package org.bytedeco.javacv.android.recognize.technowings;

import com.constant.ServerConstants;

import org.bytedeco.javacpp.indexer.FloatRawIndexer;

import org.bytedeco.javacpp.indexer.UByteIndexer;

import org.bytedeco.javacpp.opencv\_core;

import org.bytedeco.javacpp.opencv\_core.Mat;

import org.bytedeco.javacpp.opencv\_core.MatVector;

import org.bytedeco.javacpp.opencv\_core.Rect;

import org.bytedeco.javacv.FFmpegFrameGrabber;

import org.bytedeco.javacv.Frame;

import org.bytedeco.javacv.FrameGrabber;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import util.opencv.OpenCVHelper;

import util.opencv.Polygon;

import util.opencv.Polygon.Builder;

import static org.bytedeco.javacpp.opencv\_core.FONT\_HERSHEY\_PLAIN;

import static org.bytedeco.javacpp.opencv\_core.Moments;

import static org.bytedeco.javacpp.opencv\_core.Point;

import static org.bytedeco.javacpp.opencv\_core.Scalar;

import static org.bytedeco.javacpp.opencv\_core.Size;

import static org.bytedeco.javacpp.opencv\_core.TermCriteria;

import static org.bytedeco.javacpp.opencv\_core.cvRound;

import static org.bytedeco.javacpp.opencv\_highgui.destroyAllWindows;

import static org.bytedeco.javacpp.opencv\_highgui.waitKey;

import static org.bytedeco.javacpp.opencv\_imgproc.CV\_BGR2GRAY;

import static org.bytedeco.javacpp.opencv\_imgproc.CV\_CHAIN\_APPROX\_SIMPLE;

import static org.bytedeco.javacpp.opencv\_imgproc.CV\_RETR\_EXTERNAL;

import static org.bytedeco.javacpp.opencv\_imgproc.boundingRect;

import static org.bytedeco.javacpp.opencv\_imgproc.circle;

import static org.bytedeco.javacpp.opencv\_imgproc.contourArea;

import static org.bytedeco.javacpp.opencv\_imgproc.cvtColor;

import static org.bytedeco.javacpp.opencv\_imgproc.drawContours;

import static org.bytedeco.javacpp.opencv\_imgproc.findContours;

import static org.bytedeco.javacpp.opencv\_imgproc.goodFeaturesToTrack;

import static org.bytedeco.javacpp.opencv\_imgproc.line;

import static org.bytedeco.javacpp.opencv\_imgproc.moments;

import static org.bytedeco.javacpp.opencv\_imgproc.putText;

import static org.bytedeco.javacpp.opencv\_imgproc.rectangle;

import static org.bytedeco.javacpp.opencv\_video.BackgroundSubtractorMOG2;

import static org.bytedeco.javacpp.opencv\_video.OPTFLOW\_FARNEBACK\_GAUSSIAN;

import static org.bytedeco.javacpp.opencv\_video.calcOpticalFlowFarneback;

import static org.bytedeco.javacpp.opencv\_video.calcOpticalFlowPyrLK;

import static org.bytedeco.javacpp.opencv\_video.createBackgroundSubtractorMOG2;

public class PersonCounting extends CommonFunctions {

ArrayList<ArrayList<int[]>> contourList = new ArrayList<ArrayList<int[]>>();

ArrayList<Boolean> countedList = new ArrayList<Boolean>();

int pid = 0;

double learningRate = 0.001;

// BackgroundSubtractorKNN mog2 = createBackgroundSubtractorKNN(500, 400,

// true);// 500,10,

// true

BackgroundSubtractorMOG2 mog2 = createBackgroundSubtractorMOG2(500, 200, true);// 500,

// 10,

// true

boolean showWindow = false;

boolean saveImage = true;

boolean showSysout = true;

boolean drawColor = true;

// BackgroundSubtractorMOG2 mog2 = createBackgroundSubtractorMOG2(500, 10,

// true);//500, 10, true

Mat prev = null, current = null;

int last = 0;

Polygon polygon = null;

Point enterCountDrawPoint = new Point(ServerConstants.enterTop.x(), ServerConstants.enterTop.y() + 40);

Point exitCountDrawPoint = new Point(ServerConstants.exitBottom.x(), ServerConstants.exitBottom.y() - 40);

VideoWriter videoWriter = null;

public PersonCounting() {

// TODO Auto-generated constructor stub

videoWriter = new VideoWriter();

}

public static void main(String[] args) {

PersonCounting pc = new PersonCounting();

// Mat mat =

// imread("D:\\work\\project\\OMRProject\\PersonCounting-Eclipse\\Test\\database\\filename018.bmp");

// Mat GrayImage = new Mat(mat.rows(), mat.cols(), 1);

// cvtColor(mat, GrayImage, CV\_BGR2GRAY);

// Mat mat2 =

// imread("D:\\work\\project\\OMRProject\\PersonCounting-Eclipse\\Test\\database\\filename019.bmp");

// Mat GrayImage2 = new Mat(mat2.rows(), mat2.cols(), 1);

// cvtColor(mat2, GrayImage2, CV\_BGR2GRAY);

// // flow = cv2.calcOpticalFlowFarneback(prevgray, gray, 0.5, 5, 15, 3,

// // 5,

// // 1.1, cv2.OPTFLOW\_FARNEBACK\_GAUSSIAN)

// Mat flow = new Mat();

// // // calcOpticalFlowPyrLK(, arg1, arg2, arg3, arg4, arg5)

// calcOpticalFlowFarneback(GrayImage, GrayImage2, flow, 0.5, 5, 15, 3,

// 5, 1.1, OPTFLOW\_FARNEBACK\_GAUSSIAN);

// System.out.println(flow.rows() + " " + flow.cols() + " " +

// flow.type());

// FloatRawIndexer imagePixels2 = flow.createIndexer();

// for (int y = 0; y < GrayImage2.rows(); y += 20) {

// // Mat row = flow.row(y);

// for (int x = 0; x < GrayImage2.cols(); x += 20)

// {

// // get the flow from y, x position \* 3 for better visibility

// float fx=imagePixels2.get( y,x,0);

// float fy=imagePixels2.get( y,x,1);

// System.out.println();

// // int xx= get(0);

// // int yy= row.getIntBuffer(x).get(1);

// // System.out.println(fx+" === "+fy);

// // // draw line at flow direction

// line(mat2, new Point(x, y), new Point(cvRound(x + fx), cvRound(y +

// fy)),new Scalar(0,255, 0, 0));

// // draw initial point

// // circle(GrayImage2,new Point(x, y), 1,new Scalar(0,0, 0, 0), -1);

// circle(mat2, new Point(x,y), 4, new opencv\_core.Scalar(255, 0, 0,

// 0));

// }

// }

//

// // pc.printImagePixels(flow);

// showWindow("OpticalFlow", mat2);

// Mat mat =

//

// imread("D:\\work\\project\\OMRProject\\PersonCounting-Eclipse\\Test\\dataset\\1518592041693.jpg",

// CV\_8UC1);

// System.out.println(mat.channels());

//

// showWindow("testing", mat);

// waitKey();

//

pc.polygon = pc.createPolygon();

// pc.startCamera();

// pc.startCameraFeatureTracking2();

// pc.startCameraFeatureTracking2();

// pc.startFolderScan();

pc.startCameraFeatureTracking2();

}

// public void draw(Mat flow ) {

// int stepSize = 4;

//

// for (int y = 0; y < flow.rows(); y+=stepSize) {

// for (int x = 0; x < flow.cols(); x+=stepSize) {

// PVector flowVec = getFlowAt(x,y);

// parent.line(x, y, x+flowVec.x, y+flowVec.y);

// }

// }

// }

boolean breakFlag = true;

public void removeShadows(Mat mat) {

int oldValue = 127;

int newValue = 0;

UByteIndexer imagePixels2 = mat.createIndexer();

// UByteRawIndexer imagePixels2 = mat.createIndexer();

// System.out.println();

for (int col = 0; col < mat.cols(); col++) {

for (int row = 0; row < mat.rows(); row++) {

int pix = imagePixels2.get(row, col);

// System.out.print(pix+",");

if (pix < 250) {

// mat.pt.put(row, col,0);

imagePixels2.put(row, col, newValue);

}

// imagePixels2.pu

// if (pix >= 20) {

// whitePoints[col]++;

// }

}

// System.out.println();

}

}

// public void startFolderScan() {

// namedWindow("LiveCamera", CV\_WINDOW\_AUTOSIZE);

// namedWindow("RemoveShadows", CV\_WINDOW\_AUTOSIZE);

// namedWindow("Colored", CV\_WINDOW\_AUTOSIZE);

//

// String folderPath =

// "D:\\work\\project\\OMRProject\\PersonCounting-Eclipse\\Test\\database\\test";

// File f = new File(folderPath);

// File[] arr = f.listFiles();

// java.util.Arrays.sort(arr, new java.util.Comparator<File>() {

// public int compare(File f1, File f2) {

// return Long.compare(f1.lastModified(), f2.lastModified());

// }

// });

// int key2Check = KeyEvent.VK\_SPACE;

// int key2Check2 = KeyEvent.VK\_ENTER;

// int keychode = 1;

// for (int i = 0; i < arr.length; i++) {

// File current = arr[i];

// Mat mat = OpenCVHelper.file2mat(current.getAbsolutePath());

// // showWindow("image", mat);

// applyCounting(mat);

// mat.release();

// int key = cvWaitKey(keychode);

// if (key == key2Check) {

// keychode = 0;

// } else if (key == KeyEvent.VK\_Q) {

// keychode = 1;

// } else if (key == KeyEvent.VK\_A) {

// System.out.println("LEFT ");

// i = i - 2;

// } else if (key == KeyEvent.VK\_D) {

// System.out.println("RIGHT ");

// i = i + 1;

// }

// // if(keychode==0&&key==key2Check2){

// // keychode=1;

// // }

//

// }

// mog2.close();

// destroyAllWindows();

// }

VideoWriter writer = new VideoWriter();

public void startVideo() {

try {

PersonCountingNew pn = new PersonCountingNew();

// D:/work/project/OMRProject/PersonCounting-Eclipse/Test

FFmpegFrameGrabber grabber = new FFmpegFrameGrabber(

"E:\\Downloads\\Files\_downloaded\_by\_AirDroid\_38\\Original\_Video\_22\_Feb\_07\_09\_.mp4");

// long frameTime = 1;

// grabber.init();

// grabber.setFrameRate(25);

grabber.start();

System.out.println("grabber.getFrameRate() " + grabber.getFrameRate());

int i = 0;

long t = System.currentTimeMillis();

int nullFrames = 0;

while (breakFlag) {

// if (frameTime > 0) {

// grabber.setTimestamp(frameTime \* 1000l); // microseconds

// }

Frame f = grabber.grabFrame();

if (f != null && f.image != null) {

// frameTime++;

Mat mat = OpenCVHelper.frame2mat(f);

// showWindow("Mat ",mat);

// // applyCounting(mat);

// System.out.println(" mat "+i+" "+(System.currentTimeMillis()-t));

pn.countFrame(mat);

writer.writeFrame(mat);

mat.release();

i++;

} else {

nullFrames++;

if (nullFrames > 30) {

break;

}

}

waitKey(35);

}

writer.stopVideoIfAny();

grabber.stop();

grabber.close();

} catch (FrameGrabber.Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// VideoCapture vc = new

// VideoCapture("D:/work/project/OMRProject/PersonCounting-Eclipse/Test/database/Video\_13\_Feb\_10\_47\_.avi");

// while (true) {

// if (vc.isOpened()) {

// Mat mat = new Mat();

// vc.read(mat);

// applyCOunting(mat);

// int k = waitKey(1);

// if (k == 13)

// break;

// mat.release();

// }

// }

// vc.close();

}

public void startCamera() {

try {

FrameGrabber grabber = FrameGrabber.createDefault(ServerConstants.SURVELIANCE\_CAMERA\_NUM);

// FrameGrabber grabber = FrameGrabber.createDefault(new

// File("D:/work/project/OMRProject/PersonCounting-Eclipse/Test/database/Video\_13\_Feb\_10\_47\_.avi"));

grabber.setImageWidth(ServerConstants.IMAGE\_WIDTH);

grabber.setImageHeight(ServerConstants.IMAGE\_HEIGHT);

grabber.start();

int nullFrames = 0;

while (true) {

Frame f = grabber.grab();

if (f != null) {

Mat mat = OpenCVHelper.frame2mat(f);

showWindow("Windows", mat);

// System.out.println("Starting to recognize");

// applyCounting(mat);

try {

// Thread.sleep(100);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

} else {

nullFrames++;

if (nullFrames > 30) {

break;

}

}

int k = waitKey(1);

if (k == 13)

break;

}

videoWriter.stopVideoIfAny();

grabber.release();

mog2.close();

destroyAllWindows();

} catch (Exception ex) {

ex.printStackTrace();

}

}

int previousContour = -1;

public void startCameraFeatureTracking2() {

try {

// int key2Check = KeyEvent.VK\_SPACE;

// int key2Check2 = KeyEvent.VK\_ENTER;

int keychode = 1;

int track\_len = 10;

int detect\_interval = 5;

int frame\_idx = 0;

Mat current\_mat\_gray = null;

Mat prev\_mat\_gray = null;

Mat vis = null;

ArrayList<ArrayList<Float[]>> tracks = new ArrayList<ArrayList<Float[]>>();

int maxlevel = 2;

int flags = 0;

double minEigThreshold = 1e-4;

TermCriteria criteria = new TermCriteria(TermCriteria.EPS | TermCriteria.COUNT, 10, 0.3);

Mat prominentFeatures = null;

boolean useFolder = true;

// String folderPath =

// "D:\\work\\project\\OMRProject\\PersonCounting-Eclipse\\Test\\test-case-2";

// File f = new File(folderPath);

// File[] arr = f.listFiles();

// java.util.Arrays.sort(arr, new java.util.Comparator<File>() {

// public int compare(File f1, File f2) {

// return Long.compare(f1.lastModified(), f2.lastModified());

// }

// });

FrameGrabber grabber = null;

grabber = FrameGrabber.createDefault(ServerConstants.SURVELIANCE\_CAMERA\_NUM);

grabber.setImageWidth(ServerConstants.IMAGE\_WIDTH);

grabber.setImageHeight(ServerConstants.IMAGE\_HEIGHT);

grabber.start();

while (true) {

Frame frame = grabber.grab();

// Mat mat\_color =

// OpenCVHelper.file2mat(arr[frame\_idx].getAbsolutePath());

// if (mat\_color != null) {

if (frame != null) {

Mat mat\_color = OpenCVHelper.frame2mat(frame);

// im et\_color);

current\_mat\_gray = new Mat(mat\_color.rows(), mat\_color.cols(), 1);

cvtColor(mat\_color, current\_mat\_gray, CV\_BGR2GRAY);

// // Background Start

// Mat foreground = new Mat();

// mog2.apply(current\_mat\_gray, foreground, learningRate);

// // // removeShadows(foreground); ArrayList<Polygon> rects

// =

// applyContour3(foreground, mat\_color);

// Background Start

if (vis != null) {

vis.release();

}

vis = mat\_color.clone();

drawEnterExitLine(vis);

if (tracks.size() > 0) {

Mat img0 = prev\_mat\_gray;

Mat img1 = current\_mat\_gray;

Mat p0\_prev\_features = createMat(tracks);//

Mat p1 = new Mat();

Mat st = new Mat();

Mat err = new Mat();

calcOpticalFlowPyrLK(img0, img1, p0\_prev\_features, p1, st, err, new Size(15, 15), maxlevel, criteria, flags,

minEigThreshold);

Mat p0r = new Mat();

calcOpticalFlowPyrLK(img1, img0, p1, p0r, st, err, new Size(15, 15), maxlevel, criteria, flags, minEigThreshold);

ArrayList<Float[]> p0\_prev\_featuresArr = getImageData(p0\_prev\_features);

ArrayList<Float[]> p0r\_prev\_featuresArr = getImageData(p0r);

ArrayList<Float[]> p1\_featuresArr = getImageData(p1);

p1.release();

p0r.release();

st.release();

err.release();

boolean[] good = new boolean[p0\_prev\_featuresArr.size()];

for (int i = 0; i < p0\_prev\_featuresArr.size() && i < p0r\_prev\_featuresArr.size(); i++) {

Float[] p0\_points = p0\_prev\_featuresArr.get(i);

Float[] p0r\_points = p0r\_prev\_featuresArr.get(i);

float diffX = Math.abs(p0\_points[0] - p0r\_points[0]);

float diffY = Math.abs(p0\_points[1] - p0r\_points[1]);

float max = diffX > diffY ? diffX : diffY;

good[i] = max < 1 ? true : false;

}

ArrayList<ArrayList<Float[]>> new\_tracks = new ArrayList<ArrayList<Float[]>>();

for (int i = 0; i < p1\_featuresArr.size() && i < tracks.size() && i < good.length; i++) {

if (!good[i]) {

continue;

}

ArrayList<Float[]> a = tracks.get(i);

Float[] points = p1\_featuresArr.get(i);

a.add(points);

while (a.size() > track\_len) {

a.remove(0);

}

tracks.set(i, a);

circle(vis, new Point((int) points[0].floatValue(), (int) points[1].floatValue()), 2, new opencv\_core.Scalar(0,

255, 0, 0));

new\_tracks.add(a);

}

tracks = new\_tracks;

HashMap counting = new HashMap();

for (int j = 0; j < tracks.size(); j++) {

ArrayList<Float[]> newF2 = tracks.get(j);

if (newF2.size() > 1) {

Float prev[] = newF2.get(0);

boolean b = false;

for (int k = 1; k < newF2.size(); k++) {

Float current[] = newF2.get(k);

Point oldFeature = new Point((int) prev[0].floatValue(), (int) prev[1].floatValue());

Point newFeature = new Point((int) current[0].floatValue(), (int) current[1].floatValue());

line(vis, oldFeature, newFeature, new Scalar(255, 255, 0, 0), 1, 8, 0);

prev = current;

if (!b) {

boolean entryCondition = (newFeature.x() > oldFeature.x()

&& newFeature.x() >= ServerConstants.START\_POINT && oldFeature.x() < ServerConstants.START\_POINT);

boolean exitCondition = (oldFeature.x() > newFeature.x()

&& oldFeature.x() >= ServerConstants.START\_POINT && newFeature.x() < ServerConstants.START\_POINT);

// // Contour Checking Code start

if (entryCondition || exitCondition) {

// int contourIndex = -1;

// Polygon p = null;

// for (int i = 0; i < rects.size();

// i++) {

// p = rects.get(i);

// if (p.contains(new

// util.opencv.Point(newFeature.x(),

// newFeature.y()))) {

// contourIndex = i;

// break;

// }

// }

// if (contourIndex == -1) {

// continue;

// } else {

// if (counting.get(contourIndex) !=

// null) {

// continue;

// } else {

// System.out.println("Counting Contour "

// + contourIndex + " " + p);

// counting.put(contourIndex, 1);

// }

// }

// // Contour Checking Code End

if (!b && entryCondition) {

b = true;

ServerConstants.enterCount++;

}

if (!b && exitCondition) {

b = true;

ServerConstants.exitCount++;

}

}

putText(vis, "Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint, FONT\_HERSHEY\_PLAIN,

2, new opencv\_core.Scalar(0, 255, 0, 0));

putText(vis, "Exit-" + ServerConstants.exitCount, exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2,

new opencv\_core.Scalar(0, 255, 0, 0));

}

// System.out.println("Drawing Line");

}

}

}

counting.clear();

// ArrayList<Float[]> promF =

// getImageData(prominentFeatures);

// ArrayList<Integer> statusArray =

// getImageData(status);

}

// if (tracks.size() == 0) {

// }

if ((frame\_idx % detect\_interval) == 0) {

Mat mask = current\_mat\_gray.clone();

Mat ones = new Mat(Scalar.WHITE);

mask.setTo(ones);

for (int j = 0; j < tracks.size(); j++) {

ArrayList<Float[]> newF2 = tracks.get(j);

Float[] newF = newF2.get(newF2.size() - 1);

Point point = new Point((int) newF[0].floatValue(), (int) newF[1].floatValue());

circle(mask, point, 5, new Scalar(0, 0, 0, 0), -1, 1, 0);

}

if (prominentFeatures != null) {

prominentFeatures.release();

}

prominentFeatures = new Mat();

goodFeaturesToTrack(current\_mat\_gray, prominentFeatures, 500, 0.3, 7, mask, 7, true, 0.04);

if (prominentFeatures != null && !prominentFeatures.empty()) {

ArrayList<Float[]> promF = getImageData(prominentFeatures);

System.out.println("tracks.size() " + promF.size() + " " + tracks.size());

prominentFeatures.release();

for (Iterator iterator = promF.iterator(); iterator.hasNext();) {

Float[] floats = (Float[]) iterator.next();

ArrayList<Float[]> a = new ArrayList<Float[]>();

a.add(floats);

tracks.add(a);

}

// ArrayList<Float[]> a = new ArrayList<Float[]>();

// a.add(new Float[]{new Float(126),new

// Float(300)});

// tracks.add(a);

}

mask.release();

ones.release();

// showWindow("SHOW", mask);

// cvWaitKey();

}

waitKey(5);

showWindow("lk\_tracks", vis);

frame\_idx += 1;

if (prev\_mat\_gray != null) {

prev\_mat\_gray.release();

prev\_mat\_gray.deallocateReferences();

}

prev\_mat\_gray = current\_mat\_gray.clone();

current\_mat\_gray.release();

current\_mat\_gray.deallocateReferences();

// showWindow("back", foreground);

// foreground.release();

// int key = cvWaitKey(keychode);

// if (key == key2Check) {

// keychode = 0;

// } else if (key == KeyEvent.VK\_Q) {

// keychode = 1;

// } else if (key == KeyEvent.VK\_A) {

// System.out.println("LEFT ");

// frame\_idx = frame\_idx - 2;

// } else if (key == KeyEvent.VK\_D) {

// System.out.println("RIGHT ");

// frame\_idx = frame\_idx + 1;

// }

}

}

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

}

public void startCameraFeatureTracking() {

try {

FrameGrabber grabber = FrameGrabber.createDefault(ServerConstants.SURVELIANCE\_CAMERA\_NUM);

// FrameGrabber grabber = FrameGrabber.createDefault(new

// File("D:/work/project/OMRProject/PersonCounting-Eclipse/Test/database/Video\_13\_Feb\_10\_47\_.avi"));

grabber.setImageWidth(ServerConstants.IMAGE\_WIDTH);

grabber.setImageHeight(ServerConstants.IMAGE\_HEIGHT);

grabber.start();

// Mat mask = null;

Mat prominentFeatures = null;

Mat oldGray = null;

Mat drawMat = null;

// Mat oldGrayOrginal = null;

while (true) {

Frame f = grabber.grab();

if (f != null) {

Mat mat = OpenCVHelper.frame2mat(f);

drawMat = mat.clone();

prominentFeatures = new Mat();

oldGray = new Mat(mat.rows(), mat.cols(), 1);

// mask = mat.clone();

// mask=new Mat(mat.rows(),mat.cols(),mat.type());

// mask.zeros(mat.size(),mat.type());

// showWindow("MASK", mask);

cvtColor(mat, oldGray, CV\_BGR2GRAY);

drawEnterExitLine(oldGray);

goodFeaturesToTrack(oldGray, prominentFeatures, 100, 0.3, 7);

showWindow("OldGray", oldGray);

// oldGrayOrginal = oldGray.clone();

// oldGray.release();

mat.release();

waitKey(10);

break;

}

}

while (true) {

Frame f = grabber.grab();

if (f != null) {

Mat mat = OpenCVHelper.frame2mat(f);

Mat currentGray = new Mat(mat.rows(), mat.cols(), 1);

cvtColor(mat, currentGray, CV\_BGR2GRAY);

drawEnterExitLine(drawMat);

//

// Mat opGray = new Mat(mat.rows(), mat.cols(), 1);

// absdiff(currentGray, oldGrayOrginal, opGray);

// // threshold(opGray, opGray, 25, 255, THRESH\_BINARY)

// GaussianBlur(opGray, opGray, new Size(11, 11), 0);;

// // removeShadows(opGray);

// // threshold(opGray, opGray, 25, 255, THRESH\_BINARY);

//

// adaptiveThreshold(opGray, opGray, 255,

// ADAPTIVE\_THRESH\_GAUSSIAN\_C, THRESH\_BINARY\_INV, 5, 2);

// showWindow("Backgroundsubtraction", opGray);

// int contourCount = countContours(opGray);

//

// System.out.println("contourCount " + contourCount);

// opGray.release();

Mat newFeatures = new Mat();

Mat status = new Mat();

Mat error = new Mat();

int maxlevel = 2;

int flags = 0;

double minEigThreshold = 1e-4;

showWindow("CurrentGray", currentGray);

TermCriteria criteria = new TermCriteria(TermCriteria.EPS + TermCriteria.COUNT, 10, 0.3);

calcOpticalFlowPyrLK(oldGray, currentGray, prominentFeatures, newFeatures, status, error, new Size(15, 15), maxlevel,

criteria, flags, minEigThreshold);

ArrayList<Float[]> newF = getImageData(newFeatures);

ArrayList<Float[]> promF = getImageData(prominentFeatures);

ArrayList<Integer> statusArray = getImageData(status);

oldGray.release();

oldGray = currentGray.clone();

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

int stat = statusArray.get(i);

if (stat == 1) {

Point oldFeature = new Point((int) promF.get(i)[0].floatValue(), (int) promF.get(i)[1].floatValue());

Point newFeature = new Point((int) newF.get(i)[0].floatValue(), (int) newF.get(i)[1].floatValue());

if (newFeature.x() > oldFeature.x() && newFeature.x() >= ServerConstants.START\_POINT

&& oldFeature.x() < ServerConstants.START\_POINT) {

ServerConstants.enterCount++;

}

if (oldFeature.x() > newFeature.x() && oldFeature.x() >= ServerConstants.START\_POINT

&& newFeature.x() < ServerConstants.START\_POINT) {

ServerConstants.exitCount++;

}

putText(drawMat, "Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2,

new Scalar(0, 255, 0, 0));

putText(drawMat, "Exit-" + ServerConstants.exitCount, exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2, new Scalar(0,

255, 0, 0));

circle(drawMat, newFeature, 4, new Scalar(0, 0, 255, 0));

line(drawMat, oldFeature, newFeature, new Scalar(255, 255, 255, 0), 1, 8, 0);

}

showWindow("CurrentGray", currentGray);

showWindow("POINTS", drawMat);

showWindow("OldGray", oldGray);

}

// MatExpr expr= add(currentGray,mask);

// showWindow("expr", mask);

if (t % 5 == 0) {

drawMat.release();

drawMat = currentGray.clone();

System.err.println("========================================");

// oldGray.release();

// oldGray = currentGray.clone();

putText(drawMat, "IDX-" + t, new Point(20, 20), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

prominentFeatures.release();

prominentFeatures = new Mat();

goodFeaturesToTrack(oldGray, prominentFeatures, 100, 0.3, 7);

ArrayList<Float[]> promF2 = getImageData(prominentFeatures);

for (int i = 0; i < promF2.size(); i++) {

Float[] newF2 = promF2.get(i);

circle(drawMat, new Point((int) newF2[0].floatValue(), (int) newF2[1].floatValue()), 4,

new Scalar(255, 0, 0, 0));

}

t = 0;

} else {

prominentFeatures.release();

prominentFeatures = newFeatures.clone();

}

showWindow("CurrentGray", currentGray);

showWindow("POINTS", drawMat);

showWindow("OldGray", oldGray);

// drawMat.release();

newFeatures.release();

mat.release();

currentGray.release();

status.release();

error.release();

int k = waitKey(1);

t++;

// previousContour=contourCount;

// if(false){

// break;

// }

}

}

// grabber.release();

// mog2.close();

// destroyAllWindows();

} catch (Exception ex) {

ex.printStackTrace();

}

}

long t = 0;

public Polygon createPolygon() {

Builder builder = Polygon.Builder();

builder.addVertex(new util.opencv.Point(ServerConstants.START\_POINT - ServerConstants.GAP\_BETWEEN\_START\_END / 2, 0));

builder.addVertex(new util.opencv.Point(ServerConstants.START\_POINT - ServerConstants.GAP\_BETWEEN\_START\_END / 2,

ServerConstants.IMAGE\_HEIGHT));

builder.addVertex(new util.opencv.Point(ServerConstants.START\_POINT + ServerConstants.GAP\_BETWEEN\_START\_END / 2,

ServerConstants.IMAGE\_HEIGHT));

builder.addVertex(new util.opencv.Point(ServerConstants.START\_POINT + ServerConstants.GAP\_BETWEEN\_START\_END / 2, 0));

builder.addVertex(new util.opencv.Point(ServerConstants.START\_POINT - ServerConstants.GAP\_BETWEEN\_START\_END / 2, 0));

Polygon polygon = builder.build();

return polygon;

}

public int countContours(Mat thresholdedImage) {

int count = 0;

Mat contourImage = thresholdedImage.clone();

MatVector contours = new MatVector();

findContours(contourImage, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE); // CV\_RETR\_EXTERNAL

int areaMarkerBulletMin = 1000;

int areaMarkerBulletMAX = 100000;

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

Mat currentContour = contours.get(contourIndex);

if (currentContour != null) {

Rect boundRect = boundingRect(currentContour);

int contour\_width = boundRect.width();

int contour\_height = boundRect.height();

double areaOfContour = (contour\_width \* contour\_height);

// int cx = boundRect.x() + contour\_width / 2;

// int cy = boundRect.y() + contour\_height / 2;

if (areaOfContour > areaMarkerBulletMin && areaOfContour < areaMarkerBulletMAX) {

count++;

}

}

}

contours.deallocateReferences();

contourImage.release();

return count;

}

Mat newMat = null;

public void applyCounting(Mat mat) {

// newMat = mat.clone();

// showWindow("LiveCamera", newMat);

// // applyContour(mat);

// videoWriter.writeFrame(mat);

// applyBackgroundFiltering(newMat);

// mat.release();

// newMat.release();

}

// private void applyBackgroundFiltering(Mat colorImg) {

// // Mat m=getStructuringElement(MORPH\_ELLIPSE,new Size(3,3));

// // int history=100;

// // int dist2Threshold =400;

// // Pointer p=new Pointer();

// Mat grayImage = new Mat(colorImg.rows(), colorImg.cols(), 1);

// cvtColor(colorImg, grayImage, CV\_BGR2GRAY);

// // // if(true){

// GaussianBlur(grayImage, grayImage, new Size(11, 11), 0);

// current = grayImage.clone();

//

// drawEnterExitLine(newMat);

// // // adaptiveThreshold(GrayImage, GrayImage, 255,

// // // ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY, 5, 2);

// // // }else{

// // threshold(GrayImage, GrayImage, 0, 255, THRESH\_OTSU );

// // BackgroundSubtractorKNN knn=new BackgroundSubtractorKNN(p);

//

// // BackgroundSubtractorMOG2

// // mog2=createBackgroundSubtractorMOG2(500,16,true);

// // BackgroundSubtractorMOG2 mog2 = createBackgroundSubtractorMOG2(30,

// // 10, true);

//

// // mog2.setHistory(10);

// // mog2.setVarThreshold(16);

// // mog2.setDetectShadows(false);

// // Mat GrayImage = new Mat(img.rows(), img.cols(), 1);

// // cvtColor(img, GrayImage, CV\_BGR2GRAY);

// // if(true){

// // GaussianBlur(GrayImage, GrayImage, new Size(11,11), 0);

// // adaptiveThreshold(GrayImage, GrayImage, 255,

// // ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY, 5, 2);

// // threshold(GrayImage, GrayImage, 0, 255, THRESH\_OTSU);

//

// // }else{

// Mat foreground = new Mat();

// mog2.apply(grayImage, foreground, learningRate);

//

// // Mat back = new Mat();

// // mog2.getBackgroundImage(back);

// // // showWindow("back", back);

// // back.release();

//

// Mat m = new Mat(3);

// erode(foreground, foreground, m);

// // dilate(foreground, foreground, m);

// // threshold(foreground, foreground, 120, 255,

// // THRESH\_OTSU|THRESH\_BINARY\_INV);

// m.release();

// // imshow("Dilation", foreground);

// removeShadows(foreground);

// showWindow("RemoveShadows", foreground);

// // imwrite("./dataset/" + System.currentTimeMillis() + ".jpg",

// // foreground);

//

// //

// if ((System.currentTimeMillis() - videoWriter.START) > (10 \* 1000)) {

//

// applyContour(foreground, grayImage);

// }

//

// // if (last >= 10) {

// if (prev != null) {

// prev.release();

// prev.deallocateReferences();

// }

// prev = current.clone();

// last = 0;

// // }

// last++;

// current.release();

// // threshold(foreground, foreground, 128, 255, THRESH\_BINARY);

// // imwrite("test.png", foreground);

// // mog2.getBackgroundImage(foreground);

//

// // foreground.release();

// foreground.release();

// grayImage.release();

// // GrayImage.release();

// // mog2.close();

// // m.release();

//

// }

HashMap countCountours = new HashMap();

private ArrayList<Polygon> applyContour2(Mat thresholdedImage, Mat colored) {

Mat colorClone = colored.clone();

Mat contourImage = thresholdedImage.clone();

MatVector contours = new MatVector();

findContours(contourImage, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE); // CV\_RETR\_EXTERNAL

int areaMarkerBulletMin = 20000;

int areaMarkerBulletMAX = 200000;

ArrayList<Polygon> arr = new ArrayList<Polygon>();

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

Mat currentContour = contours.get(contourIndex);

if (currentContour != null) {

Rect boundRect = boundingRect(currentContour);

int contour\_width = boundRect.width();

int contour\_height = boundRect.height();

double areaOfContour = (contour\_width \* contour\_height);

int cx = boundRect.x() + contour\_width / 2;

int cy = boundRect.y() + contour\_height / 2;

if (areaOfContour > areaMarkerBulletMin && areaOfContour < areaMarkerBulletMAX) {

circle(colorClone, new Point(cx, cy), 4, new Scalar(255, 0, 0, 0));

// if (countCountours.get(contour\_width + "\_" +

// contour\_height + "\_" + cx + "\_" + cy) == null) {

// countCountours.put(contour\_width + "\_" + contour\_height +

// "\_" + cx + "\_" + cy, 1);

Builder polyBuilder = new Builder();

polyBuilder.addVertex(new util.opencv.Point(boundRect.x(), boundRect.y()));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x() + contour\_width, boundRect.y()));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x() + contour\_width, boundRect.y() + contour\_height));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x(), boundRect.y() + contour\_height));

arr.add(polyBuilder.build());

if (ServerConstants.drawColor) {

putText(colorClone, "C" + contourIndex + " (" + contour\_width + "," + contour\_height + ")", new Point(

boundRect.x(), boundRect.y()), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

rectangle(colorClone, new Point(boundRect.x(), boundRect.y()), new Point(boundRect.x() + boundRect.width(),

boundRect.y() + boundRect.height()), Scalar.YELLOW, 1, 8, 0);

// PolyXYPoints.release();

}

}

currentContour.release();

}

}

showWindow("Colored", colorClone);

// colorClone.deallocateReferences();

colorClone.release();

contourImage.release();

contourImage.deallocateReferences();

contours.deallocateReferences();

return arr;

// GrayImage.release();

}

private void applyContour3(Mat thresholdedImage, Mat colored) {

Mat colorClone = colored.clone();

Mat contourImage = thresholdedImage.clone();

// Mat m = new Mat(3);

// dilate(contourImage, contourImage, m);

MatVector contours = new MatVector();

findContours(contourImage, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE); // CV\_RETR\_EXTERNAL

int areaMarkerBulletMin = 10000;

int areaMarkerBulletMAX = 200000;

ArrayList<Polygon> arr = new ArrayList<Polygon>();

int detected = 0;

putText(colorClone, "<<---Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2, new Scalar(0, 255,

0, 0));

putText(colorClone, "Exit-" + ServerConstants.exitCount + "--->>", exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2, new Scalar(0, 255, 0,

0));

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

Mat currentContour = contours.get(contourIndex);

if (currentContour != null) {

Rect boundRect = boundingRect(currentContour);

int contour\_width = boundRect.width();

int contour\_height = boundRect.height();

double areaOfContour = (contour\_width \* contour\_height);

Moments moments = moments(currentContour);

double ma = contourArea(currentContour);

int mcx = (int) (moments.m10() / moments.m00());

int mcy = (int) (moments.m01() / moments.m00());

int cx = boundRect.x() + contour\_width / 2;

int cy = boundRect.y() + contour\_height / 2;

int x = boundRect.x();

int y = boundRect.y();

if (areaOfContour > areaMarkerBulletMin && areaOfContour < areaMarkerBulletMAX) {

detected++;

boolean found = false;

boolean countedOnce = false;

System.out.println("contourList.size() " + contourList.size());

// Find The contour that has minimum distance with prev

// point

int minIndex = -1;

double minDisance = 9999;

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

ArrayList<int[]> a = contourList.get(i);

boolean isAlreadyCounted = countedList.get(i);

// for (int j = 0; j < a.size(); j++) {

int[] lastXY = a.get(a.size() - 1);

if (Math.abs(x - lastXY[0]) <= contour\_width && Math.abs(y - lastXY[1]) <= contour\_height) {

double d = Math.sqrt((x - lastXY[0]) \* (x - lastXY[0]) + (y - lastXY[1]) \* (y - lastXY[1]));

if (d < minDisance) {

minIndex = i;

}

}

}

// for (int i = 0; i < contourList.size(); i++)

if (minIndex != -1) {

ArrayList<int[]> a = contourList.get(minIndex);

boolean isAlreadyCounted = countedList.get(minIndex);

// for (int j = 0; j < a.size(); j++) {

int[] lastXY = a.get(a.size() - 1);

if (Math.abs(x - lastXY[0]) <= contour\_width && Math.abs(y - lastXY[1]) <= contour\_height) {

found = true;

a.add(new int[] { mcx, mcy });

if (a.size() > 1) {

if (!isAlreadyCounted) {

int[] prev = a.get(a.size() - 2);

int[] current = a.get(a.size() - 1);

boolean entryCondition = (prev[0] > current[0] && prev[0] >= ServerConstants.START\_POINT && current[0] < ServerConstants.START\_POINT);

boolean exitCondition = (current[0] > prev[0] && current[0] >= ServerConstants.START\_POINT && prev[0] < ServerConstants.START\_POINT);

if (entryCondition) {

System.err.println("Counted Prev " + prev[0] + "," + prev[1] + " Current " + current[0] + ","

+ current[1]);

countedOnce = true;

ServerConstants.enterCount++;

countedList.set(minIndex, true);

}

if (!countedOnce && exitCondition) {

System.err.println("Counted Prev " + prev[0] + "," + prev[1] + " Current " + current[0] + ","

+ current[1]);

countedOnce = true;

ServerConstants.exitCount++;

countedList.set(minIndex, true);

}

if (countedOnce) {

break;

}

}

}

putText(colorClone, "<<---Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint, FONT\_HERSHEY\_PLAIN,

2, new Scalar(0, 255, 0, 0));

putText(colorClone, "Exit-" + ServerConstants.exitCount + "--->>", exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2,

new Scalar(0, 255, 0, 0));

}

// break;

// }

while (a.size() > 10) {

a.remove(0);

}

if (found) {

break;

}

}

if (!found) {

ArrayList<int[]> at = new ArrayList<int[]>();

at.add(new int[] { mcx, mcy });

contourList.add(at);

countedList.add(false);

}

if (contourList.size() == 0) {

ArrayList<int[]> at = new ArrayList<int[]>();

at.add(new int[] { mcx, mcy });

contourList.add(at);

countedList.add(false);

}

circle(colorClone, new Point(mcx, mcy), 4, new Scalar(255, 255, 0, 0));

drawContours(colorClone, contours, contourIndex, new Scalar(0, 255, 0, 0));

// if (countCountours.get(contour\_width + "\_" +

// contour\_height + "\_" + cx + "\_" + cy) == null) {

// countCountours.put(contour\_width + "\_" + contour\_height +

// "\_" + cx + "\_" + cy, 1);

Builder polyBuilder = new Builder();

polyBuilder.addVertex(new util.opencv.Point(boundRect.x(), boundRect.y()));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x() + contour\_width, boundRect.y()));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x() + contour\_width, boundRect.y() + contour\_height));

polyBuilder.addVertex(new util.opencv.Point(boundRect.x(), boundRect.y() + contour\_height));

arr.add(polyBuilder.build());

if (ServerConstants.drawColor) {

putText(colorClone, "C" + contourIndex + " (" + contour\_width + "," + contour\_height + ")", new Point(

boundRect.x(), boundRect.y()), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

rectangle(colorClone, new Point(boundRect.x(), boundRect.y()), new Point(boundRect.x() + boundRect.width(),

boundRect.y() + boundRect.height()), Scalar.YELLOW, 1, 8, 0);

// PolyXYPoints.release();

}

}

currentContour.release();

}

}

if (detected == 0) {

contourList.clear();

countedList.clear();

}

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

ArrayList<int[]> a = contourList.get(i);

if (a.size() == 1) {

int[] prev = a.get(0);

circle(colorClone, new Point(prev[0], prev[1]), 4, new Scalar(255, 255, 0, 0));

putText(colorClone, "X", new Point(prev[0], prev[1]), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

}

if (a.size() > 1) {

int[] prev = a.get(0);

for (int j = 1; j < a.size(); j++) {

int[] xy = a.get(j);

line(colorClone, new Point(prev[0], prev[1]), new Point(cvRound(xy[0]), cvRound(xy[1])), new Scalar(0, 255, 0, 0), 1,

8, 0);

putText(colorClone, "X", new Point(xy[0], xy[1]), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

prev = xy;

}

}

}

showWindow("Colored", colorClone);

// colorClone.deallocateReferences();

colorClone.release();

contourImage.release();

contourImage.deallocateReferences();

contours.deallocateReferences();

// return arr;

// GrayImage.release();

}

private void applyContour(Mat thresholdedImage, Mat colored) {

if (current == null || prev == null) {

return;

}

Mat colorClone = newMat.clone();

putText(colorClone, "Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2,

new Scalar(0, 255, 0, 0));

putText(colorClone, "Exit-" + ServerConstants.exitCount, exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 2, new Scalar(0, 255, 0, 0));

Mat flow = new Mat();

// // calcOpticalFlowPyrLK(, arg1, arg2, arg3, arg4, arg5)

calcOpticalFlowFarneback(current, prev, flow, 0.5, 5, 15, 3, 5, 1.1, OPTFLOW\_FARNEBACK\_GAUSSIAN);

FloatRawIndexer opticalFlowMat = flow.createIndexer();

Mat contourImage = thresholdedImage.clone();

MatVector contours = new MatVector();

findContours(contourImage, contours, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE); // CV\_RETR\_EXTERNAL

int areaMarkerBulletMin = 10000;

int areaMarkerBulletMAX = 100000;

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

Mat currentContour = contours.get(contourIndex);

if (currentContour != null) {

Rect boundRect = boundingRect(currentContour);

int contour\_width = boundRect.width();

int contour\_height = boundRect.height();

double areaOfContour = (contour\_width \* contour\_height);

int cx = boundRect.x() + contour\_width / 2;

int cy = boundRect.y() + contour\_height / 2;

if (areaOfContour > areaMarkerBulletMin && areaOfContour < areaMarkerBulletMAX) {

circle(colorClone, new Point(cx, cy), 4, new Scalar(255, 0, 0, 0));

float fx = opticalFlowMat.get(cy, cx, 0); // y

// row

// x=cols

float fy = opticalFlowMat.get(cy, cx, 1);

line(colorClone, new Point(cx, cy), new Point(cvRound(cx + fx), cvRound(cy + fy)), new Scalar(0, 255, 0, 0), 1, 8, 0);

if (countCountours.get(contour\_width + "\_" + contour\_height + "\_" + cx + "\_" + cy) == null) {

countCountours.put(contour\_width + "\_" + contour\_height + "\_" + cx + "\_" + cy, 1);

if (ServerConstants.drawColor) {

putText(colorClone, "Contour-" + contourIndex + " (" + boundRect.x() + "," + boundRect.y() + ")", new Point(

boundRect.x(), boundRect.y()), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 255, 0, 0));

if (polygon.contains(new util.opencv.Point(cx, cy))) {

rectangle(colorClone, new Point(boundRect.x(), boundRect.y()), new Point(boundRect.x() + boundRect.width(),

boundRect.y() + boundRect.height()), Scalar.RED, 1, 8, 0);

// System.out.println("Found Contour");

int distanceEntry = Math.abs(cx - ServerConstants.enterTop.x());

int distanceExit = Math.abs(cx - ServerConstants.exitTop.x());

if (distanceEntry < distanceExit) {

log("Enter Contour [" + contourIndex + "] " + cx + "," + cy + " Contour Width/Heigh " + contour\_width

+ " " + contour\_height + " area " + (contour\_width \* contour\_height));

// Point start = new

// opencv\_core.Point(enterTop.x(),

// enterTop.y()

// + 40);

ServerConstants.enterCount++;

putText(colorClone, "Enter-" + ServerConstants.enterCount + "", enterCountDrawPoint,

FONT\_HERSHEY\_PLAIN, 1, new Scalar(255, 255, 255, 0));

log("Directions [" + contourIndex + "] " + cx + "," + cy + " NEXT X Y " + cvRound(cx + fx) + " "

+ cvRound(cy + fy));

highlightEnter(newMat);

} else {

log("Exit Contour [" + contourIndex + "] " + cx + "," + cy + " Contour Width/Heigh " + contour\_width

+ " " + contour\_height + " area " + (contour\_width \* contour\_height));

line(colorClone, new Point(cx, cy), new Point(cvRound(cx + fx), cvRound(cy + fy)), new Scalar(0, 255,

0, 0), 1, 8, 0);

ServerConstants.exitCount++;

// Point start = new

// opencv\_core.Point(exitBottom.x(),exitBottom.y()

// - 40);

line(colorClone, new Point(cx, cy), new Point(cvRound(cx + fx), cvRound(cy + fy)), new Scalar(0, 255,

0, 0), 1, 8, 0);

log("Directions [" + contourIndex + "] " + cx + "," + cy + " NEXT X Y " + cvRound(cx + fx) + " "

+ cvRound(cy + fy));

putText(colorClone, "Exit-" + ServerConstants.exitCount, exitCountDrawPoint, FONT\_HERSHEY\_PLAIN, 1,

new Scalar(0, 255, 0, 0));

highlightExit(newMat);

}

} else {

rectangle(colorClone, new Point(boundRect.x(), boundRect.y()), new Point(boundRect.x() + boundRect.width(),

boundRect.y() + boundRect.height()), Scalar.GREEN, 1, 8, 0);

putText(colorClone, "Same Contour", new Point(cx, cy), FONT\_HERSHEY\_PLAIN, 1, new Scalar(0, 0, 255, 0));

}

}

} else {

rectangle(colorClone, new Point(boundRect.x(), boundRect.y()), new Point(boundRect.x() + boundRect.width(),

boundRect.y() + boundRect.height()), Scalar.YELLOW, 1, 8, 0);

}

// PolyXYPoints.release();

}

currentContour.release();

}

}

opticalFlowMat.release();

flow.release();

showWindow("Colored", colorClone);

colorClone.deallocateReferences();

colorClone.release();

contourImage.release();

contourImage.deallocateReferences();

contours.deallocateReferences();

contours.deallocate();

// GrayImage.release();

}

}