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\_highgui.waitKey;

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

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

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

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

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

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

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

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import org.bytedeco.javacpp.opencv\_core;

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

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

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

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

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

import org.bytedeco.javacv.Frame;

import org.bytedeco.javacv.FrameGrabber;

import util.opencv.OpenCVHelper;

public class OpticalFlowTracking extends CommonFunctions {

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

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

public static void main(String[] args) {

OpticalFlowTracking pc = new OpticalFlowTracking();

pc.startCameraFeatureTracking2();

}

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();

}

}

}