Второй отборочный этап

Представленные задачи являются результатом декомпозиции задачи заключительного этапа. Они направлены на формирование знаний и компетенций, требуемых на заключительном этапе. Участники создают систему, в которой квадрокоптер патрулирует улицы города и детектирует разнообразные происшествия, например пожар или несчастный случай на улице, о которых в информационный центр отсылаются сообщения. На основе этих данных центр формирует список заданий для беспилотных автомобилей. Перед тем как выполнять задание беспилотный автомобиль должен загрузить на борт тип оборудования, соответствующий заданию и связанному с ним происшествию. Экипировкой оборудования занимается сервисный центр.

Решение представленных задач требует командного подхода: распределения ролей, определения участков работы и разграничения ответственности. Освоение компетенций, необходимых для решения задач одним человеком, невозможно в течении второго этапа. Для решения всех пяти задач требуются слаженное взаимодействие всех участников команды, имеющих следующие компетенциями: обучение и применение нейросетевых детекторов, знание ROS, создание полетных программ для квадрокоптера Пионер, чтение технической документации, поиск объектов по цветам, построение алгоритмов и их реализация в программном коде, теория графов, реализация обмена данными через сокеты.

Представленные ниже задачи связаны с операциями, которые будут выполнять устройства автономной транспортной системы на заключительном этапе.

Задача IV.1. Обнаружение транспортных средств (35 баллов)

Темы: нейросетевые детекторы, обнаружение объектов в видеопотоке, отслеживание объектов в видеопотоке.

Условие

На видео запечатлено движение автомобилей по кольцевой развязке. Ваша задача: написать функцию, подсчитывающую общее число автомобилей на видео. Необходимо учитывать и неподвижные автомобили, и движущиеся. Не различайте транспортные средства по типу.

Рис. 1. Кольцевая развязка

Выполнение

1. Скачайте материалы задания: https://disk.yandex.ru/d/DebI1aRQyS5QXw.
2. Ознакомьтесь с материалами задания.
   Для вас подготовлены несколько файлов `.py`, набор видеофайлов и аннотации к нему.
   В списке подготовленных файлов содержатся:
   - `eval.py` — файл с функцией обнаружения и подсчета транспортных средств; именно эту функцию вам необходимо дописать;
   - `main.py` — файл, проверяющий точность работы вашего алгоритма; не редактируйте его. `main.py` использует, написанные вами функции из `eval.py` и сверяет истинные метки с предсказанием вашего алгоритма;
   - `annotations.csv` — файл, устанавливающий соотношение между видеофайлами и количеством; в каждой строке файла содержится путь к видеофайлу и числу транспортных средств на нем.

   В качестве решения необходимо сдать отредактированный файл `eval.py`. Либо архив `*.zip` с файлом `eval.py` и остальными файлами, требующимися для его работы.
3. Прочитайте файл `eval.py`.
   В файле содержится функция, которую вам необходимо дописать.
4. Допишите в файле `eval.py` функцию `load_models`.
   Эта функция отвечает за загрузку из файлов всех моделей машинного обучения, которые вы собираетесь использовать.
5. Допишите в файле `eval.py` функцию `count_vehicles`.
   Эта функция получает на вход объект `videoCapture`, связанный с видеофайлом. Усовершенствуйте функцию так, чтобы она возвращала число транспортных средств запечатленных на видео.
6. Запустите файл `main.py` и проверьте свой алгоритм.
   Если программа выдала ошибку, то найдите и исправьте ее в файле `eval`, вновь

запустите файл `main.py`.

7. Пришлите решение на онлайн платформу для проверки.
   Упакуйте файл `eval.py`. и дополнительные файлы в архив `*.zip`. В архиве должны находиться файлы, а не одноименная архиву папка.

Технические ограничения

Размер решения ограничен: не более 30 МБ. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решение можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки:

`Python 3.8.10; catboost 1.1.1; dlib 19.24.0; gast 0.4.0; h5py 3.7.0; imutils 0.5.4; keras 2.9.0; Keras-Preprocessing 1.1.2; matplotlib 3.6.2; numpy 1.23.2; opencv-python 4.6.0.66; pandas 1.5.1; scikit-image 0.19.3; scikit-learn 1.1.3; scipy 1.9.3; tensorflow-cpu 2.9.2; torch 1.13.0; torchaudio 0.13.0; torchvision 0.14.0.`

Используйте совместимые пакеты.

Материалы задачи, предоставленные участникам

https://disk.yandex.ru/d/DebI1aRQyS5QXw.

Формат входных данных

Текстовая строка.

Пример входных данных

```
medic_aid.330.114,fire.78.64,crash.31.57;122;198;gardening_tools;1;
bransboit.
```

Формат выходных данных

Последовательность текстовых сообщений отправленных через сокет.

Файл решения

https://disk.yandex.ru/d/88m9N7uv5bDIhg.

Тесты

https://disk.yandex.ru/d/NjtSsi-hicEFOw.

Решение

Для решения задания необходимо обучить нейросетевой детектор. Данных, предоставленных участникам, достаточно для разметки собственного датасета.

После того как модель обучена, необходимо детектировать автомобили на каждом кадре видеофайла.

Положение автомобилей на текущем кадре необходимо сравнить с положением на предыдущем кадре. Если в окрестности текущего положения на прошлом кадре не было автомобилей, то считаем автомобиль впервые обнаруженным и увеличиваем счетчик автомобилей на видео. Размер окрестности определяется экспериментально.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
from ultralytics import YOLO
import math

# Функция загрузки нейросетевой модели из файла
def load_models():
    models = [YOLO('car_detector.pt')]
    return models

# Основная функция,вызывается для каждого видеофайла отдельно.
def count_vehicles(video, models) -> int:
    car_count = 0  # Счетчик автомобилей на видео.
    old_box_list = []  # Список автомобилей на прошлом кадре.
    while True:
        # Читаем кадры,пока они не закончатся.
        ret, img = video.read()
        if not ret:
            break

        # Передаем детектору кадр и получаем результаты.
        results = models[0].predict(img, verbose=False)

        new_box_list = []  # Список автомобилей на текущем кадре.

        # Перебираем ограничивающие рамки для объектов,
                                        # детектированных с уверенностью > 0.5
        # Сравниваем каждый обнаруженный автомобиль с автомобилями
                                                        # на предыдущем кадре.
        # Если новый автомобиль не совпадает по координатам ни с одним из
                                                        # "прошлых" автомобилей,
        # то увеличиваем счетчик автомобилей на видео.
        for box in results[0].boxes:
            if box.conf > 0.5:
                # Точка (x,y) - верхний левый угол рамки объекта.
                x = int(box.xyxy[0, 0])
                y = int(box.xyxy[0, 1])
                # Добавляем в список объектов на кадре.
                new_box_list.append((x, y))
                # Проверяем есть-ли объекты на предыдущем кадре.
                if old_box_list != []:
                    # Перебираем объекты на предыдущем кадре.
                    # Если находим объект,в +- тех же координатах,то
```

```
                    # считаем автомобиль ранее детектированным,
                    # т.е  не увеличиваем счетчик
                    for old_box in old_box_list:
                        if math.sqrt((old_box[0]-x)**2 +
                                     (old_box[1]-y)**2) <= 120:
                            # Удаляем объект из старого списка
                            old_box_list.remove(old_box)
                            break
                    else:
                    # Операторы в else выполнятся,если цикл прошел полностью,
                    # т.е. выхода через break не произошло.

                        # Если не нашлось ни одного подходящего объекта,то
                        # считаем авт. впервые появившимся - увеличиваем счетчик.
                        car_count += 1
                else:  # Если на предыдущем кадре нет объектов,то
                    # считаем авт. найденным впервые - увеличиваем счетчик.
                    car_count += 1

        # Обновляем список автомобилей на предыдущем кадре.
        old_box_list = new_box_list

    video.release()
    return car_count
```

Задача IV.2. Обмен данными (15 баллов)

Темы: коммутация программ через сокеты, протоколы передачи данных.

Условие

Необходимо написать функцию, реализующую обмен сообщениями с двумя устройствами: сервисным центром и информационным центром. Сервисный центр формирует список заданий, которые необходимо выполнить. Задание характеризуется двумя параметрами: координатами и типом задания. Чтобы приступить к его выполнению, необходимо зарезервировать его за собой и соответствующим образом экипироваться, сообщить информационному центру о получении необходимой экипировки. Типов экипировки больше чем типов заданий, все возможные варианты представлены ниже.

Тип экипировки	Тип задания
bransboit	fire
medical_kit	med_aid
repair_tools	crush
cleaner_kit	—
gardening_tools	—
comfortable_salon	—

Функция, которую вам необходимо написать, получает на вход ваши текущие координаты и тип экипировки, которой вы располагаете. Функция должна запросить список заданий, выбрать ближайшее задание и зарезервировать его за собой. В общении с сервисным центром получить необходимую экипировку и сообщить об этом. После этого, функция должна вернуть 0.

Прием и передача данных происходят через TCP-сокет. Передаваемые сообщения текстовые, протокол общения представлен в материалах к заданию. Обмен сообщениями ведется по принципу запрос-ответ. Запрос всегда отправляет вы.

Выполнение

1. Скачайте материалы задания: https://disk.yandex.ru/d/SrAP_m99W6LaUQ.
2. Ознакомьтесь с материалами задания.
   Для вас подготовлены несколько файлов `.py`, набор входных данных и верных ответов к ним и описание протокола общения.
   В список подготовленных файлов входят:
   - `eval.py` — файл с функцией создания сокета и функцией обмена сообщениями; именно функцию обмена сообщениями вам необходимо дописать;
   - `main.py` — файл, проверяющий точность работы вашего алгоритма; не редактируйте его. `main.py` использует, написанные вами функции из `eval.py` и сверяет результат работы вашего алгоритма с правильным ответом;
   - `annotations.csv` — файл с набором входных данных и верными ответами к ним; в каждой строке файла содержатся: стартовые координаты, тип стартовой экипировки, список заданий в информационном центре, задание, которое вам необходимо зарезервировать и тип экипировки, которую необходимо получить;
   - `protocol.txt` — описание сообщений, которые вы можете отправлять и сообщений, которые будут отправлять вам.

   В качестве решения необходимо сдать отредактированный файл `eval.py`.
3. Прочитайте файл `eval.py`.
   В файле содержится функции `setup_socket` и `communication_cycle`.
4. Допишите в файле `eval.py` функцию `communication_cycle`.
   Эта функция отвечает за обмен данными с информационным и сервисным центрами.
5. Запустите файл `main.py` и проверьте свой алгоритм.
   Если программа выдала ошибку, то найдите и исправьте ее в файле `eval`, вновь запустите файл `main.py`.
6. Пришлите решение на онлайн-платформу для проверки
   В качестве решения необходимо сдать отредактированный файл `eval.py` или `*.zip` архив с ним. В архиве должны находиться файлы, а не одноименная архиву папка.

Технические ограничения

Размер решения ограничен: не более 30 МБ. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решение можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки:

`Python 3.8.10; catboost 1.1.1; dlib 19.24.0; gast 0.4.0; h5py 3.7.0; imutils 0.5.4; keras 2.9.0; Keras-Preprocessing 1.1.2; matplotlib 3.6.2;`

`numpy 1.23.2; opencv-python 4.6.0.66; pandas 1.5.1; scikit-image 0.19.3; scikit-learn 1.1.3; scipy 1.9.3; tensorflow-cpu 2.9.2; torch 1.13.0; torchaudio 0.13.0; torchvision 0.14.0.`

Используйте совместимые пакеты.

Материалы задачи, предоставленные участникам

https://disk.yandex.ru/d/SrAP_m99W6LaUQ.

Формат входных данных

Текстовая строка.

Пример входных данных

`medic_aid.330.114,fire.78.64,crash.31.57;122;198;gardening_tools;1; bransboit.`

Формат выходных данных

Последовательность текстовых сообщений отправленных через сокет.

Файл решения

https://disk.yandex.ru/d/IoYsjcBV9yMsRA.

Тесты

https://disk.yandex.ru/d/sIa2YlhlgiQziQ.

Решение

1. Инициализировать сокет для получения и отправки сообщений (реализовано в отдельной функции).
2. Получить список заданий от сервера и извлечь из него данные — названия заданий и их координаты.
3. Найти ближайшее задание и зарезервировать его за собой.
4. Запросить экипировку под задание, если мы не располагаем ей.
5. Сообщить, что экипированы для задания и оборвать соединение.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
1  import socket
2  import math
3
```

```
4  def setup_socket(ip_address, port):
5      """ Функция инициализирует сокет.
6          Входные параметры: ip-адрес и порт сервера
7          Выходные параметры: инициализированный сокет
8
9      """
10     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     server.connect((ip_address, port))
12     return server
13
14 def communication_cycle(conn: socket.socket, start_pos, start_equipment):
15     """ Эта функция отвечает за обмен данными с информационным и сервисным
16                                                                 центрами..
17         Входные параметры:
18             conn - сокет из функции setup_socket,
19             start_pos - стартовая позиция беспилотного автомобиля (x,y),
20             start_equipment - начальное снаряжение
21     """
22     conn.send('server,give_tasks|'.encode())  # запрашиваем у сервера список задач
23     msg = ''  # полученные от сервера данные,будем записывать в переменную msg
24     while True:
25         symbol = conn.recv(1).decode()  # принимаем ответ сервера по одному
26                                                                     символу
27         if symbol in ('|', ''):
28             break  # если мы получили символ конца сообщения "|" или
29                    # читать нечего,то выходим из цикла
30         msg += symbol
31
32     # Далее необходимо выбрать ближайшее к нам задание
33     # Для этого преобразуем полученную от сервера строку с заданиями и их
34                                                 координатами в два списка.
35     # В первом списке будут названия заданий,во втором дистанция до задания.
36     tasks_name_list = []  # Список для имен заданий.
37     distance_list = []  # Список для дистанции до заданий.
38
39     task_and_xy_list = msg.split(",")  # Делим список заданий на части,
40                                               разделителем считаем запятую.
41     for text in task_and_xy_list:  # Перебираем получившиеся части.
42         task_x_y = text.split(".")  # Каждую часть делим на подчасти,
43                                                 разделителем считаем точку.
44         # Подчасть с индексом 0 - имя задания.
45         # Подчасть с индексом 1 - X координата задания.
46         # Подчасть с индексом 2 - Y координата.
47
48         tasks_name_list.append(task_x_y[0])  # Добавляем имя задания в список.
49         task_x = int(task_x_y[1])  # Получаем X ко-ту задания в виде целого числа.
50         task_y = int(task_x_y[2])  # Получаем Y ко-ту задания в виде целого числа.
51         # Рассчитываем дистанцию до задания (евклидово расстояние)
52                                                       по теореме пифагора.
53         distance = math.sqrt((start_pos[0] - task_x) ** 2 +
54                                                 (start_pos[1] - task_y) ** 2)
55         distance_list.append(distance)  # Добавляем дистанцию в список.
56
57     """У нас есть два списка.
58        В первом названия задач - tasks_name_list.
59        Во втором дистанция до задач - distance_list.
60        Если найти индекс минимального элемента во втором массиве,
61        то получим и индекс названия ближайшей задачи в первом массиве.
62     """
63
```

```
64      # Находим индекс минимального элемента в списке дистанций.
65      min_dist = distance_list[0]
66      for i, dist in enumerate(distance_list):
67          if min_dist >= dist:
68              min_dist = dist
69              indx_nearest_point = i
70
71      # indx_nearest_point - индекс ближайшей задачи в списке tasks_name_list и
72                                              в списке, заданий полученном с сервера.
73      # tasks_name_list[indx_nearest_point] - название ближайшей задачи.
74      nearest_point_name = tasks_name_list[indx_nearest_point]
75
76      # Отправляем серверу индекс задачи,которую резервируем за собой.
77      for_server_message = "server,reserve_task," + str(indx_nearest_point) + "|"
78      conn.send(for_server_message.encode())
79      while True:  # Считываем ответ сервера.
80          symbol = conn.recv(1).decode()
81          if symbol in ('|', ''): break
82          msg += symbol
83
84      # Выясняем какая экипировка нужна для выполнения нашего задания.
85      if nearest_point_name == "fire":
86          required_equipment = "bransboit"
87      elif nearest_point_name == "medic_aid":
88          required_equipment = "medical_kit"
89      elif nearest_point_name == "crash":
90          required_equipment = "repair_tools"
91
92      """Теперь необходимо необходимо получить нужную экипировку,если мы ей еще
93                                                                  не располагаем.
94         Если у нас уже имеется нужная экипировка,то ничего делать не нужно.
95         Если у нас есть экипировка,но она не подходящая под задание,
96                                                  то нужно ее сдать и запросить новую.
97         Если у нас вообще нет экипировки,то нужно запросить новую.
98      """
99
100     if start_equipment != required_equipment:  # Если наша экипировка отличается
101                                                          # от требуемой...
102         if start_equipment != "no_equipment":  # Проверяем,что у нас есть
103                                                          # какая-то экипировка.
104             conn.send('hub,offload|'.encode()) # Если есть,сдаем ее.
105             msg = ''
106             while True:
107                 symbol = conn.recv(1).decode()
108                 if symbol in ('|', ''):
109                     break
110                 msg += symbol
111
112         # Запрашиваем новую экипировку,соответствующую взятому заданию.
113         equipment_code = {"bransboit": "1",
114                           "medical_kit": "2",
115                           "repair_tools": "3"}
116         conn.send(f'hub,give,{equipment_code[required_equipment]}|'.encode())
117         msg = ''
118         while True:
119             symbol = conn.recv(1).decode()
120             if symbol in ('|', ''):
121                 break
122             msg += symbol
123
```

```
    # Сообщаем,что мы получили необходимую экипировку и готовы выполнять задание
    conn.send('equip_ready|'.encode())
    msg = ''
    while True:
        symbol = conn.recv(1).decode()
        if symbol in ('|', ''):
            break
        msg += symbol
    # Разрываем соединение
    conn.close()
```

Задача IV.3. Отслеживание маршрута беспилотного автомобиля (15 баллов)

Темы: построение и хранение графов, обнаружение объектов в видеопотоке, отслеживание объектов в видеопотоке.

Условие

На видео представлено перемещение беспилотного автомобиля по дорожной сети с несколькими перекрестками. Каждая дорога двунаправленная и имеет две полосы движения, каждая в свою сторону. Движение правостороннее. Городскую дорожную сеть можно представить в виде графа, где перекрестки — вершины, а полосы движения — ребра графа. Перекрестки и соответствующие им вершины графов пронумерованы слева направо.

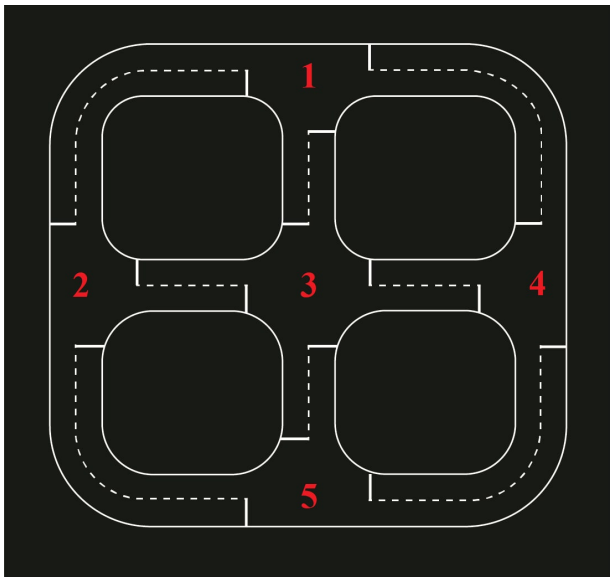

Рис. 2. Городская дорожная сеть

Ваша задача написать функцию, отслеживающую перемещения беспилотника и заполняющую матрицу. Матрица, которую нужно заполнить — аналог матрицы смежности графа. В нашем случае значения записанные в матрице обозначают число раз, которое автомобиль проехал по ребру графа.

Элемент матрицы с индексами $[2, 0]$ соответствует вертикально направленной полосе движения от третьего перекрестка к первому. Первый индекс — стартовая вершина, второй индекс — конечная вершина.

Помните, что в матрице нумерация строк и столбцов начинается с 0, а не с 1. Если значение элемента в матрице 0, значит полосы движения, соответствующей такому ребру графа не существует, либо беспилотник еще не проезжал по этой полосе. Изначально матрица заполнена нулями.

От видео к видео схема дорожной разметки не изменяется, изменяется маршрут следования беспилотника.

Выполнение

1. Скачайте материалы задания: https://disk.yandex.ru/d/mBYdqmLvAlY-vw.
2. Ознакомьтесь с материалами задания.
   Для вас подготовлены несколько файлов `.py`, набор видеофайлов и аннотации к нему. Среди подготовленных файлов есть:
   - `eval.py` — файл с функцией, отслеживающей перемещения беспилотного автомобиля; именно эту функцию вам необходимо дописать;
   - `main.py` — файл проверяющий точность работы вашего алгоритма; не редактируйте его. `main.py` использует, написанные вами, функции из `eval.py` и сверяет верные ответы с предсказанием вашего алгоритма;
   - `annotations.csv` — файл, устанавливающий соответствие между видеофайлами и матрицами, которые должны получиться; в каждой строке файла содержится путь к видеофайлу и соответствующая ему матрица.

   В качестве решения необходимо сдать отредактированный файл `eval.py`, либо архив `*.zip` с файлом `eval.py` и остальными файлами, требующимися для его работы.
3. Прочитайте файл `eval.py`.
   В файле содержатся функции `track_movement` и `load_tools`.
4. Допишите в файле `eval.py` функцию `load_tools`.
   Функция используется для загрузки моделей машинного обучения из файлов или загрузки эталонных изображений. Если вы не используете ее, пусть возвращает пустой список.
5. Допишите в файле `eval.py` функцию `track_movement`.
   Функция получает на вход объект, возвращаемый `cv2.videoCapture`, и список от `load_tools`. Функция должна возвращать массив `numpy` 5×5 с элементами типа np.uint8.
6. Запустите файл `main.py` и проверьте свой алгоритм. Если программа выдала ошибку, то найдите и исправьте ее в файле `eval`, вновь запустите файл `main.py`.
7. Пришлите решение на онлайн-платформу для проверки. В качестве решения необходимо сдать отредактированный файл `eval.py` или `*.zip` архив с ним. В архиве должны находиться файлы, а не одноименная архиву папка.

Технические ограничения

Размер решения ограничен: не более 30 МБ. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решение можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки:

`Python 3.8.10; catboost 1.1.1; dlib 19.24.0; gast 0.4.0; h5py 3.7.0; imutils 0.5.4; keras 2.9.0; Keras-Preprocessing 1.1.2; matplotlib 3.6.2; numpy 1.23.2; opencv-python 4.6.0.66; pandas 1.5.1; scikit-image 0.19.3; scikit-learn 1.1.3; scipy 1.9.3; tensorflow-cpu 2.9.2; torch 1.13.0; torchaudio 0.13.0; torchvision 0.14.0.`

Используйте совместимые пакеты.

Материалы задачи, предоставленные участникам

https://disk.yandex.ru/d/mBYdqmLvAlY-vw.

Формат входных данных

Видеофайл (объект `cv2.VideoCapture`).

Формат выходных данных

Матрица 5×5 в виде `numpy` массива `np.uint8`.

Файл решения

https://disk.yandex.ru/d/08KKZ-RM3sOaaQ.

Тесты

https://disk.yandex.ru/d/--xCth7CZR2lcQ.

Решение

Для решения данной задачи не обязательно определять координаты автомобиля на каждом кадре. Достаточно из кадра в кадр проверять, не находится ли автомобиль на том или ином перекрестке. Если составить список перекрестков, по котором проехал автомобиль, будет нетрудно заполнить требуемую матрицу.

Для локализации положения автомобиля изображение бинаризуется и выполняется операция заполнения пустот в белых объектах.

Рис. 3. Автомобиль после бинаризации и заполнения пустот

На изображении выбраны несколько ключевых пикселей, каждая соответствует конкретному перекрестку.

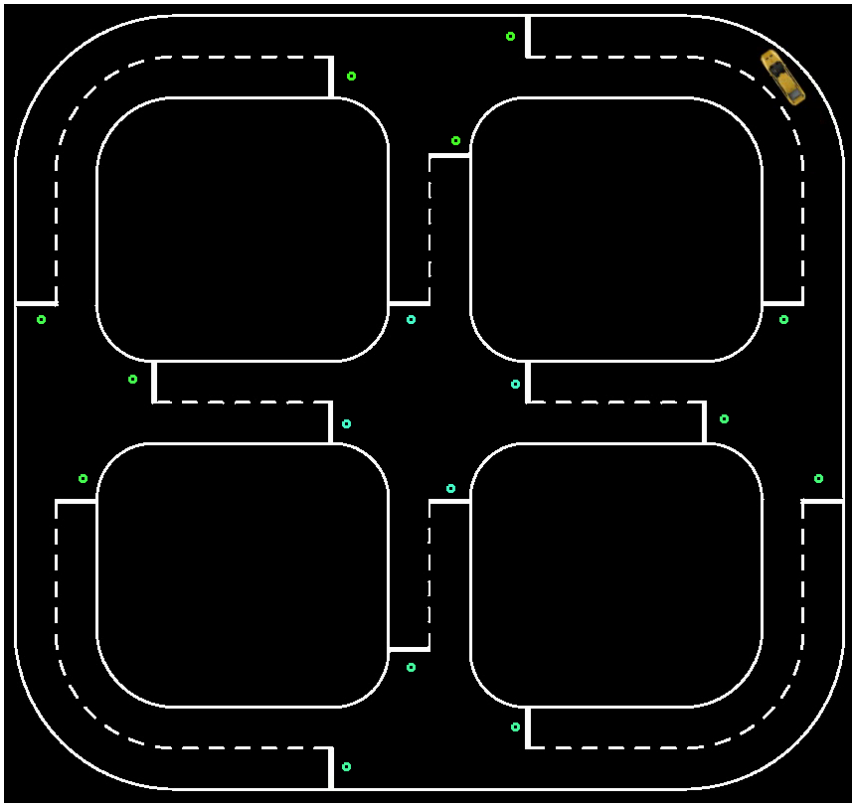

Рис. 4. Ключевые пиксели изображения

Если пиксель не черный, значит автомобиль находится на его перекрестке. Проверяя состояние ключевых пикселей определяем через какие перекрестки проехал беспилотник. Остается заполнить требуемую матрицу, этот алгоритм вынесен в отдельную функцию.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import cv2
import numpy as np
import math

# Не используется
def load_tools():
    tools = []
    return tools

# Функция возвращает номер перекрестка,на котором сейчас находится автомобиль,
                                                                    # либо 0.
def intersection_number(img):
    # Если хотя бы один из ключевых пикселей не черный,то
    # возвращаем номер перекрестка,соответствующего не черному пикселю.
    if (np.any(img[95, 360] != [0, 0, 0]) or
        np.any(img[160, 465] != [0, 0, 0]) or
        np.any(img[55, 520] != [0, 0, 0])):
```

```
18          return 1
19
20      if (np.any(img[340, 48] != [0, 0, 0]) or
21          np.any(img[400, 140] != [0, 0, 0]) or
22          np.any(img[500, 90] != [0, 0, 0])):
23          return 2
24
25      if (np.any(img[340, 795] != [0, 0, 0]) or
26          np.any(img[500, 830] != [0, 0, 0]) or
27          np.any(img[440, 735] != [0, 0, 0])):
28          return 4
29
30      if (np.any(img[790, 355] != [0, 0, 0]) or
31          np.any(img[690, 420] != [0, 0, 0]) or
32          np.any(img[750, 525] != [0, 0, 0])):
33          return 5
34
35      if (np.any(img[445, 355] != [0, 0, 0]) or
36          np.any(img[510, 460] != [0, 0, 0]) or
37          np.any(img[340, 420] != [0, 0, 0]) or
38          np.any(img[405, 525] != [0, 0, 0])):
39          return 3
40      return 0
41
42  # Функция заполняет требуемую матрицу на основе списка посещенных перекрестков.
43  def creating_graph(crossroad):
44      # Создаем матрицу 5x5,заполненную нулями.
45      result = np.zeros((5, 5), dtype=np.int8)
46      # Перебираем все посещенные перекрестки,кроме последнего.
47      for n, i in enumerate(crossroad[:-1]):
48          # n+1 - номер следующего посещенного перекрестка.
49          # crossroad[n + 1] - 1  - индекс в матрице,
50                                                  следующего посещенного перекрестка
51          # [i - 1] - индекс в матрице,текущего посещенного перекретска
52          result[i - 1][crossroad[n + 1] - 1] += 1
53      return result
54
55  # Основная функция
56  def track_movement(video, tools) -> int:
57      """ Функция для отслеживания маршрута автомобиля.
58          Входные данные: видео-объект (cv2.VideoCapture)
59          Выходные данные: матрица смежности графа в виде питру массива
60                                                              (dtype=np.uint8)
61          Примеры вывода:
62              [[0,0,0,1,0],
63               [0,0,1,0,0],
64               [1,0,0,0,0],
65               [0,0,1,0,0],
66               [0,0,0,0,0]]
67      """
68
69      crossroad = [0]  # Список посещенных перекрестков.
70      # Ядро свертки для операции заполнения внутренних пустот на черно-белом
        ↪ изображении
71      st1 = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
72
73      while True:  # Цикл чтения кадров.
74          ret_val, img = video.read()
75          if not ret_val:
76              break  # Выходим из цикла,если кадры закончились.
```

```
77
78          # Для удобства бинаризации,переводим изображение в формат HSV
79          hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
80          # Бинаризуем изображение так,чтобы белыми остались только
81                                                              # пиксели автомобиля
82          maks_e = cv2.inRange(hsv, (10, 100, 100), (56, 246, 255))
83          cv2.imshow("mask_e_1", maks_e)
84          # Заполняем темные промежутки,внутри черно-белого изображения автомобиля
85          img_car = cv2.morphologyEx(maks_e, cv2.MORPH_CLOSE, st1, iterations=3)
86          cv2.imshow("img_car", img_car)
87
88          # Проверяем не проехала ли машина,по какому то из перекрестков.
89          # Запоминаем значение последнего посещенного перекрестка.
90          # Добавляем значение в список только тогда,
91                                          # когда оно не совпадает с ранее записанным.
92          # Т.е. при первом посещении нового перекрестка.
93          old = intersection_number(img_car)
94          if crossroad[-1] != old and old:
95              crossroad.append(old)
96
97      # Заполняем матрицу посещений ребер графа.
98      # Первый элемент списка - 0,не учитываем.
99      result = creating_graph(crossroad[1:])
100     return result
```

Задача IV.4. Подсчет грузов (20 баллов)

Темы: детектирование объектов по цветам, сравнение сравнением с шаблоном, контурный анализ, бинаризация.

Условие

Есть набор изображений с кубиками. На каждом изображение кубики разложены определенным образом, отличным от остальных изображений. Кубик может занимать одну из 18 позиций. По девять позиций в каждом слое. Соседние кубики всегда соприкасаются гранями полностью — угол в угол. Кубик второго слоя не может висеть в воздухе, он обязательно стоит на кубике первого слоя.

Ваша задача написать функцию, которая определит количество кубиков и их типы. Кубики отличаются друг от друга маркировкой. Всего четыре типа кубиков.

Рис. 5. Типы маркировок кубиков

Функция должна вернуть массив $3 \times 3 \times 2$, где значение элемента определяет тип кубика на изображении.

Возможные значения: 0 — позиция пуста, 1 — невидимый кубик, 2 — красный кубик, 3 — зеленый кубик, 4 — серый кубик, 5 — оранжевый кубик.

Выполнение

1. Скачайте материалы задания: https://disk.yandex.ru/d/0Flrb4devulnCg.
2. Ознакомьтесь с материалами задания.
   Для вас подготовлены несколько файлов `.py`, набор изображений и аннотации к нему. Среди подготовленных файлов есть:
   - `eval.py` — файл с функцией, определяющей количество кубиков на изображении и их типы; именно эту функцию Вам необходимо дописать;
   - `main.py` — файл проверяющий точность работы вашего алгоритма; не редактируйте его. `main.py` использует, написанные вами функции из `eval.py` и сверяет верные ответы с предсказанием вашего алгоритма;
   - `annotations.csv` — файл, устанавливающий соответствие между изображениями и массивами для них; в каждой строке файла содержится путь к файлу с изображением и соответствующий ему правильный ответ.

   В качестве решения необходимо сдать отредактированный файл `eval.py`. Либо архив `*.zip` с файлом `eval.py` и остальными файлами, требующимися для его работы.
3. Прочитайте файл `eval.py`.
   В файле содержатся функции `count_the_types_of_cubes` и `load_tools`.
4. Допишите в файле `eval.py` функцию `load_tools`.
   Функция используется для загрузки моделей машинного обучения из файлов или загрузки эталонных изображений. Если вы не используете ее, пусть возвращает пустой список.
5. Допишите в файле `eval.py` функцию `count_the_types_of_cubes`.
   Функция получает на вход `BGR`-изображение и список от `load_tools`. Функция должна возвращать массив `numpy` $3 \times 3 \times 2$ с элементами типа `np.uint8`.
6. Запустите файл `main.py` и проверьте свой алгоритм.
   Если программа выдала ошибку, то найдите и исправьте ее в файле `eval`, вновь запустите файл `main.py`.
7. Пришлите решение на онлайн платформу для проверки.
   В качестве решения необходимо сдать отредактированный файл `eval.py` или `*.zip` архив с ним. В архиве должны находиться файлы, а не одноименная архиву папка.

Технические ограничения

Размер решения ограничен: не более 30 МБ. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решение можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки:

`Python 3.8.10; catboost 1.1.1; dlib 19.24.0; gast 0.4.0; h5py 3.7.0; imutils 0.5.4; keras 2.9.0; Keras-Preprocessing 1.1.2; matplotlib 3.6.2; numpy 1.23.2; opencv-python 4.6.0.66; pandas 1.5.1; scikit-image 0.19.3;`

`scikit-learn 1.1.3; scipy 1.9.3; tensorflow-cpu 2.9.2; torch 1.13.0; torchaudio 0.13.0; torchvision 0.14.0.`

Используйте совместимые пакеты.

Материалы задачи, предоставленные участникам

https://disk.yandex.ru/d/0Flrb4devulnCg.

Формат входных данных

Изображение `np.uint8` в формате `BGR`.

Формат выходных данных

Массив `numpy` $3 \times 3 \times 2$ элементами `np.uint8`.

Файл решения

https://disk.yandex.ru/d/zvxHIthEh8DnfQ.

Тесты

https://disk.yandex.ru/d/Gx4VIqYo6bs3tg.

Решение

Решить задачу можно множеством способов, в том числе и с применением машинного обучения. Но из-за небольшого количества предоставленных данных, постоянного освещения, простоты и ограниченности вариантов расположения объектов задачу целесообразно решать без применения машинного обучения.

После анализа предоставленных данных, можно выделить несколько областей на изображении, соответствующие граням кубиков на нем.

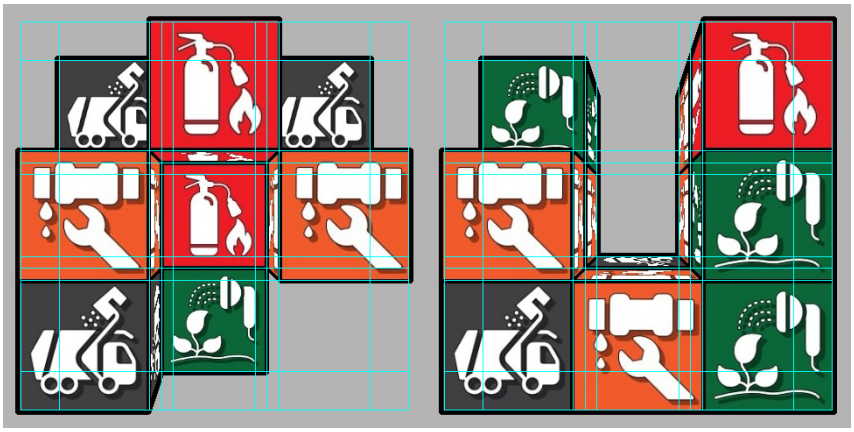

Рис. 6. Области, соответствующие граням кубиков

В каждой области подсчитывается количество пикселей цветов кубиков и черного цвета. Анализ цветов областей позволяет однозначно определить статус кубика в них. Крайние области позволяют определить статус кубиков верхнего слоя, кроме центрального.

Для определения статуса центрального кубика нужно проанализировать центральную область и соседние с ней. Если в них есть черный цвет, значи центральный верхний куб отсутствует, иначе он зарывал бы черные грани кубиков под собой. Если черного нет, значит цвет центральной области определяет цвет центрального кубка верхнего слоя.

Статус угловых кубиков нижнего слоя определяется по трем областям, соответствующим возможно видимым граням кубика. Для неугловых кубиков анализируются две области, соответствующие возможно видимым граням. Для центрального нижнего кубика анализируется одна центральная область.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import cv2
import numpy as np

# Не используется
def load_tools() -> list:
    tools = []
    return tools

# Ниже будет пояснение
def get_color(arr):
    return sum(sum(arr > 0))

# Определение статуса не угловых кубиков нижнего слоя
def get_bottom_center_color(color, top, col1, box1):
    # color - цвет верхней грани нижнего кубика
    # top - статус кубика над проверяемым
    # col1 - сторона кубика смотрящая в центр
    # box1 - статус центрального нижнего кубика
    if top > 0:  # если верхний кубик есть,то
        if box1 == 0:  # если центрального нижнего кубика нет,то
            return col1  # принимаем за его цвет грань смотрящую в центр
        else:
            return 1  # иначе считаем кубик невидимым
    else:  # если верхнего кубика нет,
        # то принимаем за цвет нижнего цвет верхней грани
        return color

# Определение статуса угловых кубиков нижнего слоя
def get_bottom_corner_color(color, top, col1, box1, col2, box2):
    # color - цвет верхней грани нижнего кубика.
    # top - статус кубика над проверяемым.
    # col1 - цвет первой из двух возможно видимых боковых сторон кубика.
    # box1 - статус кубика смежного с col1. Он закрывает col1,если есть.
    # col2 - цвет второй из двух возможно видимых боковых сторон кубика.
    # box2 - статус кубика смежного с col2. Он закрывает col2,если есть.
    if top > 0: # если кубика сверху есть,то...
        if box1 == 0:  # если нет первого кубика сбоку,то
```

```
38              # цвет грани смотрящей на него принимаем за цвет нашего кубика
39              return col1
40          elif box2 == 0:  # если нет второго кубика сбоку,то
41              return col2  # аналогично
42          else:
43              return 1  # считаем наш кубик невидимым
44      else:
45          return color
46
47  # Подобранные координаты точек. Образованные ими прямоугольные области
48  # соответствуют пикселям конкретных граней кубиков.
49  xx = [79, 123, 226, 240, 253, 347, 360, 374, 478, 522]
50  yy = [79, 123, 226, 240, 253, 347, 360, 374, 478, 522]
51
52  # Доля пикселей,необходимых для определения цвета области изображения.
53  perc = 0.01
54
55  # Для каждой области высчитываем число пикселей равных 1% от ее площади.
56  limits = np.zeros((len(xx)-1, len(yy)-1), dtype=int)
57  for ix in range(len(xx) - 1):
58      for iy in range(len(yy) - 1):
59          # Площадь области умноженная на 0.01.
60          limits[ix, iy] = int((xx[ix+1]-xx[ix])*(yy[iy+1]-yy[iy])*perc)
61
62  # Основная функция,вызывается для каждого входящего изображения отдельно
63  def count_the_types_of_cubes(image, tools, debug=False) -> np.array:
64      # Бинаризуем изображение так,чтобы белыми стали пиксели,
65      # соответствующие конкретному типу кубиков.
66      # Сохраняем черно-белые изображения полученные в результате бинаризации.
67      cubes2_red = cv2.inRange(image, (30, 20, 230), (45, 37, 250))
68      cubes3_green = cv2.inRange(image, (50, 90, 0), (65, 110, 35))
69      cubes4_gray = cv2.inRange(image, (50, 50, 50), (70, 70, 70))
70      cubes5_orange = cv2.inRange(image, (33, 80, 190), (70, 110, 255))
71
72      cubes_black = cv2.inRange(image, (0, 0, 0), (10, 10, 10))  # Белыми станут
        ↪  ребра кубиков.
73
74      # a - список хранящий статус каждой области.
75      a = []
76      colors = (cubes2_red, cubes3_green, cubes4_gray, cubes5_orange)
77      # Пробегаем по всем областям и считаем число цветных пикселей в них.
78      for ix in range(len(xx)-1):
79          row = []  # строка для хранения статусов областей.
80          for iy in range(len(yy)-1):
81              best_col = 0
82              best_val = 0
83              for col in range(len(colors)):
84                  # get_color - считает число белых пикселей в области.
85                  val = get_color(colors[col][xx[ix]:xx[ix + 1], yy[iy]:yy[iy + 1]])
86                  if val > best_val:  # Ищем область,в которой белого больше всего.
87                      best_val = val  # Запоминаем максимум белых пикселей.
88                      best_col = col  # Индекс цвета,которого больше всего.
89
90              # Для каждой области добавляем индекс цвета,0 - если ничего нет.
91              if best_val > limits[ix, iy]:  # Если белого больше чем порог,то
92                  row.append(best_col+2)  # запоминаем индекс +2.
93                  # +2 потому,что 0 - отсутствующий кубик,1 - невидимый кубик.
94              else:
95                  row.append(0)  # Если в области нет искомых цветов,
96                                                      #  считаем что нет и кубиков.
```

```
97          a.append(row)  # Добавляем строку в список.
98      a = np.array(a)  # Превращаем A в массив питру.
99
100     i_mid = len(a) // 2  # Индекс центральной области.
101
102     # Проверяем наличие черного в области вокруг центра,
103     # соответствующей боковым и нижней грани центрального нижнего куба
104     if get_color(cubes_black[xx[i_mid-1]:xx[i_mid + 2],
105                                          yy[i_mid-1]:yy[i_mid + 2]]) > 30:
106         black_center = 0  # Если черное есть,
107                           # то считаем центральный куб в верхнем отсутствующим
108     else:
109         black_center = 10
110
111     # Определяем статус кубиков верхнего слоя по областям,
112     # которые однозначно позволяют идентифицировать наличие/отсутствие и
113     # цвет кубика.
114     up_layer = np.array([
115         [a[0, 1], a[0, i_mid], a[0, -2]],
116         [a[i_mid, 0], min(black_center, a[i_mid, i_mid]), a[i_mid, -1]],
117         [a[-1, 1], a[-1, i_mid], a[-1, -2]]
118                         ])
119
120     # Заполняем 0-ми массив для хранения статусов кубиков нижнего слоя.
121     bottom_layer = np.zeros(9, dtype=int).reshape(3, 3)
122
123
124     # Определяем статус центрального кубика нижнего слоя.
125     if up_layer[1, 1] == 0:  # Если нет центрального кубика верхнего слоя,то...
126         # определяем статус кубика по центральной области
127         bottom_layer[1, 1] = a[i_mid, i_mid]
128     else:
129         bottom_layer[1, 1] = 1  # невидимый
130
131     #  Находим значение для кубиков нижнего слоя.
132     #  Логика определения статуса не угловых кубиков нижнего слоя вынесена в
133     #  отдельную функцию.
134     #  Смотри get_bottom_center_color.
135     bottom_layer[0, 1] = get_bottom_center_color(a[1, i_mid],
136                                  up_layer[0, 1], a[3, i_mid], bottom_layer[1, 1])
137     bottom_layer[2, 1] = get_bottom_center_color(a[-2, i_mid],
138                                  up_layer[2, 1], a[-4, i_mid], bottom_layer[1, 1])
139     bottom_layer[1, 0] = get_bottom_center_color(a[i_mid, 1],
140                                  up_layer[1, 0], a[i_mid, 3], bottom_layer[1, 1])
141     bottom_layer[1, 2] = get_bottom_center_color(a[i_mid, -2],
142                                  up_layer[1, 2], a[i_mid, -4], bottom_layer[1, 1])
143
144     #  Логика определения статуса угловых кубиков нижнего слоя вынесена
145                                                          # в отдельную функцию.
146     #  Смотри get_bottom_corner_color.
147     bottom_layer[0, 0] = get_bottom_corner_color(a[1, 1], up_layer[0, 0],
148                                                  a[3, 1], bottom_layer[1, 0],
149                                                  a[1, 3], bottom_layer[0, 1])
150     bottom_layer[2, 0] = get_bottom_corner_color(a[-2, 1], up_layer[2, 0],
151                                                  a[-4, 1], bottom_layer[1, 0],
152                                                  a[-2, 3], bottom_layer[2, 1])
153     bottom_layer[0, 2] = get_bottom_corner_color(a[1, -2], up_layer[0, 2],
154                                                  a[3, -2], bottom_layer[1, 2],
155                                                  a[1, -4], bottom_layer[0, 1])
156     bottom_layer[2, 2] = get_bottom_corner_color(a[-2, -2], up_layer[2, 2],
```

```
157                                                 a[-4, -2], bottom_layer[1, 2],
158                                                 a[-2, -4], bottom_layer[2, 1])
159
160     result = np.array([bottom_layer, up_layer])
161
162     return result
```

Задача IV.5. Полет квадрокоптера (15 баллов)

Темы: управление квадрокоптером в ROS, нейросетевые детекторы, обнаружение объектов в видеопотоке.

Условие

Напишите программу — полетное задание для квадрокоптера. Коптеру необходимо взлететь, выполнить перелет в точку `HOME`, после чего переместится в точку номер 1. Далее необходимо выполнить перелет по маршруту на изображении ниже.

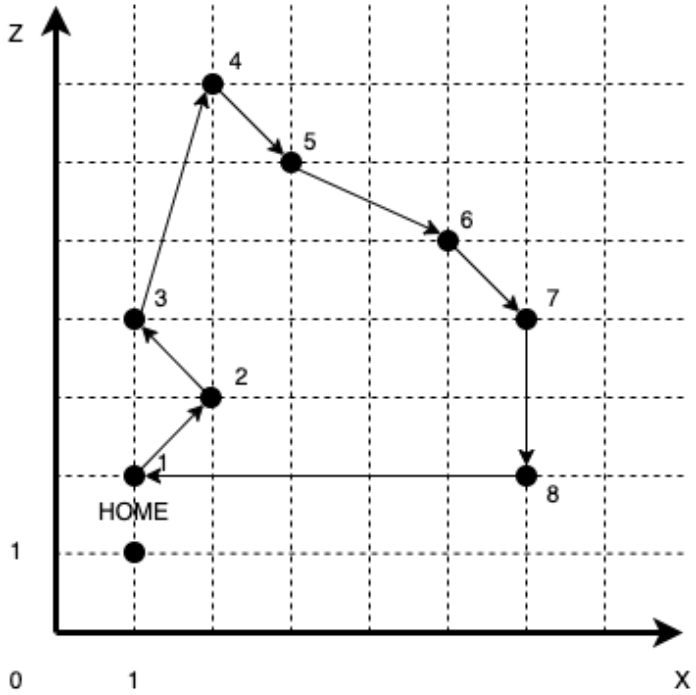

Рис. 7. Маршрут квадрокоптера

После выполнения пролета по маршруту необходимо вернуться в точку `HOME` и выполнить посадку.

Полетное задание задается с помощью массива `coordinates`. Точки в массиве задаются последовательно. Точка `HOME` задается с помощью переменной `home_point`. Для успешного выполнение задания необходимо верно изменить указанные переменные.

Полет совершается в вертикальной плоскости (координаты X и Z изменяются, Y всегда равна нулю). Для взлета используется функция `ap.takeoff()`, за посадку отвечает функция `ap.landing()`.

Выполнение: скачайте материалы задания: https://disk.yandex.ru/d/AmtYYjNWKYB6xg.

Для вас подготовлен файл `eval.py`. Он содержит шаблон полетного задания с некоторыми ошибками и недоработками. Вам необходимо внимательно ознакомиться с файлом, решить, какие части кода требуют изменений, и отредактировать их. Пример рабочего скрипта полета по заданной траектории представлен по ссылке: https://github.com/geoscan/gs_example/blob/noetic/src/flight_test.py.

В качестве решения необходимо сдать на проверку отредактированный файл `eval.py`.

Внимание! Для решения задачи вы вносите изменения только в файл `eval.py`. Никаких дополнительных библиотек, кроме подключенных в коде `eval.py`, использовать нельзя. Ознакомьтесь с документацией и примерами программирования Пионер Макс. https://github.com/geoscan/geoscan_pioneer_max.

Технические ограничения

Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решение можно прислать сразу.

Материалы задачи, предоставленные участникам

https://disk.yandex.ru/d/AmtYYjNWKYB6xg.

Файл решения

https://disk.yandex.ru/d/_YYCykWYZRTEVA.

Решение

Участникам необходимо добавить в массив с координатами точек назначения и исправить порядок вызова функций взлета и посадки.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import rospy
from gs_flight import FlightController, CallbackEvent
from gs_board import BoardManager

rospy.init_node("flight_test_node")

home_point = [1, 0, 1]

```

```
coordinates = [
    home_point,
    [1, 0, 2],
    [2, 0, 3],
    [1, 0, 4],
    [2, 0, 7],
    [3, 0, 6],
    [5, 0, 5],
    [6, 0, 4],
    [6, 0, 2],
    [1, 0, 2],
    home_point]

run = True
position_number = 0

def callback(event):
    global ap
    global run
    global coordinates
    global position_number

    event = event.data
    if event == CallbackEvent.ENGINES_STARTED:
        print("engine started")
        ap.takeoff()
    elif event == CallbackEvent.TAKEOFF_COMPLETE:
        print("takeoff complite")
        position_number = 0
        ap.goToLocalPoint(coordinates[position_number][0],
                          coordinates[position_number][1],
                          coordinates[position_number][2])
    elif event == CallbackEvent.POINT_REACHED:
        print(f"point {position_number} reached")
        position_number += 1
        if position_number < len(coordinates):
            ap.goToLocalPoint(coordinates[position_number][0],
                              coordinates[position_number][1],
                              coordinates[position_number][2])
        else:
            ap.landing()
    elif event == CallbackEvent.COPTER_LANDED:
        print("programm finish")
        run = False

board = BoardManager()
ap = FlightController(callback)
once = False

while not rospy.is_shutdown() and run:
    if board.runStatus() and not once:
        print("start programm")
        ap.preflight()
        once = True
    pass
```