

**Федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО»**

Факультет систем управления и робототехники

**Отчет о
научно-исследовательской работе**

по теме:

**«РАЗРАБОТКА АЛГОРИТМОВ ДЛЯ ВЗАИМОДЕЙСТВИЯ
С РОБОТОМ-МАНИПУЛЯТОРОМ С КОМПЬЮТЕРА (С
ИСПОЛЬЗОВАНИЕМ TCP)»**

Выполнил: студент гр. Р3341
А. А. Румянцев

Проверил: преподаватель
доцент, старший научный сотрудник, инженер В. С. Громов

Санкт-Петербург
2025 г.

Содержание

Введение	4
1 Подготовка к написанию программы	5
1.1 Ознакомление с объектом работы	5
1.2 Выбор подходящего языка программирования	5
2 Реализация программы для взаимодействия компьютера с роботом-манипулятором	6
2.1 Определение формата сообщения	6
2.2 Определение основных команд	7
2.3 Клиентская программа для общения с роботом	8
2.4 Программа на роботе для общения с клиентом	9
2.5 Проверка связи между компьютером и роботом	10
2.6 Клиентская программа для управления роботом	12
2.7 Программа управления робота при взаимодействии с клиентом .	15
3 Добавление пользовательского интерфейса	16
3.1 Описание интерфейса	16
3.2 Взаимодействие интерфейса и основной логики программы . . .	16
4 Тестирование работы программы	18
4.1 Проверка работы программы	18
4.2 Проблемы в ходе тестирования	22
5 Заключение	22
6 Список использованных источников	23
А Приложение	23
Б Приложение	24
В Приложение	25
Г Приложение	26

Д Приложение	27
Е Приложение	29
Ж Приложение	30
З Приложение	31
И Приложение	32

Введение

В настоящее время в промышленной и других сферах все чаще используются роботы-манипуляторы, управляемые со специального пульта или автоматически через загрузку программы на робота. Более предпочтительным является вариант управления без человека – это безопаснее и выгоднее. Однако роботы используют достаточно устаревший язык программирования, например MELFA-BASIC. Написание кода для подобных роботов может быть неудобным, а программы получаться громоздкими. Разработка нового языка программирования для роботов потребует больших вложений, что также не выгодно. Управление с пульта, в свою очередь, требует от оператора высокой квалификации – необходимы знания техники безопасности и принципы работы оборудования. Обучение специалиста для управления роботом-манипулятором с пульта является ресурсоемким процессом, требующим значительных временных и финансовых затрат.

Для повышения безопасности и эффективности взаимодействия с роботом, необходимо максимально отдалить человека от робота, при этом реализовать все основные функции для работы с ним так, чтобы их можно было использовать из некой виртуальной централизованной системы по нажатию кнопок. Реализовать данную идею можно в виде программного интерфейса – аналога физического пульта управления роботом в виде программы на компьютере. Такой подход также позволит упростить обучение специалистов для управления роботом-манипулятором. Кроме того, программу можно купить один раз и установить на множество компьютеров, а разработка и покупка нескольких физических пультов управления будет ресурсозатратным процессом. Однако сейчас программных интерфейсов, позволяющих взаимодействовать с роботом с компьютера сравнительно немного, а те, что уже есть, постепенно устаревают. Возникает необходимость написания нового программного интерфейса для взаимодействия с роботом. Как и любая другая программа, структурно она делится на две части – одна отвечает за внешний вид и удобство управления (интерфейс), другая же обеспечивает взаимодействие с роботом на уровне, не видном пользователю. В рамках данной работы разрабатывалась внутренняя логика программы для взаимодействия компьютера с роботом-манипулятором по протоколу TCP. Пользовательская часть интерфейса при этом рассматривалась как вспомогательная.

1. Подготовка к написанию программы

1.1. Ознакомление с объектом работы

Перед выполнением задания был проведен инструктаж по технике безопасности обращения с роботом-манипулятором.

Под наблюдением преподавателя были изучены ручной режим управления роботом со специального пульта и автоматический с помощью простейших программ на языке MELFA-BASIC, загружаемых на робота.

Для написания программ для робота был изучен язык программирования MELFA-BASIC, некоторые его основные команды и описание представлены далее:

- **SERVO ON** – включение двигателей,
- **SERVO OFF** – выключение двигателей,
- **END** – завершение программы, обязательно размещается в конце файла,
- **JOVRD 100** – скорость движения в процентах от максимальной,
- **SPD 100** – скорость движения при интерполяционных командах,
- **MOV P1** – движение в заданную точку P1,
- **WHILE, FOR** – циклы с условиями,
- **OPEN "COM3:"AS #1** – открытие TCP/IP порта 10003 для подключения интерфейса #1,
- **CLOSE #1** – закрытие TCP/IP порта 10003 для подключения интерфейса #1,
- **DEF INTE DCOMM** – объявление переменной DCOMM целочисленного типа.

1.2. Выбор подходящего языка программирования

Существует достаточно много различных языков программирования, подходящих под реализацию задачи взаимодействия с роботом с компьютера. В рамках данной работы был выбран язык программирования Python, так как он достаточно часто используется в сфере робототехники, имеет достаточно простой и легкочитаемый синтаксис, имеет большое количество готовых библиотек и является кроссплатформенным (программу можно запустить на разных операционных системах).

В ходе выполнения работы использовались следующие библиотеки:

- **socket** – для работы с сетевыми соединениями,
- **typing** – средства для статической типизации переменных и функций,
- **re** – модуль для работы с регулярными выражениями,
- **yaml** – для чтения и записи файлов в формате YAML,
- **enum** – позволяет создавать перечисления с именованными значениями.

Библиотека `socket` понадобилась для установки TCP-соединения между компьютером и роботом и передачи/получения пакетов.

Для общего улучшения и упрощения кода использовалась библиотека `typing`.

Модуль `re` понадобился для обработки ответов с робота.

Библиотека `yaml` позволила реализовать чтение и сохранение введенных настроек IP и порта, чтобы пользователю не пришлось каждый раз вводить эти данные заново при запуске программы.

Для перечисления команд, статусов сетевого взаимодействия с роботом, декартовых и сочлененных координат понадобилась библиотека `enum`.

2. Реализация программы для взаимодействия компьютера с роботом-манипулятором

2.1. Определение формата сообщения

Для начала необходимо определиться с форматом передаваемого с компьютера на робот-манипулятор сообщения и обратно.

Сообщение должно быть простое, соответствующее шаблону которое понимает робот.

Сначала будет отправляться номер команды, после чего шесть координат и пара чисел для решения обратной задачи кинематики, если хотим передвижение по декартовым координатам. Если движение сочлененное, то достаточно передать шесть углов вращения.

Было решено, что робот будет отправлять компьютеру 12 координат и пару для решения обратной задачи кинематики как одно сообщение (изменение декартовых координат влияет на сочлененные и наоборот).

В ходе выполнения работы экспериментальным путем было выяснено, что робот отправляет свои декартовы и сочлененные координаты в следующем

формате:

$$[(J_1, J_2, J_3, J_4, J_5, J_6)(X, Y, Z, A, B, C)(K_1, K_2)],$$

то есть как список, содержащий одну строку.

Это означает, что роботу нужно отправлять координаты в виде строк шаблона:

- $(J_1, J_2, J_3, J_4, J_5, J_6)$ – если движение сочлененное,
- $(X, Y, Z, A, B, C)(K_1, K_2)$ – если движение в декартовых координатах, при этом все значения с плавающей точкой, кроме K_i – они целочисленные.

2.2. Определение основных команд

Проще всего отправлять роботу не строковые команды вида 'EXIT', а численные в виде строк. Например, команда '0' – завершение работы робота.

Создадим для удобства перечисление enum, где каждой команде с названием будет присвоено собственное число.

```
1 class RobotCommand(Enum):
2     EXIT = 0
3     GET_POSITION = 1
4     MOVE_LINEAR = 2
5     MOVE_JOINTS = 3
```

Листинг 1: Определение перечисления с командами для робота.

Краткое описание команд:

- '0' – завершение работы робота. На клиентской стороне отключение связи,
- '1' – робот вышлет 12 своих координат и кинематическую пару,
- '2', '3' – движение робота по декартовым координатам или сочлененное соответственно. После робот снова высыпает свою позицию.

В программе для робота будем проверять пришедшее сообщение на совпадение с одним из этих чисел. Далее будет выполняться соответствующий алгоритм.

Например, при получении '2' робот будет двигаться по декартовым координатам в соответствии с присланной после команды в заданном шаблоне следующей позицией робота, после чего вышлет обратно 12 своих координат и пару чисел для решения обратной задачи кинематики.

2.3. Клиентская программа для общения с роботом

Напишем клиентский скрипт, который будет основой сетевого взаимодействия компьютера с роботом.

Для начала определим статусы сетевого взаимодействия как перечисление. Они помогут при отладке ошибок.

```
1 class ConnectionStatus(Enum):
2     SUCCESS = auto()
3     NOT_CONNECTED = auto()
4     CONNECTION_ERROR = auto()
5     SEND_ERROR = auto()
6     RECEIVE_ERROR = auto()
7     NONE = auto()
```

Листинг 2: Определение перечисления статусов соединения компьютера с роботом.

Нумерация статусов автоматически с 1 и по порядку. Смысл статусов полностью соответствует их названиям.

Теперь реализуем основные функции для взаимодействия компьютера с роботом. Полностью скрипт представлен в приложении А на листинге 9.

Краткое описание функций скрипта:

- **`__init__(self)`** – при создании объекта класса вызывается данная функция. Она присвоит объекту поля: `sock` – сетевой сокет с параметрами `AF_INET` (семейство адресов IPv4), `SOCK_STREAM` (протокол TCP); булевое `connected` – значение истина или ложь в зависимости от того, подключен ли сокет к указанному IP и порту;
- **`connect(self, ip: str, port: int) -> ConnectionStatus`** – подключение сокета к указанному IP адресу и порту. Возвращает статус соединения. Также присваивает значение "истина" переменной `connected`, если подключение удалось;
- **`disconnect(self) -> ConnectionStatus`** – закрывает соединение, если оно есть. Возвращает статус соединения.
- **`send(self, data: str) -> ConnectionStatus`** – кодирует переданную строку сообщение `data` и отправляет по протоколу TCP. Возвращает статус соединения;
- **`receive(self, out_data: list, buffer_size: int = 1024) -> ConnectionStatus`** –

принимает данные по протоколу TCP. Записывает в декодированном виде полученную информацию в список out_data. Возвращает статус соединения.

2.4. Программа на работе для общения с клиентом

На языке MELFA-BASIC напишем простую программу для теста связи между компьютером и роботом:

```
1 JOVRD 100
2 SPD 100
3 DEF INTE DCOMM
4 DCOMM = 1
5 PHELP = P_CURR
6 SERVO ON
7 OPEN "COM3:" as #1
8
9 WHILE DCOMM > 0
10   INPUT #1, DCOMM
11   IF DCOMM = 2 THEN
12     INPUT #1, PHELP
13     PRINT #1, PHELP
14   ENDIF
15 WEND
16
17 CLOSE #1
18 SERVO OFF
19 END
```

Листинг 3: Простая программа на работе для проверки связи с клиентом.

Краткое описание программы: в переменную DCOMM ожидается поступление номера команды от компьютера. Когда команда '2' пришла, робот записывает приходящие следующим же сообщением от компьютера декартовы координаты и кинематическую пару, после чего высылает обратно то же самое, что и получил.

Если прислали команду '0', то робот выйдет из цикла и завершит работу.

Таким образом проверяется, что робот корректно обрабатывает полученные координаты, компьютер верно получает те же координаты обратно. После чего робот успешно завершает работу.

2.5. Проверка связи между компьютером и роботом

Сообщение роботу нужно отправить в определенном формате. Также ответ от робота нужно обработать. Для этого напишем скрипт, который будем использовать каждый раз перед отправкой и получением координат. Полностью скрипт представлен на листинге 10 в приложении Б.

Краткое описание скрипта:

- **build_position_request(pos: List[float]) -> str** – на вход получает координаты как список чисел с плавающей точкой. Преобразует список в строку в соответствии с шаблоном $(v_1, v_2, v_3, v_4, v_5, v_6)$. Возвращает результат преобразования;
- **parse_position_response(response: list)** – получает ответ от робота в виде списка из строки с 12 координатами и кинематической пары и с помощью регулярного выражения делит строку внутри списка на три отдельных списка из сочлененных и декартовых координат и кинематической пары. Возвращает кортеж из трех соответствующих списков.

Также нам нужен скрипт для записи и хранения IP и порта робота. Создадим yaml файл, в котором запишем:

```
1 connection:  
2   ip: "192.168.0.4"  
3   port: 10003
```

Листинг 4: Файл, где хранятся данные для подключения к роботу-манипулятору.

Напишем скрипт, в котором во время работы программы будут сохранены эти данные. Полностью скрипт представлен в приложении В на листинге 11.

Краткое описание скрипта:

- **load(cls, path: str = "config.yaml")** – загружает IP адрес и порт из файла по указанном пути. Сохраняет в свои поля ip, port. По умолчанию файл находится в том же каталоге, где и скрипт;
- **save(cls, path: str = "config.yaml")** – сохраняет конфигурацию IP и порта в файл по указанному пути;
- **set_config(cls, ip: str, port: int)** – меняет конфигурацию IP и порта во время работы программы. Может пригодиться, если необходимо будет отключиться от одного робота и подключиться к другому, не закрывая программу.

Напишем простую программу для компьютера для проверки соединения с роботом:

```
1 import socket
2 from config_manager import Config
3 from command_manager import CommandMessageManager
4
5 if __name__ == "__main__":
6     Config.load()
7
8     pos = [100.0, -100.0, 100.0, 100.0, 100.0, 100.0]
9     test = CommandMessageManager.build_position_request(pos)
10    test += "(7,0)"
11    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12    s.connect((Config.ip, Config.port))
13    s.sendall("2".encode('utf-8'))
14    s.sendall(test.encode('utf-8'))
15    ans = s.recv(1024).decode('utf-8')
16    print(ans)
17    s.sendall("0".encode('utf-8'))
18    s.close()
```

Листинг 5: Простая программа на клиенте для проверки соединения с роботом.

Краткое описание программы: загружается конфигурация IP и порта из файла. Задается некоторый набор из шести координат, из которого создается строка на отправку. Так как отправляются декартовы координаты, необходимо еще к строке на отправку добавить в строковом виде в таком же формате кинематическую пару. После чего создается сокет с полученными из файла IP адресом и портом. Отправляется закодированная команда '2', за которой следуют закодированные обработанные декартовы координаты. После чего принимается, декодируется и выводится в консоль ответ от робота. В конце высылается закодированная команда '0' для завершения работы робота. На клиенте же закрывается сокет.

В присутствии преподавателя на робота была загружена программа, представленная на листинге 3. На компьютере была запущена программа, представленная на листинге 5.

Было выслано с компьютера:

(100.0, -100.0, 100.0, 100.0, 100.0, 100.0) (7, 0)

Было получено от робота:

$$(+100.0, -100.0, +100.0, +100.0, +100.0, +100.0) (7, 0)$$

Робот и компьютер взаимодействуют корректно – на компьютер пришло то же, что было с него выслано. Появились дополнительные плюсы, которые при обработке ответа от робота пропадут, то есть они незначащие. Ответ робота был выведен в консоль в сыром виде.

2.6. Клиентская программа для управления роботом

Доработаем нашу программу скриптами с функциями, позволяющими полноценно взаимодействовать с роботом с компьютера.

Для начала введем перечисления для декартовых и сочлененных координат.

После обработки ответа робота мы получаем, например, список из шести декартовых координат. Чтобы подвинуться по какой-то из координат, необходимо добавить шаг к этой координате. Координаты в списке располагаются по порядку. Значит, при обращении к нулевому элементу получим первую координату X , при обращении к пятому получим шестую координату C . Аналогично с сочлененными координатами.

```
1 class CartesianAxis(Enum):
2     X = 0
3     Y = 1
4     Z = 2
5     A = 3
6     B = 4
7     C = 5
8
9 class JointAxis(Enum):
10    J1 = 0
11    J2 = 1
12    J3 = 2
13    J4 = 3
14    J5 = 4
15    J6 = 5
```

Листинг 6: Определение перечислений, где каждой координате соответствует свой индекс в списке.

Также зададим тип координатам. Есть декартовы линейные, декартовы вращательные и сочлененные координаты.

- Линейные декартовы координаты X, Y, Z – mode: cartesian_linear,
- Вращательные декартовы координаты A, B, C – mode: cartesian_rotation,
- Сочлененные координаты $J_1 \dots J_6$ – mode: joint_rotation.

Для определения команды из перечисления по ее типу напишем такой словарь:

```
1 MODE_TO_COMMAND = {  
2     "cartesian_linear": RobotCommand.MOVE_LINEAR,  
3     "cartesian_rotation": RobotCommand.MOVE_LINEAR,  
4     "joint_rotation": RobotCommand.MOVE_JOINTS,  
5 }
```

Листинг 7: Словарь для приведения типа координаты к команде из перечисления.

Нам необходимо в программе на компьютере где-то хранить координаты робота и кинематическую пару. Создадим виртуального робота на клиенте. Скрипт расположен в приложении Г на листинге 12.

Краткое описание функций скрипта:

- **__init__(self, cartesian: ..., joint: ..., kin_sol: ...)** – при создании объекта класса в качестве его полей можно указать декартовы и сочлененные координаты и кинематическую пару. Иначе они будут заданы нулевыми;
- **update_cartesian(self, new_values: ..., kinematic_sol: ...)** – обновляет декартовы координаты и кинематическую пару виртуального робота;
- **update_joint(self, new_values: ...)** – обновляет сочлененные координаты виртуального робота;
- **calculate_next_move(self, mode: ..., axis_index: int, step: float) -> ...** – проверяет тип координаты и считает соответствующие следующие координаты робота. Функция принимает тип координаты, ее индекс в списке координат и шаг, на который нужно сдвинуть робота по этой координате (миллиметры для линейных координат, градусы для вращательных). Возвращает список соответствующих координат, где одно из значений смешено на некоторый шаг.

Так как обратная задача кинематики на клиенте не решается, то кинематическая пара нужна только для того, чтобы выслать ее обратно вместе с

декартовыми координатами роботу.

В интерфейсе подразумевается наличие кнопок каждой координаты в двух типах: со знаком минус и со знаком плюс. Для упрощения логики программы введем кортеж, который будем называть *move_info*. В нем будут содержаться тип координаты, ее порядковый номер в списке и направление, задаваемое знаком плюс или минус. Примеры кортежей приведены далее.

- ("cartesian_linear", **CartesianAxis.X.value**, '+') – декартова линейная координата *X*, порядковый номер определяется перечислением декартовых координат (в этом случае будет 0) и положительное направление;
- ("cartesian_rotation", **CartesianAxis.A.value**, '-') – декартова вращательная координата *A*, порядковый номер в списке определяется аналогично (в данном случае 4) и отрицательное направление;
- ("joint_rotation", **JointAxis.J1.value**, '+') – сочлененная вращательная координата *J₁*, порядковый номер 0 и положительное направление.

Теперь для всего, что мы уже написали на клиенте, нужна простая надслойка – скрипт, функции которого будут объединять несколько действий, необходимых для полноценного взаимодействия с роботом. Эту надслойку можно будет использовать, например, в обработчиках нажатия кнопок в графическом интерфейсе. Скрипт представлен в приложении Д на листинге 13.

Краткое описание функций данного скрипта:

- **__init__(self)** – создает объект класса и присваивает поля: *robot* – виртуальный робот, *client* – клиентская обертка над сокетом и *last_..._status* – статус сетевого взаимодействия для каждой последней операции компьютера с роботом (присоединение, отправка, получение и отключение; может пригодиться для удобной отладки сетевых проблем). Далее во всех функциях присваивание сетевых статусов аналогичное;
- **connect(self, ip: str, port: int)** – обертка подключения над клиентской оберткой подключения над сокетом. Возвращает булевую переменную *connected* объекта класса Client. Дополнительная обертка позволяет при необходимости добавить некоторую логику до прямого соединения с роботом;
- **disconnect(self)** – обертка отключения над клиентской оберткой отключения над сокетом. Возвращает отрицание булевой переменной *connected* объекта класса Client. В функцию до прямого отключения от робота дополн-

- нительно добавлена логика оповещения робота, что клиент отключается – высыпается команда '0' для завершения работы робота;
- **is_connected(self)** – возвращает булевую переменную connected объекта класса Client. Непрямая проверка соединения. Есть возможность добавить дополнительную логику;
 - **get_pos(self)** – высыпает команду '1' роботу, чтобы получить его текущие координаты и кинематическую пару. Возвращаеет ответ робота без обработки. Подразумевается, что обработка будет сделана извне этой функции, ее методы могут быть любыми в соответствии с нуждами;
 - **move_axis(self, move_info: ..., step: float)** – принимает кортеж из трех элементов, описанных ранее, и шаг. По направлению определяет знак шага. Вызывает функцию calculate_next_move виртуального робота, куда передает тип координаты, ее порядковый номер и шаг (подразумевается, что в виртуальном роботе уже сохранены координаты настоящего). Далее отправляет на робот номер команды в соответствии с конвертацией из словаря MODE_TO_COMMAND. С помощью build_position_request создает сообщение роботу с новыми координатами, полученными от виртуального робота. Если тип координат декартовый, то добавляет в конец сообщения кинематическую пару в соответствии с форматом сообщения. После этого отправляет номер команды и координаты. Получает ответ от робота и возвращает его без обработки.

Алгоритмы на клиенте готовы. Осталось написать алгоритм на роботе и добавить интерфейс с кнопками.

2.7. Программа управления робота при взаимодействии с клиентом

Напишем программу для робота на языке MELFA BASIC для реализации выполнения роботом команд, присыпаемых с клиента. Полностью программа представлена в приложении Е на листинге 14.

Краткое описание программы: робот ждет номер команды в переменную DCOMM. Есть вспомогательные переменные, куда будут записываться приходящие с клиента координаты – PHELP для декартовых, JHELP для сочлененных. Пока на робота не отправили команду '0', он в цикле ждет следующую команду. При получении команды '1', робот вышлет обратно свои текущие 12 координат и кинематическую пару. Если придет команда '2', то робот примет декартовы

координаты и кинематическую пару сразу после команды, с помощью MOV переместится в новые декартовы координаты, далее отправит ответ о своих текущих координатах (они должны соответствовать тем, что были отправлены). Аналогично при получении команды '3', однако координаты сочлененные и не нужна кинематическая пара. Если приходит команда '0', то цикл завершается, робот выключается, предварительно закрыв соединение.

3. Добавление пользовательского интерфейса

Данная практическая работа является частью общего проекта на тему "Разработка графического интерфейса для взаимодействия с роботом-манипулятором". Этот проект делится на две подтемы – тема данной работы и "Разработка графического интерфейса программы". Вторая подтема была выполнена студентом, с которым выполнялся общий проект. Следовательно, в данной практической работе пользовательская часть интерфейса не будет подробно рассматриваться, так как в данной теме она является вспомогательной. Разработанные алгоритмы взаимодействия с роботом возможно использовать в различных графических интерфейсах.

3.1. Описание интерфейса

Краткое описание интерфейса: в интерфейсе есть кнопки на каждую координату в двух видах – с плюсом и с минусом. Шаг задается ползунками – в миллиметрах для линейных координат, в градусах для вращательных координат. Есть возможность задавать и менять IP адрес и порт в окне программы. Есть кнопка подключения, которая после успешного подключения меняется на кнопку отключения. Пока подключение к роботу не осуществлено, интерфейс остается недоступным.

3.2. Взаимодействие интерфейса и основной логики программы

Рассмотрим взаимодействие интерфейса и основной логики программы.

Каждой кнопке в окне программы соответствует свой move_info. Структурно это словарь, который можно назвать картой кнопок окна программы. Реализация представлена на листинге 15 в приложении Ж.

При полном нажатии кнопки connect, вызывается функция обработки нажатия данной кнопки handle_connection(self). Ее реализация представлена на листинге 16 в приложении З.

Краткое описание функции handle_connection(self): проверяет функцией is_connected, есть ли подключение компьютера к роботу. Если подключения нет (например, первый запуск программы), то берется сохраненная конфигурация IP и порта, вызывается функция connect у надслойки. Если подключение не удалось, то интерфейс останется заблокированным. Чтобы снова попытаться подключиться, необходимо будет нажать кнопку connect еще раз. Если подключение удалось, то на робот сразу высылается запрос на получение координат через функцию надслойки get_pos. Далее ответ обрабатывается функцией parse_position_response. После обработки полученные координаты и кинематическая пара присваиваются виртуальному роботу, также вызывается функция обновления отображаемых в интерфейсе координат робота. Эта функция берет координаты из виртуального робота и выводит их в окно программы. В конце интерфейс разблокируется, кнопка connect меняется на disconnect. Если подключение уже есть и кнопка нажата еще раз, то произойдет отключение клиента от робота через надслойку disconnect, то есть на робот будет отправлена команда '0' и он выключится.

При нажатии кнопок изменения координат вызывается обработчик handle_axis_button(self). Реализация данной функции представлена на листинге 17 в приложении И.

Краткое описание функции handle_axis_button(self): берет из карты кнопок с move_info кортеж move_info по имени нажатой кнопки. Проверяет, какой тип координаты находится в этом кортеже. Если тип декартовый линейный, то берет шаг с ползунка с миллиметровой шкалой. Если тип декартовый вращательный, то берет шаг с ползунка для декартовых вращательных координат с градусной шкалой. Если тип сочлененный, то аналогично декартовым вращательным координатам, но ползунок для сочлененных координат. Далее вызывает функцию move_axis, куда передает move_info и шаг, обрабатывает ответ функцией parse_position_response, записывает в виртуального робота пришедшие от настоящего координаты и кинематическую пару. Далее вызывает функцию обновления координат в интерфейсе.

4. Тестирование работы программы

4.1. Проверка работы программы

Для тестирования напишем основной файл, где будет вызываться интерфейс:

```
1 import sys
2 from PySide6.QtWidgets import QApplication
3 from config_manager import Config
4
5 import main_window as mw
6
7 if __name__ == "__main__":
8     Config.load()
9
10    app = QApplication(sys.argv)
11
12    window = mw.MainWindow()
13    window.show()
14
15    sys.exit(app.exec_())
```

Листинг 8: Точка входа программы.

Сначала загружается конфиг IP и порта, после создается окно программы, к кнопкам которого привязана логика взаимодействия компьютера с роботом по TCP.

Все опыты проводились в присутствии преподавателя. На робот загружена программа, представленная на листинге 14. Подключение робота к ноутбуку осуществлялось через Ethernet кабель.

Окно клиентской программы до подключения к роботу имеет вид, представленный на рис. 1. После успешного подключения клиентской программы к роботу, окно программы принимает вид, представленный на рис. 2, 3.

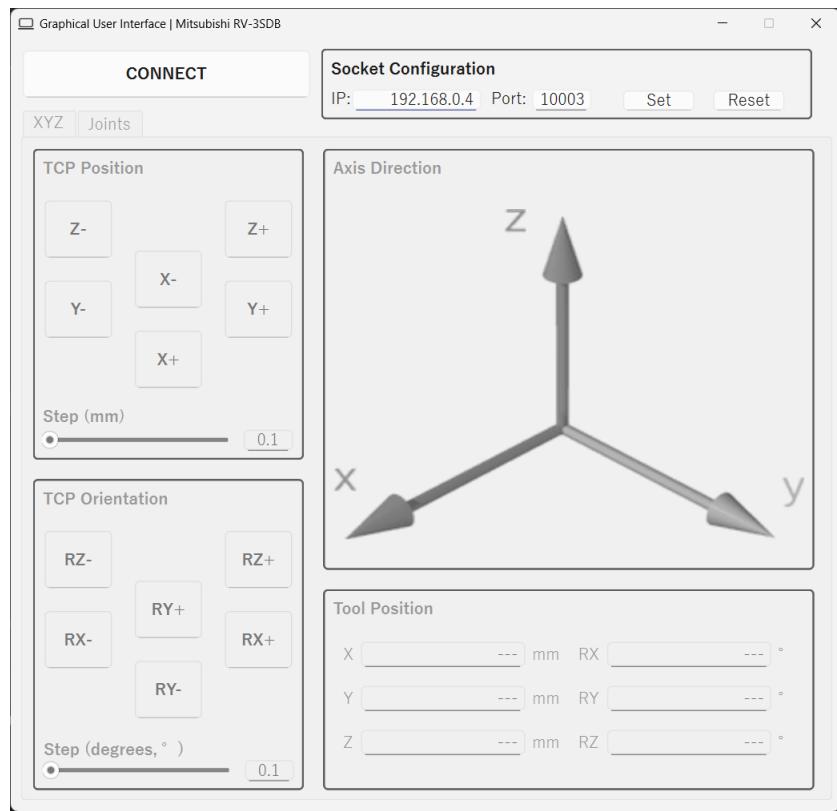


Рис. 1: Окно клиентской программы до подключения к роботу.

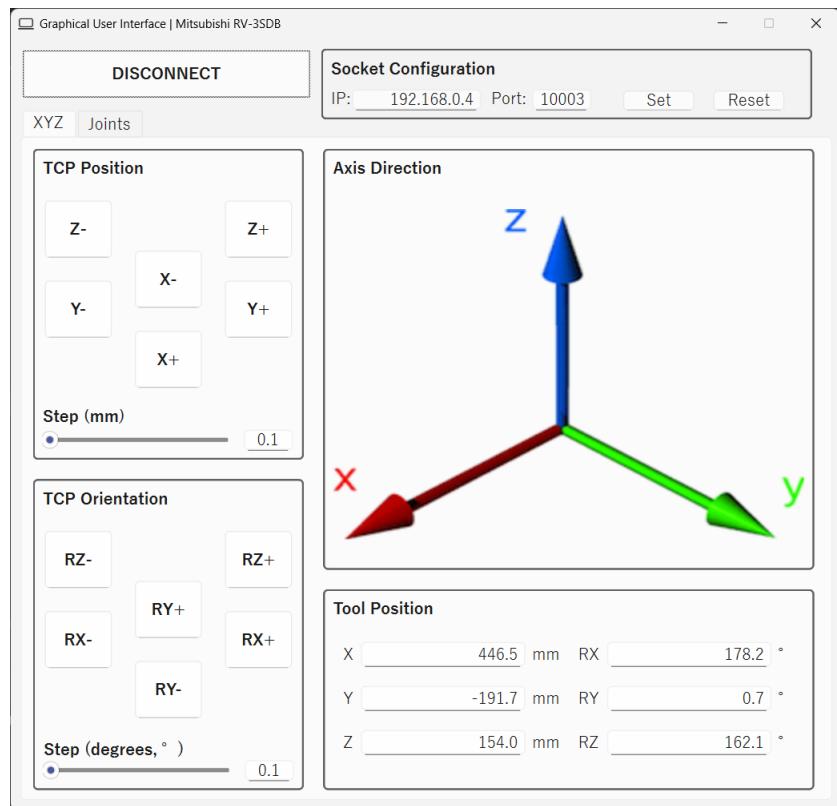


Рис. 2: Окно клиентской программы после подключения к роботу: декартовы координаты.

Для сочлененных координат необходимо поменять вкладку 'XYZ' на

'Joints'. Окно программы с переключением на сочлененные координаты имеет вид:

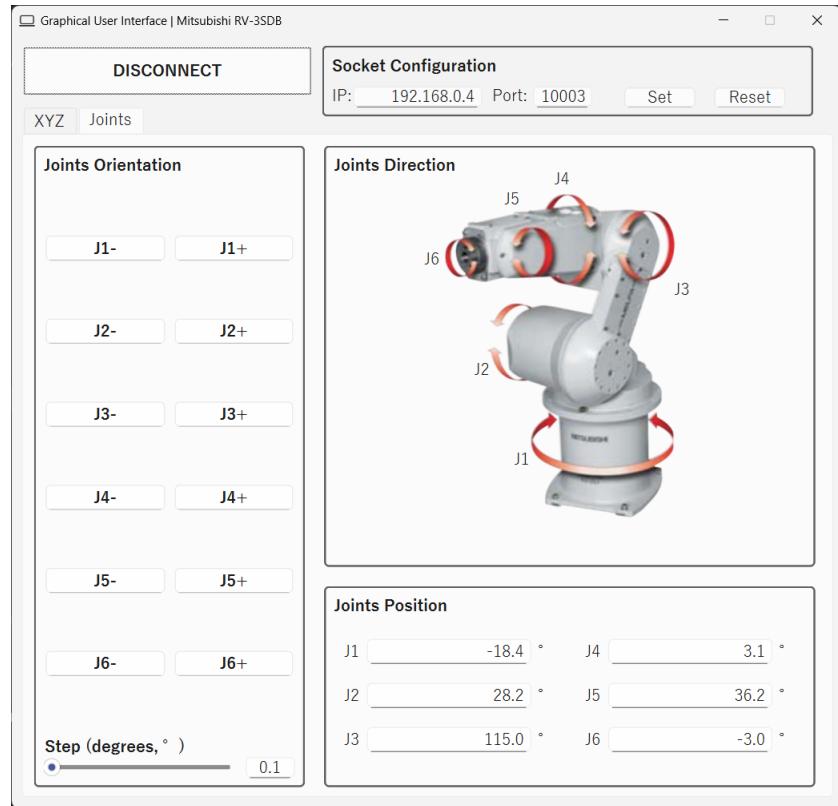


Рис. 3: Окно клиентской программы после подключения к роботу: сочлененные координаты.

Проверим соответствие координат в программе координатам, отображаемым на пульте управления.



Рис. 4: Проверка соответствия координат: декартовы координаты на пульте.

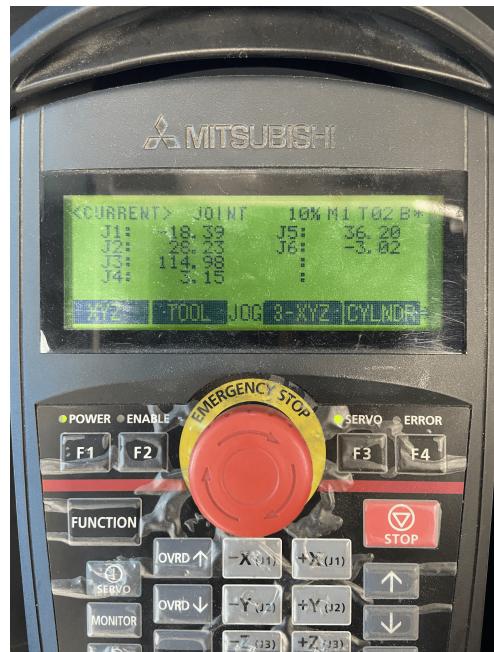


Рис. 5: Проверка соответствия координат: сочлененные координаты на пульте.

Сравнивая рис. 4 с 2 и 5 с 3 видим, что координаты совпадают с учетом округления.

Зафиксируем начальное положение робота. Несколько раз нажмем кнопку 'Y+' в окне программы, задав ползунком шаг 5 мм. Зафиксируем конечное положение робота.



Рис. 6: Начальное положение робота.



Рис. 7: Положение робота после нажатия кнопки 'Y+' несколько раз с шагом 5 мм.

Как видим, робот переместился вдоль координаты 'Y'. При автоматическом управлении роботом ручной режим недоступен.

4.2. Проблемы в ходе тестирования

В ходе тестирования работоспособности программы случались некоторые ошибки, в основном связанные с проверкой условий, обработкой ответа от робота или несовпадением ожидаемых и получаемых типов переменных внутри клиентской логики. Все проблемы были решены на месте, программа стала работать без ошибок.

5. Заключение

В ходе выполнения практической работы были разработаны алгоритмы для взаимодействия компьютера с роботом-манипулятором, их работоспособность была успешно проверена на роботе. Для выполнения работы был изучен робот-манипулятор, язык программирования MELFA BASIC и сетевое взаимодействие с помощью языка программирования Python. Был выбран формат сообщения, простестирована связь клиента и робота, доработана основная логика, подключен графический интерфейс и отлажены ошибки, возникшие в процессе тестирования.

6. Список использованных источников

1. Mitsubishi Electric MELFA RV-3SDB Series Manuals ([ссылка](#))
2. socket – Low-level networking interface ([ссылка](#))
3. re — Regular expression operations ([ссылка](#))
4. typing — Support for type hints ([ссылка](#))
5. enum — Support for enumerations ([ссылка](#))
6. Python YAML: How to Load, Read, and Write YAML ([ссылка](#))

A. Приложение

```
1 import socket
2 import connection_status as cs
3
4 class Client:
5     def __init__(self):
6         self.sock = socket.socket(socket.AF_INET, socket.
7             SOCK_STREAM)
8         self.connected = False
9
10    def connect(self, ip: str, port: int) -> cs.
11        ConnectionStatus:
12            try:
13                self.sock.connect((ip, port))
14                self.connected = True
15
16                return cs.ConnectionStatus.SUCCESS
17            except socket.error:
18                self.connected = False
19
20                return cs.ConnectionStatus.CONNECTION_ERROR
21
22    def disconnect(self) -> cs.ConnectionStatus:
23        if self.connected:
24            try:
25                self.sock.close()
26                self.connected = False
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
780
781
782
783
784
785
786
787
788
788
789
790
791
792
793
794
795
796
797
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
918
919
920
921
922
923
924
925
926
927
928
928
929
930
931
932
933
934
935
936
937
938
938
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
958
959
960
961
962
963
964
965
966
967
968
968
969
970
971
972
973
974
975
976
977
977
978
979
980
981
982
983
984
985
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1188
1189
1190
1191
1192
1193
1194
1194
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1290
1291
1292
1293
1293
1294
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1389
1390
1391
1392
1393
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1488
1489
1490
1491
1492
1493
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1586
1587
1588
1589
1590
1591
1592
1593
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1688
1689
1690
1691
1692
1693
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1788
1789
1790
1791
1792
1793
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1886
1887
1888
1889
1890
1891
1892
1893
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
21
```

```

25
26         return cs.ConnectionStatus.SUCCESS
27     except socket.error:
28         return cs.ConnectionStatus.CONNECTION_ERROR
29
30     return cs.ConnectionStatus.NOT_CONNECTED
31
32 def send(self, data: str) -> cs.ConnectionStatus:
33     if not self.connected:
34         return cs.ConnectionStatus.NOT_CONNECTED
35     try:
36         self.sock.sendall(data.encode('utf-8'))
37
38     return cs.ConnectionStatus.SUCCESS
39     except socket.error:
40         return cs.ConnectionStatus.SEND_ERROR
41
42 def receive(self, out_data: list, buffer_size: int = 1024)
43     -> cs.ConnectionStatus:
44     if not self.connected:
45         return cs.ConnectionStatus.NOT_CONNECTED
46     try:
47         received = self.sock.recv(buffer_size)
48         out_data.clear()
49         out_data.append(received.decode('utf-8'))
50
51     return cs.ConnectionStatus.SUCCESS
52     except socket.error:
53         return cs.ConnectionStatus.RECEIVE_ERROR

```

Листинг 9: Основа для общения компьютера с роботом.

Б. Приложение

```

1 from typing import List
2 import re
3
4 class CommandMessageManager:
5     @staticmethod
6     def build_position_request(pos: List[float]) -> str:
7         if len(pos) != 6:

```

```

8         raise ValueError('List of pos must contain 6
9             values')
10
11
12     @staticmethod
13     def parse_position_response(response: list):
14         try:
15             matches = re.findall(r'\((.*?)\)', response[0])
16             lists = [list(map(float, group.split(','))) for
17                     group in matches[:2]]
18             lists.append(list(map(int, matches[2].split(','))))
19
20         return tuple(lists)
21     except ValueError:
22         return None

```

Листинг 10: Скрипт с функциями для обработки координат на отправку или получение.

B. Приложение

```

1 import yaml
2
3 class Config:
4     ip: str = ""
5     port: int = 0
6
7     @classmethod
8     def load(cls, path: str = "config.yaml"):
9         with open(path, "r") as file:
10             config = yaml.safe_load(file)
11             cls.ip = config["connection"]["ip"]
12             cls.port = config["connection"]["port"]
13
14     @classmethod
15     def save(cls, path: str = "config.yaml"):
16         with open(path, "w") as file:
17             yaml.safe_dump({
18                 "connection": {

```

```

19             "ip": cls.ip,
20             "port": cls.port
21         }
22     }, file)
23
24     @classmethod
25     def set_config(cls, ip: str, port: int):
26         cls.ip = ip
27         cls.port = port

```

Листинг 11: Скрипт для работы с чтением и сохранением конфигурации IP и порта робота.

Г. Приложение

```

1 from typing import List, Literal, Tuple
2
3 class VirtualRobot:
4     def __init__(self,
5                  cartesian: List[float] = None,
6                  joint: List[float] = None,
7                  kin_sol: Tuple[int, int] = None):
8         self.cartesian = cartesian if cartesian is not None
9         else [0.0] * 6
10        self.joint = joint if joint is not None else [0.0] * 6
11        self.kinematic_sol = kin_sol if kin_sol is not None
12        else (0, 0)
13
14    def update_cartesian(self, new_values: List[float],
15                         kinematic_sol: Tuple[int, int]):
16        if len(new_values) != 6:
17            raise ValueError("Cartesian must have 6 values")
18        self.cartesian = new_values.copy()
19        self.kinematic_sol = kinematic_sol
20
21    def update_joint(self, new_values: List[float]):
22        if len(new_values) != 6:
23            raise ValueError("Joint must have 6 values")
24        self.joint = new_values.copy()
25
26    def calculate_next_move(

```

```

24     self,
25     mode: Literal["cartesian_linear", "cartesian_rotation"
26                 , "joint_rotation"],
27     axis_index: int,
28     step: float
29 ) -> List[float]:
30     if not (0 <= axis_index < 6):
31         raise ValueError("Value of axis_index must be in
32                           range from 0 to 5")
33
34     if "cartesian" in mode:
35         new_pos = self.cartesian.copy()
36     elif "joint" in mode:
37         new_pos = self.joint.copy()
38     else:
39         raise ValueError("Expected 'cartesian' or 'joint'
40                           mode")
41
42     new_pos[axis_index] += step
43
44     return new_pos

```

Листинг 12: Скрипт виртуального робота: сохранение координат робота и кинематической пары. Вычисление следующих координат по индексу и шагу.

Д. Приложение

```

1 from typing import Tuple, Literal
2
3 from virtual_robot import VirtualRobot
4 from command_manager import CommandMessageManager
5 from robot_command import RobotCommand, MODE_TO_COMMAND
6 from client import Client
7 from connection_status import ConnectionStatus
8
9 class BackendController:
10     def __init__(self):
11         self.robot = VirtualRobot()
12         self.client = Client()
13         self.last_connection_status = ConnectionStatus.NONE
14         self.last_cmd_send_status = ConnectionStatus.NONE
15         self.last_params_send_status = ConnectionStatus.NONE

```

```

16         self.last_receive_status = ConnectionStatus.NONE
17
18     def connect(self, ip: str, port: int):
19         self.last_connection_status = self.client.connect(ip,
20                 port)
21         return self.client.connected
22
23     def disconnect(self):
24         cmd = str(RobotCommand.EXIT.value)
25         self.last_cmd_send_status = self.client.send(cmd)
26
27         self.last_connection_status = self.client.disconnect()
28
29         return not self.client.connected
30
31     def is_connected(self):
32         return self.client.connected
33
34     def get_pos(self):
35         cmd = str(RobotCommand.GET_POSITION.value)
36         self.last_cmd_send_status = self.client.send(cmd)
37
38         ans = []
39         self.last_receive_status = self.client.receive(ans)
40
41         return ans
42
43     def move_axis(self,
44                 move_info: Tuple[Literal["cartesian_linear",
45                                     "cartesian_rotation", "joint_rotation"],
46                                     int, Literal['+', '-']],
47                 step: float):
48         mode, axis_index, direction = move_info
49
50         signed_step = step if direction == '+' else -step
51         pos = self.robot.calculate_next_move(mode, axis_index,
52                                             signed_step)
53
54         cmd = str(MODE_TO_COMMAND[mode].value)

```

```

52         params = CommandMessageManager.build_position_request(
53             pos)
54         if "cartesian" in mode:
55             params = f"{{params}}({', '.join(map(str, self.robot.
56                 kinematic_sol))})"
57
58         self.last_cmd_send_status = self.client.send(cmd)
59         self.last_params_send_status = self.client.send(params
60             )
61
62         ans = []
63         self.last_receive_status = self.client.receive(ans)
64
65     return ans

```

Листинг 13: Скрипт, объединяющий различные функции для полноценного взаимодействия с роботом с компьютера.

E. Приложение

```

1 JOVRD 100
2 SPD 100
3 DEF INTE DCOMM
4 DCOMM = 1
5 PHELP = P_CURR
6 JHELP = J_CURR
7 SERVO ON
8 OPEN "COM3:" AS #1
9
10 WHILE DCOMM > 0
11     INPUT #1, DCOMM
12     IF DCOMM = 1 THEN
13         PRINT #1, J_CURR, P_CURR
14     ENDIF
15     IF DCOMM = 2 THEN
16         INPUT #1, PHELP
17         MOV PHELP
18         PRINT #1, J_CURR, P_CURR
19     ENDIF
20     IF DCOMM = 3 THEN
21         INPUT #1, JHELP

```

```

22      MOV JHELP
23      PRINT #1, J_CURR, P_CURR
24  ENDIF
25 WEND
26
27 CLOSE #1
28 SERVO OFF
29 END

```

Листинг 14: Программа на роботе-манипуляторе для выполнения команд, присылаемых клиентом.

Ж. Приложение

```

1 self.axis_button_map = {
2     'btn_Xp': ("cartesian_linear", CartesianAxis.X.value, '+'),
3     'btn_Xm': ("cartesian_linear", CartesianAxis.X.value, '-'),
4     'btn_Yp': ("cartesian_linear", CartesianAxis.Y.value, '+'),
5     'btn_Ym': ("cartesian_linear", CartesianAxis.Y.value, '-'),
6     'btn_Zp': ("cartesian_linear", CartesianAxis.Z.value, '+'),
7     'btn_Zm': ("cartesian_linear", CartesianAxis.Z.value, '-'),
8     'btn_RXp': ("cartesian_rotation", CartesianAxis.A.value, '+'),
9     'btn_RXm': ("cartesian_rotation", CartesianAxis.A.value, '-'),
10    'btn_RYp': ("cartesian_rotation", CartesianAxis.B.value, '+'),
11    'btn_RYm': ("cartesian_rotation", CartesianAxis.B.value, '-'),
12    'btn_RZp': ("cartesian_rotation", CartesianAxis.C.value, '+'),
13    'btn_RZm': ("cartesian_rotation", CartesianAxis.C.value, '-'),
14    'btn_J1p': ("joint_rotation", JointAxis.J1.value, '+'),
15    'btn_J1m': ("joint_rotation", JointAxis.J1.value, '-'),
16    'btn_J2p': ("joint_rotation", JointAxis.J2.value, '+'),
17    'btn_J2m': ("joint_rotation", JointAxis.J2.value, '-'),
18    'btn_J3p': ("joint_rotation", JointAxis.J3.value, '+'),
19    'btn_J3m': ("joint_rotation", JointAxis.J3.value, '-'),
20    'btn_J4p': ("joint_rotation", JointAxis.J4.value, '+'),
21    'btn_J4m': ("joint_rotation", JointAxis.J4.value, '-'),
22    'btn_J5p': ("joint_rotation", JointAxis.J5.value, '+'),
23    'btn_J5m': ("joint_rotation", JointAxis.J5.value, '-'),
24    'btn_J6p': ("joint_rotation", JointAxis.J6.value, '+'),
25    'btn_J6m': ("joint_rotation", JointAxis.J6.value, '-')
26 }

```

Листинг 15: Карта кнопок окна программы, которым соответствуют кортежи move_info, описывающие соответствующую кнопку.

3. Приложение

```
1 def handle_connection(self):
2     if self.backend.is_connected():
3         self.backend.disconnect()
4
5         if not self.backend.is_connected():
6             self.ui.tabWidget.setEnabled(False)
7             self.ui.btn_connect.setText("CONNECT")
8     else:
9         try:
10             ip = Config.ip
11             port = Config.port
12             self.backend.connect(ip, port)
13
14             if self.backend.is_connected():
15                 response = self.backend.get_pos()
16                 joints, cartesian, kinematic_sol =
17                     CommandMessageManager.
18                     parse_position_response(response)
19
20                     self.backend.robot.update_cartesian(cartesian,
21                         kinematic_sol)
22                     self.backend.robot.update_joint(joints)
23
24                     self.update_ui_axis()
25
26                     self.ui.tabWidget.setEnabled(True)
27                     self.ui.btn_connect.setText("DISCONNECT")
28     except Exception as e:
29         print(f"Caught exception: {e}")
```

Листинг 16: Реализация обработки нажатия на кнопку connect/disconnect в интерфейсе.

И. Приложение

```
1 def handle_axis_button(self):
2     btn = self.sender()
3     if not btn:
4         return
5
6     btn_name = btn.objectName()
7     if btn_name not in self.axis_button_map:
8         print(f"Unexpected button: {btn_name}")
9         return
10
11    move_info = self.axis_button_map[btn_name]
12    mode, _, _ = move_info
13
14    if "cartesian" in mode:
15        if not "rotation" in mode:
16            step = self.linear_step_val
17        else:
18            step = self.degree_1_step_val
19    else:
20        step = self.degree_2_step_val
21
22    if not self.backend.is_connected():
23        print("Unexpected event: no connection")
24        return
25
26    try:
27        response = self.backend.move_axis(move_info, step)
28        joints, cartesian, kinematic_sol =
29            CommandMessageManager.parse_position_response(
30            response)
31
32        self.backend.robot.update_cartesian(cartesian,
33                                            kinematic_sol)
34        self.backend.robot.update_joint(joints)
35
36        self.update_ui_axis()
37    except Exception as e:
38        print(f"Caught exception: {e}")
```

Листинг 17: Реализация функции обработки нажатия кнопок изменения координат.