Автономные транспортные системы
Заключительный этап
Инженерный тур

Общая информация

На заключительном этапе профиля АТС, участникам необходимо запустить и отладить автоматизированную транспортную систему для доставки груза до адресата без вмешательства человека. Система состоит из трех программируемых устройств: беспилотного автомобиля, распределительного хаба и квадрокоптера. Участники разрабатывают и отлаживают программы, управляющие устройствами автономной транспортной системы.

Легенда задачи

Администрация города M устраивает конкурс на разработку автоматизированной системы доставки грузов между предприятиями города и горожанами. Заказываемая система должна быть полностью автономной. Она состоит из беспилотного автомобиля, автоматизированного сортировочного хаба и квадрокоптера.

В конкурсе участвует несколько команд разработчиков. Команда, которая к концу конкурса продемонстрирует наиболее совершенный прототип слаженно работающей транспортной системы, станет победителем.

Совершенность автономной транспортной системы зависит от количества объектов городской среды, с которыми система может корректно взаимодействовать.

Требования к команде и компетенциям участников

Количество участников в команде: 3–4 человека.

1. Программист беспилотного автомобиля. Базовые знания электроники, работа с алгоритмами локального позиционирования беспилотных автомобилей, разработка и отладка алгоритмов обнаружения и взаимодействия с объектами городской среды.
2. Программист квадрокоптера. Базовое знание ROS, работа с компьютерным зрением и нейронными сетями в задачах квадрокоптеров, осуществляющих навигацию над полигоном воссозданной городской среды.
3. Программист сортировочного хаба. Определение трехмерных координат объектов по изображению с камеры, работа с алгоритмами детектирования и классификации цветовых меток на грузах.
4. Капитан команды. Оценка сложности и декомпозиция задач, распределение задач по участникам команды, отслеживание сроков выполнения задач, работа с компьютерным зрением и нейронными сетями.

Оборудование и программное обеспечение

Оборудование

Полигон «АЙКАР Стенд», версия «Город НТО» и макеты зданий для полигона «Город НТО»	Интерактивный полигон городской среды с зданиями, дорожной разметкой и объектами городской среды: пешеходами, дорожными знаками, светофорами. Моделирует город, в котором необходимо разработать и запустить автономную транспортную систему. Доступ к полигону участники получают по расписанию. На полигоне участники отлаживают всю транспортную систему в целом, решают подзадачи беспилотного автомобиля и квадрокоптера.
УМК АЙКАР	Программируемая учебная модель беспилотного автомобиля АЙКАР, испытательный полигон «АЙКАР Стенд», версия «Восьмерка» и объекты городской среды: пешеходы, дорожные знаки. Доступен участникам в течение всего заключительного этапа. Используется для запуска и отладки программ локального позиционирования беспилотного автомобиля.
Квадрокоптер Пионер Макс	Доступен участникам в течение всего заключительного этапа. Используется для запуска и отладки программ квадрокоптера.
Автоматизированный сортировочный хаб	Устанавливается на полигоне «Город НТО» и является связующим звеном между беспилотным автомобилем и квадрокоптером. Доступ к сортировочному хабу участники получают по расписанию. Используется для запуска и отладки программ распознающих маркировки на грузах и перемещающих грузы в соответствии с их маркировкой.
Стационарный компьютер для обучения нейронных сетей	Доступен участникам в течение всего заключительного этапа. Используется для обучения нейронных сетей и отладки программ, использующих их.
Ноутбук для инференса нейронных сетей	Доступен участникам в течение всего заключительного этапа. Используется для взаимодействия с программируемыми устройствами на полигоне и отладки алгоритмов компьютерного зрения и нейросетевых алгоритмов.
Ноутбук для редактирования программного кода	Доступен участникам в течение всего заключительного этапа. Используется для чтения документации, коммуникации с организаторами, поиска данных в интернете и редактирования программного кода.

Программное обеспечение

Tengine	Механизм для конвертации моделей нейросетей и их высокопроизводительного инференса в встраиваемых устройствах. Используется при квантовании нейросетевого детектора и запуске квантованного нейросетевого детектора на NPU.

Python 3	Основной язык для написания алгоритмов компьютерного зрения и работы с нейросетями. Используется для написания программ всех устройств транспортной системы.
OpenCV (библиотека для python 3)	OpenCV — библиотека алгоритмов компьютерного зрения, обработки изображений, численных алгоритмов и инференса нейросетей с открытым кодом. Используется для написания программ всех устройств транспортной системы.
Darknet (фреймворк)	Darknet — это фреймворк для обучения нейросетевых детекторов с открытым исходным кодом, написанный на языке C с использованием программно-аппаратной архитектуры параллельных вычислений CUDA. Используется для обучения нейросетевых детекторов.
ROS (операционная система)	ROS — это экосистема для программирования роботов, предоставляющая функциональность для распределенной работы. Используется при написании и запуске программ для квадрокоптера.
LabelImg	Приложение для разметки датасетов.
PuTTY	Клиент для различных протоколов удаленного доступа. Используется для подключения к устройствам транспортной системы по SSH.

Описание задачи

На заключительном этапе профиля АТС, участникам необходимо разработать и отладить автоматизированную транспортную систему для доставки груза до адресата без вмешательства человека. Система состоит из трех устройств: беспилотного автомобиля, распределительного хаба и квадрокоптера.

Все перечисленные устройства работают на полигоне воссозданной городской среды с макетами зданий, дорогами, перекрестками, дорожными знаками, пешеходами и светофорами.

Рис. VI.2.1. Полигон «АЙКАР Стенд», версия «Город НТО»

Беспилотный автомобиль доставляет груз к распределительному хабу через город, соблюдая правила дорожного движения, реагируя на пешеходов и светофоры.

Распределительный хаб оснащен захватом для перемещения и сортировки грузов, привезенных автомобилем. Доставленные к хабу грузы, он должен отсортировать, выбрать груз, предназначенный для доставки квадрокоптером, и переместить груз к захвату квадрокоптера.

Квадрокоптер располагается на крыше сортировочного хаба, он захватывает груз и доставляет его в центр специальной разгрузочной площадки, располагающейся на одной из крыш зданий полигона, после чего совершает приземление на специально оборудованную крышу.

Устройства, запрограммированные участниками, должны слаженно работать в единой транспортной системе и осуществлять доставку груза до адресата без участия человека.

Задача разработки автономной транспортной системы разбита на несколько более простых подзадач. **14, 15 и 16 марта** участники решают эти подзадачи и получают за них баллы. **17 марта** — день финальных заездов. На финальном заезде участники демонстрируют слаженную работу всей транспортной системы. Если система доставки груза полностью работоспособна, участники получают баллы за финальный заезд.

- Описание подзадач находятся в разделах «Подзадачи беспилотного автомобиля», «Подзадачи сортировочного хаба», «Подзадачи квадрокоптера».
- Описание правил проведения финальных заездов находятся в разделе «Финальные заезды»

Подзадачи беспилотного автомобиля (БПА)

Материалы, предоставленные участникам на старте заключительного этапа, находятся в папке: https://disk.yandex.ru/d/HXJZKlsMGcgF-A

Количество попыток для сдачи подзадач ограничено только временем доступа к устройству. Подзадачи могут быть сданы в любом порядке.

Коммутация электронных модулей беспилотного автомобиля

Необходимо изучить инструкцию по работе с моделью беспилотного автомобиля АЙКАР, соединить электронные модули согласно схеме подключения и продемонстрировать работу базового программного кода.

Для выполнения подзадачи необходимо:

- показать разработчику профиля скоммутированные электронные модули и получить разрешение на включение питания модели беспилотного автомобиля;
- запустить на модели беспилотного автомобиля АЙКАР базовый программный код и продемонстрировать его работу разработчику профиля.

Типовые ошибки при выполнении подзадания: невнимательное чтение инструкции, включение питания без разрешения разработчика профиля.

Доставка груза к сортировочному хабу из случайной точки города

АЙКАР с грузом устанавливается в одну из трех случайных стартовых позиций.

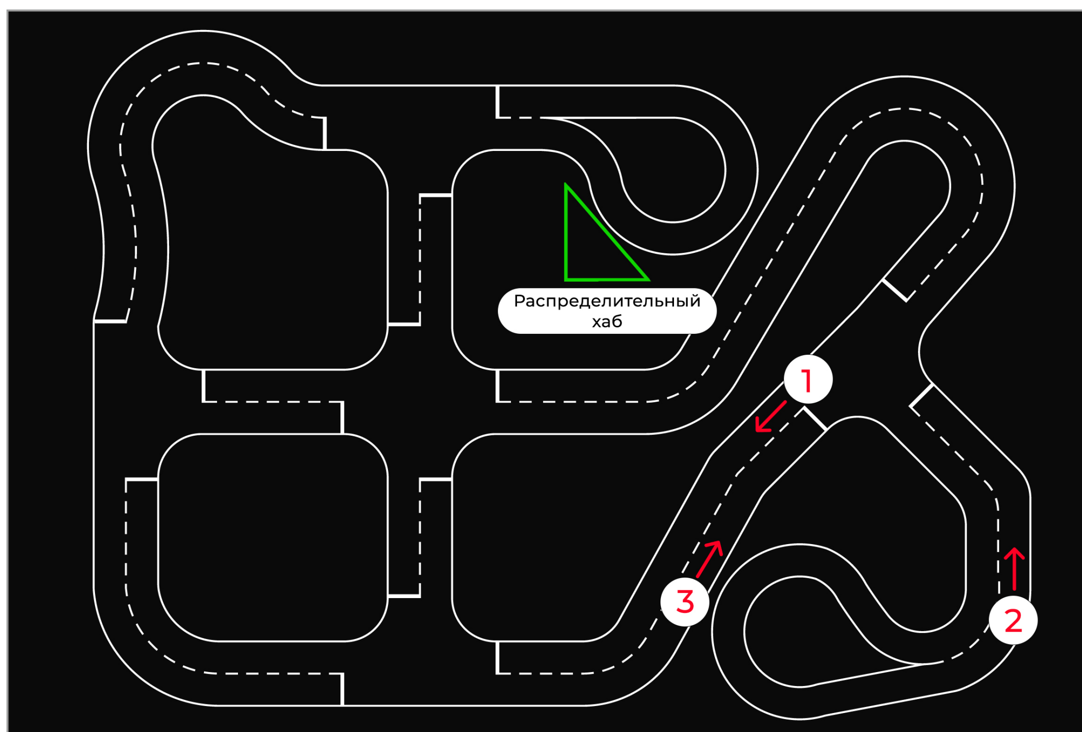

Рис. VI.2.2. Стартовые положения

По команде разработчика участники запускают программу на беспилотном автомобиле.

АЙКАР должен распознать в какой стартовой позиции он находится, проложить маршрут до сортировочного хаба, пройти по маршруту и оставить груз в зоне разгрузки.

Для проверки будет проводиться 2 испытания из разных стартовых позиций.

Подзадача считается выполненной, если:

- при движении через город беспилотный автомобиль не выходил за пределы дорожной разметки более чем одним колесом
- груз доставлен в зону разгрузки
- подряд проведены два успешных испытания

Типовые ошибки при выполнении подзадания:

- Использование таймера, встроенного в бортовой компьютер беспилотника для измерения пройденной дистанции. Подобным образом можно получить решение, но оно не будет стабильным на 100%. При последовательных запусках одной и той же программы, за одинаковое время беспилотник пройдет разное расстояние. Для измерения пройденной беспилотником дистанции необходимо использовать счетчик оборотов двигателя.
- Попытки жестко задать время начала и конца поворотов. Подобный подход приводит к нестабильно работающим решениям, которые невозможно отладить. В стабильных решениях беспилотник представляется системой с обратной связью, реагирующей на внешние обстоятельства.
- Использование нейросетевых классификаторов для определения стартовой позиции. На бортовом компьютере беспилотника гораздо проще запустить и отладить алгоритмы, основанные на сравнении кэша или гистограмм изображений.

Обучение нейросетевого детектора объектов городской среды и его запуск на нейронном процессоре

АЙКАР устанавливается в случайно выбранное место на полигоне «Восьмерка». В 50 см перед беспилотником располагаются работающий светофор и случайно выбранный знак дорожного движения. В 30 см перед автомобилем размещается случайно выбранный пешеход.

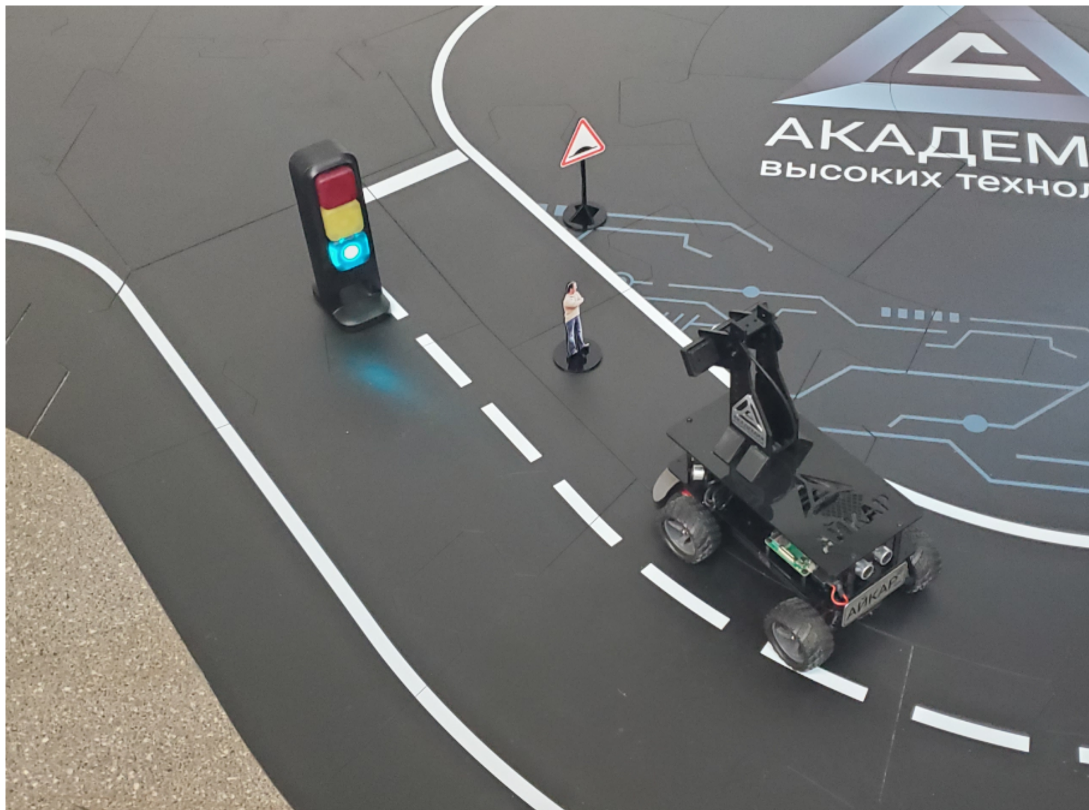

Рис. VI.2.3. АЙКАР перед объектами

Необходимо обучить нейросетевой детектор объектов городской среды, квантовать его и разработать программный код для инференса детектора на нейронном процессоре: https://ru.wikipedia.org/wiki/Нейронный_процессор.

Подзадача считается выполненным, если АЙКАР выводит в терминал верное количество объектов перед собой и изменяет это количество при удалении и добавлении объектов.

Типовые ошибки при выполнении подзадания:

- Отсутствие негативных примеров в обучающем датасете.
- Одинаковые условия освещения на всех изображениях обучающего датасета.
- В обучающем датасете только одиноко стоящие объекты или наоборот только рядом стоящие группы объектов.
- Невнимательное изучение инструкции по квантованию и запуску детектора на NPU.
- Неудачное изменение anchor boxes нейросетевого детектора.

Поиск парковочного места, отмеченного знаком, и выполнение парковки

АЙКАР устанавливается в положение аналогичное следующим.

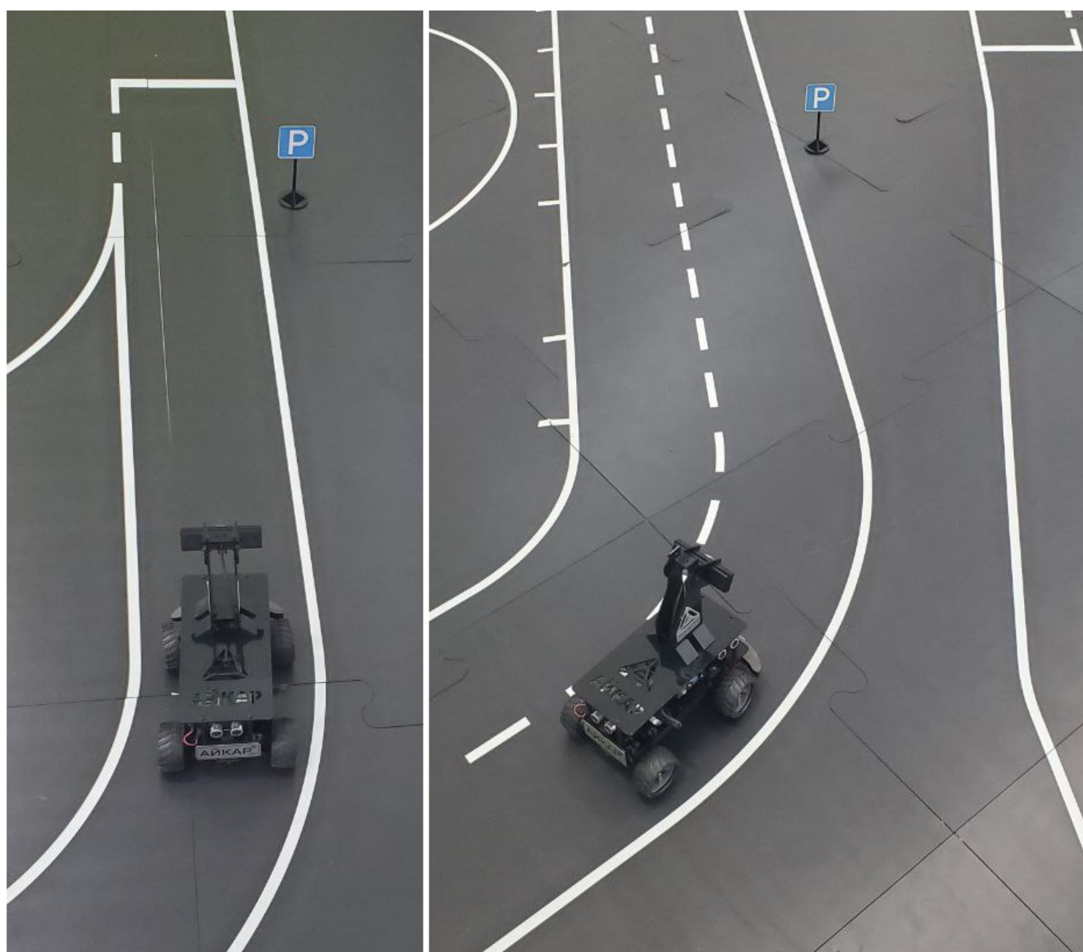

Рис. VI.2.4. АЙКАР перед парковкой

От знака «Парковка» до края дороги 10 см. Перед знаком «Парковка» обязательно прямой участок дороги минимальной длиной 70 см. Айкар устанавливается на дорогу минимум за 70 см от знака «Парковка».

Парковочное место — прямоугольник 45×25 см. Парковочное место располагается в 3 см от края дороги и в 15 см перед знаком парковка.

По команде разработчика участники запускают программу на беспилотном автомобиле.

Задание считается выполненным, если АЙКАР остановился и все четыре колеса находятся в пределах парковочного места.

Для проверки будет проводиться 2 испытания, каждое должно пройти успешно. Успешные попытки должны идти подряд.

Типовые ошибки при выполнении подзадания:

- Использование таймера, встроенного в бортовой компьютер беспилотника для измерения пройденной дистанции. При последовательных запусках одной и той же программы, за одинаковое время беспилотник пройдет разное расстояние. Для измерения пройденной беспилотником дистанции необходимо использовать счетчик оборотов двигателя
- Попытки жестко задать время начала и конца поворотов. Подобный подход приводит к нестабильно работающим решениям, которые невозможно отладить. В стабильных решениях беспилотник представляется системой с обрат-

ной связью, реагирующей на внешние обстоятельства.

Преодоление перекрестка с неисправным светофором

АЙКАР устанавливается перед перекрестком со светофорами. Светофор, который должен подавать сигнал потоку машин с беспилотным автомобилем, неисправен. Исправен светофор, направленный перпендикулярно к неработающему светофору.

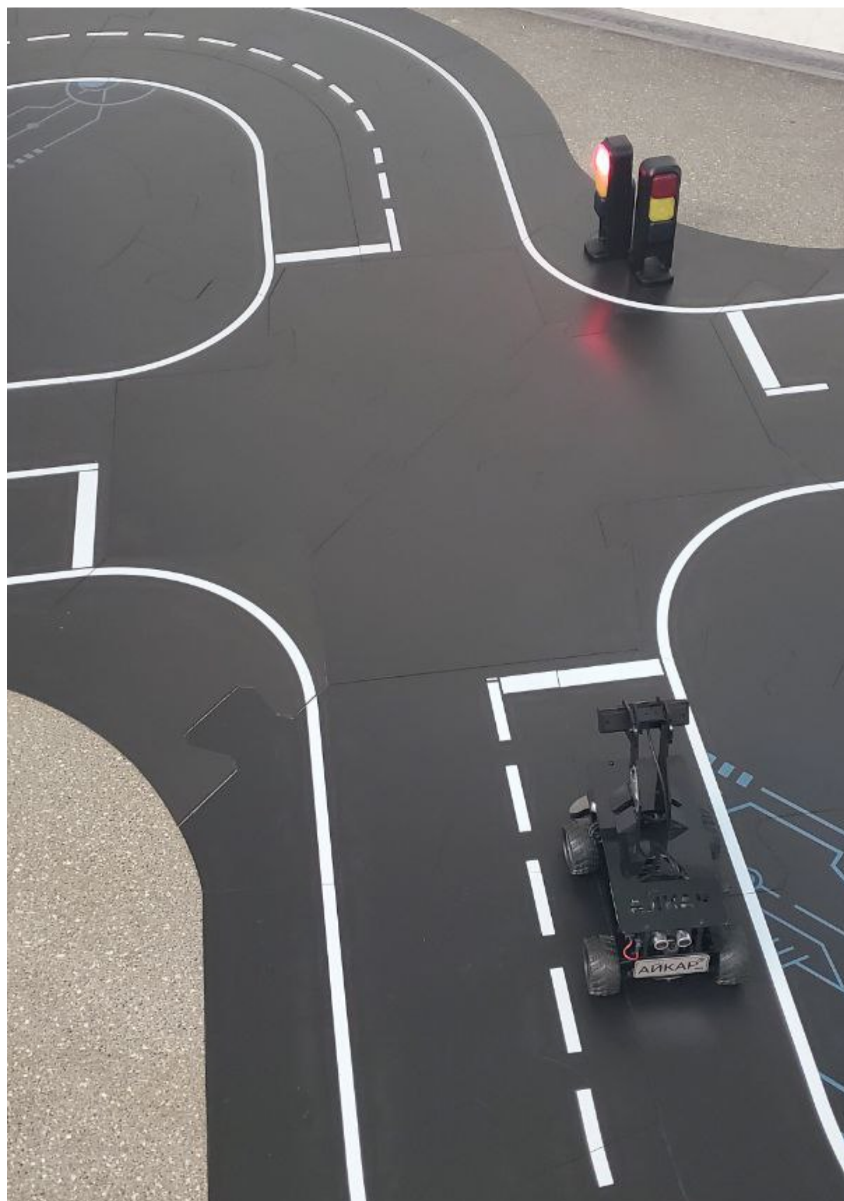

Рис. VI.2.5. АЙКАР на перекрестке

По команде разработчика участники запускают программу на беспилотном автомобиле.

Подзадача считается выполненным, если АЙКАР пересечет перекресток по прямой в течение времени, когда неисправный светофор должен подавать зеленый сигнал.

Для проверки будет проводиться 2 испытания, каждое должно пройти успешно. Успешные попытки должны идти подряд.

Типовые ошибки при выполнении подзадания:

- Обучение отдельного нейросетевого детектора для поиска светофоров, стоящих под углом 90 градусов к наблюдателю. Достаточно детектировать неисправный светофор и в его окрестности искать зеленые, желтые и красные пиксели.
- Начало движения на зеленый сигнал светофора, стоящего под углом 90 градусов к наблюдателю.

Взаимодействие с пешеходами и знаком дорожного движения «Пешеходный переход»

АЙКАР устанавливается в начало участка дороги без перекрестков. По направлению движения беспилотного автомобиля на обочине дороги и проезжей части располагаются несколько пешеходов. На обочине дороги установлен знак «Пешеходный переход».

Рис. VI.2.6. АЙКАР и пешеходы

По команде разработчика участники запускают программу на беспилотном автомобиле.

Подзадача считается выполненной, если:

- АЙКАР останавливается в 20–30 см от пешехода на проезжей части и продолжает движение после исчезновения пешехода.

- беспилотный автомобиль снижает скорость движения за 30 см до пешехода или знака «Пешеходный переход».
- АЙКАР возобновляет движение с нормальной скоростью, когда рулевые колеса поравняются с пешеходом или знаком «Пешеходный переход».

Дистанция отсчитывается от середины рулевых колес автомобиля.

Типовые ошибки при выполнении подзадания:

- Область поиска пешеходов, перед которыми нужно останавливаться одинакова для прямых участков дороги и поворотов.
- Неверное определение дистанции до объекта, по его площади.

Подзадачи сортировочного хаба (СХ)

Количество попыток для сдачи подзадач ограничено только временем доступа к устройству.

Подзадача 2 может быть сдана только после выполнения подзадачи 1.

Перемещение указанного груза из зоны разгрузки в захват квадрокоптера

После доставки груза беспилотным автомобилем АЙКАР в зоне разгрузки находятся два груза. Маркировка груза, доставленного беспилотным автомобилем, известна участникам заранее.

Подзадача считается выполненной, если доставленный беспилотником груз графически выделен на изображении с камеры сортировочного хаба, либо для всех грузов подписаны расшифровки маркировок.

Типовые ошибки при выполнении подзадания:

- Отладка программы на изображениях, полученных не с камеры сортировочного хаба, пороги бинаризации, подобранные на изображениях с другой камеры, не подойдут для камеры сортировочного хаба. Разные объективы имеют разную цветопередачу.
- Пороги бинаризации слишком узкие. При небольшой смене освещения контуры объектов перестают детектироваться.

Перемещение указанного груза из зоны разгрузки в захват квадрокоптера и распределение оставшихся грузов по накопителям

После доставки груза беспилотным автомобилем АЙКАР в зоне разгрузки находятся четыре груза.

Подзадача считается выполненной, если на изображении с камеры графически обозначено куда необходимо переместить грузы, в соответствии с следующими условиями:

- каждый груз с ровно двумя одинаковыми символами в маркировке в первый накопитель.
- во второй накопитель перемещаются грузы, маркировка которых начинается с «0».

- в третьем накопителе собираются грузы, в маркировке которых нет «G».
- грузы с маркировкой, не подходящей под условия выше, передают в захват квадрокоптера.

В зоне разгрузки гарантированно нет грузов, которые подходят под несколько условий сразу.

Типовые ошибки при выполнении подзадания:

- Отладка программы на изображениях полученных не с камеры сортировочного хаба.Пороги бинаризации, подобранные на изображениях с другой камеры, не подойдут для камеры сортировочного хаба. Разные объективы имеют разную цветопередачу.
- Пороги бинаризации слишком узкие. При небольшой смене освещения контуры.
- Логические ошибки при определении соответствия условия и расшифровки маркировки груза.

Подзадачи квадрокоптера

SDK для программирования квадрокоптера: https://github.com/geoscan/geoscan_pioneer_max.

Работа с квадрокоптерами осуществляется по следующим правилам:

1. Для разработки и отладки программного кода каждая команда-участник подключается к выданному ей квадрокоптеру. Каждой команде предоставляется один квадрокоптер. Смена квадрокоптера на протяжении соревнования не допускается.
2. Всю работу по программированию и запуску алгоритмов распознавания и полета участники производят самостоятельно. Один из участников команды производит установку квадрокоптера на стартовую позицию.
3. Запуск программы на квадрокоптере команда-участник производит только после подтверждения запуска оператором! При нарушении данного условия команда дисквалифицируется за нарушение техники безопасности при эксплуатации беспилотного летательного аппарата. Данное требование не относится к запуску программы полета в симуляторе без включения двигателей квадрокоптера.
4. Команде предоставляется две аккумуляторные батареи для квадрокоптера. За степенью зарядки аккумуляторных батарей команда-участник следит самостоятельно. В случае, если к моменту отладочной или финальной попытки у команды не осталось заряженных батарей, команда не допускается к полету.
5. За сохранность программного кода (например, путем создания бэкапов на локальном компьютере) несут ответственность команды-участники.
6. Сбор и разметку датасетов команды-участники выполняют самостоятельно.

Количество попыток для сдачи подзадач ограничено только временем доступа к устройству.

Каждая следующая подзадача квадрокоптера включает в себя предыдущие. Поэтому при выполнении любой подзадачи квадрокоптера, все предшествующие подзадачи засчитываются автоматически!

Начальные условия у всех подзадач квадрокоптера одинаковые. Перед стартом участникам сообщают пару пиктограмм, составляющих логотип получателя груза. Логотипом отмечена одна из разгрузочных площадок на крыше здания № 1. Квадрокоптер располагается на крыше сортировочного хаба. Груз уже доставлен к захвату квадрокоптера.

Выполнить захват груза и перелет с грузом на здание № 1

Подзадача считается решенной, если квадрокоптер последовательно выполнил следующие действия:

- включил магнитный захват;
- взлетел с крыши распределительного хаба на произвольную высоту вместе с захваченным грузом;
- активировал мигающую светодиодную индикацию (цвет — соответствующий логотипу получателя, интервал мигания 4 с);
- пролетел по маршруту № 1.

Типовые ошибки при выполнении подзадания: использование для перелета по маршруту №1 координат из симуляции. Координаты в симуляции и в реальной жизни отличаются на 0,5 метра по всем измерениям из-за физических принципов работы ультразвуковой системы навигации.

Верифицировать графическую метку, соответствующую подразделению-получателю груза

Подзадача считается решенной, если квадрокоптер выполнил подзадачу 1 и последовательно совершил следующие действия:

- детектировал пару пиктограмм, составляющих логотип получателя груза;
- вывел в терминал единственное текстовое сообщение с названиями: первой пиктограммы, второй пиктограммы и компании-получателя груза;
- активировал мигающую светодиодную индикацию (цвет — белый, интервал мигания 1 с);
- выполнил зависание на 10 секунд;
- отключил светодиодную индикацию.

Типовые ошибки при выполнении подзадания:

- Привязка каждой пиктограммы к координатам в ультразвуковой системе навигации.
- Детектирование пиктограмм по одной.

Произвести доставку груза подразделению-получателю

Задание считается решенным, если квадрокоптер выполнил подзадачу 2 и после этого оставил груз на крыше здания № 1. Груз должен лежать сразу на двух пиктограммах логотипа получателя, с учетом расположения пары — горизонтальное или вертикальное. Положение груза оценивается после полной остановки груза.

Типовые ошибки при выполнении подзадания:

- Привязка каждой пиктограммы к координатам в ультразвуковой системе навигации. Для точного позиционирования необходимо использовать алгоритмы компьютерного зрения и математические расчеты для корректировки курса, поскольку точности ультразвуковой системы навигации не хватает для достаточного для детектирования позиционирования.
- Сбрасывание груза с большой высоты. Кубик при падении отскакивает от крыши здания и может окончить свое движение далеко от предполагаемой точки сброса.

Произвести детектирование груза на крыше здания №2

Задание считается решенным, если квадрокоптер выполнил подзадачу 3 и после этого последовательно совершил следующие действия:

- пролетел по маршруту № 2;
- детектировал груз на крыше здания № 2;
- активировал мигающую светодиодную индикацию (цвет — попеременно мигающий красный и синий, интервал мигания 1 с);
- вывел в терминал единственное сообщение с номером грузовой площадки, на которой находится детектированный груз.

Типовые ошибки при выполнении подзадания:

- Обучение отдельного нейросетевого детектора для нахождения груза. Для выполнения задачи достаточно вычислить разницу между текущим изображением грузовой площадки и заранее подготовленным изображением этой же площадки без груза.
- Использование для перелета по маршруту № 2 координат из симуляции. Координаты в симуляции и в реальной жизни отличаются на 0,5 метра по всем измерениям из-за физических принципов работы ультразвуковой системы навигации.

Произвести захват груза на крыше здания № 2

Задание считается решенным, если квадрокоптер выполнил подзадачу 4 и после этого последовательно выполнил следующие действия:

- включение магнитного захвата, снижение и взятие груза магнитным захватом,
- взлет с крыши здания № 2 вместе с грузом.

Типовые ошибки при выполнении подзадания:

- Использование координат ультразвуковой системы навигации для позиционирования над грузом. Для точного позиционирования необходимо использовать алгоритмы компьютерного зрения и математические расчеты для корректировки курса.
- Использование встроенной в API функции `landing` для посадки. Функция `landing` гарантирует посадку, но не гарантирует что она будет произведена точно вдоль оси Z, для выполнения точной посадки необходимо выполнить позиционирование по камере, снижение на высоту порядка 15 сантиметров от крыши здания и использование функции остановки двигателей (`disarm`).

Выполнить доставку груза и произвести посадку в центре посадочной площадки

Подзадача имеет два варианта выполнения.

Первый вариант

Подзадача считается выполненной, если квадрокоптер последовательно совершил следующие действия:

- выполнил подзадачу 3;
- пролетел по маршруту 4;
- включил мигающую светодиодную индикацию (цвет — попеременно мигающий красный и белый, интервал мигания 1 с);
- завис на 5 с;
- приземлился как можно ближе к центру площадки;
- выключил светодиодную индикацию.

Второй вариант

Подзадача считается выполненной, если квадрокоптер последовательно совершил следующие действия:

- выполнил подзадачу 5;
- пролетел по маршруту 4;
- включил мигающую светодиодную индикацию (цвет — попеременно мигающий красный и белый, интервал мигания 1 с);
- завис на 5 с;
- приземлился как можно ближе к центру посадочной площадки;
- выключил светодиодную индикацию.

Чем ближе к центру посадочной площадки приземлится квадрокоптер, тем больше баллов получат участники. Центр посадочной площадки — зеленая точка. Центр квадрокоптера совпадает с центром магнита.

Типовые ошибки при выполнении подзадания:

- Использование для перелета по маршруту № 4 координат из симуляции. Координаты в симуляции и в реальной жизни отличаются на 0,5 метра по всем измерениям из-за физических принципов работы ультразвуковой системы навигации.
- Привязка центра посадочной площадки к координатам в ультразвуковой системе навигации. Для выполнения этой задачи необходимо выполнить позиционирование по камере над зеленой точкой (центр площадки).

Финальные заезды

В день финальных заездов задача участников — продемонстрировать слаженную работу всех трех программируемых устройств: беспилотного автомобиля, распределительного хаба, квадрокоптера.

Беспилотный автомобиль, должен доставить груз к распределительному хабу. Распределительный хаб должен провести сортировку грузов в зоне разгрузки и переместить один из них в захват квадрокоптера. Квадрокоптер должен доставить груз

от сортировочного хаба компании-получателю и совершить посадку на крыше одного из зданий.

Если хотя бы одно из трех устройств не справилось с поставленной задачей, то вся транспортная система считается неработоспособной, и баллы за финальный заезд не начисляются. На финальный заезд каждой команде выделяется 15 минут. Количество попыток перезапуска системы ограничено только временем.

Финальный заезд разбит на базовую (обязательную) часть и усложнения. Базовая часть — это минимум того, что должна продемонстрировать команда для получения баллов за финальный заезд. Усложнения — элементы финального заезда, дополняющие базовую часть. Усложнения увеличивают баллы, получаемые командой за финальный заезд.

Команды заранее должны сообщить какие усложнения они выбирают для финального заезда.

Список элементов финального заезда представлен в таблице, лист «Элементы финального заезда».

Правила начисления баллов за финальные заезды описаны в разделе «Система оценивания».

Система оценивания

В первые три дня участники получают баллы за выполнение подзадач. Баллы за подзадачи начисляются согласно таблице.

Максимальная сумма баллов за выполнение всех подзадач — 100 баллов.

Подзадачи беспилотного автомобиля		
	Описание подзадачи	Баллы за подзадачу
Подзадача № 1	Коммутация электронных модулей беспилотного автомобиля	2
Подзадача № 2	Доставка груза к сортировочному хабу из случайной точки города	8
Подзадача № 3	Обучение нейросетевого детектора объектов городской среды и его запуск на нейронном процессоре	9
Подзадача № 4	Поиск парковочного места, отмеченного знаком, и остановка на нем	6
Подзадача № 5	Проезд перекрестка с неисправным светофором	5
Подзадача № 6	Взаимодействие с пешеходами	10
	Сумма баллов за все подзадачи	40

Подзадачи распределительного хаба		
	Описание подзадачи	Баллы за подзадачу
Подзадача № 1	Перемещение указанного груза из зоны разгрузки в захват квадрокоптера	6
Подзадача № 2	Перемещение указанного груза из зоны разгрузки в захват квадрокоптера и распределение оставшихся грузов по накопителям	14
	Сумма за все подзадачи	20

Подзадачи квадрокоптера		
	Описание подзадачи	Баллы за подзадачу
Подзадача № 1	Выполнить захват груза и перелет с грузом на здание № 1	3
Подзадача № 2	Верифицировать графическую метку, соответствующую подразделению-получателю груза	10
Подзадача № 3	Произвести доставку груза подразделению-получателю	9 — груз находится сразу на обоих изображениях заданной пары 6 — груз находится на одном из изображений заданной пары 3 — груз находится не далее 5 см от одного из изображений заданной пары
Подзадача № 4	Произвести детектирование груза на крыше здания № 2	6
Подзадача № 5	Произвести захват груза на крыше здания № 2	4
Подзадача № 6	Выполнить доставку груза и произвести посадку в центре посадочной площадки	8 — центр квадрокоптера находится не далее 5 см от центра метки 6 — центр квадрокоптера находится не далее 10 см от центра метки 4 — центр квадрокоптера находится на расстоянии более 10 см от центра метки
	Сумма баллов за все подзадачи	40

В день финальных заездов участники получают баллы за выполнение финального заезда. Баллы за финальный заезда начисляются согласно таблице. **Максимальная сумма баллов за финальный заезда — 100 баллов.**

		Базовая часть	**Усложнение 1**	**Усложнение 2**	**Усложнение 3**
Беспилотный автомобиль	**Описание**	Доставка груза к сортировочному хабу из случайной точки города	Поиск парковочного места, отмеченного знаком, и остановка на нем	Проезд перекрестка с неисправным светофором	Взаимодействие с пешеходами
	Баллы	10	10	10	10

Распределительный хаб	**Описание**	Перемещение указанного груза из зоны разгрузки в захват квадрокоптера	Перемещение указанного груза из зоны разгрузки в захват квадрокоптера и распределение оставшихся грузов по нахкопителям		
	Баллы	6	14		
Квадрокоптер	**Описание**	Доставить груз на графическую метку получателя груза и преземлиться на крыше зания 4	Детектировать груз и вывести номер его площадки	Детектировать и захватить груз	
	Баллы	25 — метка верна, груз находится на обоих изображениях 22 — метка верна, груз находится на одном изображении 19 — метка верна, груз находится не далее 5 см от одного из изображений 5 — центр квадрокоптера находится не далее 5 см от центра посадочной площадки 3 — центр квадрокоптера находится не далее 10 см от центра посадочной площадки 1 — центр квадрокоптера находится на расстоянии более 10 см от центра посадочной площадки	6	4	

Баллы, полученные за подзадачи, складываются с баллами, полученными за финальные заезды. Первое место занимает команда с наибольшей суммой баллов.

Максимальный балл, который можно получить командой — 200 баллов.

Баллы участника в индивидуальном зачете вычисляются следующим образом:

$$\text{Физика} \times 0,15 + \text{Информатика} \times 0,15 + \text{Командный балл} \times 0,7.$$

Первое место занимает участник с наибольшим количеством баллов. Остальные участники располагаются за ним в порядке убывания количества баллов.

При равенстве баллов побеждает команда, получившая больший балл за финальный заезд. Если баллы за финальный заезд одинаковые, побеждает команда, быстрее выполнившая финальный заезд.

Автономные транспортные системы
Заключительный этап
Инженерный тур

Решение задачи

Подзадачи беспилотного автомобиля (БПА)

Материалы, предоставленные участникам на старте заключительного этапа, находятся в папке: https://disk.yandex.ru/d/HXJZKlsMGcgF-A.

Файлы с решениями подзадач находятся в папке: https://disk.yandex.ru/d/AMgiQ-gag5pYtA.

Коммутация электронных модулей беспилотного автомобиля

Необходимо изучить инструкцию по работе с моделью беспилотного автомобиля АЙКАР, соединить электронные модули согласно схеме подключения и продемонстрировать работу базового программного кода.

Доставка груза к сортировочному хабу из случайной точки города

После запуска беспилотного автомобиля он должен самостоятельно определить в какой из трех стартовых позиций находится. Для этого можно сформировать набор изображений с камеры, характерных для каждой точки старта. И при запуске сравнивать то, что видит беспилотник с эталонными изображениями. По тому с каким эталонным изображением есть совпадения, можно судить о стартовой позиции. В приведенном решении на изображениях проводится поиск особых точек и сравнивается их количество.

Стартовая позиция определяет кратчайший маршрут до сортировочного хаба, количество перекрестков и способы их преодоления.

При движении через полигон города беспилотный автомобиль должен переключаться с одного алгоритма движения на другой. Базовый алгоритм движения по разметке, должен сменяться алгоритмом пересечения перекрестка по прямой и направо. Переключиться на алгоритм пересечения перекрестка можно тогда, когда в кадре появится стоп-линия. Переключиться с алгоритма преодоления перекрестка на алгоритм движения по разметке можно если в верхней части кадра появились белые пиксели линий разметки.

При преодолении перекрестка необходимо реагировать на изменение изображения и подруливать в нужную сторону. Случай поворота направо простейший вариант того, как можно преодолевать перекресток. В правой части кадра всегда будет видна сплошная линия разметки. Для того, чтобы повернуть направо достаточно реагировать только на правую линию. В базовом алгоритме можно программно заменить не найденную левую линию на виртуальную левую линию, которая всегда отстоит от правой на заданном расстоянии. В представленном решении выполнена модификация базового алгоритма, позволяющая проезжать по перекресткам направо.

При преодолении перекрестка по прямой тоже необходим ориентир, на который можно рулить. Таким ориентиром может быть черная дорожная полоса, до которой беспилотник должен добраться. На изображении перекрестка, который видит

беспилотник, если считать от низа столбца черные пиксели пока не встретишь белый пиксель, то в столбцах нужной нам полосы движения белые пиксели встретятся максимально высоко. Столбы с наибольшем количеством черных пикселей снизу — являются нашим ориентиром, на который мы рулим. В приведенном решении реализован именно такой способ преодоления перекрестков по прямой.

```
import cv2
import numpy as np
from arduino import Arduino  # класс из базового кода для общения с Arduino
from road_utils import *  # набор функций базового кода
import yolopy  # библиотека для инференса моделей Yolo на NPU

CAR_SPEED = 1430  # начальная скорость автомобиля

# Задаем константы состояний
GO = "GO"  # автомобиль движется по разметке
STOP = "STOP"  # автомобиль в состоянии покоя - стои
# автомобиль преодолевает перекресток, поворачивая направо
CROSS_RIGHT = "CROSS_RIGHT"
# автомобиль преодолевает перекресток, проезжая прямо
CROSS_STRAIGHT = "CROSS_STRAIGHT"
# автомобиль подъезжает к сортировочному хабу
HUB = "HUB"

STATE = GO  # задаем начальное состояние
PREV_STATE = None  # переменная для хранения предыдущего состояния

def detect_stop(perspective):
    """
    Функция для детектирования стоп-линии
    Возвращает True, если линия обнаружена
    """
    # суммируем значения пикселей каждой строки
    hist = np.sum(perspective, axis=1)
    # находим индекс строки с максимальным количеством белых пикселей
    maxStrInd = np.argmax(hist)
    # Если пикселей больше 150 и максимально белая строка находится ниже 120
    ↪  строки, то
    if hist[maxStrInd]//255 > 150:
        if maxStrInd > 120:
            return True  # считаем стоп линию обнаруженной
    return False

# функция для переключения с алгоритма пересечения перекрестка на алгоритм
↪  следование по разметке
def detect_road_begin(perspective):
    """
    Функция для детектирования боковых линий после пересечения перекрестка.
    Возвращает True, если линии обнаружены.
    """
    # считаем число белых пикселей в нижней левой и нижней правой части
    ↪  изображения
    left_corner = np.sum(perspective[-50:, :perspective.shape[1] // 3])
    right_corner = np.sum(perspective[-50:, perspective.shape[1] // 3 * 2:])
    # если пикселей больше определенного порога, то считаем линии разметки
    ↪  детектированными
    if left_corner >= 170000 and right_corner >= 170000:
        return True
    else:
        return False
```

```
51
52  # Инициализация работы с камерой
53  cap = cv2.VideoCapture('/dev/video0')
54
55  # Инициализация обмена данными с Arduino через UART
56  arduino = Arduino.find_connected_arduino(baudrate=115200, timeout=10, debug=True)
57
58  # читаем кадр с камеры, чтобы определить стартовое положение автомобиля
59  _, frame = cap.read()
60  # переводим изображение в оттенки серого
61  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
62  # Чтение изображений известных точек
63  start_point1 = cv2.imread('start_point1.jpg', cv2.IMREAD_GRAYSCALE)
64  start_point2 = cv2.imread('start_point2.jpg', cv2.IMREAD_GRAYSCALE)
65  start_point3 = cv2.imread('start_point3.jpg', cv2.IMREAD_GRAYSCALE)
66
67  # Создание детектора особых точек
68  orb = cv2.ORB_create()
69
70  # Находим на каждом эталонном изображении особые точки
71  kp1, des1 = orb.detectAndCompute(start_point1, None)
72  kp2, des2 = orb.detectAndCompute(start_point2, None)
73  kp3, des3 = orb.detectAndCompute(start_point3, None)
74  # Находим особые точки на кадре с камеры
75  kp_current, des_current = orb.detectAndCompute(frame, None)
76
77  # Создаем объект для сопоставления особых точек дух изображений
78  bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
79
80  # Сопоставляем точки каждого эталонного изображения с особыми точками кадра с
    ↪  камеры
81  matches1 = bf.match(des1, des_current)
82  matches2 = bf.match(des2, des_current)
83  matches3 = bf.match(des3, des_current)1
84
85  #  В зависимости от того, на каком изображении больше совпадений с кадром,
86  #  решаем в какой позиции находимся и определяем каким образом необходимо
    ↪  преодолевать перекрестки,
87  #  чтобы добраться до сортировочного хаба
88  if len(matches1) > len(matches2) and len(matches1) > len(matches3):
89      # стартуем с первой позиции
90      path = [CROSS_RIGHT, CROSS_STRAIGHT, CROSS_RIGHT, HUB]
91  elif len(matches2) > len(matches1) and len(matches2) > len(matches3):
92      # стартуем с второй позиции
93      path = [CROSS_RIGHT, CROSS_RIGHT, CROSS_RIGHT, HUB]
94  else:
95      # стартуем с третьей позиции
96      path = [CROSS_STRAIGHT, CROSS_RIGHT, CROSS_RIGHT, HUB]
97      # path - список маневров, которые надо совершить на перекрестках,
98      # чтобы добраться до сортировочного хаба
99
100 while True:
101     success, frame = cap.read()  # читаем кадр с камеры
102     if not success:  # если не удалось получить кадр, выходим из цикла
103         print("Can't open camera")
104         break
105
106     img = cv2.resize(frame, SIZE)  # приводим изображение к удобному для
        ↪  вычислений размеру
107     binary = binarize2(img, THRESHOLD)  # бинаризуем изображение для поиска белых
        ↪  линий
```

```
108         perspective = trans_perspective(binary, TRAP, RECT, SIZE)  # извлекаем область
            ↪  перед колесами
109         left, right = centre_mass(perspective)  # получаем координаты левой и правой
            ↪  линии разметки
110
111         if STATE == GO: # если состояние "Движение вперед"
112             if detect_stop(perspective): # проверяем нет ли стоп линии, если есть
                ↪  значит
113                 # мы перед перекрестком.
114                 # Переходим в состояние преодоления перекрестка тем методом,
115                 # который записан первым в нашем списке маневров
116                 STATE = path.pop(0)
117
118         if STATE == CROSS_RIGHT: # Если поворачиваем направо, то
119             if detect_road_begin(perspective): # проверяем нет ли впереди линий
                ↪  разметки
120                 STATE = GO # если есть, значит мы проехали перекресток и
121                 # нужно переключиться на алгоритм следования по разметке
122
123             # создаем виртуальную левую линию. при повороте направо левой линии
                ↪  разметки нет
124             left = right - int(SIZE[0]*0.6)  # отступ от правой линии на заданное
                ↪  значение
125             left = max(0, left)  # чтобы значение не было отрицательным
126
127         if STATE == HUB: # если мы уже доехали до хаба,
128             STATE = STOP  # то останавливаемся и
129             arduino.drop() # разгружаемся
130
131         # вычисляем отклонение центра дороги от центра кадра
132         err = 0 - ((left + right) // 2 - SIZE[0] // 2)
133         if abs(right - left) < 100:
134             err = last_err
135         last_err = err
136
137         if STATE == CROSS_STRAIGHT: # если мы проезжаем перекресток по прямой, то
138             if detect_road_begin(perspective): # проверяем нет ли впереди линий
                ↪  разметки
139                 STATE = GO  # если есть, значит мы проехали перекресток и
140                 # нужно переключиться на алгоритм следования по разметке
141
142             # вычисляем положение полосы движения, в которую собираемся попасть после
                ↪  проезда перекрестка
143             # положение полосы движения совпадает с самыми темными стольцами пикселей
144             bin = np.flip(binary, axis=0)
145             hist = np.argmax(bin, axis=0)
146
147             # выбираем 30 столбцов, которые в большинстве своем больше среднего
148             pull_size = np.max(hist) / np.mean(hist)
149             if pull_size >= 1.6:  # 1.6
150                 pull_size = int(50 * pull_size)
151             else:
152                 pull_size = 30
153
154             # Вычисляем индексы самых темных столбцов
155             ind_max_elem = np.argsort(hist)[-pull_size:]
156             # Вычисляем отклонение самых темных столбцов от центра кадра
157             err = (bin.shape[1] // 2) - int(np.mean(ind_max_elem))
158
159         # преобразуем ошибку в угол поворота рулевых колес через ПД-регулятор
```

```
160     angle = int(90 + KP * err + KD * (err - last_err))
161 
162     # проверка на превышение допустимых углов поворота колес
163     if angle < 60:
164         angle = 60
165     elif angle > 120:
166         angle = 120
167 
168     # Если предыдущее состояние не равно текущему, то
169     if STATE != PREV_STATE:
170         print("STATE:", STATE)
171         PREV_STATE = STATE
172         # изменяем скорость в соответствии с новым состоянием
173         if STATE == GO:
174             arduino.set_speed(CAR_SPEED)
175         elif STATE == STOP:
176             arduino.stop()
177 
178     arduino.set_angle(angle)  # устанавливаем угол поворота рулевых колес
179 
180 # Остановка автомобиля и разблокировка доступа к камере.
181 arduino.stop()
182 arduino.set_angle(90)
183 cap.release()
```

Обучение нейросетевого детектора объектов городской среды и его запуск на нейронном процессоре

Сложность этого задания не в написании программного кода, а в необходимости использования нескольких цифровых инструментов: средств разметки датасета, фреймворком для обучения нейросетевых детекторов, фреймворком для конвертации моделей нейронных сетей их квантования и инференса на NPU. После обучения нейросетевого детектора, чтобы его можно было запустить на NPU, необходимо квантовать модель детектора. В стандартной модели весовые коэффициенты представлены float-числами. Математические и логические операции с вещественными числами занимают больше времени, чем с целыми. Если заменить представление весовых коэффициентов вещественными числами на представление целыми числами, то мы получим квантованную модель. Быстродействие такой модели увеличено за счет более быстрых операций и специальной архитектуры NPU. Квантование — это процесс, в ходе которого float-числа заменяются на `int`-числа.

Квантованная модель должна обладать точностью сравнимой с оригинальной. Для этого при квантовании используется набор изображений. Изображение передают на вход оригинальной и квантованной модели. Выход оригинальной сети рассматривается как верный ответ для квантованной модели. Квантованная сеть учится на ответах оригинальной, поэтому набор изображений для квантования может быть неразмеченным.

После обучения нейросетевого детектора нужно было подготовить конфигурационный файл, файл с весовыми коэффициентами, набор изображений и запустить следующую команд.

```
python3 quant_tool.py -w <путь_до_весов_модели> /
                      -c <путь_до_конфигурационного_файла> /
                      -d <путь_до_папки_с_изображениями>
```

После квантования, которое длится не более получаса, файл квантованной модели можно было использовать на бортовом компьютере беспилотного автомобиля.

Представленный ниже код использует квантованную модель детектора для обнаружения объектов в кадре и вывода их количества в терминал.

```python
import cv2
import yolopy  # библиотека yolopy для работы с моделью YOLO

# чтение списка имен классов детектора
with open('classes.txt') as file:
    class_names = file.read().splitlines()

model_file = "yolov4-tiny_uint8.tmfile"  # задаем путь к файлу модели YOLO
# загружаем модель детектора из файла
model = yolopy.Model(model_file,  use_uint8=True, use_timvx=True, cls_num=10)
# задаем якоря для модели YOLO
model.set_anchors([18, 33, 33, 48, 25, 71, 58, 76, 40, 113, 87, 140])

cap = cv2.VideoCapture('/dev/video0')  # инициализируем камеру

prev_len = None  #
while True:
    success, frame = cap.read()  # читаем кадр с камеры
    if not success:  # если не удалось прочитать кадр, выходим из цикла
        print("Can't open camera")
        break

    # обнаруживаем объекты на изображении с помощью модели YOLO
    classes, scores, boxes = model.detect(frame)

    cur_len = len(classes)  # вычисляем число обнаруженных объектов
    # выводим сообщение только тогда, когда число объектов изменилось
    if prev_len != cur_len:
        prev_len = cur_len
        print("Количество объектов:", cur_len)

cap.release()
```

Поиск парковочного места, отмеченного знаком, и выполнение парковки

При выполнении задания необходимо ответить на следующие вопросы:

- Когда нужно съезжать с дороги?
- На что ориентироваться при съезде с дороги?
- Как понять, что беспилотник уже на парковочном месте и остановиться?

Определить момент, когда нужно съезжать с дороги по удаленности от беспилотника знака парковка. Дистанцию до знака можно рассчитывать по площади знака, по положению центра знака в кадре. В приведенном примере площадь знака служит сигналом для съезда с дороги.

Во время съезда с дороги на что-то необходимо ориентироваться. Ориентироваться можно на знак парковки или на сплошную линию разметки. В представленном примере линия разметки служит ориентиром, а когда линия исчезает из кадра начинается переход к движению вдоль обочины.

По дистанции до знака можно не только начать маневр парковки, но и завершить его.

В следующем решении маневр парковки разбит на две части: смещение на обочину и движение по обочине. Съезд на обочину происходит контролируемо, исчезнувшая левая линия разметки заменяется правой линией, а вместо правой создается виртуальная линия, таким образом беспилотник съезжает с дороги направо. Этот маневр продолжается до тех пор, пока сплошная линия не пропадет из кадра. Теперь поворачиваем налево, пока вновь не увидим сплошную линию, дальше мы будем ехать вдоль ней, до тех пор, пока знак парковки не приблизится достаточно.

```
import cv2
from arduino import Arduino  # класс из базового кода для общения с Arduino
from road_utils import *  # набор функций базового кода
import yolopy  # библиотека для инференса моделей Yolo на NPU

CAR_SPEED = 1430  # начальная скорость автомобиля

# Задаем константы состояний
PARKING_SHIFT = "PARKING_SHIFT"  # смещаемся на обочину дороги
PARKING_GO = "PARKING_GO"  # движемся по обочине к парковочному месту
GO = "GO"  # следуем по своей полосе движения
STOP = "STOP"  # стоим на месте

LOST_RIGHT_LINE = False  # флаг для отслеживания потери правой линии

STATE = GO  # задаем начальное состояние
PREV_STATE = None  # переменная для хранения предыдущего состояния

model_file = "yolov4-tiny_uint8.tmfile"  # задаем путь к файлу модели YOLO
# загружаем модель детектора из файла
model = yolopy.Model(model_file, use_uint8=True, use_timvx=True, cls_num=10)
# задаем якоря для модели YOLO
model.set_anchors([18, 33, 33, 48, 25, 71, 58, 76, 40, 113, 87, 140])  # задаем
↪  якорные точки для модели YOLO

cap = cv2.VideoCapture('/dev/video0')  # инициализируем камеру
# Инициализация обмена данными с Arduino через UART
arduino = Arduino.find_connected_arduino(baudrate=115200, timeout=10, debug=True)

while True:
    success, frame = cap.read()  # читаем кадр с камеры
    if not success:  # если не удалось прочитать кадр, выходим из цикла
        print("Can't open camera")
        break

    # приводим изображение к удобному для вычислений размеру
    img = cv2.resize(frame, SIZE)
    # бинаризуем изображение для поиска белых линий
    binary = binarize2(img, THRESHOLD)
    # извлекаем область перед колесами
    perspective = trans_perspective(binary, TRAP, RECT, SIZE)
    left, right = centre_mass(perspective)  # получаем координаты левой и правой
    ↪  линии разметки

    # обнаруживаем объекты на изображении с помощью модели YOLO
    classes, scores, boxes = model.detect(img)

    parking_sign_area = 0  # переменная для хранения площади знака парковки на
    ↪  изображении
    # перебираем все обнаруженные объекты
    for cls, score, box in zip(classes, scores, boxes):
```

```
49          if cls == 2:  # если обнаружен знак парковки, то
50              parking_sign_area = box[2] * box[3]  # вычисляем его площадь
51
52      if STATE == GO:  # если текущее состояние "движение по дороге" и
53          if parking_sign_area > 870:  # площадь знака больше заданного порога
54              STATE = PARKING_SHIFT  # переходим к мещению на обочину
55
56      if STATE == PARKING_SHIFT:  # если текущее состояние "смещение на обочину" и
57          if right < SIZE[0]//2 + 10:  # правая полоса дороги обнаружена
58              LOST_RIGHT_LINE = True  # устанавливаем флаг потери правой линии
59
60          if not LOST_RIGHT_LINE:  # если правая полоса дороги не потеряна
61              left = right  # заменяем левую полосу правой
62              right = SIZE[0] - 10  # и создаем виртуальную правую полосу на краю
                ↪  изображения
63          else:  # если правая полоса дороги потеряна
64              STATE = PARKING_GO  # значит мы съехали на обочину и пора двигаться по
                ↪  ней к парковочному месту
65
66      if STATE == PARKING_GO:  # если текущее состояние "движение по обочине"
67          right = left + int(SIZE[0]*0.6)  # создаем виртуальную правую линию
68          right = min(right, SIZE[0])  # проверяем чтобы она не вышла за край
            ↪  изображения
69      # если площадь знака парковки больше заданного порога
70          if parking_sign_area >= 5000:
71              STATE = STOP  # переходим в состояние "остановка"
72              # считаем, что мы достаточно приблизились к знаку парковки и заняли
                ↪  парковочное место
73
74      # вычисляем отклонение центра дороги от центра кадра
75      err = 0 - ((left + right) // 2 - SIZE[0] // 2)  # вычисляем ошибку положения
        ↪  автомобиля на дороге
76      if abs(right - left) < 100:
77          err = last_err
78      last_err = err
79
80      # преобразуем ошибку в угол поворота рулевых колес через ПД-регулятор
81      angle = int(90 + KP * err + KD * (err - last_err))
82
83      # проверка на превышение допустимых углов поворота колес
84      if angle < 60:
85          angle = 60
86      elif angle > 120:
87          angle = 120
88
89      # Если предыдущее состояние не равно текущему, то
90      if STATE != PREV_STATE:
91          print("STATE:", STATE)
92          PREV_STATE = STATE
93          # изменяем скорость в соответствии с новым состоянием
94          if STATE == GO:
95              arduino.set_speed(CAR_SPEED)
96          elif STATE == STOP:
97              arduino.stop()
98
99      arduino.set_angle(angle)  # устанавливаем угол поворота руля на Arduino
100
101 # Остановка автомобиля и разблокировка доступа к камере.
102 arduino.stop()
103 arduino.set_angle(90)
```

```
104 cap.release()
```

Преодоление перекрестка с неисправным светофором

Для преодоления перекрестка можно использовать один из алгоритмов разобраных во второй подзадаче. Но прежде, нужно определить какой сигнал подает исправный светофор. Исправный светофор подает сигнал машинам движущимся перпендикулярно направлению нашего движения. Поэтому когда светофор зажжет красный сигнал, мы можем пересекать перекресток.

Для решения этой задачи можно переобучить уже имеющийся детектор, обнаруживать светофоры в профиль.

```
1  import cv2
2  import numpy as np
3  from arduino import Arduino  # класс из базового кода для общения с Arduino
4  from road_utils import *  # набор функций базового кода
5  import yolopy  # библиотека для инференса моделей Yolo на NPU
6
7  CAR_SPEED = 1430  # начальная скорость автомобиля
8
9  # Задаем константы состояний
10 GO = "GO"  # следуем по своей полосе движения
11 STOP = "STOP"  # беспилотный автомобиль остановился
12 CROSS = "CROSS"  # преодолеваем перекресток
13
14 STATE = STOP  # задаем начальное состояние "остановка"
15 PREV_STATE = None  # переменная для хранения предыдущего состояния
16
17 # функция для переключения с алгоритма пересечения перекрестка на алгоритм
   ↪  следование по разметке
18 def detect_road_begin(perspective):
19     """
20     Функция для детектирования боковых линий после пересечения перекрестка.
21     Возвращает True, если линии обнаружены.
22     """
23     # считаем число белых пикселей в нижней левой и нижней правой части
       ↪  изображения
24     left_corner = np.sum(perspective[-50:, :perspective.shape[1] // 3])
25     right_corner = np.sum(perspective[-50:, perspective.shape[1] // 3 * 2:])
26     # если пикселей больше определенного порога, то считаем линии разметки
       ↪  детектированными
27     if left_corner >= 170000 and right_corner >= 170000:
28         return True
29     else:
30         return False
31
32 model_file = "yolov4-tiny_uint8.tmfile"  # задаем путь к файлу модели YOLO
33 # загружаем модель детектора из файла
34 model = yolopy.Model(model_file, use_uint8=True, use_timvx=True, cls_num=10)
35 # задаем якоря для модели YOLO
36 model.set_anchors([18, 33, 33, 48, 25, 71, 58, 76, 40, 113, 87, 140])  # задаем
   ↪  якорные точки для модели YOLO
37
38 cap = cv2.VideoCapture('/dev/video0')  # инициализируем камеру
39 # Инициализация обмена данными с Arduino через UART
40 arduino = Arduino.find_connected_arduino(baudrate=115200, timeout=10, debug=True)
41
42 while True:
```

```
43        success, frame = cap.read()  # читаем кадр с камеры
44        if not success:  # если не удалось прочитать кадр, выходим из цикла
45            print("Can't open camera")
46            break
47
48        img = cv2.resize(frame, SIZE)  # приводим изображение к удобному для
          ↪  вычислений размеру
49        binary = binarize2(img, THRESHOLD)  # бинаризуем изображение для поиска белых
          ↪  линий
50        perspective = trans_perspective(binary, TRAP, RECT, SIZE)  # извлекаем область
          ↪  перед колесами
51        left, right = centre_mass(perspective)  # получаем координаты левой и правой
          ↪  линии разметки
52
53        # обнаруживаем объекты на изображении с помощью модели YOLO
54        classes, scores, boxes = model.detect(img)
55
56        tl_detected = False  # флаг обнаружения светофора
57        tl_frame = None  # участок кадра со светофором
58        # перебираем все обнаруженные объекты
59        for cls, score, box in zip(classes, scores, boxes):
60            if cls == 9:  # если обнаружен светофор, то
61                x, y, w, h = box  # сохраняем параметры ограничивающей рамки и
62                tl_detected = True  # выставляем соответствующий флаг
63                tl_frame = img[y:y+h, x:x+w]  # вырезаем светофор из изображения
64
65        if STATE == STOP:  # если состояние "Остановка", то
66            if tl_detected:  # проверяем обнаружен ли светофор
67                # если обнаружен, то вырезаем из изображения верхнюю треть
68                # именно в ней находится красный сигнал светофора
69                red_frame = tl_frame[:tl_frame.shape[0]//3, :]
70                # суммируем значение всех пикселей в интересующей нас области
71                brightness = np.sum(red_frame)
72                if brightness > 290000:  # если оно больше порогового значения, то
73                    STATE = CROSS  # считаем красный свет зажженным и переходим в
74                    # состояние преодоления перекрестка.
75
76            # вычисляем отклонение центра дороги от центра кадра
77            err = 0 - ((left + right) // 2 - SIZE[0] // 2)  # вычисляем ошибку
              ↪  положения автомобиля на дороге
78            if abs(right - left) < 100:
79                err = last_err
80            last_err = err
81
82        if STATE == CROSS:  # если мы проезжаем перекресток по прямой, то
83            if detect_road_begin(perspective):  # проверяем нет ли впереди линий
              ↪  разметки
84                STATE = GO  # если есть, значит мы проехали перекресток и
85                # нужно переключиться на алгоритм следования по разметке
86
87
88            # вычисляем положение полосы движения,
89            # в которую собираемся попасть после проезда перекрестка
90            # положение полосы движения совпадает с самыми темными стольцами пикселей
91            bin = np.flip(binary, axis=0)
92            hist = np.argmax(bin, axis=0)
93
94            # выбираем 30 столбцов, которые в большинстве своем больше среднего
95            pull_size = np.max(hist) / np.mean(hist)
96            if pull_size >= 1.6:  # 1.6
```

```
            pull_size = int(50 * pull_size)
        else:
            pull_size = 30

        # Вычисляем индексы самых темных столбцов
        ind_max_elem = np.argsort(hist)[-pull_size:]
        # Вычисляем отклонение самых темных столбцов от центра кадра
        err = (bin.shape[1] // 2) - int(np.mean(ind_max_elem))

    # преобразуем ошибку в угол поворота рулевых колес через ПД-регулятор
    angle = int(90 + KP * err + KD * (err - last_err))

    # проверка на превышение допустимых углов поворота колес
    if angle < 60:
        angle = 60
    elif angle > 120:
        angle = 120

    # Если предыдущее состояние не равно текущему, то
    if STATE != PREV_STATE:
        print("STATE:", STATE)
        PREV_STATE = STATE
        # изменяем скорость в соответствии с новым состоянием
        if STATE in (GO, CROSS):
            arduino.set_speed(CAR_SPEED)
        elif STATE == STOP:
            arduino.stop()

    arduino.set_angle(angle)  # устанавливаем угол поворота руля на Arduino

# Остановка автомобиля и разблокировка доступа к камере.
arduino.stop()
arduino.set_angle(90)
cap.release()
```

Взаимодействие с пешеходами и знаком дорожного движения «Пешеходный переход»

И пешеходов, и знак дорожного перехода можно детектировать и по площади объекта определить далеко объект или близко. Но как отличить пешехода на дороге от пешехода на обочине дороги? Можно выделить область на изображении и проверять попадает в нее пешеход или нет. Однако это сработает для дороги с постоянной кривизной. В случае изменения кривизны дорога и выделенная нами область перестанут совпадать.

При движении по дороге необходимо смещать область вместе с изменением кривизны дороги. Алгоритм движения по разметке рассчитывает угол поворота рулевых колес в зависимости от того, куда уходит дорога. Угол поворота рулевых колес связан с кривизной дороги и может служить для нас маркером того, насколько дорога поворачивает. Чем сильнее поворачивает дорога, тем сильнее нам нужно смещать область для поиска пешеходов. Не обязательно смещать всю область, можно смещать только ей часть.

В приведенном ниже примере для поиска пешеходов на проезжей части используется трапеция. Положение двух вершин трапеции рассчитывается каждый раз заново и зависит от угла поворота рулевых колес.

```python
import cv2

# shapely - библиотека для работы с сложными геометрическими фигурами
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

from arduino import Arduino  # класс из базового кода для общения с Arduino
from road_utils import *  # набор функций базового кода
import yolopy  # библиотека для инференса моделей Yolo на NPU

# Параметры трапеции - области обнаружения пешеходов
PD_UP = 0.35
PD_DOWN = 0.15
PD_H = 0.65
PD_H_INV = 1 - PD_H
X_OFFSET = 0
Y_OFFSET = 0
WIDTH_COEFF = 0

CAR_SPEED = 1430  # начальное значение скорости
CAR_SPEED_SLOW = 1440  # значение пониженной скорости

# Задаем константы состояний
GO = "GO"  # следуем по своей полосе движения
GO_SLOW = "GO_SLOW"  # движемся с пониженной скоростью
STOP = "STOP"  # беспилотный автомобиль остановился

STATE = GO  # задаем начальное состояние
PREV_STATE = None  # переменная для хранения предыдущего состояния

model_file = "yolov4-tiny_uint8.tmfile"  # задаем путь к файлу модели YOLO
# загружаем модель детектора из файла
model = yolopy.Model(model_file, use_uint8=True, use_timvx=True, cls_num=10)
# задаем якоря для модели YOLO
model.set_anchors([18, 33, 33, 48, 25, 71, 58, 76, 40, 113, 87, 140])  # задаем
↪  якорные точки для модели YOLO

cap = cv2.VideoCapture('/dev/video0')  # инициализируем камеру
# Инициализация обмена данными с Arduino через UART
arduino = Arduino.find_connected_arduino(baudrate=115200, timeout=10, debug=True)

while True:
    STATE = GO
    success, frame = cap.read()  # читаем кадр с камеры
    if not success:  # если не удалось прочитать кадр, выходим из цикла
        print("Can't open camera")
        break

    img = cv2.resize(frame, SIZE)  # приводим изображение к удобному для
    ↪  вычислений размеру
    binary = binarize2(img, THRESHOLD)  # бинаризуем изображение для поиска белых
    ↪  линий
    perspective = trans_perspective(binary, TRAP, RECT, SIZE)  # извлекаем область
    ↪  перед колесами
    left, right = centre_mass(perspective)  # получаем координаты левой и правой
    ↪  линии разметки

    # вычисляем отклонение центра дороги от центра кадра
    err = 0 - ((left + right) // 2 - SIZE[0] // 2)  # вычисляем ошибку положения
    ↪  автомобиля на дороге
```

```
55      if abs(right - left) < 100:
56          err = last_err
57      last_err = err
58
59      # преобразуем ошибку в угол поворота рулевых колес через ПД-регулятор
60      angle = int(90 + KP * err + KD * (err - last_err))
61
62      # обнаруживаем объекты на изображении с помощью модели YOLO
63      classes, scores, boxes = model.detect(img)
64
65      peds = []  # инициализация списка для хранения координат центров пешеходов
66      ped_areas = []  # инициализация списка для хранения площадей пешеходов
67      ped_sign_areas = []  # инициализация списка для хранения площадей знаков
        ↪  "Пешеходный переход"
68
69      # перебираем все обнаруженные объекты
70      for cls, score, box in zip(classes, scores, boxes):
71          if cls == 9:  # если обнаружен светофор, то
72              x, y, w, h = box  # сохраняем параметры ограничивающей рамки и
73              tl_detected = True  # выставляем соответствующий флаг
74              tl_frame = img[y:y + h, x:x + w]  # вырезаем светофор из изображения
75
76      # перебираем все обнаруженные объекты
77      for cls, score, box in zip(classes, scores, boxes):
78          if cls == 0:  # если обнаружен пешеход, то
79              x, y, w, h = box  # сохраняем параметры ограничивающей рамки и
80              xc = x + w // 2  # вычисляем координаты центра объекта
81              yc = y + h // 2
82              peds.append((xc, yc))  # добавляем центр пешехода в список
83              ped_areas.append(w * h)  # добавляем площадь пешехода в список
84
85          if cls == 4:  # если класс объекта - знак "Пешеходный переход"
86              x, y, w, h = box  # сохраняем параметры ограничивающей рамки и
87              ped_sign_areas.append(w * h)  # добавляем площадь пешехода в список
88
89      # проверяем площади пешеходов,
90      # если площадь больше определенного порога то считаем,
91      # что пешеход близко и нужно снизить скорость
92      for area in ped_areas:
93          if area > 1000:
94              STATE = GO_SLOW  # установка состояния "движение с ограниченной
                ↪  скоростью"
95
96      # Проверяем площадь знака "Дорожный переход", если она превышает порог, то
        ↪  знак близко и
97      # нужно снизить скорость
98      for area in ped_sign_areas:
99          if area > 800:
100             STATE = GO_SLOW # установка состояния "движение с ограниченной
                ↪  скоростью"
101
102     # трапеция для поиска пешеходов - область изменяющаяся вместе с кривизной
        ↪  дороги.
103     # трапеция соответствует проезжей части. Если в трапеции оказывается пешеход,
104     # Нужно немедленно остановиться, пока пешеход не покинет проезжую часть
105     # вычисляем смещение трапеции на основе угла поворота колес, зависящего от
        ↪  кривизны дороги
106     X_OFFSET = -int(abs(angle - 90) * 4.5)
107     Y_OFFSET = int(abs(angle - 90) * 1.2)
108     WIDTH_COEFF = int(abs(angle - 90)*2.3)
```

```
109
110
111     ped_h, ped_w = img.shape[:2]  # высота и ширина кадра
112
113     # создание списка вершин трапеции для обнаружения пешеходов
114     ped_zone_lst = [
115         [PD_DOWN * ped_w, ped_h],
116         [PD_UP * ped_w + X_OFFSET - WIDTH_COEFF, PD_H_INV * ped_h + Y_OFFSET],
117         [ped_w - PD_UP * ped_w + X_OFFSET + WIDTH_COEFF,
118                                                     PD_H_INV * ped_h + Y_OFFSET],
119         [ped_w - PD_DOWN * ped_w, ped_h]]
120     # создание многоугольника на основе списка вершин трапеции
121     ped_zone = Polygon(ped_zone_lst)
122
123     # проверка каждого пешехода на нахождение внутри трапеции
124     for xc, xy in peds:
125         ped_center = Point(xc, xy)  # создание точки, совпадающей с центром
            ↪   пешехода
126         if ped_zone.contains(ped_center):  # проверка вхождения точки в
            ↪   многоугольник (трапецию)
127             STATE = STOP  # останавливаемся, если пешеход в трапеции
128
129     # проверка на превышение допустимых углов поворота колес
130     if angle < 60:
131         angle = 60
132     elif angle > 120:
133         angle = 120
134
135     # Если предыдущее состояние не равно текущему, то
136     if STATE != PREV_STATE:
137         print("STATE:", STATE)
138         PREV_STATE = STATE
139         # изменяем скорость в соответствии с новым состоянием
140         if STATE == GO:
141             arduino.set_speed(CAR_SPEED)
142         elif STATE == GO_SLOW:
143             arduino.set_speed(CAR_SPEED_SLOW)
144         elif STATE == STOP:
145             arduino.stop()
146
147     arduino.set_angle(angle)  # устанавливаем угол поворота руля на Arduino
```

Подзадачи сортировочного хаба (СХ)

Перемещение указанного груза из зоны разгрузки в захват квадрокоптера

Самый надежный способ решения этой подзадачи обучить нейросетевой детектор для обнаружения грузов и классификатор для определения маркировки грузов. Однако этот способ самый трудозатратный, быстрее и проще написать алгоритм, использующий бинаризацию, контурный анализ и анализ бинарных масок. Условия освещения в сортировочном хабе такие, что цвет объектов практически не изменяется в течение дня.

В представленном ниже алгоритме сначала детектируются грузы. Поиск контуров грузов проводится на бинарной маске, где пиксели фона стали черными, а грузы и маркировки белыми. Изображение груза с маркировкой преобразуется в горизон-

тально ориентированный квадрат.

После этого на маркировке проводится поиск полосы, наиболее приближенной к краю изображения. Эта полоса та, с которой нужно начинать чтение маркировки — она всегда самая ближняя к краю изображения. Определив к какому именно краю изображения ближе всего первая полоса, можно понять, насколько градусов нужно повернуть изображение, чтобы маркировка читалась слева на право, а полосы были расположены вертикально. Когда маркировка расположена таким образом, зная центр одной полосы можно найти остальные смещением по горизонтали. В окрестности центров полос проводится анализ бинарных масок и вычисляется цвет каждой полосы, соответственно и символ, который полоса кодирует.

Все найденные на изображении грузы обводятся прямоугольником и около выводится их маркировка.

```
import cv2
import numpy as np

# Задаем координаты углов квадрата,
# в который будем преобразовывать прямоугольное изображение найденных грузов
sdr = np.float32([[0, 0],
                  [300, 0],
                  [0, 300],
                  [300, 300]])

font = cv2.FONT_HERSHEY_COMPLEX  # шрифт для надписей на изображении

cap = cv2.VideoCapture(0)  # Инициализация работы с камерой

while i < 20:  # цикл обработки первых 20 кадров
    ret, frame = cap.read()  # чтение кадра с камеры
    if not ret:
        break

    img = cv2.resize(frame, (480, 400))  # изменяем размер изображения
    img = img[:250, :]  # оставляем изображение только зоны разгрузки
    # переводим изображение в HSV формат
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # зона разгрузки имеет преимущественно черный цвет
    # бинаризуем изображение так, чтобы грузы и цветные полосы стали белыми,
    # а фон стал черным
    black_m = cv2.inRange(img, (55, 55, 55), (255, 255, 255))
    # бинаризуем HSV-изображение так, чтобы цветные полосы стали черными,
    # а фон и грузы белым
    binimg = cv2.inRange(hsv_img, (0, 0, 0), (255, 140, 255))
    # находим пересечение масок, на нем останутся только очертания грузов
    binimg = cv2.bitwise_and(binimg, black_m)
    # ищем на изображении контуры
    contours, _ = cv2.findContours(binimg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # сортируем контуры по убыванию площади
    contours = sorted(contours, key=cv2.contourArea, reverse=True)

    # Создаем списки для хранения координат центров уже обнаруженных грузов
    x_coordinates = []
    y_coordinates = []

    # Перебираем все обнаруженные контуры и проверяем,
    # соответствует ли их размер заданным параметрам для груза
    for j in range(len(contours)):
        (x, y, w, h) = cv2.boundingRect(contours[j])
```

```
46          #  Высота и ширина контура должны лежать в определенных пределах и
47          #  не сильно отличаться друг от друга, т.е. быть близким к квадрату
48          if 25 < w < 60 and 25 < h < 60 and abs(w - h) <= 7:
49              # Устанавливаем флаг, обнаружения груза
50              # Если контур соответствует уже детектированному грузу, то флаг = True
51              same = False
52              # сравниваем координаты "центра" контура с
53              # координатами "центров" уже обнаруженных грузов
54              for ii in range(len(x_coordinates)):
55                  if abs((x + w // 2) - x_coordinates[ii]) < 10 and /
56                      abs((y + h // 2) - y_coordinates[ii]) < 10:
57                      # если центры ближе 10 пикселей по вертикали и
58                      # ближе 10 пикселей по горизонтали, то
59                      # контур указывает на уже обнаруженный груз
60                      same = True   # выставляем соответствующий флаг
61                      break  # прекращаем перебор координат уже найденных грузов
62
63              # Если контур указывает на ранее не обнаруженный груз, то
64              #  распознаем его маркировку.
65              if not same:
66                  # сохраняем координаты центра груза
67                  x_coordinates.append(x + w // 2)
68                  y_coordinates.append(y + h // 2)
69
70                  # Находим минимальный прямоугольник, описывающий контур.
71                  #  Он может быть наклонен!
72                  rect = cv2.minAreaRect(contours[j])
73
74                  # получаем координаты верши прямоугольника
75                  # координаты преобразуются для передачи в функцию
76                  # вычисления матрицы преобразования
77                  box = cv2.boxPoints(rect)
78                  box = np.int0(box)
79                  box = list(box)
80                  box[2], box[3] = box[3], box[2]
81                  box = np.float32(box)
82                  # преобразуем прямоугольник в квадрат, для этого
83                  # вычисляем матрицу преобразования
84                  mm = cv2.getPerspectiveTransform(box, sdr)
85                  # применяем матрицу к изображению, на выходе получаем
86                  # горизонтально ориентированное изображение груза
87                  marker = cv2.warpPerspective(img, mm, (300, 300),
88                                                      flags=cv2.INTER_LINEAR)
89
90                  # Создаем несколько масок бинаризации для
91                  # каждого возможного цвета полос маркировки, кроме черного
92                  # Маска - бинаризованное изображение, где белые пиксели
93                  # соответствую конкретному цвету
94
95                  # для красных полос
96                  red_mask = cv2.inRange(marker, (0, 0, 140), (90, 110, 255))
97                  # для зеленых полос
98                  green_mask = cv2.inRange(marker, (50, 110, 0), (180, 255, 90))
99                  # для синих полос
100                 blue_mask = cv2.inRange(marker, (180, 90, 0), (255, 210, 100))
101                 # для белых полос
102                 white_mask = cv2.inRange(marker, (230, 230, 230), (255, 255, 255))
103
104                 # объединяем все маски, на результирующем изображении
105                 # белые пиксели соответствуют полосам маркировки
```

```
106                 mask = cv2.bitwise_or(green_mask, red_mask)
107                 mask = cv2.bitwise_or(blue_mask, mask)
108                 mask = cv2.bitwise_or(white_mask, mask)
109
110                 # на объединении масок ищем контуры
111                 contours1, _ = cv2.findContours(mask, cv2.RETR_TREE,
112                                                 cv2.CHAIN_APPROX_SIMPLE)
113                 # сортируем контуры по убыванию площади
114                 contours1 = sorted(contours1, key=cv2.contourArea, reverse=True)
115
116
117                 min_dist = 300  # переменная для хранения минимальной
118                                 # дистанции от центра полосы до края изображения
119
120                 # Перебираем все найденные контуры, подходящие под размер
121                 # полосы и вычисляем координаты центра самой
122                 # приближенной к краю изображения полосы
123
124                 # Эта полоса - кодирует первый символ маркировки,
125                 # чтение нужно начинать с нее
126                 for k in range(len(contours1)):
127                     # получаем координаты угла ограничивающего контур
128                     # прямоугольника, его ширину и высоту
129                     (x1, y1, w1, h1) = cv2.boundingRect(contours1[k])
130                     # контур полосы должен быть больше определенного размера
131                     if (w1 > 40 and h1 > 40):
132                         cx = x1 + w1 // 2   # вычисляем координаты центра полосы
133                         cy = y1 + h1 // 2   # маркировки по x и y
134                         # определяем минимальную дистанцию от центра полосы до
135                         # стороны изображения
136                         dist = min(cx, cy, 300 - cx, 300 - cy)
137
138                         # если полоса ближе всех к краю изображения, то
139                         if min_dist >= dist:
140                             min_dist = dist   # обновляем минимальную дистанцию,
141                             cx_min = cx      # сохраняем координаты центра
142                             cy_min = cy      # полосы по xи y
143
144
145                 # Поворачиваем изображение груза так, чтобы маркировка
146                 # читалась слева направо
147                 # Вместе с изображением груза поворачиваем все соответствующие
148                 # маски для полос разного цвета
149
150                 # Если расстояние от центра полосы до левого края изображения
151                 # меньше чем расстояние до остальных краев,
152                 if cx_min < cy_min and cx_min < 300 - cx_min /
153                                    and cx_min < 300 - cy_min:
154                     # то изображение не нужно поворачивать
155                     new_cx = cx_min  # сохраняем координаты центра первой полосы
156                     new_cy = cy_min
157                 # Если расстояние от центра полосы до верхнего края
158                 #  изображения меньше чем расстояние до остальных краев,
159                 elif cy_min < cx_min and cy_min < 300 - cx_min and /
160                                          cy_min < 300 - cy_min:
161                     # то поворачиваем изображение на 90 градусов
162                     # против часовой стрелки
163                     marker = cv2.rotate(marker, cv2.ROTATE_90_COUNTERCLOCKWISE)
164                     red_mask = cv2.rotate(red_mask,
165                                          cv2.ROTATE_90_COUNTERCLOCKWISE)
```

```
166                 blue_mask = cv2.rotate(blue_mask,
167                                         cv2.ROTATE_90_COUNTERCLOCKWISE)
168                 green_mask = cv2.rotate(green_mask,
169                                         cv2.ROTATE_90_COUNTERCLOCKWISE)
170                 # пересчитываем координаты центра первой полосы в
171                 # системе координат нового изображения
172                 new_cy = 300 - cx_min
173                 new_cx = cy_min
174             # Если расстояние от центра полосы до
175             # правого края изображения меньше
176             # чем расстояние до остальных краев,
177             elif 300 - cx_min < cx_min and 300 - cx_min < cy_min and /
178                                             300 - cx_min < 300 - cy_min:
179                 # то поворачиваем изображение на 180 градусов
180                 marker = cv2.rotate(marker, cv2.ROTATE_180)
181                 red_mask = cv2.rotate(red_mask, cv2.ROTATE_180)
182                 blue_mask = cv2.rotate(blue_mask, cv2.ROTATE_180)
183                 green_mask = cv2.rotate(green_mask, cv2.ROTATE_180)
184                 # пересчитываем координаты центра первой полосы
185                 # в системе координат нового изображения
186                 new_cx = 300 - cx_min
187                 new_cy = 300 - cy_min
188             else:
189                 # Если расстояние от центра полосы до
190                 # нижнего края изображения меньше
191                 # чем расстояние до остальных краев,
192                 # то поворачиваем изображение на 90 градусов
193                 # по часовой стрелке
194                 marker = cv2.rotate(marker, cv2.ROTATE_90_CLOCKWISE)
195                 red_mask = cv2.rotate(red_mask, cv2.ROTATE_90_CLOCKWISE)
196                 blue_mask = cv2.rotate(blue_mask, cv2.ROTATE_90_CLOCKWISE)
197                 green_mask = cv2.rotate(green_mask, cv2.ROTATE_90_CLOCKWISE)
198                 # пересчитываем координаты центра первой полосы
199                 # в системе координат нового изображения
200                 new_cx = 300 - cy_min
201                 new_cy = cx_min
202
203             # Теперь изображение ориентировано так, что можно читать
204             # маркировку слева направо
205             # Мы знаем координаты центра первой полосы, координаты
206             # центров остальных полос находим смещением
207             # по оси x на 50 пикселей
208             # Для каждой полосы в окрестности центра вычисляется
209             # число белых пикселей на маске конкретного цвета,
210             # по цвету маски где это число превышает некоторое
211             # значение определяем цвет полосы
212             # и кодируемый ей символ
213             name_mark = ""  # переменная для расшифровки маркировки
214             for i in range(3):  # Просматриваем окрестности центров трех полос
215                 shift = i * 50  # смещение центра по оси x
216                                 # в зависимости от номера полосы
217
218                 # Если на красной маске по координатам центра полосы
219                 # есть белые пиксели, то
220                 if np.sum(red_mask[new_cy - 5:new_cy + 5,
221                             shift + new_cx - 5:shift + new_cx + 5]) > 255 *
                            ↪  50:
222                     name_mark += "R"  # полоса красная и кодирует символ "R",
223                                       # добавляем его к расшифровке
224                 # Если на зеленой маске по координатам центра полосы
```

```
225                         # есть белые пиксели, то
226                         elif np.sum(green_mask[new_cy - 5:new_cy + 5,
227                                         shift + new_cx - 5:shift + new_cx + 5]) > 50:
228                             name_mark += "G"  # полоса зеленая и кодирует символ "G",
229                                               # добавляем его к расшифровке
230                         # Если на синей маске по координатам центра полосы
231                         # есть белые пиксели, то
232                         elif np.sum(blue_mask[new_cy - 5:new_cy + 5,
233                                         shift + new_cx - 5:shift + new_cx + 5]) > 50:
234                             name_mark += "B"  # полоса синяя и кодирует символ "B",
235                                               # добавляем его к расшифровке
236                         else:  # Если на масках не обнаружилось белых пикселей,
237                             name_mark += "0"  #  значит полоса белая или черная -
238                                               #  кодирует "0",
239                                               #  добавляем его к расшифровке
240
241                     # Отмечаем на изображении обнаруженный груз прямоугольником
242                     cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
243                     # добавляем над прямоугольником подпись соответствующую
244                     # расшифровке маркировки груза
245                     cv2.putText(img, name_mark, (x + 1, y - 4),
246                                                    font, 0.5, (0, 0, 255))
247
248         key = cv2.waitKey(0)  # блокирующее чтение нажатой пользователем клавиши
249         if key == ord('q'):  # Если нажата клавиша "q"
250             cv2.destroyAllWindows()  # Уничтожаем все открытые программой окна
251             break  # Выходим из цикла обработки кадров и завершаем работу программы
```

Перемещение указанного груза из зоны разгрузки в захват квадрокоптера и распределение оставшихся грузов по накопителям

Если первая подзадача решена так, как показано в предыдущем примере, то для решения второй задачи остается добавить совсем немного программного кода.

Как только мы детектировали груз и распознали его маркировку, можно сразу определить под какое из условий задачи она попадает.

Вместо того, чтобы выводить надпись на изображение.

```
if key == ord('q'):  # Если нажата клавиша "q"
    cv2.destroyAllWindows()  # Уничтожаем все открытые программой окна
    break  # Выходим из цикла обработки кадров и завершаем работу программы
```

Добавляем проверку на соответствие одному из условий.

```
# Отмечаем на изображении обнаруженный груз прямоугольником
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
# если маркировка составлена из двух символов, значи один из них повторяется
if len(set(name_mark)) == 2:
   # добавляем над прямоугольником подпись "1_маркировка"
   cv2.putText(img, "1_" + name_mark, (x + 1, y - 4), font, 0.5, (0, 0, 255))

# если маркировка начинается с "0", то
elif name_mark.startswith("0"):
   # добавляем над прямоугольником подпись "2_маркировка"
   cv2.putText(img, "2_" + name_mark, (x + 1, y - 4), font, 0.5, (0, 0, 255))
```

```
# если маркировка не содержит "G", то
elif "G" not in name_mark:
    # добавляем над прямоугольником подпись "3_маркировка"
    cv2.putText(img, "3_" + name_mark, (x + 1, y - 4), font, 0.5, (0, 0, 255))

else:  # груз не подошел под предыдущие условия
    # добавляем над прямоугольником подпись "LIFT_маркировка"
    cv2.putText(img, "LIFT_" + name_mark, (x + 1, y - 4), font, 0.5, (0, 0, 255))
```

Подзадачи квадрокоптера

Выполнить захват груза и перелет с грузом на здание № 1

Квадрокоптер определяет свои координаты по ультразвуковой системе позиционирования, для участников взаимодействие с ней аналогично работе с GPS. Участники знают трехмерные координаты квадрокоптера в каждый момент времени, однако координаты определяются с погрешностью ± 20 см. Участники управляют перемещением квадрокоптера, задавая координаты точки, в которую квадрокоптер должен прилететь.

Участники заранее не знают координаты точек, составляющих маршруты перелета. Определить эти координаты можно множеством способов. Наиболее технически верный — пронести квадрокоптер по полигону и в каждой точке маршрутов собрать несколько значений трехмерных координат. Проанализировав статистические данные, можно вывести наиболее вероятные значения координат точек, в программе необходимо указывать именно их.

Взаимодействие между бортовым компьютером и полетным контроллером квадрокоптера реализовано через обмен управляющими командами и событиями. Наиболее сложная часть написания программы автономного полета, заключается в организации функции обработчика событий, отслеживания последовательности наступления событий и отправки управляющих команд. Если обработчик событий реализован не верно, то поведение квадрокоптера непредсказуемо.

Дополнительная сложность — зажечь светодиодную подсветку тем цветом, который соответствует получателю груза. Цвет должен быть вычислен исходя из входных данных, получаемых участниками перед начало полета: двух пиктограмм и их взаимного расположения.

Ниже приведен код, выполняющий данную подзадачу.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import rospy
5  from gs_flight import FlightController, CallbackEvent
6  from gs_board import BoardManager
7  from gs_module import CargoController
8
9  rospy.init_node("task_one")  # инициализируем ноду
10
11 # массив с координатами точек, составляющих маршрут
12 coordinates = [
13     [3.83, 3.28, 1.5],
14     [3.1, 3.28, 1.5],
```

```
    [3.1, 3.9, 1.5],
    [2.2, 3.9, 1.5]
]

run = True  # переменная, отвечающая за работу программы
position_number = 0  # счетчик пройденных точек

# список возможных пиктограмм
pictogram_names = ["eye", "brain", "map", "battery"]

# задаем цвета, соответсвующие подразделениям получателям груза
# индекс кортежа в списке соответствует намеру получателя
# Т.е первый элемент соответвует первому элементу в списке recipient_names
colors = [
    (0.0, 0.0, 255.0),
    (0.0, 255.0, 0.0),
    (255.0, 0.0, 255.0),
    (255.0, 0.0, 0.0)
]

# задаем таблицу соответствия пиктограмм и получателя груза
# -1 - комбинации не существует
# остальные числа - номера получателя груза
pictogram2recipient = [
    [3, 3, 2, -1],
    [3, 1, 1, -1],
    [2, 1, 2, 4],
    [-1, -1, 4, 4]
]

# ввод первой пиктограммы
image1 = ''
# вводим название пиктограммы, пока не будет совпадения с возможными именами
↪  пиктограмм
while pictogram_names.count(image1) == 0:
    print("Enter first image name:")
    image1 = input()  # ввод названия первой пиктограммы

# ввод второй пиктограммы
image2 = ''
# вводим название пиктограммы, пока не будет совпадения с возможными именами
↪  пиктограмм
while pictogram_names.count(image2) == 0:
    print("Enter second image name:")
    image2 = input()  # ввод названия второй пиктограммы

# ввод взаимного расположения пиктограмм
direction = ''
while (direction != 'v') and (direction != 'h'): # защита от неверного ввода
    print("Enter direction:")
    direction = input()  # ввод направления ('v' - вертикаль, 'h' - горизонталь)

current_recipient = -1
image1_index = pictogram_names.index(image1)  # получаем индекс первой пиктограммы
image2_index = pictogram_names.index(image2)  # получаем индекс второй пиктограммы

# в зависимости от заданного направления находим номер текущего получателя груза
if direction == 'h':
    current_recipient = pictogram2recipient[image1_index][image2_index]
elif direction == 'v':
```

```
73      current_recipient = pictogram2recipient[image2_index][image1_index]
74
75  # если заданная комбинация существует, то
76  # конвертируем номер получателя в индекс списка цветов, соответствующего этому
    ↪  получателю
77  # если заданная комбинация не существует, выходим из программы,
78  # поскольку оператор ввел несуществующую комбинацию
79  if current_recipient != -1:
80      current_recipient -= 1
81  else:
82      exit()
83
84  current_color = colors[current_recipient]  # запоминаем текущий цвет индикации
85
86  # функция, включающая мигание индикации
87  # входные данные: цвет и интервал мигания
88  def indication(color, time):
89      global cargo
90      # мигаем несколько раз
91      # сколько раз мигаем, не важно, главное, чтобы интервал совпадал с условием
        ↪  задачи
92      for _ in range(5):
93          cargo.changeAllColor(*color)  # устанавливаем цвет светодиодов
94          rospy.sleep(time)  # включаем светодиоды на time секунд
95          cargo.changeAllColor()  # выключаем светодиоды
96          rospy.sleep(time)  # выключаем светодиоды на time секунд
97
98  # функция обработки событий Автопилота
99  def callback(event):
100     global ap
101     global run
102     global cargo
103     global coordinates
104     global position_number
105     global current_color
106
107     event = event.data
108     # блок обработки события запуска двигателя
109     if event == CallbackEvent.ENGINES_STARTED:
110         ap.takeoff()  # отдаем команду взлета
111
112     # блок обработки события завершения взлета
113     elif event == CallbackEvent.TAKEOFF_COMPLETE:
114         position_number = 0  # обнуляем счетчик точек в маршруте
115         indication(current_color, 4)  # производим индикацию
116         # отдаем команду полета в точку
117         ap.goToLocalPoint(coordinates[position_number][0],
118                           coordinates[position_number][1],
119                           coordinates[position_number][2])
120
121     # блок обработки события достижения точки
122     elif event == CallbackEvent.POINT_REACHED:
123         position_number += 1 # увеличиваем значение счетчика точек
124         #сравниваем число точек в маршруте с номером текущей точки
125         if position_number < len(coordinates):
126             # отдаем команду полета в точку
127             ap.goToLocalPoint(coordinates[position_number][0],
128                               coordinates[position_number][1],
129                               coordinates[position_number][2])
130         else:
```

```
131                 ap.landing() # отдаем команду посадки
132
133         # блок обработки события приземления
134         elif event == CallbackEvent.COPTER_LANDED:
135             run = False # прекращаем программу
136
137 board = BoardManager() # создаем объект бортового менеджера
138 ap = FlightController(callback)  # создаем объект управления полетом
139 cargo = CargoController()  # создаем объект управления модулем магнитного захвата
140
141 once = False  # переменная, отвечающая за первое вхождение в начало программы
142
143 # цикл работает, пока ROS включен или пока переменная run равна True
144 # цикл необходим для того, чтобы можно было получать события от автопилота
145 while not rospy.is_shutdown() and run:
146     if board.runStatus() and not once:  # проверка подключения RPi к Пионеру
147         cargo.on()  # включаем магнитный захват
148         ap.preflight()  # отдаем команду выполнения предстартовой подготовки
149         once = True
150     pass
```

Верифицировать графическую метку, соответствующую подразделению-получателю груза

Для успешного выполнения задачи необходимо было модернизировать программу из прошлой задачи так, чтобы при окончании полета к зданию №1 квадрокоптер взлетел на высоту, достаточную для верификации всех пиктограмм на крыше. Для этого в список coordinates добавим еще один элемент, который будет совпадать с координатами последней точки из предыдущей подзадачи, однако координата Z будет равна 2 метра. 2 метра — высота относительно пола, достаточная для верификации всех пиктограмм на крыше, данная величина может быть и больше. Таким образом, измененный список будет выглядеть так.

```
# массив с координатами точек, составляющих маршрут
coordinates = [
    [3.83, 3.28, 1.5],
    [3.1, 3.28, 1.5],
    [3.1, 3.9, 1.5],
    [2.2, 3.9, 1.5],
    [2.2, 3.9, 2.0]
]
```

После чего необходимо произвести верификацию пиктограмм и из полученных данных выбрать комбинацию пиктограмм, которая соответствует заданной паре. Для детектирования пиктограмм необходимо было использовать нейросетевой детектор. Одним из способов решения задачи является использование классификатора `Yolo-tiny`. Основная сложность задачи в том, что пиктограмма может иметь любой из 4-х цветов подложки. Чтобы нейросетевой детектор правильно обучился, необходимо бинаризовать изображения для обучения. Например, можно было бинаризовать изображение по порогу. Ниже приведена функция бинаризации изображения:

```
import cv2

# функция бинаризации изображения, возвращает черно-белое изображение
```

```python
def binarization(frame):
    # переводим изображение в градации серого и бинаризуем
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, frame = cv2.threshold(frame, 90, 255, 0)
    return frame
```

Применяя эту функцию к изображениям, полученным с камеры, необходимо было собрать датасет для обучения, а также обрабатывать все кадры перед детектированием.

Ниже приведена функция загрузки модели.

```python
# загрузка и настройка нейросетевого детектора
def load_model():
    # загружаем из файлов весовые коэффициенты и конфигурацию сети
    net = cv2.dnn.readNet("yolov4-tiny.weights", "yolov4-tiny.cfg")
    model = cv2.dnn_DetectionModel(net) # создаем объект, связанный с
    ↪ загруженными нами данными
    # указываем размер обрабатываемого сетью изображения и необходимость
    ↪ нормировать значения пикселей на 255
    model.setInputParams(size=(640, 480), scale=1 / 255)
    return model
```

После детектирования моделью всех пиктограмм необходимо восстановить пространственное расположение пиктограмм. Если модель обучена достаточно хорошо, то на снимке крыши будет найдено ровно 16 пиктограмм. Самый простой способ восстановить расположение пиктограмм — это отсортировать все центры по возрастанию сначала по Y координате, потом X координате, после чего преобразовать одномерный массив в двумерный размером 4 на 4. Одновременно с этим необходимо восстановить пространственное расположение индексов классов в такой же двумерный массив. Ниже приведена функция детектирования пиктограмм.

```python
import numpy as np

# порог уверенности детектора в обнаружении объекта
CONFIDENCE_THRESHOLD = 0.5
# IoU для подавления множественного детектирования одного и тоже объекта
NMS_THRESHOLD = 0.4

# функция, обнаруживающая на изображении пиктограммы
def detect_pictogramms(model, frame):
    # Получаем результаты детектирования от сети
    classes, _, boxes = model.detect(frame, CONFIDENCE_THRESHOLD, NMS_THRESHOLD)

    # нейронная сеть выдает двумерный массив classes,
    # последнее измерение - индексы классов по возрастанию вероятности,
    # самый первый элемент - объект максимально вероятного класса
    classes = [cl[0] for cl in classes] # используем только его

    # создаем массив центров
    # для удобства обращения по индексам сначала добавляем Y, потом X
    centers = []
    for box in boxes:
```

```python
        start_x, start_y, end_x, end_y = box
        center_x = (end_x - start_x) // 2 + start_x
        center_y = (end_y - start_y) // 2 + start_y
        centers.append((center_y, center_x))

    # сортируем массив центров по Y
    # свою сортровку необходимо написать, чтобы одновременно сортировались и
    ↪  индексы классов
    for i in range(len(centers) - 1):
        for j in range(i + 1, len(centers)):
            if centers[i][0] >= centers[j][0]:
                centers[i], centers[j] = centers[j], centers[i]
                classes[i], classes[j] = classes[j], classes[i]

    # сортируем массив центров по X
    for i in range(len(centers) - 1):
        for j in range(i + 1, len(centers)):
            if centers[i][0] >= centers[j][0]:
                if centers[i][1] > centers[j][1]:
                    centers[i], centers[j] = centers[j], centers[i]
                    classes[i], classes[j] = classes[j], classes[i]
    # преобразуем список координат центров в трехмерный массив
    # первое измерение - строки координат
    # второе измеренеи - столбцы координат
    # третье измерение - сами координаты
    centers = np.asarray(centers).reshape(4, 4, 2)

    # преобразование индексов классов в двухмерный массив
    classes = np.asarray(classes).reshape(4, 4)

    # возвращаем центры классов и названия соответствующих классов
    return centers, classes
```

Для работы нейронной сети необходимо получить изображение с камеры через ROS. Для этого следует получить сообщения от топика `/pioneer_max_camera/image_raw` и с помощью `CvBridge` преобразовать сообщение ROS типа `sensor_msgs/Image` в изображение `OpenCv`. Ниже приведена функция получения и преобразования кадра.

```python
from cv_bridge import CvBridge
from sensor_msgs.msg import Image

bridge = CvBridge()

# функция чтения кадра с камеры квадрокоптера
def get_img():
    global bridge
    # синхронное получение изображения из ROS
    data = rospy.wait_for_message("/pioneer_max_camera/image_raw/", Image)
    # конвертация кадра из ROS в Cv2-формат
    frame = bridge.imgmsg_to_cv2(data, 'bgr8')
    return frame
```

Сообщение о детектировании необходимо было отправить оператору. Для этого нужно создать издатель топика `/answer` типа `std_msgs/String`. После детектиро-

вания необходимо через созданный издатель отправить строку. Ниже приведен код инициализации издателя и функция отправки сообщения в нужном формате.

```
from rospy import Publisher
from std_msgs.msg import String
to_operator = Publisher('/answer', String, queue_size=10)

# функция отправления сообщения оператору
def send_msg(image1, image2, recipient):
    global to_operator
    to_operator.publish(f"{image1}, {image2}, {recipient}")
```

Для выполнения второй подзадачи необходимо модифицировать функцию `callback`, так чтобы после окончания маршрута происходило детектирование всех меток на крыше, после чего происходила проверка полученных от нейронной сети данных и отправка соответствующего сообщения оператору. С учетом всех функций функция обработки событий Автопилота принимает вид.

```
1  model = load_model()
2  recipient_names = [ # список возможных подразделений получателей
3      "1 - Communication systems",
4      "2 - Geoinformation systems",
5      "3 - Computer vision",
6      "4 - New energy sources"
7  ]
8
9  task2 = False # флаг, выполнена ли 2 подзадача или нет
10
11 # функция обработки событий Автопилота
12 def callback(event):
13     global ap
14     global run
15     global cargo
16     global coordinates
17     global position_number
18     global current_color
19     global direction
20     global recipient_names
21     global image1_index
22     global image2_index
23     global image1
24     global image2
25     global model
26     global current_recipient
27     global task2
28
29     event = event.data
30     # блок обработки события запуска двигателя
31     if event == CallbackEvent.ENGINES_STARTED:
32         ap.takeoff()  # отдаем команду взлета
33
34     # блок обработки события завершения взлета
35     elif event == CallbackEvent.TAKEOFF_COMPLETE:
36         position_number = 0  # обнуляем счетчик точек в маршруте
37         indication(current_color, 4)  # включаем индикацию
38         # отдаем команду полета в точку
39         ap.goToLocalPoint(coordinates[position_number][0],
40                           coordinates[position_number][1],
```

```
41                              coordinates[position_number][2])
42
43      # блок обработки события достижения точки
44      elif event == CallbackEvent.POINT_REACHED:
45          position_number += 1 # увеличиваем значение счетчика точек
46          #сравниваем число точек в маршруте с номером текущей точки
47          if position_number < len(coordinates):
48              # отдаем команду полета в точку
49              ap.goToLocalPoint(coordinates[position_number][0],
50                                coordinates[position_number][1],
51                                coordinates[position_number][2])
52          else:
53              image = get_img()  # получаем изображение с камеры
54              image = binarization(image)  # бинаризуем изображение
55              # обнаруживаем пиктограммы на изображении
56              centers, classes = detect_pictogramms(model, image)
57
58              send = False  #флаг пройденной проверки
59              # выполняем проверку стартовых условий,
60              # если нужная пара пиктограмм найдена - проверка считается успешной
61              for i in range(0, 3):
62                  if send:
63                      break
64                  for j in range(0, 4):
65                      if direction == 'v':
66                          send = (classes[i][j] == image1_index) and\
67                                 (classes[i + 1][j] == image2_index)
68                      elif direction == 'h':
69                          send = (classes[j][i] == image1_index) and\
70                                 (classes[j][i + 1] == image2_index)
71                      if send:
72                          break
73
74              if send:  # если проверка пройдена, выводим сообщение оператору
75                  send_msg(image1, image2, recipient_names[current_recipient])
76
77              # зажигаем белые сигнальные огни с интервалом мигания 1 секунда
78              indication((255.0, 255.0, 255.0), 1)
79              rospy.sleep(10) # выполняем зависание на 10 секунд
80              cargo.changeAllColor() # выключаем светодиоды
81
82      # блок обработки события приземления
83      elif event == CallbackEvent.COPTER_LANDED:
84          run = False # прекращаем программу
```

Произвести доставку груза подразделению-получателю

Доставку груза необходимо осуществить так, чтобы груз оказался лежащим одновременно на обеих пиктограммах, составляющих логотип получателя груза. Для решения задачи необходимо вычислить точку между центрами двух пиктограмм и выполнить центрирование квадрокоптера на эту точку. Для успешного центрирования необходимо рассчитать разницу между координатами центра камеры и точкой сброса груза, перевести ее в метры, чтобы на полученное расстояние выполнить смещение. После чего необходимо уменьшить высоту квадрокоптера, чтобы груз точнее упал на цель. Функция расчета координат смещения относительно заданной точки приведена ниже.

```
from gs_navigation import NavigationManager
from math import tan, radians

def centering(x, y, height): # на вход получаем координаты точки центрирования и
↪   высоту здания
    horizontal_view = 62.2 # горизонтальный угол обзора в градусах PiCamera V2,
    ↪   установленной на коптере
    horizontal_view = radians(horizontal_view) # переводим градусы в радианы

    copter_x, copter_y, copter_z = NavigationManager().lps.position() # получаем
    ↪   текущие координаты коптера
    image_horizontal_size = 640 # горизонтальный размер изображения в пикселях
    copter_height = copter_z - height # высота над зданием
    # находим тангенс половины угла обзора, затем из определения тангенса в
    ↪   прямоугольном треугольнике находим расстояние в метрах от центра камеры
    ↪   до края и делим на кол-во пикселей

    pixel_in_meter = tan(horizontal_view / 2) * copter_height /
    ↪   (image_horizontal_size / 2)
    delta_x = (x - (650 / 2 - 1)) * pixel_in_meter # разница между координатами
    ↪   X центра изображения и заданной точкой в метрах
    delta_y = (y - (480 /2 - 1)) * pixel_in_meter # разница между координатами Y
    ↪   центра изображения и заданной точкой в метрах

    # поскольку начало координат коптера и координаты отсчета изображения
    ↪   инвертированы относительно друг друга
    # прибавляем координаты X к Y, а Y к X
    copter_x += delta_y
    copter_y += delta_x
    copter_x -= 0.05 # смещаем коптер на расстояние от центра камеры до центра
    ↪   магнита (расстояние приблизительное)
    return copter_x, copter_y
```

Для выполнения подзадачи 3 необходимо модифицировать функцию обработки событий автопилота из подзадачи 2, добавив вычисление точки сброса груза, выполнение центрирования и сброса.

```
1  task2 = False # флаг, выполнена ли 2 подзадача или нет
2
3  # функция обработки событий Автопилота
4  def callback(event):
5      global ap
6      global run
7      global cargo
8      global coordinates
9      global position_number
10     global current_color
11     global direction
12     global recipient_names
13     global image1_index
14     global image2_index
15     global image1
16     global image2
17     global model
18     global current_recipient
19     global task2
```

```
20
21      event = event.data
22      # блок обработки события запуска двигателя
23      if event == CallbackEvent.ENGINES_STARTED:
24          ap.takeoff()  # отдаем команду взлета
25
26      # блок обработки события завершения взлета
27      elif event == CallbackEvent.TAKEOFF_COMPLETE:
28          position_number = 0  # обнуляем счетчик точек в маршруте
29          indication(current_color, 4)  # включаем индикацию
30          # отдаем команду полета в точку
31          ap.goToLocalPoint(coordinates[position_number][0],
32                            coordinates[position_number][1],
33                            coordinates[position_number][2])
34
35      # блок обработки события достижения точки
36      elif event == CallbackEvent.POINT_REACHED:
37          position_number += 1 # увеличиваем значение счетчика точек
38          #сравниваем число точек в маршруте с номером текущей точки
39          if position_number < len(coordinates):
40              # отдаем команду полета в точку
41              ap.goToLocalPoint(coordinates[position_number][0],
42                                coordinates[position_number][1],
43                                coordinates[position_number][2])
44          else:
45              # если задание 2 выполнено, то можно сбрасывать груз
46              if not task2:
47                  # объявляем переменные центров пиктаграмм
48                  pictogramm1_centre = ()
49                  pictogramm2_centre = ()
50
51                  image = get_img()  # получаем изображение с камеры
52                  image = binarization(image)  # бинаризуем изображение
53                  # обнаруживаем пикстограммы на изображении
54                  centers, classes = detect_pictogramms(model, image)
55
56                  send = False  # флаг пройденной проверки
57                  # выполняем проверку стартовых условий,
58                  # если нужная пара пиктограмм найдена - проверка считается
                    ↪  успешной
59                  for i in range(0, 3):
60                      if send:
61                          break
62                      for j in range(0, 4):
63                          if direction == 'v':
64                              send = (classes[i][j] == image1_index) and\
65                                     (classes[i + 1][j] == image2_index)
66                              # запоминаем центры пиктограмм
67                              pictogramm1_centre = centers[i][j]
68                              pictogramm2_centre = centers[i + 1][j]
69                          elif direction == 'h':
70                              send = (classes[j][i] == image1_index) and\
71                                     (classes[j][i + 1] == image2_index)
72                              # запоминаем центры пиктограмм
73                              pictogramm1_centre = centers[j][i]
74                              pictogramm2_centre = centers[j][i + 1]
75                          if send:
76                              break
77
78                  if send: # если проверка пройдена, выводим сообщение оператору
```

```
79                          send_msg(image1, image2, recipient_names[current_recipient])
80                      else:
81                          # если проверка не пройдена, останавливаем программу для
                            ↪  перезапуска на следующей попытке
82                          run = False
83                          return
84
85                      # зажигаем белые сигнальные огни с интервалом мигания 1 секунда
86                      indication((255.0, 255.0, 255.0), 1)
87                      rospy.sleep(10)  # выполняем зависание на 10 секунд
88                      cargo.changeAllColor()  # выключаем светодиоды
89
90                      task2 = True  # вторая подзадача выполнена
91
92                      # высчитываем координаты точки сброса груза в пикселях
93                      distination_centre = (
94                          (pictogramm1_centre[1] + pictogramm2_centre[1]) // 2,  # X
                            ↪  координата центра
95                          (pictogramm1_centre[0] + pictogramm2_centre[0]) // 2  # Y
                            ↪  координата центра
96                      )
97
98                      # получаем новые координаты, 0.7 метров - высота 1 здания
99                      new_x, new_y = centering(*distination_centre, 0.7)
100                     ap.goToLocalPoint(new_x, new_y, 0.85)  # выполняем перелет в новую
                        ↪  точку
101                 else:
102                     cargo.off()  # выключаем магнитный захват
103                     ap.landing()  # отдаем команду посадки
104         elif event == CallbackEvent.COPTER_LANDED: # блок обработки события
            ↪  приземления
105             run = False #  завершаем работу программы
```

Произвести детектирование груза на крыше здания № 2

Для выполнения задания достаточно было сделать фотографию каждой площадки без груза в определенной координате. Далее необходимо было бинаризовать заранее подготовленное изображение и полученное с камеры во время попытки. После чего вычесть одно изображение из другого и посчитать количество белых пикселей. Если количество таких пикселей больше заданного порога, то груз находится на площадке. Ниже приведена функция чтения заданного изображения, а также функция поиска груза на текущем кадре.

```
# Площадки, с которых нужно забирать груз, имеют разную текстуру.
# Изображения всех площадок сохранены в формате .jpg,
# Функция читает и бинаризует изображение конкретной площадки
def get_area_image(number):
    image = cv2.imread(f'{number}.jpg')
    image = binarization(image)
    return image

# функция определения наличия груза на площадке
# сравнивает изображение площадки без груза и
# изображение площадки с камеры квадрокоптера
def is_cargo(frame, old_frame):
    # вычисляем разницу между изображениями
```

```
    difference = frame - old_frame
    # рассчитываем, сколько всего пикселей в изображении
    pixels_count = difference.shape[0] * difference.shape[1]
    # находим число белых пикселей на изображении
    white = np.count_nonzero(difference == 255)
    # если отношение белых пикселей к черным больше 10 процентов,
    # то груз находится на площадке - возвращаем True
    return white / pixels_count * 100 > 10
```

По условию подзадачи необходимо провести индикацию попеременно мигающим красным и синим. Ниже приведена функция мигания двумя цветами.

```
# функция активации двухцветной мигающей индикации
def indication_two(color1, color2, time):
    global cargo
    # мигаем несколько раз
    # сколько раз мигаем, не важно, главное, чтобы интервал совпадал с условием
    for i in range(10):
        # если номер итерации четный, мигаем первым цветом, если нет - вторым
        if i % 2 == 0:
            cargo.changeAllColor(*color1)
        else:
            cargo.changeAllColor(*color2)
        rospy.sleep(time) # включаем светодиоды на time секунд
        cargo.changeAllColor() # выключаем светодиоды
        rospy.sleep(time) # выключаем светодиоды на time секунд
```

Для успешного выполнения подзадачи надо модифицировать функцию обработки событий автопилота. Также необходимо внутри нее учитывать, какая подзадача выполняется, чтобы программа смогла лететь по одному из выбранных маршрутов. Переименуем список точек маршрута № 1 из `coordinates` в `coordinates1` и заведем список точек второго маршрута `coordinates2`. Тогда с учетом всех исправлений и нововведений функция `callback` принимает следующий вид.

```
1  # точки, составляющие маршрут №2
2  coordinates2 = [
3      [3.1, 3.9, 1.5],
4      [3.1, 1.64, 1.5],
5      [1.82, 1.64, 1.2]
6  ]
7
8  area_number = 0
9
10 task1 = False # флаг, выполнена ли 1 подзадача или нет
11 task2 = False # флаг, выполнена ли 2 подзадача или нет
12 task3 = False # флаг, выполнена ли 3 подзадача или нет
13 task4 = False # флаг, выполнена ли 4 подзадача или нет
14
15 def callback(event): # функция обработки событий Автопилота
16     global ap
17     global run
18     global cargo
19     global coordinates1
20     global coordinates2
21     global position_number
22     global current_color
```

```
23      global direction
24      global recipient_names
25      global image1_index
26      global image2_index
27      global image1
28      global image2
29      global model
30      global current_recipient
31      global task1
32      global task2
33      global task3
34      global task4
35      global area_number
36      global to_operator
37
38      event = event.data
39      # блок обработки события запуска двигателя
40      if event == CallbackEvent.ENGINES_STARTED:
41          ap.takeoff()  # отдаем команду взлета
42
43      # блок обработки события завершения взлета
44      elif event == CallbackEvent.TAKEOFF_COMPLETE:
45          position_number = 0  # обнуляем счетчик точек в маршруте
46          indication(current_color, 4)  # включаем индикацию
47          # отдаем команду полета в точку
48          ap.goToLocalPoint(coordinates1[position_number][0],
49                            coordinates1[position_number][1],
50                            coordinates1[position_number][2])
51
52      # блок обработки события достижения точки
53      elif event == CallbackEvent.POINT_REACHED:
54          if not task1:  # если задача не выполнена
55              position_number += 1  # наращиваем счетчик точек
56              # сравниваем часло точек в маршруте с номером текущей точки
57              if position_number < len(coordinates1):
58                  # отдаем команду полета в точку
59                  ap.goToLocalPoint(coordinates1[position_number][0],
60                                    coordinates1[position_number][1],
61                                    coordinates1[position_number][2])
62              else:
63                  task1 = True # подзадача 1 выполнена
64                  ap.goToLocalPoint(1.8, 3.9, 1.5)  # снижаемся для выполнения
                    ↪  подзадачи 2
65          elif not task2:
66              # объявляем переменные центров пиктограмм
67              pictogramm1_centre = ()
68              pictogramm2_centre = ()
69
70              image = get_img()  # получаем изображение с камеры
71              image = binarization(image)  # бинаризуем изображение
72              # обнаруживаем пиктограммы на изображении
73              centers, classes = detect_pictogramms(model, image)
74
75              send = False  # флаг пройденной проверки
76              # выполняем проверку стартовых условий,
77              # если нужная пара пиктограмм найдена - проверка считается успешной
78              for i in range(0, 3):
79                  if send:
80                      break
81                  for j in range(0, 4):
```

```
                    if direction == 'v':
                        send = (classes[i][j] == image1_index) and \
                                (classes[i + 1][j] == image2_index)
                        # запоминаем центры пиктограмм
                        pictogramm1_centre = centers[i][j]
                        pictogramm2_centre = centers[i + 1][j]
                    elif direction == 'h':
                        send = (classes[j][i] == image1_index) and \
                                (classes[j][i + 1] == image2_index)
                        # запоминаем центры пиктограмм
                        pictogramm1_centre = centers[j][i]
                        pictogramm2_centre = centers[j][i + 1]
                    if send:
                        break

            if send:  # если проверка пройдена, выводим сообщение оператору
                send_msg(image1, image2, recipient_names[current_recipient])
            else:
                # если проверка не пройдена, останавливаем программу
                # для перезапуска на следующей попытке
                run = False
                return

            # зажигаем белые сигнальные огни с интервалом мигания 1 секунда
            indication((255.0, 255.0, 255.0), 1)
            rospy.sleep(10)  # выполняем зависание на 10 секунд
            cargo.changeAllColor()  # выключаем светодиоды

            task2 = True  # вторая подзадача выполнена

            # высчитываем координаты точки сброса груза в пикселях
            distination_centre = (
                (pictogramm1_centre[1] + pictogramm2_centre[1]) // 2,  # X координата центра
                (pictogramm1_centre[0] + pictogramm2_centre[0]) // 2  # Y координата центра
            )

            # получаем новые координаты, 0.7 метров - высота 1 здания
            new_x, new_y = centering(*distination_centre, 0.7)
            ap.goToLocalPoint(new_x, new_y, 0.85)  # выполняем перелет в новую точку
        elif not task3:  # если задача 3 не выполнена, то
            cargo.off()  # выключаем магнитный захват
            task3 = True  # третья подзадача выполнена

            position_number = 0  # обнуляем счетчик точек в маршруте
            # отдаем команду полета в точку
            ap.goToLocalPoint(coordinates2[position_number][0],
                                coordinates2[position_number][1],
                                coordinates2[position_number][2])
        elif not task4:
            position_number += 1
            # сравниваем часло точек в маршруте с номером текущей точки
            if position_number < len(coordinates2):
                # отдаем команду полета в точку
                ap.goToLocalPoint(coordinates2[position_number][0],
                                    coordinates2[position_number][1],
                                    coordinates2[position_number][2])
            elif position_number == len(coordinates2):
```

```
                # если квадрокоптер закончил маршрут 2,
                # начинаем облет площадок
                # отдаем команду полета в точку
                ap.goToLocalPoint(coordinates2[-1][0] - 0.3,
                                    coordinates2[-1][1] + 0.3,
                                    coordinates2[-1][2])
            else:
                area_number += 1  # увеличиваем счетчик площадок
                image = get_img()  # получаем изображение с камеры
                image = binarization(image)  # бинаризация изображения

                area_image = get_area_image(area_number)  # загружаем изображения
                ↪  площадки
                cargo_found = is_cargo(image, area_image)  # поиск груза
                if cargo_found:
                    # индикация красным и синим
                    indication_two((255.0, 0.0, 0.0),
                                    (0.0, 0.0, 255.0), 1.0)
                    # отправка сообщения оператору
                    to_operator.publish(str(area_number))
                    task4 = True # четвертая подзадача выполнена
                    ap.landing() # отдаем команду посадки
                else:
                    # выполняем перелет на следующую площадку,
                    # в зависимости от номера текущей площадки
                    # вычисляем координаты следующей
                    delta_x = 0
                    delta_y = 0
                    if area_number == 1:
                        # смещение для перелета ко второй площадке
                        delta_x = 0.3
                        delta_y = 0.3
                    elif area_number == 2:
                        # смещение для перелета к третьей площадке
                        delta_x = -0.3
                        delta_y = -0.3
                    elif area_number == 3:
                        # смещение для перелета к четвертой площадке
                        delta_x = 0.3
                        delta_y = -0.3

                    # отдаем команду перелета к площадке
                    ap.goToLocalPoint(coordinates2[-1][0] + delta_x,
                                        coordinates2[-1][1] + delta_y,
                                        coordinates2[-1][2])
    # блок обработки события приземления
    elif event == CallbackEvent.COPTER_LANDED:
        run = False # прекращем программу
```

Произвести захват груза на крыше здания № 2

Для выполнения этой подзадачи необходимо воспользоваться функцией centering из предыдущей задачи, однако для выполнения этой функции необходимо знать точку центровки. Чтобы узнать точку центровки модифицируем функцию `is_cargo`, так чтобы она возвращала еще и центр самой большой несовпадающей области. Таким образом, функция `is_cargo` принимает вид.

```
# функция определения наличия груза на площадке
# сравнивает изображение площадки без груза и
# изображение площадки с камеры квадрокоптера
def is_cargo(frame, old_frame):
    # вычисляем разницу между изображениями
    difference = frame - old_frame
    # рассчитываем, сколько всего пикселей в изображении
    pixels_count = difference.shape[0] * difference.shape[1]
    # находим число белых пикселей на изображении
    white = np.count_nonzero(difference == 255)
    # если отношение белых пикселей к черным больше 10 процентов,
    # то груз находится на площадке
    cargo = white / pixels_count * 100 > 10
    if cargo:
        center = [] # список координат центров

        # находим все контуры в изображении разницы
        contours, _ = cv2.findContours(difference, cv2.RETR_EXTERNAL,
        ↪  cv2.CHAIN_APPROX_SIMPLE)

        # находим самый большой по площади контур, он и является  грузом
        max_cont = max(contours, key = cv2.contourArea)

        # находим момент контура
        moments = cv2.moments(max_cont)

        # с помощью момента находим центр контура
        center.append(int(moments["m10"] / moments["m00"]))
        center.append(int(moments["m01"] / moments["m00"]))
        return cargo, center
    return cargo, None
```

Одной из сложностей посадки является то, что функция `ap.landing()` гарантирует посадку, но не гарантирует посадку строго по вертикали без смещения в сторону. Поэтому для точной посадки необходимо снизиться на небольшую высоту (порядка 20 сантиметров) от поверхности и выполнить выключение двигателей (`ap.disarm()`). Еще одной сложностью является то, что после выполнения функции `disarm` автопилот не выдает никакого события, поэтому невозможно отследить выполнение этой команды, чтобы дальше продолжить программу и снова взлететь. Необходимо выполнить задержку, за время которой коптер точно приземлится. Таким образом, с учетом всех исправлений, функция `callback` принимает вид.

```
1  task5 = False # флаг, выполнена ли 5 подзадача или нет
2
3  def callback(event): # функция обработки событий Автопилота
4      global ap
5      global run
6      global cargo
7      global coordinates1
8      global coordinates2
9      global position_number
10     global current_color
11     global direction
12     global recipient_names
13     global image1_index
```

```
14        global image2_index
15        global image1
16        global image2
17        global model
18        global current_recipient
19        global task1
20        global task2
21        global task3
22        global task4
23        global task5
24        global area_number
25        global to_operator
26
27        event = event.data
28        # блок обработки события запуска двигателя
29        if event == CallbackEvent.ENGINES_STARTED:
30            ap.takeoff()  # отдаем команду взлета
31
32        # блок обработки события завершения взлета
33        elif event == CallbackEvent.TAKEOFF_COMPLETE:
34            if not task1:
35                position_number = 0  # обнуляем счетчик точек в маршруте
36                indication(current_color, 4)  # включаем индикацию
37                # отдаем команду полета в точку
38                ap.goToLocalPoint(coordinates1[position_number][0],
39                                coordinates1[position_number][1],
40                                coordinates1[position_number][2])
41            else:
42                rospy.sleep(10) # выполняем зависание на 10 секунд
43                ap.landing() # отдаем команду посадки
44
45        # блок обработки события достижения точки
46        elif event == CallbackEvent.POINT_REACHED:
47            if not task1:  # если задача не выполнена
48                position_number += 1  # наращиваем счетчик точек
49                # сравниваем число точек в маршруте с номером текущей точки
50                if position_number < len(coordinates1):
51                    # отдаем команду полета в точку
52                    ap.goToLocalPoint(coordinates1[position_number][0],
53                                    coordinates1[position_number][1],
54                                    coordinates1[position_number][2])
55                else:
56                    task1 = True # подзадача 1 выполнена
57                    ap.goToLocalPoint(1.8, 3.9, 1.5)  # снижаемся для выполнения
                      ↪  подзадачи 2
58            elif not task2:
59                # объявляем переменные центров пиктограмм
60                pictogramm1_centre = ()
61                pictogramm2_centre = ()
62
63                image = get_img()  # получаем изображение с камеры
64                image = binarization(image)  # бинаризуем изображение
65                # обнаруживаем пиктограммы на изображении
66                centers, classes = detect_pictogramms(model, image)
67
68                send = False  # флаг пройденной проверки
69                # выполняем проверку стартовых условий,
70                # если нужная пара пиктограмм найдена - проверка считается успешной
71                for i in range(0, 3):
72                    if send:
```

```
73                     break
74                 for j in range(0, 4):
75                     if direction == 'v':
76                         send = (classes[i][j] == image1_index) and \
77                                (classes[i + 1][j] == image2_index)
78                         # запоминаем центры пиктограмм
79                         pictogramm1_centre = centers[i][j]
80                         pictogramm2_centre = centers[i + 1][j]
81                     elif direction == 'h':
82                         send = (classes[j][i] == image1_index) and \
83                                (classes[j][i + 1] == image2_index)
84                         # запоминаем центры пиктограмм
85                         pictogramm1_centre = centers[j][i]
86                         pictogramm2_centre = centers[j][i + 1]
87                     if send:
88                         break
89 
90             if send:  # если проверка пройдена, выводим сообщение оператору
91                 send_msg(image1, image2, recipient_names[current_recipient])
92             else:
93                 # если проверка не пройдена, останавливаем программу
94                 # для перезапуска на следующей попытке
95                 run = False
96                 return
97 
98             # зажигаем белые сигнальные огни с интервалом мигания 1 секунда
99             indication((255.0, 255.0, 255.0), 1)
100            rospy.sleep(10)  # выполняем зависание на 10 секунд
101            cargo.changeAllColor()  # выключаем светодиоды
102 
103            task2 = True  # вторая подзадача выполнена
104 
105            # высчитываем координаты точки сброса груза в пикселях
106            distination_centre = (
107                (pictogramm1_centre[1] + pictogramm2_centre[1]) // 2,  # X
    ↪   координата центра
108                (pictogramm1_centre[0] + pictogramm2_centre[0]) // 2  # Y
    ↪   координата центра
109            )
110 
111            # получаем новые координаты, 0.7 метров - высота 1 здания
112            new_x, new_y = centering(*distination_centre, 0.7)
113            ap.goToLocalPoint(new_x, new_y, 0.85)  # выполняем перелет в новую
    ↪   точку
114        elif not task3:  # если задача 3 не выполнена, то
115            cargo.off()  # выключаем магнитный захват
116            task3 = True  # третья подзадача выполнена
117 
118            position_number = 0  # обнуляем счетчик точек в маршруте
119            # отдаем команду полета в точку
120            ap.goToLocalPoint(coordinates2[position_number][0],
121                              coordinates2[position_number][1],
122                              coordinates2[position_number][2])
123        elif not task4:
124            position_number += 1
125            # сравниваем часло точек в маршруте с номером текущей точки
126            if position_number < len(coordinates2):
127                # отдаем команду полета в точку
128                ap.goToLocalPoint(coordinates2[position_number][0],
129                                  coordinates2[position_number][1],
```

```
130                                     coordinates2[position_number][2])
131             elif position_number == len(coordinates2):
132                 # если квадрокоптер закончил маршрут 2,
133                 # начинаем облет площадок
134                 # отдаем команду полета в точку
135                 ap.goToLocalPoint(coordinates2[-1][0] - 0.3,
136                                     coordinates2[-1][1] + 0.3,
137                                     coordinates2[-1][2])
138             else:
139                 area_number += 1  # увеличиваем счетчик площадок
140                 image = get_img()  # получаем изображение с камеры
141                 image = binarization(image)  # бинаризация изображения
142 
143                 area_image = get_area_image(area_number)  # загружаем изображения
                ↪  площадки
144                 cargo_found, cargo_center = is_cargo(image, area_image)  # поиск
                ↪  груза
145                 if cargo_found:
146                     # индикация красным и синим
147                     indication_two((255.0, 0.0, 0.0),
148                                     (0.0, 0.0, 255.0), 1.0)
149                     # отправка сообщения оператору
150                     to_operator.publish(str(area_number))
151                     task4 = True # четвертая подзадача выполнена
152 
153                     new_x, new_y = centering(*cargo_center, 0.9) # находим
                    ↪  координаты смещения
154                     # отдаем команду перелета к первой площадке
155                     ap.goToLocalPoint(new_x, new_y, 1.1)
156                 else:
157                     # выполняем перелет на следующую площадку,
158                     # в зависимости от номера текущей площадки
159                     # вычисляем координаты следующей
160                     delta_x = 0
161                     delta_y = 0
162                     if area_number == 1:
163                         # смещение для перелета ко второй площадке
164                         delta_x = 0.3
165                         delta_y = 0.3
166                     elif area_number == 2:
167                         # смещение для перелета к третьей площадке
168                         delta_x = -0.3
169                         delta_y = -0.3
170                     elif area_number == 3:
171                         # смещение для перелета к четвертой площадке
172                         delta_x = 0.3
173                         delta_y = -0.3
174 
175                     # отдаем команду перелета к площадке
176                     ap.goToLocalPoint(coordinates2[-1][0] + delta_x,
177                                         coordinates2[-1][1] + delta_y,
178                                         coordinates2[-1][2])
179 
180         elif not task5: # если пятая подзадача не выполнена,
181             cargo.on() # включаем магнитный захват
182             ap.disarm() # выключаем двигатели для точной посадки
183             rospy.sleep(10) # ожидаем 10 секунд, чтобы коптер точно приземлился
184             ap.preflight() # отдаем команду выполенения предстартовой подготовки
185             task5 = True # пятая подзадача выполнена
186 
```

```
187     # блок обработки события приземления
188     elif event == CallbackEvent.COPTER_LANDED:
189         run = False  # прекращаем программу
```

Выполнить доставку груза и произвести посадку в центре посадочной площадки

Для выполнения этой задачи необходимо перелететь на крышу здания № 4. В центре крыши находится посадочная площадка, представляющая собой знак НТО, с зеленой точкой посередине. Есть множество способов приземлиться в центр посадочной площадки, однако самым простым является подлететь на достаточно низкое расстояние над посадочной площадкой, выполнить бинаризацию кадра, как было сделано в предыдущих подзадачах, получить точку центра посадочной площадки самого большого контура и выполнить центрирование относительно этой точки. Самым большим контуром при параметре, выставленном в функции `binarization`, будет знак НТО, соответственно центр этого контура совпадает с зеленой точкой. Чтобы не дописывать много кода, предлагается вынести часть функционала по нахождению центра контуров из функции `is_cargo` в отдельную функцию. Обновленная функция `is_cargo` и функция нахождения центра контура приведены ниже.

```
# функция определения наличия груза на площадке
# сравнивает изображение площадки без груза и
# изображение площадки с камеры квадрокоптера
def is_cargo(frame, old_frame):
    # вычисляем разницу между изображениями
    difference = frame - old_frame
    # рассчитываем, сколько всего пикселей в изображении
    pixels_count = difference.shape[0] * difference.shape[1]
    # находим число белых пикселей на изображении
    white = np.count_nonzero(difference == 255)
    # если отношение белых пикселей к черным больше 10 процентов,
    # то груз находится на площадке - возвращаем True
    return white / pixels_count * 100 > 10, get_center_bin(difference)

# функция, возвращающая центр самого большого контура
def get_center_bin(image):
    center = [] # список, в котором будут храниться координаты
    # находим все контуры в изображении разницы
    contours, _ = cv2.findContours(image, cv2.RETR_EXTERNAL,
     ↪  cv2.CHAIN_APPROX_SIMPLE)

    # находим самый большой контур, он и является  грузом
    max_cont = max(contours, key = cv2.contourArea)

    # находим момент контура
    moments = cv2.moments(max_cont)

    # с помощью момента находим центр контура
    center.append(int(moments["m10"] / moments["m00"]))
    center.append(int(moments["m01"] / moments["m00"]))
    return center
```

Таким образом, новая функция `callback` будет выглядеть следующим образом.

```
# точки, составляющие маршрут №4
coordinates4 = [
    [3.1, 1.64, 1.5],
    [3.1, 3.9, 1.5],
    [5.5, 3.9, 1.5],
    [6.57, 1.65, 1.3]
]

task6 = False # флаг, выполнена ли 6 подзадача или нет

def callback(event): # функция обработки событий Автопилота
    global ap
    global run
    global cargo
    global coordinates1
    global coordinates2
    global coordinates4
    global position_number
    global current_color
    global direction
    global recipient_names
    global image1_index
    global image2_index
    global image1
    global image2
    global model
    global current_recipient
    global task1
    global task2
    global task3
    global task4
    global task5
    global task6
    global area_number
    global to_operator

    event = event.data
    # блок обработки события запуска двигателя
    if event == CallbackEvent.ENGINES_STARTED:
        ap.takeoff()  # отдаем команду взлета

    # блок обработки события завершения взлета
    elif event == CallbackEvent.TAKEOFF_COMPLETE:
        position_number = 0  # обнуляем счетчик точек в маршруте
        if not task1:
            indication(current_color, 4)  # включаем индикацию
            # отдаем команду полета в точку
            ap.goToLocalPoint(coordinates1[position_number][0],
                              coordinates1[position_number][1],
                              coordinates1[position_number][2])
        elif not task6:
            rospy.sleep(10) # выполняем зависание на 10 секунд
            # отдаем команду полета в точку
            ap.goToLocalPoint(coordinates4[position_number][0],
                              coordinates4[position_number][1],
                              coordinates4[position_number][2])
    # блок обработки события достижения точки
    elif event == CallbackEvent.POINT_REACHED:
        if not task1:  # если задача не выполнена
```

```
            position_number += 1  # наращиваем счетчик точек
            # сравниваем число точек в маршруте с номером текущей точки
            if position_number < len(coordinates1):
                # отдаем команду полета в точку
                ap.goToLocalPoint(coordinates1[position_number][0],
                                    coordinates1[position_number][1],
                                    coordinates1[position_number][2])
            else:
                task1 = True # подзадача 1 выполнена
                ap.goToLocalPoint(1.8, 3.9, 1.5)  # снижаемся для выполнения
                ↪  подзадачи 2
        elif not task2:
            # объявляем переменные центров пиктограмм
            pictogramm1_centre = ()
            pictogramm2_centre = ()

            image = get_img()  # получаем изображение с камеры
            image = binarization(image)  # бинаризуем изображение
            # обнаруживаем пиктограммы на изображении
            centers, classes = detect_pictogramms(model, image)

            send = False  # флаг пройденной проверки
            # выполняем проверку стартовых условий,
            # если нужная пара пиктограмм найдена - проверка считается успешной
            for i in range(0, 3):
                if send:
                    break
                for j in range(0, 4):
                    if direction == 'v':
                        send = (classes[i][j] == image1_index) and \
                            (classes[i + 1][j] == image2_index)
                        # запоминаем центры пиктограмм
                        pictogramm1_centre = centers[i][j]
                        pictogramm2_centre = centers[i + 1][j]
                    elif direction == 'h':
                        send = (classes[j][i] == image1_index) and \
                            (classes[j][i + 1] == image2_index)
                        # запоминаем центры пиктограмм
                        pictogramm1_centre = centers[j][i]
                        pictogramm2_centre = centers[j][i + 1]
                    if send:
                        break

            if send:  # если проверка пройдена, выводим сообщение оператору
                send_msg(image1, image2, recipient_names[current_recipient])
            else:
                # если проверка не пройдена, останавливаем программу
                # для перезапуска на следующей попытке
                run = False
                return

            # зажигаем белые сигнальные огни с интервалом мигания 1 секунда
            indication((255.0, 255.0, 255.0), 1)
            rospy.sleep(10)  # выполняем зависание на 10 секунд
            cargo.changeAllColor()  # выключаем светодиоды

            task2 = True  # вторая подзадача выполнена

            # высчитываем координаты точки сброса груза в пикселях
            distination_centre = (
```

```
119                 (pictogramm1_centre[1] + pictogramm2_centre[1]) // 2,   # X
                    ↪  координата центра
120                 (pictogramm1_centre[0] + pictogramm2_centre[0]) // 2   # Y
                    ↪  координата центра
121             )
122 
123             # получаем новые координаты, 0.7 метров - высота 1 здания
124             new_x, new_y = centering(*destination_centre, 0.7)
125             ap.goToLocalPoint(new_x, new_y, 0.85)  # выполняем перелет в новую
                ↪  точку
126         elif not task3:  # если задача 3 не выполнена, то
127             cargo.off()  # выключаем магнитный захват
128             task3 = True  # третья подзадача выполнена
129 
130             position_number = 0  # обнуляем счетчик точек в маршруте
131             # отдаем команду полета в точку
132             ap.goToLocalPoint(coordinates2[position_number][0],
133                                 coordinates2[position_number][1],
134                                 coordinates2[position_number][2])
135         elif not task4:
136             position_number += 1
137             # сравниваем число точек в маршруте с номером текущей точки
138             if position_number < len(coordinates2):
139                 # отдаем команду полета в точку
140                 ap.goToLocalPoint(coordinates2[position_number][0],
141                                     coordinates2[position_number][1],
142                                     coordinates2[position_number][2])
143             elif position_number == len(coordinates2):
144                 # если квадрокоптер закончил маршрут 2,
145                 # начинаем облет площадок
146                 # отдаем команду полета в точку
147                 ap.goToLocalPoint(coordinates2[-1][0] - 0.3,
148                                     coordinates2[-1][1] + 0.3,
149                                     coordinates2[-1][2])
150             else:
151                 area_number += 1  # увеличиваем счетчик площадок
152                 image = get_img()  # получаем изображение с камеры
153                 image = binarization(image)  # бинаризация изображения
154 
155                 area_image = get_area_image(area_number)  # загружаем изображения
                    ↪  площадки
156                 cargo_found, cargo_center = is_cargo(image, area_image)  # поиск
                    ↪  груза
157                 if cargo_found:
158                     # индикация красным и синим
159                     indication_two((255.0, 0.0, 0.0),
160                                     (0.0, 0.0, 255.0), 1.0)
161                     # отправка сообщения оператору
162                     to_operator.publish(str(area_number))
163                     task4 = True # четвертая подзадача выполнена
164 
165                     new_x, new_y = centering(*cargo_center, 0.9) # находим
                        ↪  координаты смещения
166                     # отдаем команду перелета к первой площадке
167                     ap.goToLocalPoint(new_x, new_y, 1.1)
168                 else:
169                     # выполняем перелет на следующую площадку,
170                     # в зависимости от номера текущей площадки
171                     # вычисляем координаты следующей
172                     delta_x = 0
```

```
173                     delta_y = 0
174                     if area_number == 1:
175                         # смещение для перелета ко второй площадке
176                         delta_x = 0.3
177                         delta_y = 0.3
178                     elif area_number == 2:
179                         # смещение для перелета к третьей площадке
180                         delta_x = -0.3
181                         delta_y = -0.3
182                     elif area_number == 3:
183                         # смещение для перелета к четвертой площадке
184                         delta_x = 0.3
185                         delta_y = -0.3
186
187                     # отдаем команду перелета к площадке
188                     ap.goToLocalPoint(coordinates2[-1][0] + delta_x,
189                                         coordinates2[-1][1] + delta_y,
190                                         coordinates2[-1][2])
191
192         elif not task5: # если пятая подзадача не выполнена,
193             cargo.on() # включаем магнитный захват
194             ap.disarm() # выключаем двигатели для точной посадки
195             rospy.sleep(10) # ожидаем 10 секунд, чтобы коптер точно приземлился
196             ap.preflight() # отдаем команду выполенения предстартовой подготовки
197             task5 = True # пятая подзадача выполнена
198         elif not task6:
199             position_number += 1
200             # сравниваем число точек в маршруте с номером текущей точки
201             if position_number < len(coordinates4):
202                 ap.goToLocalPoint(coordinates4[position_number][0],
203                                     coordinates4[position_number][1],
204                                     coordinates4[position_number][2])
205             else:
206                 # индикация красным и белым
207                 indication_two((255.0, 0.0, 0.0),
208                                 (255.0, 255.0, 255.0), 1.0)
209
210                 rospy.sleep(5) # выполняем зависание на 5 секунд
211                 image = get_img()   # получаем изображение с камеры
212                 image = binarization(image)   # бинаризуем изображение
213
214                 bin_x, bin_y = get_center_bin(image) # получаем центр посадочной
                    ↪   площадки
215                 new_x, new_y = centering(bin_x, bin_y, 0.9) # находим координаты
                    ↪   смещения
216
217                 task6 = True # шестая подзадача выполнена
218                 # отдаем команду полета в точку центра посадочной площадки
219                 ap.goToLocalPoint(new_x, new_y, 1.1)
220                 run = False # прекращаем программу
221         else:
222             ap.disarm() # выключаем двигатели для точной посадки
223             cargo.changeAllColor() # выключаем светодиоды
224
225     # блок обработки события приземления
226     elif event == CallbackEvent.COPTER_LANDED:
227         run = False   # прекращаем программу
```

Материалы для подготовки

Программа подготовки к профилю «Автономные транспортные системы» Национальной технологической олимпиады — https://avt.global/nto_program.

Задачи профиля «Автономные транспортные системы» 2021-2022 — https://ntcontest.ru/docs/ats-assignements.pdf.

Задачи профиля «Автономные транспортные системы» 2020-2021 — https://drive.google.com/file/d/1jJwI_5MgX-wvmwK7WUh_mhQrEcFAhB_K/view.

Руководство по OpenCV — https://docs.opencv.org/4.x/d9/df8/tutorial_root.html.

SDK для программирования квадрокоптера Пионер Макс — https://github.com/geoscan/geoscan_pioneer_max.