Второй отборочный этап

Индивидуальная часть

Инженер-программист

Программирование автономного полета

Программа для автономного полета пишется на языке Python. Программа может получать телеметрию (заряд батареи, ориентацию, положение и т. д.) и отправлять команды: полететь в точку, установить ориентацию, угловую скорость и т. д.

Основные сервисы: `get_telemetry` (получение телеметрии), `navigate` (полет в заданную точку по прямой), `navigate_global` (полет в глобальную точку по прямой), `land` (переход в режим посадки). Сервисы, осуществляющие полет, только отправляют задание, но не ожидают его выполнения. Поэтому в некоторых случаях используется простая задержка по времени `rospy.sleep`, в других — специальные алгоритмы, использующие телеметрию, которые определяют, прилетел ли квадрокоптер в указанную точку.

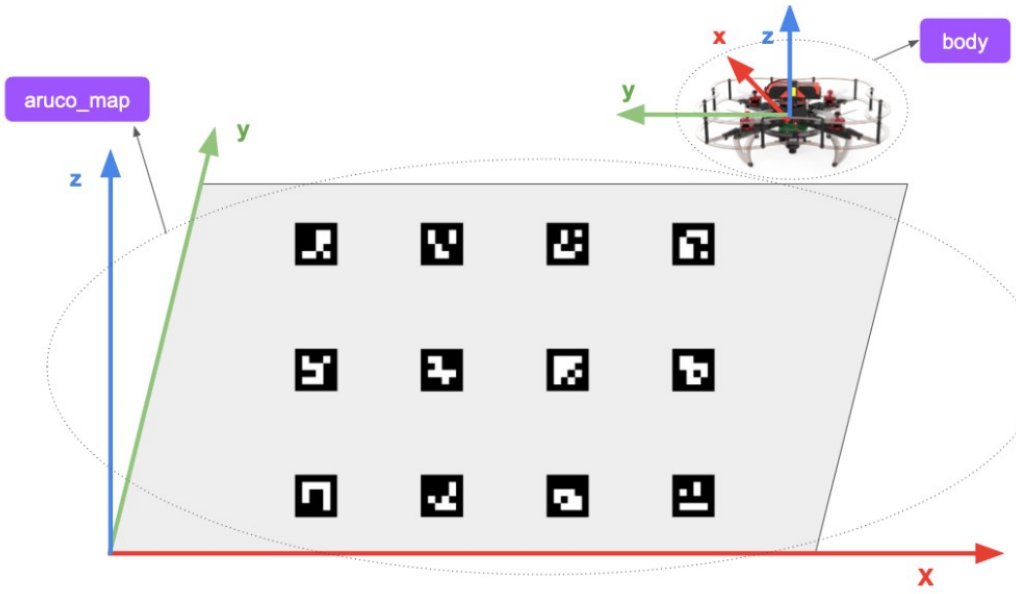
Сервис — это некоторый аналог функции, которая может быть вызвана из одной программы, а обработана в другой.

Для использования сервисов в вашей программе необходимо создать объекты, указывающие на тот или иной сервис, иначе код не будет знать, куда ему обращаться. Например:

```
get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
```

Также квадрокоптер может летать по нескольким системам координат. Основные из них:

- `map` — координаты относительно точки инициализации полетного контроллера;
- `aruco_map` — координаты относительно карты ArUco-маркеров;
- `body` — координаты относительно квадрокоптера без учета наклонов по тангажу и крену.



Задача II.1.1.1. Разбор полетов (2 балла)

После подключения к квадрокоптеру вы увидели загруженную программу. Мы бы не советовали запускать код неизвестного происхождения на реальном оборудовании, поэтому сначала проанализируем этот код.

```
import rospy
from clover import srv
from std_srvs.srv import Trigger

rospy.init_node('flight')

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)

navigate (x=0, y=0, z=1.5, speed=1, frame_id='body')
rospy.sleep(3)
navigate (x=1, y=1, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
navigate (x=1, y=3, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
navigate (x=3, y=3, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
navigate (x=3, y=1, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
navigate (x=1, y=1, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
land()
```

Какие действия выполнит квадрокоптер при запуске данного кода? Выберите один вариант из списка:

1. Нет правильного варианта.
2. Взлетит, опишет пятиугольник и сядет.
3. Взлетит и сядет.
4. Взлетит, опишет квадрат и сядет.

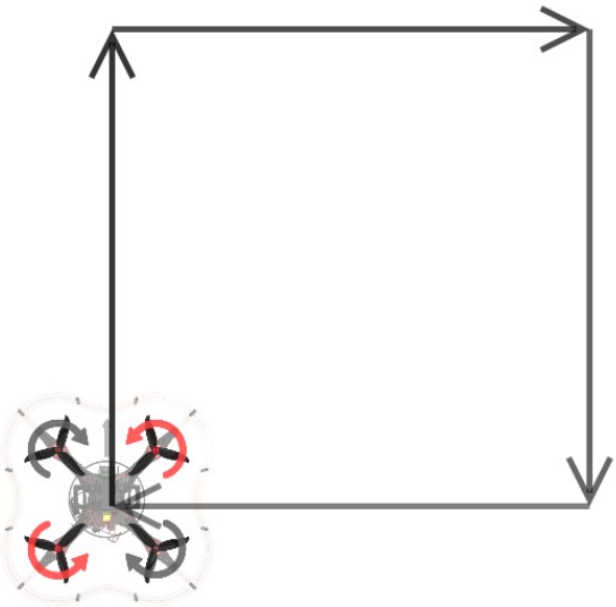
Решение

Как видим, в данной программе не объявлена функция-прокси `navigate`. Таким образом при запуске программный код выдаст ошибку.

Ответ: 1.

Задача II.1.1.2. Тесты (2 балла)

Для разработчиков систем автономного полета обычно первой задачей является полет по квадрату, поскольку по этому полету можно оценить работу всех подсистем квадрокоптера.



Заполните пропуски в коде так, чтобы при полете по этой программе квадрокоптер очертил квадрат со стороной 2 м по часовой стрелке. Изначально квадрокоптер находится в нижней левой точке.

Квадрокоптер использует систему удержания позиции *Optical Flow*, поэтому мы работаем в системе координат *Body*, центр которой всегда соответствует текущей позиции квадрокоптера.

Заполните пропуски

```
navigate (x=0, y=0, z=2, speed=1, frame_id='body')
rospy.sleep(3)
navigate (x= _____[1], y=_____[2], z=2, speed=1, frame_id='body')
```

```
rospy.sleep(3)
navigate (x= _____[3], y= _____[4], z=2, speed=1, frame_id='body')
rospy.sleep(3)
navigate (x= _____[5], y= _____[6], z=2, speed=1, frame_id='body')
rospy.sleep(3)
navigate (x= _____[7], y= _____[8], z=2, speed=1, frame_id='body')
rospy.sleep(3)
land()
```

Решение

Навигация квадрокоптера осуществляется относительно его корпуса. Квадрокоптеру необходимо последовательно пролететь вперед, вправо, назад, влево.

Для движения вперед квадрокоптеру необходимо изменить координату x в большую сторону на 2, т. е. $x = 2$, при этом координата $y = 0$.

Для движения вправо квадрокоптеру необходимо изменить координату y в меньшую сторону на 2, т. е. $y = -2$, при этом координата $x = 0$.

Для движения назад квадрокоптеру необходимо изменить координату x в меньшую сторону на 2, т. е. $x = -2$, при этом координата $y = 0$.

Для движения влево квадрокоптеру необходимо изменить координату y в большую сторону на 2, т. е. $y = 2$, при этом координата $x = 0$.

Ответ: 1 — 2; 2 — 0; 3 — 0; 4 — (-2); 5 — (-2); 6 — 0; 7 — 0; 8 — 2.

Задача II.1.1.3. Ожидание (3 балла)

Напишите программу, которая не использует константное/расчетное время полета, а ожидает прибытия в целевую точку.

На вход программе в стандартный поток ввода (`input()`) подаются координаты целевой точки, в которую квадрокоптер должен будет прилететь, и команда ожидать прибытия до этой точки.

Для ожидания завершения полета квадрокоптера до целевой точки нельзя использовать функции `rospy.sleep` и `time.sleep` (функция `rospy.sleep` может быть использована в одном цикле вместе с `get_telemetry` для уменьшения частоты вызова `get_telemetry`).

Можно использовать только систему координат map.

Библиотеки `math`, `rospy` и `time` уже импортированы.

Формат ввода:

На одной строке вводится 3 числа (x, y, z).

Пример:

0.8 -0.4 2.1

Описание сервисов квадрокоптера для выполнения данной задачи:

`get_telemetry`

Параметры:

- `frame_id` — система координат.

Формат ответа:

- `x, y, z` — локальная позиция коптера (м).

`navigate`

Пример программы-решения

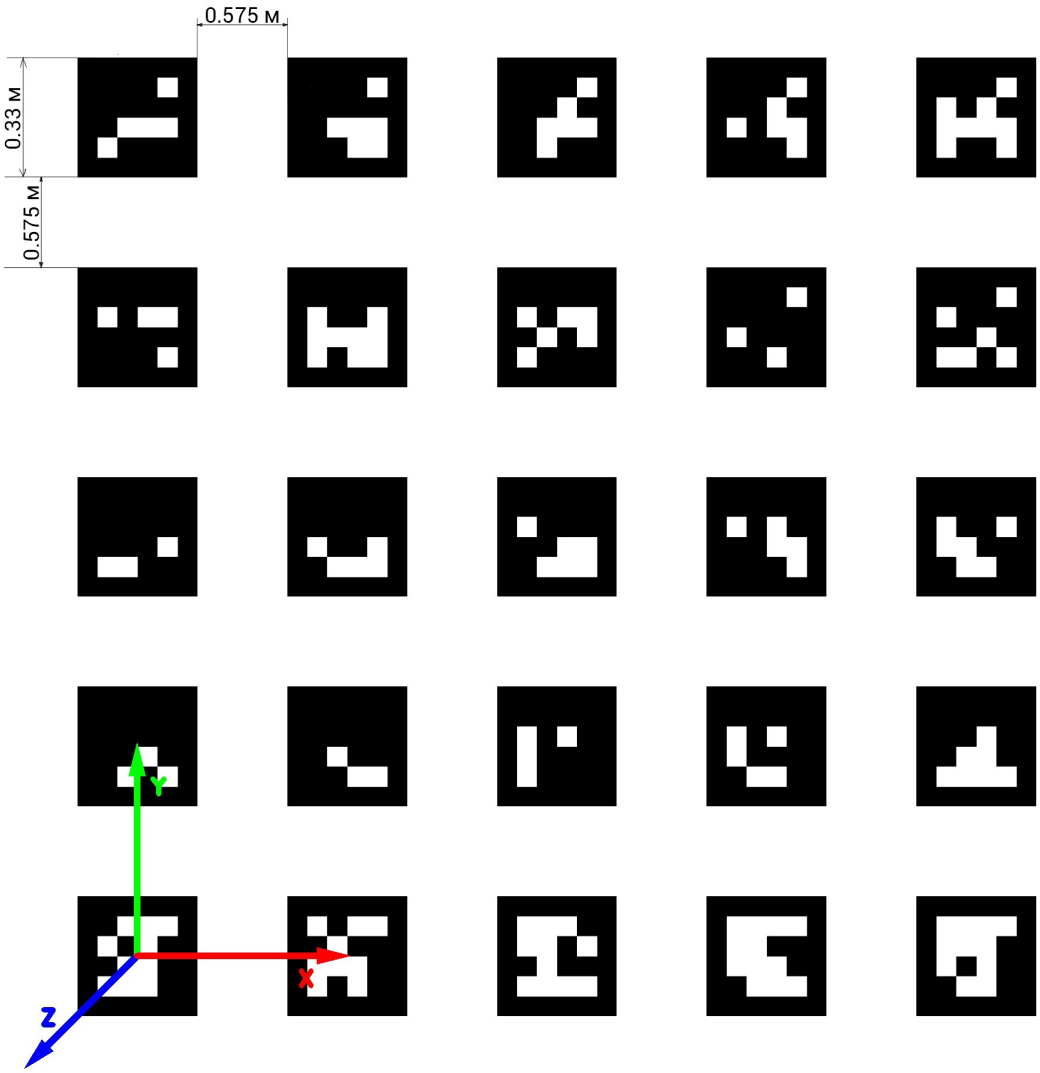
Ниже представлено решение на языке Python.

```
### Service proxy уже объявлены

def navigate_wait(x=0, y=0, z=0, yaw=float('nan'), speed=0.5, frame_id='',
↪  auto_arm=False, tolerance=0.2):
    navigate(x=x, y=y, z=z, yaw=yaw, speed=speed, frame_id=frame_id,
    ↪  auto_arm=auto_arm)

    while not rospy.is_shutdown():
        telem = get_telemetry(frame_id='map')
        if math.sqrt((telem.x - x) ** 2 + (telem.y - y) ** 2 + (telem.z - z) ** 2) <
        ↪  tolerance:
            break
        rospy.sleep(0.2)

x,y,z = map(float, input().split())
navigate_wait(x,y,z, frame_id="map")
```

Задача II.1.1.4. Настройте поле ArUco-маркеров (3 балла)

Пришло время настроить систему позиционирования квадрокоптера — в первую очередь по карте ArUco-маркеров.

Активируйте позиционирование робота по карте ArUco-маркеров (инструкция: https://clover.coex.tech/ru/aruco_map.html#%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и сгенерируйте поле с метками со следующими параметрами:



Считайте, что нумерация маркеров идет с левого нижнего угла и номер первого маркера — 100.

Для генерации меток нужно ввести команду с определенными значениями.

Пример команды для генерации поля, где:

- длина маркера = 0.335 м (`length`);
- 10 столбцов (`x`);
- 10 строк (`y`);
- расстояние между центрами меток по оси $x = 1$ м (`dist_x`);
- расстояние между центрами меток по оси $y = 1$ м (`dist_y`);
- номер первого маркера = 0 (`first`);
- название карты остается стандартным: map.txt;
- нумерация идет с нижнего левого угла (ключ `-bottom-left`).

```
rosrun aruco\_pose genmap.py 0.335 10 10 1 1 0 --bottom-left >
~/catkin\_ws/src/clover/aruco\_pose/map/map.txt
```

После начала решения у вас будет 60 минут, чтобы решить данную задачу. Для открытия командной строки нажмите Open Terminal. Если терминал не отвечает

на нажатие клавиш (завис), просто немного подождите. Не стоит нажимать какие-либо клавиши и еще больше нагружать сервер. После выполнение задания нажмите «Отправить». Задача будет проверена автоматически.

Решение

Выполняем указанные команды:

```
box@a23e5c620a30:~ $ nano catkin_ws/src/clover/clover/launch/clover.launch
box@a23e5c620a30:~ $ nano catkin_ws/src/clover/clover/launch/aruco.launch
box@a23e5c620a30:~ $ rosrun aruco_pose genmap.py 0.33 5 5 0.575 0.575 100
↪  --bottom-left > ~/catcin_ws/src/clover/aruco_pose/map/map.txt
box@a23e5c620a30:~ $
```

Заходим в файл `clover.launch` и включаем модуль `aruco`, в файле `aruco.launch` включаем все 3 настройки (согласно инструкции).

Аргумент aruco в файле ∼ */catkin_ ws/src/clover/clover/launch/clover.launch* должен быть в значении true:

```
<arg name="aruco"default="true"/>
```

Для включения распознавания карт маркеров аргументы aruco_map и aruco_detect в файле ∼ `/catkin_ws/src/clover/clover/launch/aruco.launch` должны быть в значении `true`:

```
<arg name="aruco_detect" default="true"/>
<arg name="aruco_map" default="true"/>
```

Для включения передачи координат в полетный контроллер по механизму VPE аргумент `aruco_vpe` должен быть в значении `true`:

```
<arg name="aruco_vpe"default="true"/>
```

Выполняем последнюю команду для генерации карты.

Инженер-техник

«Клевер» — это учебный конструктор программируемого квадрокоптера, состоящего из популярных открытых компонентов, а также набор необходимой документации и библиотек для работы с ним.

Набор включает в себя полетный контроллер Pixracer с полетным стеком PX4, Raspberry Pi 4 (https://clover.coex.tech/ru/raspberry.html) в качестве управляющего бортового компьютера, модуль камеры (https://clover.coex.tech/ru/camera.html) для реализации полетов с использованием компьютерного зрения, а также набор различных датчиков и другой периферии.

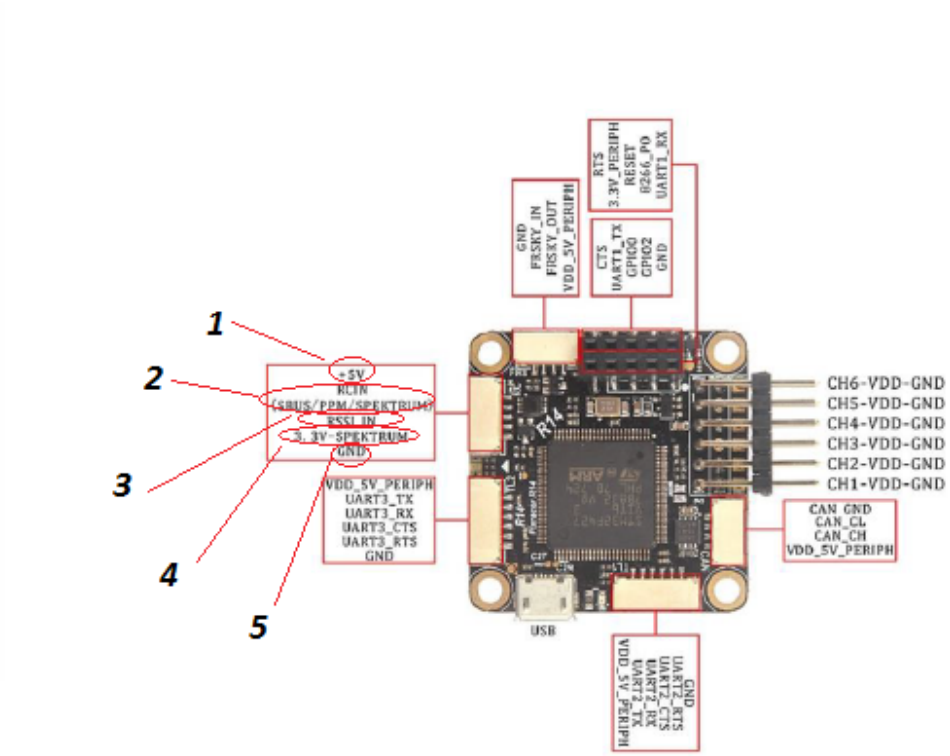
Платформа «Клевер» также включает в себя преднастроенный образ для Raspberry Pi (https://clover.coex.tech/ru/image.html) с полным набором необходимого ПО для работы со всей периферией и программирования автономных полетов (https://clover.coex.tech/ru/simple_offboard.html). Исходный код платформы «Клевер»

и всей документации открыт и доступен на GitHub (`https://github.com/CopterExpress/clover`).

Для отработки полетов рекомендуем использовать https://clover.coex.tech/ru/simulation.html

Задача II.1.2.1. Сборка (2 балла)

Какой из выделенных цифрами 1—5 контактов соответствует сигнальному проводу радиоприемника?



Выберите все подходящие ответы из списка:

1. Вариант 1.
2. Вариант 2.
3. Вариант 3.
4. Вариант 4.
5. Вариант 5.

Ответ: 2.

Задача II.1.2.2. Сборка (3 балла)

Тимофею нужно собрать гексакоптер. Моторы с пиковым током 30 А и регуляторы на 4-6S он уже выбрал. Помогите Тимофею выбрать аккумуляторы с максимальной возможной емкостью при условии, что он хочет взять свой гексакоптер на отдых, куда полетит со своей коллегой Викторией на самолете. (Подсказка: Виктория может помочь Тимофею с перевозкой. Обратите внимание на закон о перевозке аккумуляторов в самолетах.) Напряжение на банку считать равным 3,7 В.

**В вариантах решения 2 правильных ответа, но выбрать необходимо тот, который позволит работать дольше.*

Выберите все подходящие ответы из списка:

1. 6S 10000 mah 15c.
2. 6S 4000 mah 50c.
3. 6S 4500 mah 20c.
4. 2 аккумулятора 4S 6500 mah 25c, подключенных последовательно.
5. 2 аккумулятора 2S 12000 mah 15c, подключенных последовательно.

Решение

Согласно правилам Международной ассоциации воздушного транспорта (IATA) о перевозке опасных грузов от 1 января 2018 года, максимальная разрешенная к перевозке мощность аккумуляторов на одного пассажира составляет 100 Вт/ч. Т. к. по условию задачи в перелете принимают участие 2 человека, то возможен вариант перевозки 2-х аккумуляторов, суммарная мощность которых превышает 100 Вт/ч, но каждый из них соответствует этому ограничению.

Также по условию задачи максимальный потребляемый ток составляет 180 А (по 30 А на каждый из 6 моторов). Стоит учесть, что при последовательном подключении напряжение аккумуляторов суммируется и может превысить максимальное допустимое. Вычислим мощность каждого из аккумуляторов по формуле $P = U \cdot C$ (где U — напряжение аккумулятора — 3,7 В на банку, а C — его емкость в А· ч) и максимальный ток для каждого аккумулятора по формуле $I = C \cdot cI$ (где c — токоотдача).

$$P = 222 \text{ Вт/ч}, I = 150\ A$$

Не подходит из-за слишком большой перевозимой мощности и низкого максимального тока.

$$P = 88,8 \text{ Вт/ч}, I = 200\ A$$

Подходит, мощность 88,8 Вт/ч.

$$P = 99,9 \text{ Вт/ч}, I = 90\ A$$

Не подходит из-за низкого пикового тока.

$$P = 2 \cdot 109,2 \text{ Вт/ч}, I = 162,5\ A$$

Не подходит из-за слишком высокого напряжения (8S) и мощности.

$$P = 2 \cdot 88,8 \text{ Вт/ч}, I = 177,6\ A$$

Подходит, мощность 177,6 Вт/ч.

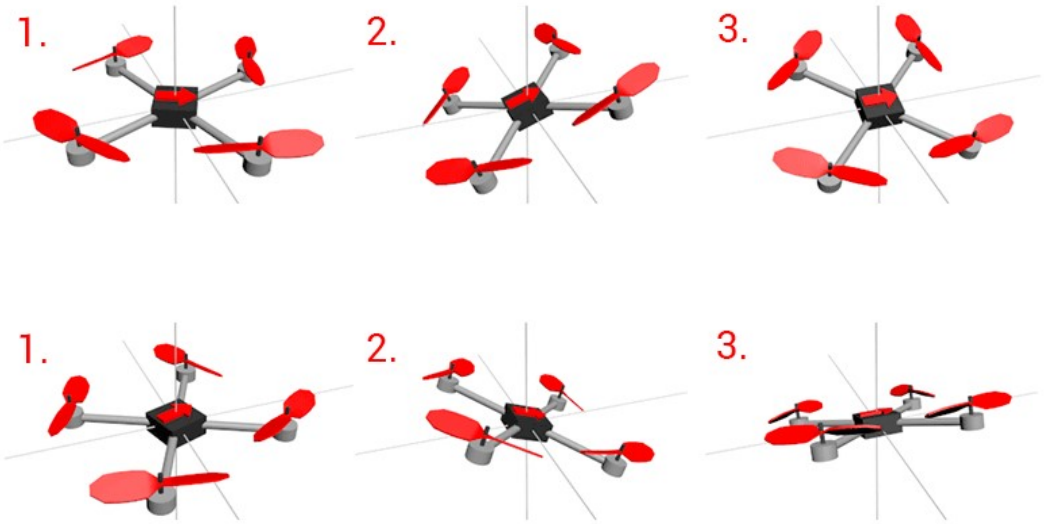
Варианты 2 и 5 подходят, но аккумуляторы в варианте 5 будут работать дольше, т. к. являются более мощными.

Ответ: 5.

Задача II.1.2.3. Ручное управление квадрокоптером (1 балл)

Управление квадрокоптером происходит с помощью двух стиков на аппаратуре. По умолчанию левый стик отвечает за газ и рысканье, а правый — за крен и тангаж. Данные термины используются для всех летательных аппаратов.

Сопоставьте картинки с правильными названиями осей связанной системы координат:



Пронумеруйте значения из списка:

- Рысканье.
- Крен.
- Тангаж.

Решение

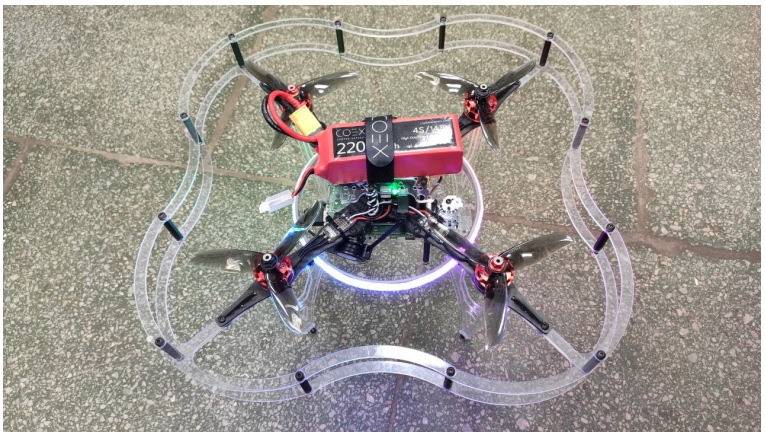
Опираясь на условия задачи, содержащее иллюстрации и описание компонентов, найдем соответствия между ними и расположим названия напротив цифр, наиболее соответствующих данному описанию и картинке.

- Газ (`throttle`) — отвечает за скорость вращения двигателей.
- Рысканье (`yaw`) — отвечает за повороты вокруг вертикальной оси (Z), по часовой (при наклоне вправо) и против часовой (при наклоне влево) стрелки.
- Тангаж (`pitch`) — отвечает за наклон или движение вперед/назад.
- Крен (`roll`) — отвечает за наклон или движение влево/вправо.

Ответ: 1 — Рысканье; 2 — Тангаж; 3 — Крен.

Задача II.1.2.4. Предполетная подготовка (1 балл)

После осмотра квадрокоптера участника не допустили до полетов. Укажите причину.



Выберите все подходящие ответы из списка:

1. Балансир попадает в зону вращения пропеллеров.
2. На квадрокоптер неправильно установлены пропеллеры.
3. Квадрокоптер полностью исправен.
4. На двух ближайший регуляторах оборотов отсутствуют стяжки.

Ответ: 1.

Задача II.1.2.5. Подключение (2 балла)

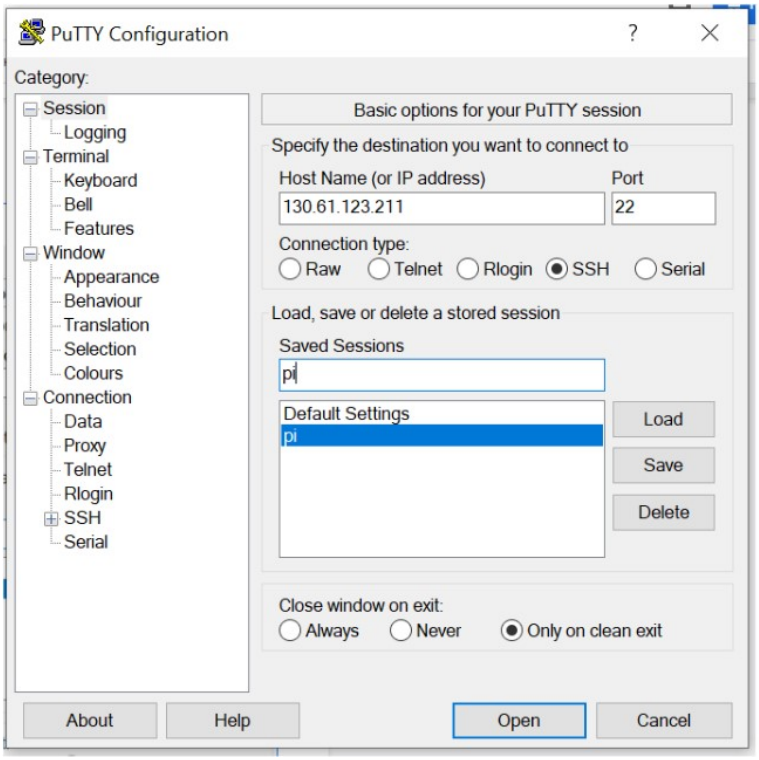
Подключитесь к одноплатному компьютеру по протоколу SSH.

Откройте файл `answer.txt` и введите набор символов, который написан после слов «`Copy this text to answer field on STEPIK`»:.

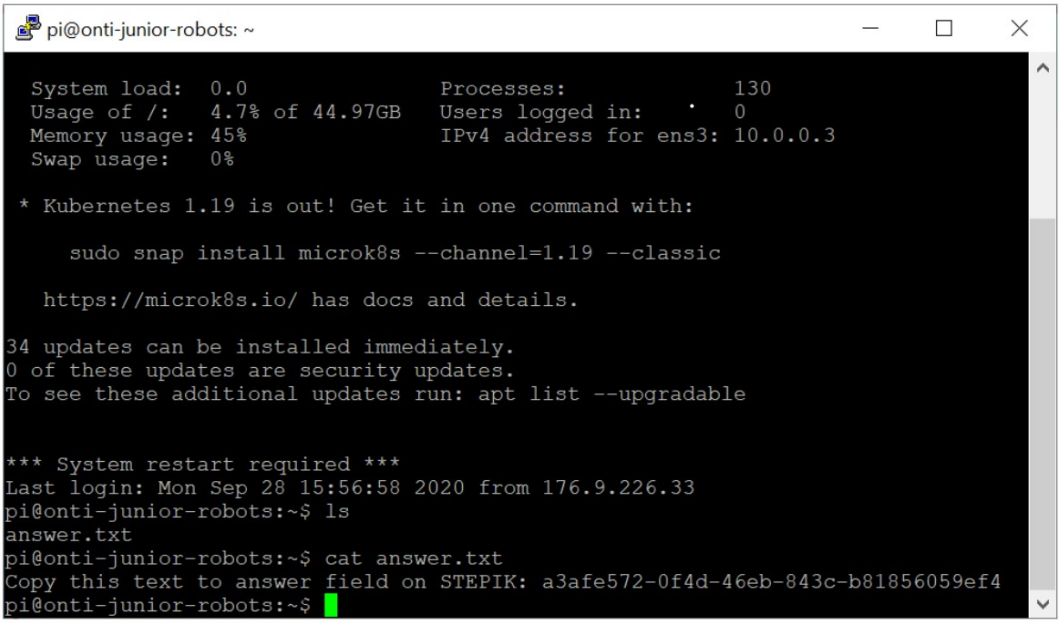
Данные для подключения:
IP: `130.61.123.211`
Port: `22`
Login: `pi`
Password: `raspberry`

Решение

1. Необходимо скачать программу `PuTTY`.
2. Ввести данные для подключения (IP, порт и логин) и нажать кнопку Open.



3. Ввести пароль, просмотреть содержимое текущей директории и вывести содержимое файла answer.txt.



4. Скопировать набор символов, который написан после слов «`Copy this text to answer field on STEPIK`»: и ввести в поле ответа `a3afe572-0f4d-46eb-843c-b81856059ef4`

Ответ: `a3afe572-0f4d-46eb-843c-b81856059ef4`.

Задача II.1.2.6. Настройка образа, неисправности (6 баллов)

Просмотрите видео из предыдущего шага (https://stepik.org/lesson/420781/step/7?unit=422315) и определите допущенные ошибки. Опишите обнаруженные ошибки и способ устранения данных ошибок. Пример описания:

- Описание ошибки №1. Неверное направление вращения мотора №3 (CCW).
- Способ устранения ошибки №1: Отпаять два соседних фазных провода от регулята оборотов мотора №3, поменять их местами, припаять.

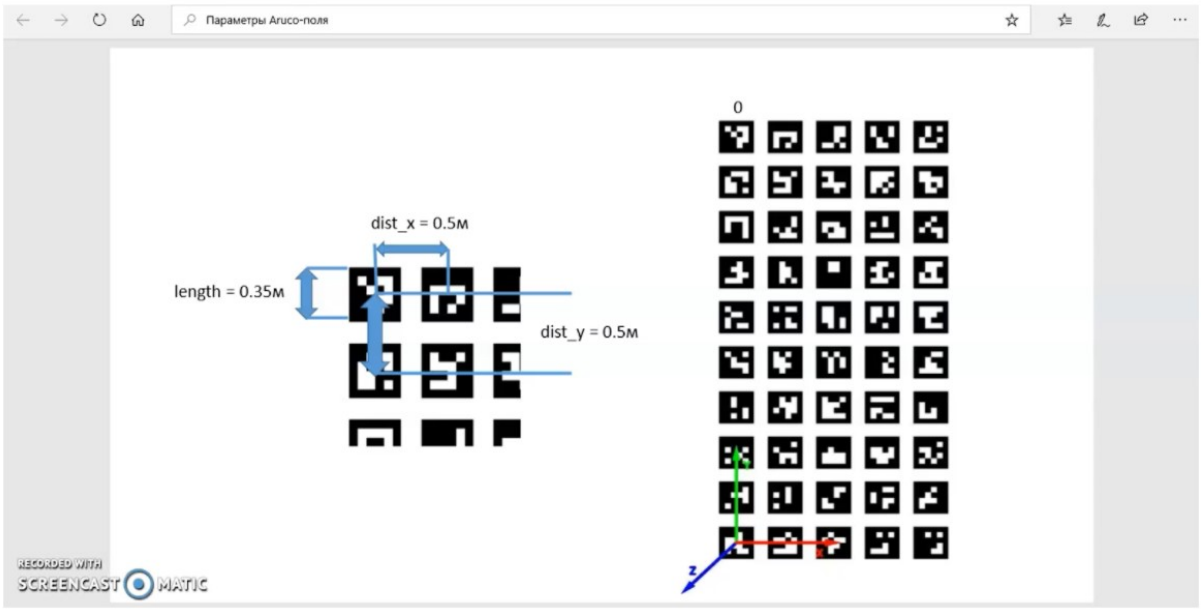
Оценивается:

- правильность ответа;
- применение профессиональной терминологии.

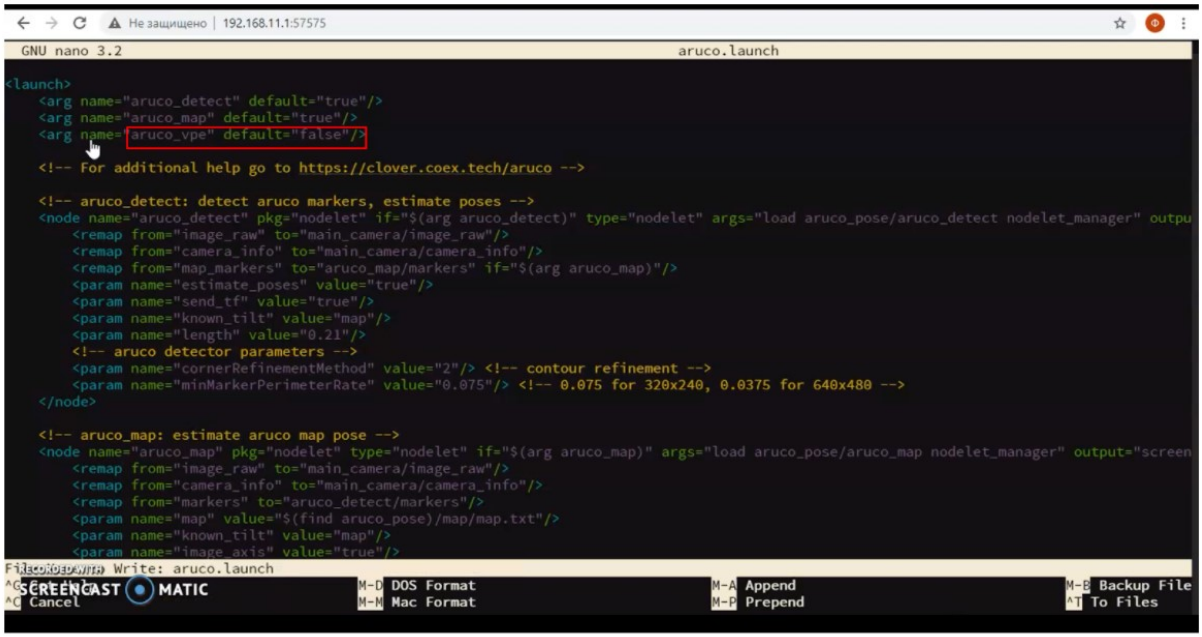
Решение

На видео продемонстрирован процесс настройки образа Clover для автономного полета квадрокоптера.

Характеристики карты ArUco-маркеров для настраиваемого образа:

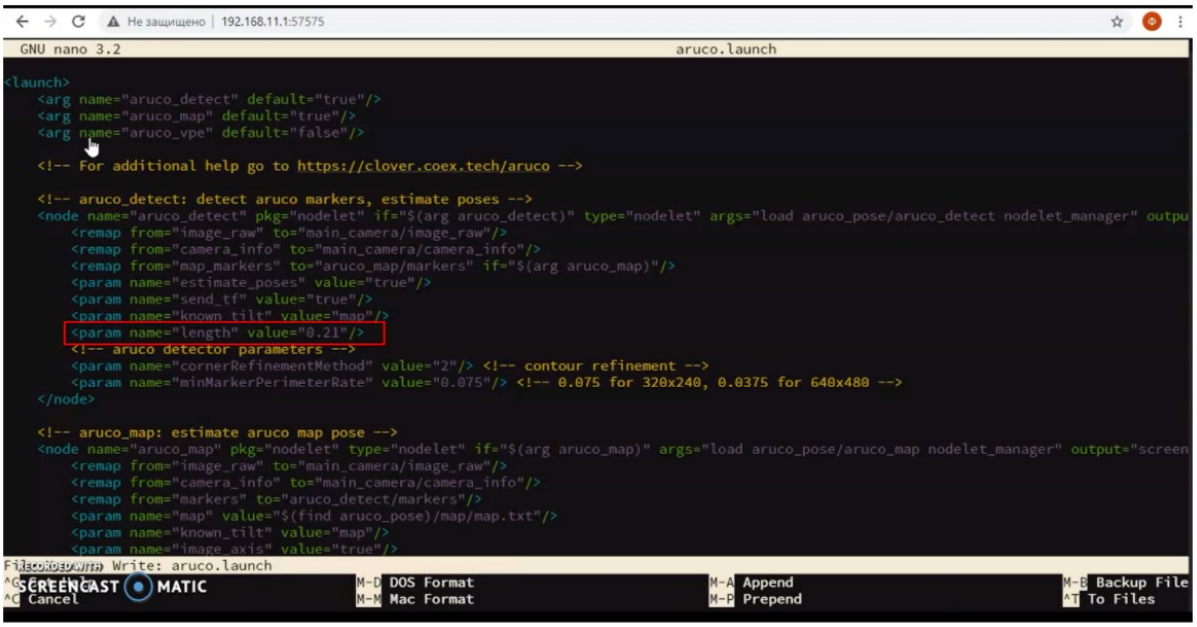


1. Описание ошибки №1. В файле в файле `aruco.launch` выключена передача данных о позиции квадрокоптера с использованием визуальной навигации для работы estimator'а (*visual position estimation*) в полетный контроллер.



Способ устранения ошибки №1. Необходимо аргумент `aruco_vpe` в файле aruco.launch изменить на «`true`» и сохранить измененный файл.

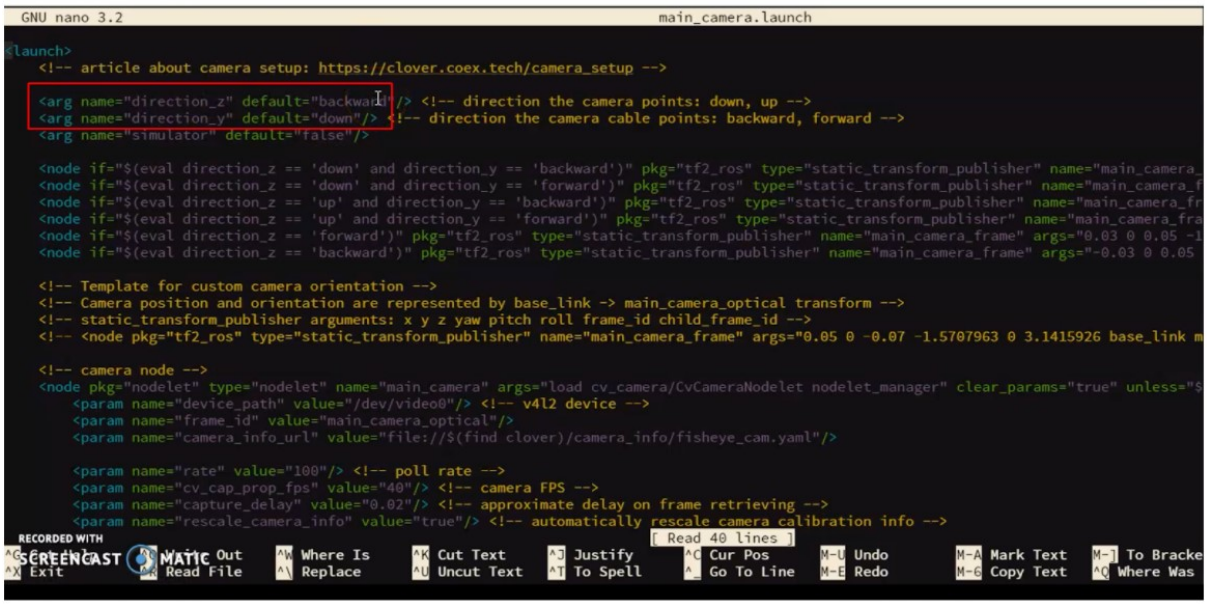
2. Описание ошибки №2. Размер ArUco-маркера на реальной карте 0,35 м. Аргумент `length`, который отвечает за ширину маркеров на карте имеет значение 0,21 м.



Способ устранения ошибки №2. Перезаписать аргумент `length` в файле `aruco.launch` на значение 0.35.

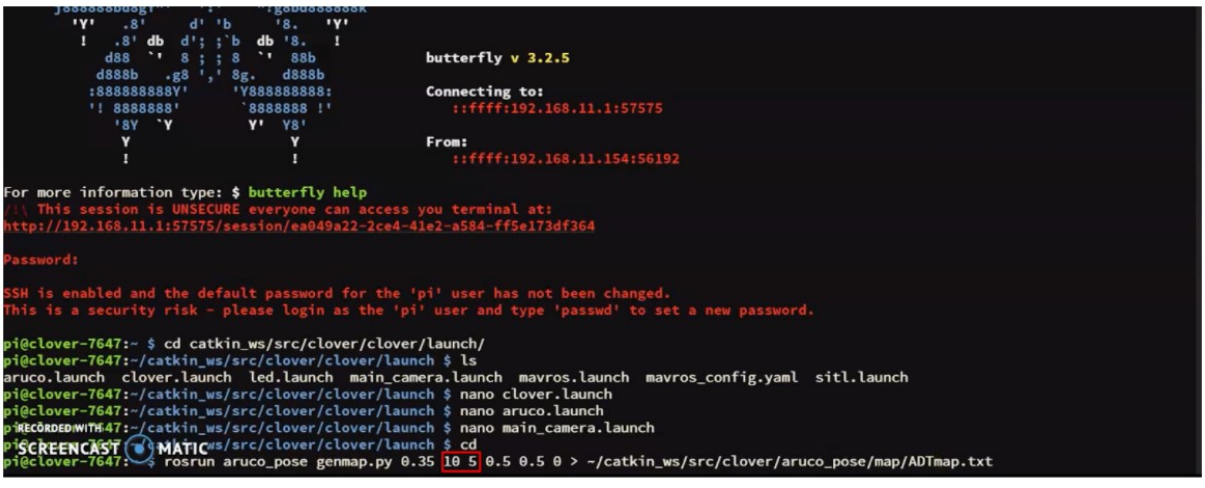
3. Описание ошибки № 3. Согласно документации (https://clover.coex.tech/ru/camera_setup.html#frame), аргумент `direction_z` не должен принимать значение «`backward`», а аргумент `direction_y` — значение «`down`». Оба аргумента задают расположение камеры на дроне. Для того чтобы задать ориентацию, необходимо установить:
    - направление обзора камеры `direction_z`: вниз (`down`) или вверх (`up`);

- направление, в которое указывает шлейф камеры `direction_y`: назад (`backward`) или вперед (`forward`).



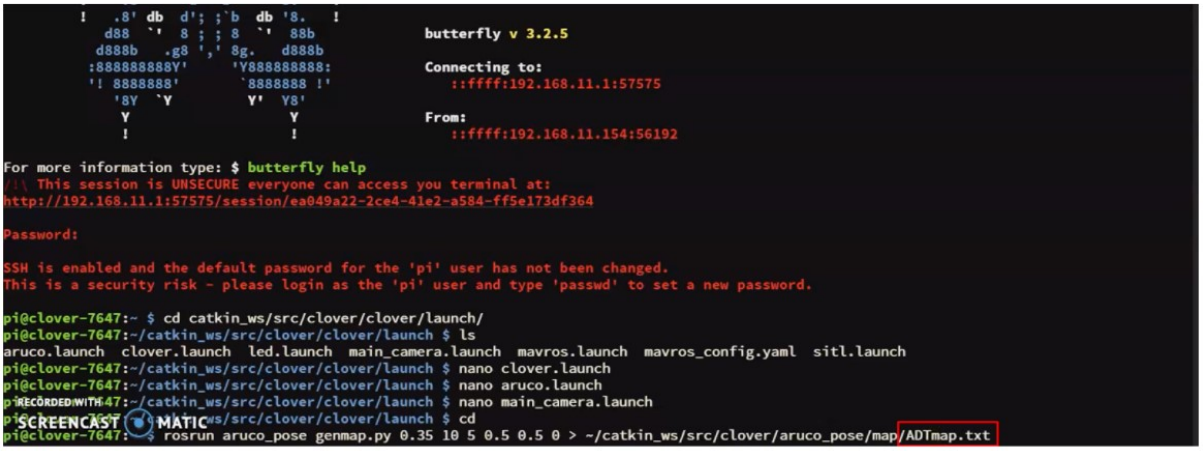
Способ устранения ошибки №3. Необходимо перезаписать аргументы `direction_z` и `direction_y` на корректные значение («`down`» или «`up`», «`backward`» или «`forward`» соответственно, в зависимости от расположения камеры).

4. Описание ошибки №4. При использовании утилиты `genmap.py` для генерации карты были перепутаны местами аргументы, отвечающие за количество маркеров по оси абсцисс и ординат. Это приведет к построению карты с 10 маркерами по оси X и с 5 маркерами по оси Y, что не соответствует входным данным.



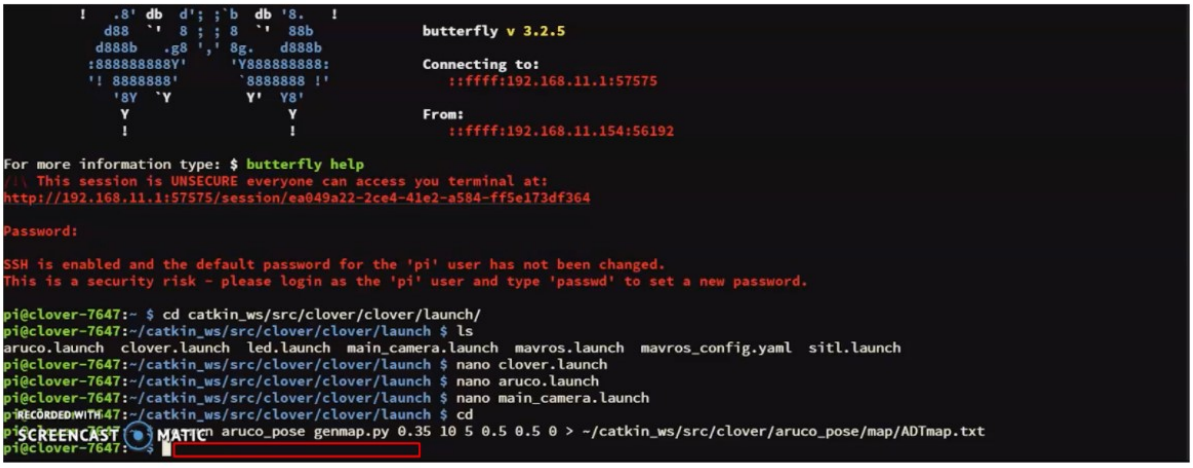
Способ устранения ошибки №4. Командой передать аргументы программе в корректной последовательности (5 для параметра X, 10 для параметра Y).

5. Описание ошибки №5. При генерации карты с помощью утилиты genmap.py карта создается в документе с названием `ADTmap.txt`, хотя в файле aruco.launch карта подгружается с файла `map.txt`.



Способ устранения ошибки №5. При вызове утилиты `genmap.py` указать путь к файлу `map.txt` и перезаписать его либо исправить путь к нужной карте в файле `aruco.launch`.

6. Описание ошибки №6. После проведения настроек и изменения launch-файлов не были перезапущены сервисы clover'a.



Способ устранения ошибки №6. В терминале выполнить команду:

```
sudo systemctl restart clover
```

Математик/аналитик

Беспилотники — одни из самых ценных устройств для сбора данных. Основная их задача: фиксация, хранение и передача информации, а с недавнего времени — доставка последней мили.

Последняя миля — это самый последний этап доставки товара в любой логистической цепочке, даже самой длинной, или последний этап поездки из дома на работу или обратно.Существует так называемая «проблема последней мили» — сложность, связанная с последним этапом доставки товаров до двери покупателя или до магазина, и сложность с тем, чтобы быстро определить оптимальный путь доставки, учитывающий минимальное расстояние и затрачиваемое время.

До недавнего времени коммерческое использование беспилотников фокусировалось на точном сборе и визуализации данных, а не на интеграции в IT-процессы.

Для дронов во всех отраслях давно разработано множество приложений: в сельском хозяйстве, энергетике, горнодобывающей промышленности, телекоммуникациях. Налажена связь беспилотников с облачными хранилищами. Однако большая часть этих приложений только создает и обслуживает карты, например карты местоположения для управления и обслуживания инфраструктуры компании или ее активов, но дроны еще не способны самостоятельно искать кратчайшие маршруты и преодолевать препятствия на должном уровне.

Анализ данных — наука, в которой работают хорошо обоснованные теоретические методы и эвристики, но лишь их грамотное сочетание позволяет успешно решать практические задачи в различных отраслях, включая логистику доставки товаров беспилотниками.

Задача II.1.3.1. Скорость движения (3 балла)

Квадрокоптер правонарушителя Филиппа летит по прямой траектории параллельно земле. Когда он пролетает над головой полицейского Елены, она стреляет сеткой из ловушки для дронов параллельно движению коптера под углом 30° к земле.

Найдите, с какой скоростью двигался квадрокоптер, если известно, что высота его полета — 35 метров, Елена попала сеткой, а скорость полета сетки в момент вылета составляла 80 м/с. Ответ дать с точностью до десятых(через запятую). Ускорение свободного падения считать равным 10 м/с^2, сопротивлением воздуха пренебречь.

Решение

В данной задаче сетка, выпущенная Еленой, представляет собой тело, брошенное под углом к горизонту. Нам необходимо определить траекторию полета сетки и в какой точке произошло ее попадание в квадрокоптер.

Вектор скорости тела в любой точке траектории можно разложить на 2 составляющих: вектор V_x и вектор V_y.

$$V = V_x + V_y$$

Составляющие скорости по осям O_x и O_y в начальный момент времени равны:

$$V_x = V_0 cos\alpha$$

$$V_y = V_0 sin\alpha - gt$$

Расчет координат по оси O_y после начала полета вычисляем по формуле:

$$H = V_0 sin\alpha t - gt^2/2$$

Найдем время, когда сетка была на высоте 35 метров, чтобы произошло ее попадание в квадрокоптер. Решаем квадратное уравнение:

$$35 = 80 sin30t - 5t^2$$

$$5t^2 - 40t + 35 = 0$$

Таким образом, время через которое сетка попадет в квадрокоптер, равно 1 с. Найдем расстояние, которое пролетела сетка с момента выстрела до попадания в коптер по оси Ox, по формуле:

$$S = V_0 cos\alpha t$$

$$S = 69,28$$

Итак, вычислим скорость полета квадрокоптера, зная, что от пролета над Еленой до момента попадания коптер пролетел 69,28 м за 1 с:

$$V_{\text{коптера}} = S/t$$

$$V_{\text{коптера}} = 69,28 \text{ м/с}$$

Ответ: 69,3.

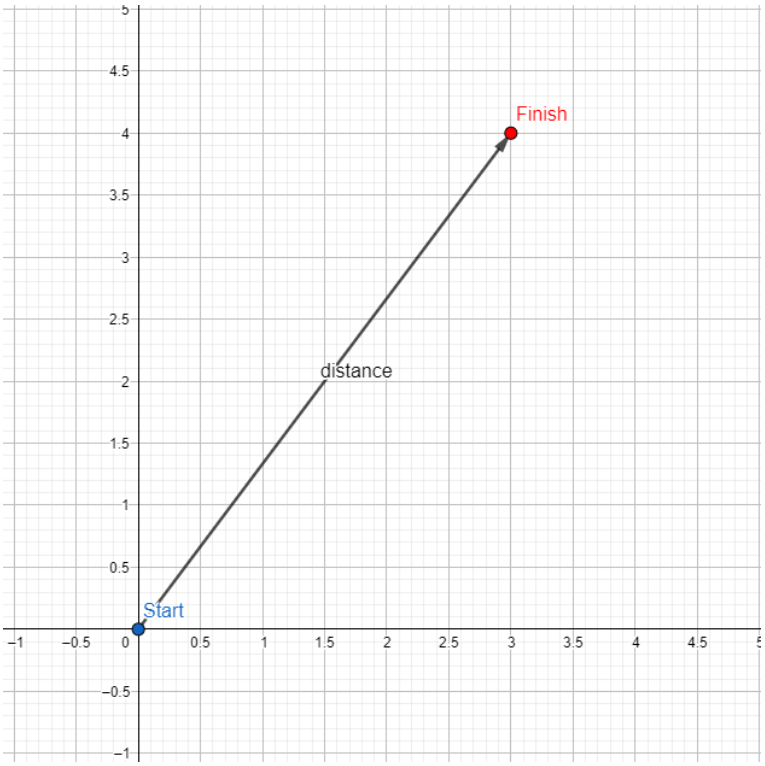
Задача II.1.3.2. Полеты (3 балла)

Для ведения статистики всегда интересно, какое в сумме расстояние пролетает квадрокоптер. Напишите программу программу для расчета расстояния, которое пролетает квадрокоптер.

Вам известны две стороны, осталось найти третью, по которой он возвращался на точку старта.

По заданным координатам точек найдите расстояние между ними.

Например:



Согласно графику выше, квадрокоптер начинает полет из точки (0,0) в точку (3,4).

Расстояние между точками равно $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Напишите программу для расчета расстояния между двумя координатами.

Необходимо вывести ответ в стандартный вывод (`print()`) и округлить до сотых.

Примеры

Пример №1

Стандартный ввод
`87 36 0 89`
Стандартный вывод
`101.87`

Пример №2

Стандартный ввод
`49 96 97 17`
Стандартный вывод
`92.44`

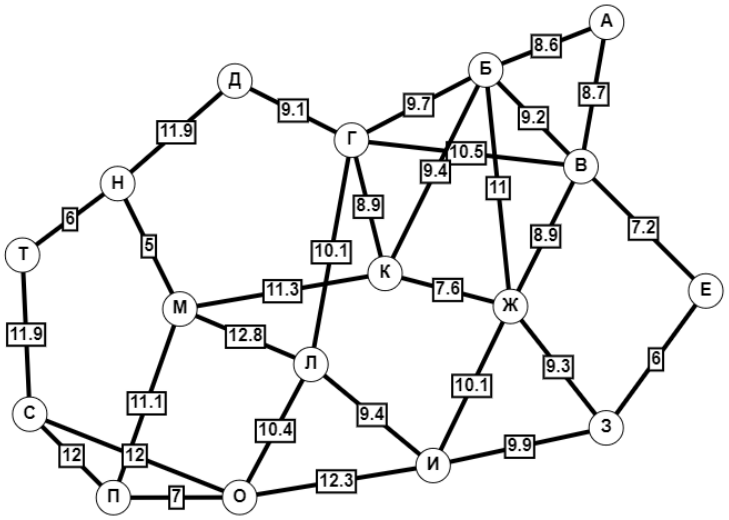
Пример программы-решения

```python
import math

# Считываем данные
x1, y1, x2, y2 = map(int, input().split())

# Производим расчет расстояния по формуле из условия
distance = math.sqrt((x2-x1) ** 2 + (y2-y1) **2)

# Выводим значение, округленное до 2 знаков после запятой
print(round(distance, 2))
```

Задача II.1.3.3. Кратчайший путь (4 балла)

На рисунке представлена схема воздушных дорог, связывающих пункты доставки (вершины графа). Числами указаны длины дорог в километрах. Дрону необходимо доставить груз из точки **A** в точку **C** по оптимальному маршруту.

Сколько метров составляет кратчайший путь от склада **A** до пункта доставки **C**?



Решение

Для решения задачи поиска кратчайшего пути на графе между парой вершин используется алгоритм Дейкстры.

1. Устанавливаем расстояние $D\,[i]$ от начальной вершины s до всех остальных в ∞.
2. Полагаем $D\,[?] = 0$.
   Помещаем все вершины в очередь с приоритетом Q.
3. Запускаем цикл из n итераций (по числу вершин).

Итерация №0

Извлекаем из очереди Q вершину $v = 0$ с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Возможно пути из $s = 0$ через вершину $v = 0$ стали короче, выполняем проверку: для каждой вершины u смежной с вершиной v и не включенной в H проверяем и корректируем расстояние $D\,[u]$.

n	*0*	*А*	*Б*	*В*	*Г*	*Д*	*Е*	*Ж*	*З*	*И*	*К*	*Л*	*М*	*Н*	*О*	*П*	*С*
D	*0*	*0+8.6=8.6*	*0+8.7=8.7*	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Итерация №1

Извлекаем из очереди Q вершину $v =$ А с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	*0*	*А*	*Б*	*В*	*Г*	*Д*	*Е*	*Ж*	*З*	*И*	*К*	*Л*	*М*	*Н*	*О*	*П*	*С*
D	*0*	*8.6*	*8.7*	*8.6+9.7=18.3*	∞	∞	*8.6+11=19.6*	∞	∞	*8.6+9.4=18*	∞	∞	∞	∞	∞	∞	∞

Итерация №2

Извлекаем из очереди Q вершину $v =$ Б с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	*0*	*А*	*Б*	*В*	*Г*	*Д*	*Е*	*Ж*	*З*	*И*	*К*	*Л*	*М*	*Н*	*О*	*П*	*С*
D	*0*	*8.6*	*8.7*	*18.3*	∞	*8.7+7.2=15.9*	*8.7+8.9=17.6*	∞	∞	*18*	∞	∞	∞	∞	∞	∞	∞

Итерация №3

Извлекаем из очереди Q вершину $v =$ Д с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	*0*	*А*	*Б*	*В*	*Г*	*Д*	*Е*	*Ж*	*З*	*И*	*К*	*Л*	*М*	*Н*	*О*	*П*	*С*
D	*0*	*8.6*	*8.7*	*18.3*	∞	*15.9*	*17.6*	*15.9+6=21.9*	∞	*18*	∞	∞	∞	∞	∞	∞	∞

Итерация №4

Извлекаем из очереди Q вершину $v =$ Е с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

n	*0*	*А*	*Б*	*В*	*Г*	*Д*	*Е*	*Ж*	*З*	*И*	*К*	*Л*	*М*	*Н*	*О*	*П*	*С*
D	*0*	*8.6*	*8.7*	*18.3*	∞	*15.9*	*17.6*	*21.9*	∞	*18*	∞	∞	∞	∞	∞	∞	∞

Итерация №5

Извлекаем из очереди Q вершину $v =$ И с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	∞	15.9	17.6	21.9	∞	18	∞	18+11.3=29.3	∞	∞	∞	∞	∞

Итерация №6

Извлекаем из очереди Q вершину $v =$ В с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	18.3+9.1=27.4	15.9	17.6	21.9	∞	18	18.3+10.1=28.4	29.3	∞	∞	∞	∞	∞

Итерация №7

Извлекаем из очереди Q вершину $v =$ Ж с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	21.9+9.9=31.8	18	28.4	29.3	∞	∞	∞	∞	∞

Итерация №8

Извлекаем из очереди Q вершину $v =$ Г с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	27.4+11.9=39.3	∞	∞	∞	∞

Итерация №9

Извлекаем из очереди Q вершину $v =$ К с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	39.3	28.4+10.4=38.8	∞	∞	∞

Итерация №10

Извлекаем из очереди Q вершину $v = $ Л с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	29.3+5=34.3	38.8	29.3+11.1=40.4	∞	∞

Итерация №11

Извлекаем из очереди Q вершину $v = $ З с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	34.3	38.8	40.4	∞	∞

Итерация №12

Извлекаем из очереди Q вершину $v = $ М с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	34.3	38.8	40.4	∞	34.3+6=40.3

Итерация №13

Извлекаем из очереди Q вершину $v = $ Н с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

Корректируем расстояние $D\,[u]$.

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	34.3	38.8	40.4	38.8+12=50.8	40.3

Итерация №14

Извлекаем из очереди Q вершину $v = $ С с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	34.3	38.8	40.4	50.8	40.3

Итерация №15

Извлекаем из очереди Q вершину $v = $ О с минимальным приоритетом — ближайшую к $s = 0$ вершину.

Отмечаем вершину v как посещенную (помещаем v во множество H).

n	0	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	С
D	0	8.6	8.7	18.3	27.4	15.9	17.6	21.9	31.8	18	28.4	29.3	34.3	38.8	40.4	50.8	40.3

Путь: А → Б → Г → Л → О → С. Длина пути составит: 50,8 км = 50800 м.

Ответ: 50800.

Командная часть

Инструкции по установке и проверке работы необходимого для выполнения задания программного обеспечения:

1. VMware Workstation Player 15: `https://my.vmware.com/en/web/vmware/downloads/info/slug/desktop_end_user_computing/vmware_workstation_player/15_0`.
2. Архив виртуальной машины для задач: `https://clovervm.ams3.digitaloceanspaces.com/clover-devel_v0.3.ova`.
3. launch-файлы для подготовки полигона в симуляторе — https://drive.google.com/drive/u/2/folders/1XfVBU5dRUMud3grG5YwlGt5IzWnuus6_.
4. Документация: https://clover.coex.tech/ru/simulation_vm.html.
5. Видеоинструкции: https://youtu.be/-UA9ZOUk5ws, https://www.youtube.com/watch?v=oo0VZhF4UtM&feature=youtu.b.
6. Вводный вебинар: https://youtu.be/X2OLG7oho74.

Задача II.2.1. Полет (30 баллов)

Необходимо подготовить программный код для выполнения БПЛА следующих заданий:

1. Взлететь с точки старта.
2. Распознать закодированное в QR-коде сообщение и вывести его в терминал.
3. Полететь в координату, распознанную в пункте 2.
4. Распознать закодированное в QR-коде сообщение и вывести его в терминал.
5. Полететь в координату, распознанную в пункте 4.
6. Распознать закодированное в QR-коде сообщение и вывести его в терминал.
7. Полететь в координату, распознанную в пункте 6, и приземлится на нее.

Примечание. Запишите видео полета БПЛА, демонстрирующее выполнение задания 1. На видео должен быть виден полет БПЛА от взлета до посадки, также видео должно содержать отдельное окно с выводом терминала.

Материально-техническая база. Задание можно выполнить как с использованием физического полигона и БПЛА, так и с использованием симулятора Gazebo.

Входные данные.

- Если вы используете физический БПЛА: 3 QR-кода (https://docs.google.com/document/d/1DUJOD6Eid70hvQNk_ewLI9qJa-EDUX87G6NrgneSZvA/edit), в которых закодированы координаты x и y. В закодированном сообщении координаты указаны в метрах.
  Подготовка физического полигона. Необходимо расположить QR-коды на полигоне в следующих координатах:
  1. $x = 0$, $y = 0$ (точка старта);
  2. $x = 1,5$, $y = 1,5$;
  3. $x = 1$, $y = 2$.

- Если вы работаете в симуляторе Gazebo, используйте launch-файлы для подготовки полигона в симуляторе (https://drive.google.com/drive/u/2/folders/1XfVBU5dRUMud3grG5YwlGt5IzWnuus6_).
  Полигон Gazebo. Координаты точки старта и первого QR-кода:
  1. $x = 0$, $y = 0$ (точка старта);
  2. $x = 2$, $y = 0$ (QR-код №1).

№	Критерий	Баллы за случай		Кол-во случаев
		Физ. БПЛА	Gazebo	
1.	БПЛА взлетел с точки взлета	3	2	1
2.	БПЛА распознал закодированное в QR-коде сообщение и вывел значение в терминал	6	3	3
3.	БПЛА пролетел в координату закодированную в QR-коде	2	1	3
4.	БПЛА приземлился в координату, распознанную в QR-коде №3	3	2	1

Пример программы-решения

```
# coding: utf-8
import rospy #вызов библиотеки rospy
from clover import srv #вызов библиотеки srv
from std_srvs.srv import Trigger #вызов библиотеки Trigger
from cv_bridge import CvBridge #вызов библиотеки CvBridge
from sensor_msgs.msg import Image #вызов библиотеки Image
import math #вызов библиотеки math
from pyzbar import pyzbar #вызов библиотеки pyzbar

rospy.init_node('flight') #

bridge = CvBridge() #задаем значение переменной bridge

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry) #вызов сервиса get_telemetry
navigate = rospy.ServiceProxy('navigate', srv.Navigate) #вызов сервиса navigate
land = rospy.ServiceProxy('land', Trigger) #вызов сервиса land

cord = ['0','0'] #выбор начальных координат
Z = 0.9 #задаем переменной Z значение 0.9

#init defs
def navigate_wait(x=0, y=0, z=0, yaw=float('nan'), speed=0.5, frame_id='', auto_arm=False, tolerance=0.2):
    navigate(x=x, y=y, z=z, yaw=yaw, speed=speed, frame_id=frame_id, auto_arm=auto_arm)
    while not rospy.is_shutdown():
        telem = get_telemetry(frame_id='navigate_target')
        if math.sqrt(telem.x ** 2 + telem.y ** 2 + telem.z ** 2) < tolerance:
            break
        rospy.sleep(0.2)

def land_wait(): #выполняет посадку квадрокоптера
        land() #посадка квадрокоптера
        while get_telemetry().armed: #условие
                rospy.sleep(0.2) #задержка 0.2 сек

def qr_read(cord): #выполняет считывание QR-кодов
    img = bridge.imgmsg_to_cv2(rospy.wait_for_message('main_camera/image_raw', Image), 'bgr8')  # OpenCV image
    p = pyzbar.decode(img) #задаем значение переменной p
    global pac # задаем флаг
    pac = False #значение переменной pac
    print('reading') #вывод сообщения
    if len(p) != 0: #условие длина p не равна 0
        pac = True #значение переменной pac
        for p in p: #условие p в p
                    pdata = p.data.encode("utf-8") #значение переменной pdata
        x, y = map(str, pdata.split()) #значение переменной x и y
        print(x,y)
        X = float(x[2:])
        Y = float(y[2:])
        cord = [X, Y] #значение переменной cord
    return cord #возвращение изначальной переменной

navigate(x=0, y=0, z=1.5, speed = 0.5, frame_id = 'body', auto_arm = True) #перелет квадрокоптера в определенную точку с определенной скоростью
rospy.sleep(5) #задержка 5 сек
```

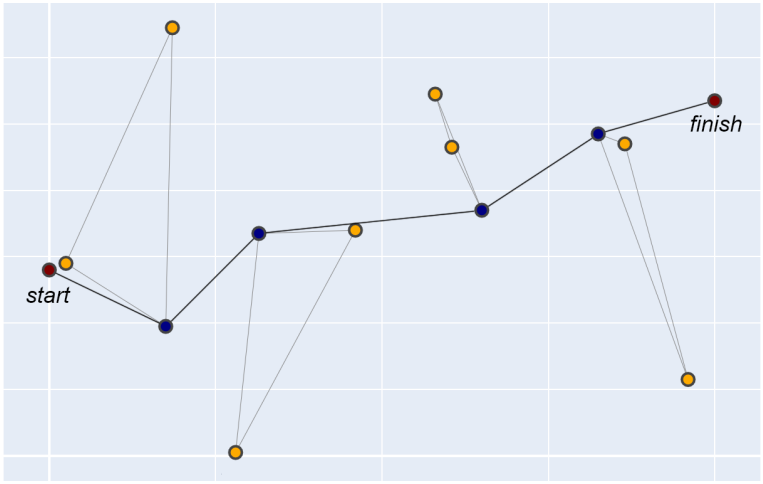
```
54
55 for i in range(4): #задаем значение для i
56     navigate_wait(x=float(cord[0]), y=float(cord[1]), z=1.5, speed = 0.5, frame_id =
       ↪  'aruco_map') #перелет квадрокоптера в определенную точку с определенной
       ↪  скоростью
57     if i != 3: #условие i не равен 3
58         for j in range(5): #задаем значение для j
59             navigate_wait(x=float(cord[0]), y= float(cord[1]), z=Z, speed = 0.5,
               ↪  frame_id = 'aruco_map') #перелет квадрокоптера в определенную точку с
               ↪  определенной скоростью
60             cord = qr_read(cord) #значение переменной cord
61             if pac: #условие если переменная pac
62                 Z = 1 #значение переменной Z
63                 pac = False #значение переменной pac
64                     break #конец операции
65             elif not pac and Z > 0.5: #условие Z больше 0.5
66                     Z -= 0.1 #вычитаем из Z 0.1 при выполнение условия
67             elif Z <= 0.5: #условие Z меньше или равен 0.5
68                         land_wait() #посадка квадрокоптера
69                         break #конец операции
70             rospy.sleep(1) #задержка 1 сек
71         if Z <= 0.5: #условие Z меньше или равен 0.5
72                 break #конец операции
73
74         print('Go to x = ', float(cord[0]), 'y = ', float(cord[1])) #вывод в какую
           ↪  клетку летит квадрокоптер по x и y
75         rospy.sleep(4) #задержка 4 секунды
76
77 land_wait()        #посадка квадрокоптера
```

Задача II.2.2. Кратчайший путь (30 баллов)

Дрону необходимо пролететь из точки `start` в точку `finish` по кратчайшему маршруту. На пути дрона встречаются препятствия, координаты которых известны и случайно генерируются перед каждым полетом.

- Препятствия представлены в виде треугольников.
- Препятствия не пересекаются и не касаются друг друга.
- Размер дрона не учитывается.
- Можно пролетать вплотную к стенам и углам препятствий.
- Высота полета не учитывается и не задается в условиях и в ответе.

Визуализация решения варианта задачи:



На вход программы поступают данные, где в первой строке указано количество последующих строк; во второй — точка `start`; в третьей — точка `finish`; в четвертой и в последующих строках — набор координат вершин препятствия (треугольника): $x1$ $y1$ $x2$ $y2$ $x3$ $y3$.

Программа должна рассчитывать и выводить координаты кратчайшего маршрута. Каждая точка маршрута указывается на новой строке. В одной строке должно быть два числа: x и y данной точки. В первой строке указывается точка `start`, в последней — `finish`.

Ответ считается верным, если:

- длина маршрута из точки `start` в точку `finish` не превышает 102% от длины кратчайшего маршрута;
- отсутствуют пересечения с препятствиями;
- первая точка соответствует точке `start`;
- последняя точка соответствует точке `finish`.

Рекомендованные библиотеки для Python: networkx для операций с графами, sympy для геометрических операций.

Примеры

Пример №1

Стандартный ввод
6 0 98 200 40 10 43 32 139 10 126 91 76 58 102 92 31 148 3 118 39 127 36 153 84 154 127 194 51

Стандартный вывод
0 98 10 43 92 31 118 39 200 40

Решение

Для решения задачи необходимо применить алгоритм поиска кратчайшего пути между двумя вершинами графа.

Граф можно получить из препятствий, точки старта и финиша. Для этого набора объектов необходимо составить граф видимости, чтобы понять, из каких вершин в какие может перемещаться коптер. Оптимальный маршрут будет обязательно проходить через вершины, причем между вершинами необходимо двигаться по прямой.

Для корректного составления графа видимости необходимо проложить ребро через каждую пару вершин и проверить, не пересекает ли оно препятствия.

Найти кратчайший маршрут можно при помощи алгоритма Дейкстры (`Dijkstra`), представленного в библиотеке `networkx`.

Пример программы-решения

Ниже представлено решение на языке Python.

```
import math
from sympy import Segment2D, Polygon
import networkx as nx

def tri_seg_intersection(triangle, segment):
    inter = triangle.intersection(segment)
    if len(inter) == 0:
        return False
    if len(inter) == 2:
        return True
    if isinstance(inter[0], Segment2D):
        return False

    if len(inter) == 1:
        if triangle.encloses(segment.p1) or triangle.encloses(segment.p2):
            return True
        else:
            return False

def get_edge_distance(edge):
    return math.sqrt((edge[0][0] - edge[1][0]) ** 2 + (edge[0][1] - edge[1][1]) **2)

def solve(dataset):
    #dataset = dataset.splitlines()
    start = tuple(map(int, dataset[0].split()))
    finish = tuple(map(int, dataset[1].split()))

    dataset = dataset[2:]

    # turn string of numbers into list of sets with coordinates in each element of
    ↪  dataset
    shapes = []
    for line in dataset:
        line = list(map(int, line.split()))
        line = [(line[0], line[1]), (line[2], line[3]), (line[4], line[5])]
        shapes.append(line)

    # convert lists of points to sympy Polygons
```

```
    obstacles = []
    for shape in shapes:
        obstacles.append(Polygon(*shape))


    # extract points from shapes
    points = []
    for shape in shapes:
        for point in shape:
            points.append(point)

    points.append(start)
    points.append(finish)

    # create visibility graph edges
    vis_edges = []
    for i in range(len(points)):
        for j in points[i+1:]:
            seg = Segment2D(points[i], j)
            no_intersection = True
            for obs in obstacles:
                if tri_seg_intersection(obs, seg):
                    no_intersection = False
                    break
            if no_intersection:
                vis_edges.append((points[i], j))

    # create graph from shapes
    G1 = nx.Graph()
    G1.add_edges_from(vis_edges)

    # calculate edge weight
    for edge in G1.edges:
        G1.add_edge(*edge, weight=get_edge_distance(edge))

    # calculate shortest path and shortest distance using Networkx
    shortest_path = nx.dijkstra_path(G1, start, finish, weight="weight")

    result = ''
    for i in shortest_path:
        result += '{} {}'.format(i[0], i[1]) + '\n'
    print(result)

n = int(input())
inp = []
for i in range(n):
    inp.append(input())

#print(inp)
#se = Segment((0,2),(3,3))

solve(inp)
```