Your submission (especially hardcopy) should have the following information for a cover sheet.

|  |
| --- |
| Assignment #5  Course Number and Section: CS3376-502  Last Name: Eskridge  First Name: Gwendolynn  NETID: gce090020  Due Date and Time: 8:30pm on 28 April 2014  I have uploaded all the softcopy to elearning: Yes/No    I have completed: \_\_\_ % Comment to Grader: |

===============================================

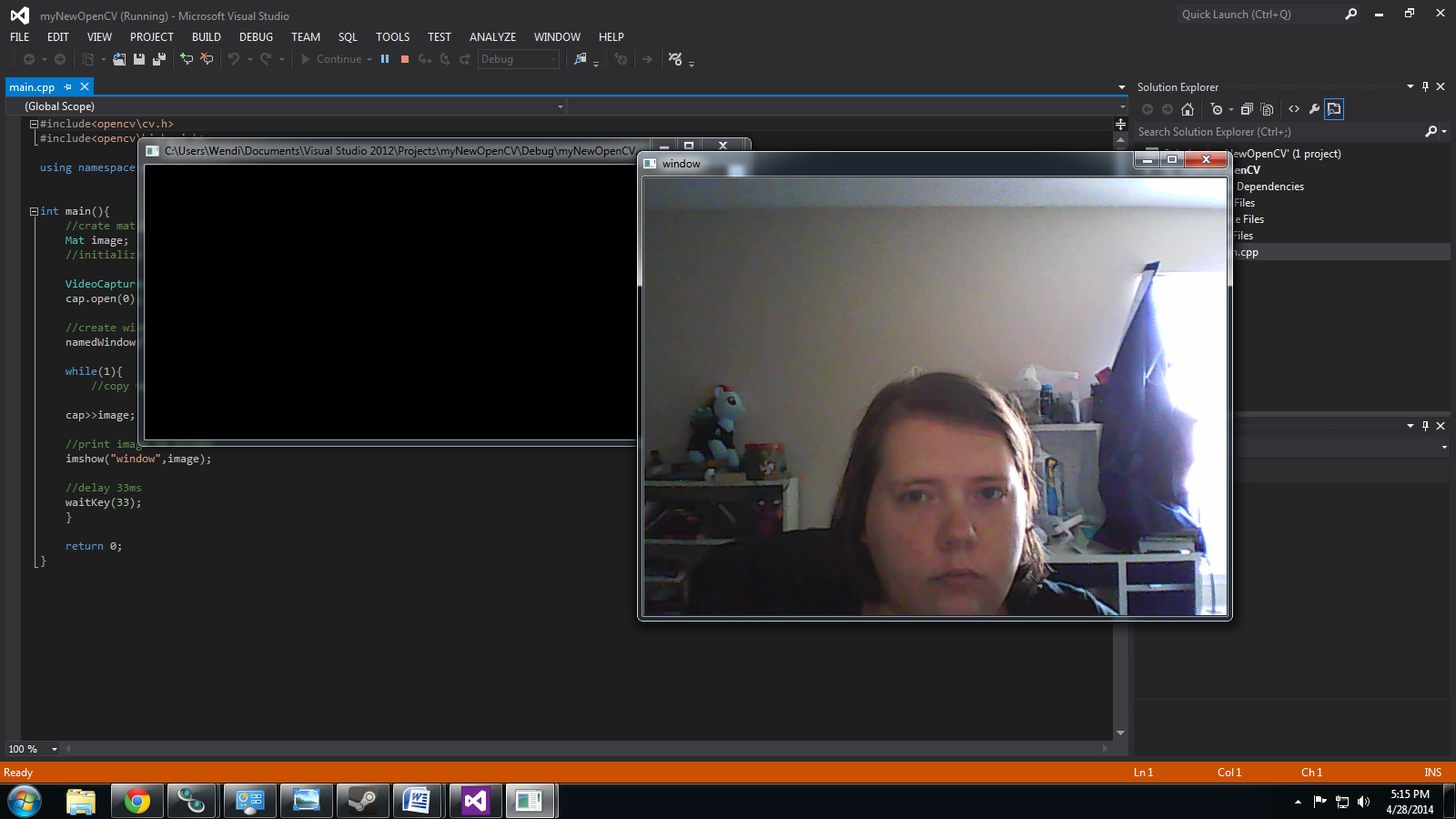
Note1. The Grade will be 100/100 max (including bonus).

Note2. Extra Point accumulated \_0\_ since last time report.

(You should keep a record of when and how you got for each extra point and write here).

# Part One:

## Installation and Sample Project



#include<opencv\cv.h>

#include<opencv\highgui.h>

using namespace cv;

int main(){

//crate matrix to store image

Mat image;

//initialize capture

VideoCapture cap;

cap.open(0);

//create window to show it

namedWindow("window", 1);

while(1){

//copy webcam stream to image

cap>>image;

//print image to screen

imshow("window",image);

//delay 33ms

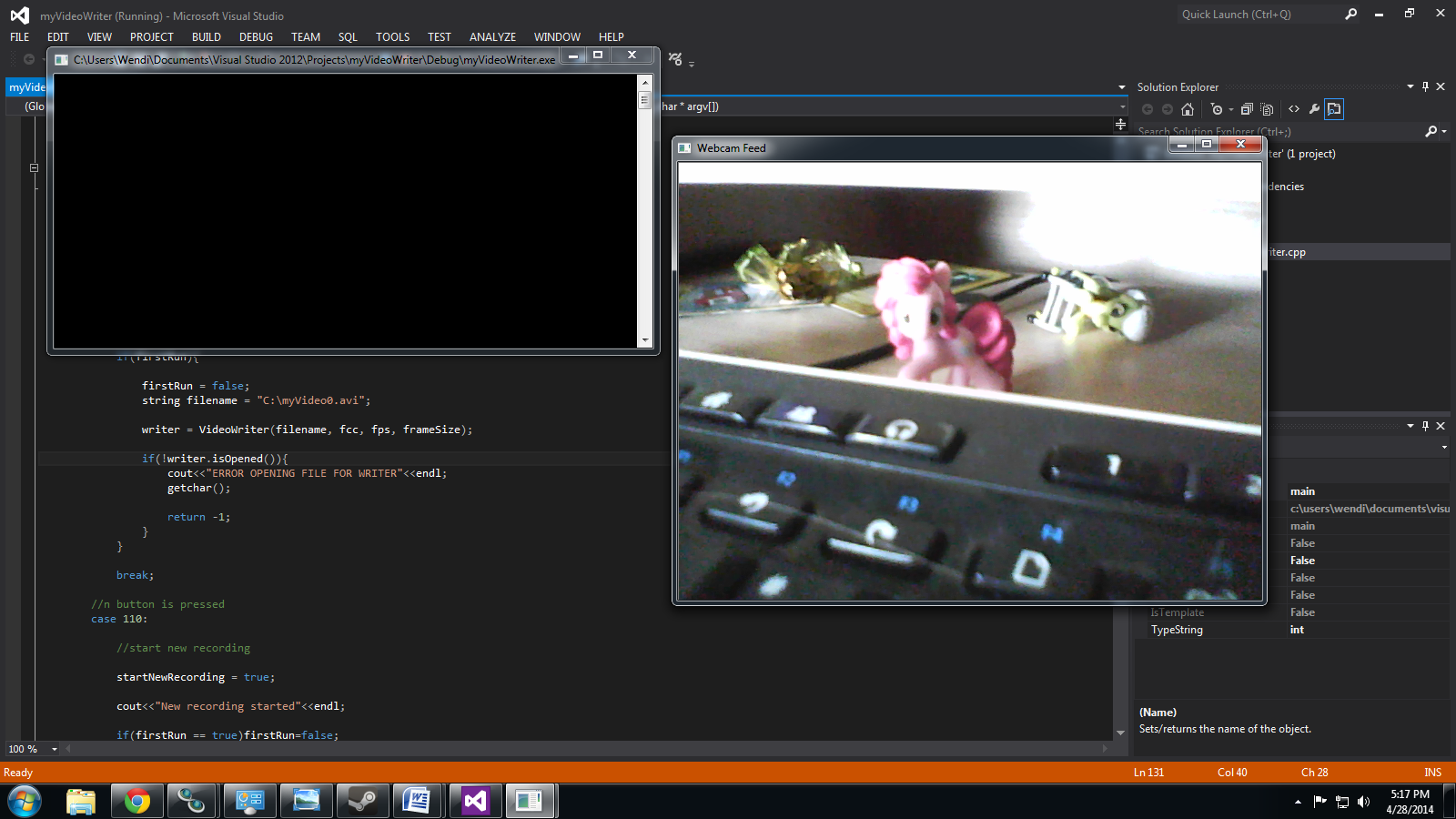
waitKey(33);

}

return 0;

}

## Writing a Video to a File:

#include <opencv\highgui.h>

#include <opencv\cv.h>

#include <iostream>

using namespace cv;

using namespace std;

string intToString(int number){

std::stringstream ss;

ss << number;

return ss.str();

}

int main(int argc, char\* argv[])

{

bool recording = false;

bool startNewRecording = false;

bool firstRun = true;

int videoNumber = 0;

VideoCapture cap(0); // open the video camera no. 0

cv::VideoWriter writer;

if (!cap.isOpened()) // if not success, exit program

{

cout << "ERROR INITIALIZING VIDEO CAPTURE" << endl;

return -1;

}

char\* windowName = "Webcam Feed";

namedWindow(windowName,CV\_WINDOW\_AUTOSIZE); //create a window to display our webcam feed

//filename string

//fourcc integer

int fcc = CV\_FOURCC('D', 'I', 'V', '3');

//frames per sec integer

int fps = 10;

//frame size

cv::Size frameSize(cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH), cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT));

Mat frame;

while (1) {

if(startNewRecording == true){

startNewRecording = false;

recording = true;

videoNumber++;

string filename = "C:\myVideo"+ intToString(videoNumber) + ".avi";

writer = VideoWriter(filename, fcc, fps, frameSize);

if(!writer.isOpened()){

cout<<"ERROR OPENING FILE FOR WRITER"<<endl;

getchar();

return -1;

}

}

bool bSuccess = cap.read(frame); // read a new frame from camera feed

if (!bSuccess) //test if frame successfully read

{

cout << "ERROR READING FRAME FROM CAMERA FEED" << endl;

break;

}

if(recording == true){

writer.write(frame);

putText(frame,"REC",Point(0,60),2,2,Scalar(0,0,255));

}

imshow(windowName, frame); //show the frame in "MyVideo" window

//listen for 10ms for a key to be pressed

switch(waitKey(10)){

case 27:

//'esc' has been pressed (ASCII value for 'esc' is 27)

//exit program.

return 0;

//r button is pressed

case 114:

//toggle recording

recording = !recording;

if(recording == true) cout<<"Begin Recording"<<endl;

else cout<<"Recording Paused"<<endl;

if(firstRun){

firstRun = false;

string filename = "C:\myVideo0.avi";

writer = VideoWriter(filename, fcc, fps, frameSize);

if(!writer.isOpened()){

cout<<"ERROR OPENING FILE FOR WRITER"<<endl;

getchar();

return -1;

}

}

break;

//n button is pressed

case 110:

//start new recording

startNewRecording = true;

cout<<"New recording started"<<endl;

if(firstRun == true)firstRun=false;

break;

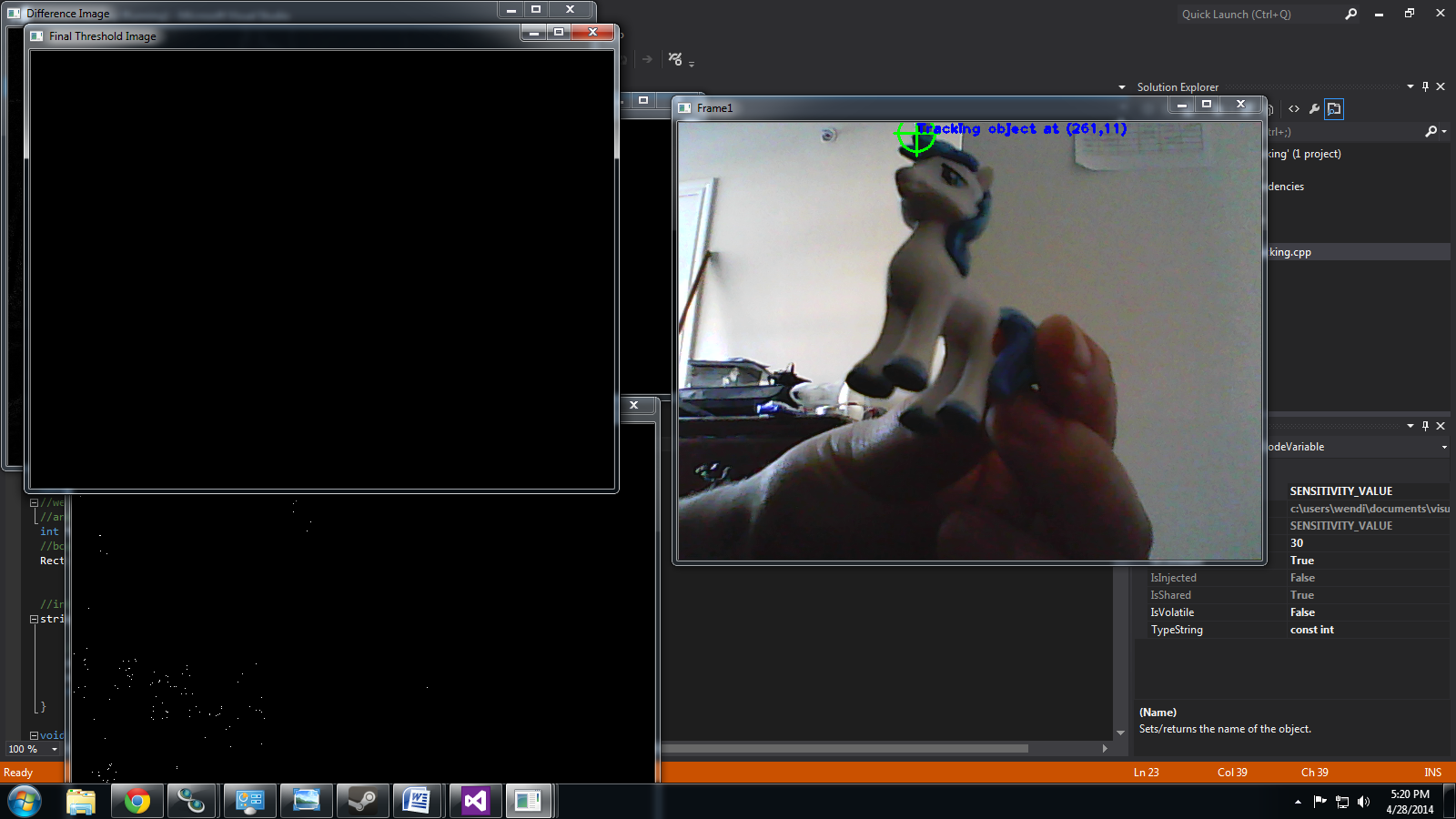
}

}

return 0;

}

## Real Time Object Tracking:



//motionTracking.cpp

//Written by Kyle Hounslow, January 2014

//Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software")

//, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,

//and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

//The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS

//IN THE SOFTWARE.

#include <opencv\cv.h>

#include <opencv\highgui.h>

using namespace std;

using namespace cv;

//our sensitivity value to be used in the threshold() function

const static int SENSITIVITY\_VALUE = 30;

//size of blur used to smooth the image to remove possible noise and

//increase the size of the object we are trying to track. (Much like dilate and erode)

const static int BLUR\_SIZE = 10;

//we'll have just one object to search for

//and keep track of its position.

int theObject[2] = {0,0};

//bounding rectangle of the object, we will use the center of this as its position.

Rect objectBoundingRectangle = Rect(0,0,0,0);

//int to string helper function

string intToString(int number){

//this function has a number input and string output

std::stringstream ss;

ss << number;

return ss.str();

}

void searchForMovement(Mat thresholdImage, Mat &cameraFeed){

//notice how we use the '&' operator for the cameraFeed. This is because we wish

//to take the values passed into the function and manipulate them, rather than just working with a copy.

//eg. we draw to the cameraFeed in this function which is then displayed in the main() function.

bool objectDetected=false;

Mat temp;

thresholdImage.copyTo(temp);

//these two vectors needed for output of findContours

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

//find contours of filtered image using openCV findContours function

//findContours(temp,contours,hierarchy,CV\_RETR\_CCOMP,CV\_CHAIN\_APPROX\_SIMPLE );// retrieves all contours

findContours(temp,contours,hierarchy,CV\_RETR\_EXTERNAL,CV\_CHAIN\_APPROX\_SIMPLE );// retrieves external contours

//if contours vector is not empty, we have found some objects

if(contours.size()>0)objectDetected=true;

else objectDetected = false;

if(objectDetected){

//the largest contour is found at the end of the contours vector

//we will simply assume that the biggest contour is the object we are looking for.

vector< vector<Point> > largestContourVec;

largestContourVec.push\_back(contours.at(contours.size()-1));

//make a bounding rectangle around the largest contour then find its centroid

//this will be the object's final estimated position.

objectBoundingRectangle = boundingRect(largestContourVec.at(0));

int xpos = objectBoundingRectangle.x+objectBoundingRectangle.width/2;

int ypos = objectBoundingRectangle.y+objectBoundingRectangle.height/2;

//update the objects positions by changing the 'theObject' array values

theObject[0] = xpos , theObject[1] = ypos;

}

//make some temp x and y variables so we dont have to type out so much

int x = theObject[0];

int y = theObject[1];

//draw some crosshairs on the object

circle(cameraFeed,Point(x,y),20,Scalar(0,255,0),2);

line(cameraFeed,Point(x,y),Point(x,y-25),Scalar(0,255,0),2);

line(cameraFeed,Point(x,y),Point(x,y+25),Scalar(0,255,0),2);

line(cameraFeed,Point(x,y),Point(x-25,y),Scalar(0,255,0),2);

line(cameraFeed,Point(x,y),Point(x+25,y),Scalar(0,255,0),2);

putText(cameraFeed,"Tracking object at (" + intToString(x)+","+intToString(y)+")",Point(x,y),1,1,Scalar(255,0,0),2);

}

int main(){

//some boolean variables for added functionality

bool objectDetected = false;

//these two can be toggled by pressing 'd' or 't'

bool debugMode = false;

bool trackingEnabled = false;

//pause and resume code

bool pause = false;

//set up the matrices that we will need

//the two frames we will be comparing

Mat frame1,frame2;

//their grayscale images (needed for absdiff() function)

Mat grayImage1,grayImage2;

//resulting difference image

Mat differenceImage;

//thresholded difference image (for use in findContours() function)

Mat thresholdImage;

//video capture object.

VideoCapture capture;

capture.open(0);

if(!capture.isOpened()){

cout<<"ERROR ACQUIRING VIDEO FEED\n";

getchar();

return -1;

}

while(1){

//we can loop the video by re-opening the capture every time the video reaches its last frame

//check if the video has reach its last frame.

//we add '-1' because we are reading two frames from the video at a time.

//if this is not included, we get a memory error!

//while(capture.get(CV\_CAP\_PROP\_POS\_FRAMES)<capture.get(CV\_CAP\_PROP\_FRAME\_COUNT)-1){

//read first frame

capture.read(frame1);

//convert frame1 to gray scale for frame differencing

cv::cvtColor(frame1, grayImage1,COLOR\_BGR2GRAY);

//copy second frame

capture.read(frame2);

//convert frame2 to gray scale for frame differencing

cv::cvtColor(frame2, grayImage2,COLOR\_BGR2GRAY);

//perform frame differencing with the sequential images. This will output an "intensity image"

//do not confuse this with a threshold image, we will need to perform thresholding afterwards.

cv::absdiff(grayImage1,grayImage2,differenceImage);

//threshold intensity image at a given sensitivity value

cv::threshold(differenceImage,thresholdImage,SENSITIVITY\_VALUE,255,THRESH\_BINARY);

if(debugMode==true){

//show the difference image and threshold image

cv::imshow("Difference Image", differenceImage);

cv::imshow("Threshold Image", thresholdImage);

}else{

//if not in debug mode, destroy the windows so we don't see them anymore

cv::destroyWindow("Difference Image");

cv::destroyWindow("Threshold Image");

}

//use blur() to smooth the image, remove possible noise and

cv::blur(thresholdImage,thresholdImage,cv::Size(BLUR\_SIZE,BLUR\_SIZE));

//increase the size of the object we are trying to track. (Much like dilate and erode)

//threshold again to obtain binary image from blur output

cv::threshold(thresholdImage,thresholdImage,SENSITIVITY\_VALUE,255,THRESH\_BINARY);

if(debugMode==true){

//show the threshold image after it's been "blurred"

imshow("Final Threshold Image", thresholdImage);

}

else {

//if not in debug mode, destroy the windows so we don't see them anymore

cv:destroyWindow("Final Threshold Image");

}

//if tracking enabled, search for contours in our thresholded image

if(trackingEnabled){

searchForMovement(thresholdImage,frame1);

}

//show our captured frame

imshow("Frame1",frame1);

//check to see if a button has been pressed.

//this 10ms delay is necessary for proper operation of this program

//if removed, frames will not have enough time to referesh and a blank

//image will appear.

switch(waitKey(10)){

case 27: //'esc' key has been pressed, exit program.

return 0;

case 116: //'t' has been pressed. this will toggle tracking

trackingEnabled = !trackingEnabled;

if(trackingEnabled == false) cout<<"Tracking disabled."<<endl;

else cout<<"Tracking enabled."<<endl;

break;

case 100: //'d' has been pressed. this will debug mode

debugMode = !debugMode;

if(debugMode == false) cout<<"Debug mode disabled."<<endl;

else cout<<"Debug mode enabled."<<endl;

break;

case 112: //'p' has been pressed. this will pause/resume the code.

pause = !pause;

if(pause == true){ cout<<"Code paused, press 'p' again to resume"<<endl;

while (pause == true){

//stay in this loop until

switch (waitKey()){

//a switch statement inside a switch statement? Mind blown.

case 112:

//change pause back to false

pause = false;

cout<<"Code resumed."<<endl;

break;

}

}

}

}

//}

//release the capture before re-opening and looping again.

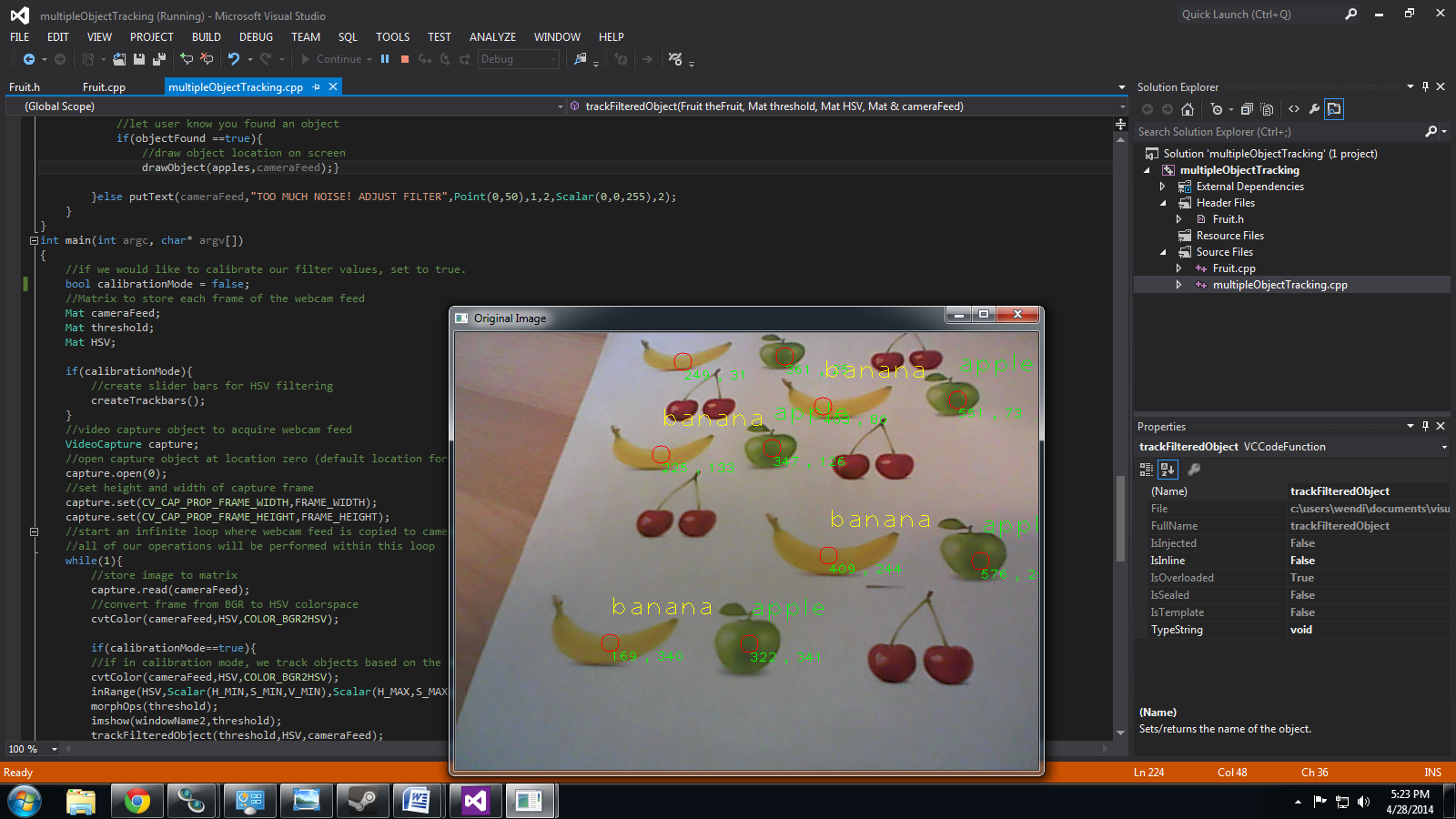
//capture.release();

}

return 0;

}

## Multiple Object Tracking:



//objectTrackingTutorial.cpp

//Written by Kyle Hounslow 2013

//Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software")

//, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,

//and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

//The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS

//IN THE SOFTWARE.

#include <sstream>

#include <string>

#include <iostream>

#include<vector>

#include "Fruit.h"

using namespace cv;

//initial min and max HSV filter values.

//these will be changed using trackbars

int H\_MIN = 0;

int H\_MAX = 256;

int S\_MIN = 0;

int S\_MAX = 256;

int V\_MIN = 0;

int V\_MAX = 256;

//default capture width and height

const int FRAME\_WIDTH = 640;

const int FRAME\_HEIGHT = 480;

//max number of objects to be detected in frame

const int MAX\_NUM\_OBJECTS=50;

//minimum and maximum object area

const int MIN\_OBJECT\_AREA = 20\*20;

const int MAX\_OBJECT\_AREA = FRAME\_HEIGHT\*FRAME\_WIDTH/1.5;

//names that will appear at the top of each window

const string windowName = "Original Image";

const string windowName1 = "HSV Image";

const string windowName2 = "Thresholded Image";

const string windowName3 = "After Morphological Operations";

const string trackbarWindowName = "Trackbars";

void on\_trackbar( int, void\* )

{//This function gets called whenever a

// trackbar position is changed

}

string intToString(int number){

std::stringstream ss;

ss << number;

return ss.str();

}

void createTrackbars(){

//create window for trackbars

namedWindow(trackbarWindowName,0);

//create memory to store trackbar name on window

char TrackbarName[50];

sprintf( TrackbarName, "H\_MIN", H\_MIN);

sprintf( TrackbarName, "H\_MAX", H\_MAX);

sprintf( TrackbarName, "S\_MIN", S\_MIN);

sprintf( TrackbarName, "S\_MAX", S\_MAX);

sprintf( TrackbarName, "V\_MIN", V\_MIN);

sprintf( TrackbarName, "V\_MAX", V\_MAX);

//create trackbars and insert them into window

//3 parameters are: the address of the variable that is changing when the trackbar is moved(eg.H\_LOW),

//the max value the trackbar can move (eg. H\_HIGH),

//and the function that is called whenever the trackbar is moved(eg. on\_trackbar)

// ----> ----> ---->

createTrackbar( "H\_MIN", trackbarWindowName, &H\_MIN, H\_MAX, on\_trackbar );

createTrackbar( "H\_MAX", trackbarWindowName, &H\_MAX, H\_MAX, on\_trackbar );

createTrackbar( "S\_MIN", trackbarWindowName, &S\_MIN, S\_MAX, on\_trackbar );

createTrackbar( "S\_MAX", trackbarWindowName, &S\_MAX, S\_MAX, on\_trackbar );

createTrackbar( "V\_MIN", trackbarWindowName, &V\_MIN, V\_MAX, on\_trackbar );

createTrackbar( "V\_MAX", trackbarWindowName, &V\_MAX, V\_MAX, on\_trackbar );

}

void drawObject(vector<Fruit> theFruits,Mat &frame){

for(int i = 0; i<theFruits.size(); i++){

cv::circle(frame,cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()),10,cv::Scalar(0,0,255));

cv::putText(frame,intToString(theFruits.at(i).getXPos())+ " , " + intToString(theFruits.at(i).getYPos()),cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()+20),1,1,Scalar(0,255,0));

cv::putText(frame,theFruits.at(i).getType(),cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()-30),1,2,theFruits.at(i).getColour());

}

}

void morphOps(Mat &thresh){

//create structuring element that will be used to "dilate" and "erode" image.

//the element chosen here is a 3px by 3px rectangle

Mat erodeElement = getStructuringElement( MORPH\_RECT,Size(3,3));

//dilate with larger element so make sure object is nicely visible

Mat dilateElement = getStructuringElement( MORPH\_RECT,Size(8,8));

erode(thresh,thresh,erodeElement);

erode(thresh,thresh,erodeElement);

dilate(thresh,thresh,dilateElement);

dilate(thresh,thresh,dilateElement);

}

void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed){

vector<Fruit> apples;

Mat temp;

threshold.copyTo(temp);

//these two vectors needed for output of findContours

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

//find contours of filtered image using openCV findContours function

findContours(temp,contours,hierarchy,CV\_RETR\_CCOMP,CV\_CHAIN\_APPROX\_SIMPLE );

//use moments method to find our filtered object

double refArea = 0;

bool objectFound = false;

if (hierarchy.size() > 0) {

int numObjects = hierarchy.size();

//if number of objects greater than MAX\_NUM\_OBJECTS we have a noisy filter

if(numObjects<MAX\_NUM\_OBJECTS){

for (int index = 0; index >= 0; index = hierarchy[index][0]) {

Moments moment = moments((cv::Mat)contours[index]);

double area = moment.m00;

//if the area is less than 20 px by 20px then it is probably just noise

//if the area is the same as the 3/2 of the image size, probably just a bad filter

//we only want the object with the largest area so we safe a reference area each

//iteration and compare it to the area in the next iteration.

if(area>MIN\_OBJECT\_AREA){

Fruit apple;

apple.setXPos(moment.m10/area);

apple.setYPos(moment.m01/area);

apples.push\_back(apple);

//x = moment.m10/area;

//y = moment.m01/area;

objectFound = true;

}else objectFound = false;

}

//let user know you found an object

if(objectFound ==true){

//draw object location on screen

drawObject(apples,cameraFeed);}

}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);

}

}

void trackFilteredObject(Fruit theFruit, Mat threshold,Mat HSV, Mat &cameraFeed){

vector<Fruit> apples;

Mat temp;

threshold.copyTo(temp);

//these two vectors needed for output of findContours

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

//find contours of filtered image using openCV findContours function

findContours(temp,contours,hierarchy,CV\_RETR\_CCOMP,CV\_CHAIN\_APPROX\_SIMPLE );

//use moments method to find our filtered object

double refArea = 0;

bool objectFound = false;

if (hierarchy.size() > 0) {

int numObjects = hierarchy.size();

//if number of objects greater than MAX\_NUM\_OBJECTS we have a noisy filter

if(numObjects<MAX\_NUM\_OBJECTS){

for (int index = 0; index >= 0; index = hierarchy[index][0]) {

Moments moment = moments((cv::Mat)contours[index]);

double area = moment.m00;

//if the area is less than 20 px by 20px then it is probably just noise

//if the area is the same as the 3/2 of the image size, probably just a bad filter

//we only want the object with the largest area so we safe a reference area each

//iteration and compare it to the area in the next iteration.

if(area>MIN\_OBJECT\_AREA){

Fruit apple;

apple.setXPos(moment.m10/area);

apple.setYPos(moment.m01/area);

apple.setType(theFruit.getType());

apple.setColour(theFruit.getColour());

apples.push\_back(apple);

//x = moment.m10/area;

//y = moment.m01/area;

objectFound = true;

}else objectFound = false;

}

//let user know you found an object

if(objectFound ==true){

//draw object location on screen

drawObject(apples,cameraFeed);}

}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);

}

}

int main(int argc, char\* argv[])

{

//if we would like to calibrate our filter values, set to true.

bool calibrationMode = false;

//Matrix to store each frame of the webcam feed

Mat cameraFeed;

Mat threshold;

Mat HSV;

if(calibrationMode){

//create slider bars for HSV filtering

createTrackbars();

}

//video capture object to acquire webcam feed

VideoCapture capture;

//open capture object at location zero (default location for webcam)

capture.open(0);

//set height and width of capture frame

capture.set(CV\_CAP\_PROP\_FRAME\_WIDTH,FRAME\_WIDTH);

capture.set(CV\_CAP\_PROP\_FRAME\_HEIGHT,FRAME\_HEIGHT);

//start an infinite loop where webcam feed is copied to cameraFeed matrix

//all of our operations will be performed within this loop

while(1){

//store image to matrix

capture.read(cameraFeed);

//convert frame from BGR to HSV colorspace

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

if(calibrationMode==true){

//if in calibration mode, we track objects based on the HSV slider values.

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,Scalar(H\_MIN,S\_MIN,V\_MIN),Scalar(H\_MAX,S\_MAX,V\_MAX),threshold);

morphOps(threshold);

imshow(windowName2,threshold);

trackFilteredObject(threshold,HSV,cameraFeed);

}else{

Fruit apple("apple"), banana("banana"), cherry("cherry");

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,apple.getHSVmin(),apple.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(apple,threshold,HSV,cameraFeed);

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,banana.getHSVmin(),banana.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(banana,threshold,HSV,cameraFeed);

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,cherry.getHSVmin(),cherry.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(cherry,threshold,HSV,cameraFeed);

}

//show frames

//imshow(windowName2,threshold);

imshow(windowName,cameraFeed);

//imshow(windowName1,HSV);

//delay 30ms so that screen can refresh.

//image will not appear without this waitKey() command

waitKey(30);

}

return 0;

}

#include "Fruit.h"

Fruit::Fruit(void)

{

}

Fruit::Fruit(string name){

setType(name);

if(name == "apple"){

setHSVmin(Scalar(35,95,47));

setHSVmax(Scalar(51,219,174));

setColour(Scalar(0,255,0));

}

if(name == "banana"){

setHSVmin(Scalar(19,132,105));

setHSVmax(Scalar(32,179,181));

setColour(Scalar(0,255,255));

}

if(name == "cherry"){

setHSVmin(Scalar(0,140,65));

setHSVmax(Scalar(16,224,144));

setColour(Scalar(0,0,255));

}

}

Fruit::~Fruit(void)

{

}

int Fruit::getXPos(){

return Fruit::xPos;

}

void Fruit::setXPos(int x){

Fruit::xPos = x;

}

int Fruit::getYPos(){

return Fruit::yPos;

}

void Fruit::setYPos(int y){

Fruit::yPos = y;

}

Scalar Fruit::getHSVmin(){

return HSVmin;

}

Scalar Fruit::getHSVmax(){

return HSVmax;

}

void Fruit::setHSVmin(Scalar min){

Fruit::HSVmin = min;

}

void Fruit::setHSVmax(Scalar max){

Fruit::HSVmax = max;

}

#pragma once

#include <string>

#include<opencv\cv.h>

#include <opencv\highgui.h>

using namespace std;

using namespace cv;

class Fruit

{

public:

Fruit(void);

~Fruit(void);

Fruit(string name);

int getXPos();

void setXPos(int x);

int getYPos();

void setYPos(int y);

Scalar getHSVmin();

Scalar getHSVmax();

void setHSVmin(Scalar min);

void setHSVmax(Scalar max);

string getType(){return type;}

void setType(string t){type = t;}

Scalar getColour(){return Colour;}

void setColour(Scalar c){Colour = c;}

private:

int xPos, yPos;

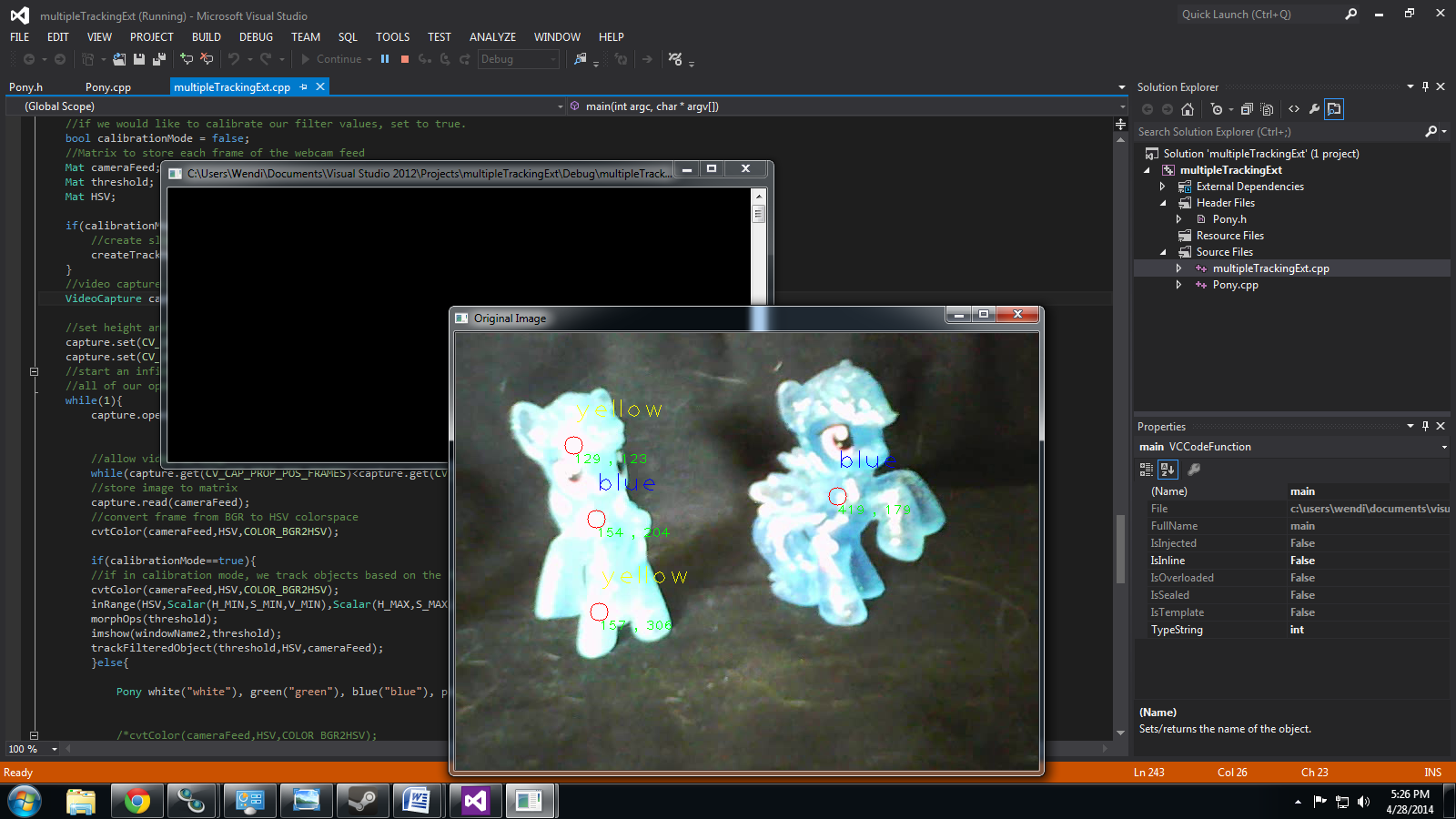
string type;

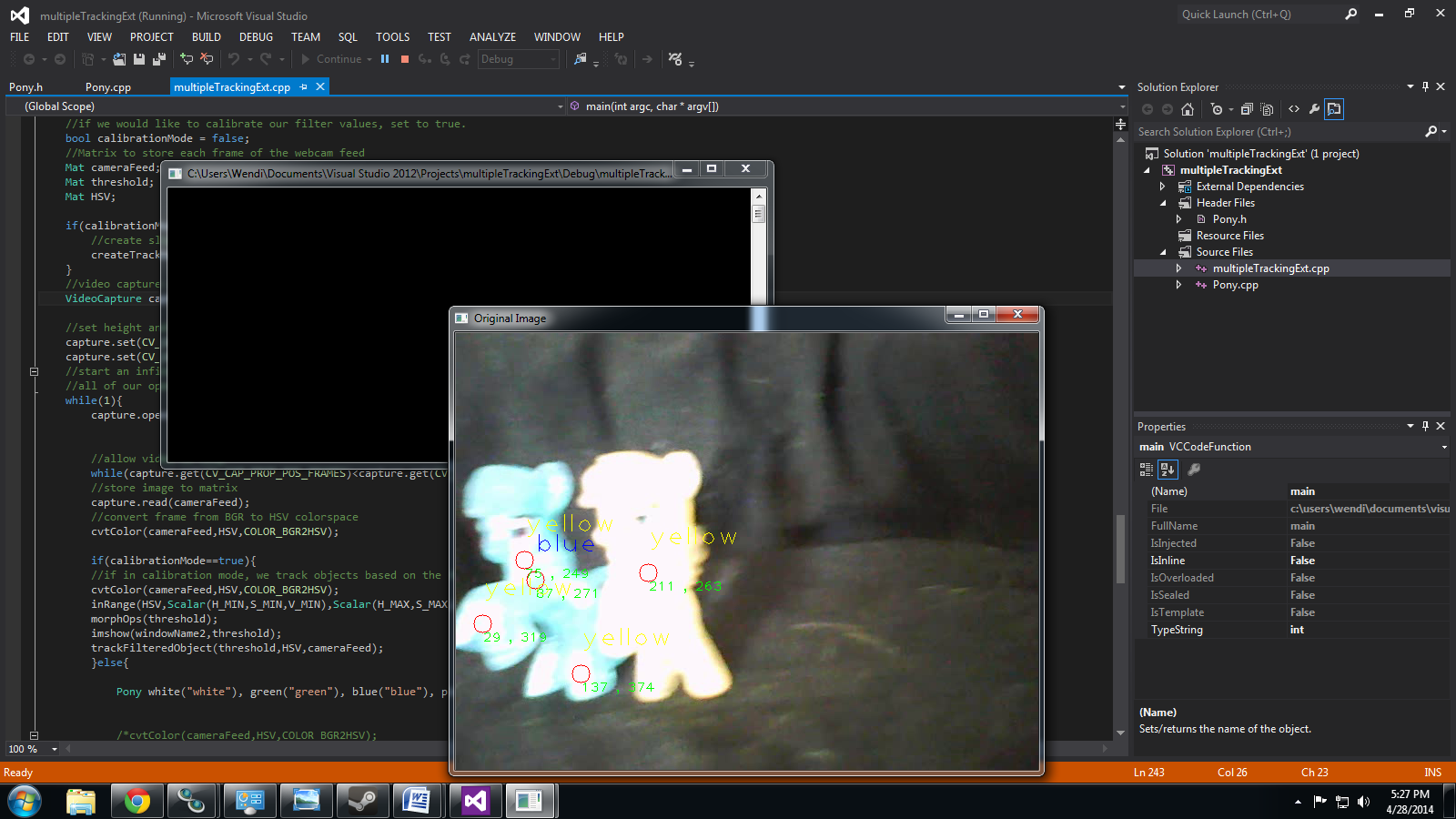
Scalar HSVmin, HSVmax;

Scalar Colour;

};

# Part Two:





//objectTrackingTutorial.cpp

//Written by Kyle Hounslow 2013

//Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software")

//, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,

//and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

//The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS

//IN THE SOFTWARE.

#include <sstream>

#include <iostream>

#include<vector>

#include "Pony.h"

using namespace cv;

//initial min and max HSV filter values.

//these will be changed using trackbars

int H\_MIN = 0;

int H\_MAX = 256;

int S\_MIN = 0;

int S\_MAX = 256;

int V\_MIN = 0;

int V\_MAX = 256;

//default capture width and height

const int FRAME\_WIDTH = 640;

const int FRAME\_HEIGHT = 480;

//max number of objects to be detected in frame

const int MAX\_NUM\_OBJECTS=50;

//minimum and maximum object area

const int MIN\_OBJECT\_AREA = 35\*35;

const int MAX\_OBJECT\_AREA = FRAME\_HEIGHT\*FRAME\_WIDTH/1.5;

//names that will appear at the top of each window

const string windowName = "Original Image";

const string windowName1 = "HSV Image";

const string windowName2 = "Thresholded Image";

const string windowName3 = "After Morphological Operations";

const string trackbarWindowName = "Trackbars";

void on\_trackbar( int, void\* )

{//This function gets called whenever a

// trackbar position is changed

}

string intToString(int number){

std::stringstream ss;

ss << number;

return ss.str();

}

void createTrackbars(){

//create window for trackbars

namedWindow(trackbarWindowName,0);

//create memory to store trackbar name on window

char TrackbarName[50];

sprintf( TrackbarName, "H\_MIN", H\_MIN);

sprintf( TrackbarName, "H\_MAX", H\_MAX);

sprintf( TrackbarName, "S\_MIN", S\_MIN);

sprintf( TrackbarName, "S\_MAX", S\_MAX);

sprintf( TrackbarName, "V\_MIN", V\_MIN);

sprintf( TrackbarName, "V\_MAX", V\_MAX);

//create trackbars and insert them into window

//3 parameters are: the address of the variable that is changing when the trackbar is moved(eg.H\_LOW),

//the max value the trackbar can move (eg. H\_HIGH),

//and the function that is called whenever the trackbar is moved(eg. on\_trackbar)

// ----> ----> ---->

createTrackbar( "H\_MIN", trackbarWindowName, &H\_MIN, H\_MAX, on\_trackbar );

createTrackbar( "H\_MAX", trackbarWindowName, &H\_MAX, H\_MAX, on\_trackbar );

createTrackbar( "S\_MIN", trackbarWindowName, &S\_MIN, S\_MAX, on\_trackbar );

createTrackbar( "S\_MAX", trackbarWindowName, &S\_MAX, S\_MAX, on\_trackbar );

createTrackbar( "V\_MIN", trackbarWindowName, &V\_MIN, V\_MAX, on\_trackbar );

createTrackbar( "V\_MAX", trackbarWindowName, &V\_MAX, V\_MAX, on\_trackbar );

}

void drawObject(vector<Pony> theFruits,Mat &frame){

for(int i = 0; i<theFruits.size(); i++){

cv::circle(frame,cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()),10,cv::Scalar(0,0,255));

cv::putText(frame,intToString(theFruits.at(i).getXPos())+ " , " + intToString(theFruits.at(i).getYPos()),cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()+20),1,1,Scalar(0,255,0));

cv::putText(frame,theFruits.at(i).getType(),cv::Point(theFruits.at(i).getXPos(),theFruits.at(i).getYPos()-30),1,2,theFruits.at(i).getColour());

}

}

void morphOps(Mat &thresh){

//create structuring element that will be used to "dilate" and "erode" image.

//the element chosen here is a 3px by 3px rectangle

Mat erodeElement = getStructuringElement( MORPH\_RECT,Size(3,3));

//dilate with larger element so make sure object is nicely visible

Mat dilateElement = getStructuringElement( MORPH\_RECT,Size(8,8));

erode(thresh,thresh,erodeElement);

erode(thresh,thresh,erodeElement);

dilate(thresh,thresh,dilateElement);

dilate(thresh,thresh,dilateElement);

}

void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed){

vector<Pony> apples;

Mat temp;

threshold.copyTo(temp);

//these two vectors needed for output of findContours

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

//find contours of filtered image using openCV findContours function

findContours(temp,contours,hierarchy,CV\_RETR\_CCOMP,CV\_CHAIN\_APPROX\_SIMPLE );

//use moments method to find our filtered object

double refArea = 0;

bool objectFound = false;

if (hierarchy.size() > 0) {

int numObjects = hierarchy.size();

//if number of objects greater than MAX\_NUM\_OBJECTS we have a noisy filter

if(numObjects<MAX\_NUM\_OBJECTS){

for (int index = 0; index >= 0; index = hierarchy[index][0]) {

Moments moment = moments((cv::Mat)contours[index]);

double area = moment.m00;

//if the area is less than 20 px by 20px then it is probably just noise

//if the area is the same as the 3/2 of the image size, probably just a bad filter

//we only want the object with the largest area so we safe a reference area each

//iteration and compare it to the area in the next iteration.

if(area>MIN\_OBJECT\_AREA){

Pony apple;

apple.setXPos(moment.m10/area);

apple.setYPos(moment.m01/area);

apples.push\_back(apple);

//x = moment.m10/area;

//y = moment.m01/area;

objectFound = true;

}else objectFound = false;

}

//let user know you found an object

if(objectFound ==true){

//draw object location on screen

drawObject(apples,cameraFeed);}

}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);

}

}

void trackFilteredObject(Pony theFruit, Mat threshold,Mat HSV, Mat &cameraFeed){

vector<Pony> apples;

Mat temp;

threshold.copyTo(temp);

//these two vectors needed for output of findContours

vector< vector<Point> > contours;

vector<Vec4i> hierarchy;

//find contours of filtered image using openCV findContours function

findContours(temp,contours,hierarchy,CV\_RETR\_CCOMP,CV\_CHAIN\_APPROX\_SIMPLE );

//use moments method to find our filtered object

double refArea = 0;

bool objectFound = false;

if (hierarchy.size() > 0) {

int numObjects = hierarchy.size();

//if number of objects greater than MAX\_NUM\_OBJECTS we have a noisy filter

if(numObjects<MAX\_NUM\_OBJECTS){

for (int index = 0; index >= 0; index = hierarchy[index][0]) {

Moments moment = moments((cv::Mat)contours[index]);

double area = moment.m00;

//if the area is less than 20 px by 20px then it is probably just noise

//if the area is the same as the 3/2 of the image size, probably just a bad filter

//we only want the object with the largest area so we safe a reference area each

//iteration and compare it to the area in the next iteration.

if(area>MIN\_OBJECT\_AREA){

Pony apple;

apple.setXPos(moment.m10/area);

apple.setYPos(moment.m01/area);

apple.setType(theFruit.getType());

apple.setColour(theFruit.getColour());

apples.push\_back(apple);

//x = moment.m10/area;

//y = moment.m01/area;

objectFound = true;

}else objectFound = false;

}

//let user know you found an object

if(objectFound ==true){

//draw object location on screen

drawObject(apples,cameraFeed);}

}else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);

}

}

int main(int argc, char\* argv[])

{

//if we would like to calibrate our filter values, set to true.

bool calibrationMode = false;

//Matrix to store each frame of the webcam feed

Mat cameraFeed;

Mat threshold;

Mat HSV;

if(calibrationMode){

//create slider bars for HSV filtering

createTrackbars();

}

//video capture object to acquire webcam feed

VideoCapture capture;

//set height and width of capture frame

capture.set(CV\_CAP\_PROP\_FRAME\_WIDTH,FRAME\_WIDTH);

capture.set(CV\_CAP\_PROP\_FRAME\_HEIGHT,FRAME\_HEIGHT);

//start an infinite loop where webcam feed is copied to cameraFeed matrix

//all of our operations will be performed within this loop

while(1){

capture.open("MyVideo0.avi");

//allow video to loop

while(capture.get(CV\_CAP\_PROP\_POS\_FRAMES)<capture.get(CV\_CAP\_PROP\_FRAME\_COUNT)-1){

//store image to matrix

capture.read(cameraFeed);

//convert frame from BGR to HSV colorspace

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

if(calibrationMode==true){

//if in calibration mode, we track objects based on the HSV slider values.

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,Scalar(H\_MIN,S\_MIN,V\_MIN),Scalar(H\_MAX,S\_MAX,V\_MAX),threshold);

morphOps(threshold);

imshow(windowName2,threshold);

trackFilteredObject(threshold,HSV,cameraFeed);

}else{

Pony white("white"), green("green"), blue("blue"), pink("pink"), yellow("yellow");

/\*cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,white.getHSVmin(),white.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(white,threshold,HSV,cameraFeed);\*/

/\*cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,green.getHSVmin(),green.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(green,threshold,HSV,cameraFeed);\*/

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,blue.getHSVmin(),blue.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(blue,threshold,HSV,cameraFeed);

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,pink.getHSVmin(),pink.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(pink,threshold,HSV,cameraFeed);

cvtColor(cameraFeed,HSV,COLOR\_BGR2HSV);

inRange(HSV,yellow.getHSVmin(),yellow.getHSVmax(),threshold);

morphOps(threshold);

trackFilteredObject(yellow,threshold,HSV,cameraFeed);

}

//show frames

//imshow(windowName2,threshold);

imshow(windowName,cameraFeed);

//imshow(windowName1,HSV);

//delay 30ms so that screen can refresh.

//image will not appear without this waitKey() command

waitKey(30);

}

capture.release();

}

return 0;

}

#pragma once

#include <string>

#include<opencv\cv.h>

#include <opencv\highgui.h>

using namespace std;

using namespace cv;

class Pony

{

public:

Pony(void);

~Pony(void);

Pony(string name);

int getXPos();

void setXPos(int x);

int getYPos();

void setYPos(int y);

Scalar getHSVmin();

Scalar getHSVmax();

void setHSVmin(Scalar min);

void setHSVmax(Scalar max);

string getType(){return type;}

void setType(string t){type = t;}

Scalar getColour(){return Colour;}

void setColour(Scalar c){Colour = c;}

private:

int xPos, yPos;

string type;

Scalar HSVmin, HSVmax;

Scalar Colour;

};

#include "Pony.h"

Pony::Pony(void)

{

}

Pony::~Pony(void)

{

}

Pony::Pony(string name){

setType(name);

if(name == "white"){

setHSVmin(Scalar(0,47,179));

setHSVmax(Scalar(46,100,224));

setColour(Scalar(0,0,0));

}

if(name == "green"){

setHSVmin(Scalar(21,107,79));

setHSVmax(Scalar(44,191,160));

setColour(Scalar(0,255,0));

}

if(name == "blue"){

setHSVmin(Scalar(86,12,135));

setHSVmax(Scalar(105,128,256));

setColour(Scalar(255,0,0));

}

if(name == "pink"){

setHSVmin(Scalar(147,26,200));

setHSVmax(Scalar(182,256,256));

setColour(Scalar(77,51,196));

}

if(name == "yellow"){

setHSVmin(Scalar(140,0,247));

setHSVmax(Scalar(160,12,256));

setColour(Scalar(0,255,255));

}

}

int Pony::getXPos(){

return Pony::xPos;

}

void Pony::setXPos(int x){

Pony::xPos = x;

}

int Pony::getYPos(){

return Pony::yPos;

}

void Pony::setYPos(int y){

Pony::yPos = y;

}

Scalar Pony::getHSVmin(){

return HSVmin;

}

Scalar Pony::getHSVmax(){

return HSVmax;

}

void Pony::setHSVmin(Scalar min){

Pony::HSVmin = min;

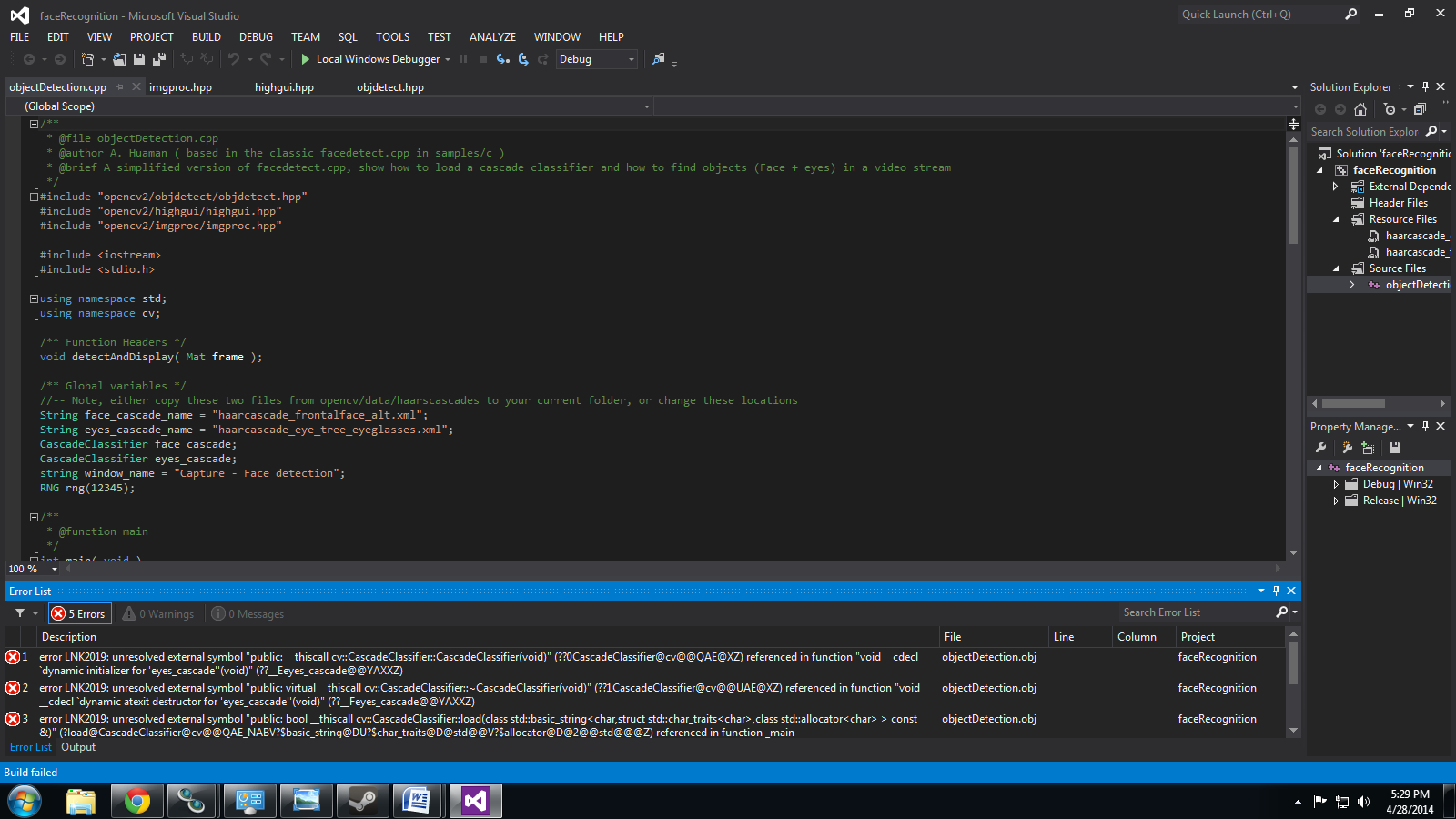
}

void Pony::setHSVmax(Scalar max){

Pony::HSVmax = max;

}

# Part Three:



/\*\*

\* @file objectDetection.cpp

\* @author A. Huaman ( based in the classic facedetect.cpp in samples/c )

\* @brief A simplified version of facedetect.cpp, show how to load a cascade classifier and how to find objects (Face + eyes) in a video stream

\*/

#include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

using namespace std;

using namespace cv;

/\*\* Function Headers \*/

void detectAndDisplay( Mat frame );

/\*\* Global variables \*/

//-- Note, either copy these two files from opencv/data/haarscascades to your current folder, or change these locations

String face\_cascade\_name = "haarcascade\_frontalface\_alt.xml";

String eyes\_cascade\_name = "haarcascade\_eye\_tree\_eyeglasses.xml";

CascadeClassifier face\_cascade;

CascadeClassifier eyes\_cascade;

string window\_name = "Capture - Face detection";

RNG rng(12345);

/\*\*

\* @function main

\*/

int main( void )

{

VideoCapture capture;

Mat frame;

//-- 1. Load the cascades

if( !face\_cascade.load( face\_cascade\_name ) ){ printf("--(!)Error loading\n"); return -1; };

if( !eyes\_cascade.load( eyes\_cascade\_name ) ){ printf("--(!)Error loading\n"); return -1; };

//-- 2. Read the video stream

capture.open( -1 );

if( capture.isOpened() )

{

for(;;)

{

capture >> frame;

//-- 3. Apply the classifier to the frame

if( !frame.empty() )

{ detectAndDisplay( frame ); }

else

{ printf(" --(!) No captured frame -- Break!"); break; }

int c = waitKey(10);

if( (char)c == 'c' ) { break; }

}

}

return 0;

}

/\*\*

\* @function detectAndDisplay

\*/

void detectAndDisplay( Mat frame )

{

std::vector<Rect> faces;

Mat frame\_gray;

cvtColor( frame, frame\_gray, COLOR\_BGR2GRAY );

equalizeHist( frame\_gray, frame\_gray );

//-- Detect faces

face\_cascade.detectMultiScale( frame\_gray, faces, 1.1, 2, 0|CV\_HAAR\_SCALE\_IMAGE, Size(30, 30) );

for( size\_t i = 0; i < faces.size(); i++ )

{

Point center( faces[i].x + faces[i].width/2, faces[i].y + faces[i].height/2 );

ellipse( frame, center, Size( faces[i].width/2, faces[i].height/2), 0, 0, 360, Scalar( 255, 0, 255 ), 2, 8, 0 );

Mat faceROI = frame\_gray( faces[i] );

std::vector<Rect> eyes;

//-- In each face, detect eyes

eyes\_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV\_HAAR\_SCALE\_IMAGE, Size(30, 30) );

for( size\_t j = 0; j < eyes.size(); j++ )

{

Point eye\_center( faces[i].x + eyes[j].x + eyes[j].width/2, faces[i].y + eyes[j].y + eyes[j].height/2 );

int radius = cvRound( (eyes[j].width + eyes[j].height)\*0.25 );

circle( frame, eye\_center, radius, Scalar( 255, 0, 0 ), 3, 8, 0 );

}

}

//-- Show what you got

imshow( window\_name, frame );

}