#include "stdafx.h"

#include<opencv2/core/core.hpp>

#include<opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

#include<sstream>

#include<iostream>

using namespace cv;

using namespace std;

int positionReturn(int currentPos) {

return currentPos;

}

int areaReturn(int currentArea) {

return currentArea;

}

int main() {

VideoCapture capWebcam(1); //use webcam 1 if working with the usb webcam, cam 0 is laptop webcam

if (capWebcam.isOpened() == false) // To check if object was associated to webcam successfully

{

cout << "error: Webcam connect unsuccessful\n"; // if not then print error message

cout << CV\_VERSION << endl;

system("pause");

return(0); // and exit program

}

capWebcam.set(CV\_CAP\_PROP\_FRAME\_WIDTH, 600); //sets input capture to a set resolution of 1024x1024

capWebcam.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 600);

Mat imgOriginal; //original image capture

Mat hsvImg; //image to hold the hsv values

Mat threshImg; //hsv image we isolate the color from

ostringstream outputss; //for console clarity

bool coordsOK = false;

char charCheckForEscKey = 0; //esc key

int lowH = 0;//149;//21; // Set Hue

int highH = 7;//179;//30;

int lowS = 50;//110;//200; // Set Saturation

int highS = 255;//228;//255;

int lowV = 145;//185;//102; // Set Value

int highV = 255;//255;//225;

int iLastX = -1;

int iLastY = -1;

vector<vector<Point>> contours; //holds contour positions

vector<Point> largestContour;

vector<vector<Point>> largestContourHolder(1);

Scalar color = Scalar(0, 0, 255);

// HUE for YELLOW is 21-30.

//HSV reference table

//Yellow H: 21-30

//Pink H: 149-179 S: 110 - 228 V: 185,255

while (charCheckForEscKey != 27 && capWebcam.isOpened()) { // until the Esc is pressed or webcam connection is lost

bool blnFrameReadSuccessfully = capWebcam.read(imgOriginal); // get next frame

if (!blnFrameReadSuccessfully || imgOriginal.empty()) { // if frame read unsuccessfully

cout << "error: frame can't read \n"; // print error message

break; // jump out of loop

}

cvtColor(imgOriginal, hsvImg, CV\_BGR2HSV); // Convert Original Image to HSV Thresh Image

inRange(hsvImg, Scalar(lowH, lowS, lowV), Scalar(highH, highS, highV), threshImg);

GaussianBlur(threshImg, threshImg, Size(3, 3), 0); //Blur Effect

dilate(threshImg, threshImg, 0); // Dilate Filter Effect

erode(threshImg, threshImg, 0); // Erode Filter Effect

dilate(threshImg, threshImg, 0);

erode(threshImg, threshImg, 0);

Moments oMoments = moments(threshImg);

//TODO: Fuck with the two parameters (3 and 4)

findContours(threshImg, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_NONE);

//draws the contour

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

//drawContours(imgOriginal, contours, i, color, 3, 8);

}

//TODO: Calculate an area of said contour

//TODO: Maybe look at fitEllipse? Probably good with contour

double dM01 = oMoments.m01;

double dM10 = oMoments.m10;

double dArea = oMoments.m00;

//added 4/26/18

//this filters out the tiny contours (for the most part) and returns us the biggest contour value

for (unsigned int i = 1; i < contours.size(); i++)

{

//std::cout << "Contour number: " << i << " Area: " << contourArea(contours[i]) << std::endl;

if (contourArea(contours[i]) > contourArea(contours[i - 1])) {

largestContour = contours[i];

}

}

cout << contourArea(largestContour) << endl;

largestContourHolder[0] = largestContour;

drawContours(imgOriginal, largestContourHolder, 0, color, 3, 8);

//Anything in this if handles the x and y positions

if (dArea > 10000) {

int posX = dM10 / dArea;

int posY = dM01 / dArea;

//Put the little circle on the center

if (iLastX >= 0 && iLastY >= 0 && posX >= 0 && posY >= 0) {

circle(imgOriginal, cvPoint(posX, posY), 20, Scalar(0, 255, 0));

coordsOK = true;

if (posX >= 0 && posX < 50) {

//hard left

outputss << "X: " << posX << " Y: " << posY << "| pos 1" << "|Area: " << contourArea(largestContour) << endl;

//cout << outputss.str() << endl;

positionReturn(1);

}

else if (posX >= 50 && posX < 200) {

//soft left

outputss << "X: " << posX << " Y: " << posY << "| pos 2" << "|dArea: " << dArea << endl;

//cout << outputss.str() << endl;

positionReturn(2);

}

else if (posX >= 200 && posX < 400) {

//straight on

outputss << "X: " << posX << " Y: " << posY << "| pos 3" << "|dArea: " << dArea << endl;

//cout << outputss.str() << endl;

positionReturn(3);

}

else if (posX >= 400 && posX < 550) {

//soft right

outputss << "X: " << posX << " Y: " << posY << "| pos 4" << "|dArea: " << dArea << endl;

//cout << outputss.str() << endl;

positionReturn(4);

}

else if (posX >= 550 && posX < 600) {

//hard right

outputss << "X: " << posX << " Y: " << posY << "| pos 5" << "|dArea: " << dArea << endl;

//cout << outputss.str() << endl;

positionReturn(5);

}

else {

//rotate in place

outputss << "X: " << posX << " Y: " << posY << "| pos 6" << "|dArea: " << dArea << endl;

//cout << outputss.str() << endl;

positionReturn(6);

}

}

iLastX = posX;

iLastY = posY;

/\*Legend for positioning

1: Make a hard left. Within 0-50

2: Make a soft left. Within 50-300

3: Go straight on. Within 300-600

4: Make a soft right. Within 600-850

5: Make a hard right. Within 850-900

6: Don't move forward, but rotate in place till the rest of the algo takes over.

This is for when the object is out of sight

\*/

}

// declare windows

namedWindow("imgOriginal", CV\_WINDOW\_AUTOSIZE);

namedWindow("threshImg", CV\_WINDOW\_AUTOSIZE);

/\* Create trackbars in "threshImg" window to adjust according to object and environment.\*/

/\*

createTrackbar("LowH", "threshImg", &lowH, 179); //Hue (0 - 179)

createTrackbar("HighH", "threshImg", &highH, 179);

createTrackbar("LowS", "threshImg", &lowS, 255); //Saturation (0 - 255)

createTrackbar("HighS", "threshImg", &highS, 255);

createTrackbar("LowV", "threshImg", &lowV, 255); //Value (0 - 255)

createTrackbar("HighV", "threshImg", &highV, 255);

\*/

//cout << "Width : " << threshImg.cols << endl;

//cout << "Height : " << threshImg.rows << endl;

imshow("imgOriginal", imgOriginal); // show windows

imshow("threshImg", threshImg);

charCheckForEscKey = waitKey(1); // delay and get key press

}

return(0);

}