# Задание на собеседование. Практическая часть часть

## 1 Программирование для манипулятора

### 1.1 Условия


Рис. 1: Манипулятор Z1

Манипулятор Z1 - шестизвенный манипулятор от компании Unitree, изготовленный специально под робособак Unitree B1. Официальная документация: https://dev-z1.unitree.com/ - Здесь также находятся инструкции по управлению манипулятором.

Для отправки команд на манипулятор из программного кода действует следующая архитектура: Манипулятор имеет свой ip адрес и выделенные udp порты. Манипулятор соединяется с ПК по интернету. На ПК запускается программа Z1_controller (https://github.com/unitreerobotics/z1_controller) , которая связывается с манипулятором по ip и определённым сокетам по UDP. Далее, Z1_controller слушает уже свои сокеты на localhost. С помощью SDK манипулятора (https://github.com/unitreerobotics/z1_sdk), выполненную как библиотека c++ или python, можно отправлять на сокеты z1_controller команды, которые z1_controller уже перенаправит в манипулятор. Также в программе z1_controller реализовано управление с клавиатуры для проверки работоспособности.

Вам предоставлен докер контейнер, который содержит симуляцию манипулятора Z1 в Gazebo, которая полностью повторяет работу на реальном манипуляторе (подключение через z1_controller и прочее). Подробнее о докере в приложении. При решении задач можно пользоваться также функционалом SDK, позволяющим решать ОКЗ и пр. (ArmModel.h)

## Задание №1

Дан CSV файл с 7 столбцами и время выполнения траектории.

В каждой строке CSV файла - 7 углов манипулятора в радианах, от первого до 7, (7 - это угол поворота захватного устройства). Номер столбца соответствует номеру сочленения (например, первом столбце записаны углы первого сочленения). Каждая строка задает точку в конфигурационном пространстве манипулятора (конфигурационное пространство - 7 мерное пространство, по осям которого - углы поворота звеньев). В CSV файле последовательностью точек в конфигурационном пространстве задана траектория манипулятора. Траектория манипулятора - функция углов манипулятора от времени. Точки траектории находятся на одинаковом удалении друг от друга во времени.

Время выполнения траектории t - время, при котором манипулятор находится в последней точке записанной траектории. То есть в момент времени 0 манипулятор находится в начальном положении, а в момент времени t манипулятор находится в положении, задаваемое последней строкой в csv файле. Остальные точки находятся равноудалённо по времени. Например, если задано 10 строк и 9 секунд, то 1 строка - положение в 0 секунде, 2 строка - положение в 1 секунде ,..., 9 строка - положение в 8 секунде, 10 строка - положение в 10 секунде). Манипулятор должен выполнять траекторию плавно, не останавливаясь в какой-либо точке. Гарантируется, что в csv файле 1 строка будет соответствовать положению манипулятора в момент выполнения программы.

Необходимо написать программу на C++, принимающую на вход 2 аргумента: путь до csv файла и время выполнения траектории в сек. В симуляции в Gazebo должно быть видно выполнение манипулятором траектории. Далее необходимо получить график ошибки от времени для выполненной траектории. График можно строить с помощью python.

## Задание №2

Дана конечная позиция манипулятора (координаты и углы схвата). Необходимо найти collison-free путь от стартового положения манипулятора в начальное положение, а затем из начального в конечное. На сцене присутствует препятствие в виде сферы радиусом 0.5м, центр которой находится в (x = 0.75м, y=0м,z=-0м) относительно базовой системы координат. При проверке коллизий достаточно немного увеличить радиус сферы и представить манипулятор как набор отрезков. Но можно использовать любой coliision check, в том числе и готовые библиотеки (не нужен псевдокод).

Начальное положение joint space q от 1 до 7 соответственно: -1.08675 2.96708 -2.62189 -0.43586 0.00026 -0.03351

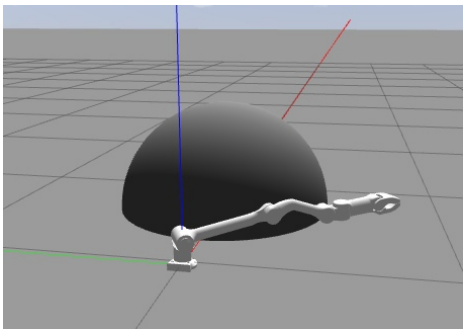


Рис. 2: Начальное положение

Конечное положение joint space q от 1 до 7 соответственно: 1.17255 2.96708 -2.62106 -0.43589 0.00013 -0.03345

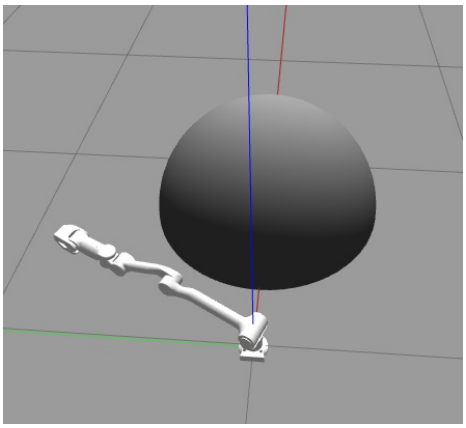


Рис. 3: Конечное положение

Необходимо написать программу на C++, строящий collision-free траекторию в виде csv файла, который подаётся в первую программу. Время можно задать любое, но лучше рассчитать его исходя из ограничений по скорости (ограничения выбрать самим). Можно использовать всё, что угодно. При использовании готовых планировщиков необходимо предоставить псевдокод. Использование MoveIt допускается только с Ros2.

# 2 Приложение

## Настройка окружения

1. Склонировать репозиторий https://github.com/kaizer-nurik/Intert_practice_tast

```
git clone https://github.com/kaizer-nurik/Intert\_practice\_tast
```

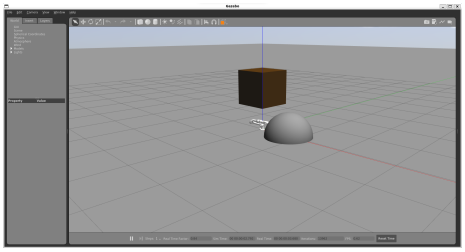
h 

Рис. 4: Окно gazebo, которое откроется при старте контейнера.

2. Инициализировать сабмодули. В директории репозитория выполнить:

   ```
   git submodule init
   git submodule update
   ```

3. Собрать докер изображение:

   ```
   docker compose build
   ```

4. Запустить докер контейнер:

   ```
   docker compose up
   ```

## Управление манипулятором с клавиатуры

Нужен запущенный докер.

1. Открыть bash среду контейнера

   ```
   docker exec -it z1_sim_container bash
   ```

2. Перейти в директорию с контроллером

   ```
   cd /z1_package/z1_controller/build
   ```

3. Запустить контроллер для симуляции в режиме управления с клавиатуры

   ```
   ./sim_ctrl k
   ```

   Или

4. Запустить контроллер для симуляции в режиме управления с командр sdk

```
./sim_ctrl
```

Инструкция по управлению с клавиатуры: https://dev-z1.unitree.com/use/keyboard.html
Например, нажав на "2 можно перейти в режим управления по звеньям, и нажимая соответствующие кнопки на клавиатуре, можно двигать определёнными звеньями.

| Joint ID | 0 | 1 | 2 | 3 | 4 | 5 | Gripper |
|---|---|---|---|---|---|---|---|
| Keyboard | Q/A | W/S | D/E | R/F | T/G | Y/H | up/down |
| Joint Action (right hand) | positive/ negative | positive/ negative | positive/ negative | positive/ negative | positive/ negative | positive/ negative | positive/ negative |

Рис. 5: Кнопки для управления звеньями с клавиатуры.