B.O.S.S.

A Battery Operated Smart Servant

University of Florida
Department of Electrical and Computer Engineering
EEL 5666c
Intelligent Machines Design Laboratory

Gregory Garcia
August 5, 2006

Table of Contents

ABSTRACT

B.O.S.S. is an autonomous ground vehicle whose function is to follow his master. He will use computer vision to distinguish his master from other people that might cross his line of sight and use sonar to stay close behind him as he walks through a grocery store. While in motion he will maintain a safe trailing distance, but when he detects that his master has stopped B.O.S.S. will enter a new behavior state where he slowly creeps up to his master. In addition to tracking his master B.O.S.S. will need to avoid hitting any obstacles that might lie between his position and his goal location.

EXECUTIVE SUMMARY

B.O.S.S. is a skid steered autonomous robot agent that tracks a target of interest. The platform is propelled by two right angle DC gear motors. These motors are driven by an H-bridge motor controller that receives a PWM signal from the Mavric IIB microcontroller. While capable of traveling in both the forward and reverse directions, the motors are only used to actuate the platform for turning and forward driving scenarios. The motors are retrofitted with Transorb bi-directional transient voltage suppressors, along with a 3 capacitor array to prevent both voltage spikes and for reducing noise. All the motor controller hardware is located in its own modular compartment below the vehicle. The enclosure is outfitted with a fan thereby forcing convection. This will allow for higher current drawing by the motors without the possibility of damaging the transistors. Furthermore the actuator power system is on a separate fuse from the rest of the systems and also has a dedicated kill switch. Commands are sent to the motor controllers via the microcontroller, which makes a decision based on the perception information it receives. The main sensor deployed on B.O.S.S is computer vision via a USB webcam. This information is processed on the onboard single board computer and then transmitted to the microcontroller via a RS-232 serial connection. Additional sensors deployed are sonar, and IR. The sonar is the most reliable ranging sensor deployed. It's intent is for longer range obstacle avoidance, namely the target it is following. The IR are used for close range avoidance. Together this integrated system offers a suitable solution for the object recognition and tracking problem.

INTRODUCTION

In light of technological advances the world moves at a faster pace today than it did yesterday. The heartbeat of the world is consistently accelerating. This enables events to happen faster and thus necessitates that we Americans accomplish more in order to optimize our productivity and stay competitive in the world marketplace. That said, we Americans have become creatures of convenience. In America today the corner grocer has been replaced by the SUPER-market because it allows us to get all our shopping done at once, thereby optimizing our productivity. The result is the creation of mega-stores the size of warehouses. Hand baskets have been replaced by push carts to accommodate the heavier loads we have to carry. One can easily get lost in such a super center let alone loose your shopping cart. We have enough issues to worry about on a daily basis and more important tasks to devote our attention to than to preoccupying ourselves with where our shopping cart is and figuring out how to navigate a crowded grocery store aisle. Super-markets go out of their way to cater to the clientele by providing a happier more stress free environment and user friendly and convenient environment.

For years engineers and scientists have understood and developed the concepts and technologies governing autonomous vehicle navigation. But with each passing day the technology is refined and made more compact and for a smaller cost. This has enabled to technology to extend itself beyond the research community and into the consumer market place. This paper will commence by defining the problem and proposing a solution. Subsequently it will discuss all the technology and other means that are required for the proposed solution. Lastly it will conclude by describing the method and metrics by which the solution will be evaluated and validated while providing some analysis of the collected results.

INTEGRATED SYSTEM

Upon completion B.O.S.S. will be comprised of several main components:

1. Microcontroller
2. Computer
3. Sensor suite
4. Actuation
5. Power system

MICROCONTROLLER:

MAVRIC-IIB Mega AVR Integrated Controller with ATmega 126 MCU

This board is used to perform all the I/O. It is the core component of this agent. The only operation not contained on this board is the actual image processing. That information is handled on single board computer and transferred via RS-232 serial connection.

NOVA single board computer:

1 GHz Pentium 3 computer with 512 MB of ram

This board is used to connect to a USB camera and it runs my image processing algorithm. All the tracking is handled onboard and the outputs sensor metadata to the microcontroller. This metadata is in the form of a single character that represents the algorithms best recommendation to the microcontroller as to what course of action it should take.

SENSOR SUITE: see sensor section

ACTUATION: see actuation section

POWER SYSTEM: At the heart of B.O.S.S. is a 4 cell SLA (sealed Lead Acid) 12V DC battery bank. Each cell is rated at 7.2 amp hours. This power system is the basis for the entire power grid distribution. The 12V power is fed through a 50A rated toggle switch which function as the master power breaker. From there it enters the enclosure and is fed into a Buss ATC fuse block. From there it branches to all the different components that will require power. Each branch is independently fused. One branch is sent to a bus bar where low current 12V sensors and peripherals such as the singleboard will receive their power. See FIGURE (I'll add later) One branch will power the motor drivers, another powers a CAL Ex 12V to 5V DC/DC converter see figure below.

Figure 1 Cal Ex DC/DC converter

This output is then sent to a 5V bus bar. The 5 V power bus is used to supply “clean” and “conditioned” power both the microcontroller and the single board computer.

MOBILE PLATFORM

The base platform for this vehicle was a stock shopping cart. Modifications were done to accommodate actuation, computing hardware, power systems, and sensor packages. It is important to note that was not necessary to use a shopping cart for development purposes.

Figure 2 B.O.S.S. Platform

Any forward caster rear steered vehicle should suffice so long as whatever system is developed can be physically mounted to the target platform at a later time. A small shopping cart was chosen to ensure that the system developed herein could be applied in a wide spread manner to a majority of existing carts. Possible modifications to the existing platform include either adding a bearing carrier to handle all types of dynamic loading scenarios, or perhaps the best solution is employing a set of smaller wheel that are always in contact with the existing cart wheels such that no loads are actually realized by the motor. This would also facilitate an easier, more seamless conversion of existing carts into ones with full actuation.

ACTUATION

The underlying goal behind vehicle actuation is to keep it simple and cost effective. Shopping carts currently are steered by applying a torque about the z-axis (coming out of the ground and pointing up to the sky). This torque in turn causes the rear wheels which have a fixed position and orientation, to rotate with a prescribed angular speed and direction that is directly related to the amount of effort being applied. For this purpose both a right hand and left hand Denso 12V power window motors were chosen. The motor, shown below, is a gear motor that is set to spin at 150 revolutions per minute.

Figure 3 Drive system actuation

At this rate and with an 8 inch wheel will produce a maximum linear speed of approximately 4 miles per hour. This should allow B.O.S.S. to keep up with even the fastest pace shoppers. Because these motors were initially designed for high torque applications such as power window motors they are strong robust motors that can be run continuously and require a 12V DC supply voltage. The only drawback to these motors is their high current consumption. A conservative estimate of 5 amps current draw by each motor will be used to specify the appropriate motor driver to be used. Research is currently underway into finding the appropriate combination of motor drivers/ transient voltage suppressors to help power these devices cleanly.

SENSORS

The same fundamental sensor technologies and intelligence algorithms found in the research vehicles at CIMAR (Center for Intelligent Machines and Robotics) will be applied to B.O.S.S. only at a fraction of the cost, making it a practical and feasible shopping aide. The primary sensor package is a vision system that will use a Logitech webcam to gather a 640x480 image. This image will then be processed by a NOVA 7986

Figure 4 Vision Sensor

Figure 5 Single board computer

single board computer with 512 megabytes of RAM and 1.4 GHz PIII processor. The data will then be transmitted via RS-232 serial cable to the Mavric IIB microcontroller board. While it is possible to use a PC104 I/O stack and by-pass the microcontroller entirely there was educational value to integrate both systems.

Figure 6 Vision and Sonar implementation

Figure 7 Vision and Sonar with Buzzer

The two figures above depict the relative positioning of the 2 major sensors for obstacle tracking. Note how the sonar has been mounted inline with the camera. This configuration allows for the camera feedback to maintain the item of interest directly inline of the sonar sensor. Any errors in tracking should be made up by the sonar cone as shown in figure 9.

An object tracking algorithm was written to scan the image file. Using a set of empirically determined color ranges it will scan each frame of the video and paint the desired target object a specific shade of orange. This data is then filtered using a mean filter.

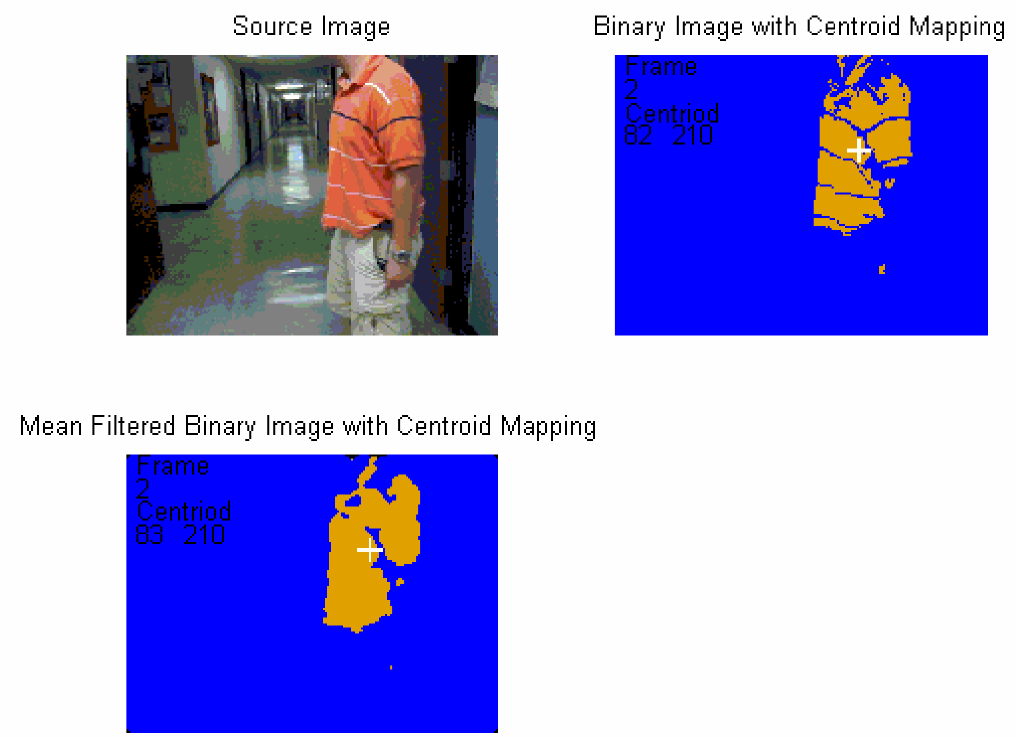

Figure 8 Processed Image File

Another algorithm was developed to locate the centriod of each target region and paint a target around the centriod location. The centriod location along with the frame number is also printed on screen. The centriod location is the parameter which will be used for feedback control.

The platform will also be equipped 2 other sensor types, the first of which is sonar, see below.

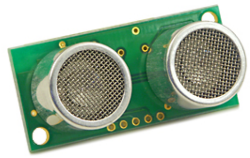

Figure 9 Devantec Sonar

The sonar will be used to determine the vehicle's distance to target and two other sonar sensors will be pointed away from each side of the vehicle in order to judge distance to the sides and determine the presence of nearby obstacles. Sonar was chosen because of its ability to work in all lighting conditions, and because it allowed for the greatest range of measurements (3 inches to 10 feet).

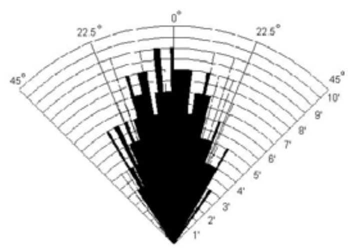

Figure 10 Devantec SRF04 Beam Pattern

The last sensor type to be included in B.O.S.S's sensor suite is infrared, seen below.

Figure 11 Sharp IR sensor

An array of Sharp GP2D12 IR range sensors will be placed around the front of the vehicle as shown in the figure below.

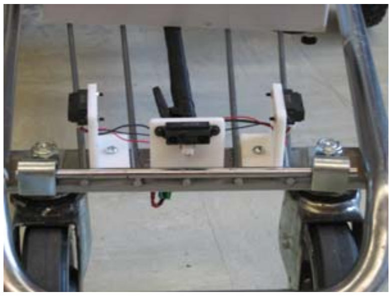

Figure 12 IR Array

Figure 13 IR Array

These sensors were secured to brackets made of abs plastic that was formed using a rapid prototype machine with soluble support material. These brackets were then secured by with 10-32 tapped holes into the chassis. Enough free space is available for fine adjustment and alignment of the IR sensors should the necessity arise.

During testing the issue of signal noise became a problem. In order to solve this problem all the wiring for the IR sensors was removed and replace with a 6 conductor shielded cable. Furthermore, as illustrated in figures 13 and 14, a new entry location was added to the enclosure to allow for the separation of power and data lines. This was initially not considered to be a problem since all the onboard power systems run on DC power, however there are lines carrying high amperage DC voltage that may have contributed to this problem. A 10 microfarad capacitor was also added to the voltage supply line to ensure that the IR emitters received a clean power source to ensure the output was consistent.

Figure 14 Sensor Enclosure; separate Data and Power

Figure 15 Data Lines along rail closest to Enclosure; Power is away

Data is read in from the IR sensors at using the analog to digital converter on the microcontroller. This data is sampled every 32 milliseconds. The data is then read into and stored in a ring buffer that contains 32 elements. Once the ring buffer is fully populated an average is computed and this averaged value is used for decision making. Once the ring buffer is full the values overwrite themselves so there is no memory leak.

In order for B.O.S.S. to be marketable it must be both smart enough and robust enough to handle the complex and demanding real world settings.

BEHAVIORS

In order to successfully function in a real world setting boss must be able to effectively meet a number of criteria. These criteria will be satisfied within multiple vehicle intelligence behaviors. The first behavior will be to acquire the target. It must be able to effectively recognize the target to follow it. Another behavior associated with the first will be to train to a number of perspective target colors at startup. The next behavior will be a predictor. Based on where the target is in the image the robot will attempt to maintain it in the center of its field of view. The predictor will govern what type of maneuvers the vehicle makes. If for some reason the vehicle is unable to reacquire the target it will enter a lost behavior mode where it will give an audible notice to alert the operator that the vehicle has faulted in some way. Because supermarkets are high traffic areas it is likely that there will be other carts and people in the aisles, therefore it must have a obstacle locating behavior that is operational whenever the vehicle is moving and works in parallel with all other behavior modes. When an obstacle is encountered and the vehicle is unable to move in that direction the vehicle will stop so as to avoid a collision.

EXPERIMENTAL LAYOUT AND RESULTS

Integrated System Test:

For the purposes of validating the designed platform and behavior modes a test environment similar to that of a supermarket will be created. The robot will learn who I am (may just have to program my color into it) and then follow me. In this case testing occurred on the 3rd floor of Mechanical and Aerospace Engineering Building B. The course layout involved a long wide hallway with two turns. This should be suitable for representing a supermarket aisle. I walked down the aisle avoiding obstacles myself and stopping at random intervals. For the most part B.O.S.S. successfully completed his objective. Some problematic issues were encountered during testing. The lighting in the hallway was inconsistent. It was noted that if the vehicle was trained in a bright segment of the hallway it had a greater probability of getting lost in those darker areas. The sonar sensor worked successfully without any issue. The IR sensors performed as expected, accurate on most surfaces but still had trouble seeing other. During the course of testing other individuals would traverse the hallway but B.O.S.S. did not detect them. It should be noted that they were wearing different color clothing.

Vision Test:

Once the camera was installed onto the body several tests were performed on the vision system to determine if it could effectively locate and track an image. The first test consisted of hard coding in the RGB color values it was to look for. This method worked well if we allowed the appropriate amount of tolerance for each color. However this method is subject to failure if the environmental conditions change enough to adversely affect the camera's perceived image color. In order to remedy this problem a training routine was added where any one of a number of colors could be placed anywhere inside of the camera's field of view and it would learn to search for that color. This algorithm was tested extensively in the lab under varying lighting conditions and proved to be successful and repeatable.

IR Test:

The infrared sensors were tested for their ability to successfully range a hand, a trash can, and a wall. By far, the IR sensor found the trash can with the greatest of ease. Next came its ability to find a wall, followed by the hand. In all cases the sensor reported reasonable data, however I would not trust it the primary ranging device due to its sensitivity to lighting and object recognition.

Sonar Test:

The accuracy of the sonar ranger was tested on a number of different surfaces to ensure that the echo times would be similar. The first test simply consisted of moving a small object, my hand, back and forth to known distances. The sonar values were tabulated. Next different surfaces were used to include a trash can, clothing (on a person), and carpet. All the results showed that findings were similar enough to ensure this sensor would report accurate readings against any surface the robot is likely to encounter.

Kill Switch Test:

Deployed on B.O.S.S. are 2 separate kill switches. The first switch is a master big red button, kill everything type of switch. It is rated for 50 amps of current. This switch was tested using a millimeter to measure resistance. The second kill switch is rated at 30 amps and is for the actuators only. This switch interrupts power to the drive motors so that any faulty behaviors can be debugged without losing data by disconnecting power to the computers.

CONCLUSION

B.O.S.S. proved to be a successful companion for any shopper. The agent satisfactorily achieved all of the goals it was intended to accomplish. It proved that it could learn a color, and successfully recognize and dynamically track it. The sonar was capable of detecting obstacles at allotted distances in order to avoid collisions. Additionally the infrared sensors proved they could offer some limited close range perception data. This solution would not be a feasible nor acceptable solution for a real world application due to its limitations. Current camera technology is very susceptible to false positives. In reality some sort of pattern recognition or other more sophisticated location method should be employed. Furthermore current sensing and behavior modes are not equipped to hand multiple robot scenarios. Very few areas exceeded my expectations except that it actually worked. Otherwise, just about every task seemed to be a multi-hour undertaking. If I had to start the project over I would use a smaller platform, find a better camera, and put bearings on my motor shafts. Due to the size of the platform, maneuverability is an issue. Not only is it hard to navigate, but without an increased number of sensors it is difficult to perceive the environment enough to perform effective obstacle avoidance. Some problems I encountered using the USB webcam is that it continually tries to auto adjust the image. In turn the camera adjusts its properties to find colors that don't really exist. This made tracking increasingly difficult in that when I was not in the image, the camera auto adjust and get false hits. Lastly when I mounted the drive motors I did include in the design bearings to support the load on the tires. In the future I would either include bearings, or modify the design completely such that the original wheels were left on the cart and the motor's tires would directly contact the other wheels to propel the cart. In effect the motor would see no radial loading.

DOCUMENTATION

[1] Acroname Inc., http://www.acroname.com/index.html

[2] Intel Open CV, http://www.intel.com/research/mrl/research/opencv/

[3] BD Micro, http://www.bdmicro.com/mavric-iib/

APPENDICES

CodeVision:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.9 Standard
Automatic Program Generator
© Copyright 1998-2006 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : IMDL B.O.S.S.
Version : I2C Sonar setup but no code in while loop, IR and motor drivers
Date    : 7/17/2006
Author  : LAB
Company : LAB
Comments:


Chip type           : ATmega128
Program type        : Application
Clock frequency     : 14.745600 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 1024
*****************************************************/
#include <mega128.h>
#include <delay.h>
#include <stdio.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
#include <i2c.h >
/*
  CodeVisionAVR C Compiler
  (C) 1998-2000 Pavel Haiduc, HP InfoTech S.R.L.

  Prototypes for I2C bus master functions

  BEFORE #include -ING THIS FILE YOU
  MUST DECLARE THE I/O ADDRESS OF THE
  DATA REGISTER OF THE PORT AT WHICH
  THE I2C BUS IS CONNECTED AND
  THE DATA BITS USED FOR SDA & SCL

  EXAMPLE FOR PORTB:

    #asm
        .equ __i2c_port=0x18
        .equ __sda_bit=3
        .equ __scl_bit=4
    #endasm
    #include <i2c.h>
*/

#ifndef _I2C_INCLUDED_
#define _I2C_INCLUDED_

#pragma used+
void i2c_init(void);
unsigned char i2c_start(void);
void i2c_stop(void);
unsigned char i2c_read(unsigned char ack);
unsigned char i2c_write(unsigned char data);
#pragma used-

#endif

#include <math.h>
```

```
#include <string.h>
#include <stdlib.h>

#define ADC_VREF_TYPE 0x40

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

// Declare your global variables here
    void buzzer(void);
    void stop(void);
    void steer_left(void);
    void steer_hardLeft(void);
    void steer_right(void);
    void steer_hardRight(void);
    void forwardTravelSpeed_fast(void);
    void forwardTravelSpeed_slow(void);
    unsigned char D;
    unsigned char P;
    unsigned char F;
    char lcd_buffer[33];
    short int F_IR_buffer[32];
    short int D_IR_buffer[32];
    short int P_IR_buffer[32];
    short int irc=0;
    short int F_avg;
    short int D_avg;
    short int P_avg;
    short int F_last;
    short int D_last;
    short int P_last;
    short int F_sum;
    short int D_sum;
    short int P_sum;
    long  int cntr=0;
    short int forward_clear;
    short int driver_clear;
    short int passenger_clear;
    short int vision_comm=0;
    unsigned char sonar_range;
    void ping_sonar(void);



void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
DDRA=0b01000000;PORTA=0x00;

// Port B initialization
// Func7=In Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=in Func0=Out
// State7=T State6=0 State5=0 State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x7D;

// Port C initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=1 State1=T State0=T
PORTD=0x04;
DDRD=0x04;

// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;

// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: NOT SURE but SENDS PWM at 56 Hz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-inverted
// OC1B output: Non-inverted
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0xAB;
TCCR1B=0x2C;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
```

```
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: 14.400 kHz
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x10;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: Off
// USART0 Mode: Asynchronous
// USART0 Baud rate: 9600
UCSR0A=0x00;
UCSR0B=0x10;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x5F;
```

```
// ADC initialization
// ADC Clock frequency: 921.600 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x84;

// LCD module initialization
lcd_init(16);


    lcd_clear();
    delay_ms(250);
    sprintf(lcd_buffer, "Starting up press any key",F_avg,D_avg,P_avg);
    lcd_clear();
    lcd_puts(lcd_buffer);
    delay_ms(250);

    //TEST FOR SERIAL PORT AND PRINT TO SCREEN
    //Do nothing till the serial port becomes active

while (1)
    {
    vision_comm=(int)getchar();
    sprintf(lcd_buffer, "Vision:%-u",vision_comm);
    lcd_clear();
    lcd_puts(lcd_buffer);
    //delay_ms(250);


    //read IR values
    //Buffer for IR data handling

     F_last=0;
     D_last=0;
     P_last=0;
    for( irc=0; irc<=31; irc++)
    {
      D=read_adc(1);    //Driver's side IR sensor
      P=read_adc(3);    //Passenger's side IR sensor
      F=read_adc(2);    //Forward IR sensor
      F_IR_buffer[irc]=F;
      D_IR_buffer[irc]=D;
      P_IR_buffer[irc]=P;
      F_sum=F_last+F_IR_buffer[irc];
      D_sum=D_last+D_IR_buffer[irc];
      P_sum=P_last+P_IR_buffer[irc];
      F_last=F_sum;
      D_last=D_sum;
      P_last=P_sum;
    }
    //end of buffer

    F_avg=(F_sum)/32;
    D_avg=(D_sum)/32;
    P_avg=(P_sum)/32;

    //SONAR SRF05 code
    ping_sonar();

   // if(cntr<1000)            //GO SLOW

    //Set view conditions for IR
    if(F_avg<75)
    {
     forward_clear=1;
    }



    else forward_clear=0;
```

```
 if(D_avg<125)
 {
  driver_clear=1;
 }
 else driver_clear=0;

 if(P_avg<125)
 {
  passenger_clear=1;
 }
 else passenger_clear=0;

//sprintf(lcd_buffer, "F:%-u",F_avg);
 //sprintf(lcd_buffer, "D:%-u,   F:%-u,    P:%-u",D_avg,F_avg,P_avg);



 //Vision Tracking ======================================================================
 //RECEIVE SERIAL DATA FROM CAMERA
 //Decision making for how much turn effort
                                //1= steer_hardLeft
                                //2= steer_left
                                //3= go straight
                                //4= steer_right
                                //5= steer_hardRight
                                //6= fault --Buzzer

 if (vision_comm==1 && driver_clear && cntr>500)
 {
   steer_hardLeft();
 }

 else if (vision_comm==2 && driver_clear && cntr>500)
 {
   steer_left();
 }

  else if (vision_comm==3 && cntr>1000)
 {
   forwardTravelSpeed_fast();
 }

 else if (vision_comm==3 && cntr>=500 && cntr<=1000)
 {
   forwardTravelSpeed_slow();
 }

  else if (vision_comm==3 && cntr<=500 ||!forward_clear)
 {
   stop();
 }

  else if (vision_comm==4 && passenger_clear && cntr>500)
 {
   steer_right();
 }

  else if (vision_comm==5 && passenger_clear && cntr>500)
 {
   steer_hardRight();
 }

  else if (vision_comm==6 && cntr<500)
  {
   stop();
  }
  else if (vision_comm==6)
 {
   stop();
   buzzer();
```

```
    }

    //End of Vision tracking ===============================================================

    /*
    //Obstacle avoidance------------------------------------------------------


      //forward_clear,
      if (forward_clear && driver_clear && passenger_clear)
      {
        forwardTravelSpeed_fast();
      }

   //   else if (!forward_clear )
   //   {
   //     stop();
   //     buzzer();
   //   }



      else if(!forward_clear && !driver_clear && !passenger_clear)
      {
      stop();
      buzzer();
      }

      else if (!driver_clear && forward_clear && passenger_clear)
      {
            steer_hardRight();
      }

      else if (driver_clear && forward_clear && !passenger_clear)
      {
            steer_hardLeft();
      }

      else if (!driver_clear && forward_clear && !passenger_clear)
      {
            forwardTravelSpeed_slow();
      }

      else if (!driver_clear && !forward_clear && passenger_clear)
      {
            steer_hardRight();
      }

      else if (driver_clear && !forward_clear && !passenger_clear)
      {
            steer_hardLeft();
      }

   //Insert code for cases where vehicle is blocked in front and on some of the sides
    */
    }
}


//SONAR Functions


void ping_sonar()

{
            cntr=0;
            PORTA.6=1;
            delay_ms(1);
```

```
            PORTA.6=0;      //now start the timer
            //delay_ms(1);
            while(PINA.7==0){}
            while(PINA.7==1)
            {
             cntr++;
             delay_us(1);
            }
            //printf( "Sonar: %d",cntr);
            sprintf(lcd_buffer, "Sonar: %d",cntr);
            lcd_clear();
            lcd_puts(lcd_buffer);
            //delay_ms(1000);
}

//Steering functions
void steer_left(void)
{
     OCR1AL=250;       //Passenger Motor speed
     OCR1BL=125;       //Driver Motor speed
     PORTB.0=1;
     PORTB.3=0;
     PORTB.2=1;
     PORTB.4=0;
}

void steer_hardLeft(void)
{
     OCR1AL=250;       //Passenger Motor speed
     OCR1BL=50;        //Driver Motor speed
     PORTB.0=1;
     PORTB.3=0;
     PORTB.2=1;
     PORTB.4=0;
     //delay_ms(500);
}

void steer_right(void)
{
     OCR1AL=125;       //Passenger Motor speed
     OCR1BL=250;       //Driver Motor speed
     PORTB.0=1;
     PORTB.3=0;
     PORTB.2=1;
     PORTB.4=0;
     //delay_ms(500);
}

void steer_hardRight(void)
{
     OCR1AL=50;        //Passenger Motor speed
     OCR1BL=250;       //Driver Motor speed
     PORTB.0=1;
     PORTB.3=0;
     PORTB.2=1;
     PORTB.4=0;
}


//Driving Speed functions

void stop(void)
{
     OCR1AL=225;       //Passenger Motor speed
     OCR1BL=225;       //Driver Motor speed
     PORTB.0=1;
     PORTB.3=1;
     PORTB.2=1;
     PORTB.4=1;
}
```

```
void forwardTravelSpeed_fast(void)
{
      OCR1AL=250;       //Passenger Motor speed
      OCR1BL=250;       //Driver Motor speed
      PORTB.0=1;
      PORTB.3=0;
      PORTB.2=1;
      PORTB.4=0;
}

void forwardTravelSpeed_slow(void)
{
      OCR1AL=200;       //Passenger Motor speed
      OCR1BL=200;       //Driver Motor speed
      PORTB.0=1;
      PORTB.3=0;
      PORTB.2=1;
      PORTB.4=0;
}
void buzzer(void)
{
   PORTD.2=0;
   delay_ms(750);
   PORTD.2=1;
   delay_ms(750);

}
```

B.O.S.S. Vision Image Processing and Serial Communications

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "cv.h"
#include "cvcam.h"
#include "highgui.h"
#include "cxcore.h"
#include "windows.h"
#include "time.h"
#include "string.h"
#include "stdio.h"

BOOL SetCommDefaults(HANDLE hSerial);
OVERLAPPED ovWrite;

double Rmin=255;
double Rmax=0;
double Gmin=255;
double Gmax=0;
double Bmax=255;
double Bmin=0;
double maxR[100]={0};
double maxG[100]={0};
double maxB[100]={0};
double minR[100]={0};
double minG[100]={0};
double minB[100]={0};
double avgRmax=0;
double avgGmax=0;
double avgBmax=0;
double avgRmin=0;
double avgGmin=0;
```

```
double avgBmin=0;
int matcnt=0;
int hsum = 0;
int wsum = 0;
int havg = 0;
int wavg = 0;
int avg_counter = 0;
char output;
int count = 0;

double pxlRed = 0;
double pxlGreen = 0;
double pxlBlue = 0;
long double redCount = 0;
long double greenCount = 0;
long double blueCount = 0;
long double avgRed = 0;
long double avgGreen = 0;
long double avgBlue = 0;

IplImage* ImgResult = 0;
IplImage* ImgResultM = 0;
IplImage* ImgResultR = 0;
IplImage* ImgResultG = 0;
IplImage* ImgResultB = 0;
//check source image size
//preparing separat RGB image


int main(int argc, char* argv[])
{
char counter = 0;
//serial comm stuff
HANDLE hSerial = CreateFile("COM1", GENERIC_READ | GENERIC_WRITE,0,NULL,OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED | FILE_FLAG_NO_BUFFERING,NULL);
SetCommDefaults(hSerial);
DWORD dwBytesWritten = 0;
memset(&ovWrite,0,sizeof(ovWrite));




//

/*              //serail stuff -----------------
                int        c;
   int        erreur;
   char       txt[32];
   Tserial_event *com;
                com = new Tserial_event();
                com->setManager(SerialEventManager);
      erreur = com->connect("COM1", 19200, SERIAL_PARITY_NONE, 8, true);
*/              //end serial stuff--------------

   IplImage* Iplimage = 0;
                CvCapture* capture = cvCaptureFromCAM(0); // NEED one time call
                //cvNamedWindow( "BossVision", 0 );
                //cvResizeWindow( "BossVision", 352, 288 );

                //cvNamedWindow( "Filtered Result", 0 );
                //cvResizeWindow( "Filtered Result", 352, 288 );





                //one time routine for checking image size
                cvGrabFrame( capture );
   Iplimage = cvRetrieveFrame( capture );
```

```
//check source image size
CvSize sourceSize;
sourceSize=cvGetSize(Iplimage);
printf("Source image size is width= %d, height = %d, \n",sourceSize.height, sourceSize.width);


CvSize szResult;
szResult.height = sourceSize.height;
szResult.width  = sourceSize.width;

//Creating Image Structured Objects
ImgResult = cvCreateImage(szResult, 8, 3); //create RGB image
ImgResultM = cvCreateImage(szResult, 8, 3); //create RGB image
ImgResultR = cvCreateImage(szResult, 8, 1); //create gray image
ImgResultG = cvCreateImage(szResult, 8, 1); //create gray image
ImgResultB = cvCreateImage(szResult, 8, 1); //create gray image
IplImage* IplRed     = cvCreateImage(cvGetSize(Iplimage), IPL_DEPTH_8U,1);
IplImage* IplGreen   = cvCreateImage(cvGetSize(Iplimage), IPL_DEPTH_8U,1);
IplImage* IplBlue    = cvCreateImage(cvGetSize(Iplimage), IPL_DEPTH_8U,1);


cvFillImage(ImgResultR, 255);
cvFillImage(ImgResultG, 255);
cvFillImage(ImgResultB, 255);
/////////////////////////////////////////////////////////////
//-------------------------------------------------------------------------------------
//Training Image procedure


for(matcnt=0;matcnt<100;matcnt++)
{


        //capture image
        cvGrabFrame( capture );
Iplimage = cvRetrieveFrame( capture );
        //cvFillImage(ImgResultR, 255);
        //cvFillImage(ImgResultG, 255);
        //cvFillImage(ImgResultB, 255);



        cvCvtPixToPlane(Iplimage, IplBlue,IplGreen, IplRed,NULL);


        for(int h=0; h<sourceSize.height; h++)
                //for(int h=sourceSize.height; h<0; h--)
{
                for(int w=0; w<sourceSize.width; w++)
                {
                        //memo : you can make arrary for saving pixel data instead of pxlRed …
                        pxlRed   = cvGetReal2D(IplRed     , h ,w);
                        pxlGreen = cvGetReal2D(IplGreen   , h ,w);
                        pxlBlue  = cvGetReal2D(IplBlue    , h ,w);

                        redCount   += pxlRed;
                        greenCount += pxlGreen;
                        blueCount  += pxlBlue;
                }
}
}

avgRed  = (redCount)/(100*288*352);
avgGreen= (greenCount)/(100*288*352);
avgBlue = (blueCount)/(100*288*352);

avgRmin = avgRed - 30;
avgGmin = avgGreen - 30;
```

```
            avgBmin = avgBlue - 30;
            avgRmax = avgRed + 30;
            avgGmax = avgGreen + 30;
            avgBmax = avgBlue + 30;

            if(avgRmin < 0)
            {
                        avgRmin = 0;
            }
            else if(avgRmax > 255)
            {
                        avgRmax = 255;
            }
            if(avgGmin < 0)
            {
                        avgGmin = 0;
            }
            else if(avgGmax > 255)
            {
                        avgGmax = 255;
            }
            if(avgBmin < 0)
            {
                        avgBmin = 0;
            }
            else if(avgBmax > 255)
            {
                        avgBmax = 255;
            }


            printf("Average MIN: R: %lf  G: %lf  B: %lf\n",avgRmin,avgGmin,avgBmin);
            printf("Average MAX: R: %lf  G: %lf  B: %lf\n",avgRmax,avgGmax,avgBmax);
            cvNamedWindow( "Result", 0 );
            cvResizeWindow( "Result", 352, 288 );
            //------------------------------------------------------------------------------------------


while(1)
            {

                        //Centroid calculation variables
                        hsum=0;
                        wsum=0;
                        havg=0;
                        wavg=0;
                        avg_counter=0;
                        count = count +1;

                        // capture
                        cvGrabFrame( capture );
  Iplimage = cvRetrieveFrame( capture );
                        cvFillImage(ImgResultR, 255);
                        cvFillImage(ImgResultG, 255);
                        cvFillImage(ImgResultB, 255);

                        //-----------------------------------------------------------------//
                        //preparing separate RGB image


                        cvSmooth(Iplimage, ImgResultM, CV_MEDIAN,5,5,5);
                        cvCvtPixToPlane(Iplimage, IplBlue,IplGreen, IplRed,NULL);

                        for(int h=1; h<sourceSize.height; h++)
                                    //for(int h=sourceSize.height; h<0; h--)
  {
                                    for(int w=0; w<sourceSize.width; w++)
                                    {
                                                //memo : you can make arrary for saving pixel data instead of pxlRed …
                                                pxlRed   = cvGetReal2D(IplRed     , h ,w);
```

```
                                        pxlGreen = cvGetReal2D(IplGreen   , h ,w);
                                        pxlBlue  = cvGetReal2D(IplBlue    , h ,w);

                                        //if ((pxlRed>=170) && (pxlGreen <=140) && (pxlGreen >= 40) &&
(pxlBlue <= 120)) //replace fix value with the training frame
                                        if ((pxlRed>=avgRmin) && (pxlRed<=avgRmax) && (pxlGreen
<=avgGmax) && (pxlGreen >= avgGmin) && (pxlBlue <= avgBmax) && (pxlBlue>=avgBmin))
                                        {
                                                ((uchar*)(ImgResultR->imageData + ImgResultR-
>widthStep*(sourceSize.height-h)))[w] = (char)(255);
                                                ((uchar*)(ImgResultG->imageData + ImgResultG-
>widthStep*(sourceSize.height-h)))[w] = (char)(150);
                                                ((uchar*)(ImgResultB->imageData + ImgResultB-
>widthStep*(sourceSize.height-h)))[w] = (char)(0);
                                                hsum=hsum+h;
                                                wsum=wsum+w;
                                                avg_counter++;
                                        }
                                }
    }
                        cvCvtPlaneToPix(ImgResultB, ImgResultG, ImgResultR, NULL, ImgResultM);
                        //cvSmooth(ImgResultM, ImgResult, CV_MEDIAN,5,5);
                        cvShowImage( "Result", ImgResultM );
                        if( cvWaitKey(10) >= 0 )
      break;

                        //Compute the Centriod
                        if (avg_counter!=0)
                        {
                                havg=(hsum/avg_counter);
                                wavg=(wsum/avg_counter);
                        }
                        else
                        {
                                havg=0;
                                wavg=0;
                        }


                        //Decision making for how much turn effort
                        //1= steer_hardLeft
                        //2= steer_left
                        //3= go straight
                        //4= steer_right
                        //5= steer_hardRight
                        //6= fault --Buzzer

                        if (wavg>=1 && wavg<=77)
                        {
                                output=1;
                        }
                        else if (wavg>=78 && wavg<=155)
                        {
                                output=2;
                        }
                        else if (wavg>=156 && wavg<=196)
                        {
                                output=3;
                        }
                        else if (wavg>=197 && wavg<=274)
                        {
                                output=4;
                        }
                        else if (wavg>=275 && wavg<=352)
                        {
                                output=5;
                        }
                        else if (wavg==0)
                        {
                                output=6;
```

```
				}

				//------------------------------------------------------------------------
				//Transmit to Serial Port
				;
				char buf[50];
	sprintf(buf,"%c\n\r",(char)output);
				WriteFile(hSerial,buf,strlen(buf), &dwBytesWritten,&ovWrite);




				//------------------------------------------------------------------------
				//Median Filtering

				printf("CenHeight = %d, CenWidth= %d\n",havg, wavg);

				//cvShowImage( "BossVision", Iplimage );

				//cvShowImage( "Filtered Result", ImgResultM );

				/*if(count == 300)
				{
				cvSaveImage("orange image.jpg",ImgResult);
				}

				*/


	//if( cvWaitKey(10) >= 0 )
				// break;

				//


   }

		CloseHandle(hSerial);
		//Release memory and ETC
   cvReleaseCapture( &capture );
		cvReleaseImage( &ImgResult);
		cvReleaseImage( &ImgResultM);
		cvReleaseImage( &ImgResultR);
		cvReleaseImage( &ImgResultG);
		cvReleaseImage( &ImgResultB);
		cvReleaseImage( &Iplimage);
		cvReleaseImage( &IplRed);
		cvReleaseImage( &IplGreen);
		cvReleaseImage( &IplBlue);
   cvDestroyWindow( "BossVision" );
		cvDestroyWindow( "Result" );
		cvDestroyWindow( "Filtered Result");

		return 0;

}

BOOL SetCommDefaults(HANDLE hSerial)
{
 DCB dcb;
 memset(&dcb,0,sizeof(dcb));
 dcb.DCBlength=sizeof(dcb);
 if (!GetCommState(hSerial,&dcb))
  return FALSE;
 dcb.BaudRate=38400;
 dcb.ByteSize=8 ;
 dcb.Parity=0;
 dcb.StopBits=ONESTOPBIT;
```

```
 if (!SetCommState(hSerial,&dcb))
  return FALSE;
 return TRUE;
}
```

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "cv.h"
#include "cvcam.h"
#include "highgui.h"
#include "cxcore.h"
#include "windows.h"
#include "time.h"
#include "string.h"
#include "stdio.h"

BOOL SetCommDefaults(HANDLE hSerial);
OVERLAPPED ovWrite;



double Rmin=255;
double Rmax=0;
double Gmin=255;
double Gmax=0;
double Bmax=255;
double Bmin=0;
double maxR[100]={0};
double maxG[100]={0};
double maxB[100]={0};
double minR[100]={0};
double minG[100]={0};
double minB[100]={0};
double avgRmax=0;
double avgGmax=0;
double avgBmax=0;
double avgRmin=0;
double avgGmin=0;
double avgBmin=0;
int matcnt=0;
int hsum = 0;
int wsum = 0;
int havg = 0;
int wavg = 0;
int avg_counter = 0;
char output;
int count = 0;

double pxlRed = 0;
double pxlGreen = 0;
double pxlBlue = 0;
long double redCount = 0;
```

```
long double greenCount = 0;
long double blueCount = 0;
long double avgRed = 0;
long double avgGreen = 0;
long double avgBlue = 0;

IplImage* ImgResult = 0;
IplImage* ImgResultM = 0;
IplImage* ImgResultR = 0;
IplImage* ImgResultG = 0;
IplImage* ImgResultB = 0;
//check source image size
//preparing separat RGB image


int main(int argc, char* argv[])
{
char counter = 0;
//serial comm stuff
HANDLE hSerial = CreateFile("COM1", GENERIC_READ |
GENERIC_WRITE,0,NULL,OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_OVERLAPPED | FILE_FLAG_NO_BUFFERING,NULL);
SetCommDefaults(hSerial);
DWORD dwBytesWritten = 0;
memset(&ovWrite,0,sizeof(ovWrite));




//

/*      //serail stuff -----------------
        int            c;
    int            erreur;
    char           txt[32];
    Tserial_event *com;
        com = new Tserial_event();
        com->setManager(SerialEventManager);
        erreur = com->connect("COM1", 19200, SERIAL_PARITY_NONE, 8,
true);
*/      //end serial stuff--------------

    IplImage* Iplimage = 0;
        CvCapture* capture = cvCaptureFromCAM(0); // NEED one time call
        //cvNamedWindow( "BossVision", 0 );
        //cvResizeWindow( "BossVision", 352, 288 );

        //cvNamedWindow( "Filtered Result", 0 );
        //cvResizeWindow( "Filtered Result", 352, 288 );





        //one time routine for checking image size
        cvGrabFrame( capture );
    Iplimage = cvRetrieveFrame( capture );
```

```
        //check source image size
        CvSize sourceSize;
        sourceSize=cvGetSize(Iplimage);
        printf("Source image size is width= %d, height = %d,
\n",sourceSize.height, sourceSize.width);


        CvSize szResult;
        szResult.height = sourceSize.height;
        szResult.width  = sourceSize.width;

        //Creating Image Structured Objects
        ImgResult = cvCreateImage(szResult, 8, 3); //create RGB image
        ImgResultM = cvCreateImage(szResult, 8, 3); //create RGB image
        ImgResultR = cvCreateImage(szResult, 8, 1); //create gray image
        ImgResultG = cvCreateImage(szResult, 8, 1); //create gray image
        ImgResultB = cvCreateImage(szResult, 8, 1); //create gray image
        IplImage* IplRed     = cvCreateImage(cvGetSize(Iplimage),
IPL_DEPTH_8U,1);
        IplImage* IplGreen   = cvCreateImage(cvGetSize(Iplimage),
IPL_DEPTH_8U,1);
        IplImage* IplBlue    = cvCreateImage(cvGetSize(Iplimage),
IPL_DEPTH_8U,1);


        cvFillImage(ImgResultR, 255);
        cvFillImage(ImgResultG, 255);
        cvFillImage(ImgResultB, 255);
        ////////////////////////////////////////////////////////////
        //-------------------------------------------------------------
------------------------
        //Training Image procedure


        for(matcnt=0;matcnt<100;matcnt++)
        {


                //capture image
                cvGrabFrame( capture );
        Iplimage = cvRetrieveFrame( capture );
                //cvFillImage(ImgResultR, 255);
                //cvFillImage(ImgResultG, 255);
                //cvFillImage(ImgResultB, 255);



                cvCvtPixToPlane(Iplimage, IplBlue,IplGreen,
IplRed,NULL);


                for(int h=0; h<sourceSize.height; h++)
                        //for(int h=sourceSize.height; h<0; h--)
        {
```

```
                        for(int w=0; w<sourceSize.width; w++)
                        {
                                //memo : you can make arrary for saving
pixel data instead of pxlRed …
                                pxlRed   = cvGetReal2D(IplRed     , h
,w);
                                pxlGreen = cvGetReal2D(IplGreen   , h
,w);
                                pxlBlue  = cvGetReal2D(IplBlue    , h
,w);

                                redCount   += pxlRed;
                                greenCount += pxlGreen;
                                blueCount  += pxlBlue;
                        }
        }
        }

        avgRed  = (redCount)/(100*288*352);
        avgGreen= (greenCount)/(100*288*352);
        avgBlue = (blueCount)/(100*288*352);

        avgRmin = avgRed - 30;
        avgGmin = avgGreen - 30;
        avgBmin = avgBlue - 30;
        avgRmax = avgRed + 30;
        avgGmax = avgGreen + 30;
        avgBmax = avgBlue + 30;

        if(avgRmin < 0)
        {
                avgRmin = 0;
        }
        else if(avgRmax > 255)
        {
                avgRmax = 255;
        }
        if(avgGmin < 0)
        {
                avgGmin = 0;
        }
        else if(avgGmax > 255)
        {
                avgGmax = 255;
        }
        if(avgBmin < 0)
        {
                avgBmin = 0;
        }
        else if(avgBmax > 255)
        {
                avgBmax = 255;
        }


        printf("Average MIN: R: %lf  G: %lf  B:
%lf\n",avgRmin,avgGmin,avgBmin);
```

```
        printf("Average MAX: R: %lf  G: %lf  B: 
%lf\n",avgRmax,avgGmax,avgBmax);
        cvNamedWindow( "Result", 0 );
        cvResizeWindow( "Result", 352, 288 );
        //------------------------------------------------------------
-------------------------


    while(1)
        {

                //Centroid calculation variables
                hsum=0;
                wsum=0;
                havg=0;
                wavg=0;
                avg_counter=0;
                count = count +1;

                // capture
                cvGrabFrame( capture );
        Iplimage = cvRetrieveFrame( capture );
                cvFillImage(ImgResultR, 255);
                cvFillImage(ImgResultG, 255);
                cvFillImage(ImgResultB, 255);

                //-------------------------------------------------------
----------//
                //preparing separate RGB image


                cvSmooth(Iplimage, ImgResultM, CV_MEDIAN,5,5,5);
                cvCvtPixToPlane(Iplimage, IplBlue,IplGreen, 
IplRed,NULL);

                for(int h=1; h<sourceSize.height; h++)
                        //for(int h=sourceSize.height; h<0; h--)
        {
                        for(int w=0; w<sourceSize.width; w++)
                        {
                                //memo : you can make arrary for saving 
pixel data instead of pxlRed …
                                pxlRed   = cvGetReal2D(IplRed     , h 
,w);
                                pxlGreen = cvGetReal2D(IplGreen   , h 
,w);
                                pxlBlue  = cvGetReal2D(IplBlue    , h 
,w);

                                //if ((pxlRed>=170) && (pxlGreen <=140) 
&& (pxlGreen >= 40) && (pxlBlue <= 120)) //replace fix value with the 
training frame
                                if ((pxlRed>=avgRmin) && 
(pxlRed<=avgRmax) && (pxlGreen <=avgGmax) && (pxlGreen >= avgGmin) && 
(pxlBlue <= avgBmax) && (pxlBlue>=avgBmin))
                                {
```

```
                                        ((uchar*)(ImgResultR->imageData +
ImgResultR->widthStep*(sourceSize.height-h)))[w] = (char)(255);
                                        ((uchar*)(ImgResultG->imageData +
ImgResultG->widthStep*(sourceSize.height-h)))[w] = (char)(150);
                                        ((uchar*)(ImgResultB->imageData +
ImgResultB->widthStep*(sourceSize.height-h)))[w] = (char)(0);
                                        hsum=hsum+h;
                                        wsum=wsum+w;
                                        avg_counter++;
                                }
                        }
        }
                cvCvtPlaneToPix(ImgResultB, ImgResultG, ImgResultR,
NULL, ImgResultM);
                //cvSmooth(ImgResultM, ImgResult, CV_MEDIAN,5,5);
                cvShowImage( "Result", ImgResultM );
                if( cvWaitKey(10) >= 0 )
            break;

                //Compute the Centriod
                if (avg_counter!=0)
                {
                        havg=(hsum/avg_counter);
                        wavg=(wsum/avg_counter);
                }
                else
                {
                        havg=0;
                        wavg=0;
                }


                //Decision making for how much turn effort
                //1= steer_hardLeft
                //2= steer_left
                //3= go straight
                //4= steer_right
                //5= steer_hardRight
                //6= fault --Buzzer

                if (wavg>=1 && wavg<=77)
                {
                        output=1;
                }
                else if (wavg>=78 && wavg<=155)
                {
                        output=2;
                }
                else if (wavg>=156 && wavg<=196)
                {
                        output=3;
                }
                else if (wavg>=197 && wavg<=274)
                {
                        output=4;
                }
                else if (wavg>=275 && wavg<=352)
```

```
		{
			output=5;
		}
		else if (wavg==0)
		{
			output=6;
		}

		//-------------------------------------------------------
-------------------
		//Transmit to Serial Port
		;
		char buf[50];
	sprintf(buf,"%c\n\r",(char)output);
		WriteFile(hSerial,buf,strlen(buf),
&dwBytesWritten,&ovWrite);




		//-------------------------------------------------------
--------------------
		//Median Filtering

		printf("CenHeight = %d, CenWidth= %d\n",havg, wavg);

		//cvShowImage( "BossVision", Iplimage );

		//cvShowImage( "Filtered Result", ImgResultM );

		/*if(count == 300)
		{
		cvSaveImage("orange image.jpg",ImgResult);
		}

		*/


	//if( cvWaitKey(10) >= 0 )
		// break;

		//


    }

	CloseHandle(hSerial);
	//Release memory and ETC
    cvReleaseCapture( &capture );
	cvReleaseImage( &ImgResult);
	cvReleaseImage( &ImgResultM);
	cvReleaseImage( &ImgResultR);
	cvReleaseImage( &ImgResultG);
	cvReleaseImage( &ImgResultB);
	cvReleaseImage( &Iplimage);
```

```
        cvReleaseImage( &IplRed);
        cvReleaseImage( &IplGreen);
        cvReleaseImage( &IplBlue);
    cvDestroyWindow( "BossVision" );
        cvDestroyWindow( "Result" );
        cvDestroyWindow( "Filtered Result");

        return 0;

}

BOOL SetCommDefaults(HANDLE hSerial)
{
 DCB dcb;
 memset(&dcb,0,sizeof(dcb));
 dcb.DCBlength=sizeof(dcb);
 if (!GetCommState(hSerial,&dcb))
  return FALSE;
 dcb.BaudRate=38400;
 dcb.ByteSize=8 ;
 dcb.Parity=0;
 dcb.StopBits=ONESTOPBIT;
 if (!SetCommState(hSerial,&dcb))
  return FALSE;
 return TRUE;
}
```