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

import com.constant.ServerConstants;

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

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

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.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.createBackgroundSubtractorMOG2;

import java.awt.event.KeyEvent;

import java.io.File;

import java.util.ArrayList;

import java.util.Arrays;

import org.bytedeco.javacpp.indexer.UByteIndexer;

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

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

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

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

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

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

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

import org.bytedeco.javacv.FFmpegFrameGrabber;

import org.bytedeco.javacv.Frame;

import org.bytedeco.javacv.FrameGrabber;

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

import util.opencv.OpenCVHelper;

public class PersonCountingNew extends CommonFunctions {

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

// //

// 10,

// //

// true

double learningRate = 0.001;

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

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

boolean showWindow = true;

boolean saveImage = true;

boolean showSysout = true;

boolean drawColor = true;

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

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

VideoWriter video = new VideoWriter();

public static void main(String[] args) {

PersonCountingNew p = new PersonCountingNew();

String folderPath = "E:\\Downloads\\Files\_downloaded\_by\_AirDroid\_37\\";

File f = new File(folderPath);

File[] arr = f.listFiles();

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

public int compare(File f1, File f2) {

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

}

});

int keychode = 1;

int key2Check = KeyEvent.VK\_SPACE;

try {

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

File current = arr[i];

if (!current.isFile()) {

continue;

}

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

p.video.writeFrame(mat);

// showWindow("image", mat);

p.countFrame(mat);

mat.release();

int key = waitKey(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;

// }

}

} catch (Exception e) {

e.printStackTrace();

p.video.stopVideoIfAny();

}

}

VideoWriter writer = new VideoWriter();

public void startVideoTesting() {

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\_02\_55\_.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 (true) {

// 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();

}

}

public void countFrame(Mat mat\_color) {

if (showWindow) {

showWindow("LiveCamera", mat\_color);

}

Mat 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);

if (showWindow) {

showWindow("Background", foreground);

}

foreground.release();

}

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();

}

}

private void applyContour3(Mat thresholdedImage, Mat colored) {

Mat colorClone = colored;

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;

int detected = 0;

if (drawColor) {

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

0, 255, 0));

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

255, 0));

}

drawEnterExitLine(colored);

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 (ma > areaMarkerBulletMin && ma < areaMarkerBulletMAX) {

detected++;

boolean found = false;

boolean countedOnce = false;

log("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) {

if (showSysout)

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) {

if (showSysout)

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;

}

}

}

if (drawColor) {

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

FONT\_HERSHEY\_PLAIN, 2, new Scalar(0, 0, 255, 0));

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

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

}

if (drawColor) {

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

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

}

if (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);

}

}

currentContour.release();

}

}

if (detected == 0) {

contourList.clear();

countedList.clear();

}

if (drawColor) {

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;

}

}

}

}

if (showWindow) {

showWindow("Colored", colorClone);

}

// colorClone.deallocateReferences();

// colorClone.release();

contourImage.release();

contourImage.deallocateReferences();

contours.deallocateReferences();

contours.deallocateReferences();

// return arr;

// GrayImage.release();

}

}