// RedBallTracker.cpp

#include <opencv2/core/utility.hpp>

#include <opencv2/videoio.hpp>

#include <opencv2/highgui.hpp>

#include <sstream>

#include <string>

#include <iostream>

#include <opencv\highgui.h>

#include <opencv\cv.h>

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

using namespace cv;

using namespace std;

//initial min and max HSV filter values.

//these will be changed using trackbars

//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) {

stringstream ss;

ss << number;

return ss.str();

}

// found this tracking bar online

// easy way of figuring out what colors to filter out

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);

}

// this is the part we need to firgure out for the clyinder

// maybe rect2d?

void drawObject(int x, int y, Mat &frame) {

//use some of the openCV drawing functions to draw crosshairs

//on your tracked image!

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

if (y - 25>0)

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

else line(frame, Point(x, y), Point(x, 0), Scalar(0, 255, 0), 2);

if (y + 25<FRAME\_HEIGHT)

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

else line(frame, Point(x, y), Point(x, FRAME\_HEIGHT), Scalar(0, 255, 0), 2);

if (x - 25>0)

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

else line(frame, Point(x, y), Point(0, y), Scalar(0, 255, 0), 2);

if (x + 25<FRAME\_WIDTH)

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

else line(frame, Point(x, y), Point(FRAME\_WIDTH, y), Scalar(0, 255, 0), 2);

putText(frame, intToString(x) + "," + intToString(y), Point(x, y + 30), 1, 1, Scalar(0, 255, 0), 2);

}

// to filter out noise making the HSV clearier

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(int &x, int &y, Mat threshold, Mat &cameraFeed) {

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 && area<MAX\_OBJECT\_AREA && area>refArea) {

x = moment.m10 / area;

y = moment.m01 / area;

objectFound = true;

refArea = area;

}

else objectFound = false;

}

//let user know you found an object

if (objectFound == true) {

putText(cameraFeed, "Tracking Object", Point(0, 50), 2, 1, Scalar(0, 255, 0), 2);

//draw object location on screen

drawObject(x, y, cameraFeed);

}

}

else putText(cameraFeed, "Error. Too much Noise", Point(0, 50), 1, 2, Scalar(0, 0, 255), 2);

}

}

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

{

//some boolean variables for different functionality within this

//program

bool trackObjects = true;

bool useMorphOps = true;

//Matrix to store each frame of the webcam feed

Mat cameraFeed;

//matrix storage for HSV image

Mat HSV;

//matrix storage for binary threshold image

Mat threshold;

//x and y values for the location of the object

int x = 0, y = 0;

//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(1); // this will be

//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);

//filter HSV image between values and store filtered image to

//threshold matrix

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

//perform morphological operations on thresholded image to eliminate noise

//and emphasize the filtered object(s)

if (useMorphOps)

morphOps(threshold);

//pass in thresholded frame to our object tracking function

//this function will return the x and y coordinates of the

//filtered object

if (trackObjects)

trackFilteredObject(x, y, threshold, 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;

}