Приложение А

```
import cv2.cv as cv
from cv2.cv import *
import math
import numpy as np
import json
from shapely.geometry import LineString
with open('config.json', 'r') as f:
    config = json.load(f)

first_point = 0
cur_track_id = 0
cur_count_id = 0
point_with_no_track = ()
list_of_tracks = []
list_of_counters = []


class Statistics:
    def __init__(self):
        self.epsilon = config['epsilon']

    def get_equation_coefficients(self, line):
        f_x = line.xy[0][0]
        f_y = line.xy[0][1]
        s_x = line.xy[1][0]
        s_y = line.xy[1][1]
        A = [[f_x, 1], [f_y, 1]]
        b = [s_x, s_y]
        A = np.array(A)
        b = np.array(b)
        x = list(np.linalg.solve(A, b))
        return x[0], x[1]

    def line_as_a_function(self, x, a, b):
        return a * x + b

    def get_objects_crossing_line_count_up_down(self, counter, track, line):
        prev = track.coordinates[len(track.coordinates) - 2]
        cur = track.coordinates[len(track.coordinates) - 1]
        line_from_track = LineString([prev, cur])
        a_coefficient_line, b_coefficient_line = Statistics.get_equation_coefficients(self,
        line)
        line_function_value_prev = Statistics.line_as_a_function(self, line_from_track.xy[0][
        0], a_coefficient_line, b_coefficient_line)
        line_function_value_cur = Statistics.line_as_a_function(self, line_from_track.xy[1][0
        ], a_coefficient_line, b_coefficient_line)
        track_function_value_prev = prev[1]
        track_function_value_cur = cur[1]
        if line_function_value_cur + self.epsilon > track_function_value_cur >
        line_function_value_cur - self.epsilon and not counter.counter_id in track.
        list_of_counters:
            track.set_epsilon_true(counter.counter_id)
            if track_function_value_prev > line_function_value_prev + self.epsilon and
            track_function_value_prev > line_function_value_prev - self.epsilon:
                counter.add_up_counter()
            elif track_function_value_prev < line_function_value_prev + self.epsilon and
            track_function_value_prev < line_function_value_prev - self.epsilon:
                counter.add_down_counter()
```

```
        return counter.counter_up_value, counter.counter_down_value


class Counter:
    def __init__(self):
        self.counter_up_value = 0
        self.counter_down_value = 0
        self.counter_id = 0

    def add_new_counter_id(self, counter_id):
        self.counter_id = counter_id

    def add_up_counter(self):
        self.counter_up_value += 1

    def add_down_counter(self):
        self.counter_down_value += 1


class Visualization():
    def __init__(self):
        self.line_color = (250, 200, 0)

    def add_line(self, color_image, frame_height, frame_width, font, line_p1_x=0, line_p2_x=
    200, line_p1_y=200,
                 line_p2_y=200, line_color=(100, 200, 0), line_width=10):
        cv.Line(color_image, (line_p1_x, line_p1_y), (line_p2_x, line_p2_y), line_color,
        line_width)

    def add_text(self, color_image, p_x, p_y, font, title="up", text_color=(250, 200, 0)):
        cv.PutText(color_image, title, (p_x, p_y), font, text_color)


class Track:
    def __init__(self):
        self.track_id = ""
        self.coordinates = []
        self.list_of_counters = []
        self.coordinates_time = []
        self.coordinates_speed = []

    def add_point(self, point):
        self.coordinates.append(point)

    def add_point_time(self, time):
        self.coordinates_time.append(time)

    def set_track_id(self, track_id):
        self.track_id = "track %d" % track_id

    def set_epsilon_true(self, counter_id):
        self.list_of_counters.append(counter_id)


class Tracking:
    def __init__(self):
        self.max_distance_between = config['max_distance_between']
        self.min_distance_between = config['min_distance_between']
```

```
        self.list_of_points = []
        self.track = Track()

    def find_nearest_track(self, point, cur_time):
        global first_point, point_with_no_track
        point_with_no_track = ()
        we_found_nearest_track = False
        if first_point == 0:
            point_with_no_track = point
            first_point = 1
        if first_point == 2:
            for track in list_of_tracks:
                if track:
                    last_track_coord = len(track.coordinates) - 1
                    distance_between = math.hypot(track.coordinates[last_track_coord][0] -
                    point[0],
                                                  track.coordinates[last_track_coord][1] -
                                                  point[1])
                    if (distance_between <= self.max_distance_between) and distance_between
                    >= self.min_distance_between:
                        track.add_point(point)
                        we_found_nearest_track = True
            if not we_found_nearest_track:
                point_with_no_track = point
        return first_point, point_with_no_track

    def add_points_to_tracks(self, point, cur_time):
        global first_point, cur_track_id, point_with_no_track
        first_point, point_with_no_track = self.find_nearest_track(point, cur_time)
        if point_with_no_track:
            if first_point == 1:
                self.track.set_track_id(cur_track_id)
                self.track.add_point(point_with_no_track)
                list_of_tracks.append(self.track)
                cur_track_id += 1
                first_point = 2
            else:
                self.track.set_track_id(cur_track_id)
                self.track.add_point(point_with_no_track)
                list_of_tracks.append(self.track)
                cur_track_id += 1


class BorderLine:
    def __init__(self):
        self.line_point_1 = []
        self.line_point_2 = []
        self.line = LineString()

    def set_line_points(self, line_point1, line_point2):
        self.line_point1 = line_point1
        self.line_point2 = line_point2
        self.line = LineString([self.line_point1, self.line_point2])


class Target:
    def __init__(self):
        self.capture = cv.CaptureFromFile(config['video_file'])
```

```
        frame = cv.QueryFrame(self.capture)
        self.frame_size = cv.GetSize(frame)
        self.grey_image = cv.CreateImage(self.frame_size, cv.IPL_DEPTH_8U, 1)
        self.moving_average = cv.CreateImage(self.frame_size, cv.IPL_DEPTH_32F, 3)
        self.min_area = config['min_area']
        self.max_area = config['max_area']
        self.frame_width = self.frame_size[0]
        self.frame_height = self.frame_size[1]
        self.list_of_points = []


    def image_difference(self, first):
        global temp, moving_difference
        color_image = cv.QueryFrame(self.capture)
        if first:
            moving_difference = cv.CloneImage(color_image)
            temp = cv.CloneImage(color_image)
            first = False
        cv.AbsDiff(color_image, temp, moving_difference)
        cv.CvtColor(moving_difference, self.grey_image, cv.CV_RGB2GRAY)
        cv.Threshold(self.grey_image, self.grey_image, 70, 255, cv.CV_THRESH_BINARY)
        cv.Dilate(self.grey_image, self.grey_image, None, 18)
        return color_image, first

    def add_contour_in_storage(self):
        storage = cv.CreateMemStorage(0)
        contour = cv.FindContours(self.grey_image, storage, cv.CV_RETR_CCOMP, cv.
        CV_CHAIN_APPROX_SIMPLE)
        return contour

    @staticmethod
    def get_rectangle_parameters(bound_rect, color_image):
        pt1 = (bound_rect[0], bound_rect[1])
        pt2 = (bound_rect[0] + bound_rect[2], bound_rect[1] + bound_rect[3])
        x_center = abs(pt1[0] - pt2[0]) / 2 + pt1[0]
        y_center = abs(pt1[1] - pt2[1]) / 2 + pt1[1]
        point = (x_center, y_center)
        y_length = abs(pt1[0] - pt2[0])
        x_length = abs(pt1[1] - pt2[1])
        area = x_length * y_length
        return pt1, pt2, point, area

    def get_points_tracking(self, point, area):
        if self.min_area < area < self.max_area:
            self.list_of_points.append(point)

    def run(self):
        line_point_1 = config['line_point_1']
        line_point_2 = config['line_point_2']
        line = LineString([line_point_1, line_point_2])
        counter = Counter()
        statistic = Statistics()
        counter.add_new_counter_id(cur_count_id)
        list_of_counters.append(counter)
        first = True
        writer = cv.CreateVideoWriter("out.avi", cv.CV_FOURCC('D','I','V','X'), 30, self.
        frame_size, True)
        while True:
```

```
            color_image, first = self.image_difference(first)
            contour = self.add_contour_in_storage()
            font = cv.InitFont(cv.CV_FONT_HERSHEY_SIMPLEX, 1, 0.2, 0, 1, 1)
            line_and_text = Visualization()
            line_and_text.add_line(color_image, self.frame_height, self.frame_width, font,
            line_p1_x=line_point_1[0],
                                   line_p1_y=line_point_1[1], line_p2_x=line_point_2[0],
                                   line_p2_y=line_point_2[1],
                                   line_width=4, line_color=(0, 222, 322))
            while contour:
                bound_rect = cv.BoundingRect(list(contour))
                contour = contour.h_next()
                pt1, pt2, point, area = self.get_rectangle_parameters(bound_rect, color_image)
                cv.Rectangle(color_image, pt1, pt2, cv.CV_RGB(255, 0, 0), 1)
                self.list_of_points = []
                self.get_points_tracking(point, area)
                tracking = Tracking()
                cur_time = GetCaptureProperty(self.capture, CV_CAP_PROP_POS_MSEC)
                for point in self.list_of_points:
                    tracking.add_points_to_tracks(point, cur_time)
                for track in list_of_tracks:
                    track.add_point_time(cur_time)
                    if len(track.coordinates_time) > 5*len(track.coordinates):
                        list_of_tracks.remove(track)
                    x_prev = track.coordinates[len(track.coordinates) - 2][0]
                    y_prev = track.coordinates[len(track.coordinates) - 2][1]
                    cur_x = track.coordinates[len(track.coordinates) - 1][0]
                    cur_y = track.coordinates[len(track.coordinates) - 1][1]
                    line_and_text.add_line(color_image, self.frame_height, self.frame_width,
                    font, line_p1_x=x_prev,
                                           line_p1_y=y_prev, line_p2_x=cur_x, line_p2_y=cur_y
                                           , line_width=4,
                                           line_color=(0, 222, 322))
                    line_and_text.add_text(color_image, cur_x, cur_y, font, title=track.
                    track_id)

                    up, down = statistic.get_objects_crossing_line_count_up_down(counter,
                    track, line)
                    line_and_text.add_text(color_image, int(line.xy[0][0]), int(line.xy[1][0
                    ])-10, font, text_color=(100, 200, 50), title="U " + str(up))
                    line_and_text.add_text(color_image, int(line.xy[0][0]), int(line.xy[1][0
                    ])+20, font, text_color=(100, 200, 50), title="D " + str(down))

            cv.WriteFrame(writer, self.grey_image)

            cv.ShowImage("target", color_image)
            cv.ShowImage("target", self.grey_image)
            c = cv.WaitKey(config['wait_key']) % 0x100
            if c == 27:
                break
if __name__ == "__main__":
    t = Target()
    t.run()
```