Final Report

Name: Lee Tsakh
Robot: Ball-E
Course: EEL5666C IMDL
Instructors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz
TAs: Ryan Stevens, Tim Martin

Table of Contents

Abstract

This document describes the purpose and implementation of a tennis ball launching robot named Ball-E. This robot is designed to launch tennis balls at certain targets. It does this through the use of the Epiphany DIY board as its main control platform. An IP Cam, sonar sensor, and two IR sensors are employed to make Ball-E aware of its surroundings. Ball-E launches tennis balls using a high-RPM motor. Ball-E is also able to engage in obstacle avoidance while it searches for a target and can indicate a successful launch once it has fired a ball.

Introduction

Tennis balls machines are a popular device used by many tennis players to practice their game and improve their strokes. One of the most difficult shots to hit is a volley when the player is close to the net and the ball is headed straight for the volleying player. Ball-E is designed to simulate this event by finding a target person and launching a tennis ball straight at him or her. A person could use Ball-E to practice this very difficult shot in an indoor environment.

Integrated System

Ball-E is based around the Epiphany DIY board developed by Tim Martin. This board is the main control platform for the robot. Figure 1 below shows the basic flowchart for Ball-E. The figure shows the operation of the robot after it is started up.

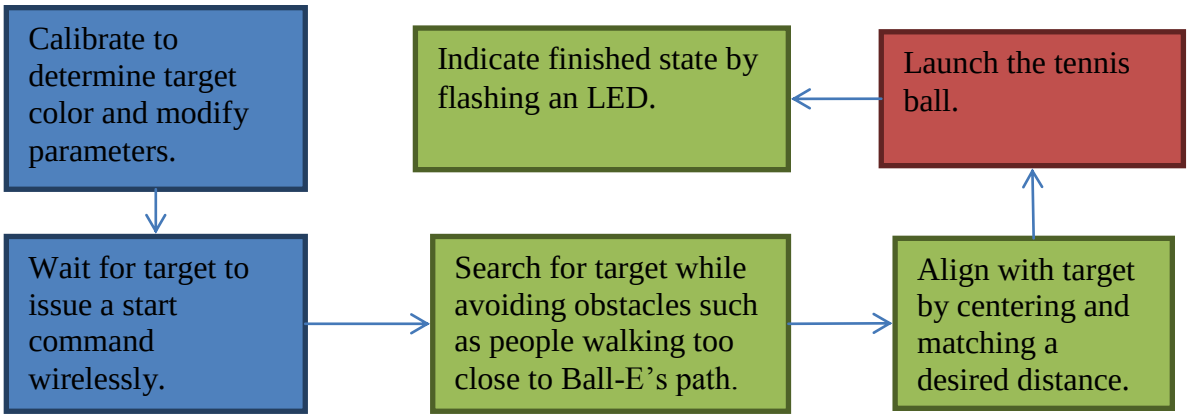

Figure 1: Basic control flowchart for Ball-E

Ball-E includes a calibration routine. This routine first determines the appropriate color of the target for its current lighting environment, and it also outputs debug values so that the user can ensure proper sensor calibration.

Additionally, Ball-E will execute obstacle avoidance during its searching routine so that user(s) can walk around it safely. If a target is detected during the searching routine, Ball-E will transition to an alignment routine before firing at the target. Ball-E's systems will be based on a

state machine paradigm similar to the one traditionally used with FPGAs. Different behavioral actions and sensor responses will be associated with different states.

Vision processing is accomplished using the OpenCV library on a laptop. In order to send the processed data to Ball-E, XBEEs are used. Additionally, during the calibration routine, Ball-E sends its sensor reading to the computer using XBEE. These readings can be used to adjust the defines in the C code for the sensors based on the environment.

Mobile Platform

The platform for Ball-E is made to accomplish the task of housing its control systems, actuation devices, and sensors. It is designed to be robust enough to remain steady while launching the tennis balls. The structural layout is set up to facilitate the flow of the tennis balls from the hopper, through the launcher, and out toward the target.

Figure 2 below shows a rough sketch of the platform as viewed from the side. The figure shows the curved hopper, motor battery, sensor placement, and wheels. The platform uses a space frame design for maximum rigidity, with a top and bottom panel made of plywood for mounting of components. In early tests, the platform was shown to support a 170 pound person.

Figure 2: Image of the Platform

Actuation

The actuation systems for Ball-E are responsible for tennis balls and moving around Ball-E's environment. For the first task, a relatively powerful scooter motor is activated to spin a wheel that can propel a tennis ball out of the launcher tube. The circuit used to activate this motor from the Epiphany DIY board is shown below in Figure 3. Resistor R2 is used to represent the coil of a relay. The 3.3V input going to the base of Q1 is from a GPIO pin on the Atxmega. The relay is connected to the motor using the normally open pins, so that when the transistor is turned on by receiving a logical high from the microprocessor, the motor will spool up.

The second task of moving Ball-E is accomplished using two drive wheels and a caster in the back. Since Ball-E is relatively heavy, the drive motors are set near their maximum for every movement.

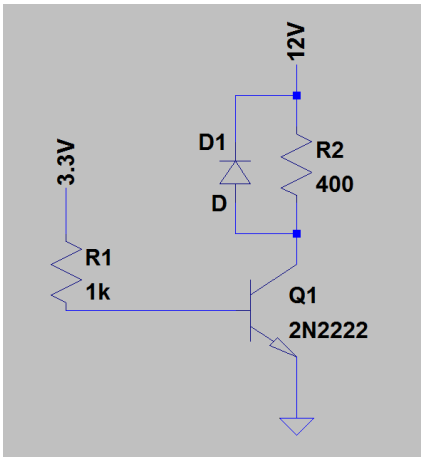

Figure 3: Motor Activation Circuit

Additionally, a solenoid is employed to drop a tennis ball from the hopper onto the spinning wheel when Ball-E is ready to launch. The solenoid is mounted in a "normally-out" position so that impedes the movement of tennis balls into the launch tube. It was experimentally determined that 100 milliseconds were required to allow a single ball to pass into the launcher without letting the rest fall.

Sensors

Ball-E requires several sensors to complete its goals. It uses an IP Cam in order to locate and lock onto objects that are the shape and color of its target. Sonar is employed to determine the distance between Ball-E and the target or obstacles. IR sensors inform Ball-E if obstacles are in close proximity to the sides of Ball-E so that it can avoid them.

Originally, Ball-E was going to use a CMUcam for the vision system, but there were no units available. The IP Cam operational paradigm is as follows:

1. Allow enough time for the IP Cam to connect to wireless network before activating calibration routine.
2. On a host computer, grab a frame from the IP Camera over an ad-hoc network set up between the IP Camera and the host computer.
3. Using OpenCV, find the center of a color blob as chosen by the user.
4. Send to Ball-E whether the center of the blob is to the left, right, or center of the frame. If the blob isn't present in the frame, then no data will be sent.
5. Go to step 2.

It was experimentally determined that an object was too close when Sonar values read less than 270 or IR values read more than 1000. These values are used in the current software

shown in the appendix. Also included in the appendix is the modified camshiftdemo file used for the IP Cam (written in C++) as well as the XBEE-related C code.

Behaviors

The main behavioral provisions that Ball-E has pertain to its wandering and firing routines. These behaviors allow it to execute basic obstacle avoidance and to lock onto a target, respectively. For the former, when an obstacle is detected using an IR sensor, Ball-E will turn away from the obstacle. When an obstacle is detected using the sonar sensor, Ball-E will move backward a short distance. When no obstacles are detected, it will move randomly. Using XBEE, Ball-E can receive an interrupt to its current routine at any time to switch its state.

For the firing routine, Ball-E tries to center itself on its target. Once the target is centerer (to within a certain margin), Ball-E will determine if it is at an appropriate range using the sonar sensor. If it is, it launches three tennis balls.

Conclusion

Ball-E's systems and components are designed to enable it to accomplish its primary task, as laid out in the introduction above. After testing, it was determined that Ball-E could hold three tennis balls comfortably in its hopper. Launching force was sufficient to propel a ball at least twenty feet horizontally, with high enough of an arc to clear the net on a tennis court. The color tracking was fairly sophisticated, though it suffered from certain limitations in OpenCV and limitations involving the ad-hoc network.

Appendix

```
/**
 * \file
 *
 * \brief Main Application
 *
 */

/*
 * Include header files for all drivers that have been imported from
 * AVR Software Framework (ASF).
 */
#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>
#include "motor.h"
#include "lcd.h"
```

```
#include "uart.h"
#include "RTC.h"
#include "picServo.h"
#include "ADC.h"
#include "sonar.h"
#include "xbee.h"

#define DbLedOn()         (PORTR.OUTCLR = 0x02)           //Turns the debug led on.
The led is connected with inverted logic
#define DbLedOff()        (PORTR.OUTSET = 0x02)           //Turns the debug led off.
The led is connected with inverted logic
#define DbLedToggle()     (PORTR.OUTTGL = 0x02)           //Toggles the debug led
off.  The led is connected with inverted logic
#define Relay_On()        (PORTD.OUTSET = 1 << 0)
#define Relay_Off()       (PORTD.OUTCLR = 1 << 0)
#define Relay_Init()      (PORTD.DIRSET = 1 << 0)

// New defines
#define SONAR_CHAN_OFFSET 0 //offset for the sonar ADC channel
#define SONAR_INPUT_OFFSET 0 //offset for the sonar ADC input pin
#define IR_CHAN_OFFSET 1 //offset for the IR channels (using channels 1 and 2)
#define IR_INPUT_OFFSET 1 //offset for the IR input pins (using pins 1 and 2)
#define IR_INPUT_SPACING 1 //just in case I separate the IR input pins, I can use
this
#define SONAR_TOO_CLOSE 350
#define IR_TOO_CLOSE 1000
#define ALIGN_LEFT 'L'
#define ALIGN_RIGHT 'R'
#define BALL_DROP_TIME 100
#define BALL_DROP_DELAY 500

// Function prototypes
int read_sonar(void);
int read_ir(int ir_number);
void wandering_routine(void);
void motor_jump_back(void);
void turn_robot(int degrees);
void aligning_routine(void);
void ranging_routine(void);
void FIRE(void);

// Enumerated type for state machine states
enum state_t {calibrating, wandering, aligning, ranging, firing, finished};
enum state_t state = calibrating;

int x = 100; //test variable
int rand_count = 0;
char aligning_value = 'X';

int main (void)
{
      board_init();/*This function originates in the file init.c, and is used to
initialize the Epiphany DIY
                                    motorInit() is declared within because by default
you the user should define what your
```

```
                                    motor setup is to prevent hurting the Epiphany.
You can do this by
                                  */
      RTC_DelayInit();//initializes the Real time clock this seems to actually take
an appreciable amount of time
      DbLedOn();   //I like to do this by default to show the board is no longer
suspended in the bootloader.
      servoControlInit();//initializes the servo module ***including enabling global
interrupts*** required for the servo control module
      uartInit(&USARTC0,57600);//as can be seen in the schematic.  This uart is
connected to the USB port.  This function initializes this uart
      stdout = &USB_str;//This function points stdout to the USB device.  So this
means printf will send data out the USB port.
      ADCsInits();//this function initializes the ADCs inside the Xmega
      stdout = &xbee_str;

      //you should add any further initializations here
      xbee_init();
      Relay_Init();


      DbLedOn();
      //setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
      //setMotorDuty(3,600,MOTOR_DIR_BACKWARD_gc);
      //setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
      //setMotorDuty(4,600,MOTOR_DIR_BACKWARD_gc);
      //while (1) {}


      int sonar_result, ir_result_0, ir_result_1 = 0;
      int wait_time = 50;
      while (1){
            switch (state)
            {
                  case calibrating:
                        //for now, just flash the LED while I set up the colors
                        // sonar
                        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
                        setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
                        sonar_result = read_sonar();
                        printf("Sonar = %d\r\n",sonar_result);


                        // IR
                        ir_result_0 = read_ir(0);
                        printf("IR0 = %d\r\n",ir_result_0);
                        ir_result_1 = read_ir(1);
                        printf("IR0 = %d\r\n",ir_result_1);
                        DbLedToggle();
                        RTC_Delay_ms(x);
                        while (delayNotOver) {}
                        break;
                  case firing:
                        RTC_Delay_ms(500);
                        while (delayNotOver) {}
```

```
                        Relay_On();
                        RTC_Delay_ms(3000);
                        while (delayNotOver) {}
                        Relay_Off();
                        state = finished;
                        x = 1000;
                        break;
                case aligning:
                        aligning_routine();
                        break;
                case wandering:
                        wandering_routine();
                        //printf("Hello again \r\n");
                        break;
                case ranging:
                        ranging_routine();
                        break;
                case finished:
                        DbLedToggle();
                        RTC_Delay_ms(x);
                        while (delayNotOver) {}
                        break;
                default:
                        // Insert code for failure state here (flash debug LED
fast)
                        break;
        }
    }
}

int read_sonar(void)
{
    int sonar_output = 0;
    ADCA_request(SONAR_CHAN_OFFSET,SONAR_INPUT_OFFSET);
    sonar_output = ADCA_getVal(SONAR_CHAN_OFFSET);
    return sonar_output;
    //printf("Sonar = %d\r\n",sonar_output);
}

int read_ir(int ir_number)
{
    int ir_output = 0;
    ADCA_request(IR_CHAN_OFFSET + ir_number, IR_INPUT_OFFSET + ir_number);
    ir_output = ADCA_getVal(IR_CHAN_OFFSET + ir_number);
    return ir_output;
    //printf("IR = %d\r\n", ir_output);
}

void wandering_routine(void)
{
    int sonar_result, ir_result_0, ir_result_1 = 0;
    int wait_time = 50;
    // sonar
    sonar_result = read_sonar();
    //printf("Sonar = %d\r\n",sonar_result);
```

```
	////printf("Sonar = %d\r\n",sonar_result);

	// IR
	ir_result_0 = read_ir(0);
	////printf("IR = %d\r\n",ir_result_0);
	ir_result_1 = read_ir(1);
	////printf("%d\r\n", ir_result_1);
	////printf("IR2 = %d\r\b",ir_result_1);

	if (sonar_result  <= SONAR_TOO_CLOSE) // check if something is too close to
the front sensor
	{
		motor_jump_back();
		//printf("Sonar! \r\n");
		wait_time = 2*wait_time;
	}
	else if (ir_result_0 >= IR_TOO_CLOSE) // check if something too close to
left_ir
	{
		turn_robot(-30);
		//printf("left_ir!\r\n");
	}
	else if (ir_result_1 >= IR_TOO_CLOSE) // check if something too close top
right_ir
	{
		turn_robot(30);
		//printf("right_ir!\r\n");
		//printf("%d\r\n", ir_result_1);
	}
	else // no obstacle avoidance needed, maybe I should expand this to move
forward sometimes
	{
		////actually random stuff would use srand(ir_result0) and rand()
		//if (ir_result_0 > ir_result_1) // relatively random
		//{
			//turn_robot(-30);
			////printf("random1!\r\n");
		//}
		//else
		//{
			//turn_robot(30);
			////printf("random2!\r\n");
		//}

		if (rand_count >= 10)
		{
			rand_count = 0;
			turn_robot(60);
		}
		else
		{
			setMotorDuty(1,1000,MOTOR_DIR_FORWARD_gc);
			setMotorDuty(3,1000,MOTOR_DIR_FORWARD_gc);
			//RTC_Delay_ms(1000);
			RTC_Delay_ms(100);
```

```
                    while (delayNotOver) {
                    }
                    rand_count += 1;
              }
              //setMotorDuty(1,0,MOTOR_DIR_BACKWARD_gc);
              //setMotorDuty(3,0,MOTOR_DIR_BACKWARD_gc);
              //wait_time = 2*wait_time;

       }

       DbLedToggle();
       RTC_Delay_ms(wait_time);
       while (delayNotOver) {
       }
}

void aligning_routine(void)
{
       int wait_time = 50;
       if (aligning_value == ALIGN_LEFT)
       {
              turn_robot(30);
       }
       else if (aligning_value == ALIGN_RIGHT)
       {
              turn_robot(-30);
       }
       DbLedToggle();
       RTC_Delay_ms(wait_time);
       while (delayNotOver) {
       }
}

void ranging_routine(void)
{
       int sonar_result = 0;
       int wait_time = 50;
       sonar_result = read_sonar();
       if (sonar_result <= 425) //make this a define!
       {
              motor_jump_back();
              DbLedToggle();
              RTC_Delay_ms(wait_time);
              while (delayNotOver) {}
       }
       else
       {
              setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
              setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
              state = firing;
       }

}

void motor_jump_back(void)
```

```
{
	setMotorDuty(1,1000,MOTOR_DIR_BACKWARD_gc);
	setMotorDuty(3,1000,MOTOR_DIR_BACKWARD_gc);
	RTC_Delay_ms(100);
	while (delayNotOver) {
	}
	//setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
	//setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
}

void turn_robot(int degrees)
{
	double abs_degrees = (double) degrees;
	if (degrees >= 0) {
		setMotorDuty(1,1000,MOTOR_DIR_FORWARD_gc);
		setMotorDuty(3,1000,MOTOR_DIR_BACKWARD_gc);
	}
	else
	{
		abs_degrees = -abs_degrees;
		setMotorDuty(1,1000,MOTOR_DIR_BACKWARD_gc);
		setMotorDuty(3,1000,MOTOR_DIR_FORWARD_gc);
	}
	RTC_Delay_ms((abs_degrees/360)*500);
	//RTC_Delay_ms(500);
	while (delayNotOver) {
	}
	//setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
	//setMotorDuty(3,0,MOTOR_DIR_FORWARD_gc);
}

void FIRE(void)
{
	setMotorDuty(2,1023,MOTOR_DIR_FORWARD_gc);
	RTC_Delay_ms(BALL_DROP_TIME);
	while (delayNotOver){}
	setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
	RTC_Delay_ms(BALL_DROP_DELAY);
	while (delayNotOver){}
}

ISR(XBEE_RXC_vect) //basic ISR code
{
	cli(); //mask all interrupts
	//uint16_t temp = stdout; //save the current stream
	//stdout = &xbee_str;
	char c = XBEE_USART.DATA; //get the character
	switch (state)
	{
		case calibrating:
			if (c =='S')
			{
				state = wandering;
			}
			else if (c == 'Z')
```

```
                {
                        FIRE();
                        FIRE();
                        FIRE();
                }
                break;
        case firing:
                break;
        case wandering:
                if (c == 'L' || c == 'R')
                {
                        aligning_value = c;
                        state = aligning;
                }
                else if (c == 'F')
                {
                        state = ranging;
                }
                break;
        case ranging:
                break;
        case finished:
                break;
        case aligning:
                if (c == 'F')
                {
                        state = ranging;
                }
                else if (c == 'L' || c == 'R')
                {
                        aligning_value = c;
                }
                break;
        default:
                // Insert code for failure state here (flash debug LED fast)
                break;
    }
    if (c == 'X') //stop robot and go back to calibrating
    {
            state = calibrating;
    }
    //stdout = temp; //replace the stream
    sei(); //enable interrupts

}




/*
 * xbee.c
 *
 * Created: 11/1/2011 3:18:05 PM
 *  Author: Lee
 */
```

```
#include "xbee.h"

FILE xbee_str = FDEV_SETUP_STREAM(xbee_putchar, NULL, _FDEV_SETUP_WRITE);

void xbee_init(void)
{
	cli(); //interrupts must be disabled during initialization according to AVR
manual
	XBEE_PORT.DIRSET = TX1_bm; //set TX pin to output
	XBEE_PORT.DIRCLR = RX1_bm; //set RX pin to input
	XBEE_PORT.OUTSET = TX1_bm; //set TX pin high

	XBEE_USART.CTRLA |= USART_RXCINTLVL_MED_gc; //set interrupt level to medium

	//set baud rate to 9600 (the default rate used by XBEE)
	#if (F_CPU == 32000000) // frequency = 32 MHz
		XBEE_USART.BAUDCTRLA = 3317;
		XBEE_USART.BAUDCTRLB = 0xC0 | (uint16_t)(0x0FFF & 3317>>8);
	#else //frequency = default 2 MHz
		XBEE_USART.BAUDCTRLA = 1539;
		XBEE_USART.BAUDCTRLB = 0x9 | (uint16_t)(0x0FFF & 1539>>8);
	#endif

	XBEE_USART.CTRLC = USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc |
USART_CHSIZE_8BIT_gc; //asynch, no parity, 8 bits data
	XBEE_USART.CTRLC &= ~(USART_SBMODE_bm); //1 stop bit, (this line is probably
not needed, since the default is 1 stop bit
	XBEE_USART.CTRLB = USART_RXEN_bm | USART_TXEN_bm; //enable TX and RX

	sei(); //enable interrupts


}

void xbee_putchar(char c, FILE *unused)
{
	while (!(XBEE_USART.STATUS &  USART_DREIF_bm));
	XBEE_USART.DATA = c;
}

/*
 * xbee.h
 *
 * Created: 11/1/2011 3:08:05 PM
 *  Author: Lee
 */

#ifndef XBEE_H
#define XBEE_H

#include <avr/io.h>
#include <asf/common/boards/user_board/user_board.h>
#include <asf.h>
```

```
#define XBEE_USART USARTE1
#define XBEE_PORT PORTE
#define XBEE_RXC_vect USARTE1_RXC_vect
#define XBEE_DRE_vect USARTE1_RXC_vect
#define TX1_bm (1 << 7) //bit mask for TX on USART1
#define RX1_bm (1 << 6) //bit mask for RX on USART1
#define XCK1_bm (1 << 5) //bit mask for XCK on USART1

void xbee_init(void);
void xbee_putchar(char c, FILE *unused);

FILE xbee_str;
#endif


// ColorTrack_Serial.cpp : main project file.

#include "stdafx.h"
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <ctype.h>

using namespace cv;
using namespace std;
using namespace System;
using namespace System::IO::Ports;

void help()
{
       cout << "\nThis is a demo that shows mean-shift based tracking\n"
               <<   "You select a color objects such as your face and it tracks it.\n"
               <<   "This reads from video camera (0 by default, or the camera number
the user enters\n"
               << "Call:\n"
               << "\n./camshiftdemo [camera number]"
               << "\n" << endl;

       cout << "\n\nHot 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"
         "\tp - pause video\n"
         "To initialize tracking, select the object with mouse\n" << endl;
}

Mat image, image_unmod; //image_unmod added for debug

bool backprojMode = false;
bool selectObject = false;
int trackObject = 0;
bool showHist = true;
```

```
Point origin;
Rect selection;
int vmin = 10, vmax = 256, smin = 30;
int error_count = 0;
int error_happened;
int robot_started = 0;

void onMouse( int event, int x, int y, int, void* )
{
    if( selectObject )
    {
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);
        selection.width = std::abs(x - origin.x);
        selection.height = std::abs(y - origin.y);

        selection &= Rect(0, 0, image.cols, image.rows);
    }

    switch( event )
    {
    case CV_EVENT_LBUTTONDOWN:
        origin = Point(x,y);
        selection = Rect(x,y,0,0);
        selectObject = true;
        break;
    case CV_EVENT_LBUTTONUP:
        selectObject = false;
        if( selection.width > 0 && selection.height > 0 )
            trackObject = -1;
        break;
    }
}



int main( int argc, char** argv )
{
      SerialPort^ _serialPort;
      System::String^ portChosen = "COM4";
      VideoCapture cap;
    Rect trackWindow;
    RotatedRect trackBox;
      RotatedRect oldTrackBox; //debug
      bool load_ok = true;
    int hsize = 16;
    float hranges[] = {0,180};
    const float* phranges = hranges;
      //cap.open("http://192.168.137.196/videostream.cgi?user=admin&pwd=&resolution=
32&rate=0&a=.mjpg");

      if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])))
        cap.open(argc == 2 ? argv[1][0] - '0' : 0);
    else if( argc == 2 )
        cap.open(argv[1]);
```

```
    if( !cap.isOpened() )
    {
       help();
        cout << "***Could not initialize capturing...***\n";
        return 0;
    }

    help();

    namedWindow( "Histogram", 0 );
    namedWindow( "CamShift Demo", 0 );
    setMouseCallback( "CamShift Demo", onMouse, 0 );
    createTrackbar( "Vmin", "CamShift Demo", &vmin, 256, 0 );
    createTrackbar( "Vmax", "CamShift Demo", &vmax, 256, 0 );
    createTrackbar( "Smin", "CamShift Demo", &smin, 256, 0 );

    Mat frame, hsv, hue, mask, hist, histimg = Mat::zeros(200, 320, CV_8UC3),
backproj;
    bool paused = false;

    for(;;)
    {
        if( !paused )
        {
            cap >> frame;
            if( frame.empty() )
                break;
        }

        frame.copyTo(image);
             image_unmod = image;

        if( !paused )
        {
            cvtColor(image, hsv, CV_BGR2HSV);

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

                inRange(hsv, Scalar(0, smin, MIN(_vmin,_vmax)),
                        Scalar(180, 256, MAX(_vmin, _vmax)), mask);
                int ch[] = {0, 0};
                hue.create(hsv.size(), hsv.depth());
                mixChannels(&hsv, 1, &hue, 1, ch, 1);

                if( trackObject < 0 )
                {
                    Mat roi(hue, selection), maskroi(mask, selection);
                    calcHist(&roi, 1, 0, maskroi, hist, 1, &hsize, &phranges);
                    normalize(hist, hist, 0, 255, CV_MINMAX);

                    trackWindow = selection;
                    trackObject = 1;
```

```
                    histimg = Scalar::all(0);
                    int binW = histimg.cols / hsize;
                    Mat buf(1, hsize, CV_8UC3);
                    for( int i = 0; i < hsize; i++ )
                        buf.at<Vec3b>(i) = Vec3b(saturate_cast<uchar>(i*180./hsize),
255, 255);
                    cvtColor(buf, buf, CV_HSV2BGR);

                    for( int i = 0; i < hsize; i++ )
                    {
                        int val =
saturate_cast<int>(hist.at<float>(i)*histimg.rows/255);
                        rectangle( histimg, Point(i*binW,histimg.rows),
                                   Point((i+1)*binW,histimg.rows - val),
                                   Scalar(buf.at<Vec3b>(i)), -1, 8 );
                    }
                }
                          if (load_ok == true)
                          {
                                oldTrackBox = trackBox;
                          }
                calcBackProject(&hue, 1, 0, hist, backproj, &phranges);
                backproj &= mask;
                          try
                          {
                                if (error_happened) {
                                      trackWindow = selection;
                                }
                                RotatedRect trackBox = CamShift(backproj,
trackWindow, TermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ));
                                printf("x = %f\n",trackBox.center.x);
                                load_ok=true;
                                if( trackWindow.area() <= 1 )
                                {
                    int cols = backproj.cols, rows = backproj.rows, r = (MIN(cols,
rows) + 5)/6;
                    trackWindow = Rect(trackWindow.x - r, trackWindow.y - r,
                                       trackWindow.x + r, trackWindow.y + r) &
                                  Rect(0, 0, cols, rows);
                                }

                                if( backprojMode )
                                      cvtColor( backproj, image, CV_GRAY2BGR );
                                ellipse( image, trackBox, Scalar(0,0,255), 3, CV_AA
);
                                error_happened = 0;
                                error_count = 0;
                                if (robot_started)
                                {
                                      _serialPort->Open();
                                      if (trackBox.center.x > 370)
                                      {
                                            _serialPort->Write("R"); //Right
                                      }
```

```
                                        else if (trackBox.center.x < 350)
                                        {
                                                _serialPort->Write("L"); //Left
                                        }
                                        else
                                        {
                                                _serialPort->Write("F"); //Fire
                                        }
                                        _serialPort->Close();
                                }
                        }
                        catch(cv::Exception& e)
                        {
                                printf("Error number %d\n", error_count);
                                load_ok = false;
                                if( backprojMode )
                                        cvtColor( backproj, image, CV_GRAY2BGR );
                                ellipse( image, oldTrackBox, Scalar(0,0,255), 3,
CV_AA );
                                //image = image_unmod;
                                error_count = error_count + 1;
                                error_happened = 1;
                                //maybe put another serial command here
                        }
            }
        }
        else if( trackObject < 0 )
            paused = false;

        if( selectObject && selection.width > 0 && selection.height > 0 )
        {
            Mat roi(image, selection);
            bitwise_not(roi, roi);
        }

        imshow( "CamShift Demo", image );
        imshow( "Histogram", histimg );

        char c = (char)waitKey(10);
        if( c == 27 )
                    //if (robot_started)
                    //{
                    //    _serialPort->Close();
                    //}
            break;
        switch(c)
        {
        case 'b':
            backprojMode = !backprojMode;
            break;
        case 'c':
            trackObject = 0;
            histimg = Scalar::all(0);
            break;
        case 'h':
```

```
            showHist = !showHist;
            if( !showHist )
                destroyWindow( "Histogram" );
            else
                namedWindow( "Histogram", 1 );
            break;
        case 'p':
            paused = !paused;
            break;
             case 's':
                    robot_started = 1;
                    _serialPort = gcnew SerialPort(portChosen, 9600, Parity::None, 8,
StopBits::One);
                    _serialPort->Open();
                    _serialPort->Write("S");
                    _serialPort->Close();
                    break;
        default:
            ;
        }
    }

    return 0;
}
```