Командный практический тур

Сроки проведения заключительного этапа (финала) профиля: с 22 марта по 26 марта 2021 г.

Задача командного тура заключительного этапа — транспортировка легкого и негабаритного груза по складу в автономном режиме при помощи беспилотного летательного аппарата (БПЛА), включая облет препятствий и распознавание объектов.

Формат проведения — распределенный.

ЦУП — это испытательная площадка, в которой располагаются БПЛА и оборудование. Площадка контролируется экспертами. Из центра ведется трансляция полетной зоны. Обратная связь с участниками осуществляется посредством видеозвонков в Zoom и мессенджеров Telegram и Discord.

Команды в режиме реального времени, дистанционно от ЦУП, пишут программы для запуска и управления БПЛА и получают доступ к тестированию своих программ на реальном оборудовании. Каждая команда имеет фиксированное время взаимодействия с оборудованием. Участники одной команды из разных регионов имеют возможность одновременного подключения к ЦУП.

1. Команда работает над решением задачи командного тура заключительного этапа удаленно и передает решение в ЦУП.
2. Эксперты запускают программный код автономной миссии БПЛА на физическом полигоне.
3. Команды отслеживают результаты своей удаленной работы в режиме реального времени и оперативно вносят изменения в решение командной задачи заключительного этапа.

Описание задачи и системы оценки

Информационные разработки и научно-технический прогресс в складской логистике имеют большое значение для повышения производительности выполнения складских операций. Исходя из этого тематика задачи была связана со складской логистикой, а именно транспортировкой легкого и негабаритного груза по складув автономном режиме при помощи БПЛА, включая облет препятствий и распознавание объектов.

Так, на финале участникам предлагалось пошагово проработать решение для БПЛА и используя БПЛА.

Задание состояло из двух частей.

Часть 1. Программная часть

Команда получала задачу в виде отдельных заданий на каждый день. Задача состояла в разработке решений для выполнения автономной миссии БПЛА.

Схема полигона

Координаты Qr-кода: $x = 0.4, y = 0.8$.

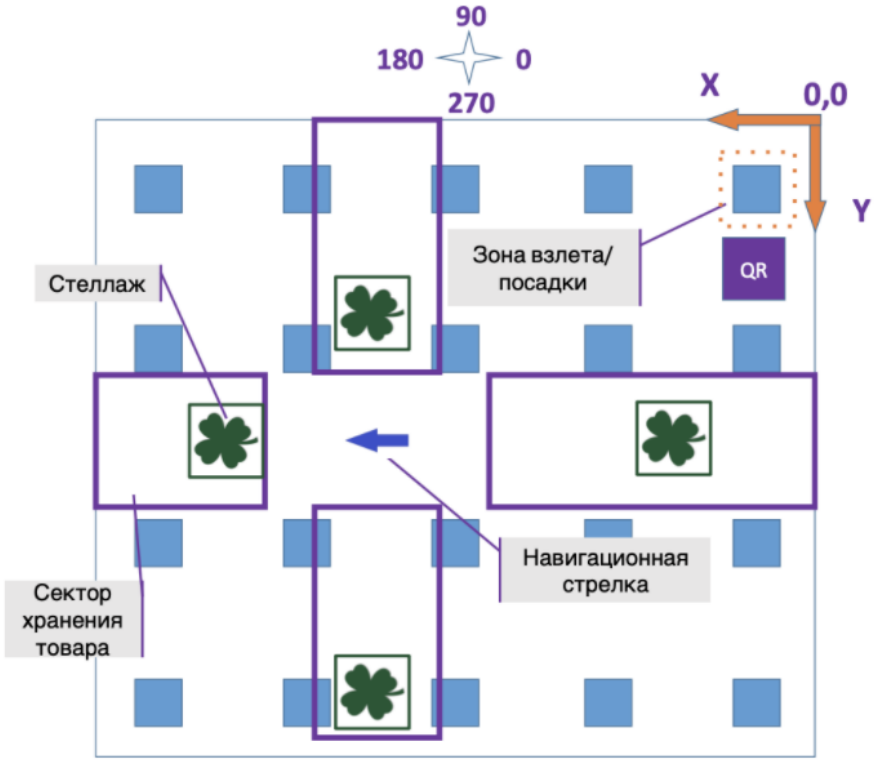


Рис. III.2.1: Пример полигона. Задание 3 (*вид с веб-камеры обеспечивающей трансляцию)

Для успешного выполнения задания участникам необходимо написать программный код, в котором БПЛА необходимо выполнить следующие задания:

1. **Взлет**:
   - Совершить взлет с зоны взлет. Положение постоянно, известно.
   - Начать запись в файл Report.txt значений телеметрии коптера.
2. **QR-код**:
   - Распознать QR-код, в котором закодирована информация о местоположении препятствий и навигационной стрелки. Формат закодированных данных:

Общий вид содержание QR-кода	**Пример содержание QR-кода**
x y x y X Y N	1 2 3.5 4.1 5.1 5 4 
Где **x** и **y** — координаты препятствий, **X** и **Y** — координата навигационной стрелки, **N** — номер заказа.	

***Препятствие** — колонна с известными координатами центра и радиусом 20 см (физически не существует).*

- Вывести в терминал сообщение о результатах распознавания в виде:

Общий вид формата вывода	**Пример вывода**
Column area x=x, y=y Navigation area x=X, y=Y Order number: N	Column area x=1, y=2 Column area x=3.5, y=4.1 Navigation area x=5, y=5 Order number: 4
Где **x** и **y** — координаты препятствий, **X** и **Y** — координата навигационной стрелки, **N** — номер заказа.	

- Произвести обводку Qr-кода в топике «Detect».
- Пролететь к стеллажу, совершая облет препятствий.

3. **Навигационная стрелка**:
   - Обнаружить навигационную стрелку.
   - Распознать положение навигационной стрелки, ее цвет и угол поворота. Размер стрелки (лист А3), четыре возможных положения, 0, 90, 180, 270 градусов.

Поворот стрелки	**Сектор хранения товара**	**Возможные положения**
up	Sector A	270
down	Sector B	90
right	Sector C	180
left	Sector D	0

   - Цвет навигационной стрелки — высота стеллажа, цвет индикации варьируются в соответствии с таблицей.

Цвет навигационной стрелки	**Высота склада**	**Цвет индикации**
yellow	0.25	yellow
black	0.5	green
blue	0.75	blue
red	1	red

Рис. III.2.2: Пример навигационной стрелки

- Вывести в терминал сообщение о результатах распознавания в виде:

Общий вид	**Пример**
n required	**Sector A** required
Где **n** — название сектора размещения стеллажа	

- Произвести обводку навигационной стрелки в топике «Detect».

4. **Доставка груза**:
   - Произвести полет в соответствии с направлением навигационной стрелки.
   - Обнаружить необходимый стеллаж.
   - Включить световую индикацию в соответствии с цветом навигационной стрелки.
   - Доставить груз на стеллаж (совершить посадку БПЛА на стеллаж).
   - Вывести в терминал сообщение о результатах доставки в виде:

Общий вид	**Пример**
N delivered in **n**	**3** delivered in **Sector A**
Где **N** — номер заказа, **n** — название сектора размещения стеллажа.	

5. **Посадка**:
   - Произвести полет в зону «Старт».
   - Произвести посадку на зону «Старт».
   - Закончить запись в файл Report.txt значений телеметрии коптера.
   - Сформировать автоматический отчет и вывести в терминал сообщение о доставке в виде:

Общий вид	**Пример**
N delivered in **n** for **t**	**3** delivered in **Sector A** for **4 min 30 sec**
Где **n** — название сектора размещения стеллажа, **N** — номер заказа, **t** — время с момента взлета до посадки.	

   - При распознавании объектов (QR-код, навигационная стрелка) необходимо выделить распознанный объект обводкой. Ширина обводки 3 пикселя, цвет обводки розовый (255,105,180), обработанное изображение записывается в топик «Detect».

6. **Облет препятствий**:
   - Облет препятствий (без столкновений).
   - Формирование корректного отчета полета.

**При построении маршрута необходимо учитывать габариты коптера (40×40 см) и радиус препятствий.*

В таблице представлены критерии по оцениванию успешности прохождения миссии.

№	Критерий (условия выполнения миссии)	Баллы за один случай	Кол-во случаев	Баллы за все случаи
1. Взлет *(оценивается во время полета)*				
1.1	БПЛА взлетел в пределах зоны «Старт».	1	1	1
2. QR-код *(оценивается во время полета, результат выводится в терминале).*				
2.1	QR код распознан и его значение выведено корректно (см. пример), соответствующее действительности сообщение о содержании QR кода.	3	1	3
2.2	Выделение QR кода в топике «Detect».	1	1	1
3. Навигационная стрелка *(оценивается во время полета)*				
3.1	Положение навигационной стрелки распознано корректно, в терминал выведено корректное сообщение о результатах распознавания.	10	1	10
3.2	Совершена обводка навигационной стрелки в топике «Detect».	2	1	2
3.3	Световая индикация совпадает с цветом навигационной стрелки.	2	1	2
3.4	Длительность корректной световой индикации 5 сек.	2	1	2
4. Доставка груза (оценивается во время полета)				
4.1	Полет был совершен в соответствии с направлением навигационной стрелки.	4	1	4
4.4	Совершена посадка на стеллаж без падений (опорные элементы не выходят за пределы стеллажа).	6	1	6
4.6	Выведено корректное сообщение о доставке (см. пример).	2	1	2
5. Посадка *(оценивается во время полета).*				
5.1	БПЛА совершил посадку в обозначенной зоне (опорные элементы не выходят за пределы зоны посадки/старта).	1	1	1
5.2	Отчет выведен в терминал по окончанию миссии (в процессе полета отчет оцениваться не будет).	1	1	1
5.3	Финальный отчет записан корректно (см. пример).	2	1	2
6. Облет препятствий *(оценивается после полета).*				
6.1	Облет препятствия на протяжении всей попытки (без столкновений	2	6	12
6.3	Код финальной зачетной попытки с комментариями (Github).	3	1	3
Итого за решение задачи				52

Часть 2. Инженерная часть.

1. **Gazebo**
   1.1. Создание сцены финальной зачетной попытки в симуляторе Gazebo, сцена должна повторять полигон ЦУПа (финальная зачетная попытка) и содержать:
      - Карта Aruco маркеров;
      - Стеллажи;
      - Qr-код;
      - Навигационную стрелка.
2. **Rviz**
   2.1. Записать все топики на финальной попытке.
   2.2. Визуализация финальной зачетной попытки при помощи инструмента Rviz (в качестве reference frame рекомендуется установить фрейм map):
      - /aruco_map/visualization
      - /wp_markers
      - Axes aruco_map
      - /main_camera/image_raw
      - /Detect
      - /vehicle_marker
      - /rangefinder/range
   2.3. Сделать запись экрана с демонстрацией работы rviz, формат видео *.mp4 или *.avi.
   2.4. Отчетные материалы отправляются в формате ссылки на Google Drive и должны содержать:
      - video.mp4/video.avi;
      - fly.bag;
      - config_rviz.rviz.

Так, команда могла набрать максимально 166 баллов: 39 баллов за инженерное задание; Задание день 1 — 31 балл; Задание день 2 — 44 балла; Задание день 3 — 52 балла.

Для ознакомления и погружения участников в проблематику были подготовлены ознакомительные статьи и видео, в которых содержались небольшие части решения от разработчиков трека. Участники могли использовать эти части, если правильно адаптировали их к своим условиям.

- Статья про компьютерное зрение:
  https://clever.coex.tech/ru/camera.html
- Лекция про автономные полеты:
  https://www.youtube.com/watch?v=ThXiNG1IzvI
- Заметки по работе с OpenCV через Python:
  https://github.com/sfalexrog/coex_kb/blob/master/kb014_opencv_python.md

Часть 1. Решение задачи по программированию квадрокоптера (выполнение миссии)

Для автономного полета использовался API модуля simple_offboard (https://clover.coex.tech/ru/simple_offboard.html).

Для распознавания QR-кодов использовалась библиотека pyZBar. Она уже установлена в последнем образе для Raspberry Pi.

Для реализации алгоритмов компьютерного зрения рекомендовалось использовать предустановленную на образ библиотеку OpenCV.

Пример создания подписчика и публикатора на топик с изображением с основной камеры для обработки с использованием OpenCV:

```
# -*- coding: utf-8 -*-
import cv2
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from clover import srv
from clover.srv import SetLEDEffect

class Main_Config():
    def __init__(self):
        rospy.init_node('detecting', anonymous=True)
        self.image_pub = rospy.Publisher("Detect",Image,queue_size=10)
    self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("main_camera/image_raw",Image,self.callback)

  def callback(self,data):
        try:
            img = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except:pass
        try:
            self.image_pub.publish(self.bridge.cv2_to_imgmsg(img, "bgr8"))
        except CvBridgeError as e:
            print(e)
```

Получаемые изображения можно просматривать, используя `web_video_server`. По умолчанию `web_video_server` показывает изображение из топиков со сжатием (например, `/main_camera/image_raw/compressed`). Ноды на Python не публикуют такие топики, исходя из чего для их просмотра следует добавлять `&type=mjpeg` в адресную строку страницы `web_video_server` или изменить параметр `default_stream_type` на `mjpeg` в файле `clever.launch`.

При использовании русских букв в скрипте на языке программирования Python2 в начало программы следует добавить следующую строку:

```
# -*- coding: utf-8 -*-
```

Пример программного кода для выполнения миссии Первого оценочного дня (`Day_1.py`):

```
# -*- coding: utf-8 -*-
# Обязательная конструкция для работы с кириллицей
```

```
3
4  # Инициализация библиотек
5  import rospy
6  import time
7  from clover import srv
8  from std_srvs.srv import Trigger
9  # Инициализация класса со всеми функциями
10 from Main_Config import Main_Config
11 main = Main_Config()
12 # Основные данные по заказу и его транспортировке
13 main.zz = 0.5
14 main.order = 3
15
16 # Инициализация сервисов
17 get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
18 navigate = rospy.ServiceProxy('navigate', srv.Navigate)
19 navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
20 set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
21 set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
22 set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
23 set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
24 land = rospy.ServiceProxy('land', Trigger)
25 # Засекаем время полета
26 start_t = time.time()
27
28 # Взлетаем с площадки взлета
29 print navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
30 rospy.sleep(2)
31 # Подлетаем к навигационной стрелке
32 main.navigate_wait(x=0.8, y=1.2, z=0.5, frame_id='aruco_map')
33 # Начинаем распознавать положение навигационной стрелки
34 main.Arrow = True
35 rospy.sleep(2)
36
37 # Летим в секторе, пока не найдем стеллаж
38 main.sect_fly()
39 rospy.sleep(20)
40 # Проверяем распознали ли мы стеллаж
41 if main.cx == -1:
42     # Если нет, то выводим соответствующее сообщение
43     print("Dronpoint don't detecting")
44     # Производим полет в зону "Старт"
45     main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
46     # Производим посадку
47     land()
48     # Выводим данные по выполненной задаче
49     print("{} don't delivered in {} for {} min {} sec".format(main.order, main.arrow,
       ↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
50     # Выходим из программы
51     exit()
52 # Производим посадку на стеллаж, доставку груза
53 main.land_on_dronpoint()
54
55 # Взлетаем со стеллажа
56 print navigate(x=0, y=0, z=0.5, speed=0.5, frame_id='body', auto_arm=True)
57 rospy.sleep(2)
58 # Производим полет в зону "Старт"
59 main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
60
61 # Производим посадку
```

```
land()
# Выводим данные по выполненной задаче
print('{} delivered in {} for {} min {} sec'.format(main.order, main.arrow,
↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
```

Пример программного кода для выполнения миссии Второго оценочного дня (Day_2.py):

```
# -*- coding: utf-8 -*-
# Обязательная конструкция для работы с кириллицей

# Инициализация библиотек
import rospy
import time
from clover import srv
from std_srvs.srv import Trigger
# Инициализация класса со всеми функциями
from Main_Config import Main_Config
main = Main_Config()
# Основные данные по транспортировке заказа
main.zz = 0.75

# Инициализация сервисов
get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
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)

# Засекаем время полета
start_t = time.time()

# Взлетаем с площадки взлета
print navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
rospy.sleep(2)
# Подлетаем к Qr-коду
main.navigate_wait(x=0.4, y=0.8, z=0.5, frame_id='aruco_map')
# Начинаем распознавать содержание Qr-кода
main.Qr = True
rospy.sleep(2)
# Проверяем распознали ли мы содержание Qr-кода
if main.order == -1:
    # Если нет, то пробуем еще раз распознавать содержание Qr-кода, но на другой
    ↪  высоте
    main.Qr = True
    main.navigate_wait(x=0.4, y=0.8, z=0.7, frame_id='aruco_map')
    # Проверяем распознали ли мы содержание Qr-кода
    if main.order == -1:
        # Если не распознали содержание Qr-кода, то выводим соответствующее сообщение
        print("Qr don't detecting")
        # Производим полет в зону "Старт"
        main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
        # Производим посадку
        land()
        # Выходим из программы
        exit()
# Подлетаем к навигационной стрелке, облетая препятствия, записанные в Qr-коде
```

```
main.navigate_avoidece([0.4,0.8], main.nav)
main.navigate_mas()
main.navigate_wait(x=main.nav[0], y=main.nav[1], z=0.5, frame_id='aruco_map')
# Начинаем распознавать положение навигационной стрелки
main.Arrow = True
rospy.sleep(2)

# Облетаем сектор, пока не найдем стеллаж
main.sect_fly()
rospy.sleep(20)
# Проверяем распознали ли мы стеллаж
if main.cx == -1:
    # Если нет, то выводим, что не нашли стеллаж
    print("Dronpoint don't detecting")
    # Производим полет в зону "Старт"
    main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
    # Производим посадку
    land()
    # Выводим данные по выполненной задаче
    print("{} don't delivered in {} for {} min {} sec".format(main.order, main.arrow,
    ↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
    # Выходим из программы
    exit()
# Производим посадку на стеллаж, доставку груза
main.land_on_dronpoint()
# Взлетаем со стеллажа
print navigate(x=0, y=0, z=0.5, speed=0.5, frame_id='body', auto_arm=True)
rospy.sleep(2)

# Производим полет в зону "Старт"
main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')

# Производим посадку
land()
# Выводим данные по выполненной задаче
print('{} delivered in {} for {} min {} sec'.format(main.order, main.arrow,
↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
```

Пример программного кода для выполнения миссии Третьего оценочного дня (Day_3.py):

```
# -*- coding: utf-8 -*-
# Обязательная конструкция для работы с кириллицей

# Инициализация библиотек
import rospy
import time
from clover import srv
from std_srvs.srv import Trigger
# Инициализация класса со всеми функциями
from Main_Config import Main_Config
main = Main_Config()

# Инициализация сервисов
get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
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)
```

```
20 set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
21 land = rospy.ServiceProxy('land', Trigger)
22
23 # Засекаем время полета
24 start_t = time.time()
25
26 # Взлетаем с площадки взлета
27 print navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
28 rospy.sleep(2)
29 # Подлетаем к Qr-коду
30 main.navigate_wait(x=0.4, y=0.8, z=0.5, frame_id='aruco_map')
31 # Начинаем распознавать содержание Qr-кода
32 main.Qr = True
33 rospy.sleep(2)
34 # Проверяем распознали ли мы содержание Qr-кода
35 if main.order == -1:
36     # Если нет, то пробуем еще раз распознавать содержание Qr-кода, но на другой
       ↪  высоте
37     main.Qr = True
38     main.navigate_wait(x=0.4, y=0.8, z=0.7, frame_id='aruco_map')
39     # Проверяем распознали ли мы содержание Qr-кода
40     if main.order == -1:
41         # Если не распознали содержание Qr-кода, то выводим соответствующее сообщение
42         print("Qr don't detecting")
43         # Производим полет в зону "Старт"
44         main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
45         # Производим посадку
46         land()
47         # Выходим из программы
48         exit()
49 # Подлетаем к навигационной стрелке, облетая препятствия, записанные в Qr-коде
50 main.navigate_avoidece([0.4,0.8], main.nav)
51 main.navigate_mas()
52 main.navigate_wait(x=main.nav[0], y=main.nav[1], z=0.5, frame_id='aruco_map')
53 # Начинаем распознавать цвет навигационной стрелки
54 main.Color = True
55 rospy.sleep(2)
56 # Начинаем распознавать положение навигационной стрелки
57 main.Arrow = True
58 rospy.sleep(2)
59
60 # Облетаем сектор, пока не найдем стеллаж
61 main.sect_fly()
62 rospy.sleep(20)
63 # Проверяем распознали ли мы стеллаж
64 if main.cx == -1:
65     # Если нет, то выводим, что не нашли стеллаж
66     print("Dronpoint don't detecting")
67     # Производим полет в зону "Старт"
68     main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')
69     # Производим посадку
70     land()
71     # Выводим данные по выполненной задаче
72     print("{} don't delivered in {} for {} min {} sec".format(main.order, main.arrow,
       ↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
73     # Выходим из программы
74     exit()
75 # Производим посадку на стеллаж, доставку груза
76 main.land_on_dronpoint()
77 # Взлетаем со стеллажа
```

```
print navigate(x=0, y=0, z=0.5, speed=0.5, frame_id='body', auto_arm=True)
rospy.sleep(2)

# Производим полет в зону "Старт"
main.navigate_wait(x=0, y=0, z=0.5, frame_id='aruco_map')

# Производим посадку
land()
# Выводим данные по выполненной задаче
print('{} delivered in {} for {} min {} sec'.format(main.order, main.arrow,
↪  (time.time() - start_t) // 60, (time.time() - start_t) % 60))
```

Пример программного кода содержащий, функции для выполнения миссии всех дней (`Main_Config.py`):

```
# -*- coding: utf-8 -*-
# Обязательная конструкция для работы с кириллицей

# Инициализация библиотек
import numpy as np
import math
import cv2
import time
#import roslib
import sys
import rospy
from pyzbar import pyzbar
from sensor_msgs.msg import Image
import threading
from cv_bridge import CvBridge, CvBridgeError
from mavros_msgs.srv import CommandBool
from clover import srv
from std_srvs.srv import Trigger
from sensor_msgs.msg import Range
from clover.srv import SetLEDEffect

# Инициализация сервисов
get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
arming = rospy.ServiceProxy('mavros/cmd/arming', CommandBool)
land = rospy.ServiceProxy('land', Trigger)
set_effect = rospy.ServiceProxy('led/set_effect', SetLEDEffect)

#
class Main_Config():
    def __init__(self, simulator=False):
        # Создаем ноду
        rospy.init_node('detecting', anonymous=True)
        # Создаем топик "Detect" для вывода измененного изображения(обводка
        ↪  распознанных объектов)
        self.image_pub = rospy.Publisher("Detect",Image,queue_size=10)

        # Если мы летаем в симуляторе, то класс нужно вызывать со значением True
        ↪  (Пример: col = Main_Config(True) )
        self.simulator = simulator
        # Присвоим значения цветов в соответствии со средой симулятор/реальный мир
        # Ваши параметры могут разниться, не забудьте подобрать новые
        if self.simulator:
            # Параметры для симулятора
```

```
44              self.red_low = np.array([0,0,240])
45              self.red_high = np.array([10,10,255])
46
47              self.blue_low = np.array([240,0,0])
48              self.blue_high = np.array([255,10,10])
49
50              self.yellow_low = np.array([0,240,240])
51              self.yellow_high = np.array([10,255,255])
52          else:
53              # Параметры для реального мира
54              self.red_low = np.array([55,55,170])
55              self.red_high = np.array([135,125,255])
56
57              self.blue_low = np.array([120,90,0])
58              self.blue_high = np.array([210,140,80])
59
60              self.yellow_low = np.array([10,160,160])
61              self.yellow_high = np.array([120,230,220])
62
63          # Задаем параметры для создания масок цвета
64          self.st1 = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 21), (10, 10))
65          self.st2 = cv2.getStructuringElement(cv2.MORPH_RECT, (11, 11), (5, 5))
66          # Создаем видео для отладки цветовых порогов и кода
67          self.out =
            ↪  cv2.VideoWriter('Video_output.avi',cv2.VideoWriter_fourcc('X','V','I','D'),
            ↪  20, (320,240))
68
69          # Переменные условий начала действий
70          # Отвечает за распознавание положения навигационной стрелки
71          self.Arrow = False
72          # Отвечает за распознавание Qr-кода
73          self.Qr = False
74          # Отвечает за распознавание цвета навигационной стрелки
75          self.Color = False
76          # Отвечает за распознавание стеллажа
77          self.Dron_point = False
78
79          # Переменные хранящие данные
80          # Номер заказа
81          self.order = -1
82          # Координаты препятствий
83          self.col_ar = []
84          # Координата навигационной стрелки
85          self.nav = []
86          # Координаты маршрута, построенного в обход препятствий
87          self.mas = []
88          # Высота стеллажа
89          self.zz = 0.5
90          # Координаты положения стеллажа относительно матрицы камеры
91          self.cx, self.cy = -1, -1
92          # Цвет навигационной стрелки
93          self.color_arrow = 'black'
94          # Все возможные значение навигационной стрелки, сектора
95          self.sect = ['Sector B','Sector D','Sector A','Sector C']
96          # Значение навигационной стрелки, сектор
97          self.arrow = 'Sector D'
98          # Переменная необходимая для работы с изображением из топика
99          self.bridge = CvBridge()
100
101         # Подписание на топик, содержащий изображение
```

```
102         self.image_sub = rospy.Subscriber("main_camera/image_raw",Image,self.callback)
103     def navigate_wait(self, x=0, y=0, z=0, yaw=math.radians(90), speed=1,
        ↪  frame_id='aruco_map', auto_arm=False, tolerance=0.15):
104         # Функция полета до указанной точки
105         # Летим до указанной точки
106         # И пока не прилетим, не выходим из цикла
107         navigate(x=x, y=y, z=z, yaw=yaw, speed=speed, frame_id=frame_id,
            ↪  auto_arm=auto_arm)
108         while not rospy.is_shutdown():
109             # Берем координаты коптера в пространстве
110             telem = get_telemetry(frame_id='navigate_target')
111             if math.sqrt(telem.x ** 2 + telem.y ** 2 + telem.z ** 2) < tolerance:
112                 break
113             rospy.sleep(0.1)
114     def navigate_mas(self):
115         # Функция полета по координатам маршрута, построенного в обход препятствий
116         for x, y in self.mas:
117             self.navigate_wait(x=x*0.2, y=y*0.2, z=1.5, frame_id='aruco_map')
118             if math.sqrt((self.nav[0]-x*0.2) ** 2 + (self.nav[1]-y*0.2) ** 2) < 0.2:
                ↪  break
119     def normalize(self, x, y):
120         # Функция нормализации координат
121         return x / math.sqrt(x ** 2 + y ** 2), y / math.sqrt(x ** 2 + y ** 2)
122     def land_on_dronpoint(self):
123         # Функция посадки на стеллаж
124         x0, y0 = 320 // 2, 240 // 2
125         # Пока дрон не над центром стеллажа
126         while math.sqrt((x0 - self.cx) ** 2 + (y0 - self.cy) ** 2) > 20:
127             # Берем координаты коптера в пространстве
128             telem = get_telemetry(frame_id='aruco_map')
129             # Создаем вектор движения до стеллажа, с скоростью полета
130             dx, dy = self.normalize(self.cx - x0, self.cy - y0)
131             dx /= 15
132             dy /= 15
133             dy = -dy
134             # Летим по заданному вектору к центру стеллажа
135             set_position(x=telem.x + dx, y=telem.y + dy, z=self.zz+0.2,
                ↪  yaw=math.radians(90), frame_id='aruco_map')
136             rospy.sleep(0.1)
137         # Когда прилетели производим индикацию, в соответствии с цветом навигационной
            ↪  стрелки
138         if self.color_arrow == 'black':
139             set_effect(effect='fade', r=255, g=255, b=255)
140         elif self.color_arrow == 'red':
141             set_effect(effect='fade', r=255, g=0, b=0)
142         elif self.color_arrow == 'blue':
143             set_effect(effect='fade', r=0, g=0, b=255)
144         elif self.color_arrow == 'yellow':
145             set_effect(effect='fade', r=255, g=255, b=0)
146         # Садимся
147         land()
148         rospy.sleep(1)
149         # Выключаем двигатели
150         arming(False)
151         # Производим корректную световую индикацию
152         rospy.sleep(5)
153         # Выключаем индикацию
154         set_effect(effect='fade', r=0, g=0, b=0)
155         # Выводим корректно сообщение о доставке
156         print('{} delivered in {}'.format(self.order,self.arrow))
```

```
157     def dop_oblet(self):
158         # Функция находящая дополнительные препятствия, на случай когда расстояние
            ↪  между краями двух препятствий меньше размеров дрона
159         lenn = len(self.col_ar)
160         for i in range(lenn-1):
161             for j in range(i+1,lenn):
162                 if i!=j and ((self.col_ar[i][0] - self.col_ar[j][0])**2 +
                    ↪  (self.col_ar[i][1] - self.col_ar[j][1])**2)**0.5 <= 0.8:
163                     self.col_ar.append([min(self.col_ar[i][0],self.col_ar[j][0]) +
                        ↪  abs(self.col_ar[i][0] - self.col_ar[j][0])/2,
                        ↪  min(self.col_ar[i][1],self.col_ar[j][1]) +
                        ↪  abs(self.col_ar[i][1] - self.col_ar[j][1])/2])
164     def check(self,x,y):
165         # Функция проверки координаты на возможность подлета в нее, учитывая
            ↪  препятствия
166         if True in [True for i in range(len(self.col_ar)) if
            ↪  ((x*0.2-self.col_ar[i][0])**2 + (y*0.2-self.col_ar[i][1])**2)**(1/2) <
            ↪  0.4]: return True
167         else: return False
168     def check_line(self, a=1,b=1):
169         # Функция находящая координаты конца вектора, со сдвигом по двум осям,
            ↪  учитывая препятствия
170         for i in range(17):
171             if a == 0:
172                 if self.check(self.nav_x+i,self.f_y + abs(self.f_x-self.nav_x)) ==
                    ↪  False and self.nav_x+i <= 12:
173                     self.v1 = self.nav_x+i
174                     if b == 1: self.v2 = self.f_y + abs(self.f_x-self.nav_x)
175                     else: self.v2 = self.f_y
176                     return True
177                 elif self.check(self.nav_x-i,self.f_y + abs(self.f_x-self.nav_x)) ==
                    ↪  False and self.nav_x-i >= 0:
178                     self.v1 = self.nav_x-i
179                     if b == 1: self.v2 = self.f_y + abs(self.f_x-self.nav_x)
180                     else: self.v2 = self.f_y
181                     return True
182             else:
183                 if self.check(self.f_x + abs(self.f_y-self.nav_y),self.nav_y+i) ==
                    ↪  False and self.nav_y+i <= 12:
184                     self.v2 = self.nav_y+i
185                     if b == 1 : self.v1 = self.f_x + abs(self.f_y-self.nav_y)
186                     else: self.v1 = self.f_x
187                     return True
188                 elif self.check(self.f_x + abs(self.f_y-self.nav_y),self.nav_y-i) ==
                    ↪  False and self.nav_y-i >= 0:
189                     self.v2 = self.nav_y-i
190                     if b == 1: self.v1 = self.f_x + abs(self.f_y-self.nav_y)
191                     else: self.v1 = self.f_x
192                     return True
193             return False
194     def check_circle(self):
195         # Функция находящая координаты начала вектора, со сдвигом по двум осям,
            ↪  учитывая препятствия
196         for i in range(17):
197             if self.det_line(self.f_x+i,self.f_y,self.v1,self.v2) == True and
                ↪  self.f_x+i <= 16 and
                ↪  self.det_line(self.f_x,self.f_y,self.f_x+i,self.f_y) == True:
198                 self.mas.append([self.f_x+i,self.f_y])
199                 self.mas.append([self.v1,self.v2])
200                 self.f_x, self.f_y = self.v1, self.v2
```

```
201                 return True
202             elif self.det_line(self.f_x-i,self.f_y,self.v1,self.v2) == True and
                ↪  self.f_x-i >= 0 and
                ↪  self.det_line(self.f_x,self.f_y,self.f_x-i,self.f_y) == True:
203                 self.mas.append([self.f_x-i,self.f_y])
204                 self.mas.append([self.v1,self.v2])
205                 self.f_x, self.f_y = self.v1, self.v2
206                 return True
207             elif self.det_line(self.f_x,self.f_y+i,self.v1,self.v2) == True and
                ↪  self.f_y+i <= 12 and
                ↪  self.det_line(self.f_x,self.f_y,self.f_x,self.f_y+i) == True:
208                 self.mas.append([self.f_x,self.f_y+i])
209                 self.mas.append([self.v1,self.v2])
210                 self.f_x, self.f_y = self.v1, self.v2
211                 return True
212             elif self.det_line(self.f_x,self.f_y-i,self.v1,self.v2) == True and
                ↪  self.f_y-i >= 0 and
                ↪  self.det_line(self.f_x,self.f_y,self.f_x,self.f_y-i) == True:
213                 self.mas.append([self.f_x,self.f_y-i])
214                 self.mas.append([self.v1,self.v2])
215                 self.f_x, self.f_y = self.v1, self.v2
216                 return True
217         return False
218     def det_line(self,x1,y1,x2,y2,r=0.39):
219         # Функция проверяющая наличие касания прямой окружности
220         x1,y1,x2,y2 = x1*0.2,y1*0.2,x2*0.2,y2*0.2
221         for x,y in self.col_ar:
222             try:
223                 k = (y1 - y2)/(x1 - x2)
224                 b0 = y1 - k*x1
225                 a = k**2 + 1
226                 b = 2*k*(b0 - y) - 2*x
227                 c = (b0 - y)**2 + x**2 - r**2
228                 delta = b**2 - 4*a*c
229                 if delta >= 0: return False
230             except: pass
231         return True
232     def vector_x(self):
233         # Функция находящая координаты конца вектора, без сдвига по оси y, учитывая
            ↪  препятствия
234         self.v1,self.v2 = self.nav_x, self.nav_y
235         if self.check_circle() == False:
236             for i in range(12):
237                 self.v1,self.v2 = self.nav_x+i, self.nav_y
238                 if self.check_circle(): break
239                 self.v1,self.v2 = self.nav_x-i, self.nav_y
240                 if self.check_circle(): break
241     def vector_y(self):
242         # Функция находящая координаты конца вектора, без сдвига по оси x, учитывая
            ↪  препятствия
243         self.v1,self.v2 = self.nav_x, self.nav_y
244         if self.check_circle() == False:
245             for i in range(12):
246                 self.v1,self.v2 = self.nav_x, self.nav_y+i
247                 if self.check_circle(): break
248                 self.v1,self.v2 = self.nav_x, self.nav_y-i
249                 if self.check_circle(): break
250
251     def navigate_avoidece(self, start, finish):
252         # Функция облета препятствий
```

```
253         # Координаты конечной точки полета
254         self.nav_x, self.nav_y = (finish[0]*100)//20, (finish[1]*100)//20
255         # Координаты движения дрона
256         self.f_x, self.f_y = start[0]//0.2, start[1]//0.2
257         # Переменная, хранящая значения количества координат для облета, чтобы коптер
            ↪  не зациклился
258         h = 5
259         while (self.f_x != self.nav_x or self.f_y != self.nav_y) and h > 0:
260             h-=1
261             if self.f_x == self.nav_x:
262                 # Если мы выровнялись по оси x
263                 # То выравниваем по y
264                 self.vector_y()
265             elif self.f_y == self.nav_y:
266                 # Если мы выровнялись по оси y
267                 # То выравниваем по x
268                 self.vector_x()
269             else:
270                 # Если мы не выровнялись не по одной из осей
271                 # То ищем самую кратчайшую ось для выравнивания
272                 if self.nav_x < self.nav_y:
273                     # И пробуем выравнивается по x
274                     self.check_line(a=0)
275                 else:
276                     # И пробуем выравнивается по y
277                     self.check_line(a=1)
278                 self.check_circle()
279 
280     def find_arrow(self,im):
281         # Функция распознавания положения навигационной стрелки
282         # Берем модели стрелок
283         samples = np.loadtxt('generalsamples.data', np.float32)
284         responses = np.loadtxt('generalresponses.data', np.float32)
285         responses = responses.reshape((responses.size, 1))
286         model = cv2.ml.KNearest_create()
287         model.train(samples, cv2.ml.ROW_SAMPLE, responses)
288 
289         # Создаем копию изображения, для обводки навигационной стрелки
290         out = im.copy()
291         # Создаем маску
292         hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
293         kernal = np.ones((5, 5), np.uint8)
294         gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
295         ret, thresh = cv2.threshold(gray, 70, 255, cv2.THRESH_BINARY_INV)
296         # Ищем контуры стеллажа навигационной стрелки
297         contours = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[1]
298         num = 0
299         arr = []
300 
301         # Проходимся по контурам
302         for cnt in contours:
303             # Берем координаты центра навигационной стрелки, относительно матрицы
                ↪  камеры
304             [x, y, w, h] = cv2.boundingRect(cnt)
305             try:
306                 # Если нашли, то обводим навигационною стрелку
307                 cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0), 2)
308 
309                 roi = thresh[y:y + h, x:x + w]
310                 roismall = cv2.resize(roi, (10, 10))
```

```
311                 roismall = roismall.reshape((1, 100))
312                 roismall = np.float32(roismall)
313                 retval, results, neigh_resp, dists = model.findNearest(roismall, k=1)
314                 num = int(results[0][0])
315                 arr.append((cnt, dists[0][0], num))
316             except: pass
317 
318         # Записываем ее значение в переменную, для дальнейшей работы с ней
319         need_arr = min(arr, key=lambda x: x[1])
320         num = need_arr[2]
321         # Обводим на изображении навигационную стрелку
322         cv2.drawContours(out, [need_arr[0]], -1, (255,105,180), 3)
323         # Выводим в топик измененное изображение
324         try:
325             self.image_pub.publish(self.bridge.cv2_to_imgmsg(out, "bgr8"))
326         except CvBridgeError as e:
327             print(e)
328 
329         # Заносим в переменную значение положения навигационной стрелки
330         self.arrow = self.sect[num]
331         # Вывод в терминал сообщение о результатах распознавания
332         print('{} required'.format(self.arrow))
333         # Останавливаем поиск положения навигационной стрелки
334         self.Arrow = False
335     def color(self,mask,a):
336         # Функция распознавания цвета навигационной стрелки
337         # Создаем маску
338         thresh = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, self.st1)
339         thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, self.st2)
340         # Ищем контуры цветного объекта(навигационной стрелки)
341         cnt = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[1]
342         # И проходим по ним
343         for c in cnt:
344             try:
345                 # Определяем количество пикселей занятых цветом
346                 moments = cv2.moments(c, 1)
347                 sum_pixel = moments['m00']
348                 # Если их больше 300, то мы нашли цвет навигационной стрелки
349                 if sum_pixel > 300:
350                     # Заносим в переменную значение цвета навигационной стрелки
351                     self.color_arrow = a
352                     # Останавливаем поиск цвета навигационной стрелки
353                     self.Color = False
354             except:pass
355     def dronpoint_detect(self,img):
356         # Функция распознавания цвета навигационной стрелки
357         # Создаем маску
358         hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
359         mask = cv2.inRange(hsv, (50, 55, 50), (85, 255, 255))
360         # Ищем контуры стеллажа
361         contours = cv2.findContours(mask, cv2.RETR_EXTERNAL,
        ↪  cv2.CHAIN_APPROX_SIMPLE)[1]
362         # Если контуры есть
363         if len(contours) != 0:
364             # Берем наибольший, тот который ближе всего к камере
365             self.cnt = max(contours, key=cv2.contourArea)
366             # Находим координаты центра стеллажа
367             [x, y, w, h] = cv2.boundingRect(self.cnt)
368             self.cx = x + w // 2
369             self.cy = y + h // 2
```

```
370             # Возвращаем True если нашли стеллаж
371             return True
372     def sect_fly(self):
373         # Функция полета от навигационной стрелки до стеллажа
374         # Выбираем направление движения в соответствии с положение навигационной
            ↪  стрелки, сектором
375         if self.arrow == 'Sector A':
376             self.y_f = 0.4
377             self.x_f = 0
378         elif self.arrow == 'Sector B':
379             self.y_f = -0.4
380             self.x_f = 0
381         elif self.arrow == 'Sector C':
382             self.y_f = 0
383             self.x_f = 0.4
384         elif self.arrow == 'Sector D':
385             self.y_f = 0
386             self.x_f = -0.4
387         self.mas = []
388         # Берем координаты коптера в пространстве
389         telem = get_telemetry(frame_id='aruco_map')
390         # Если координата не приводит к столкновению с препятствиями, летим подальше,
            ↪  чтобы не принять стеллажи из других секторов за необходимый
391         if self.check(telem.x+self.x_f*4, telem.y+self.y_f*4)== False:
392             # Летим еще подальше, чтобы не столкнуться с препятствиями
393             self.navigate_avoidece([telem.x,telem.y],[telem.x+self.x_f*6,
                ↪  telem.y+self.y_f*6])
394         else: self.navigate_avoidece([telem.x,telem.y],[telem.x+self.x_f*4,
            ↪  telem.y+self.y_f*4])
395         self.navigate_mas()
396         for i in range(9):
397             # Начинаем поиск стеллажа
398             self.Dron_point = True
399             rospy.sleep(1)
400             # Если нашли стеллаж, выходим из функции
401             if self.cx != -1 : break
402             # Заканчиваем поиск стеллажа
403             self.Dron_point = False
404             self.mas = []
405             # Берем координаты коптера в пространстве
406             telem = get_telemetry(frame_id='aruco_map')
407             # И если координата не приводит к столкновению с препятствиями, летим в
                ↪  нее
408             if self.check(telem.x+self.x_f*i, telem.y+self.y_f*i)== False and
                ↪  telem.x+self.x_f*i <= 3.2 and telem.x+self.x_f*i >= 0 and
                ↪  telem.y+self.y_f*i <= 2.4 and telem.y+self.y_f*i >= 0:
409                 self.navigate_avoidece([telem.x,telem.y],[telem.x+self.x_f*i,
                    ↪  telem.y+self.y_f*i])
410     def callback(self,data):
411         # Функция обрабатывающая значение из топика, в ней происходит обработка
            ↪  изображения
412         # Считывание и конвертация изображения в вид пригодный для дальнейшей работы
            ↪  (try - для отсутствия ошибки если топик будет пустой)
413         try:
414             img = self.bridge.imgmsg_to_cv2(data, "bgr8")
415         except:pass
416         # Если в реальном мире, выравниваем изображение
417         if self.simulator == False:
```

```
418             img = cv2.undistort(
                ↪  img,np.array([[166.23942373073172,0,162.19011246829268],
                ↪  [0,166.5880923974026,109.82227735714285], [0,0,1]]), np.array([
                ↪  2.15356885e-01,  -1.17472846e-01,  -3.06197672e-04,-1.09444025e-04,
                ↪  -4.53657258e-03,   5.73090623e-01,-1.27574577e-01,  -2.86125589e-02,
                ↪  0.00000000e+00,0.00000000e+00,   0.00000000e+00,
                ↪  0.00000000e+00,0.00000000e+00,   0.00000000e+00]),
                ↪  np.array([[166.23942373073172,0,162.19011246829268],
                ↪  [0,166.5880923974026,109.82227735714285], [0,0,1]]))
419         # Записываем видео, для отладки цветовых порогов и кода
420         self.out.write(img)
421         # Начинаем распознавать Qr-код
422         if self.Qr == True:
423             # Производим анализ изображения на наличие Qr-кода
424             barcodes  = pyzbar.decode(img)
425             # Если нашли Qr-код, то работаем с ним
426             if barcodes:
427                 for bar in barcodes:
428                     # Если до этого Qr-код не распознавали
429                     if self.order == -1:
430                         # Получаем данные из Qr-кода
431                       self.qr_data = (bar.data.decode("utf-8")).split('\n')
432                        for i in range (0,len(self.qr_data[0].split())-1,2):
433                             # Корректно выводим координаты препятствий
434 print('Column area x={},
    ↪  y={}'.format(self.qr_data[0].split()[i],self.qr_data[0].split()[i+1]))
435                             # И записываем в переменную, для дальнейшей работы с ними
436                             self.col_ar.append([float(self.qr_data[0].split()[i]),
                                ↪  float(self.qr_data[0].split()[i+1])])
437                         # Корректно выводим координату навигационной стрелки
438                         print('Navigation area x={},
                            ↪  y={}'.format(self.qr_data[1].split()[0],
                            ↪  self.qr_data[1].split()[-1]))
439                         # И записываем в переменную, для дальнейшей работы с ней
440                         self.nav.append(float(self.qr_data[1].split()[0]))
441                         self.nav.append(float(self.qr_data[1].split()[-1]))
442                         # Корректно выводим номер заказа
443                         print('Order number: {}'.format(self.qr_data[-1]))
444                         # И записываем в переменную, для дальнейшей работы с ним
445                         self.order = int(self.qr_data[-1])
446                         # Запускаем функцию, находящую дополнительные препятствия
447                         self.dop_oblet()
448                     # Ищем координаты Qr-кода
449                     (x, y, w, h) = bar.rect
450                     # Обводим на изображении Qr-код
451                     cv2.rectangle(img, (x, y), (x + w, y + h), (255,105,180), 2)
452         # Начинаем распознавать цвет навигационной стрелки
453         if self.Color:
454             # Ищем цвета на изображении
455             # Красный
456             self.color(cv2.inRange(img, self.red_low, self.red_high),'red')
457             # Желтый
458             self.color(cv2.inRange(img, self.yellow_low, self.yellow_high),'yellow')
459             # Синий
460             self.color(cv2.inRange(img, self.blue_low, self.blue_high),'blue')
461 
462             # В соответствии с цветом, выставляем высоту стеллажа
463             if self.color_arrow == 'yellow': self.zz = 0.25
464             elif self.color_arrow == 'blue': self.zz = 0.75
465             elif self.color_arrow == 'red': self.zz = 1
```

```
466         # Начинаем распознавать положение навигационной стрелки
467         if self.Arrow:
468             self.find_arrow(img.copy())
469         # Начинаем поиск на изображении стеллажа
470         if self.Dron_point:
471             if self.dronpoint_detect(img.copy()):
472                 # Если нашли стеллаж, обводим его
473                 cv2.drawContours(img, [self.cnt], 0, (255,105,180), 2)
474 
475         # Выводим в топик измененное изображение
476         try:
477             self.image_pub.publish(self.bridge.cv2_to_imgmsg(img, "bgr8"))
478         except CvBridgeError as e:
479             print(e)
```

Пример программного кода, заполняющий Report.txt для выполнения миссии всех дней (`Generate_Report.py`):

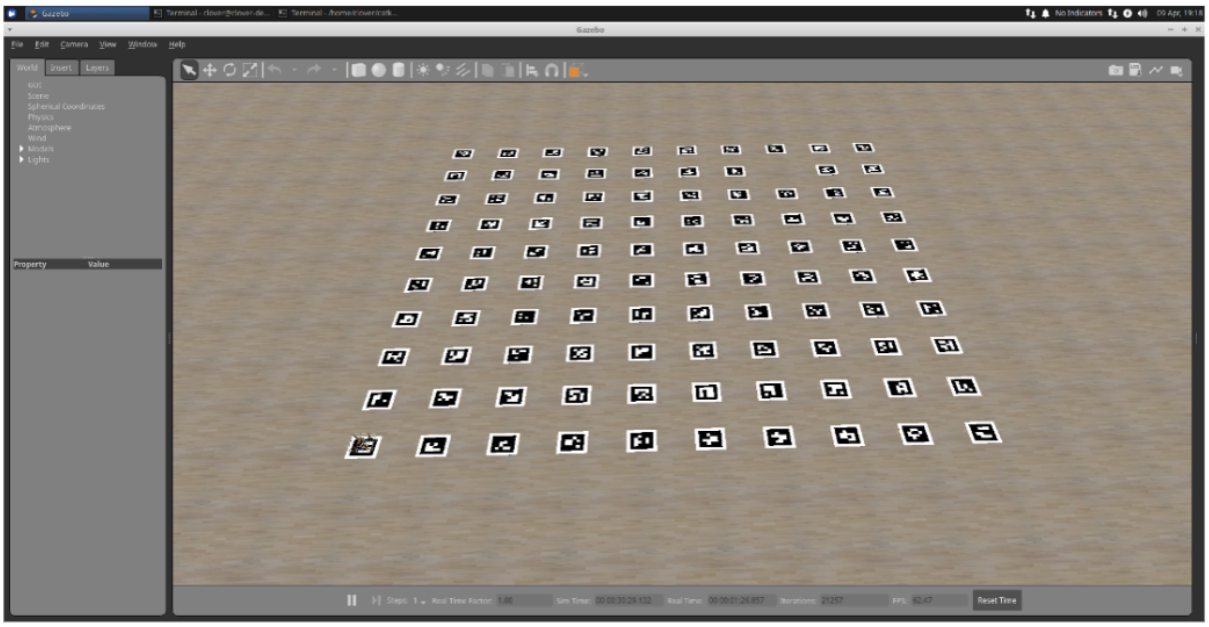
```
1  # -*- coding: utf-8 -*-
2  # Обязательная конструкция для работы с кириллицей
3  
4  # Инициализация библиотек
5  import rospy
6  from clover import srv
7  from std_msgs.msg import Float32MultiArray
8  
9  # Инициализация ноды
10 rospy.init_node('fly')
11 # Инициализация сервисов
12 get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
13 
14 # Создаем файл для записи данных
15 f.open('Report.txt', 'w')
16 # Пока программа работает
17 while not rospy.is_shutdown():
18     # Берем данные коптера(его координаты положения в пространстве относительно
       ↪  frame_id = 'aruco_map')
19     telem = get_telemetry(frame_id = 'aruco_map')
20     # Записываем эти данные в файл, в соответствии с требованиями регламента
21     f.write('{} {} {}\n'.format(telem.x, telem.y, telem.z))
22     # Выводим в терминал эти данные
23     print('{} {} {}'.format(telem.x, telem.y, telem.z))
24     # Производим задержку, для ограничения количества данных
25     rospy.sleep(0.1)
```

Часть 2. Решение задачи Инженерной части

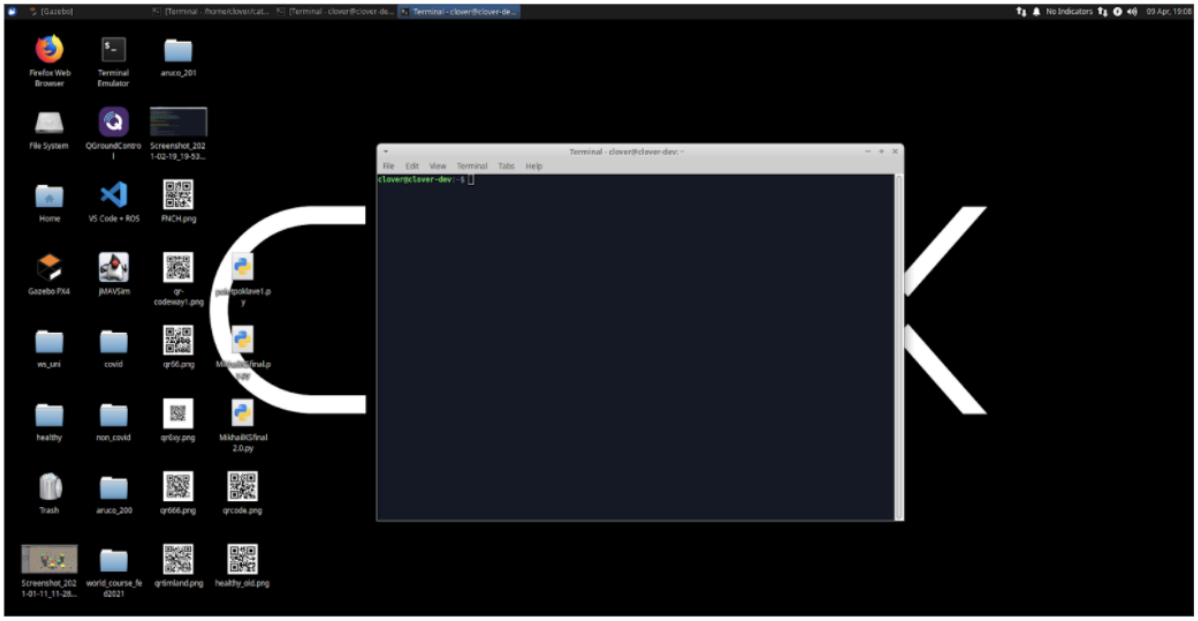
Создание сцены финальной зачетной попытки в симуляторе Gazebo

Загрузка карты и создание мира.

Запускаем виртуальную машину с образом симулятора Gazebo:



Для создания карты открываем Terminal Emulator:

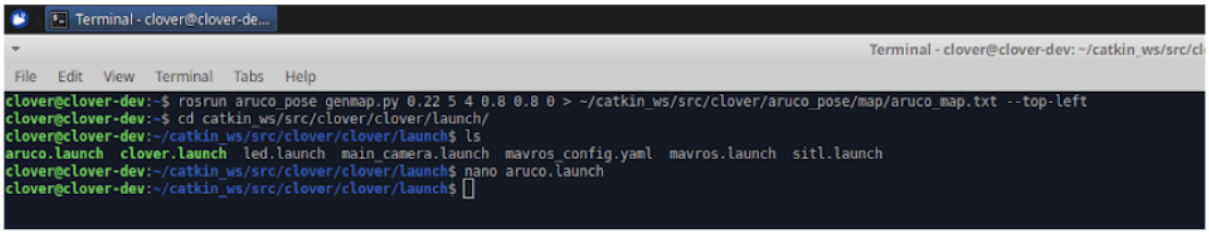


Далее нам необходимо создать ArUco-карту финального полетного полигона в формате .txt так же, как и на реальном дроне. Название карты — aruco_map.txt.

Команда для создания файла:

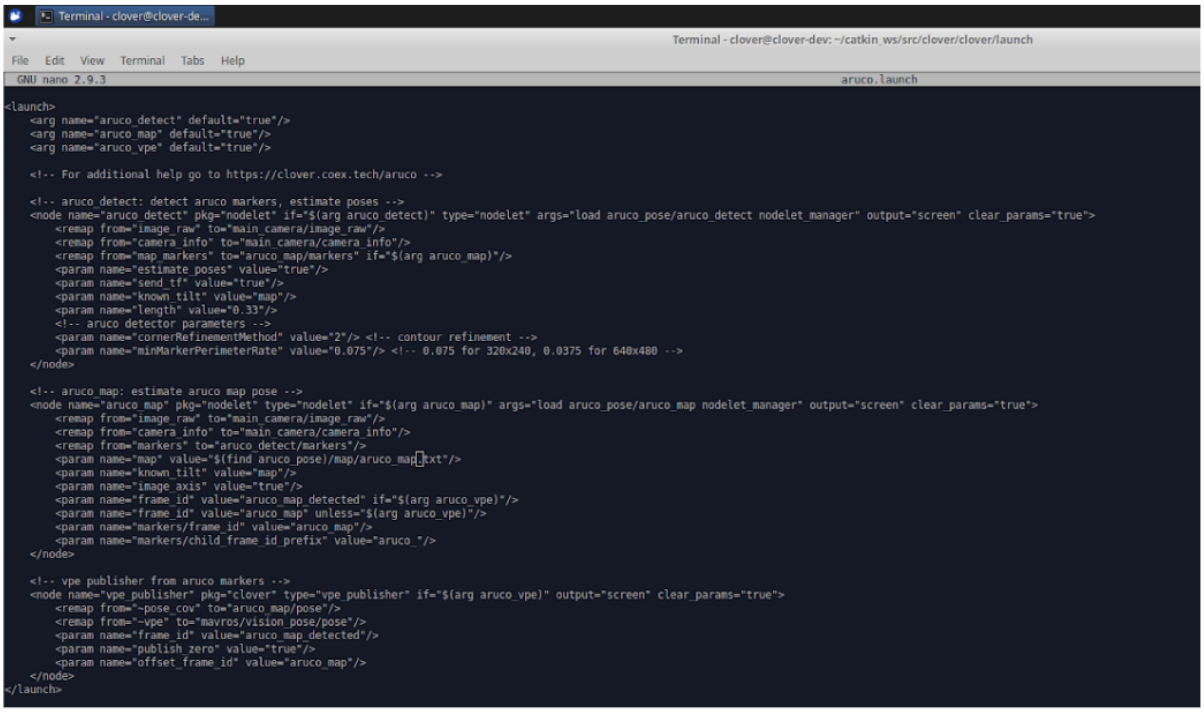
```
rosrun aruco_pose genmap.py 0.22 5 4 0.8 0.8 0 >
↪  ~/catkin_ws/src/clover/aruco_pose/map/aruco_map.txt --top-left
```

В `aruco_launch` необходимо выбрать созданный нами файл с картой меток. Директория launch файлов: `cd catkin_ws/src/clover/clover/launch/`



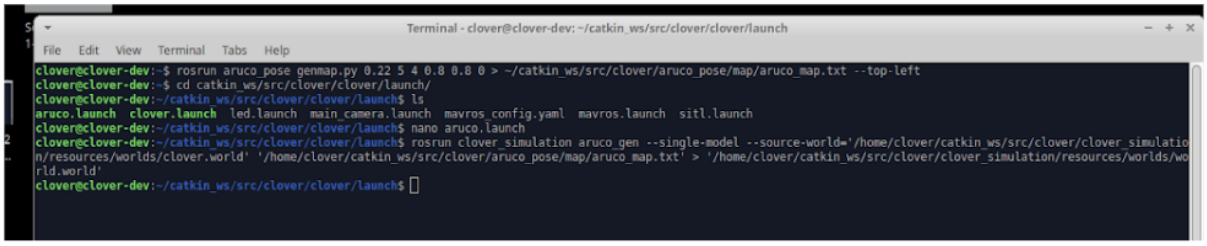
Открываем файл `aruco_launch`: `nano aruco_launch`.

Прописываем название созданного ранее файла с картой для его использования в образе: `aruco_map.txt`.



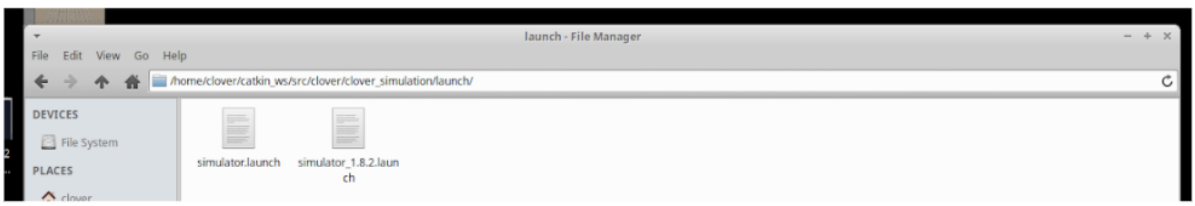
При помощи команды создаем модели маркеров в мире с необходимым нам названием:

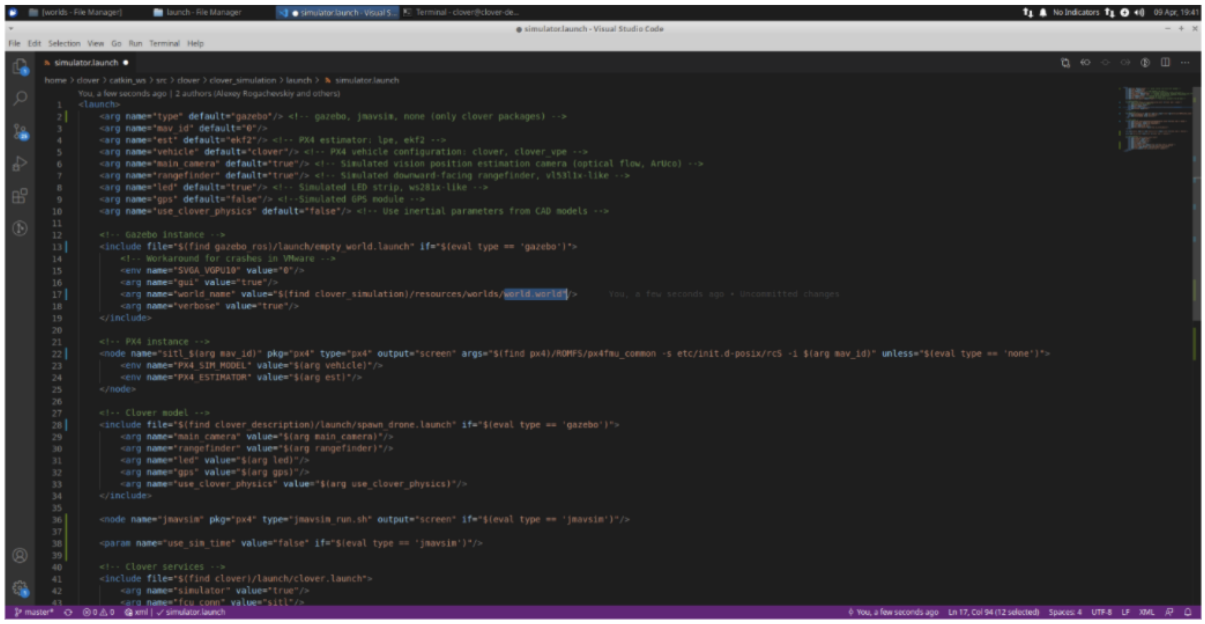
```
rosrun clover_simulation aruco_gen --single-model --source-world='/home/clover/
catkin_ws/src/clover/clover_simulation/resources/worlds/clover.world'
↪  '/home/clover/
catkin_ws/src/clover/aruco_pose/map/aruco_map.txt' >
↪  '/home/clover/catkin_ws/src/
clover/clover_simulation/resources/worlds/world.world'
```



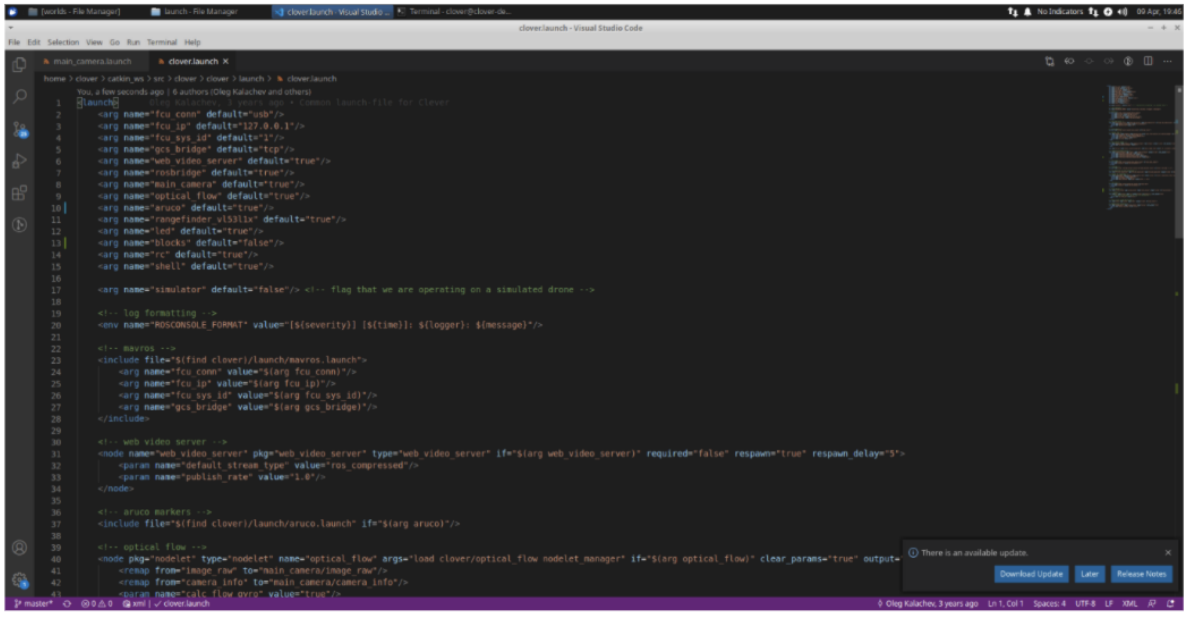
В файле `simulator.launch` необходимо выбрать название созданного мира `world.world`.

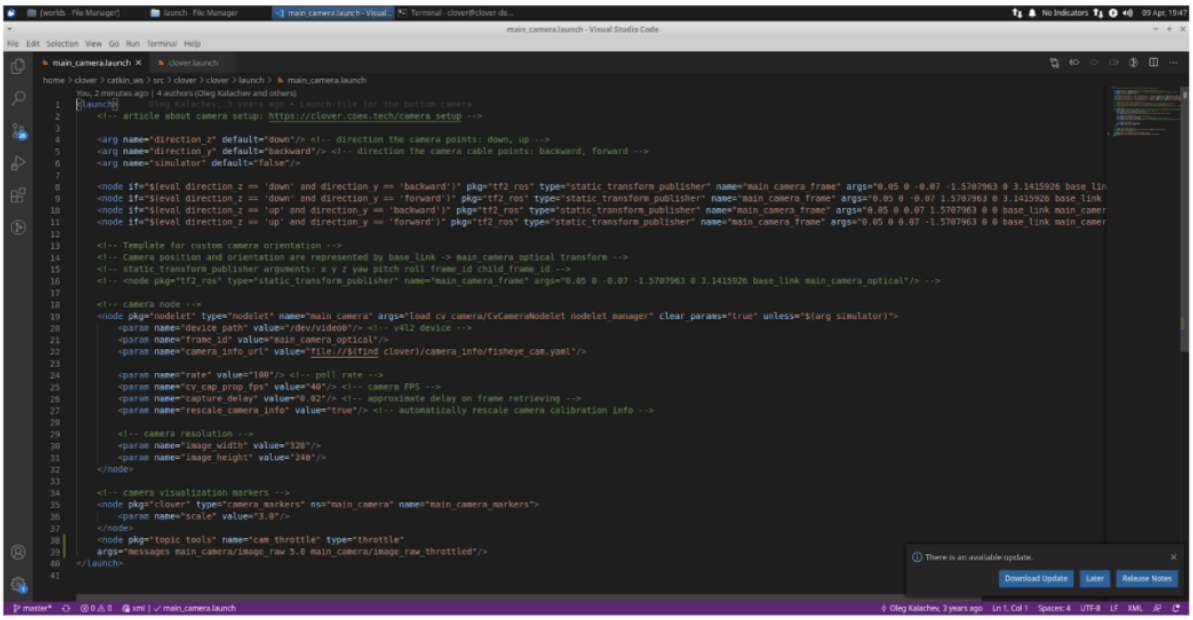
Путь в данную папку продемонстрирован на картинке:



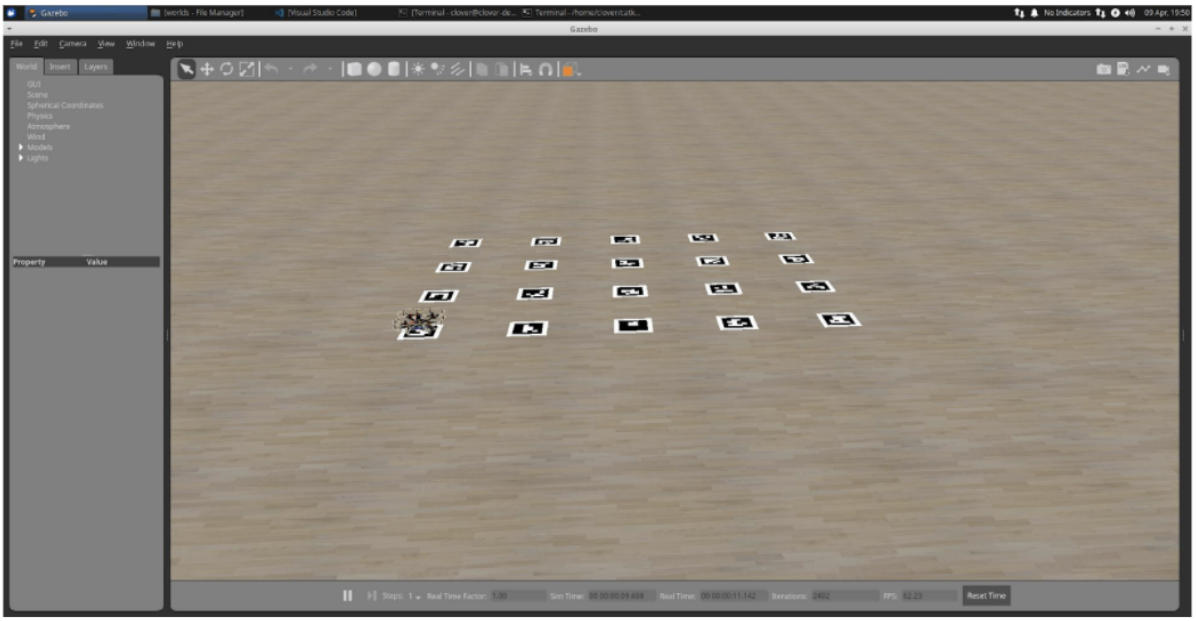


Проверяем файлы `clover.launch` и `main_camera.launch`



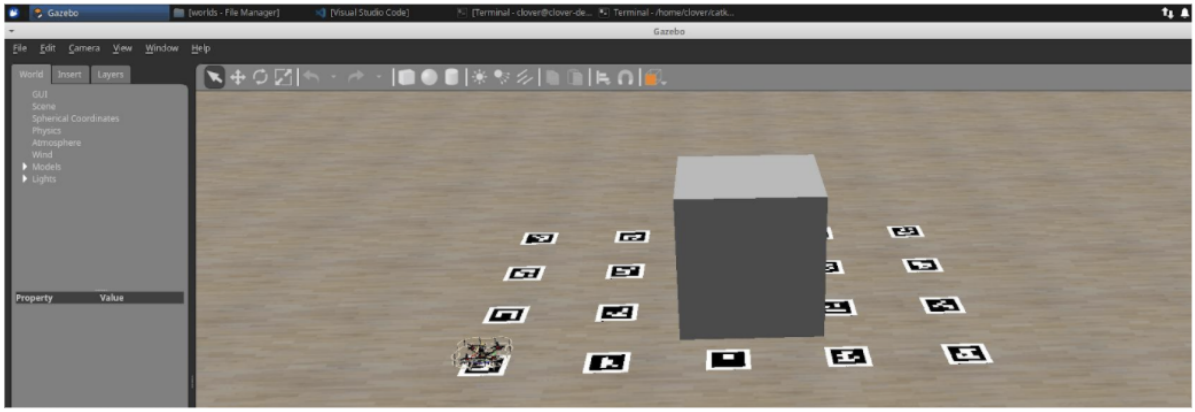


Запускаем Gazebo с созданным нами миром и с требуемой картой маркеров:



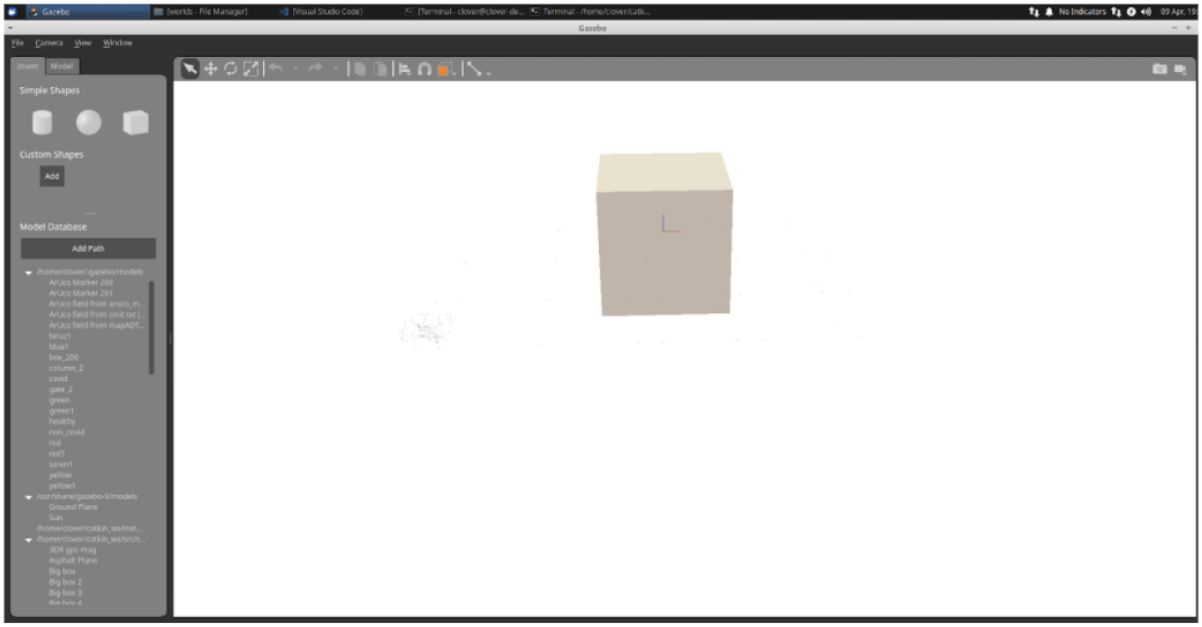
Переходим к созданию моделей и расположению их на карте.

Создаем стеллажи размерами: длина 0,5 м, ширина 0,5 и высотами 0,25, 0,5, 0,75 и 1 м.

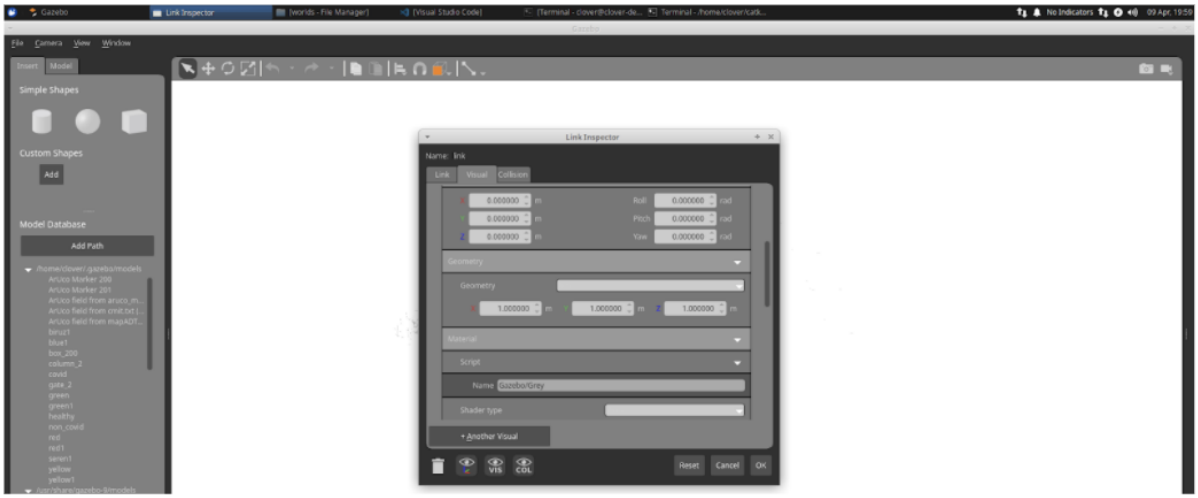


Импортируем стандартный куб для редактирования.

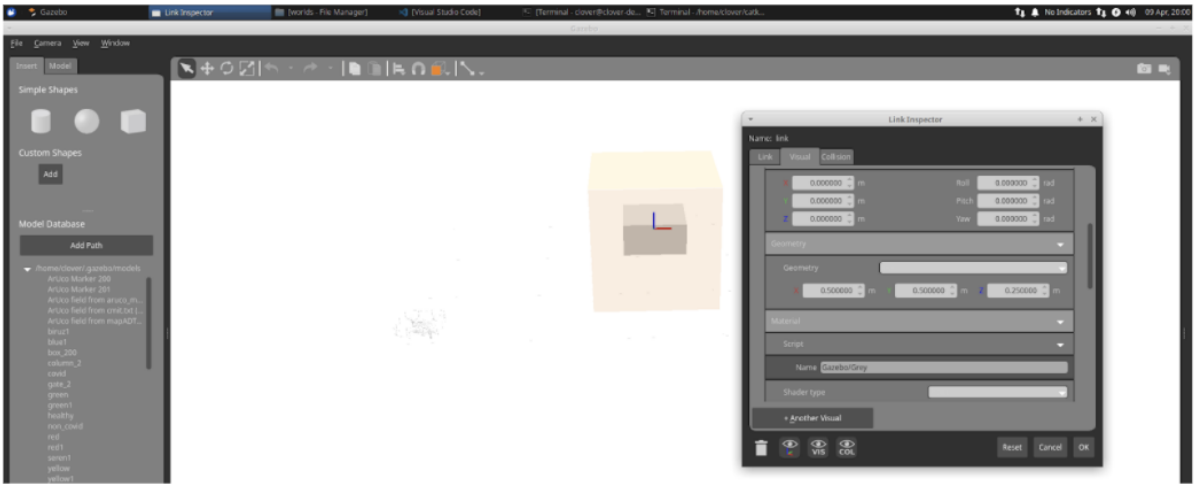
Правой кнопки мыши нажимаем на куб и переходим в раздел `edit model`:



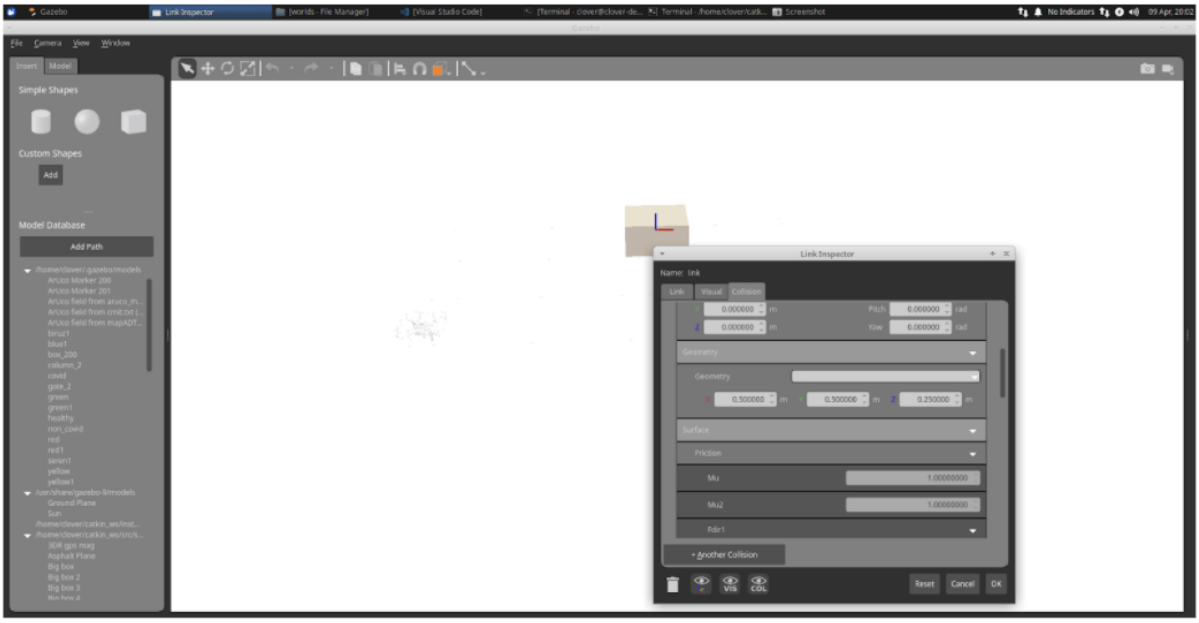
Также правой кнопки мыши нажимаем на куб и переходим в раздел `Open Link Inspector`:



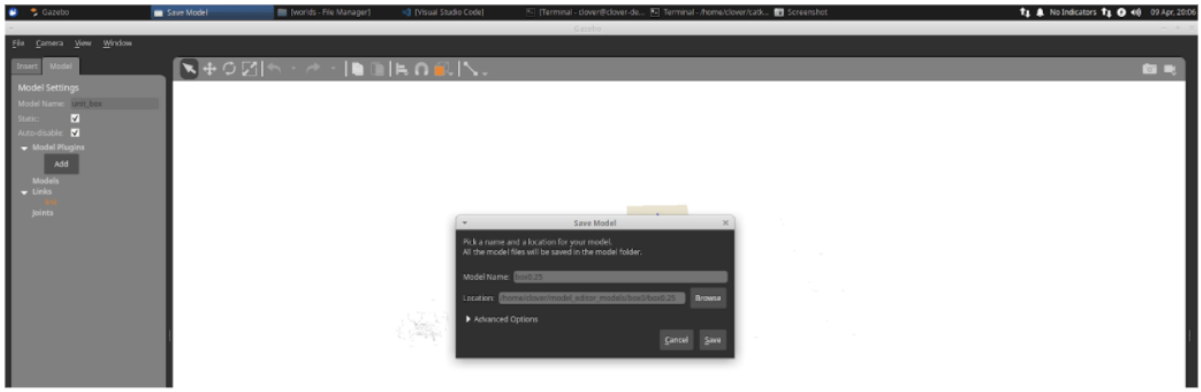
Во вкладке `Visual` и `Collision` выставляем необходимую геометрию куба:

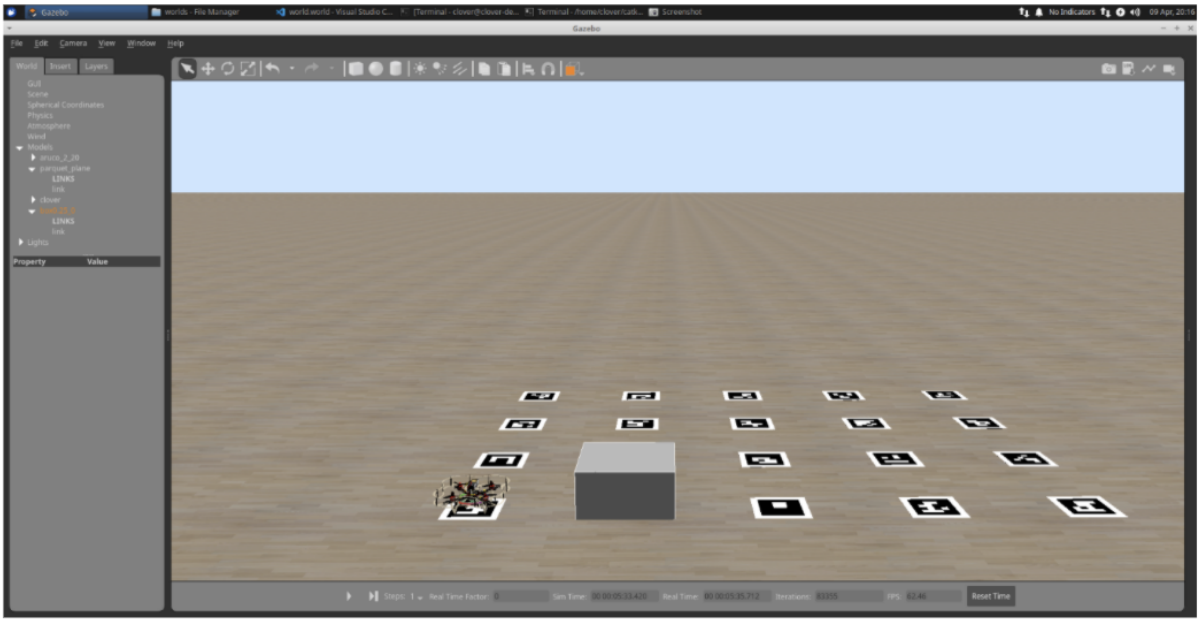


Во вкладке model отмечаем нашу модель статичной (`static`):



Сохраняем куб: `File -- Exit model editor -- Save and Exit`:

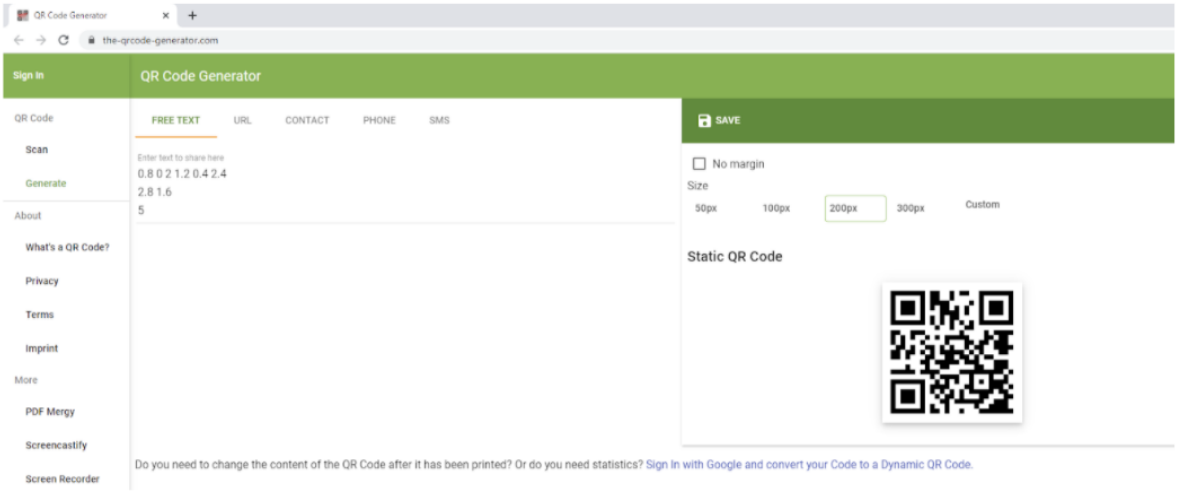




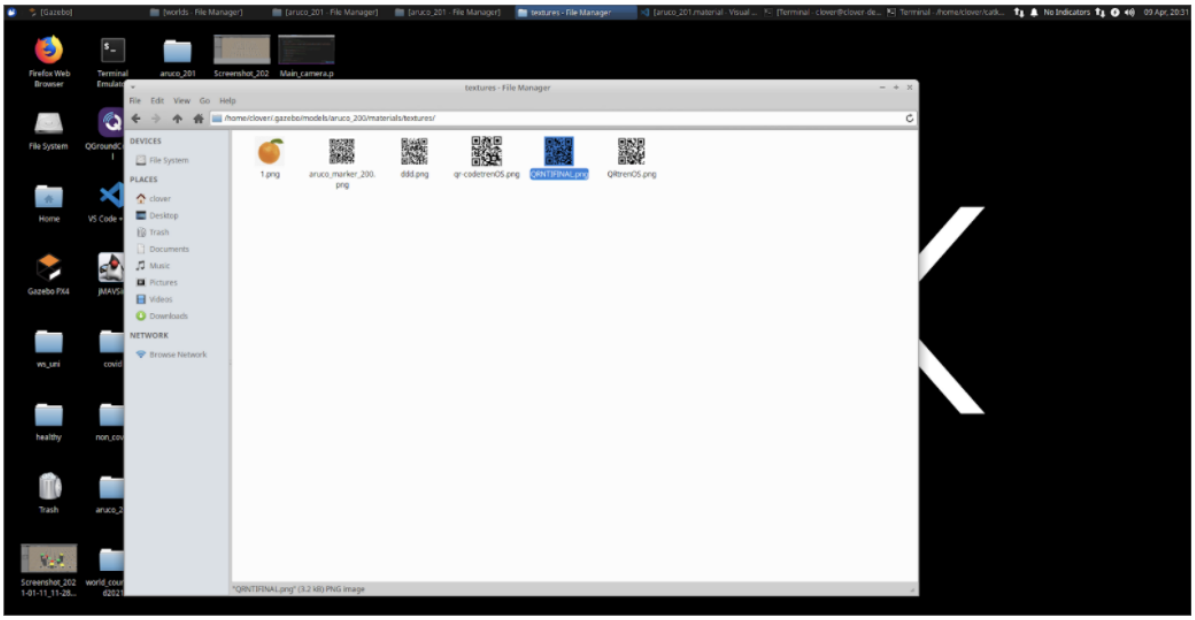
Используя данную методику создаем остальные стеллажи (кубы).

Переходим к созданию QR-кода:

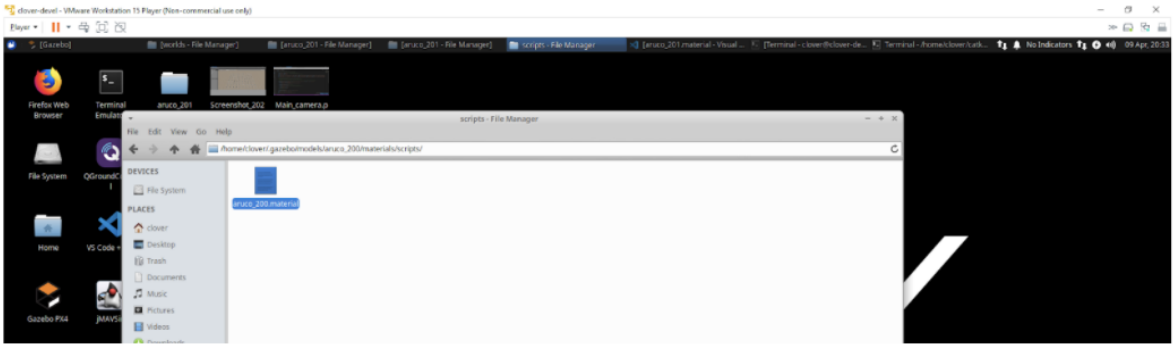
В генераторе QR-кодов создаем .png файл QR-кода с зашитой в него информацией для финальной попытки.



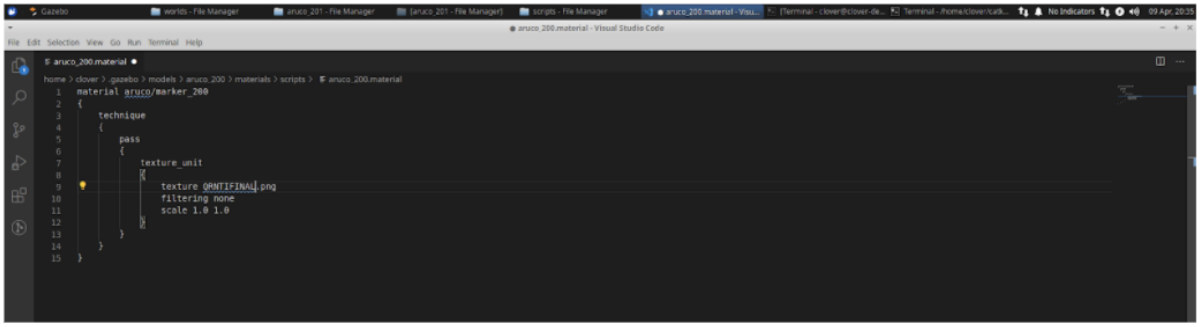
Полученный .png файл необходимо заменить в файле с любой подготовленной моделью с текстурой и расположить в .gazebo по продемонстрированному пути:



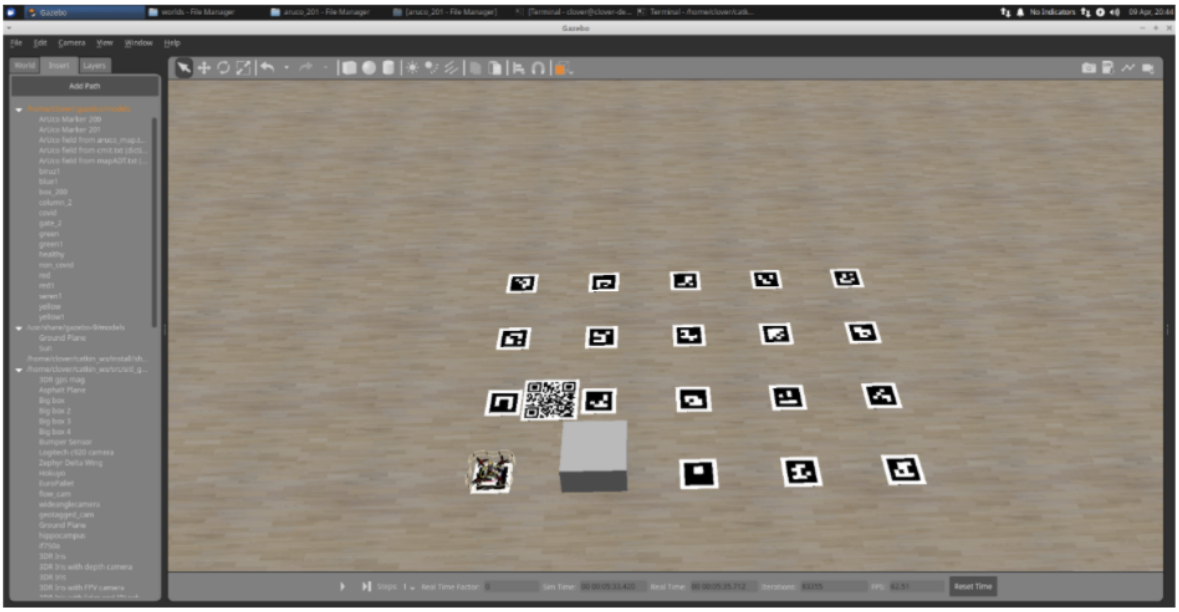
Открываем скрипт модели в папке `scripts`:



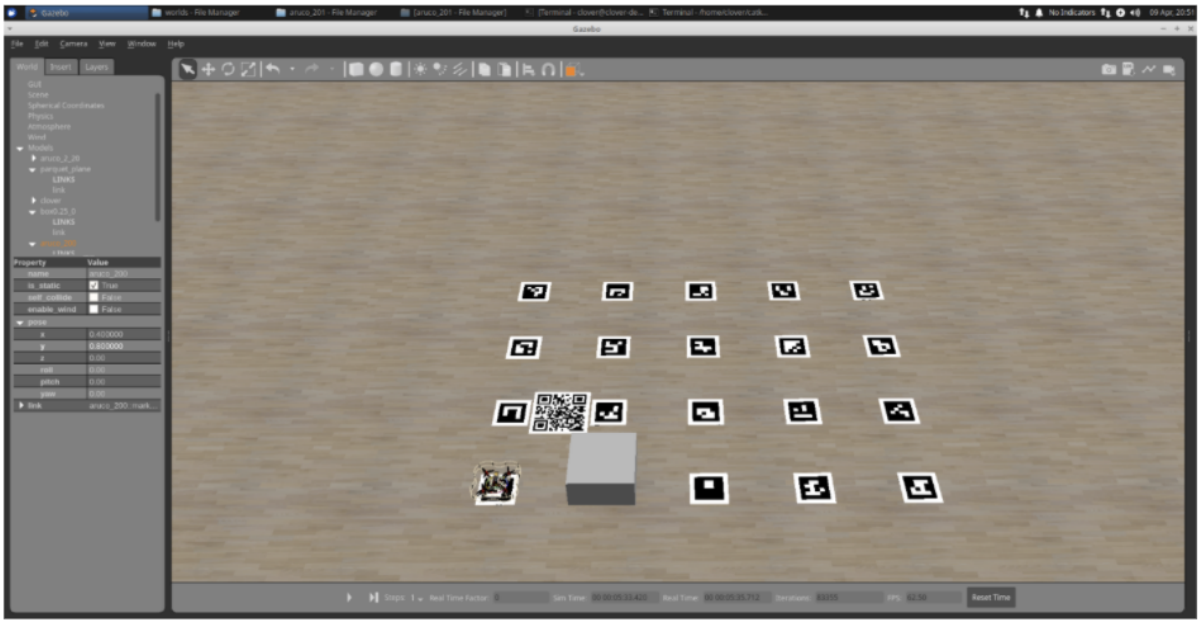
Меняем название текстуры на название созданного нами ранее QR-кода. Сохраняем:



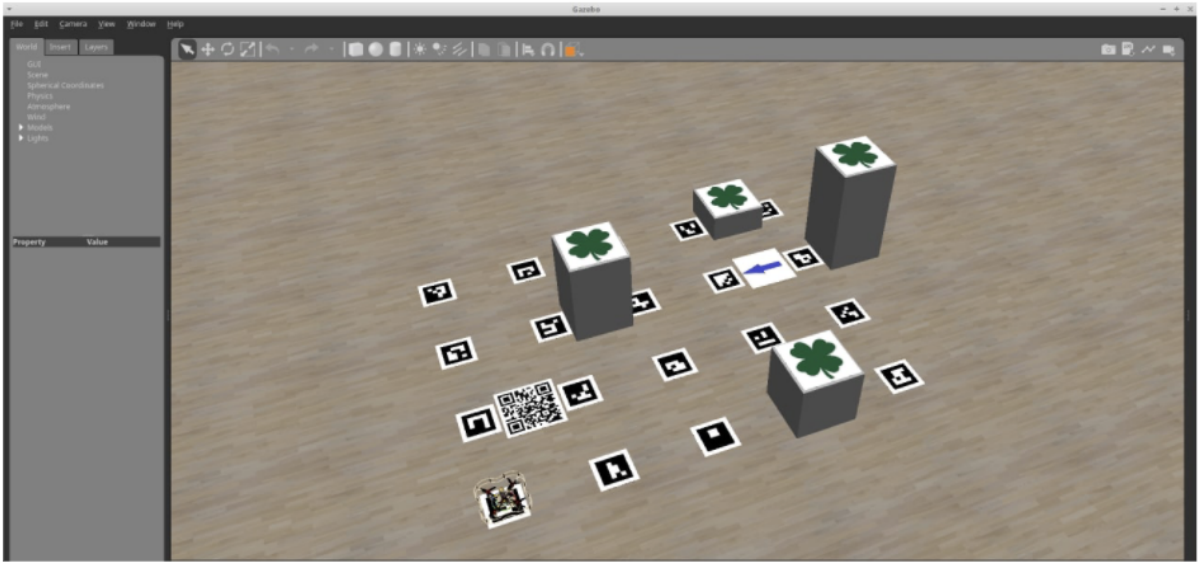
Для расположения QR-кода на полетном полигоне необходимо во вкладке `Insert` выбрать созданную модель из директории `/home/clover/.gazebo/models`:



Выбираем точное расположение QR-кода на полетном полигоне при нажатии на него и переходе во вкладку `World-Property-Pose`:



По аналогии создаем модель клевера и цветной стрелки (синей) и также по аналогии размещаем их на полетном полигоне по X, Y и Z, соответствующим физическому полетному полигону финальный попытки:

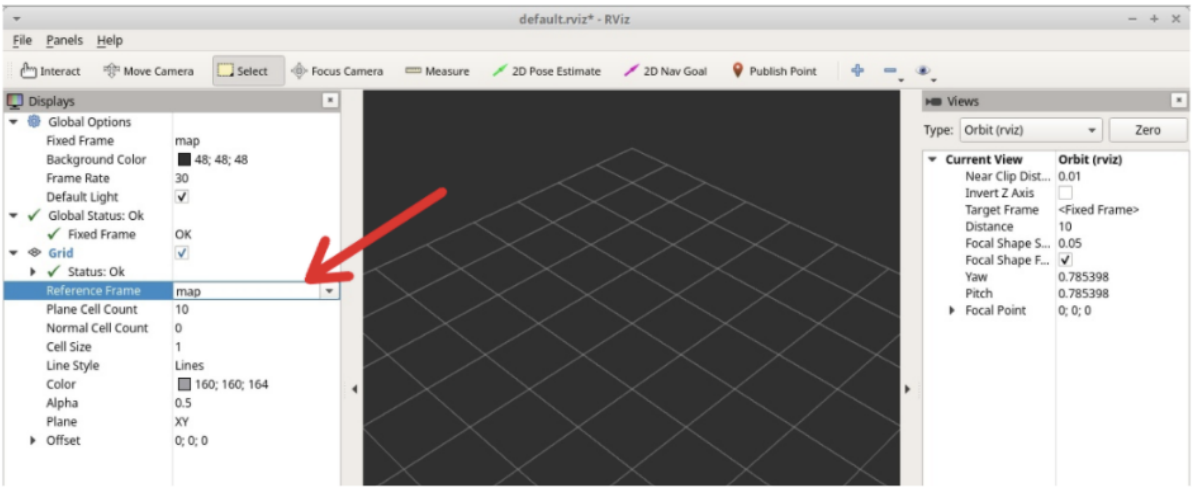


Собираем созданные файлы в папку, находящуюся на Google disk, согласно заданию:

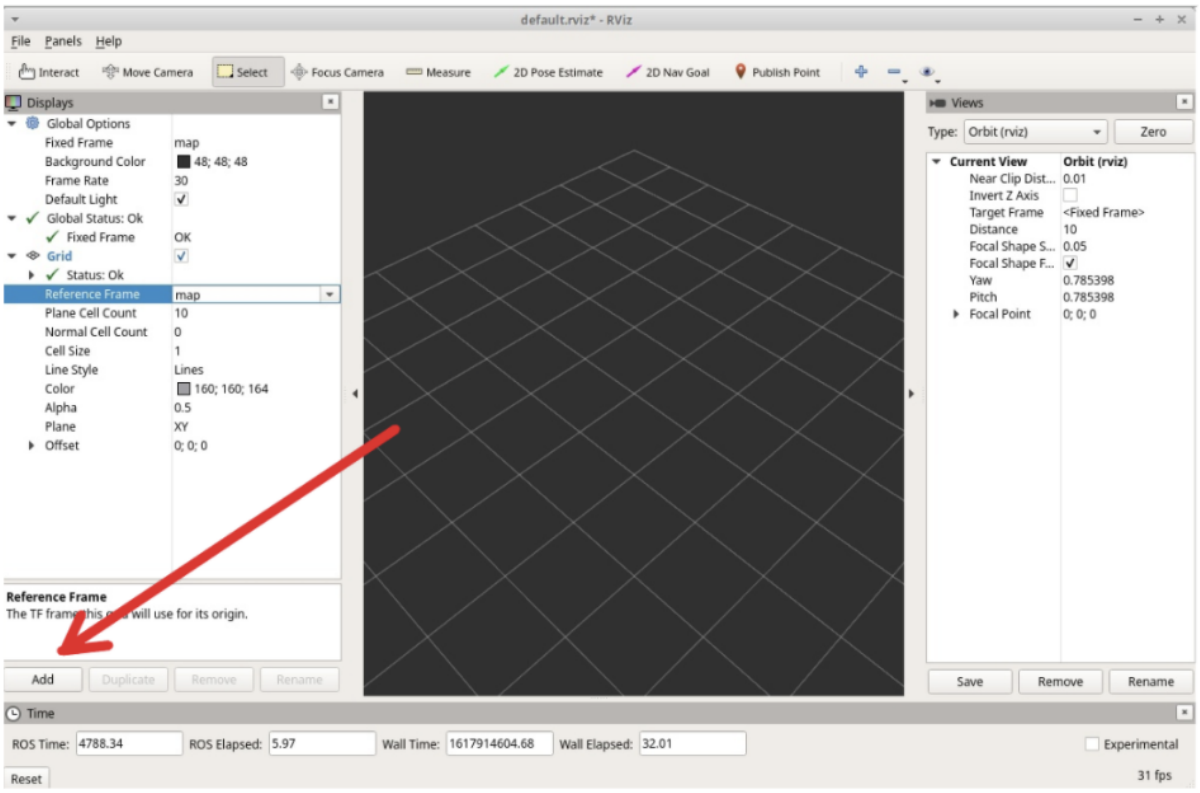
- `world.world`;
- `aruco.launch`;
- `main_camera.launch`;
- `clover.launch`;
- `aruco_map.txt`;
- `simulator.launch`;
- папка `models`.

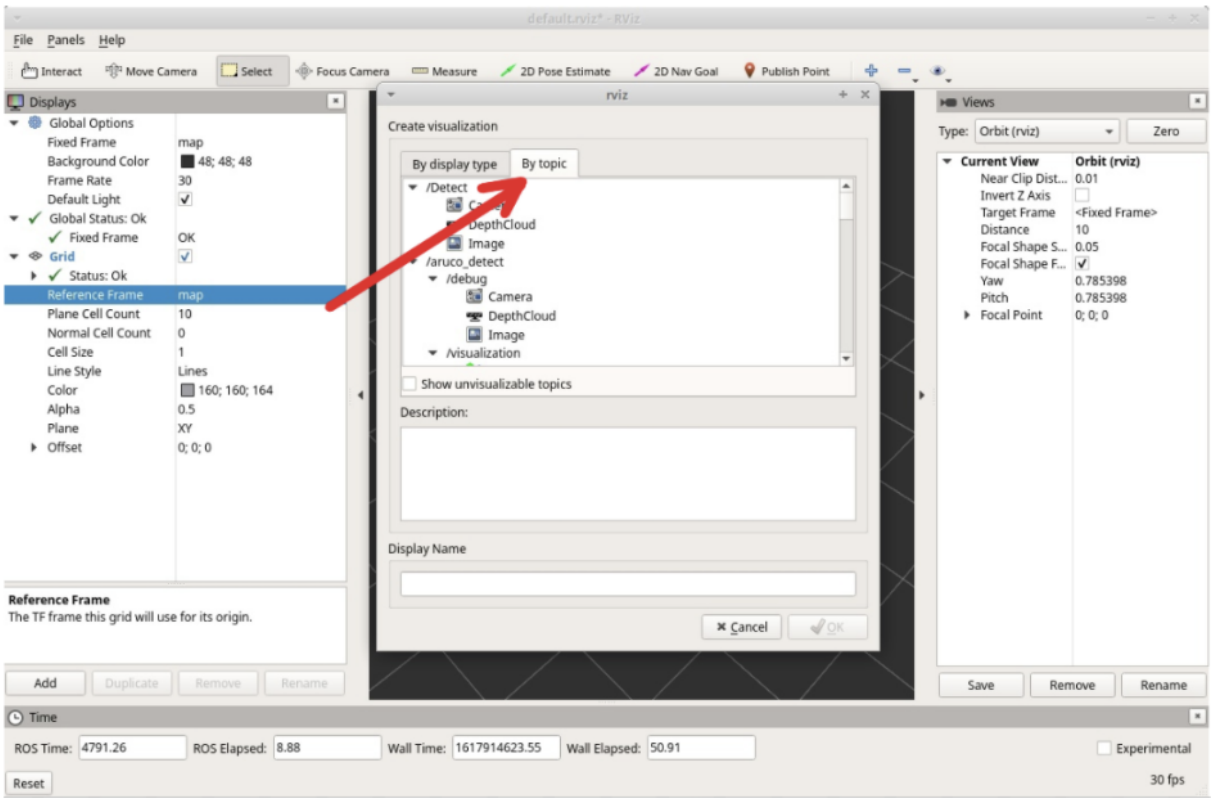
Визуализация финальной зачетной попытки при помощи инструмента Rviz.

- Необходимо осуществить запись значений с топиков во время выполнения финальной миссии 3-го зачетного дня с помощью команды: `rosbag record -a`.
- Скачать полученный в формате *.bag и сохранить его с названием «fly.bag».
- Открыть терминал в симуляторе или другой среде где установлен Ros и rviz.
- Запустить в терминале симулятора команду: `rosbag play fly.bag -loop`.
- Запустить в терминале rviz, командой: `rosrun rviz rviz`.
- Выставить «Reference frame» значение «map».

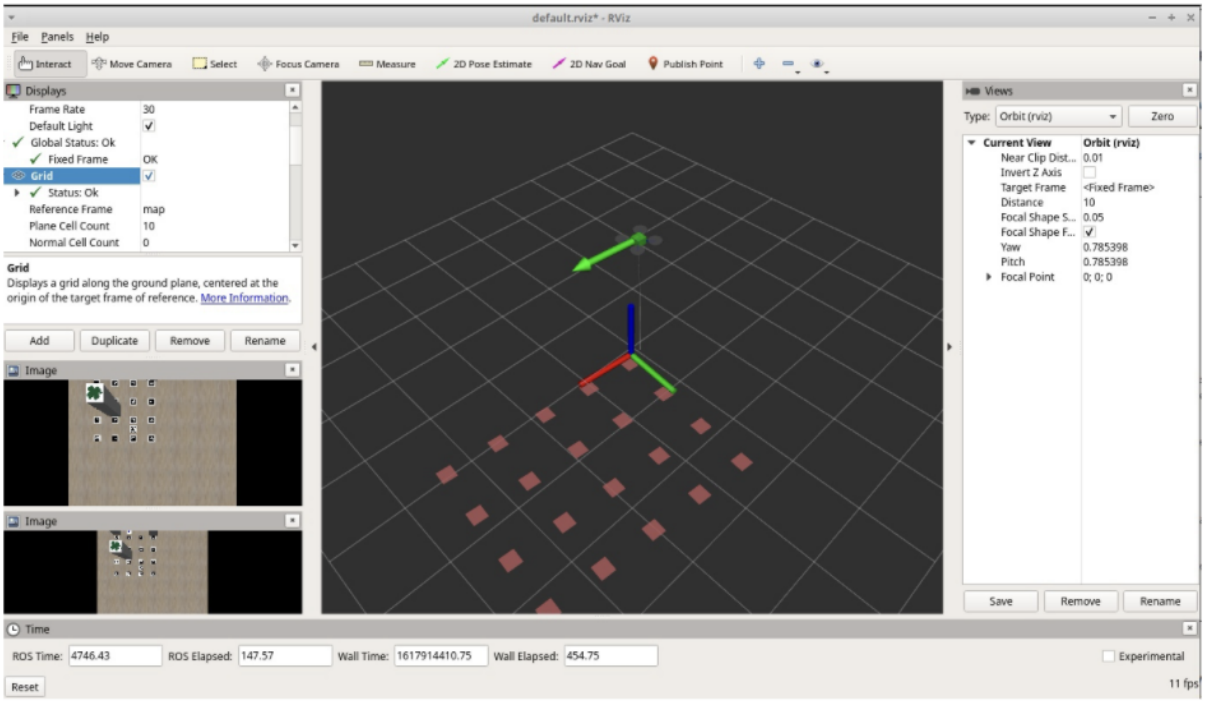


- Нажать на кнопку «Add» и выбрать «By topic»:

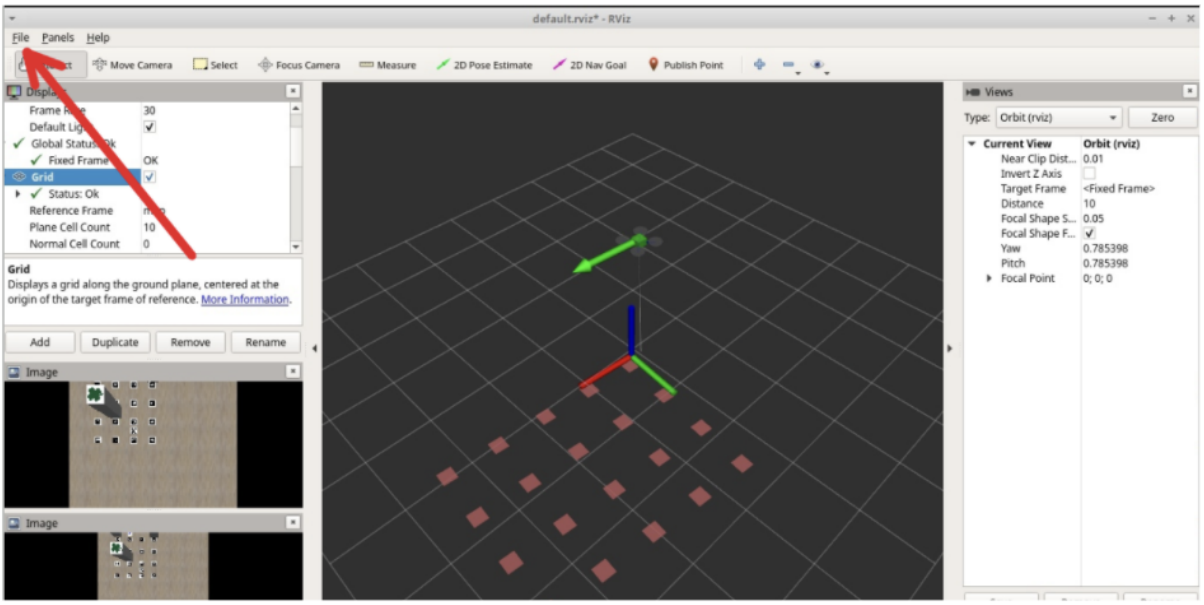


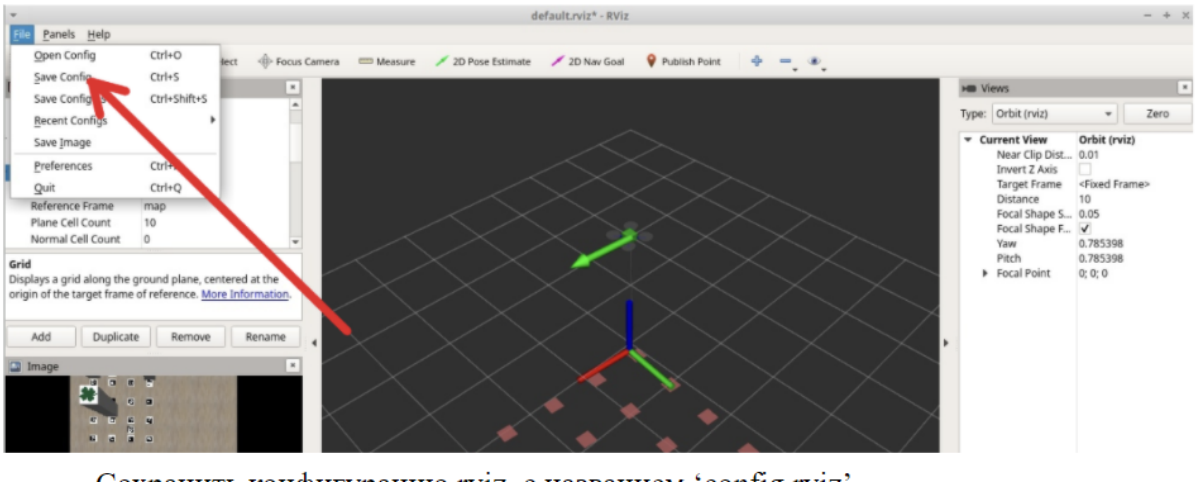


- Добавить все топики которые требовались в соответствии с их «Display Name»:
- `/aruco_map/visualization` — MarkerArray;
- `/wp_markers` — Marker;
- `aruco_map` — Axes;
- `/main_camera/image_raw` — Image;
- `/Detect` — Image;
- `/vehicle_marker` — MarkerArray;
- `/rangefinder/range` — Range.

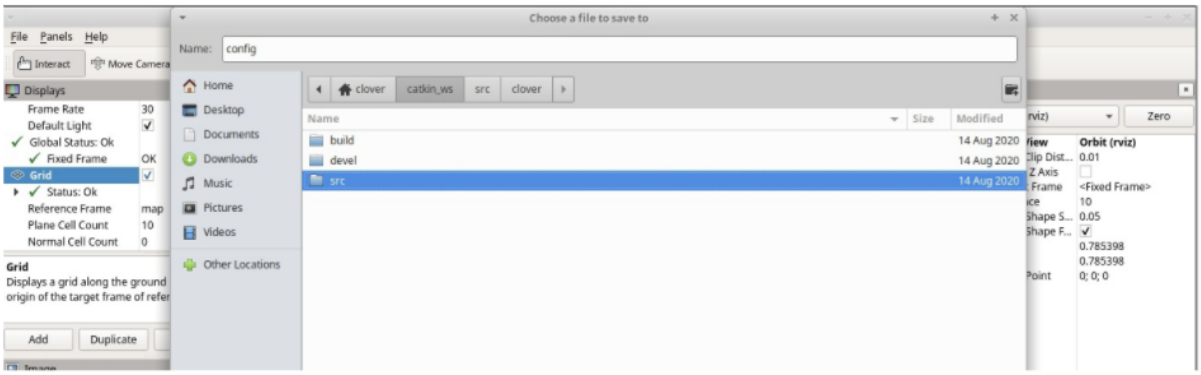


- Нажать на вкладку «File» и выбрать «Save Config».





- Сохранить конфигурацию rviz, с названием «config.rviz».



Собираем созданные файлы в папку, находящуюся на Google Drive, согласно заданию:

- *video.mp4/video.avi*;
- *fly.bag*;
- *config_rviz.rviz.*