1) & UART_TX_BUFFER_MASK;
    UART1_TxTail = tmptail;
    /* get one byte from buffer and write it to UART */
    UART1_DATA = UART1_TxBuf[tmptail]; /* start transmission */
} else {
    /* tx buffer empty, disable UDRE interrupt */
    UART1_CONTROL &= ~_BV(UART1_UDRIE);
}
}

int uartrec(void)
{
    unsigned int c;
    char buffer[7];
    int num=134;

    itoa( num, buffer, 10); // convert interger into string (decimal format)

    uart1_putc('s');
    ms_sleep(50);
    c = uart1_getc();

    return c;
}

void uart1_init(unsigned int baudrate)
{
    UART1_TxHead = 0;
    UART1_TxTail = 0;
    UART1_RxHead = 0;
    UART1_RxTail = 0;

    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART1_STATUS = (1<<U2X1); //Enable 2x speed
        baudrate &= ~0x8000;
    }
    /*UBRR1H = (unsigned char)baudrate>>8);
    UBRR1L = (unsigned char) baudrate;*/
    UBRR1H = 0; UBRR1L = 103;

    /* Enable USART receiver and transmitter and receive complete interrupt */
}
```

```
UART1_CONTROL = _BV(RXCIE1)|(1<<RXEN1)|(1<<TXEN1);

/* Set frame format: asynchronous, 8data, no parity, 1stop bit */
#ifdef URSEL1
UCSR1C = (1<<URSEL1)|(3<<UCSZ10);
#else
UCSR1C = (3<<UCSZ10);
#endif
}/* uart_init */

unsigned int uart1_getc(void)
{
    unsigned char tmptail;
    unsigned char data;

    if ( UART1_RxHead == UART1_RxTail ) {}

    /* calculate /store buffer index */
    tmptail = (UART1_RxTail + 1) & UART_RX_BUFFER_MASK;
    UART1_RxTail = tmptail;

    /* get data from receive buffer */
    data = UART1_RxBuf[tmptail];

    return (UART1_LastRxError << 8) + data;
}/* uart1_getc */

void uart1_putc(unsigned char data)
{
    unsigned char tmphead;

    tmphead = (UART1_TxHead + 1) & UART_TX_BUFFER_MASK;

    while ( tmphead == UART1_TxTail ){
        /* wait for free space in buffer */
    }

    UART1_TxBuf[tmphead] = data;
    UART1_TxHead = tmphead;

    /* enable UDRE interrupt */
    UART1_CONTROL |= _BV(UART1_UDRIE);
```

```
    }/* uart1_putc */

void uart1_puts(const char *s )
{
    while (*s)
        uart1_putc(*s++);

}/* uart1_puts */

void uart1_puts_p(const char *progmem_s )
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) )
        uart1_putc(c);

}/* uart1_puts_p */

#endif
```

```
*****MAINPROGRAM.C*****
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
#include "Sonar.h"
#include "LCD.h"
#include "motor.h"
#include "uart.h"

void FlashLight(int );

void ObstacleAvoidance(int num);
void ObjectTracking(int num);

int right_dist, left_dist;
unsigned int c;
char buffer[7], ch;

/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 4000000UL
#endif

/* 9600 baud */
#define UART_BAUD_RATE 9600

int main(void)
{
    int num=0;

    lcdInit();
    init_timer();
    uart1_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );
    sei();

    lcdString("Hello Everyone! I am CT!");
    ms_sleep(1000);

    lcdData(0x01);
    lcdString("I am getting ready....");
    ms_sleep(1000);
```

```
    initSonar();
    lcdData(0x01);
    lcdString("Sonar Ready....");
    ms_sleep(1000);

    MOTOR_Init_Ports();
    MOTOR_Init_Timer1_PWM();
    lcdData(0x01);
    lcdString("Motor Ready....");
    ms_sleep(1000);

    uart1_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );
    lcdData(0x01);
    lcdString("UART ports ready...");
    ms_sleep(1000);

    lcdData(0x01);
    lcdString("Object Initialized....");
    ms_sleep(1000);
    while(1)
    {
        ObstacleAvoidance(100);
        ROBOT_Stop();
        ObjectTracking(100);
    }
    return 0;
}

void FlashLight(int x)
{
    int i;
    for (i=0;i<x;x++){
        ms_sleep(100);
        PORTB = 0b11111111;
        ms_sleep(100);
        PORTB = 0b00000000;
        ms_sleep(100);
    }
}

void ObstacleAvoidance(int num)
{
    while(num>0)
    {
        lcdData(0x01);
```

```

right_dist = smooth(sonarRDist());
left_dist = smooth(sonarLDist());
uart1_putc('o');
MOTOR_Motor1_Set_PWMduty(50);
MOTOR_Motor2_Set_PWMduty(50);

if(left_dist == 0)
{
    lcdString("Hard Right");
    ROBOT_Right_Centrepoint();
}
if((right_dist == 0) && (left_dist==40 || left_dist == 80 || left_dist == 120 ||
left_dist ==200))
{
    lcdString("Hard Left");
    ROBOT_Left_Centrepoint();
}
if(left_dist==40 && right_dist>=40)
{
    lcdString("Soft Right");
    MOTOR_Motor1_Set_PWMduty(30);
    MOTOR_Motor2_Set_PWMduty(30);
    ROBOT_Right_Centrepoint();
}
if((right_dist == 40) && (left_dist==80 || left_dist==120 || left_dist==200))
{
    lcdString("Soft Left");
    MOTOR_Motor1_Set_PWMduty(30);
    MOTOR_Motor2_Set_PWMduty(30);
    ROBOT_Left_Centrepoint();
}
if(((left_dist == 80) && (right_dist == 80 || right_dist == 120 || right_dist==200))
|| ((right_dist==80) && (left_dist==120 || left_dist ==200)))
{
    lcdString("Straight @ 30%");
    MOTOR_Motor1_Set_PWMduty(30);
    MOTOR_Motor2_Set_PWMduty(30);
    ROBOT_Forward();
}
if(((left_dist == 120) && (right_dist == 120 || right_dist==200)) ||
((right_dist==120) && (left_dist==200)))
{
    MOTOR_Motor1_Set_PWMduty(50);
    MOTOR_Motor2_Set_PWMduty(50);
    lcdString("Straight @ 50%");
    ROBOT_Forward();
}

```

```
    }  
  
    if((left_dist >=200) && (right_dist >= 200))  
    {  
        MOTOR_Motor1_Set_PWMduty(65);  
        MOTOR_Motor2_Set_PWMduty(65);  
        lcdString("Straight @ 70%");  
        ROBOT_Forward();  
    }  
    MOTOR_Motor1_Set_PWMduty(30);  
    MOTOR_Motor2_Set_PWMduty(30);  
    num = num-1;ms_sleep(100);  
}  
}
```

```
void ObjectTracking(int num)  
{  
    lcdData(0x01);  
    lcdString("Object Tracking");  
  
    while(num>0)  
    {  
  
        lcdData(0x01);  
        right_dist = sonarRDist();  
        left_dist = sonarLDist();  
  
        c = uartrec();  
        ch = (unsigned char) c;  
  
        if(ch == 'r')  
        {  
            lcdString("RIGHT");  
            MOTOR_Motor1_Set_PWMduty(25);  
            MOTOR_Motor2_Set_PWMduty(25);  
            ROBOT_Right_Centrepoint();  
        }  
        if(ch == 'l')  
        {  
            lcdString("LEFT");  
            MOTOR_Motor1_Set_PWMduty(25);  
            MOTOR_Motor2_Set_PWMduty(25);  
            ROBOT_Left_Centrepoint();  
        }  
    }  
}
```

```
    if(ch == 'c')
    {
        lcdString("CENTER");
        MOTOR_Motor1_Set_PWMduty(20);
        MOTOR_Motor2_Set_PWMduty(20);
        ROBOT_Forward();
    }

    num=num-1; ms_sleep(80);
}
}
```

```
*****CODES FOR OBSTACLE DETECTION AND SERIAL DATA*****
```

```
*****OBJECTTRACKING.C*****
```

```
/ ObjectTracking.cpp : Defines the entry point for the console application.
```

```
//
```

```
#include "stdafx.h"  
#include <cv.h>  
#include <cxcv.h>  
#include <highgui.h>  
#include <stdio.h>  
#include <ctype.h>  
#include "serial.h"
```

```
IplImage *image = 0, *hsv = 0, *hue = 0, *mask = 0, *backproject = 0, *histimg = 0;  
CvHistogram *hist = 0;
```

```
int backproject_mode = 0;  
int select_object = 0;  
int track_object = 0;  
int show_hist = 1;  
CvPoint origin;  
CvRect selection;  
CvRect track_window;  
CvBox2D track_box;  
CvConnectedComp track_comp;  
int hdims = 16;  
float hranges_arr[] = {0,180};  
float* hranges = hranges_arr;  
int vmin = 10, vmax = 256, smin = 30;
```

```
void on_mouse( int event, int x, int y, int flags, void* param )  
{  
    if( !image )  
        return;  
  
    if( image->origin )  
        y = image->height - y;  
  
    if( select_object )  
    {  
        selection.x = MIN(x,origin.x);  
        selection.y = MIN(y,origin.y);  
        selection.width = selection.x + CV_IABS(x - origin.x);  
        selection.height = selection.y + CV_IABS(y - origin.y);  
    }
```



```
if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])))
    capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
else if( argc == 2 )
    capture = cvCaptureFromAVI( argv[1] );

if( !capture )
{
    fprintf(stderr, "Could not initialize capturing...\n");
    return -1;
}

printf( "Hot keys: \n"
        "\tESC - quit the program\n"
        "\tc - stop the tracking\n"
        "\tb - switch to/from backprojection view\n"
        "\th - show/hide object histogram\n"
        "To initialize tracking, select the object with mouse\n" );

cvNamedWindow( "Histogram", 1 );
cvNamedWindow( "CamShiftDemo", 1 );
cvSetMouseCallback( "CamShiftDemo", on_mouse, 0 );
cvCreateTrackbar( "Vmin", "CamShiftDemo", &vmin, 256, 0 );
cvCreateTrackbar( "Vmax", "CamShiftDemo", &vmax, 256, 0 );
cvCreateTrackbar( "Smin", "CamShiftDemo", &smin, 256, 0 );

for(;;)
{
    IplImage* frame = 0;
    int i, bin_w, c;

    frame = cvQueryFrame( capture );
    if( !frame )
        break;

    if( !image )
    {
        /* allocate all the buffers */
        image = cvCreateImage( cvGetSize(frame), 8, 3 );
        image->origin = frame->origin;
        hsv = cvCreateImage( cvGetSize(frame), 8, 3 );
        hue = cvCreateImage( cvGetSize(frame), 8, 1 );
        mask = cvCreateImage( cvGetSize(frame), 8, 1 );
        backproject = cvCreateImage( cvGetSize(frame), 8, 1 );
        hist = cvCreateHist( 1, &hdims, CV_HIST_ARRAY, &hranges, 1 );
        histimg = cvCreateImage( cvSize(320,200), 8, 3 );
    }
}
```

```

    cvZero( histimg );
}

cvCopy( frame, image, 0 );
cvCvtColor( image, hsv, CV_BGR2HSV );

if( track_object )
{
    int _vmin = vmin, _vmax = vmax;

    cvInRangeS( hsv, cvScalar(0,smin,MIN(_vmin,_vmax),0),
                cvScalar(180,256,MAX(_vmin,_vmax),0), mask );
    cvSplit( hsv, hue, 0, 0, 0 );

    if( track_object < 0 )
    {
        float max_val = 0.f;
        cvSetImageROI( hue, selection );
        cvSetImageROI( mask, selection );
        cvCalcHist( &hue, hist, 0, mask );
        cvGetMinMaxHistValue( hist, 0, &max_val, 0, 0 );
        cvConvertScale( hist->bins, hist->bins, max_val ? 255. / max_val : 0., 0 );
        cvResetImageROI( hue );
        cvResetImageROI( mask );
        track_window = selection;
        track_object = 1;

        cvZero( histimg );
        bin_w = histimg->width / hdims;
        for( i = 0; i < hdims; i++ )
        {
            int val = cvRound( cvGetReal1D(hist->bins,i)*histimg->height/255 );
            CvScalar color = hsv2rgb(i*180.f/hdims);
            cvRectangle( histimg, cvPoint(i*bin_w,histimg->height),
                        cvPoint((i+1)*bin_w,histimg->height - val),
                        color, -1, 8, 0 );
        }
    }
}

xcod = track_window.x; ycod = track_window.y;
area = track_window.height * track_window.width;
ch = abc.receive(1);
printf("%c", ch[0]);
if(ch[0]== 's')
{

```

```
        if (xcod <250)
        {
            printf("Left\n");
            x=abc.send("l",1);
        }
        if (xcod >450)
        {
            printf("Right\n");
            x=abc.send("r",1);
        }
        if (xcod >= 250 && xcod <= 450)
        {
            printf("Centre\n");
            x=abc.send("c",1);
        }
    }
```

```
        cvCalcBackProject( &hue, backproject, hist );
cvAnd( backproject, mask, backproject, 0 );
cvCamShift( backproject, track_window,
cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),
    &track_comp, &track_box );
track_window = track_comp.rect;

if( backproject_mode )
    cvCvtColor( backproject, image, CV_GRAY2BGR );
if( !image->origin )
    track_box.angle = -track_box.angle;
cvEllipseBox( image, track_box, CV_RGB(255,0,0), 3, CV_AA, 0 );
}

if( select_object && selection.width > 0 && selection.height > 0 )
{
    cvSetImageROI( image, selection );
    cvXorS( image, cvScalarAll(255), image, 0 );
    cvResetImageROI( image );
}

cvShowImage( "CamShiftDemo", image );
cvShowImage( "Histogram", histimg );

c = cvWaitKey(10);
if( (char) c == 27 )
    break;
switch( (char) c )
```

```
{
  case 'b':
    backproject_mode ^= 1;
    break;
  case 'c':
    track_object = 0;
    cvZero( histing );
    break;
  case 'h':
    show_hist ^= 1;
    if( !show_hist )
      cvDestroyWindow( "Histogram" );
    else
      cvNamedWindow( "Histogram", 1 );
    break;
  default:
    ;
}
}

cvReleaseCapture( &capture );
cvDestroyWindow("CamShiftDemo");

return 0;
}

#ifdef _EiC
main(1,"ObjectTracking.cpp");
#endif
```

```
*****SERIAL.H*****
```

```
#ifndef __Serial_h__
#define __Serial_h__

#include <iostream>
#include <windows.h>
#include "serial.h"
#include <stdio.h>
#include <time.h>
#include <string>
#include <stdlib.h>

#define MAX_NR_BYTES 64

using namespace std;

class Serial{

private:
    HANDLE handle;
    BYTE byte[MAX_NR_BYTES];
    DWORD nrBytes;
    COMMTIMEOUTS timeout;
    DCB dcb;
public:
    Serial();
    char * receive(int length);
    int send(char *command, int bytes);
    int connect(char *port);
    int connect(char *port, int set_timeout, int bytesize, int baudrate, int parity);
    int disconnect();
    int flush();
};

#endif
```

```
*****SERIAL.C*****
```

```
#include "stdafx.h"  
#include "serial.h"
```

```
using namespace std;
```

```
Serial::Serial(){
```

```
}
```

```
int Serial::send(char *command, int bytes){
```

```
    int i=0;
```

```
    if (handle == INVALID_HANDLE_VALUE){
```

```
        return 0;
```

```
    }
```

```
    while(i<bytes){
```

```
        byte[i]=command[i];
```

```
        i++;
```

```
    }
```

```
    if(WriteFile(handle, byte, bytes, &nrBytes, NULL)!=0){
```

```
        return 1;
```

```
    }
```

```
    else
```

```
        return 0;
```

```
}
```

```
char *Serial::receive(int length){
```

```
    static char answer[MAX_NR_BYTES];
```

```
    int i=0;
```

```
    nrBytes=0;
```

```
    answer[0]='\0';
```

```
    if (ReadFile(handle, byte, length, &nrBytes, NULL) == 0){
```

```
        answer[0]='e';
```

```
        answer[1]='r';
```

```
        answer[2]='r';
```

```
        answer[3]='o';
```

```
        answer[4]='r';
```

```
        answer[5]='\0';
```

```
        return &answer[0];
```

```
    }
```

```
    //printf("Number of bytes %d\n",nrBytes);
```

```
    while(i < nrBytes){
```

```
        answer[i]=byte[i];
        //printf("%d ",answer[i]);
        i++;
    }
    //printf("\n");
    answer[i]='\0';
    return &answer[0];
}

int Serial::flush(){
    int i=0;
    clock_t endwait;
    //endwait = clock () + 1000 * CLK_TCK ;
    while(nrBytes > 0){

        if(ReadFile(handle, byte, 1, &nrBytes, NULL)==0){
            return 0;
        }
        i++;
    }
    return i;
}

int Serial::connect(char *port){
    return connect(port, 200, 8, 19200, 0);
}

int Serial::connect(char *port, int set_timeout, int bytesize, int baudrate, int parity){
    handle = CreateFile(port, GENERIC_READ | GENERIC_WRITE,
0,0,OPEN_EXISTING,0,0);
    if (handle == INVALID_HANDLE_VALUE){
        return 0;
    }
    dcb.ByteSize = bytesize;/*sets bit-size*/
    dcb.BaudRate = baudrate;/*sets baudrate*/
    /*Sets timeout for reading - 0 = deactivate*/
    timeout.ReadIntervalTimeout=set_timeout;
    timeout.ReadTotalTimeoutConstant=set_timeout;
    dcb.Parity=parity;
    timeout.ReadTotalTimeoutMultiplier=set_timeout;
    SetCommTimeouts(handle, &timeout);
    SetCommState(handle, &dcb);
    printf("COM Port Initialized");
    return 1;
}
```

```
int Serial::disconnect(){
    if (handle){
        CloseHandle(handle);
        return 1;
    }
    else{
        return 0;
    }
}
```