Python, pioneersdk, geobotsdk
Конкурсное задание

Лукашова Ирина

Архипелаг 2024.

План нашей работы

План по дням

1. Теоретическое обучение (python, pioneer sdk, geobot sdk).

2. Теоретическое обучение (ортофотопланы, обучение нейронной сети). Разметка данных.

3. Выполнение конкурсного задания.

4. Тестовая попытка запуска. Время на доработку программ.

5. Зачетная попытка. Подведение итогов, награждение.

Конкурсное задание

Последовательность попытки

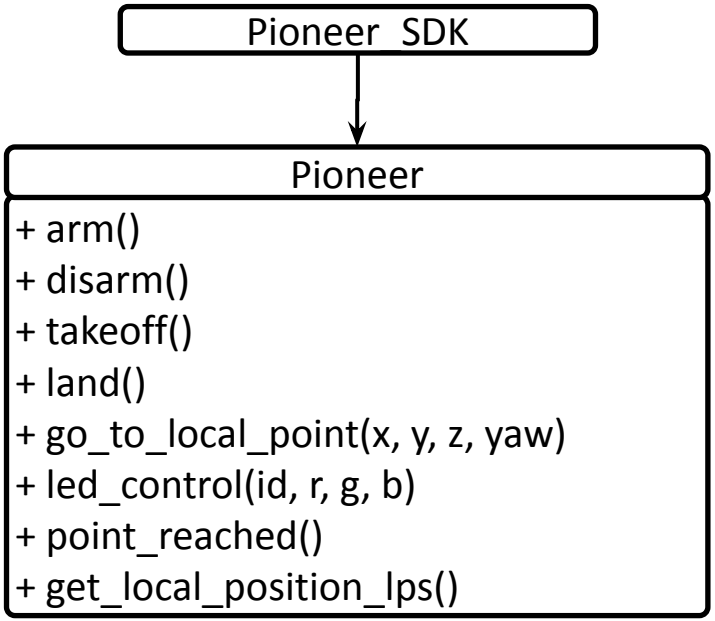
- Съемка набора снимков коптером в полете по траектории (параметры траектории задаются участниками). Ограничение по времени - 10 минут.
- Распознавание объекта интереса на снимках и получение его точных координат.
- Автономное построение траектории наземным роботом и поездка в необходимую локацию. Ограничение по времени - 5 минут.

Содержание

PioneerSDK

Программирование.
Pioneer_SDK.
Описание работы

Программирование. Pioneer_SDK. Описание классов



Написание программ с использованием piosdk требует придерживаться простой идеи – создать объекты, а затем вызывать их методы.

Основные функции

arm()	Завести моторы
disarm()	Заглушить моторы
takeoff()	Взлет
land()	Посадка
go_to_local_point(x, y, z)	Полёт в точку с указанными координатами
point_reached() -> bool	Достигнута ли предыдущая заданная точка
get_local_position_lps() -> [x, y, z]	Получить текущие координаты коптера
emergency_detection()	Сигнал об обнаружении объекта интереса

Основы работы с sdk

Некоторые операции не выполняются мгновенно, например, взлет и посадка - как только соответствующая команда отдана коптеру, программа продолжает выполнение.

```
drone = Pioneer()
drone.arm()
drone.takeoff()
time.sleep(2)
```

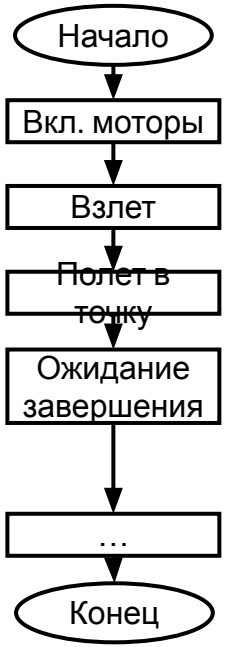
Пример полетного задания

```python
from piosdk import Pioneer

drone = Pioneer()

drone.arm()
drone.takeoff()
drone.go_to_local_point(0, 1)
while not drone.point_reached():
    pass
drone.land()
drone.disarm()
```

Важно!

Проверять статус полета в точку с помощью point_reached(). Этот метод возвращает True только один раз и только тогда, когда последняя отправленная точка была достигнута.

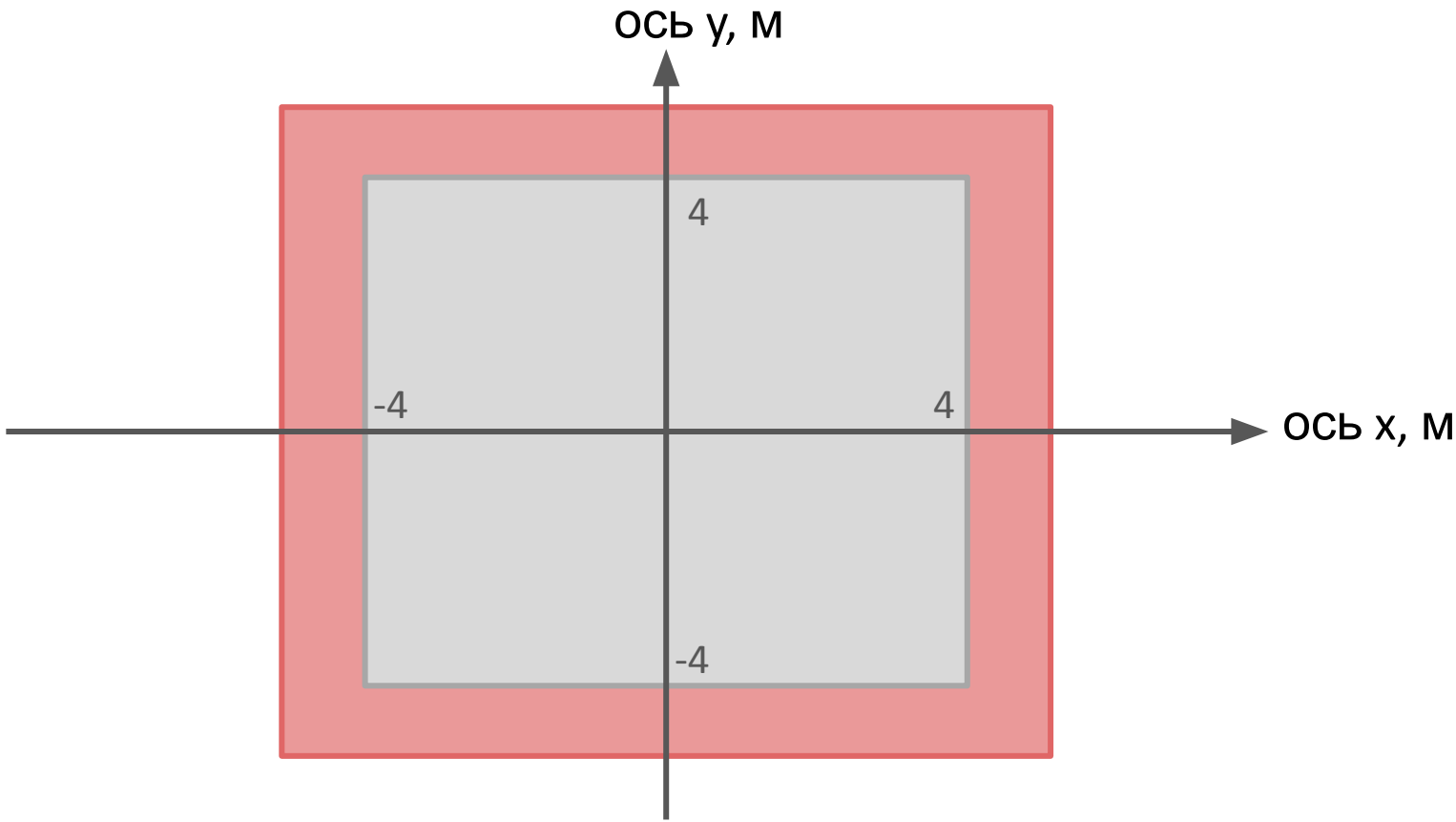
Пример полетного задания



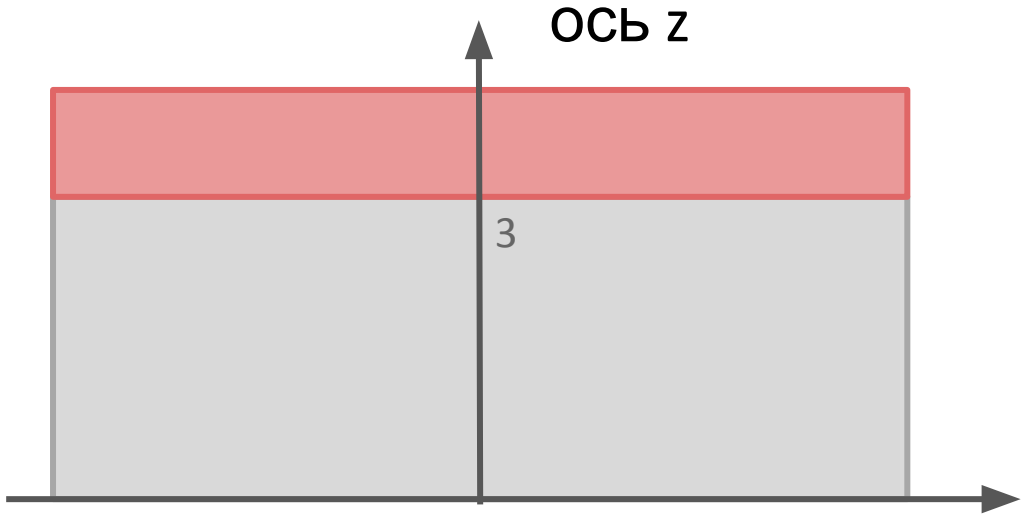

```python
import time
from piosdk import Pioneer

drone = Pioneer()
drone.arm()
drone.takeoff()
time.sleep(2)

drone.go_to_local_point( x: 1,  y: 1,  z: 1.3,  yaw: 0)
while not drone.point_reached():
    time.sleep(0.1)
print("point is reached")

drone.land()
time.sleep(2)
drone.disarm()
```

Ограничения полигона

Если смотреть сверху, рабочая зона полигона имеет размеры 8x8 метров:



Ограничения полигона



Высота рабочей зоны составляет 3 метра.

None-значение

В некоторых случаях функции piosdk возвращают None. Если не проверить полученное значение и попытаться произвести с ним какие-то действия (например, математические), программа может выдать ошибку, и ее выполнение остановится.

```
def get_local_position_lps(self, blocking=False):
    """ Возвращает данные от системы навигации LPS """
    position = self.__mavlink_socket.recv_match(type='LOCAL_POSITION_NED', blocking=blocking,
                                                timeout=self.__ack_timeout)
    if not position:
        return None
    if position.get_type() == "BAD_DATA":
        if mavutil.all_printable(position.data):
            sys.stdout.write(position.data)
            sys.stdout.flush()
    else:
        if position._header.srcComponent == 26:
            return [position.x, position.y, position.z]
        else:
            return None
```

Проверка на None

Самый простой способ - проверить, что метод вернул не None, и тогда уже производить какие-то действие.

```
while True:
    position = drone.get_local_position_lps()
    if position is not None:
        print(position[0], position[1], position[2])
```

Обработка исключения

Альтернативный способ - обработать исключение. Однако в большинстве случаев проверки хватает.

```
while True:
    position = drone.get_local_position_lps()
    try:
        print(position[0], position[1], position[2])
    except TypeError:
        print("Получено значение None")
```

Особенности полета в точку

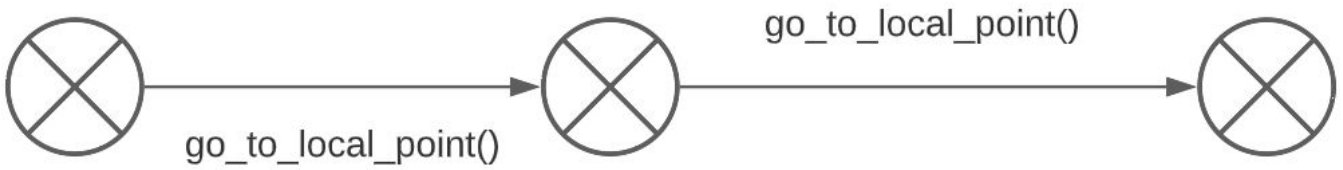
В идеальном мире траектория выглядела бы так:

В реальности скорее так:

Что делать?

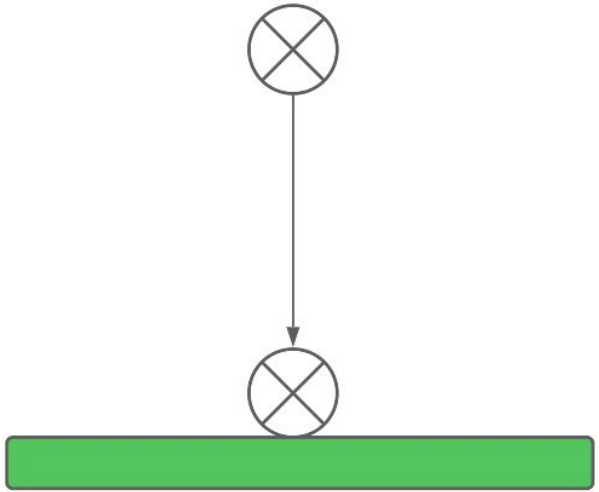
Особенности полета в точку

Траектория движения станет более прямой, если добавить промежуточную точку:

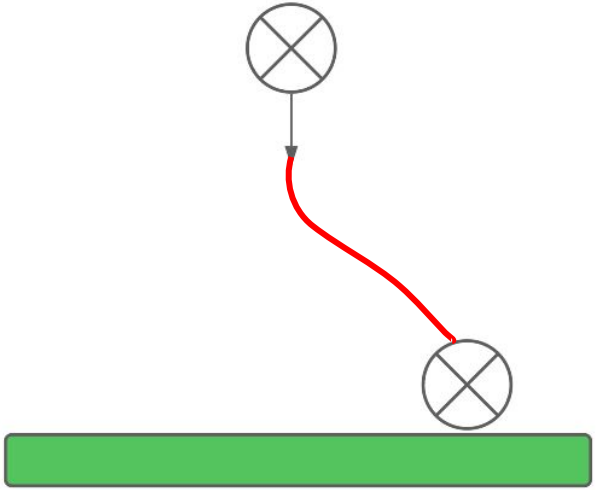


Особенности посадки

Ожидание

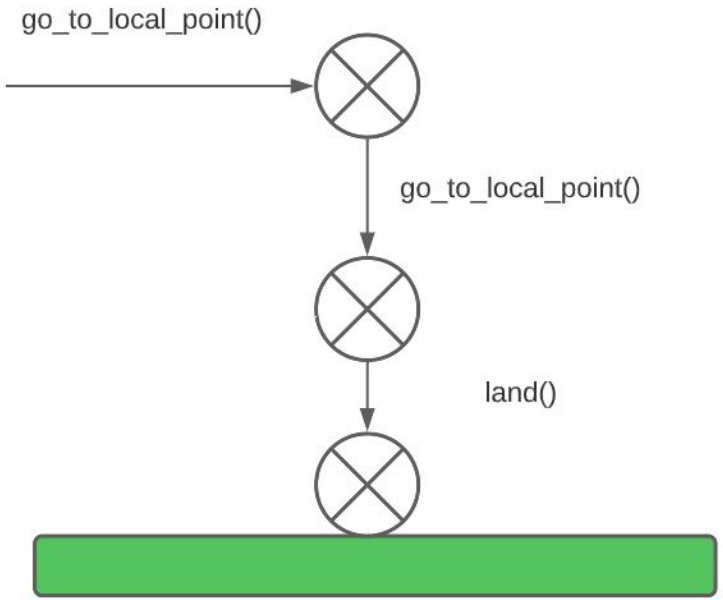


Реальность



Особенности посадки

Добавим промежуточную точку, чтобы снизиться и сесть именно туда, куда хотели.



А причем здесь потоки?

Если написать программу таким образом, она будет блокироваться во время движения одного коптера, и другой будет простаивать.

```
drone.go_to_local_point(0, 1)
drone2.go_to_local_point(1, 0)
while not drone.point_reached():
    pass
while not drone2.point_reached():
    pass
```

Многопоточность

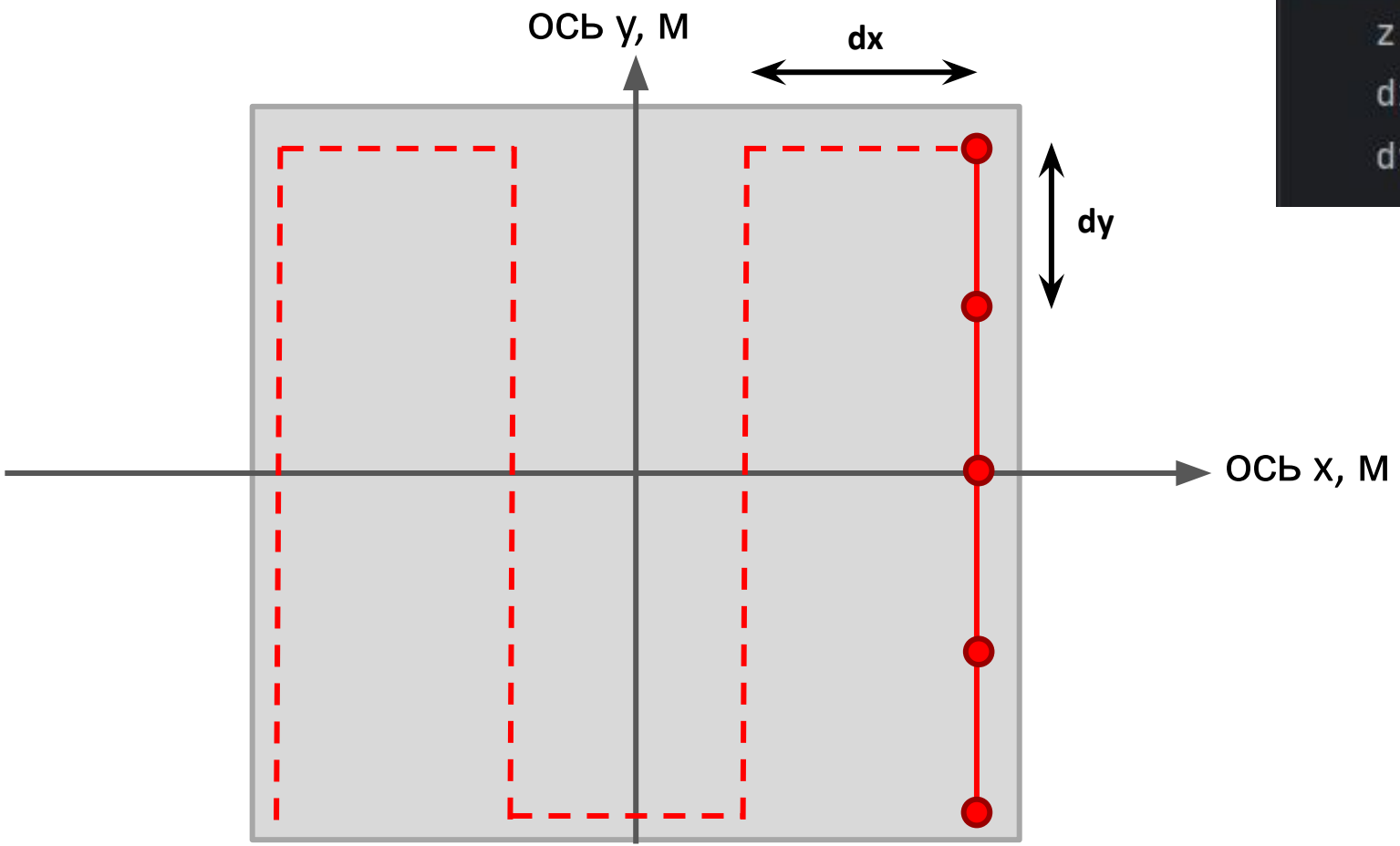
```python
def drone_task():
    drone.arm()
    drone.takeoff()
    drone.go_to_local_point(0, 1)
    while not drone.point_reached():
        pass
    drone.land()
    drone.disarm()
```

```python
thread1 = threading.Thread(target=drone_task_1)
thread2 = threading.Thread(target=drone_task_2)
thread1.start()
thread2.start()
```

Полетное задание для каждого из коптеров можно вынести в функцию и запустить эту функцию отдельным потоком.

В рамках конкурсного задания

```
class Params:
    z = 1.5
    dx = 1
    dy = 1
```



GeobotSDK

GeobotSDK

Общая суть GeobotSDK совпадает с PioneerSDK, за исключением того, что теперь мы управляем не коптером, а роботом. Нам нет необходимости выполнять взлет и посадку, а также заводить и глушить двигатели.

go_to_local_point(x, y, z)	Поездка в точку с указанными координатами
point_reached() -> bool	Достигнута ли предыдущая заданная точка
get_local_position_lps() -> [x, y, z]	Получить текущие координаты робота

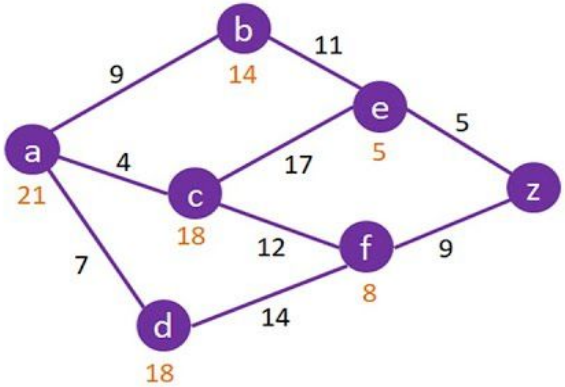
GeobotSDK: пример использования

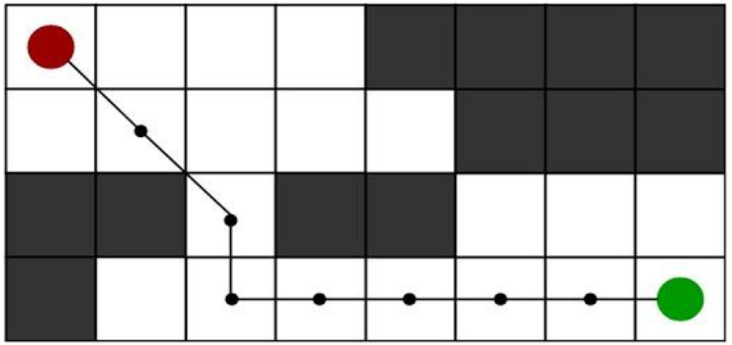
```
robot = geobot_sdk.GeobotClient()
robot.go_to_local_point(-2, y: 2)
while not robot.point_reached():
    pass
print("the point is reached")
```

Алгоритм для построения траектории A*

Map

Geobotsdk включает в себя класс Map, который представляет полигон в виде графа, и с помощью него ищет оптимальный маршрут с учетом объезда препятствий.





Map

Пример использования класса Map. Мы создаем разбиение на участки (create_map), добавляем местоположение препятствий (add_block), и можем получить траекторию

```
MAP_BLOCK_LIST = [
    (-3.9, 0.7),
    (-2.4, -1.15),
    (-2.8, 0.5),
    (-2.6, 0.5),
    (-2.65, 1.6)
]

m = Map()
m.create_map( num_node: 22,  height: 11,  weight: 11)

for blockPoint in MAP_BLOCK_LIST:
    m.add_block(Point(blockPoint[0], blockPoint[1]))

startPoint = Point(-4, -4)
endPoint = Point(0, 0)

tr = m.get_trajectory(startPoint, endPoint)
print(tr)
```

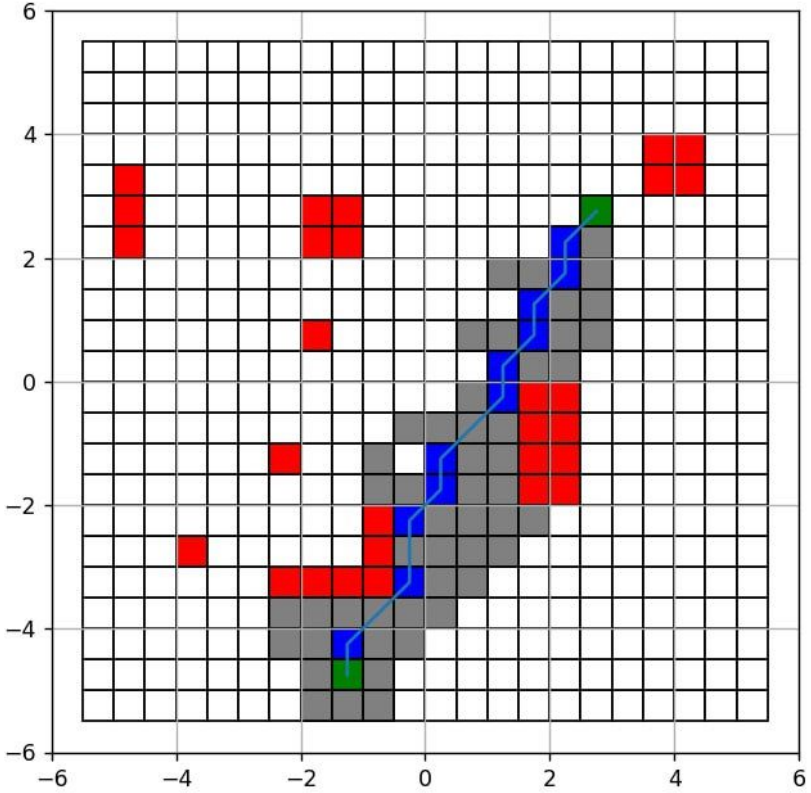
метод create_map()

Принимает в себя три аргумента:

- num_node - количество равных отрезков, на которые будет разбита сторона карты
- height - длина карты в метрах
- width - ширина

```
MAP_BLOCK_LIST = [
    (-3.9, 0.7),
    (-2.4, -1.15),
    (-2.8, 0.5),
    (-2.6, 0.5),
    (-2.65, 1.6)
]

m = Map()
m.create_map( num_node: 22,  height: 11,  weight: 11)

for blockPoint in MAP_BLOCK_LIST:
    m.add_block(Point(blockPoint[0], blockPoint[1]))

startPoint = Point(-4, -4)
endPoint = Point(0, 0)

tr = m.get_trajectory(startPoint, endPoint)
print(tr)
```

Map



Пример карты с готовым маршрутом.

Красным цветом обозначены препятствия, синим - точки маршрута, серым - узлы, которые алгоритм рассматривал, но в итоге отбросил.

Если препятствие большое по площади, можно добавить несколько точек.

Вопросы