Командный практический тур

Описание задачи финала

Создать и запустить мультимодальную транспортную систему для доставки товара с фабрики до покупателя без вмешательства человека. Беспилотный автомобиль везет груз на склад через весь город, соблюдая правила дорожного движения. На складе товар проходит автоматизированную сортировку и подается на квадрокоптер. Далее коптер в режиме автономного полета доставляет груз до конечного покупателя и отправляется на дозаправку.

Работа участников происходит на большом полигоне городской среды, на котором расположены здания и дороги с перекрестками, дорожными знаками, светофорами и пешеходами.

Беспилотный автомобиль

Беспилотный автомобиль АЙКАР передвигается по городу. Его задача — проехать трассу с соблюдением ПДД и доставить груз до распределительного хаба. Ему предстоит детектировать объекты на маршруте следования, распознавать их и корректно реагировать.

Задача участников — написать программы для детектирования, распознавания и реагирования на объекты городской среды, встречающиеся на маршруте беспилотника: светофоры, дорожные знаки, пешеходные переходы и самих пешеходов. Автомобиль должен двигаться по своей полосе дорожного полотна и преодолеть маршрут, соблюдая правила дорожного движения.

Распределительный хаб

Конвейер забирает груз с беспилотного автомобиля при помощи магнитного захвата, перемещает в накопитель и проводит автоматизированную сортировку.

Задача участников — написать программы для распознавания числовой маркировки на коробках и проведения сортировки грузов при помощи магнитного манипулятора, расположенного над лентой конвейера.

Квадрокоптер

Квадрокоптер получает от сервера адрес заказчика, подхватывает груз при помощи магнитного захвата и далее в режиме автономного полета сначала доставляет посылку по нужному адресу, а затем отправляется на дозаправку.

Задача участников — написать программу автономного полета квадрокоптера, которая позволит ему достигать мест назначения, ориентируясь по графическим меткам на поверхности полигона, стабилизировать свое положение над данными метками и помещать груз в их центр так, чтобы он не повредился.

Ход финала

Задания финала выполняются на испытательном полигоне.

Размер полигона: 6×8 м.

Рис. III.2.1: Вид полигона

На полигоне присутствуют 4 объекта:

1. Беспилотный автомобиль «Айкар».
2. Беспилотный автомобиль «Айкар».
3. Квадрокоптер «Пионер Макс».
4. Конвейер внутри распределительного хаба.
5. Каждым объектом управляет оператор, посредник между участниками и устройством.

Связь с полигоном

Участники подключаются к компьютерам операторов при помощи ПО AnyDesk и присылают файлы с программами. По указанию участников оператор загружает программы на устройство и запускает их, переносит устройство в разные положения и редактирует код программ. В случае с квадрокоптерами каждой команде дается свой испытательный стенд, к которому она может подключаться напрямую и проводить испытания без участия оператора. За происходящим на полигоне участники могут следить через трансляции в YouTube с разных камер.

Ход работы

Основную часть времени участники работают на своих рабочих местах: создают и редактируют программы. В отведенное расписанием время участники подключаются к операторам оборудования и присылают свои программы. Операторы запускают данные программы на оборудовании, участники видят результат работы на трансляциях с полигона.

Этап 1

Этап предполагает работу с двумя устройствами: компьютером оператора и беспилотным автомобилем АЙКАР. Беспилотный автомобиль передает видеопоток на компьютер по Wi-Fi. Компьютер обрабатывает видеопоток и отправляет беспилотному автомобилю управляющие сигналы: скорость и угол поворота.

- server.py
  Эта программа отвечает за захват видеопотока с камеры и передачу кадров на компьютер оператора. Изменять ее запрещается!
- Cmd_receiver.py
  Эта программа принимает управляющие сигналы от компьютера оператора и направляет их двигателю и сервоприводу. Участники могут редактировать данную программу по своему усмотрению.

На беспилотном автомобиле, по умолчанию выполняются следующие программы:

На компьютере оператора выполняется программа Road_detect.py. Она обрабатывает видеопоток с беспилотника и отправляет обратно управляющие сигналы. По умолчанию программе реализовано автономное движение по своей полосе дорожной разметки без учета перекрестков и резких поворотов.

Основная часть заданий этапа решается редактированием именно этой программы!

Есть вспомогательные файлы, из которых подгружаются функции и классы, они так же доступны и могут быть отредактированы или дополнены.

Ссылка на предоставленный участникам базовый код: https://drive.google.com/file/d/1nXkmP5k8-FVQRXPzLyD_Qq5faHxaCdTs/view?usp=sharing.

Задание этапа

Беспилотный автомобиль должен преодолеть маршрут по полигону городской среды и доставить груз от места его производства до распределительного хаба. Точки старта и финиша маршрута неизменны.

Этап разбит на несколько задач. Начинать решение можно с любой из первых 6 задач.

Правила выполнения заданий

1. На полигоне присутствует 2 беспилотных автомобиля АЙКАР. Они равноценны.
2. Решения с помощью delay и sleep не принимаются!
   Измерение времени следует делать методами, не блокирующими обработку видеопотока и остальные операции.
3. При решении задач подразумевается, что беспилотник движется плавно и без рывков. Остановки возможны, например, на стоп-линии, но не более того.

Описание заданий:

Задача 1. Доехать от стартовой позиции до перекрестка и остановиться.

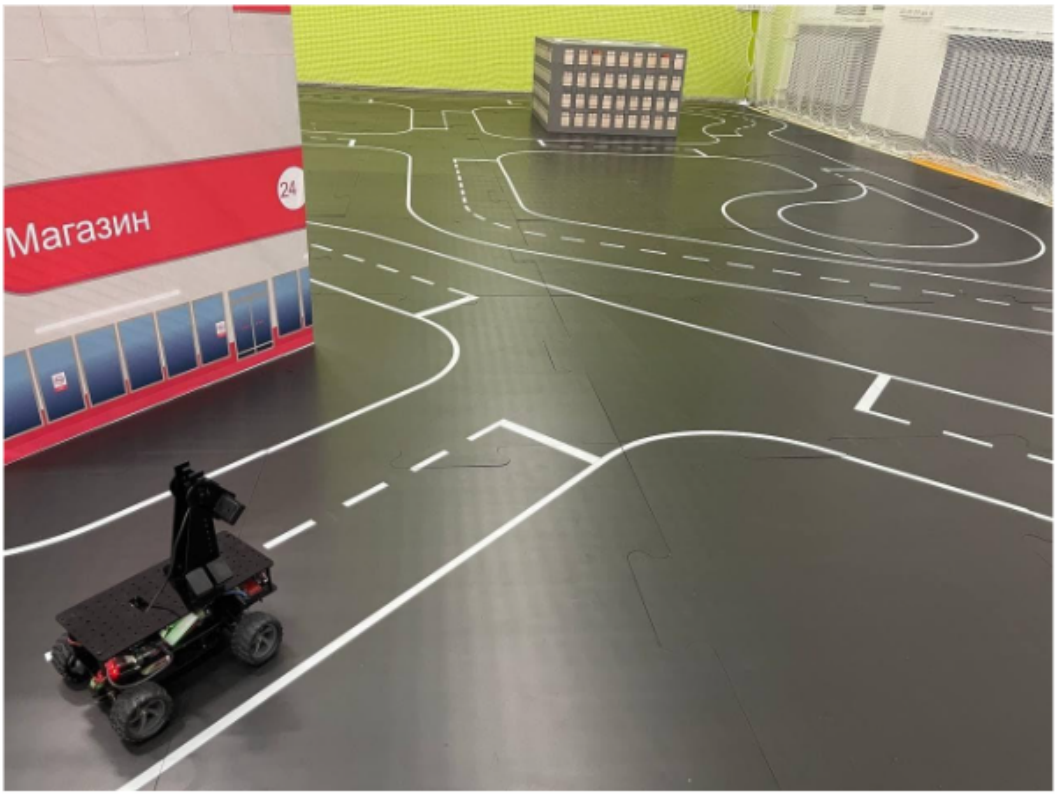
Айкар ставится в стартовое положение:

Задание считается выполненным, если беспилотный автомобиль остановился так, чтобы колеса оказались не дальше 12 см от стоп-линии, и не пересекли ее полностью:

Для проверки будет проводиться 3 заезда, каждый должен увенчаться успехом.

Задача 2. Повернуть на перекрестке направо.

Айкар устанавливается на расстоянии $\sim$ 50 см от перекрестка:



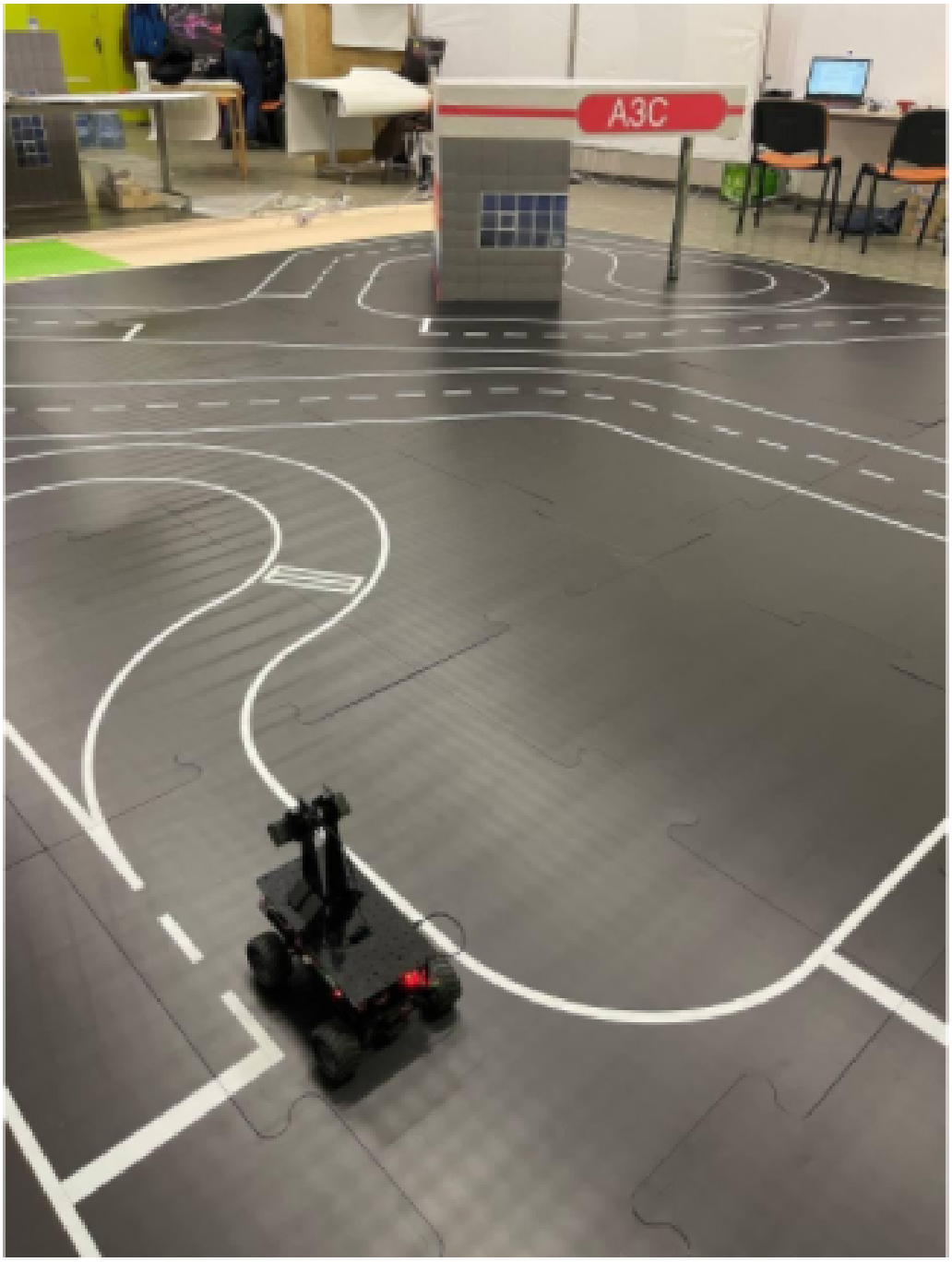
Задание считается выполненным, если беспилотник повернет направо, не заезжая на линии разметки. Останавливаться после пересечения перекрестка не обязательно. Для проверки будут проводиться 3 запуска, каждый должен увенчаться успехом.

Задача 3. Повернуть на перекрестке налево

Реализация аналогична реализации задания 2.

Задача 4. Пересечь перекресток по прямой

Реализация аналогична реализации задания 2.



Задача 5. Остановиться в точке разгрузки

Айкар устанавливается на расстоянии 50 см от распределительного хаба:

Задание считается выполненным если Айкар остановится так, чтобы магнитный захват опускался ровно на груз в кузове. Для проверки будут проводиться 3 запуска, каждый должен увенчаться успехом.

Задача 6. Доставить груз от стартовой позиции к распределительному хабу

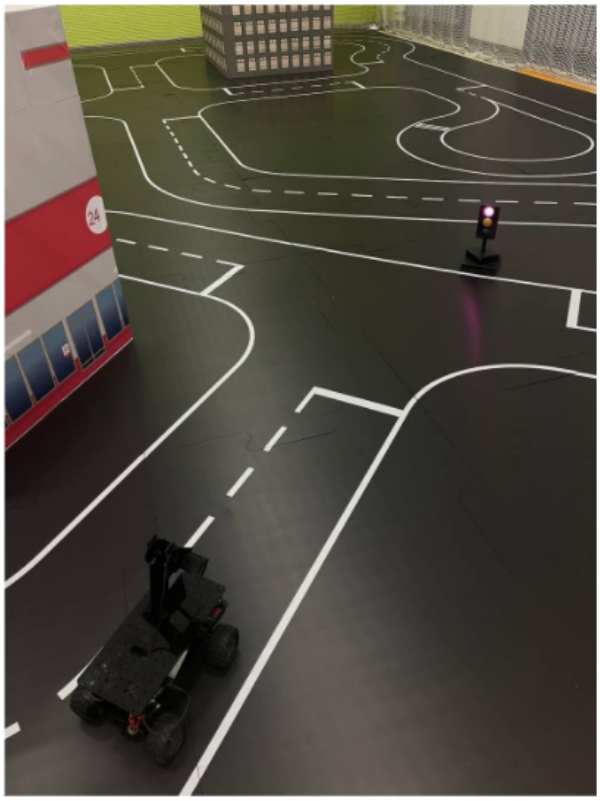
Айкар ставится в стартовое положение.

Задание считается выполненным, если беспилотный автомобиль доехал до точки разгрузки и магнитный захват опустился ровно на груз в кузове.

Для проверки будут проводиться 2 запуска, каждый должен увенчаться успехом.

Задача 7. Пересечь перекресток с учетом сигналов светофора

Айкар устанавливается на расстоянии 50 см от перекрестка со светофором:

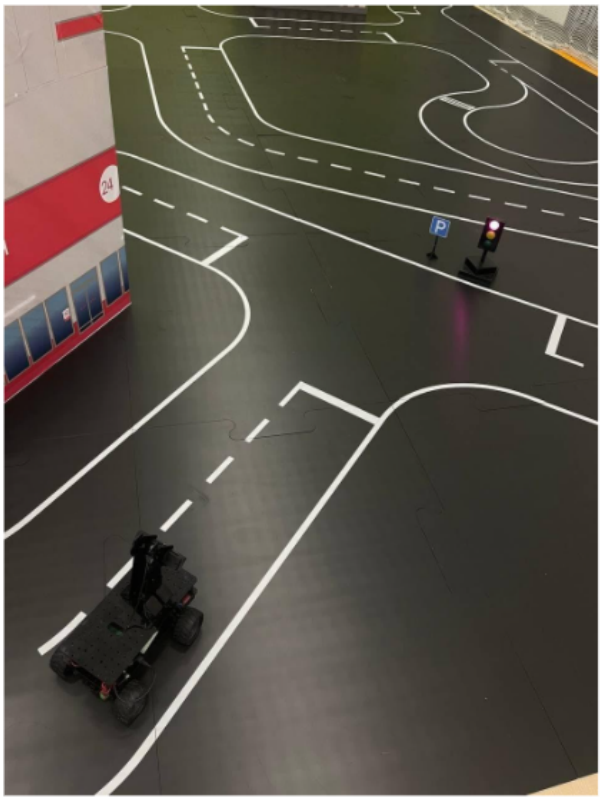


Светофор работает в автоматическом режиме, сменяются пять сигналов: красный, красный + желтый, зеленый, мигающий зеленый, желтый. Задание считается выполненным, если беспилотный автомобиль пересекает перекресток только на зеленый сигнал светофора.

Для проверки будут проводиться 3 запуска, каждый должен увенчаться успехом.

Задача 8. Пересечь перекресток с учетом сигналов светофора и дорожного знака

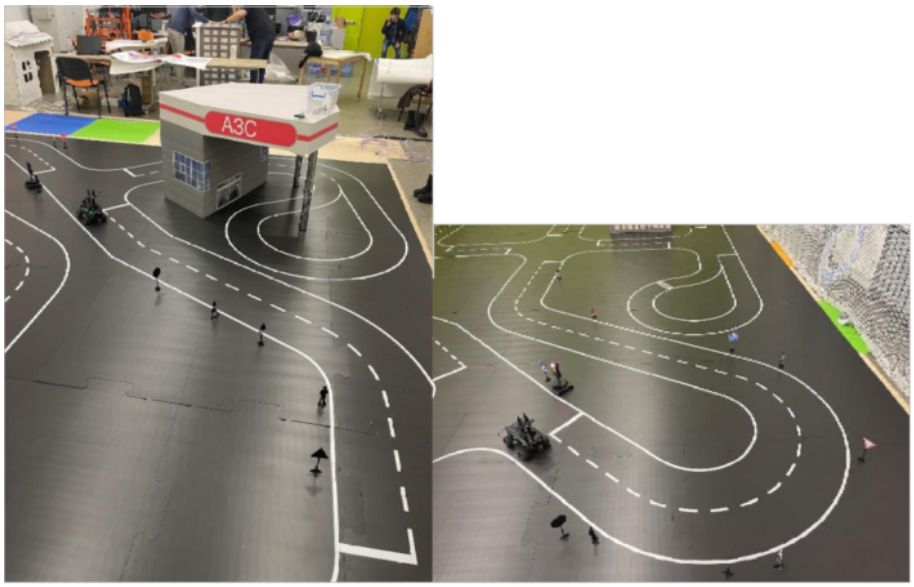
К условиям задания 7 добавляется дорожный знак:



Задание считается выполненным, если беспилотный автомобиль пересек перекресток с учетом сигналов светофора и повернул в нужную сторону: для знаков парковка и стоп — поворот налево, для знаков пешеходный переход и кипич — поворот направо. Для проверки будут проводиться 3 запуска, каждый должен увенчаться успехом.

Задача 9. Распознавание дорожных знаков

Айкар ставится в одно из двух положений:



Задание считается выполненным, если беспилотный автомобиль доехал по трассе

до стоп-линии и вывел в командную строку названия знаков встреченных на пути. Количество выведенных названий должно совпадать с числом встреченных знаков.

Число знаков может меняться от 2 до 4.

На пути следования беспилотника будут и знаки и пешеходы.

Если верно указано число знаков, а названия перепутаны, начисляется только половина баллов.

Для проверки будут проводиться 2 запуска, каждый должен увенчаться успехом.

Задача 10. Детектирование пешеходов

Айкар ставится в одно из двух положений задачи 9.

Задание считается выполненным, если беспилотный автомобиль доехал по трассе до стоп-линии и вывел в командную строку число встреченных им по пути пешеходов. На пути следования беспилотника будут и знаки и пешеходы.

Для проверки будут проводиться 2 запуска, каждый должен увенчаться успехом.

Задача 11. Доставка груза с учетом всех объектов городской среды

Айкар ставится в стартовое положение.

Задание считается выполненным если беспилотный автомобиль доехал до точки разгрузки по полному пути на полигоне и:

- преодолел перекресток со светофором и дорожным знаком,
- подсчитал и распознал встреченные дорожные знаки,
- подсчитал встреченных на пути пешеходов,
- остановился так, что магнитный захват поднял груз из кузова в точке разгрузке.

Для проверки будут проводиться 2 запуска, каждый должен увенчаться успехом.

Финальное задание этапа, составлено так, чтобы состоять из решений предшествующих задач. Таким образом, сложность решения финальной задачи состоит в составлении из нескольких алгоритмов, разработанных ранее, одного с единой логикой работы для всех под-алгоритмов.

Описание решений задача 1 этапа

Задача 1

Решение задачи подразумевает корректировку предоставленного участникам базового файла. Откорректировать необходимо коэффициенты ПИД-регулятора, пороги бинаризации для детектирования разметки и область исходного кадра, рассматриваемую как область перед колесами беспилотника. Значения следует подобрать экспериментально, либо вывести аналитически, зная параметры беспилотника и трассы.

Кроме того, необходимо добавить возможность детектирования стоп-линий перед перекрестком.

Детектирование стоп-линии, задача не сложная и как правило решается самыми простыми методами: бинаризацией и подсчетом числа белых пикселей, как и задача детектирования линий дорожной разметки. Однако, зачастую для детектирования применяют контурный анализ. Это излишне и допустимо, только пока алгоритм успевает выполняться с достаточной скоростью.

```
ret, frame = cam.read()
binary = cv2.inRange(frame, (200, 200, 200), (250, 250, 250))
vertical_hist = np.sum(binary, axis=1)
ind_max_str = np.argmax(vertical_hist)
```

Для бинаризованного изображения, достаточно посчитать сумму значений пикселей в каждой строке, выбрать строку с наибольшей суммой. Если сумма превышает определенный порог, то считаем, что стоп линия детектирована. Индекс строки, позволит понять, насколько стоп линия далеко от беспилотника и остановится иименно на том расстоянии от стоп линии, на котором требуется. При остановке следует учитывать инерцию движения беспилотника, для ее гашения, можно на несколько миллисекунд подавать напряжение на двигатель в режиме реверса.

Задачи 2, 3, 4

Помимо прочего, энкодер позволяет отмерять расстояние, пройденное беспилотником. Энкодер срабатывает известное число раз, когда беспилотник проходит известное расстояние. При срабатывании энкодера, значение переменной-счетчика увеличивается на 1. Таким образом, в срабатываниях энкодера измеряется пройденное беспилотником расстояние.

Любой маневр можно описать углом поворота колес в течении маневра и числом срабатываний энкодера, необходимым для завершения маневра. Угол поворота колес задает радиус кривизны дуги, по которой будет ехать беспилотник, а число срабатываний энкодера задает расстояние, которое беспилотник проедет по дуге.

Различные маневры преодоления перекрестка можно организовать следующим образом:

```
# список того, как нужно преодолевать перекрестки
maneuvers_list = ['Left', 'Right', 'Forvard']

# словарь хранящий углы поворота и расстояния для разных маневров.
d = {'Forvard': (87, 3000), 'Left': (71, 4900), 'Right': (102, 1500)}

# как только мы прошли перекресток - удаляем 0 элемент, тогда
d[maneuvers_list[0]][0]  # угол поворота колес, для текущего маневра
d[maneuvers_list[0]][1]  # число срабатываний энкодера, для преодоления перекрестка
```

Значения следует подобрать экспериментально, либо вывести аналитически, зная параметры беспилотника и трассы.

Допустимо составление алгоритма пересечения перекрестков без использования энкодера. Возможно организовать виртуальные датчики линии и преодолевать перекрестки ориентируясь по их показаниям. Виртуальный датчик линии, это граппа

пикселей, чаще всего квадрат. Для такого квадрата рассчитывается значение суммы всех его пикселей — значение суммы, фактически тоже самое, что аналоговый сигнал от датчика линии. Чем выше сумма, тем больше белого в квадрате.

```
ret, frame = cam.read()
binary = cv2.inRange(frame, (200, 200, 200), (250, 250, 250))
datchik1 = np.sum(binary[200:220, 100:120])
datchik2 = np.sum(binary[200:220, 300:320])
datchik3 = np.sum(binary[100:120, 190:210])
```

Разместив на бинаризованном изображении несколько виртуальных датчиков линии, можно определять место положение беспилотника и принимать решение о переходе к следующему маневру.

Задача 5

Остановиться в точке разгрузке.

Задача аналогична задаче 1, за исключением расстояния от стоп линии, на котором нужно остановиться. Необходимо было скорректировать расстояние так, чтобы захват от конвейера успешно захватывал груз.

```
ret, frame = cam.read()
binary = cv2.inRange(frame, (200, 200, 200), (250, 250, 250))
vertical_hist = np.sum(binary, axis=1)
ind_max_str = np.argmax(vertical_hist)
if vertical_hist[ind_max_str] > POROG:
    if ind_max_str > POROG2:
        Stop()
```

POROG2 отвечает за дистанцию, на которой беспилотник остановится перед стоплинией.

Задача 6

Сложность задачи в составлении ранее организованных алгоритмов в работоспособную систему. Общая схема алгоритма аналогична той, что будет разобрана ниже в задаче 11, за исключением этапов детектирования и отсева объектов.

Задача 7

Для того чтобы пересечь перекресток с учетом сигналов светофора, необходимо распознавать его сигналы. В самом простом случае, достаточно распознавать зеленый сигнал светофора. Для этого достаточно бинаризовать исходное изображение так, чтобы зеленый цвет превратился в белый, а все остальное в черный. Пороги бинаризации подбираются экспериментально, так чтобы только зажженный зеленый сигнал становился белой областью на бинаризованном изображении.

На полученном изображении следует найти контуры и выбрать наибольший по площади. Для этого контура проверить соотношение длины и площади, если оно

близко к соотношению площади круга и периметра окружности, то считаем что детектировали зеленый сигнал светофора. Чтобы отличить зеленый сигнал светофора от зеленого мигающего, достаточно засечь время после первого распознавания. Если в течении 0,6 секунд сигнал светофора не менялся, значит перед нами именно зеленый сигнал, а не мигающий зеленый.

```
def green_detect(frame):
    binary = cv2.inRange(frame, (50, 150, 50), (100, 250, 100))
    contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE,
    ↪  cv2.CHAIN_APPROX_NONE)
    if contours:
        contours = sorted(contours, key=cv2.contourArea, reverse=True)
        area = cv2.contourArea(contours[0])
        perimeter = cv2.arcLength(contours[0], True)
        x, y, w, h = cv2.boundingRect(contours[0])
        R = w / 2  # radius
        # perimetr = 2 * pi * R
        # area = pi * R * R
        # area / perimetr = pi * R * R / 2 * pi * R = R / 2 = 0.5 * R
        if (area / perimeter <= R * 0.55) and (area / perimeter >= R * 0.45):
            return True
        else:
            return False

ret, frame = cam.read()
start_time = 0
Go = False
while (Go != True):
    if green_detect(frame):
        if start_time == 0:
            start_time = time.monotonic()
    else:
        if start_time != 0:
            end_time = time.monotonic()
            if end_time - start_time >= 0.6:
                Go = True
            else:
                start_time = 0
```

После чего совершаем маневр пересечения перекрестка.

Задача 8

К описанным в задаче 7 действия добавляется детектирование и распознавание дорожного знака. Для детектирования дорожного знака можно использовать SVM-детектор, нейросетевой детектор, алгоритм детектирования по цветам.

Наиболее точным методом является использование нейросетевого детектора, если использовать cv2.dnn с поддержкой CUDA, скорости обработки кадров хватит для своевременного реагирования на дорожный знак.

Работа с нейросетевым детектором приведена ниже:

```
import numpy as np
import cv2
import time

```

```
model_w_path = "yolo_Hsign_v2/yolov3_cfg_524000.weights"
wh = model_w_path

model_c_path = "yolo_sign_v2/yolov3_cfg.cfg"
cfg = model_c_path

model_n_path = "yolo_sign_v2/classes.txt"
CLASSESPath = model_n_path

net = cv2.dnn.readNetFromDarknet(cfg, wh)
↪  net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

ln = net.getLayerNames()
print(ln)
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
print("-----")
print(ln)
print(net.getUnconnectedOutLayers())

CLASSES = open(CLASSESPath).read().strip().split("\n")
COLORS = np.random.randint(0, 255, size=(len(CLASSES), 3), dtype="uint8")

def detect(image, conf=0.3, thresh=0.3, s=(608, 608)):  # s=(320, 320)
    (H, W) = image.shape[:2]
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, s,  # 416 416
                                 swapRB=True, crop=False)
    net.setInput(blob)

    layerOutputs = net.forward(ln)
    boxes = []
    confidences = []
    classIDs = []
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability) of
            # the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter out weak predictions by ensuring the detected
            # probability is greater than the minimum probability
            if confidence > conf:
                # scale the bounding box coordinates back relative to the
                # size of the image, keeping in mind that YOLO actually
                # returns the center (x, y)-coordinates of the bounding
                # box followed by the boxes' width and height
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top and
                # and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                # update our list of bounding box coordinates, confidences,
                # and class IDs
                boxes.append([x, y, int(width), int(height)])
```

```
64                  confidences.append(float(confidence))
65                  classIDs.append(classID)
66      idxs = cv2.dnn.NMSBoxes(boxes, confidences, conf, thresh)
67      bx = []
68      cids = []
69      confs = []
70
71      if len(idxs) > 0:
72          for i in idxs.flatten():
73              bx.append(boxes[i])
74              cids.append(classIDs[i])
75              confs.append(confidences[i])
76
77      return bx, cids, confs
78
79
80  def draw_boxes(img, boxes, ids, confs, CLASSES, COLORS=None):
81      for i in range(len(boxes)):
82          # extract the bounding box coordinates
83          (x, y) = (boxes[i][0], boxes[i][1])
84          (w, h) = (boxes[i][2], boxes[i][3])
85          # x /= 0.35
86          # y /= 0.35
87          # w /= 0.35
88          # h /= 0.35
89          x = int(x)
90          y = int(y)
91          w = int(w)
92          h = int(h)
93          color = (0, 255, 150)
94          if COLORS is not None:
95              color = [int(c) for c in COLORS[ids[i]]]
96
97          cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
98          text = "{}: {:.4f}".format(CLASSES[ids[i]], confs[i])
99          cv2.putText(img, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
100                     0.5, color, 2)
101     return img
102
103 cam = cv2.VideoCapture(1)
104 ESCAPE = 27
105 key = 1
106 while key != ESCAPE:
107     ret, frame = cam.read()
108     cv2.imshow("Frame", frame)
109     frame = frame[:, frame.shape[1] // 6 * 2:]
110     cv2.imshow("Frame crop", frame)
111
112     boxes, classIDs, confidences = detect(frame)  # s=(self.model_res,
        ↪  self.model_res), conf=conf
113     img_out = draw_boxes(frame, boxes, classIDs, confidences, CLASSES, COLORS=COLORS)
114
115     cv2.imshow("Detected", img_out)
116     key = cv2.waitKey(1)
117
118 cv2.destroyAllWindows()
119 cam.release()
```

Нейросеть должна быть обучена, чтобы детектировать разные виды знаков. В зависимости от того, какой знак будет детектирован на перекрестке, беспилотник

будет принимать решение о направление поворота.

Задача 9

Для детектирования и распознавания знаков в движении следует применять нейросетевой детектор, работа с детектором описана выше — для задачи 8.

Отличие состоит в том, что необходимо вывести число детектированных знаков и порядок встречи с ними.

Для решения задачи необходимо отличать друг от друга знак, детектированный в первый раз, и знак, детектированный не первый раз. Необходимо отличать какой из детектированных на изображении знаков ближний к беспилотнику.

Определить какой из детектированных знаков ближайший к беспилотнику можно по площади его изображения. Чем больше площадь знака, тем он ближе к беспилотнику. Сколько бы знаков не было детектировано на изображении, всегда работаем с ближайшим.

Отличать детектированный в первый раз знак от детектированного не в первый можно по их именам. Знаки на трассе не повторяются. Но детектор, может ошибочно распознавать знаки, поэтому еще более надежным вариантом будет отслеживание координат знаков. И сравнение координат на предыдущем кадре с координатами на текущем. Чем ближе знак к беспилотнику, тем сильнее сместятся его координаты на следующем кадре. Необходимо определить какое смещение координат между кадрами типично для знаков, это делается экспериментально. Если смещение объекта между кадрами больше допустимого, значит произошло ложное срабатывание детектора, которое не стоит учитывать.

Для того, чтобы определять в какой раз детектирован знак, достаточно для каждого из них завести счетчик срабатываний детектора. Если знак детектирован первый раз, добавляем его к списку встреченных знаков.

Число встреченных по дороге знаков, будет длинной полученного ранее списка.

Задача 10

Задача детектирования пешеходов сложнее детектирования дорожных знаков. Она требует меньших усилий по написанию программного кода и больших усилий по обучению нейросетевого детектора.

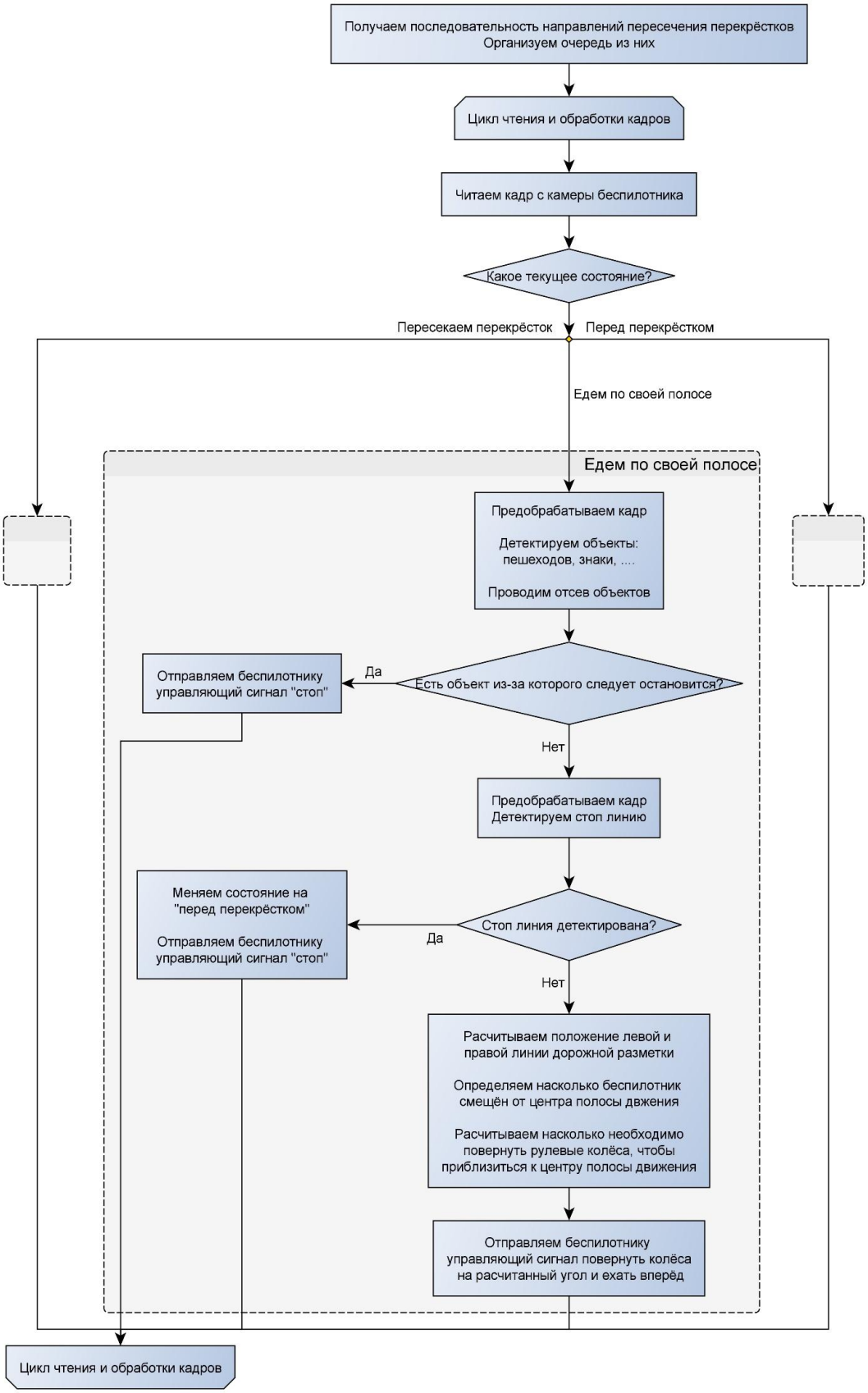
От знаков пешеходы отличаются тем, что их детектор не распознает. И задача отличать детектированных пешеходов друг от друга лежит на программисте. Для решения нужно использовать принципы изложенные в описании задачи 9.

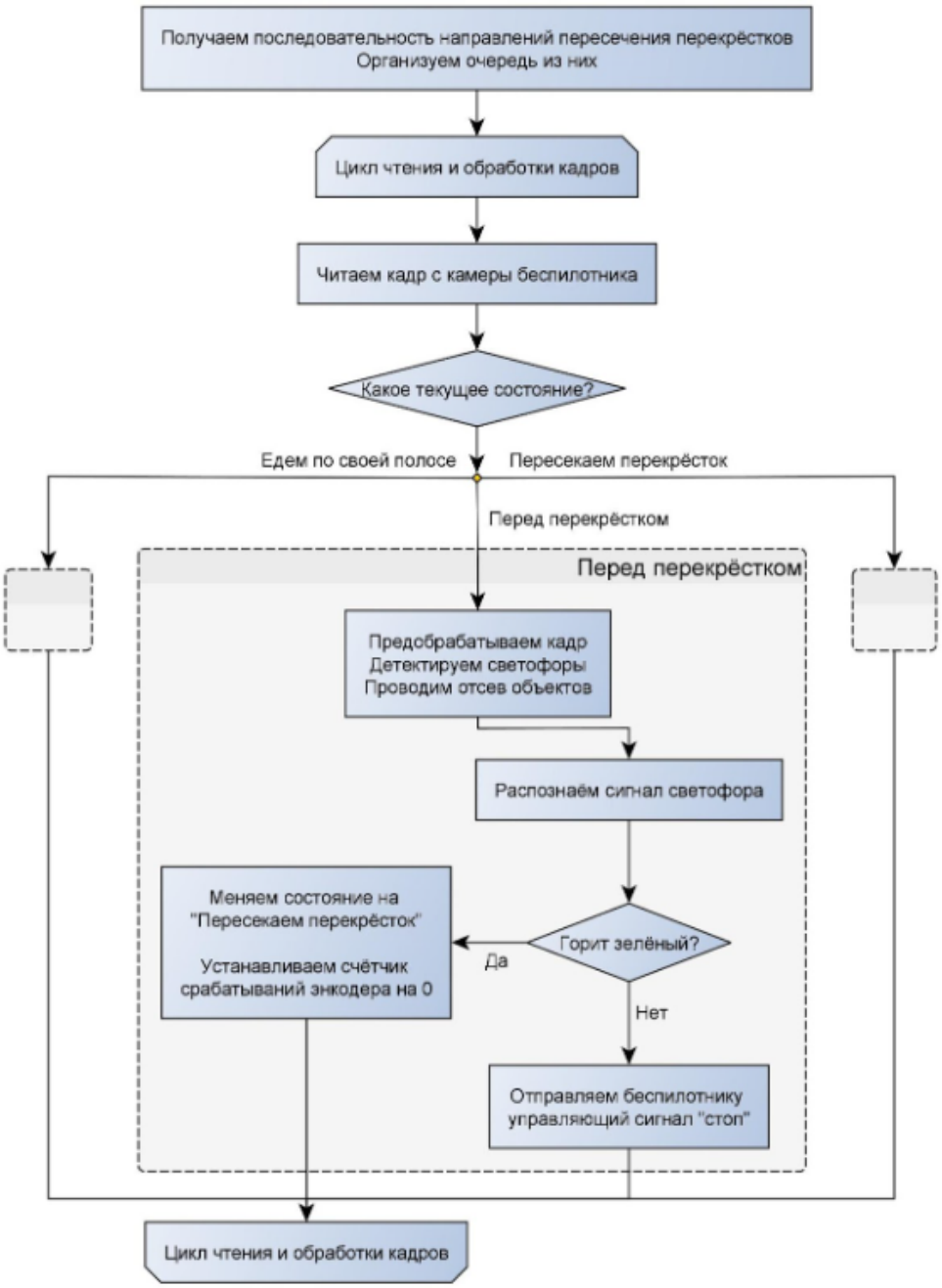
Работа всегда ведется с ближним к беспилотнику пешеходу. По смещению координат детектированных объектов между кадрами, определяется это тот же пешеход что на прошлом кадре, или уже новый. Если новый счетчик увеличивается.

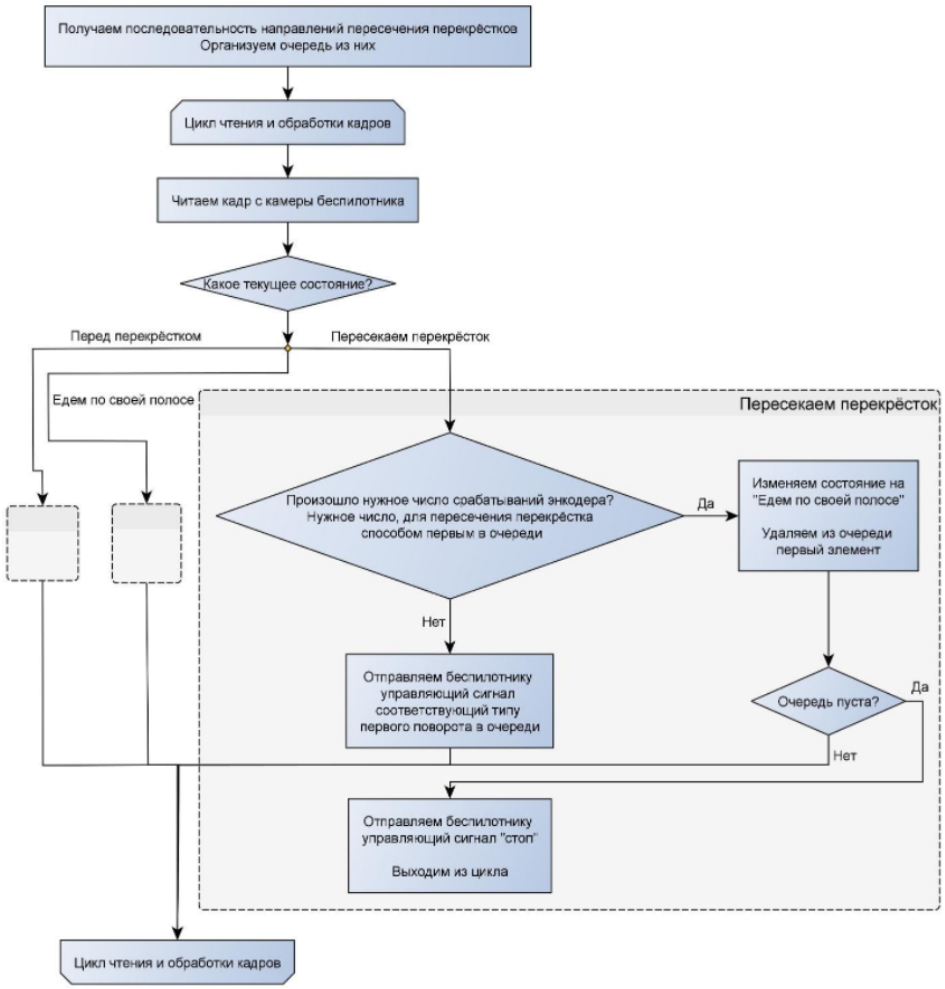
Задача 11

Для успешного выполнения финальной задачи, необходимо объединить описанные выше решения в единый алгоритм.

Ниже, представлена возможная схема алгоритма. После алгоритма разобраны аспекты его технической реализации.




Получаем последовательность направлений пересечения перекрёстков
Организуем очередь из них
Цикл чтения и обработки кадров
Читаем кадр с камеры беспилотника
Какое текущее состояние?
Едем по своей полосе
Пересекаем перекрёсток
Перед перекрёстком
Перед перекрёстком
Предобрабатываем кадр
Детектируем светофоры
Проводим отсев объектов
Распознаём сигнал светофора
Меняем состояние на
"Пересекаем перекрёсток"
Устанавливаем счётчик
срабатываний энкодера на 0
Горит зелёный?
Да
Нет
Отправляем беспилотнику
управляющий сигнал "стоп"
Цикл чтения и обработки кадров



Предобработка кадра

Предобработка кадра, включает в себя различные операции от масштабирования и обрезания изображения, до бинаризации, наложения фильтров, нормализации, применения сверток и получения HOG. Правильная предобработка может существенно сказаться на скорости и точности работы детектора. Например, если вы детектируете знаки, то можете не искать их в нижней части изображения, значит можно обрезать нижнюю половину изображения.

Детектирование объектов: пешеходов, знаков, светофоров

Детектирование объектов можно выполнять различными способами. От примитивного поиска по цветам и сравнением с эталоном, до применения нейронных сетей, таких как Yolo и т. п. Ключевым в выборе детектора должна быть скорость его работы и точность. Детектирование по цветам работает молниеносно, однако при смене освещенности точность будет стремиться к нулю. SVM — детектор, может дать достаточную скорость работы и точность вне зависимости от освещенности. Нейронные сети являются самыми точными и независящими от условий освещенности, однако скорость их работы напрямую зависит от реализации и способа использования, но они бесспорно самый многообещающий вариант. И нейронную сеть и SVM-детектор необходимо обучать на некотором наборе данных, как его составлять зависит от вы-

бранного детектора. Составление обучающего набора данных, не менее важная часть работы, чем применение детектора.

Если детектор работает недостаточно быстро, можно детектировать объекты не на каждом кадре, а через один, через два.

Отсев объектов

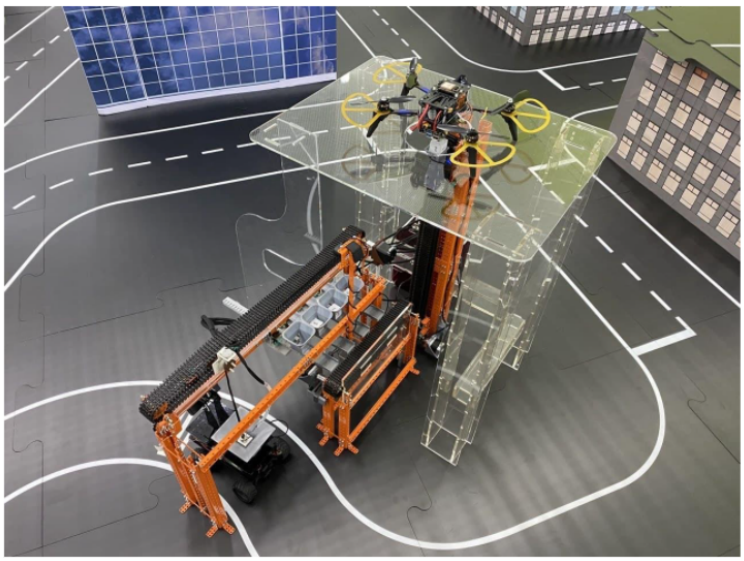
Отсев объектов из всего детектированного набора объектов призван оставить только те, на которые мы хотим реагировать. Предположим, нам необходимо остановиться перед пешеходом, вышедшим на дорогу. Глупо реагировать на любого пешехода в кадре. Нам следует выяснить насколько далеко пешеход находится, это можно сделать, обратив внимание на размер пешехода и координаты его нижнего края. Если пешеход находится близко, то следует понять находится ли он на нашей полосе движения или стоит на обочине. Так и с остальными заданиями, если нам необходимо реагировать на знак «СТОП», то все остальные знаки можно не замечать.

Для решения задачи на базовом уровне, составлен обучающий видеокурс, в котором разобраны решения определения сигналов светофора, распознавания и детектирования знаков, детектирования с помощью SVM-детекторов простых объектов (пешеходы, знаки, светофоры), детектирование дорожной разметки: https://avt.global/cv.

Для решения задач на более высоком техническом уровне, составлен видеокурс по нейронным сетям: https://avt.global/neuralnets.

Этап 2. Сортировка грузов

Основная задача распределительного хаба — хранить и сортировать привозимые беспилотным автомобилем грузы. Сортировка выполняется так, чтобы груз с наименьшим сроком хранения подавался на коптер в первую очередь.



Этап разбит на несколько заданий. Задания можно выполнять в любом порядке.

Средства для выполнения заданий

Участникам предоставлен доступ к компьютеру распределительного хаба. Участники могут запускать на компьютере программы, управляющие узлами хаба. Для этого подготовлены стандартные функции:

- включить магнитный захват;
- выключить магнитный захват;
- поднять магнитный захват на верхний уровень;
- опустить магнитный захват на уровень груза в накопителе;
- переместить каретку к накопителю X (X — номер накопителя, от 1 до 4).

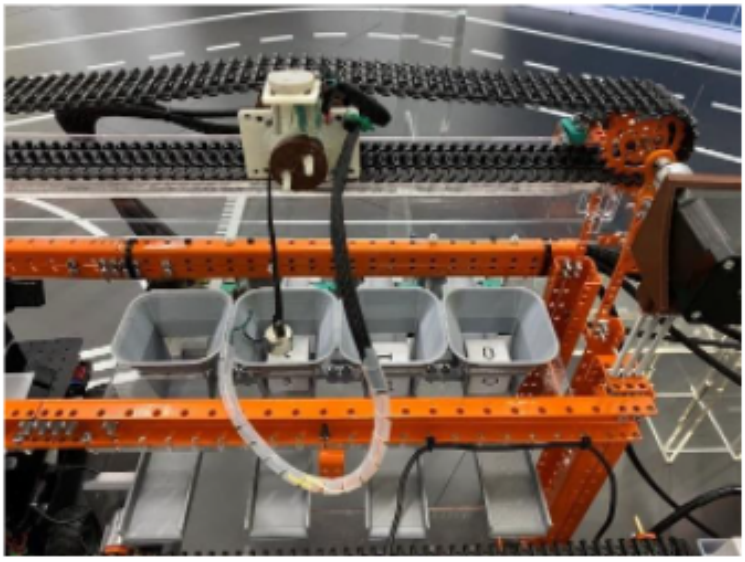


Рис. III.2.2: Каретка над вторым накопителем. Магнитный захват на уровне груза в накопителе

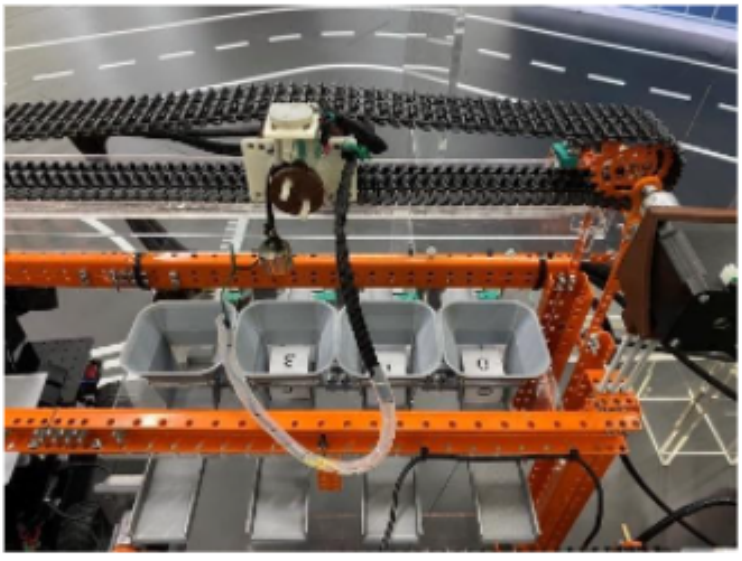


Рис. III.2.3: Каретка над вторым накопителем. Магнитный захват на верхнем уровне

Кроме приведенных выше функций, участники могут написать собственные, используя библиотеки numpy 1.18.5, OpenCV 4.5.1, TensorFlow 2.3. Библиотеки установлены через pip и не оптимизированы под архитектуру Raspberry. Участники могут использовать любой программный подход.

Подготовка к заданию

Ознакомьтесь с перечнем доступных файлов:

- facility_api.py — программа сортировки грузов в накопителях. В этом же файле описаны готовые функции.
- Documentation.docx — документ с подробным описанием функций управления процессом сортировки и примерами использования,
- facility.py — сервисная программа. Ее изменять запрещено!

Ход работы

Участники пишут программу распознавания цифр и сортировки грузов на ПК в файле facility_api.py и загружают в папку команды на Google Drive.

Подключаются по AnyDesk к Raspberry Pi 3 Model B, которая непосредственно управляет лентой конвейера, кареткой и магнитным захватом.

На ленте в случайном порядке расположены 3 кубика с номерами 1, 3 или 4.

Оператор под руководством участника запускает код команды на Raspberry.

При верной работе алгоритма сортировки команда получает 4 балла.

Правила поведения

Участники обязаны следовать инструкциям оператора.

Участникам запрещается изменять файл facility.py.

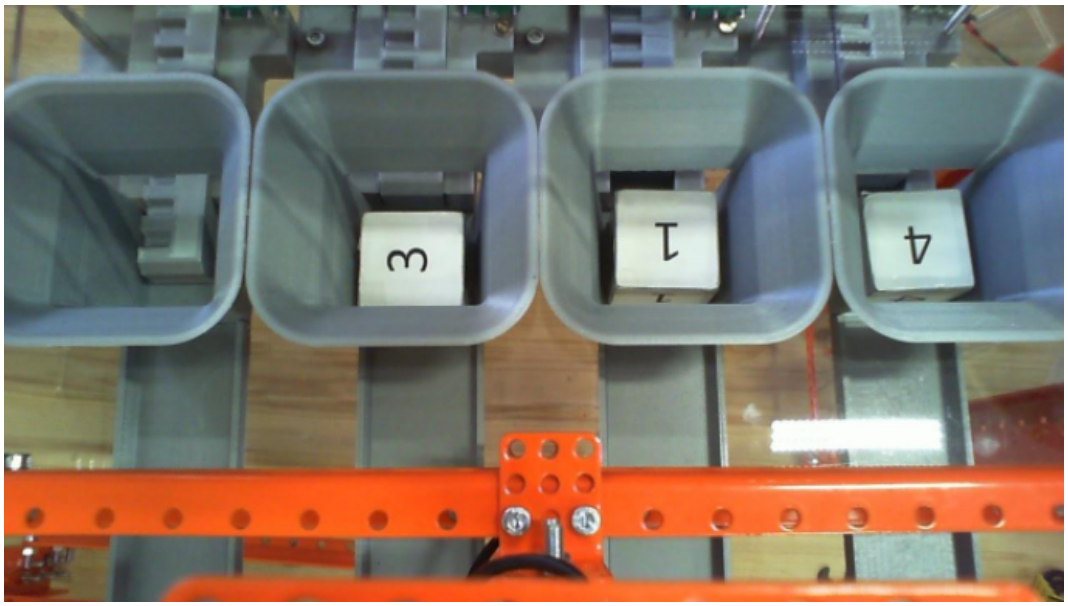
Изображения и видео, предоставленные командам: https://drive.google.com/drive/folders/14siTBYFAHy0iLyo3c3Cw9ywaJrdnp1ME.

Работы участников можно найти по ссылке: https://drive.google.com/drive/folders/15shdSowaF6pKfuK2fWyifLYrREC_TpB9.

В качестве базового кода команды получили архив: https://drive.google.com/drive/folders/1gwE1UmZdDnn1ZaRX40JdAzbepiNrgLq-.

Задача 1. Распознать цифры на грузах в накопителе

На распределительном хабе есть 4 накопителя. 3 правых всегда содержат 1 груз.



Задание считается выполненным, если управляющая программа будет выводить в командную строку последовательность цифр на грузах слева направо.

Задача 2. Расположить грузы в накопителях по возрастанию номеров

Необходимо переместить грузы так, чтобы они лежали по возрастанию номеров слева направо. Пустые накопители, если такие есть, должны остаться с правой стороны.

Описание решений задача 2 этапа

Представленное решение является эталоном выполнения задачи как по распознаванию промаркированных контейнеров и их классификации, так и по подходу к использованию функций, заранее написанных для школьников организаторами. В решении приводится пример использования прикладных методов компьютерного зрения и функций для работы установки. Решение представлено в виде отдельного файла, использующего все необходимые библиотеки и зависимости для автономной работы включая прием груза, сортировку и транспортировку грузов на лифте до квадрокоптера.

Следующий код готовит изображение для дальнейшего детектирования цифр:

```
def get_image_from_rect(rect, src):
    center, size, theta = rext
    center, size = tuple(map(int, center)), tuple(map(int, size))

    M = cv2.getRotationMatrix2D(center, theta, 1)
    dst = cv2.wrapAffine(src, M, (src.shape[1], src.shape[0]))

    out = cv2.getRectSubPix(dst, size, center)
    return out
```

Проверка соотношения сторон полученного контура — проверка: может ли полученный контур на изображение быть кубиком.

```
def check_contour(contour):
    rect = cv2.minAreaRect(contour)

    if rect[1][1] == 0:
        return False

    return 0.5 < rect[1][0] / rect[1][1] < 1.5
```

Следующий код позволяет из уже детектированных контуров, составить так называемые регионы интереса, облечь их в наглядные прямоугольники, которые отображаются на выводящихся на экран кадрах видеопотока, и вырезать изображения грузов из общего изображения.

```
def find_cubes(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 190, 255, cv2.THRESH_BINARY)[1]

    erode = cv2.erode(thresh, np.ones((5, 5), np.uint8), iterations=2)
    contours, _ = cv2.findContours(erode, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    contours = list(filter(check_contour, contours))
    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:3]
    contours = sorted(contours, key=lambda c: cv2.minAreaRect(c)[0][0])

    # Отображает контуры
    image_for_contours = np.copy(image)
    cv2.drawContours(image_for_contours, contours, -1, (255, 0, 0), 3)
    cv2.imshow("Countours", image_for_contours)

    cube_images = []
    for contour in contours:
        rect = cv2.minAreaRect(contour)

        cube_images.append(
            trim_background(get_image_from_rect(rect, image))
        )

    return cube_images
```

Функция ниже, вырезает из полученного ранее изображения груза, цифру.

```
def trim_background(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    th, threshed = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)

    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
    morphed = cv2.morphologyEx(threshed, cv2.MORPH_CLOSE, kernel)

    cnts = cv2.findContours(morphed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
```

Код ниже, представляет финальную функцию распознавания, она определяет на раннее вырезанных изображениях цифр.

```
def clasificate_cube:
    # Определяем кубики через заданные границы
    prediction = []
    print("Использую определение кубиков по заданным границам...")
```

```
5
6       for i in range(len(hoardersPosition)):
7           position = hoardersPosition[i]
8
9           crop_image = image[position[0]:position[2], position[1]:position[3]]
10          crop_image = trim_background(crop_image)
11
12          prediction.append(
13              [1, 3, 4][classifier.classify(crop_image)[0] - 1]
14          )
15
16      if len(prediction) == len(set(prediction)):
17          return prediction
18
19      # Автоматическое определение границ
20      print("Использую автоматическое определение кубиков...")
21      cube_images = find_cubes(image)
22
23      prediction = []
24      for image_num in range(len(cube_images)):
25          image = cube_images[image_num]
26
27          cv2.imshow(f'Hoarder #{image_num + 1}', image)
28
29          prediction.append(
30              [1, 3, 4][classifier.classify(crop_image)[0] - 1]
31          )
32
33      return prediction
```

Код ниже — функция перемещения к n накопителю, созданная для удобства из базовых функций.

```
1   def moveToHoarder(hoarder: int):
2       print(f'Двигаюсь к накопителю {hoarder}')
3
4       if hoarder == 0:
5           ToFirstHoarder()
6       elif hoarder == 1:
7           ToSecondHoarder
8       elif hoarder == 2:
9           ToThirdHoarder
10      else: # hoarder == 3
11          ToFirstHoarder()
```

Данная программа является финальным этапом той части программы, которая связана с сортировкой грузов в порядке убывания их номеров (цифр). Сортировка производится последовательным перемещением кубиков по ячейкам, в зависимости от начального расположения.

```
1   MagnetUp()
2   CarriageEnd()
3   CarriageStert()
4
5   for currentHoarder in range(3):
6       cube = final_hoarder_state[currentHoarder]
7
8       if hoarder_state[currentHoarder] == cube:
9           continue
```

```
10
11      currentCubePosition = hoarder_state.index(cube)
12
13      if hoarder_state[currentHoarder] is None:
14          moveToHoarder(currentCubePosition)
15
16          MagnetHalfDown()
17          MagnetOn()
18          MagnetUp()
19
20          moveToHoarder(currentHoarder)
21
22          MagnetHalfDown()
23          MagnetOff()
24          MagnetUp()
25
26          hoarder_state[currentHoarder] = cube
27          hoarder_state[currentCubePosition] = None
28      else:
29          moveToHoarder(currentHoarder)
30
31          MagnetHalfDown()
32          MagnetOn()
33          MagnetUp()
34
35          emptyHoarder = hoarder_state.index(None)
36          moveToHoarder(emptyHoarder)
37
38          MagnetHalfDown()
39          MagnetOff()
40          MagnetUp()
41
42          moveToHoarder(currentCubePosition)
43
44          MagnetHalfDown()
45          MagnetOn()
46          MagnetUp()
47
48          hoarder_state[emptyHoarder] = hoarder_state[currentHoarder]
49          hoarder_state[currentHoarder] = cube
50          hoarder_state[currentCubePosition] = None
51
52      print("Расположение кубиков:", *[str(i) + " - " + str(hoarder_state[i]) for i in
        ↪  range(4)], sep='\n')
53
54      image = read_image()
55
56  print("Готово")
57
58  end()
```

Этап 3. Доставка груза при помощи квадрокоптера

Задача квадрокоптера — в режиме автономного полета произвести доставку груза на указанное здание и затем проследовать на дозаправку. На крыше каждого здания в макете города находится графическая метка. Коптеру необходимо:

- получить координаты адреса назначения и точки дозаправки и подхватить груз

при помощи магнитного захвата;

- прилететь в точку назначения и удостовериться в соответствии метки зданию;
- в случае успешной верификации стабилизировать положение груза относительно центра метки и произвести сброс груза;
- долететь до лняет полет к крыше второго здания, центрирование по графической метке и посадку.

Задача 1. Выполнить захват груза и перелет с грузом на первое здание

Груз доставляется лифтом распределительного хаба непосредственно под квадрокоптер. Задача участников:

- включить магнитный захват,
- захватить груз,
- взлететь с крыши хаба,
- включить световую индикацию (цвет — зеленый),
- выполнить зависание на заданное время (10 секунд),
- долететь до первой точки назначения.

Задача 2. Верифицировать графическую метку

После подлета к крыше указанного здания (2 или 4) квадрокоптер должен:

- распознать графическую метку при помощи нейронной сети (определить наличие метки и ее принадлежность к одному из классов изображений);
- после распознавания включить световую индикацию «метка определена» (переливающиеся цвета),
- выполнить зависание на заданное время (10 секунд),
- передать оператору название графической метки. Название метки передается и учитывается лишь один раз.

Возможные классы графических меток:

Задача 3. Произвести сброс груза в центр графической метки

Квадрокоптеру необходимо:

- рассчитать местоположение центра графической метки,
- произвести центрирование груза над центром метки,
- сбросить груз (в зачет идет точность попадания груза в центр метки)
- включить световую индикацию (цвет — красный)
- выполнить зависание на указанное время (10 секунд).

Задача 4. Выполнить перелет на крышу второго здания (2 или 4) и произвести посадку в центр крыши

На крыше второго здания расположена посадочная площадка. После подлета к крыше второго здания квадрокоптер должен:

- распознать графическую метку при помощи нейронной сети (определить наличие метки и ее принадлежность к одному из классов изображений);
- после распознавания включить световую индикацию «метка определена» (переливающиеся цвета),
- выполнить зависание на заданное время (15 секунд),
- передать оператору название графической метки. Название метки передается и учитывается лишь один раз. Классы графических меток те же, что в Задаче 2.
- Произвести посадку квадрокоптера в центр графической метки. В зачет идет точность посадки (центр квадрокоптера должен находиться над центром метки. Центром квадрокоптера считается центр магнита).

Доступные функции для программирования квадрокоптера:

Приведены по ссылке https://github.com/IlyaDanilenko/onti2020.

Правила выполнения заданий

1. Для разработки и отладки программного кода команды подключаются к стендам дистанционно. Каждой команде предоставляется один стенд. Смена стенда на протяжении соревнования не допускается. Работа со стендами может производиться на протяжении всего рабочего дня финала: с 9:30 до 17:00 МСК.
2. Всю работу по программированию, запуску алгоритмов распознавания и полета участники производят самостоятельно. Оператор устанавливает коптер на стартовую позицию, меняет аккумуляторные батареи и графические метки для тестирования функций распознавания (только по запросу участников).
3. Запуск программы на квадрокоптере команда производит строго после подтверждения оператором! При нарушении данного условия команда дисквалифицируется за нарушение техники безопасности при эксплуатации беспилотного летательного аппарата.
4. За сохранность собственного программного кода (например, путем создания бэкапов на локальном компьютере) несут ответственность участники.

Рекомендуемый порядок подготовки

1. Ознакомиться со стендом, провести проверку стенда запуском тестовых скриптов.
2. Разработать и отладить алгоритм распознавания графических меток и передачи сообщения с текстом распознанной графической метки.
3. Разработать и отладить алгоритм расчета центра метки и центрирования над меткой (требуются тестовые полеты).
4. Разработать программный код выполнения задачи доставки груза.
5. Провести тестовые полеты на полигоне и финальную отладку алгоритмов (последний день соревнования).

Состав оборудования и ПО на площадке:

1. Стенд (одноплатный компьютер Raspberry Pi 4 с камерой, полетный контроллер Геоскан «Пионер», модуль захвата груза). Стенд предназначен для отладки алгоритмов, которые не требуют полета (захват и отпускание груза, световая индикация, детектирование метки, определение центра метки).
2. Квадрокоптер Геоскан «Пионер Макс» с системой оптического позиционирования и модулем захвата груза.
3. Среда программирования VS Code (установлена на борту квадрокоптера), TensorFlow Lite (для запуска обученного классификатора на борту квадрокоптера).

Датасеты предоставленные командам: https://github.com/IlyaDanilenko/onti2020/releases/tag/onti2020.

Библиотека для работы с квадрокоптером: https://github.com/IlyaDanilenko/onti2020/blob/main/pioneer_max_onti.py.

Система оценки

9, 10 и 11 марта команды подключаются к оператору в соответствии с расписанием подключений. Во время подключения участники говорят, какую задачу они будут сейчас сдавать. Оператор ставит оборудование в стартовое положение, проводит визуальный контроль за выполнением всех условий задачи и по итогам работы оборудования проставляет баллы в таблицу. Оба Айкара равноценны.

В случае, если команда сделала на оборудовании все, что хотела, а минуты на подключение еще остались, их можно перенести на досдачи в конце блока подключений. Между блоками подключений минуты на досдачи не переносятся!

В случае потери минут ввиду непредвиденных технических сложностей на стороне организаторов, потерянные участниками минуты компенсируются. Проблемы с интернетом на стороне участника не являются причиной для компенсации минут.

Число попыток в одной задаче не ограничено.

12 марта команды подключаются ко всем операторам одновременно и проводят финальный тест всей системы.

Баллы за задачи, которые будут успешно реализованы в ходе финального теста, удваиваются!

Этап 1. Проезд беспилотного автомобиля по полигону городской среды

№	Задача	Балл
1	Доехать до перекрестка и остановиться	1
2	Повернуть на перекрестке направо	2
3	Повернуть на перекрестке налево	2
4	Пересечь перекресток по прямой	2
5	Остановиться в точке разгрузки	2
6	Доставить груз к распределительному хабу	8
7	Пересечь перекресток с учетом сигналов светофора	5
8	Пересечь перекресток с учетом сигналов светофора и дорожного знака	7
9	Распознавание дорожных знаков	8
10	Детектирование пешеходов	8
11	Доставка груза с учетом всех объектов городской среды	10

Этап 2. Сортировка грузов

№	Задача	Балл
1	Распознать цифры на грузах в накопителе	5
2	Расположить грузы в накопителях по возрастанию номеров	10

Этап 3. Доставка груза при помощи квадрокоптера

№	Оцениваемый параметр	Балл
1	Индикация при захвате груза включена. Цвет верный	1
2	Зависание с грузом на заданное время (10 секунд) выполнено	2
3	Класс графической метки №1 определен верно	4
4	Индикация «метка определена» включена. Цвет верный	1
5	Зависание на заданное время (10 секунд) выполнено	2
6	Индикация над местом сброса груза включена. Цвет верный	1
7	Зависание над местом сброса груза на заданное время (10 секунд) выполнено	2
8	Сброс груза выполнен	5 — груз находится в радиусе 5 см от центра метки 3 — груз находится на расстоянии от 5 до 10 см от центра метки 1 — груз находится на расстоянии более 10 см от центра метки 0 — груз упал с крыши

№	Оцениваемый параметр	Балл
9	Класс графической метки №2 определен верно	4
10	Индикация «метка определена» включена. Цвет верный	1
11	Зависание на заданное время (15 секунд) выполнено	2
12	Произведена посадка на посадочную площадку	5 — центр квадрокоптера находится в радиусе 5 см от центра метки 3 — центр квадрокоптера находится на расстоянии от 5 см до 10 см от центра метки 1 — центр квадрокоптера находится на расстоянии более 10 см от центра метки

Максимальное число баллов за все задачи — 200.

Команда, набравшая наибольшее количество баллов по сумме задач становится победителем профиля «Автономные транспортные системы» Олимпиады Кружкового движения НТИ.