

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

|  |
| --- |
| **ИНСТИТУТ МИРОВОГО ОКЕАНА (ШКОЛА)** |

**Департамент автоматики и робототехники**

Тумас Мария Владиславовна

**КУРСОВАЯ РАБОТА**

Разработка информационно-управляющей системы для динамического объекта

Вариант 10

|  |
| --- |
| Студент группы  Б11120-15.03.06мхрб  Тумас М.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель  д.т.н., доцент  Юхимец Д.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Регистрационный № \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О.Фамилия  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 г. | Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись И.О.Фамилия  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г. |
|  |  |

г. Владивосток

2023

СОДЕРЖАНИЕ

[Цель работы 6](#_Toc75151747)

[Задание на курсовую работу 7](#_Toc75151748)

[Техническое задание 8](#_Toc75151749)

[Содержание 10](#_Toc75151750)

[1 Введение 12](#_Toc75151751)

[1.1 Наименование 12](#_Toc75151752)

[1.2 Область применения 12](#_Toc75151753)

[2 Основания для разработки 12](#_Toc75151754)

[3 Назначение разработки 13](#_Toc75151755)

[3.1 Функциональное назначение 13](#_Toc75151756)

[3.2 Эксплуатационное назначение 13](#_Toc75151757)

[4 Требования к программе 13](#_Toc75151758)

[4.1 Требования к функциональным характеристикам 13](#_Toc75151759)

[4.1.1 Требование к составу выполняемых функций 13](#_Toc75151760)

[4.1.2 Требования к организации входных и выходных данных 14](#_Toc75151761)

[4.1.3 Требования к временным характеристикам 15](#_Toc75151762)

[4.2 Требования к надежности 15](#_Toc75151763)

[4.2.1 Требования к обеспечению надежного (устойчивого) функционирования программы 15](#_Toc75151764)

[4.2.2 Время восстановления после отказа 15](#_Toc75151765)

[4.2.3 Отказы из-за некорректных действий оператора 16](#_Toc75151766)

[4.3 Условия эксплуатации 16](#_Toc75151767)

[4.3.1 Климатические условия эксплуатации 16](#_Toc75151768)

[4.3.2 Требование к видам обслуживания 16](#_Toc75151769)

[4.3.3 Требование к численности и квалификации персонала 16](#_Toc75151770)

[4.4 Требования к составу и параметрам технических средств 17](#_Toc75151771)

[4.5 Требования к информационной и программной совместимости 17](#_Toc75151772)

[4.5.1 Требования к информационным структурам и методам решения 17](#_Toc75151773)

[4.5.2 Требования к исходным кодам и языкам программирования 18](#_Toc75151774)

[4.5.3 Требование к защите информации и программ 18](#_Toc75151775)

[4.6 Требования к маркировке и упаковке 18](#_Toc75151776)

[4.7 Требования к транспортированию и хранению 18](#_Toc75151777)

[4.8 Специальные требования 18](#_Toc75151778)

[5 Требования к программной документации 18](#_Toc75151779)

[6 Технико-экономические показатели 19](#_Toc75151780)

[7 Стадии и этапы разработки 19](#_Toc75151781)

[7.1 Стадии разработки 19](#_Toc75151782)

[7.2 Этапы разработки 19](#_Toc75151783)

[7.3 Содержание работ по этапам 19](#_Toc75151784)

[8 Порядок контроля и приемки 21](#_Toc75151785)

[8.1 Виды испытаний 21](#_Toc75151786)

[8.2 Общие требования к приемке работы 21](#_Toc75151787)

[Описание программы 22](#_Toc75151788)

[Аннотация 24](#_Toc75151789)

[1. Общие сведения 26](#_Toc75151790)

[1.1 Обозначение и наименование программы 26](#_Toc75151791)

[1.2 Программное обеспечение, необходимое для функционирования программы 26](#_Toc75151792)

[1.3 Языки программирования, на которых написана программа 26](#_Toc75151793)

[2. Функциональное назначение 26](#_Toc75151794)

[2.1 Краткое описание функций 26](#_Toc75151795)

[2.2 Состав функций 27](#_Toc75151796)

[2.3 Сведения о функциональных ограничениях на применение 27](#_Toc75151797)

[3. Описание логичекой структуры 28](#_Toc75151798)

[3.1 Структура программы 28](#_Toc75151799)

[3.2 Алгоритм программы 38](#_Toc75151800)

[4. Используемые технические средства 48](#_Toc75151801)

[5. Вызов и загрузка 49](#_Toc75151802)

[6. Входные данные 50](#_Toc75151803)

[7. Выходные данные 51](#_Toc75151804)

[Лист регистрации изменений 52](#_Toc75151805)

[Руководство системного программиста 53](#_Toc75151806)

[Аннотация 55](#_Toc75151807)

[1. Общие сведения о программе 57](#_Toc75151808)

[1.1 Назначение программы 57](#_Toc75151809)

[1.2 Функции программы 57](#_Toc75151810)

[1.3 Сведения о технических и программных средствах 57](#_Toc75151811)

[2. Структура программы 58](#_Toc75151812)

[2.1 Сведения о структуре программы 58](#_Toc75151813)

[2.2 Взаимодействие с другими программами 58](#_Toc75151814)

[3. Настройка на состав программных средств 58](#_Toc75151815)

[4. Проверка программы 58](#_Toc75151816)

[Лист регистрации изменений 60](#_Toc75151817)

[Руководство оператора 61](#_Toc75151818)

[1. Общие сведения о программе 65](#_Toc75151819)

[1.1 Назначение программы 65](#_Toc75151820)

[1.2 Функции программы 65](#_Toc75151821)

[2. Условия выполнения программы 65](#_Toc75151822)

[2.1 Требования к аппаратным средствам 65](#_Toc75151823)

[2.2 Требования к программным средствам 66](#_Toc75151824)

[2.3 Требования к оператору 66](#_Toc75151825)

[3. Выполнение программы 66](#_Toc75151826)

[3.1 Запуск программы 66](#_Toc75151827)

[3.2 Начало работы 67](#_Toc75151828)

[3.3 Настройка начальных параметров 69](#_Toc75151829)

[3.4 Запуск работы системы 72](#_Toc75151830)

[Лист регистрации изменений 74](#_Toc75151831)

[Текст программы 75](#_Toc75151832)

[1. Листинг модуля gui/server 78](#_Toc75151833)

[1.1. Файл ui\_loading.py 78](#_Toc75151834)

[1.2. Файл ui\_functions.py 80](#_Toc75151835)

[1.3. Файл ui\_main.py 80](#_Toc75151836)

[1.4. Файл main.py 100](#_Toc75151837)

[2. Листинг модуля control system 110](#_Toc75151838)

[3. Листинг модуля navigation system 117](#_Toc75151839)

[4. Листинг модуля camera 123](#_Toc75151840)

[5. Листинг модуля trajectory system 127](#_Toc75151841)

[Лист регистрации изменений 131](#_Toc75151842)

[Результаты работы 132](#_Toc75151843)

[Испытание 1 132](#_Toc75151844)

[Испытание 2 133](#_Toc75151845)

[Испытание 3 134](#_Toc75151846)

[Испытание 4 135](#_Toc75151847)

[Испытание 5 137](#_Toc75151848)

[Испытание 6 138](#_Toc75151849)

[Вывод 141](#_Toc75151850)

# Цель работы

Целью курсовой работы является разработка информационно-управляющей системы (ИУС) для динамического объекта, разработка архитектуры и программной реализации алгоритмов управления.

Разработанная ИУС должна обеспечивать выполнение типового задания, с помощью беспилотного летательного аппарата, при этом выполнение указанной задачи предполагает движение робота по некоторому маршруту, заданному на тестовом полигоне, содержащему заранее неизвестные стационарные препятствия.

Типовое задание робота описывает траекторию его движения, проходящую через последовательность базовых точек по траекториям заданного типа: движение к точке, прямолинейное движение к точке и движение по криволинейной траектории, задаваемой сплайном третьего порядка. Задание перед началом движения робота должен задавать пользователь. В процессе выполнения миссии робот должен двигаться с заданной скоростью при движении по участкам траекторий между базовыми точками. Также беспилотный летательный аппарат должен двигаться на заданной высоте от поверхности. При движении робот должен обходить препятствия, расположение которых заранее неизвестно. Движение в штатном режиме происходит «носом вперед». ИУС должна содержать ПИД-регуляторы, обеспечивающие им выход в заданную точку пространства.

В процессе выполнения миссии поиска робот должен обеспечить выполнение функции поиска объекта заданного цвета, расположенного в заранее неизвестном месте карты. Сенсор для поиска – бортовая видеокамера. Результат поиска – координаты объекта.

Разработанная в процессе выполнения курсовой работы информационно-управляющая система робота должна иметь интерфейс пользователя, позволяющий задать задание для робота и отобразить процесс ее выполнения.

# Задание на курсовую работу

В таблице 1 приведены исходные данные для выполнения курсовой работы в соответствии с вариантом.

Таблица 1 – Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Объект управления | Архитектура ИУС | Функция |
| 10 | Беспилотный летательный аппарат | Объектно-ориентированная | Поиск |

Согласно таблице 1 в данной курсовой работе будет рассмотрена разработка системы управления моделью беспилотного летательного аппарата (БПЛА) с использованием архитектуры объектно-ориентированного программирования (ООП). ООП представляет собой подход к программированию, который позволяет создавать программное обеспечение, основанное на объектах, которые взаимодействуют друг с другом. Основной миссией робота является поиск объектов заданного цвета и вывод их координат, как результат поиска.

Рисунок 1 – Вид собранного робота в среде CoppeliaSim

|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Д.т.н., доцент департамента автоматики и робототехники |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_ Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2023 |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

Информационно-управляющая система «FlightOps»

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

**Техническое задание**

**ЛИСТ УТВЕРЖДЕНИЯ**

**Б11120-15.03.06 ТЗ 10.01 ЛУ**

|  |  |  |
| --- | --- | --- |
|  |  | Представители предприятия-разработчика  Руководитель разработки |
|  |  | Д.т.н., доцент |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |
|  |  |  |
|  |  | Исполнитель |
|  |  | Студент группы  Б11120-15.03.06мхрб |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_Тумас М.В. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |

**2023**

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  Б11120-15.03.06 ТЗ 10.01 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Информационно-управляющая система «FlightOps»

**Техническое задание**

**Б11120-15.03.06 ТЗ 10.01**

**Листов 12**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**2023**

СОДЕРЖАНИЕ

[1 Введение 12](#_Toc75151751)

[1.1 Наименование 12](#_Toc75151752)

[1.2 Область применения 12](#_Toc75151753)

[2 Основания для разработки 12](#_Toc75151754)

[3 Назначение разработки 13](#_Toc75151755)

[3.1 Функциональное назначение 13](#_Toc75151756)

[3.2 Эксплуатационное назначение 13](#_Toc75151757)

[4 Требования к программе 13](#_Toc75151758)

[4.1 Требования к функциональным характеристикам 13](#_Toc75151759)

[4.1.1 Требование к составу выполняемых функций 13](#_Toc75151760)

[4.1.2 Требования к организации входных и выходных данных 14](#_Toc75151761)

[4.1.3 Требования к временным характеристикам 15](#_Toc75151762)

[4.2 Требования к надежности 15](#_Toc75151763)

[4.2.1 Требования к обеспечению надежного (устойчивого) функционирования программы 15](#_Toc75151764)

[4.2.2 Время восстановления после отказа 15](#_Toc75151765)

[4.2.3 Отказы из-за некорректных действий оператора 16](#_Toc75151766)

[4.3 Условия эксплуатации 16](#_Toc75151767)

[4.3.1 Климатические условия эксплуатации 16](#_Toc75151768)

[4.3.2 Требование к видам обслуживания 16](#_Toc75151769)

[4.3.3 Требование к численности и квалификации персонала 16](#_Toc75151770)

[4.4 Требования к составу и параметрам технических средств 17](#_Toc75151771)

[4.5 Требования к информационной и программной совместимости 17](#_Toc75151772)

[4.5.1 Требования к информационным структурам и методам решения 17](#_Toc75151773)

[4.5.2 Требования к исходным кодам и языкам программирования 18](#_Toc75151774)

[4.5.3 Требование к защите информации и программ 18](#_Toc75151775)

[4.6 Требования к маркировке и упаковке 18](#_Toc75151776)

[4.7 Требования к транспортированию и хранению 18](#_Toc75151777)

[4.8 Специальные требования 18](#_Toc75151778)

[5 Требования к программной документации 18](#_Toc75151779)

[6 Технико-экономические показатели 19](#_Toc75151780)

[7 Стадии и этапы разработки 19](#_Toc75151781)

[7.1 Стадии разработки 19](#_Toc75151782)

[7.2 Этапы разработки 19](#_Toc75151783)

[7.3 Содержание работ по этапам 19](#_Toc75151784)

[8 Порядок контроля и приемки 21](#_Toc75151785)

[8.1 Виды испытаний 21](#_Toc75151786)

[8.2 Общие требования к приемке работы 21](#_Toc75151787)

# ВВЕДЕНИЕ

## Наименование

Информационно-управляющая система (ИУС) «FlightOps».

## Область применения

ИУС предназначена для использования в симуляции миссии поиска объектов в рабочей зоне моделью беспилотного летательного аппарата. Симуляция проводится в программе CoppeliaSim. Сенсор для поиска – бортовая видеокамера.

# ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ

## 2.1 Документ, на основании которого ведутся разработки

Документы, на основании которых ведется разработка: Д.А. Юхимец – учебное электронное издание «Разработка информационно-управляющих систем для робототехнических объектов», «Методические указания к работе со средой моделирования V-REP», ГОСТ 19.201-78, ГОСТ 19.106-78, ГОСТ 19.104-78.

## 2.2 Организация, утвердившая документ

Основанием для разработки является задание на курсовую работу по дисциплине «Автоматизированные информационно-управляющие системы» от 6.03.2022 года;

## 2.3 Наименование разработки

Наименование темы разработки – «Разработка информационно-управляющей системы для динамического объекта». Условное обозначение разработки – «FlightOps».

# НАЗНАЧЕНИЕ РАЗРАБОТКИ

## Функциональное назначение

Основные функции ИУС:

1. реализация движения БПЛА на заданной высоте;
2. реализация движения БПЛА в точку с заданной скоростью;
3. реализация движения БПЛА прямолинейно в точку с заданной скоростью;
4. реализация движения БПЛА криволинейно с заданной скоростью;
5. выполнение функции поиска объекта определенного цвета с выводом координат объекта;
6. отображение характеристик движения БПЛА;
7. объезд препятствий, расположение на карте которых заранее не известно;
8. консольный интерфейс, позволяющий пользователю задавать параметры типового задания.

## Эксплуатационное назначение

ИУС предназначена для эксплуатации на стационарном компьютере или ноутбуке.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## Требования к функциональным характеристикам

### **Требование к составу выполняемых функций**

Информационно-управляющая система должна обеспечивать возможность выполнения перечисленных ниже функций:

* + 1. движение робота в заданную точку;
    2. движение робота прямолинейно в заданную точку;
    3. движение робота по криволинейной траектории, описанной сплайном 3-го порядка;
    4. движение робота с заданной скоростью по траекториям всех типов, описанных выше (от 0.5 м/с до 3 м/с);
    5. движение робота на заданной высоте от поверхности (от 4 до 7 м);
    6. поиск объектов определённого цвета (красный);
    7. вывод координат найденных объектов в терминал программы во время работы и консольное окно по завершении миссии;
    8. обнаружение и обход роботом препятствий;
    9. задание пользователем миссии робота через интерфейс: обозначение ключевых точек, скорости, высоты полета и режима движения;
    10. вывод данных о движении (скорость, положение, высота полета, ошибка по курсу) по завершении миссии;
    11. автоматический расчет траектории обхода рабочей области для поиска объектов;
    12. старт миссии по команде;
    13. возможность остановки выполнения миссии;
    14. вывод информационного окна о завершении задания после его выполнения.

### **Требования к организации входных данных**

Входными данными являются параметры движения, имеющие тип float, такие как скорость, высота и ключевые точки траектории. А также значения типа bool, определяющие режим миссии. Все входные данные вводятся в программу через консоль с использованием виджета Entry для значений float и флажков для типа данных bool.

### **Требования к организации выходных данных**

Программа должна транслировать графический интерфейс для управления роботом, а также по окончании задания отображать в информационных окнах информацию о завершении миссии, графическую информацию характеристик движения и координаты найденных объектов в случае выполнения соответствующей задачи.

### **Требования к временным характеристикам**

Требования к временным характеристикам программы не предъявляются.

## 4.2 Требования к надежности

### **4.2.1 Требования к обеспечению надежного (устойчивого) функционирования программы**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

* 1. организацией бесперебойного питания технических средств;
  2. использованием лицензионного программного обеспечения;
  3. регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
  4. регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов;
  5. соблюдение условий эксплуатации, указанным в настоящем техническом задании.

### **4.2.2 Время восстановления после отказа**

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 15 минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

### **4.2.3 Отказы из-за некорректных действий оператора**

После запуска программы отказ программы вследствие некорректных действий оператора требует перезапуска программы.

## 4.3 Условия эксплуатации

### **4.3.1 Климатические условия эксплуатации**

Климатические условия эксплуатации должны соответствовать условиям эксплуатации ЭВМ, на которой данный продукт установлен. ЭВМ должен обладать высокой степенью защищенности от влаги; низких температур. Нормальная функциональность системы будет при следующих условиях:

* + 1. температура воздуха: –10…+30 ;
    2. относительная влажность воздуха: 11 – 81%;
    3. наличие места для зарядки ЭВМ.

### **4.3.2 Требование к видам обслуживания**

Программа не требует проведения каких-либо видов обслуживания

### **Требование к численности и квалификации персонала**

Для управления системой достаточно одного человека, способного запустить систему управления роботом. Оператор должен обладать практическими навыками использования персонального компьютера, а также перед работой с программой должен ознакомится с соответствующим руководством оператора.

В перечень задач, выполняемых оператором, входят:

1. Установка параметров работы программы;

2. Запуск программы;

3. Отключение программы.

## 4.4 Требования к составу и параметрам технических средств

В состав технических средств должен входить персональный компьютер, включающий в себя:

* + 1. Процессор Core i5 с максимальной тактовой частотой от 2.4 ГГц;
    2. Оперативную память объемом от 8 Гб;
    3. Твердотельный накопитель от 100 Гб;
    4. Свободное место на твердотельном накопителе - от 50 Гб;
    5. Видеокартой с объемом видео памяти от 4 Гб и тактовой частотой от 1 МГц;
    6. Операционную систему Windows 10;
    7. Устройства управления и ввода: клавиатура, мышь.
    8. Устройство вывода: монитор с разрешением от 1920х1080

## 4.5 Требования к информационной и программной совместимости

### **4.5.1 Требования к информационным структурам и методам решения**

Информационная структура вводимых значений в случае использования вещественных чисел обязывает использование точки в качестве разделяющего знака. В случае введения нескольких значений в виджет Entry необходимо разделение пробелом. К методам решения требования не предоставляются.

### **4.5.2 Требования к исходным кодам и языкам программирования**

Исходные коды программы должны быть реализованы на языке программирования Python 3.9. Требований к средам разработки не предоставляется.

### **4.5.3 Требование к защите информации и программ**

В качестве защиты программных данных применяется ГОСТ 51188-98. Защита информации.

## 4.6 Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

## 4.7 Требования к транспортированию и хранению

Программа поставляется в виде программного изделия – на внешнем носителе (ноутбук) или на внешнем дистрибутивном носителе (Flash-носитель). Требования к транспортировке не предъявляются.

## 4.8 Специальные требования

Специальные требования не предъявляются.

# ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Документы к программе должны быть выполнены в соответствии с ГОСТ 19.106-78 и ГОСТами к каждому виду документа;

Состав программной документации должен включать в себя:

техническое задание;

руководство системного программиста;

руководство оператора;

1. программу и методики испытаний.

# ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ



## Экономическая эффективность

Ориентировочная экономическая эффективность и предполагаемая годовая потребность не рассчитываются.

# 7 СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

## 7.1 Стадии разработки

Разработка должна быть проведена в три стадии:

разработка технического задания;

рабочее проектирование;

внедрение.

## 7.2 Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

разработка программы;

разработка программной документации;

испытания программы.

## 7.3 Содержание работ по этапам

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. постановка задачи;
2. определение и уточнение требований к техническим средствам;
3. определение требований к программе;
4. определение стадий, этапов и сроков разработки программы и документации на неё;
5. выбор языка программирования;
6. согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (написанию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101-77 с требованием п. Предварительный состав программной документации настоящего технического задания.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1. разработка, согласование и утверждение программы и методики испытаний;
2. проведение приемо-сдаточных испытаний;
3. корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах Заказчика.

# 8 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

## 8.1 Виды испытаний

Приемо-сдаточные испытания должны проводиться на объекте Заказчика в оговорённые сроки.

Приемо-сдаточные испытания программы должны проводиться согласно разработанной Исполнителем и согласованной Заказчиком программы и методик испытаний.

Ход проведения приемо-сдаточных испытаний Заказчик и Исполнитель документируют в протоколе проведения испытаний.

## 8.2 Общие требования к приемке работы

На основании Протокола проведения испытаний Исполнитель совместно с Заказчиком подписывают Акт приемки-сдачи программы в эксплуатацию

|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Д.т.н., доцент департамента автоматики и робототехники |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_ Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2023 |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

Информационно-управляющая система «FlightOps»

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

**Описание программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**Б11120-15.03.06 ОП 10.01 ЛУ**

|  |  |  |
| --- | --- | --- |
|  |  | Руководитель разработки |
|  |  | Д.т.н., доцент |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |
|  |  |  |
|  |  | Исполнитель |
|  |  | Студент группы  Б11120-15.03.06мхр, |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Тумас М.В. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |

**2023**

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  Б11120-15.03.06 ОП 10.01 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Информационно-управляющая система «FlightOps»

**Описание программы**

**Б11120-15.03.06 ОП 10.01**

**Листов 29**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**2023**

# АННОТАЦИЯ

В данном документе представлено писание программы «FlightOps», предназначенной для использования в модели БПЛА с симуляцией миссии поиска объектов в программе CopelliaSim. Документ содержит информацию о функциональном назначении программы, её логике и условиях применения. Вся разработка программного обеспечения осуществлялась на основе технического задания под номером Б11120-15.03.06 ТЗ 10.01 в соответствии с требованиями ЕСПД ГОСТ 19.402-78.

СОДЕРЖАНИЕ

[1. Общие сведения 26](#_Toc75151790)

[1.1 Обозначение и наименование программы 26](#_Toc75151791)

[1.2 Программное обеспечение, необходимое для функционирования программы 26](#_Toc75151792)

[1.3 Языки программирования, на которых написана программа 26](#_Toc75151793)

[2. Функциональное назначение 26](#_Toc75151794)

[2.1 Краткое описание функций 26](#_Toc75151795)

[2.2 Состав функций 27](#_Toc75151796)

[2.3 Сведения о функциональных ограничениях на применение 27](#_Toc75151797)

[3. Описание логичекой структуры 28](#_Toc75151798)

[3.1 Структура программы 28](#_Toc75151799)

[3.2 Алгоритм программы 38](#_Toc75151800)

[4. Используемые технические средства 48](#_Toc75151801)

[5. Вызов и загрузка 49](#_Toc75151802)

[6. Входные данные 50](#_Toc75151803)

[7. Выходные данные 51](#_Toc75151804)

[Лист регистрации изменений 52](#_Toc75151805)

# ОБЩИЕ СВЕДЕНИЯ

## 1.1 Обозначение и наименование программы

Наименование программы: «FlightOps». Программа предназначена для применения в модели БПЛА с симуляцией функцией поиска объектов определенного цвета.

## 1.2 Программное обеспечение, необходимое для функционирования программы

Данное программное обеспечение может использоваться на следующих операционных системах: Microsoft Windows 10. Дополнительное ПО не используется.

## 1.3 Языки программирования, на которых написана программа

Программа реализована на языке программирования Python 3.9 с использованием компилятора PyCharm Community.

# ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

## 2.1 Краткое описание функций

Основные функции ИУС:

реализация движения БПЛА на заданной высоте;

1. реализация движения БПЛА в точку с заданной скоростью;
2. реализация движения БПЛА прямолинейно в точку с заданной скоростью;
3. реализация движения БПЛА криволинейно с заданной скоростью;
4. выполнение функции поиска объекта определенного цвета с выводом координат объекта;
5. отображение характеристик движения БПЛА после завершения миссии;
6. объезд препятствий, расположение на карте которых заранее не известно;
7. консольный интерфейс, позволяющий пользователю задавать параметры типового задания.

## 2.2 Состав функций

Информационно-управляющая система обеспечивает возможность выполнения перечисленных ниже функций:

* + 1. движение робота в заданную точку;
    2. движение робота прямолинейно в заданную точку;
    3. движение робота по криволинейной траектории, описанной сплайном 3-го порядка;
    4. движение робота с заданной скоростью по траекториям всех типов, описанных выше (от 0.5 м/с до 3 м/с);
    5. движение робота на заданной высоте от поверхности (от 4 до 7 м);
    6. поиск объектов определённого цвета (красный);
    7. вывод координат найденных объектов в терминал программы во время работы и консольное окно по завершении миссии;
    8. обнаружение и обход роботом препятствий;
    9. задание пользователем миссии робота через интерфейс: обозначение ключевых точек, скорости, высоты полета и режима движения;
    10. вывод данных о движении (скорость, положение, высота полета, ошибка по курсу) по завершении миссии;
    11. автоматический расчет траектории обхода рабочей области для поиска объектов;
    12. старт миссии по команде;
    13. возможность остановки выполнения миссии;
    14. вывод информационного окна о завершении задания после его выполнения.

## 2.3 Сведения о функциональных ограничениях на применение

В качестве функциональных ограничений выступает возникновение критического состояния модуля системы, что обосновано последующей невозможностью выполнения заявленных функций.

В качестве ограничений, накладываемых на область применения выступает соблюдение условий эксплуатации технических средств клиентской части

# ОПИСАНИЕ ЛОГИЧЕКОЙ СТРУКТУРЫ

## 3.1 Структура программы

Для информационно-управляющей системы объекта использовалась объектно-ориентированная архитектура.

Выделим основные классы объектов (см.рис.1), которые могут быть использованы для реализации системы:

1. Simulation\_App – класс, предназначен для отображения интерфейса пользователя на экране и активацию сценария работы робота по запросам пользователя, а также отображения результата работы;
2. Air\_Robot – класс, описывающий модель поведения беспилотного летательного аппарата, и получающий данные с бортовых датчиков о характере движения;
3. Trajectory – класс, реализующий построение маршрута движения робота;
4. Regulator – класс, реализующий расчет управляющего воздействия для модели робота;
5. Calman\_Filter – класс, реализующий корректировку данных с бортовых датчиков в соответствии с моделью;
6. Camera – класс, отвечающий за получение изображения с бортовой камеры и расчет координат объектов.

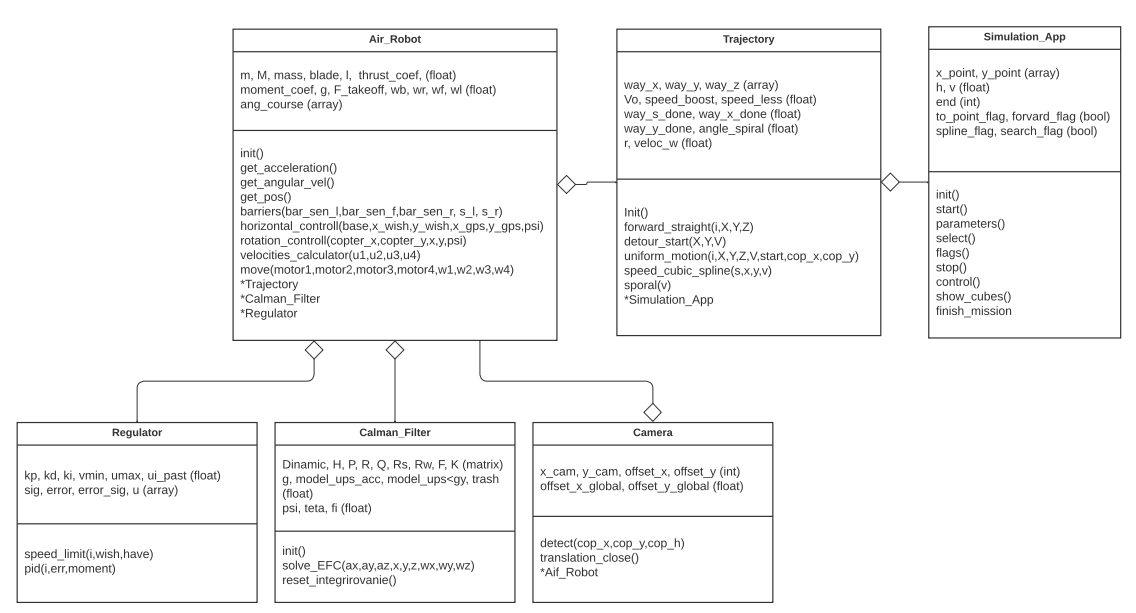


Рисунок 3 – Диаграмма классов

Разберем диаграмму классов, изображенную на рисунке 1.

Класс Simulation\_App отвечает за связь модели работа и человека, выполняет функции интерфейса работы с системой для оператора и определения сценария работы программы. В качестве данных используется полученные данные от пользователя о ключевых точках, скорости движения и высоте полета теля с использованием функций «parameters», которая в качестве значений возвращает параметры симуляции. Выбор миссии осуществляется через считывание отмеченного флажка функцией «select» и передачей значений в основной код функцией «flags». Запуск и остановка симуляции производятся вызовом функций «start» и «stop» соответственно. По завершении миссии вывод информационного окна с соответствующим текстом производится функцией «finish\_mission» и в случае работы робота в режиме поиска, результат работы в виде координат найденных объектов – «show\_cubes».

Класс Air\_robot отвечает за расчет моментов и сил, подаваемых на двигатели БПЛА, а также за считывание данных с бортовых датчиков. Реализация горизонтальной стабилизации робота происходит в функции «horizontal\_controll». В качестве параметров функции выступают объект управления, желаемые координаты по осям X и Y и координаты робота, а также угол вокруг оси Z. В функции создается два единичных вектора, направленных вдоль осей X и Y связанной системы координат, после чего производится расчет ошибки между осями квадрокоптера и глобальными осями X и Y соответственно. Предварительно связанные с БПЛА оси переводятся в глобальную систему с использованием матрицы поворота :

где угол поворота вокруг оси Z.

Также в данной функции производится обнуление нынешних координат квадрокоптера и перевод желаемых координат из абсолютной системы координат в связанную с использованием транспонированной матрицы :

Возвращаемыми значениями являются отклонения по осям X и Y, а также желаемые и нынешние координаты в связанной с роботом системе координат.

Расчет курса робота проводится в функции «rotation\_controll». На вход функция принимает позицию робота, желаемую позицию и угол поворота вокруг оси Z. В функции рассчитывается угол между векторами, длина которых соответствует разнице между нынешними ( и желаемыми ( координатами:

где курс робота, , .

Добавление в расчете синуса и косинуса обосновано движением квадрокоптера «крестом».

Расчет желаемых угловых скоростей вращения через управляющие воздействия производится в функции «velocities\_calculator» на вход функция принимает управляющие воздействия , отвечающих за контроль высоты, перемещения по осям X и Y и поворот вокруг оси Z соответственно. После чего происходит расчет желаемой угловой скорости для каждого двигателя:

где угловые скорости подаваемые на задний, правый, передний и левый двигатели соответственно, а коэффициенты тяги и момента двигателей.

Постепенное раскручивание или замедление вращения винтов рассчитывается через уравнение апериодического звена в функции «move»:

где угловая скорость, желаемая скорость, шаг, постоянная времени.

Перевод угловых скоростей вращения винтов в подаваемую силу и моменты осуществлен с использованием формул:

В этой же функции производится подача значений на симуляцию.

Класс Trajectory отвечает за расчет промежуточных точек траектории между ключевыми. В данном классе рассчитываются все варианты траектории, по которым может передвигаться квадрокоптер. В функции «forvard\_straight» происходит заполнение массивов траектории желаемыми значениями, подаваемыми на вход, данная функция используется в основном для удержания робота в одной точке. В функции «uniform\_motion» производится расчет прямолинейного движения в желаемую точку. Для корректной работы в функции также реализован постепенный набор скорости до желаемой и постепенное замедление по прохождении роботом пути. В функции «speed\_cubic\_spline» производится расчет траектории, задаваемой кубическим сплайном:

где .

В функции «spiral» рассчитывается Архимедова спираль по формулам:

где радиус спирали, расстояние между витками,угол поворота спирали.

В функции «detour\_start» производится расчет до каждой промежуточной точки траектории с заданной скоростью, данная функция необходима для реализации возможности объезда без потери расчета основной траектории.

Все функции класса Trajectiry принимают ключевые точки, высоту и значение скорости движения, а возвращают промежуточные точки траектории. Равномерное движение обеспечивается расчетом одинакового расстояния между двумя соседними точками.

Класс Calman\_Filter реализует в себе рекурсивный фильтр в функции «solve\_EFC», оценивающий вектор состояния динамической системы, используя ряд неполных и зашумленных измерений. Фильтр Калмана учитывает как ошибку модели, так и ошибку измерения при обновлении текущего состояния, что позволяет добиться наибольшей точности оценки. В начале производится инициализация, где обозначается вектор начальных состояний и матрицы ковариации P. Затем производится предсказание состояния системы и ошибки ковариации:

где вектор состояний, F – матрица динамики, Р – матрица ковариации, матрица шума модели.

После чего производится корректировка рассчитанных значений:

где матрица усиления фильтра, H – матрица датчиков, R – матрица шума датчиков, z – вектор значений с датчиков, а знак «–» означает расчет на предыдущем этапе. В функции «reset\_integrirovanie» производится сброс накопленной ошибки интегрирования значения угла

В классе Camera в функции «detect» реализовано распознавание объектов красного цвета с помощью библиотеки OpenCV. Изображение, полученное от камеры, преобразуется в HSV формат. Далее на изображение накладывается маска с верхней и нижней границей заданного цвета. Координаты середины определенного контура в пикселях определяются по следующим формулам:

где m01, m10, m00 – суммарный момент вдоль оси *Ox*, суммарный момент вдоль оси *Oy*, суммарный момент области кадра. Далее полученные координаты записываются в массив с учетом текущего положения робота. Функция «translation\_close» позволяет закрыть окно, в котором выводится изображение с бортовой камеры.

В классе Regulator реализован PID регулятор в одноименной функции «pid». Здесь на вход поступает ошибка, после чего рассчитывается управляющее воздействие:

где *P, I, D* – коэффициенты регулятора, *k* – номер дискретного шага, *е* – значение ошибки для регулирования.

Ограничение нарастающего сигнала реализовано в функции «speed\_limit». Реализацию данного ограничения можно описать при помощи следующего алгоритма:

1. Рассчитывается разность между сигналом , который подается на вход и который планируется подать
2. Если , то .
3. Если , то , где желаемая скорость входного сигнала.

## 3.2 Алгоритм программы

Диаграмма состояний представлена на рисунке 4.

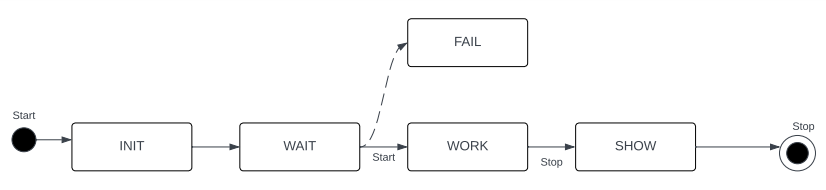


Рисунок 4 – Диаграмма состояний ИУС

Первое состояние, в котором находится ИУС после запуска – состояние инициализации «INIT». В этом состоянии происходит инициализация всех объектов и параметров ИУС. После окончания инициализации система переходит в состояние ожидания «WAIT», где она ожидает получение команды «Start» и контролирует отображение пользовательской консоли. После получения команды ИУС считывает все параметры, введенные в пользовательскую консоль, и переходит в состояние работы «WORK», где происходят все расчеты для выполнения миссии. В случае введения некорректных данных системы переходит в состояние «FAIL». По окончании выполнения мисси и по команде «Stop» система переходит в состояние «SHOW» в котором отображает результат работы программы.

Процесс взаимодействия объектов различных классов показан на диаграмме последовательности, представленной на рисунке 5.

Изначально производится считывание параметров движения, таких как ключевые точки, скорость, высота полета и траектория с использованием функций «parameters», «select» и «flags», затем робот считывает значения с бортовых датчиков («get\_pos», «get\_acceleration», «get\_angular\_vel»), после чего проводится фильтрация данных в функции «solve\_EFC». и получение значения промежуточных точек траектории («start\_detour»), рассчитанных предварительно в классе Trajectory в соответствии с параметрами x, y, h, v, flags. Затем следует расчет параметров движения («rotation\_controll» и «horizontall\_controll»), управляющих воздействий u1, u2, u3, u4 в функции «pid», перевод значений угловых скоростей в силы и моменты («velocities\_calculate») и подача значений на симуляцию через функцию «move». В случае обнаружения препятствия производится перерасчет траектории для объезда через обозначение расстояния до объекта (dist\_for\_barrier на диаграмме, в коде – показания с бортового дальномера, обнаружившего препятствие). В случае работы робота в режиме поиска дополнительно к алгоритму работы подключается поиск объектов – функция «detect». По завершении программы или при ее остановке выводится окно c найденными объектами – «show\_cebes».

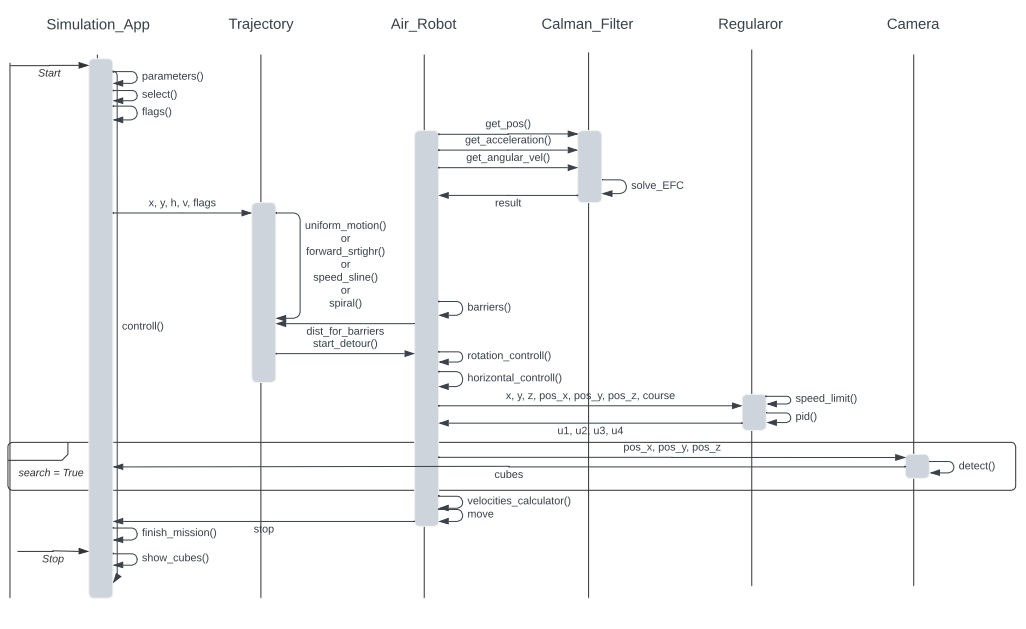


Рисунок 5 – Диаграмма последовательности

# ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

В состав технических средств должен входить персональный компьютер, включающий в себя:

* + 1. Процессор Core i5 с максимальной тактовой частотой от 2.4 ГГц;
    2. Оперативную память объемом от 8 Гб;
    3. Твердотельный накопитель от 100 Гб;
    4. Свободное место на твердотельном накопителе - от 50 Гб;
    5. Видеокартой с объемом видео памяти от 4 Гб и тактовой частотой от 1 МГц;
    6. Операционную систему Windows 10;
    7. Устройства управления и ввода: клавиатура, мышь.
    8. Устройство вывода: монитор с разрешением от 1920х1080

# ВЫЗОВ И ЗАГРУЗКА

Запуск программы осуществляется посредством открытия главного исполняемого файла «main.py» и запуска программы.

Запуск исполняемых файлов происходит с помощью среды разработки PyCharm Community.

# ВХОДНЫЕ ДАННЫЕ

Входными данными являются параметры движения, имеющие тип float, такие как скорость, высота и ключевые точки траектории. А также значения типа bool, определяющие режим миссии. Структура вводимых значений в случае использования вещественных чисел обязывает использование точки в качестве разделяющего знака. В случае введения нескольких значений в виджет Entry необходимо разделение пробелом.

# ВЫХОДНЫЕ ДАННЫЕ

Выходными данными является графический интерфейс для управления роботом, а также по окончании задания информационные окна с информацией о завершении миссии, графической информацией характеристик движения и координаты найденных объектов в случае выполнения соответствующей задачи. Также окно транслирующее изображение с бортовой камеры.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Д.т.н., доцент департамента автоматики и робототехники |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_ Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2023 |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

Информационно-управляющая система «FlightOps»

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

**Руководство системного программиста**

**ЛИСТ УТВЕРЖДЕНИЯ**

**Б11120-15.03.06 РСП 10.01 ЛУ**

|  |  |  |
| --- | --- | --- |
|  |  | Руководитель разработки |
|  |  | Д.т.н., доцент |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2021 |
|  |  |  |
|  |  | Исполнитель |
|  |  | Студент группы  Б11120-15.03.06мхрб |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Тумас М.В |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |

**2023**

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  Б11120-15.03.06 РСП 10.01 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Информационно-управляющая система «FlightOps»

**Руководство системного программиста**

**Б11120-15.03.06 РСП 10.01**

**Листов 6**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**2023**

# АННОТАЦИЯ

Данный документ содержит инструкцию для системного программиста по использованию программы «FlightOps», которая предназначена для модели беспилотного летательного аппарата с симуляцией функцией поиска объектов на земле.

«Руководство системного программиста» описывает общие сведения о программе, дает инструкции по настройке программы для конкретных условий использования и способам проверки ее работоспособности.

Документ оформлен в соответствии с требованиями ГОСТ 19.503-79.

СОДЕРЖАНИЕ

[1. Общие сведения о программе 57](#_Toc75151808)

[1.1 Назначение программы 57](#_Toc75151809)

[1.2 Функции программы 57](#_Toc75151810)

[1.3 Сведения о технических и программных средствах 57](#_Toc75151811)

[2. Структура программы 58](#_Toc75151812)

[2.1 Сведения о структуре программы 58](#_Toc75151813)

[2.2 Взаимодействие с другими программами 58](#_Toc75151814)

[3. Настройка на состав программных средств 58](#_Toc75151815)

[4. Проверка программы 58](#_Toc75151816)

[Лист регистрации изменений 60](#_Toc75151817)

# ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

## 1.1 Назначение программы

Программа предназначена для осуществления навигации робота по заданной пользователем траектории или в одном из трёх видов движения, а также поиска объектов на заданной площади карты.

## 1.2 Функции программы

Программа имеет возможность осуществлять движение робота в заданную точку с обходом препятствий с заданной скоростью на заданной высоте от поверхности или высоте(глубине); нахождение объектов с помощью камер и отображение координат объектов в терминале программы.

## 1.3 Сведения о технических и программных средствах

Программа должна эксплуатироваться на персональном компьютере включающего в себя:

* + 1. Процессор Core i5 с максимальной тактовой частотой от 3.6 ГГц;
    2. Оперативную память объемом от 16 Гб;
    3. Твердотельный накопитель от 100 Гб;
    4. Свободное место на твердотельном накопителе - от 50 Гб;
    5. Видеокартой с объемом видео памяти от 4 Гб и тактовой частотой от 1 МГц;
    6. Операционную систему Windows 10;
    7. Устройства управления и ввода: клавиатура, мышь.
    8. Устройство вывода: монитор с разрешением от 1920х1080

# СТРУКТУРА ПРОГРАММЫ

## 2.1 Сведения **о структуре программы**

Подробное описание структуры и работы программы приведено в программном документе «Описание программы» настоящего (см. стр. 19)

## 2.2 Взаимодействие с другими программами

Для работы программы необходима виртуальная среда моделирования CoppeliaSim, в сцену которого загружен необходимый объект управления (беспилотный летательный аппарат) с установленным набором сенсоров и бортовых датчиков.

# 3. НАСТРОЙКА НА СОСТАВ ПРОГРАММНЫХ СРЕДСТВ

Запуск программы осуществляется посредством открытия главного исполняемого файла «main.py». Открытие файла осуществляется через программу PyCharm Community. Затем необходимо загрузить сцену с объектом управления в среде моделирования CopelliaSim. После открытия всех программ необходимо провести запуск исполняемого файла.

# 4. ПРОВЕРКА ПРОГРАММЫ

После запуска программ,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Д.т.н., доцент департамента автоматики и робототехники |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_ Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

Информационно-управляющая система «FlightOps»

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

**Руководство оператора**

**ЛИСТ УТВЕРЖДЕНИЯ**

**Б11120-15.03.06 РO 10.01 ЛУ**

|  |  |  |
| --- | --- | --- |
|  |  | Руководитель разработки |
|  |  | Д.т.н., доцент |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |
|  |  |  |
|  |  | Исполнитель |
|  |  | Студент группы  Б11120-15.03.06мхрб |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Тумас М.В. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2023 |

**2023**

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  Б11120-15.03.06 РO 10.01 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Информационно-управляющая система «FlightOps»

**Руководство оператора**

**Б11120-15.03.06 РO 10.01**

**Листов 11**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**2021**

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора для программы «Purple», предназначенной для применения в подводных аппаратах с функцией поиска объектов. Программа реализована на языке программирования Python 3.9.4.

В данном программном документе описаны общие сведения о программе; условия, необходимые для выполнения программы; последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведены описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды; тексты сообщений, выдаваемых в ходе выполнения программы, описание их содержания и соответствующие действия оператора.

Оформление программного документа «Руководство оператора» произведено по требованиям ГОСТ 19.505-79.

[СОДЕРЖАНИЕ](#_Toc75151818)

[1. Общие сведения о программе 65](#_Toc75151819)

[1.1 Назначение программы 65](#_Toc75151820)

[1.2 Функции программы 65](#_Toc75151821)

[2. Условия выполнения программы 65](#_Toc75151822)

[2.1 Требования к аппаратным средствам 65](#_Toc75151823)

[2.2 Требования к программным средствам 66](#_Toc75151824)

[2.3 Требования к оператору 66](#_Toc75151825)

[3. Выполнение программы 66](#_Toc75151826)

[3.1 Запуск программы 66](#_Toc75151827)

[3.2 Начало работы 67](#_Toc75151828)

[3.3 Настройка начальных параметров 69](#_Toc75151829)

[3.4 Запуск работы системы 72](#_Toc75151830)

[Лист регистрации изменений 74](#_Toc75151831)

# ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

## 1.1 Назначение программы

Программа предназначена для осуществления навигации робота по одному из режимов функционирования: движение в точку с заданной скоростью, движение в точку по сплайну с заданной скоростью, движение в точку по прямой, выполнение миссии (проход последовательно-установленных точек с заданным типом движения), поиск объектов на карте; с заданными параметрами робота.

## 1.2 Функции программы

Программа имеет возможность осуществлять движение робота в заданную точку с обходом препятствий с заданной скоростью и на заданной высоте от поверхности/глубине; находить объекты красного цвета, расположенного в заранее неизвестном месте карты.

# УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

## 2.1 Требования к аппаратным средствам

Программа эксплуатируется на персональном компьютере включающего в себя:

* + 1. Процессор Core i5 с максимальной тактовой частотой от 3.6 ГГц;
    2. Оперативную память объемом от 16 Гб;
    3. Твердотельный накопитель от 100 Гб;
    4. Свободное место на твердотельном накопителе - от 50 Гб;
    5. Видеокартой с объемом видео памяти от 4 Гб и тактовой частотой от 1 МГц;
    6. Операционную систему Windows 10;
    7. Устройства управления и ввода: клавиатура, мышь.
    8. Устройство вывода: монитор с разрешением от 1920х1080

## 2.2 Требования к программным средствам

Для работы программы необходима виртуальная среда моделирования CoppeliaSim, в сцену которого загружен необходимый объект управления (подводный робот).

## 2.3 Требования к оператору

Требуется навык использования персонального компьютера, иметь опыт работы с графическими интерфейсами, иметь опыт работы в среде Visual Studio Code.

# 3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

## 3.1 Запуск программы

Запуск программы осуществляется посредством открытия главного исполняемого файла «main.py». После запуска интерфейса необходимо запустить сервер и после чего запустить исполнение файлов «controlSys.py», «navigationSys.py», «CAM.py», «trajectory.py». Запуск всех файлов необходимо производить в среде разработки Visual Studio Code. Также в CoppeliaSim необходимо загрузить сцену с объектом управления. В настройках сервера CoppeliaSim значение портов должно быть равно 19997 и 20001.

## 3.2 Начало работы

При запуске файла «main.py» оператор увидит пользовательский интерфейс, представленный на рисунке 6.

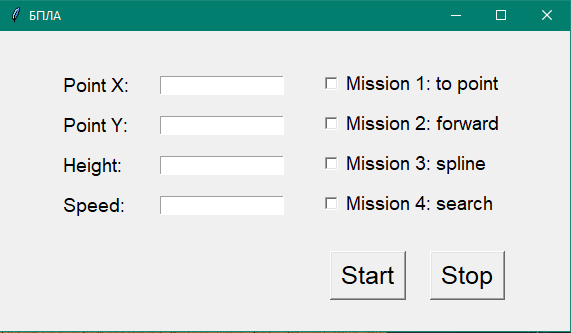


Рисунок 6 – интерфейс пользователя

В крайнем левом столбце представлены подписи, соответствующие вводимым значениям в виджеты Entry, расположенные правее. Обозначение «Point X» соответствует запросу на введение координат, расположенных по оси X глобальной системы координат, аналогично «Point Y» для координат на оси Y. Текстовое обозначение «Height» указывает на введение в соседний выждет значение желаемой высоты над поверхностью, а «Speed» на значение желаемой скорости.

Информационная структура вводимых значений в случае использования вещественных чисел обязывает использование точки в качестве разделяющего знака. В случае введения нескольких значений в виджет Entry необходимо разделение пробелом. Виджет советующий высоте полета предполагает использование одного значения, виджет соответствующий скорости предполагает использование 2 параметров в случае запуска миссии поиска и одного параметра в остальных случаях.

В правом столбце расположен chek-box позволяющий обозначить выполнение конкретной миссии. Для этого необходимо поставить отметку напротив соответствующей подписи.

«Mission 1: to point» соответствует мисси, реализующей движение в точку с заданной скоростью. «Mission 2: forward» соответствует мисси, реализующей прямолинейное движение в точку с заданной скоростью. «Mission 3: spline» соответствует мисси, реализующей криволинейное движение, задаваемое кубическим сплайном с заданной скоростью. «Mission 4: to point» соответствует мисси, реализующей движение БПЛА по Архимедовой спирали с поиском объектов красного цвета.

## 3.3 Настройка начальных параметров

Пример настройки начальных параметров приведен на рисунке 7.

Ограничения на вводимые параметры:

* + 1. Скорость робота от 0.5 до 3;
    2. Высота полета от 3.5 до 7;
    3. Вещественные значения вводятся через точку;
    4. В случае использования нескольких значений в одном виджете разделительный знак – пробел;
    5. Отметка одного флага в chek-box;
    6. В случае выбора мисси поиска виджеты для координат X и Y заполняются «0»;
    7. В случае выбора миссии поиска в виджете поиска первым значением указывается желаемая скорость движения по спирали, вторым – желаемая скорость движения в точку (0, 0) от позиции робота.

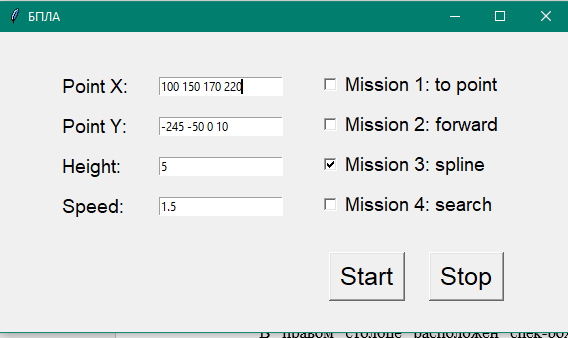


Рисунок 7 – пример заполнения консоли

## Запуск работы системы

После заполнения всех параметров необходимо нажать кнопку «Start».

По завершении миссии высветится информационное окно о завершении миссии (см. Рис 8).

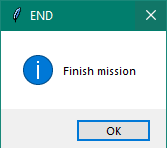


Рисунок 8 – информационное окно о завершении миссии

После чего необходимо нажать кнопку «ok» на информационном окне и «Stop» на пользовательской консоли. При выполнении вышеуказанных действий выведется окно с графическим отображением характеристик движения (см. Рис 9). В случае если окно с графическим отображением характеристик не высветилось необходимо закрыть пользовательскую консоль нажатием на крестик.

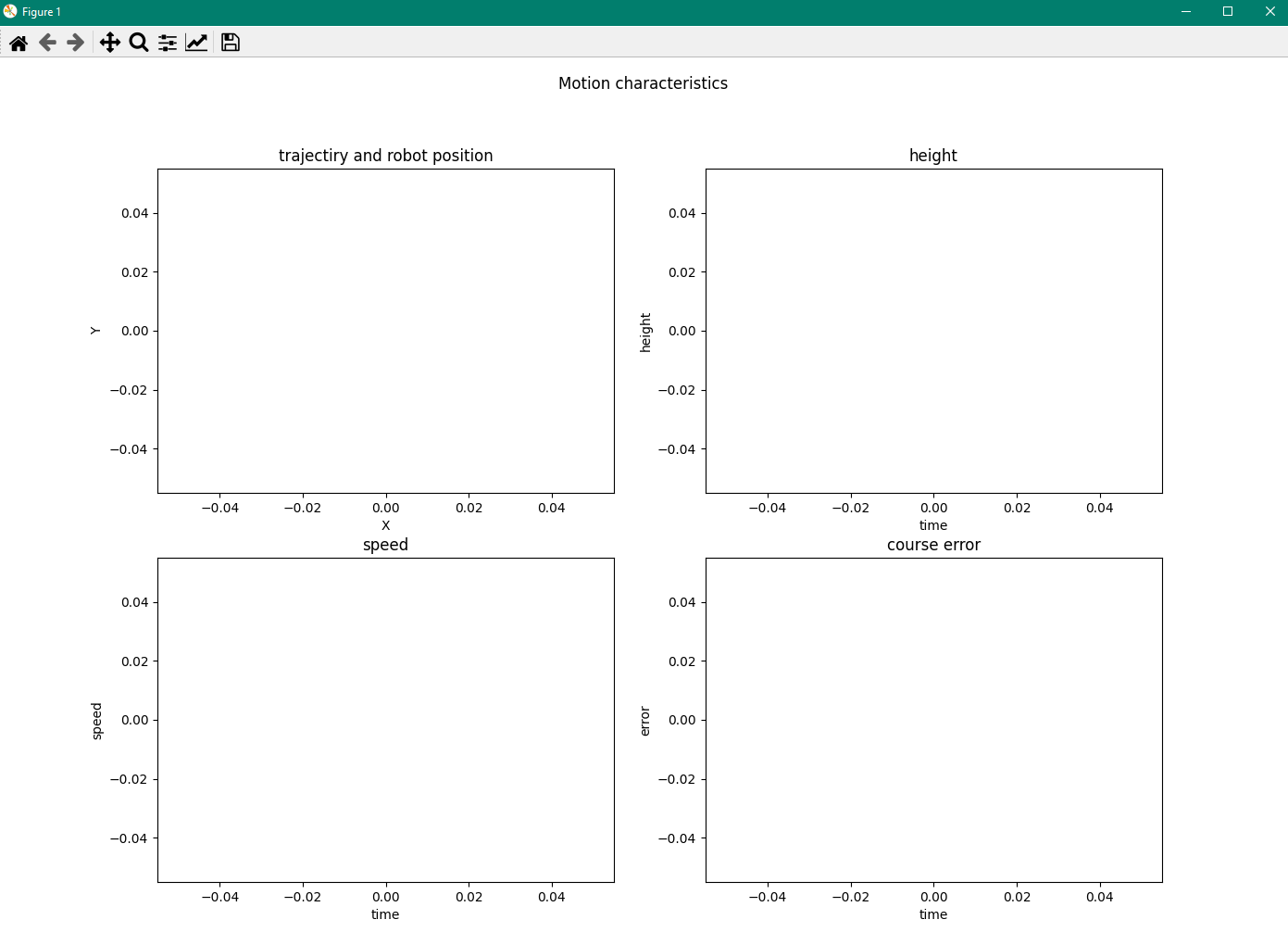


Рисунок 9 – окно с отображением графических характеристик движения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  | УТВЕРЖДАЮ |
|  |  | Д.т.н., доцент департамента автоматики и робототехники |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_ Юхимец Д.А. |
|  |  | «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Дальневосточный федеральный университет»**

Информационно-управляющая система «Purple»

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**РФ.ДВФУ.02067942-01 12 ТП 11-05 ЛУ**

|  |  |  |
| --- | --- | --- |
|  |  | Руководитель разработки |
|  |  | Д.т.н., доцент |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Юхимец Д.А. |
|  |  | «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2021 |
|  |  |  |
|  |  | Исполнитель |
|  |  | Студент группы  Б3118-15.03.06мхрт |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Садыков Р.М. |
|  |  | «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2021 |

2021

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН  РФ.ДВФУ.02067942-01 12 ТП 11-05 ЛУ |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Информационно-управляющая система «Purple»

**Текст программы**

**РФ.ДВФУ.02067942-01 12 ТП 11-05**

**Листов 53**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

2021

[СОДЕРЖАНИЕ](#_Toc75151832)

[1. Листинг модуля gui/server 78](#_Toc75151833)

[1.1. Файл ui\_loading.py 78](#_Toc75151834)

[1.2. Файл ui\_functions.py 88](#_Toc75151835)

[1.3. Файл ui\_main.py 88](#_Toc75151836)

[1.4. Файл main.py 100](#_Toc75151837)

[2. Листинг модуля control system 110](#_Toc75151838)

[3. Листинг модуля navigation system 117](#_Toc75151839)

[4. Листинг модуля camera 123](#_Toc75151840)

[5. Листинг модуля trajectory system 127](#_Toc75151841)

[Лист регистрации изменений 131](#_Toc75151842)

# 1. ЛИСТИНГ ПРОГРАММЫ

**import** numpy **as** np  
**import** cv2  
**from** zmqRemoteApi **import** RemoteAPIClient  
**import** matplotlib.pyplot **as** plt  
**from** tkinter **import** \*  
**from** tkinter **import** messagebox  
  
**class** Simulation\_App():  
 **def** \_\_init\_\_(self):  
 self.window = Tk()  
 self.window.title(**'БПЛА'**)  
 self.window.geometry(**'570x300'**)  
  
 self.end = 0  
  
 self.x\_point = []  
 self.y\_point = []  
 self.h = 1  
 self.v = 0.5  
  
 self.to\_point\_flag = **False** self.forward\_flag = **False** self.spline\_flag = **False** self.search\_flag = **False** self.x\_lb = Label(text=**"Point X: "**, font=**"device 14 normal roman"**)  
 self.x\_lb.place(x=60, y=40)  
  
 self.y\_lb = Label(text=**"Point Y: "**, font=**"device 14 normal roman"**)  
 self.y\_lb.place(x=60, y=80)  
  
 self.height\_lb = Label(text=**"Height: "**, font=**"device 14 normal roman"**)  
 self.height\_lb.place(x=60, y=120)  
  
 self.speed\_lb = Label(text=**"Speed: "**, font=**"device 14 normal roman"**)  
 self.speed\_lb.place(x=60, y=160)  
  
 self.x\_tf = Entry()  
 self.x\_tf.place(x=160, y=45)  
  
 self.y\_tf = Entry()  
 self.y\_tf.place(x=160, y=85)  
  
 self.height\_tf = Entry()  
 self.height\_tf.place(x=160, y=125)  
  
 self.speed\_tf = Entry()  
 self.speed\_tf.place(x=160, y=165)  
  
 self.to\_point = IntVar()  
 self.mission\_1\_to\_point = Checkbutton(text=**"Mission 1: to point"**, font=**"device 14 normal roman"**,  
 variable=self.to\_point, command=self.select)  
 self.mission\_1\_to\_point.place(x=320, y=35)  
  
 self.uniform = IntVar()  
 self.mission\_2\_uniform = Checkbutton(text=**"Mission 2: forward"**, font=**"device 14 normal roman"**,  
 variable=self.uniform, command=self.select)  
 self.mission\_2\_uniform.place(x=320, y=75)  
  
 self.spline = IntVar()  
 self.mission\_3\_spline = Checkbutton(text=**"Mission 3: spline"**, font=**"device 14 normal roman"**,  
 variable=self.spline, command=self.select)  
 self.mission\_3\_spline.place(x=320, y=115)  
  
 self.search = IntVar()  
 self.mission\_4\_search = Checkbutton(text=**"Mission 4: search"**, font=**"device 14 normal roman"**,  
 variable=self.search, command=self.select)  
 self.mission\_4\_search.place(x=320, y=155)  
  
 self.start\_btn = Button(text=**'Start'**, font=**"device 19 normal roman"**, command=self.start)  
 self.start\_btn.place(x=330, y=220)  
  
 self.stop\_btn = Button(text=**'Stop'**, font=**"device 19 normal roman"**, command=self.stop)  
 self.stop\_btn.place(x=430, y=220)  
  
 **def** start(self):  
 sim.startSimulation()  
  
 **def** parameters(self):  
 self.x\_point = (self.x\_tf.get())  
 self.y\_point = (self.y\_tf.get())  
 self.h = (self.height\_tf.get())  
 self.v = (self.speed\_tf.get())  
  
 self.x\_point = list(map(float, self.x\_point.split()))  
 self.y\_point = list(map(float, self.y\_point.split()))  
 self.h = list(map(float, self.h.split()))  
 self.v = list(map(float, self.v.split()))  
 **return**(self.x\_point, self.y\_point, self.h, self.v)  
  
  
 **def** select(self):  
 **if** self.to\_point.get() == 1:  
 self.to\_point\_flag = **True** self.forward\_flag = **False** self.spline\_flag = **False** self.search\_flag = **False  
  
 if** self.uniform.get() == 1:  
 self.to\_point\_flag = **False** self.forward\_flag = **True** self.spline\_flag = **False** self.search\_flag = **False  
  
 if** self.spline.get() == 1:  
 self.to\_point\_flag = **False** self.forward\_flag = **False** self.spline\_flag = **True** self.search\_flag = **False  
  
 if** self.search.get() == 1:  
 self.to\_point\_flag = **False** self.forward\_flag = **False** self.spline\_flag = **False** self.search\_flag = **True  
  
 def** flags(self):  
 **return** self.to\_point\_flag, self.forward\_flag, self.spline\_flag, self.search\_flag  
  
  
 **def** stop(self):  
 sim.stopSimulation()  
 self.end = 1  
  
 **def** controll(self):  
 self.window.update()  
  
 **def** show\_cubes(self, x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8, x9, y9, x10, y10):  
 self.msg1 = **'found cubes coordinates\n'** self.msg2 = **f'1 : {**round(x1, 4), round(y1, 4)**}\n'** self.msg3 = **f'2 : {**round(x2, 4), round(y2, 4)**}\n'** self.msg4 = **f'3 : {**round(x3, 4), round(y3, 4)**}\n'** self.msg5 = **f'4 : {**round(x4, 4), round(y4, 4)**}\n'** self.msg6 = **f'5 : {**round(x5, 4), round(y5, 4)**}\n'** self.msg7 = **f'6 : {**round(x6, 4), round(y6, 4)**}\n'** self.msg8 = **f'7 : {**round(x7, 4), round(y7, 4)**}\n'** self.msg9 = **f'8 : {**round(x8, 4), round(y8, 4)**}\n'** self.msg10 = **f'9 : {**round(x9, 4), round(y9, 4)**}\n'** self.msg11 = **f'10 : {**round(x10, 4), round(y10, 4)**}'** self.msg = self.msg1 + self.msg2 + self.msg3 + self.msg4 + self.msg5 + self.msg6 + self.msg7 + self.msg8 + self.msg9 + self.msg10 + self.msg11  
 top = Toplevel()  
 top.geometry(**'300x350'**)  
 Label(top, text = self.msg, font=**"device 14 normal roman"**).pack(expand=1)  
 top.mainloop()  
  
 **def** finish\_mission(self):  
 self.msg = **'Finish mission'** messagebox.showinfo(title=**"END"**, message=self.msg)  
  
  
**class** Trajectory():  
 **def** \_\_init\_\_(self):  
 self.way\_x = np.zeros(int(Time / step))  
 self.way\_y = np.zeros(int(Time / step))  
 self.way\_z = np.zeros(int(Time / step))  
 self.Vo = 1  
 self.speed\_boost = 0.005  
 self.speed\_less = 0.005  
 self.way\_s\_done = 0  
 self.way\_x\_done = 0  
 self.way\_y\_done = 0  
  
 self.angle\_spiral = 0  
 self.r = 0  
 self.veloc\_w = 0.01  
  
 **def** forward\_straight(self, i, X, Y, Z):  
 self.way\_x[i] = X  
 self.way\_y[i] = Y  
 self.way\_z[i] = Z  
 **return** self.way\_x[i], self.way\_y[i], self.way\_z[i]  
  
 **def** detour\_start(self, X, Y, V):  
  
 **if** X > 0 **and** Y > 0:  
 self.Angle = np.arctan2(Y, X)  
 **if** X > 0 **and** Y < 0:  
 self.Angle = np.arctan2(Y, X)  
 **if** X < 0 **and** Y >= 0:  
 self.Angle = np.arctan2(Y, X) - np.pi  
 **if** X < 0 **and** Y < 0:  
 self.Angle = np.arctan2(Y, X) + np.pi  
 **if** X == 0 **and** Y > 0:  
 self.Angle = np.pi / 2  
 **if** X == 0 **and** Y < 0:  
 self.Angle = -np.pi / 2  
 **if** X == 0 **and** Y == 0:  
 self.Angle = 0  
  
 self.Vx\_new = np.cos(self.Angle) \* V  
 self.Vy\_new = np.sin(self.Angle) \* V  
  
 **if** X >= 0:  
 self.way\_x\_detour = self.Vx\_new  
 self.way\_y\_detour = self.Vy\_new  
 **if** X < 0:  
 self.way\_x\_detour = -self.Vx\_new  
 self.way\_y\_detour = -self.Vy\_new  
  
 **if** abs(self.way\_x\_detour) > abs(X):  
 self.way\_x\_detour = X  
 **if** abs(self.way\_y\_detour) > abs(Y):  
 self.way\_y\_detour = Y  
  
 **return** self.way\_x\_detour, self.way\_y\_detour  
  
 **def** uniform\_motion(self, i, X, Y, Z, V, start, cop\_x, cop\_y):  
 **if** start == 0:  
 self.way\_x[i - 1] = cop\_x  
 self.way\_y[i - 1] = cop\_y  
 self.ang\_x = X - self.way\_x[i - 1]  
 self.ang\_y = Y - self.way\_y[i - 1]  
 self.way\_s = np.sqrt((X - self.way\_x[i-1]) \*\* 2 + (Y - self.way\_y[i-1]) \*\* 2)  
  
 **if** self.ang\_x > 0 **and** self.ang\_y > 0:  
 self.angle = np.arctan2(self.ang\_y, self.ang\_x)  
 **if** self.ang\_x > 0 **and** self.ang\_y < 0:  
 self.angle = np.arctan2(self.ang\_y, self.ang\_x)  
 **if** self.ang\_x < 0 **and** self.ang\_y >= 0:  
 self.angle = np.arctan2(self.ang\_y, self.ang\_x) - np.pi  
 **if** self.ang\_x < 0 **and** self.ang\_y < 0:  
 self.angle = np.arctan2(self.ang\_y, self.ang\_x) + np.pi  
 **if** self.ang\_x == 0 **and** self.ang\_y > 0:  
 self.angle = np.pi / 2  
 **if** self.ang\_x == 0 **and** self.ang\_y < 0:  
 self.angle = -np.pi / 2  
 **if** self.ang\_x == 0 **and** self.ang\_y == 0:  
 self.angle = 0  
 **if** V > 1:  
 self.Vo = self.Vo + self.speed\_boost  
 self.speed\_boost = self.speed\_boost + 0.001  
 **if** self.Vo >= V:  
 self.Vo = V  
 self.speed\_boost = 0  
 **else**:  
 self.Vo = V  
 **if** self.way\_s\_done >= 0.85 \* self.way\_s:  
 self.Vo = self.Vo - self.speed\_less  
 self.speed\_less = self.speed\_less + 0.001  
 **if** self.Vo < 0.1:  
 self.Vo = 0.1  
 **else**:  
 self.speed\_less = 0.001  
  
 self.Vx = np.cos(self.angle) \* self.Vo  
 self.Vy = np.sin(self.angle) \* self.Vo  
 **if** self.ang\_x >= 0:  
 **if** X <= self.way\_x[i - 1] + self.Vx \* step:  
 self.way\_x[i] = X  
 **else**:  
 self.way\_x[i] = self.way\_x[i - 1] + self.Vx \* step  
 self.way\_y[i] = self.way\_y[i - 1] + self.Vy \* step  
 **if** Y <= self.way\_y[i - 1] + self.Vy \* step **and** self.ang\_y > 0:  
 self.way\_y[i] = Y  
 **if** Y >= self.way\_y[i - 1] + self.Vy \* step **and** self.ang\_y < 0:  
 self.way\_y[i] = Y  
 **if** self.ang\_x < 0:  
 **if** X >= self.way\_x[i - 1] - self.Vx \* step:  
 self.way\_x[i] = X  
 **else**:  
 self.way\_x[i] = self.way\_x[i - 1] - self.Vx \* step  
 self.way\_y[i] = self.way\_y[i - 1] - self.Vy \* step  
 **if** Y <= self.way\_y[i - 1] - self.Vy \* step **and** self.ang\_y > 0:  
 self.way\_y[i] = Y  
 **if** Y >= self.way\_y[i - 1] - self.Vy \* step **and** self.ang\_y < 0:  
 self.way\_y[i] = Y  
 self.way\_s\_done = self.way\_s\_done + self.Vo \* step  
 self.way\_z[i] = Z  
 **return** self.way\_x[i], self.way\_y[i], self.way\_z[i]  
  
 **def** speed\_cubic\_spline(self, s, x, y, v):  
 self.sigment\_want = v \* step  
 self.n = len(x)  
 self.h = np.zeros(self.n - 1)  
 **for** i **in** range(self.n - 1):  
 self.h[i] = x[i + 1] - x[i]  
  
 self.A = np.zeros((self.n, self.n))  
 self.A[0][0] = 1  
 self.A[self.n - 1][self.n - 1] = 1  
 **for** i **in** range(1, self.n - 1):  
 self.A[i][i] = 2 \* (self.h[i - 1] + self.h[i])  
 self.A[i][i - 1] = self.h[i - 1]  
 self.A[i][i + 1] = self.h[i]  
  
 self.b = np.zeros(self.n)  
 **for** i **in** range(1, self.n - 1):  
 self.b[i] = 3 \* ((y[i + 1] - y[i]) / self.h[i] - (y[i] - y[i - 1]) / self.h[i - 1])  
  
 self.c = np.linalg.solve(self.A, self.b)  
 self.d = np.zeros(self.n - 1)  
 self.b = np.zeros(self.n - 1)  
 **for** i **in** range(self.n - 1):  
 self.d[i] = (self.c[i + 1] - self.c[i]) / (3 \* self.h[i])  
 self.b[i] = (y[i + 1] - y[i]) / self.h[i] - (self.h[i] / 3) \* (2 \* self.c[i] + self.c[i + 1])  
 self.n = len(x)  
 self.xx = np.linspace(x[0], x[-1], 10000)  
 self.yy = np.zeros(len(self.xx))  
 self.l = 0  
 self.spline\_x = []  
 self.spline\_y = []  
 **for** i **in** range(len(self.xx)):  
 j = 0  
 **while** j < self.n - 1 **and** self.xx[i] > x[j + 1]:  
 j += 1  
 self.yy[i] = y[j] + self.b[j] \* (self.xx[i] - x[j]) + self.c[j] \* (self.xx[i] - x[j]) \*\* 2 + self.d[j] \* (self.xx[i] - x[j]) \*\* 3  
  
 self.spline\_x.append(self.xx[0])  
 self.spline\_y.append(self.yy[0])  
 **for** k **in** range(len(self.xx)):  
 self.len\_sigment = np.sqrt((self.xx[self.l] - self.xx[k]) \*\* 2 + (self.yy[self.l] - self.yy[k]) \*\* 2)  
 **if** self.sigment\_want <= self.len\_sigment:  
 self.spline\_x.append(self.xx[k])  
 self.spline\_y.append(self.yy[k])  
 self.l = k  
 self.spline\_x.append(self.xx[len(self.xx) - 1])  
 self.spline\_y.append(self.yy[len(self.xx) - 1])  
 **if** s >= len(self.spline\_x) - 1:  
 **return** self.xx[len(self.xx) - 1], self.yy[len(self.yy) - 1]  
 **else**:  
 **return** self.spline\_x[s], self.spline\_y[s]  
  
 **def** spiral(self, v):  
 self.v\_start = self.veloc\_w \* self.r / 0.05  
 self.dist\_vit = 8  
 **if** self.v\_start >= v:  
 self.veloc\_w = v \* 0.05 / self.r  
 self.r = self.dist\_vit / (2 \* np.pi) \* self.angle\_spiral  
 self.angle\_spiral = self.angle\_spiral + self.veloc\_w  
 x = self.r \* np.cos(self.angle\_spiral)  
 y = self.r \* np.sin(self.angle\_spiral)  
 **return** x, y  
  
  
**class** Air\_Robot():  
 **def** \_\_init\_\_(self, mass\_dvig, mass\_base, blade, l, Fd, Md):  
 self.m = mass\_dvig  
 self.M = mass\_base  
 self.mass = 4 \* self.m + self.M  
 self.blade = blade  
 self.l = l  
 self.thrust\_coefficient = Fd  
 self.moment\_coefficient = Md  
  
 self.g = 9.81  
 self.F\_takeoff = self.mass \* self.g  
  
 self.wb = 0  
 self.wr = 0  
 self.wf = 0  
 self.wl = 0  
  
 self.ang\_course = np.zeros(int(Time / step))  
  
 **def** get\_acceleration(self):  
 self.acc = np.zeros(3)  
 self.acc = sim.getStringSignal(**"ACC"**)  
 **if** self.acc != **None**:  
 self.acc = sim.unpackFloatTable(self.acc)  
 **return** self.acc[0], self.acc[1], self.acc[2]  
 **else**:  
 **return** 0, 0, 0  
  
 **def** get\_angular\_vel(self):  
 self.angels = np.zeros(3)  
 self.angels = sim.getStringSignal(**"GYRO"**)  
 **if** self.angels != **None**:  
 self.angels = sim.unpackFloatTable(self.angels)  
 **return** round(self.angels[0], 4), round(self.angels[1], 4), round(self.angels[2], 4)  
 **else**:  
 **return** 0, 0, 0  
  
 **def** get\_pos(self):  
 self.position = np.zeros(3)  
 self.position = sim.getStringSignal(**"GPS"**)  
 **if** self.position != **None**:  
 self.position = sim.unpackFloatTable(self.position)  
  
 self.hight = sim.handleProximitySensor(dist\_laser)  
 self.position[2] = self.hight[1]  
 **return** round(self.position[0], 4), round(self.position[1], 4), round(self.position[2], 4)  
 **else**:  
 **return** 0, 0, 0  
  
 **def** barriers(self, bar\_sen\_l, bar\_sen\_f, bar\_sen\_r, s\_l, s\_r):  
 self.bar\_l = sim.handleProximitySensor(bar\_sen\_l)  
 self.bar\_f = sim.handleProximitySensor(bar\_sen\_f)  
 self.bar\_r = sim.handleProximitySensor(bar\_sen\_r)  
 self.bar\_Sl = sim.handleProximitySensor(s\_l)  
 self.bar\_Sr = sim.handleProximitySensor(s\_r)  
 **return** self.bar\_l[0], self.bar\_f[0], self.bar\_r[0], self.bar\_l[1], self.bar\_f[1], self.bar\_r[1], self.bar\_Sl[0], self.bar\_Sr[0], self.bar\_Sl[1], self.bar\_Sr[1]  
  
 **def** horizontal\_controll(self, base, x\_wish, y\_wish, x\_gps, y\_gps, psi):  
 self.base = base  
 self.transion\_matrix = sim.getObjectMatrix(self.base, sim.handle\_world)  
 self.Ox = [1, 0, 0]  
 self.Ox = sim.multiplyVector(self.transion\_matrix, self.Ox)  
 self.Oy = [0, 1, 0]  
 self.Oy = sim.multiplyVector(self.transion\_matrix, self.Oy)  
  
 self.x\_copter = x\_gps \* np.cos(psi) + y\_gps \* np.sin(psi)  
 self.y\_copter = -x\_gps \* np.sin(psi) + y\_gps \* np.cos(psi)  
  
 self.x\_wish\_c = x\_wish \* np.cos(psi) + y\_wish \* np.sin(psi) - self.x\_copter  
 self.y\_wish\_c = -x\_wish \* np.sin(psi) + y\_wish \* np.cos(psi) - self.y\_copter  
  
 self.x\_copter = 0  
 self.y\_copter = 0  
  
 self.err\_teta = self.Ox[2] - self.transion\_matrix[11]  
 self.err\_fi = self.Oy[2] - self.transion\_matrix[11]  
  
 **return** self.err\_teta, self.err\_fi, self.x\_wish\_c, self.y\_wish\_c, self.x\_copter, self.y\_copter  
  
 **def** rotation\_controll(self, copter\_x, copter\_y, x, y, psi):  
  
 self.ex = (x - copter\_x)  
 self.ey = (y - copter\_y)  
 self.rx = np.cos(psi+np.pi/4)  
 self.ry = np.sin(psi+np.pi/4)  
  
 self.ang\_course[i] = np.arccos(((self.ex \* self.rx) + (self.ey \* self.ry)) / (np.sqrt(self.ex\*\*2 +self.ey \*\* 2) \* np.sqrt(self.rx\*\*2 + self.ry\*\*2)))  
 self.ang\_course[i] = self.ang\_course[i] \* np.sign(-self.ex \* np.sin(psi+np.pi/4) + self.ey \* np.cos(psi+np.pi/4))  
 **if** abs(self.ex) <= 0.0001 **and** abs(self.ey) <= 0.0001:  
 self.ang\_course[i] = 0  
 **return** self.ang\_course[i]  
  
  
 **def** velocities\_calculator(self, u1, u2, u3, u4):  
 u1 = u1 + self.F\_takeoff  
 self.wb\_wish = np.sqrt(abs(1 / (4 \* self.thrust\_coefficient) \* u1 - 1 / (2 \* self.thrust\_coefficient \* self.l) \* u3 + 1 / (4 \* self.moment\_coefficient) \* u4))  
 self.wr\_wish = np.sqrt(abs(1 / (4 \* self.thrust\_coefficient) \* u1 - 1 / (2 \* self.thrust\_coefficient \* self.l) \* u2 - 1 / (4 \* self.moment\_coefficient) \* u4))  
 self.wf\_wish = np.sqrt(abs(1 / (4 \* self.thrust\_coefficient) \* u1 + 1 / (2 \* self.thrust\_coefficient \* self.l) \* u3 + 1 / (4 \* self.moment\_coefficient) \* u4))  
 self.wl\_wish = np.sqrt(abs(1 / (4 \* self.thrust\_coefficient) \* u1 + 1 / (2 \* self.thrust\_coefficient \* self.l) \* u2 - 1 / (4 \* self.moment\_coefficient) \* u4))  
 **return** self.wb\_wish, self.wr\_wish, self.wf\_wish, self.wl\_wish  
  
 **def** move(self, motor1, motor2, motor3, motor4, w1, w2, w3, w4): *# back, right, front, left* self.Td = 0.05  
 self.wb = self.wb + (w1 - self.wb) \* step / self.Td  
 self.wr = self.wr + (w2 - self.wr) \* step / self.Td  
 self.wf = self.wf + (w3 - self.wf) \* step / self.Td  
 self.wl = self.wl + (w4 - self.wl) \* step / self.Td  
  
 self.Fb = self.wb \*\* 2 \* self.thrust\_coefficient  
 self.Fr = self.wr \*\* 2 \* self.thrust\_coefficient  
 self.Ff = self.wf \*\* 2 \* self.thrust\_coefficient  
 self.Fl = self.wl \*\* 2 \* self.thrust\_coefficient  
  
 self.Mb = self.wb \*\* 2 \* self.moment\_coefficient  
 self.Mr = self.wr \*\* 2 \* self.moment\_coefficient  
 self.Mf = self.wf \*\* 2 \* self.moment\_coefficient  
 self.Ml = self.wl \*\* 2 \* self.moment\_coefficient  
  
 sim.addForce(motor1, (0, 0, 0), (0, 0, self.Fb)) *# куда, позиция, сколько силы* sim.addForce(motor3, (0, 0, 0), (0, 0, self.Ff))  
 sim.addForce(motor2, (0, 0, 0), (0, 0, self.Fr))  
 sim.addForce(motor4, (0, 0, 0), (0, 0, self.Fl))  
  
 sim.addForce(motor4, (0, -self.blade, 0), (-self.Ml, 0, 0))  
 sim.addForce(motor4, (0, self.blade, 0), (self.Ml, 0, 0))  
  
 sim.addForce(motor1, (0, self.blade, 0), (-self.Mb, 0, 0)) *# пара моментов* sim.addForce(motor1, (0, -self.blade, 0), (self.Mb, 0, 0))  
  
 sim.addForce(motor3, (0, -self.blade, 0), (self.Mf, 0, 0))  
 sim.addForce(motor3, (0, self.blade, 0), (-self.Mf, 0, 0))  
  
 sim.addForce(motor2, (0, self.blade, 0), (self.Mr, 0, 0))  
 sim.addForce(motor2, (0, -self.blade, 0), (-self.Mr, 0, 0))  
  
  
**class** Camera():  
 **def** detect(self, cop\_x, cop\_y, cop\_h):  
 self.color\_red = (0, 0, 0)  
 self.color\_blue = (255, 0, 0)  
 self.img, self.resX, self.resY = sim.getVisionSensorCharImage(visionSensorHandle)  
 self.img = np.frombuffer(self.img, dtype=np.uint8).reshape(self.resY, self.resX, 3)  
 self.img = cv2.flip(cv2.cvtColor(self.img, cv2.COLOR\_BGR2RGB), 0)  
 self.Centr\_X = int(self.resX / 2)  
 self.Centr\_Y = int(self.resY / 2)  
 self.h\_min = np.array((0, 0, 161), np.uint8)  
 self.h\_max = np.array((33, 71, 255), np.uint8)  
 self.thresh = cv2.inRange(self.img, self.h\_min, self.h\_max)  
 self.moments = cv2.moments(self.thresh, 1)  
 self.dM01 = self.moments[**'m01'**]  
 self.dM10 = self.moments[**'m10'**]  
 self.dArea = self.moments[**'m00'**]  
 self.offset\_x, self.offset\_y = 0, 0  
 self.offset\_x\_global, self.offset\_y\_global = 0, 0  
  
 **if** self.dArea > 7:  
 self.x\_cam = int(self.dM10 / self.dArea)  
 self.y\_cam = int(self.dM01 / self.dArea)  
 self.offset\_x = (self.x\_cam - self.Centr\_X)  
 self.offset\_y = (self.Centr\_Y - self.y\_cam)  
  
 self.offset\_x\_global = (self.offset\_x \* (cop\_h-0.03) / 128) + abs(cop\_x)  
 self.offset\_y\_global = (self.offset\_y \* (cop\_h-0.03) / 128) + abs(cop\_y)  
 self.offset\_x\_global = self.offset\_x\_global \* np.sign(cop\_x)  
 self.offset\_y\_global = self.offset\_y\_global \* np.sign(cop\_y)  
  
 cv2.circle(self.img, (self.x\_cam, self.y\_cam), 2, self.color\_blue, 1)  
 cv2.putText(self.img, **"x%d;y%d"** % (self.x\_cam, self.y\_cam), (self.x\_cam + 5, self.y\_cam - 5),  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, self.color\_blue, 1)  
  
 cv2.circle(self.img, (self.Centr\_X, self.Centr\_Y), 2, self.color\_red, 1)  
 cv2.putText(self.img, **"x%d;y%d"** % (self.x\_cam - self.Centr\_X, self.Centr\_Y - self.y\_cam),  
 (self.Centr\_X + 5, self.Centr\_Y - 5),  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, self.color\_red, 1)  
 cv2.line(self.img, (self.x\_cam, self.y\_cam), (self.Centr\_X, self.Centr\_Y), self.color\_blue, 1)  
  
 cv2.imshow(**'CopterIsSeeing'**, self.img) *# название окна и вывод изображения* cv2.imshow(**'result'**, self.thresh)  
 cv2.waitKey(1) *# число-миллисекунд через сколько обновится изображение 0-ждет любую клавишу для отображения* **return** self.offset\_x, self.offset\_y, self.offset\_x\_global, self.offset\_y\_global  
  
 **def** translation\_close(self):  
 cv2.destroyAllWindows()  
  
  
**class** Calman\_Filrter():  
 **def** \_\_init\_\_(self):  
 self.Dinamic = ([[0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]])  
 self.H = np.zeros((9, 15))  
 self.H[3][6] = 1  
 self.H[4][7] = 1  
 self.H[5][8] = 1  
 self.H[6][9] = 1  
 self.H[7][10] = 1  
 self.H[8][11] = 1  
 self.P = np.identity(15) \* 10  
 self.g = 9.81  
 self.trash = 0.1  
 self.R = np.identity(9) \* self.trash  
 self.model\_ups\_acc = 1  
 self.model\_ups\_gy = 70  
 self.Q = np.diag([self.model\_ups\_acc, self.model\_ups\_acc, self.model\_ups\_acc,  
 0, 0, 0,  
 0, 0, 0,  
 self.model\_ups\_gy, self.model\_ups\_gy, self.model\_ups\_gy,  
 0, 0, 0])  
 self.Q = self.Q \* step  
  
 **def** solve\_EFC(self, ax\_input, ay\_input, az\_input, x\_input, y\_input, z\_input, wx\_input, wy\_input, wz\_input):  
 self.input = (  
 [[-ax\_input], [-ay\_input], [-az\_input], [x\_input], [y\_input], [z\_input], [wx\_input], [wy\_input],  
 [wz\_input]])  
  
 self.fi = self.Dinamic[12][0]  
 self.teta = self.Dinamic[13][0]  
 self.psi = self.Dinamic[14][0]  
  
 self.Rs = np.array([[np.cos(self.psi) \* np.cos(self.teta),  
 np.cos(self.psi) \* np.sin(self.fi) \* np.sin(self.fi) - np.cos(self.fi) \* np.sin(self.psi),  
 np.sin(self.fi) \* np.sin(self.psi) + np.cos(self.psi) \* np.sin(self.teta)],  
 [np.sin(self.psi) \* np.cos(self.teta),  
 np.cos(self.fi) \* np.cos(self.psi) + np.sin(self.teta) \* np.sin(self.psi),  
 np.cos(self.fi) \* np.sin(self.psi) \* np.sin(self.teta) - np.cos(self.psi) \* np.sin(  
 self.fi)],  
 [-np.sin(self.teta), np.cos(self.teta) \* np.sin(self.fi),  
 np.cos(self.fi) \* np.cos(self.teta)]])  
  
 self.Rw = np.array([[1, 0, -np.sin(self.teta)],  
 [0, np.cos(self.fi), np.sin(self.fi) \* np.cos(self.teta)],  
 [0, -np.sin(self.fi), np.cos(self.fi) \* np.cos(self.teta)]])  
  
 self.F = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [self.Rs[0][0] \* step, self.Rs[0][1] \* step, self.Rs[0][2] \* step, 1, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0],  
 [self.Rs[1][0] \* step, self.Rs[1][1] \* step, self.Rs[1][2] \* step, 0, 1, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0],  
 [self.Rs[2][0] \* step, self.Rs[2][1] \* step, self.Rs[2][2] \* step, 0, 0, 1, 0, 0, 0, 0, 0, 0,  
 0, 0, 0],  
 [self.Rs[0][0] \* step \*\* 2 / 2, self.Rs[0][1] \* step \*\* 2 / 2, self.Rs[0][2] \* step \*\* 2 / 2,  
 step, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
 [self.Rs[1][0] \* step \*\* 2 / 2, self.Rs[1][1] \* step \*\* 2 / 2, self.Rs[1][2] \* step \*\* 2 / 2,  
 0, step, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  
 [self.Rs[2][0] \* step \*\* 2 / 2, self.Rs[2][1] \* step \*\* 2 / 2, self.Rs[2][2] \* step \*\* 2 / 2,  
 0, 0, step, 0, 0, 1, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, self.Rw[0][0] \* step, self.Rw[0][1] \* step, self.Rw[0][2] \* step,  
 1, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, self.Rw[1][0] \* step, self.Rw[1][1] \* step, self.Rw[1][2] \* step,  
 0, 1, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, self.Rw[2][0] \* step, self.Rw[2][1] \* step, self.Rw[2][2] \* step,  
 0, 0, 1]])  
  
 self.H[0][0] = 1 + self.g \* np.sin(self.teta)  
 self.H[1][1] = 1 - self.g \* np.cos(self.teta) \* np.sin(self.fi)  
 self.H[2][2] = 1 - self.g \* np.cos(self.teta) \* np.cos(self.fi)  
  
 self.Dinamic = self.F.dot(self.Dinamic)  
 self.P = (self.F.dot(self.P)).dot(self.F.transpose()) + self.Q  
 self.K = (self.P.dot(self.H.transpose())).dot(  
 np.linalg.inv(((self.H.dot(self.P)).dot(self.H.transpose())) + self.R))  
 self.Dinamic = self.Dinamic + (self.K.dot((self.input - self.H.dot(self.Dinamic))))  
 self.P = (np.identity(15) - self.K.dot(self.H)).dot(self.P)  
  
 **return** self.Dinamic  
  
 **def** reset\_integrirovanie(self):  
 self.Dinamic[14][0] = 0  
  
  
**class** Regulator():  
 **def** \_\_init\_\_(self, kp, ki, kd, vmin, umax):  
 self.kp = kp  
 self.ki = ki  
 self.kd = kd  
 self.vmin = vmin  
 self.umax = umax  
 self.sig = np.zeros(int(Time / step))  
 self.error = np.zeros(int(Time / step))  
 self.error\_sig = np.zeros(int(Time / step))  
 self.ui\_past = 0  
 self.u = np.zeros(int(Time / step))  
  
 **def** speed\_limit(self, i, wish, have):  
 self.err\_v = wish - self.sig[i - 1]  
 **if** abs(self.err\_v) > self.vmin \* step:  
 self.sig[i] = self.sig[i - 1] + self.vmin \* step \* np.sign(self.err\_v)  
 **else**:  
 self.sig[i] = self.sig[i - 1] + self.err\_v  
 self.error\_sig[i] = self.sig[i] - have  
 **return** self.error\_sig[i]  
  
 **def** pid(self, i, err, moment):  
 self.error[i] = err + moment  
 up = self.kp \* self.error[i]  
 *# Ограничение интегральной составляющей* **if** self.u[i - 1] < self.umax:  
 ui = self.ui\_past + self.ki \* step \* self.error[i]  
 **else**:  
 ui = self.ui\_past  
 ud = self.kd \* (self.error[i] - self.error[i - 1]) / step  
 self.u[i] = up + ui + ud  
 self.ui\_past = ui  
 **return** self.u[i]  
  
  
client = RemoteAPIClient()  
sim = client.getObject(**'sim'**)  
defaultIdleFps = sim.getInt32Param(sim.intparam\_idle\_fps)  
sim.setInt32Param(sim.intparam\_idle\_fps, 0)  
rob\_base = sim.getObject(**'/Copter'**)  
motor\_left = sim.getObject(**'/M\_L'**)  
motor\_Back = sim.getObject(**'/M\_B'**)  
motor\_Front = sim.getObject(**'/M\_F'**)  
motor\_Right = sim.getObject(**'/M\_R'**)  
dist\_laser = sim.getObject(**'/laser'**)  
barrier\_sensor\_left = sim.getObject(**'/pr\_sensor\_l'**)  
barrier\_sensor\_front = sim.getObject(**'/pr\_sensor\_f'**)  
barrier\_sensor\_right = sim.getObject(**'/pr\_sensor\_r'**)  
stop\_left = sim.getObject(**'/pr\_l\_stop'**)  
stop\_right = sim.getObject(**'/pr\_r\_stop'**)  
visionSensorHandle = sim.getObject(**'/Vision\_sensor'**)  
  
Time, step = 100000, 0.05  
App = Simulation\_App()  
Copter = Air\_Robot(0.5, 0.2, 0.05, 0.091, 7.5 \* 10 \*\* -4, 1.2 \* 10 \*\* -5)  
Route = Trajectory()  
Data = Calman\_Filrter()  
Cam = Camera()  
  
U1 = Regulator(0.66, 0, 4, 10, 0)  
Ang\_fi = Regulator(0.193, 0, 0.115, 0, 0)  
Ang\_teta = Regulator(0.193, 0, 0.115, 0, 0)  
Mx\_y = Regulator(0.031, 0, 0.101, 20, 0)  
My\_x = Regulator(0.031, 0, 0.101, 20, 0)  
U4 = Regulator(0.92, 0, 1.5, 0, 0)  
  
start\_move = **False**start\_rotation = **False**Spline\_Done = **False**mission\_1\_to\_point = **False**mission\_2\_forward = **False**mission\_3\_spline = **False**mission\_4\_saerch = **False**finish = **False**get\_param = **False**x\_points = []  
y\_points = []  
flight\_height = []  
flight\_speed = []  
  
i = 0  
j = 0  
do\_serch = -1  
count\_spline = -1  
stab\_rotation = 0  
down = 0  
end = 0  
x\_for\_spline = []  
y\_for\_spline = []  
x\_for\_spline.append(0)  
y\_for\_spline.append(0)  
  
b\_l, b\_f, b\_r = 0, 0, 0  
u, v, w, x, y, z, p, q, r = 0, 0, 0, 0, 0, 0, 0, 0, 0  
Mx, My = 0, 0  
speed\_limit\_Mx = 0  
speed\_limit\_My = 0  
speed\_limit\_rotation = 0.07  
err\_teta\_x, err\_fi\_y = 0, 0  
v\_coeff = 1  
v\_for\_dif\_mission = 1  
div = 0  
u\_1 = np.zeros(int(Time / step))  
u\_2 = np.zeros(int(Time / step))  
u\_3 = np.zeros(int(Time / step))  
u\_4 = np.zeros(int(Time / step))  
  
pos\_x = np.zeros(int(Time / step))  
pos\_y = np.zeros(int(Time / step))  
pos\_z = np.zeros(int(Time / step))  
fi = np.zeros(int(Time / step))  
teta = np.zeros(int(Time / step))  
rob\_x = np.zeros(int(Time / step))  
rob\_y = np.zeros(int(Time / step))  
course = np.zeros(int(Time / step))  
psi = np.zeros(int(Time / step))  
veloc\_xy = np.zeros(int(Time / step))  
  
trajectory\_x = np.zeros(int(Time / step))  
trajectory\_y = np.zeros(int(Time / step))  
trajectory\_z = np.zeros(int(Time / step))  
start\_now = 0  
  
cubes = np.zeros((10, 2))  
count\_cube = 0  
found\_cube = **False**repeat = **False**curse\_reset = **False**timer = np.zeros(int(Time / step))  
pos\_x\_graph = []  
pos\_y\_graph = []  
tr\_x\_graph = []  
tr\_y\_graph = []  
speed\_graph = []  
h\_graph = []  
course\_graph = []  
time\_graph = []  
  
client.setStepping(**True**)  
print(**"start"**)  
state = sim.getSimulationState()  
  
**while** state == 0:  
 App.controll()  
 state = sim.getSimulationState()  
  
**while** state != 0:  
 **if** get\_param == **False**:  
 x\_points, y\_points, flight\_height, flight\_speed = App.parameters()  
 mission\_1\_to\_point, mission\_2\_forward, mission\_3\_spline, mission\_4\_saerch = App.flags()  
 get\_param = **True** App.controll()  
  
 x, y, z = Copter.get\_pos()  
 u, v, w = Copter.get\_acceleration()  
 p, q, r = Copter.get\_angular\_vel()  
 result = Data.solve\_EFC(u, v, w, x, y, z, p, q, r)  
 pos\_x[i], pos\_y[i], pos\_z[i] = x, y, z  
 fi[i], teta[i], psi[i] = result[12][0], result[13][0], result[14][0]  
 veloc\_xy[i] = np.sqrt(result[3][0] \*\* 2 + result[4][0] \*\* 2)  
 KS = 0.1\*flight\_speed[0]  
 K = 4  
  
 **if** (start\_move == **True and** mission\_1\_to\_point == **True**):  
 v\_for\_dif\_mission = 10  
 count\_point = len(x\_points) - len(x\_points) + j  
 trajectory\_x[i], trajectory\_y[i], trajectory\_z[i] = Route.uniform\_motion(i, x\_points[count\_point],y\_points[count\_point],flight\_height[0],flight\_speed[0] \* v\_coeff, start\_now,pos\_x[i], pos\_y[i])  
 start\_now = 1  
 course[i] = Copter.rotation\_controll(pos\_x[i - 1], pos\_y[i - 1], pos\_x[i], pos\_y[i], psi[i])  
 **if** round(pos\_x[i - 1], 1) == x\_points[count\_point] **and** round(pos\_y[i - 1], 1) == y\_points[count\_point] **and** len(x\_points) - j > 1:  
 j = j + 1  
 start\_now = 0  
 **if** abs(x\_points[count\_point] - pos\_x[i]) <= 2:  
 course[i] = Copter.rotation\_controll(pos\_x[i], pos\_y[i],x\_points[count\_point] + 0.5 \* x\_points[count\_point] \* np.sign(x\_points[count\_point]),y\_points[count\_point] + 0.5 \* y\_points[count\_point] \* np.sign(y\_points[count\_point]), psi[i])  
 **if** abs(x\_points[count\_point] - pos\_x[i]) <= 0.5 **and** abs(  
 y\_points[count\_point] - pos\_y[i]) <= 0.5 **and** finish == **False and** mission\_4\_saerch == **False**:  
 finish = **True  
 if** finish == **True**:  
 App.finish\_mission()  
  
 **if** (start\_move == **True and** mission\_2\_forward == **True**):  
 v\_for\_dif\_mission = 10  
 count\_point = len(x\_points) - len(x\_points) + j  
 trajectory\_x[i], trajectory\_y[i], trajectory\_z[i] = Route.uniform\_motion(i, x\_points[count\_point],y\_points[count\_point],flight\_height[0], flight\_speed[0]\*v\_coeff, start\_now, pos\_x[i], pos\_y[i])  
 start\_now = 1  
 course[i] = Copter.rotation\_controll(pos\_x[i-1], pos\_y[i-1], pos\_x[i], pos\_y[i], psi[i])  
 **if** round(pos\_x[i - 1], 1) == x\_points[count\_point] **and** round(pos\_y[i - 1], 1) == y\_points[count\_point] **and** len(x\_points) - j > 1:  
 j = j + 1  
 start\_now = 0  
 **if** abs(x\_points[count\_point] - pos\_x[i]) <= 2:  
 course[i] = Copter.rotation\_controll(pos\_x[i], pos\_y[i], x\_points[count\_point] + 0.5\*x\_points[count\_point]\*np.sign(x\_points[count\_point]), y\_points[count\_point] + 0.5\*y\_points[count\_point]\*np.sign(y\_points[count\_point]), psi[i])  
 **if** abs(x\_points[count\_point] - pos\_x[i]) <= 0.5 **and** abs(y\_points[count\_point] - pos\_y[i]) <= 0.5 **and** finish == **False and** mission\_4\_saerch == **False**:  
 finish = **True  
 if** finish == **True**:  
 App.finish\_mission()  
  
 **if** (start\_move == **True and** mission\_3\_spline == **True**):  
 v\_for\_dif\_mission = 2  
 **if** count\_spline == -1:  
 x\_for\_spline[0], y\_for\_spline[0] = pos\_x[1], pos\_y[1]  
 **for** a **in** range(len(x\_points)):  
 x\_for\_spline.append(x\_points[a])  
 y\_for\_spline.append(y\_points[a])  
 count\_spline = count\_spline + 1  
 **else**:  
 trajectory\_x[i], trajectory\_y[i] = Route.speed\_cubic\_spline(count\_spline, x\_for\_spline, y\_for\_spline,flight\_speed[0]\*v\_coeff)  
 course[i] = Copter.rotation\_controll(pos\_x[i-1], pos\_y[i-1], pos\_x[i], pos\_y[i], psi[i])  
 trajectory\_z[i] = flight\_height[0]  
 count\_spline = count\_spline + 1  
  
 **if** abs(x\_for\_spline[len(x\_for\_spline)-1] - pos\_x[i]) <= 2:  
 course[i] = Copter.rotation\_controll(pos\_x[i], pos\_y[i],x\_for\_spline[len(x\_for\_spline)-1] + 0.2\*x\_for\_spline[len(x\_for\_spline)-1]\*np.sign(x\_for\_spline[len(x\_for\_spline)-1]), y\_for\_spline[len(y\_for\_spline)-1] + 0.2\*y\_for\_spline[len(y\_for\_spline)-1]\*np.sign(y\_for\_spline[len(y\_for\_spline)-1]), psi[i])  
 **if** abs(x\_for\_spline[len(x\_for\_spline)-1] - pos\_x[i]) <= 0.5 **and** abs(y\_for\_spline[len(y\_for\_spline)-1] - pos\_y[i]) <= 0.5 **and** finish == **False**:  
 finish = **True  
 if** finish == **True**:  
 App.finish\_mission()  
  
 **if** (start\_move == **True and** mission\_4\_saerch == **True**):  
 v\_for\_dif\_mission = 10000  
 **if** do\_serch < 17:  
 trajectory\_x[i], trajectory\_y[i], trajectory\_z[i] = Route.uniform\_motion(i, 0.001, 0, flight\_height[0], flight\_speed[1], start\_now, pos\_x[i], pos\_y[i] )  
 start\_now = 1  
 course[i] = Copter.rotation\_controll(pos\_x[i-1], pos\_y[i-1], pos\_x[i], pos\_y[i], psi[i])  
 **if** abs(round(pos\_x[i - 1], 1)) <= 0.5 **and** abs(round(pos\_y[i - 1], 1)) <= 0.5:  
 course[i] = course[i-1]  
 **if** (abs(course[i]) <= 0.05):  
 do\_serch = do\_serch + 1  
 **else**:  
 **if** psi[i] >= 6.28:  
 Data.reset\_integrirovanie()  
 test = 0  
 repeat = **False** camX, camY, cube\_x, cube\_y = Cam.detect(pos\_x[i], pos\_y[i], pos\_z[i])  
 trajectory\_x[i], trajectory\_y[i] = Route.spiral(flight\_speed[0]\*v\_coeff)  
 course[i] = Copter.rotation\_controll(pos\_x[i-1], pos\_y[i-1], pos\_x[i], pos\_y[i], psi[i])  
 trajectory\_z[i] = flight\_height[0]  
 do\_serch = do\_serch + 1  
  
 **if** camY > 0:  
 cubes[count\_cube][0] = cube\_x  
 cubes[count\_cube][1] = cube\_y  
 found\_cube = **True  
 if** camY < 0 **and** found\_cube == **True**:  
 print(**"ухожу"**)  
 **if** count\_cube >= 1:  
 **for** test **in** range (count\_cube):  
 print(test)  
 **if** abs(abs(cubes[test][0]) - abs(cubes[count\_cube][0])) < 9.5 **and** abs(abs(cubes[test][1]) - abs(cubes[count\_cube][1])) < 9.5:  
 cubes[test][0] = (cubes[test][0] + cubes[count\_cube][0]) / 2  
 cubes[test][1] = (cubes[test][1] + cubes[count\_cube][1]) / 2  
 repeat = **True** cubes[count\_cube][0] = 0  
 cubes[count\_cube][1] = 0  
 print(**"повтор"**)  
 **else**:  
 print (**"не повтор"**)  
 **if** repeat == **True**:  
 count\_cube = count\_cube - 1  
 found\_cube = **False** count\_cube = count\_cube + 1  
 **if** count\_cube == 0:  
 count\_cube = 1  
 found\_cube = **False** print(**"нашел"**,count\_cube)  
 print(cubes)  
 **if** count\_cube == 10:  
 App.finish\_mission()  
 count\_cube = 11  
  
 **if** (start\_move == **True**):  
 **if** trajectory\_x[i] == 0 **and** trajectory\_y[i] == 0 **and** trajectory\_z[i] == 0:  
 trajectory\_x[i], trajectory\_y[i], trajectory\_z[i] = trajectory\_x[i-1], trajectory\_y[i-1], trajectory\_z[i-1]  
 course[i] = course[i-1]  
 **if** pos\_z[i-1] - pos\_z[i] > 1.5:  
 pos\_z[i] = pos\_z[i-1]  
  
 err\_teta\_x, err\_fi\_y, x\_way\_c, y\_way\_c, x\_c, y\_c = Copter.horizontal\_controll(rob\_base, trajectory\_x[i - 2], trajectory\_y[i - 2], pos\_x[i],pos\_y[i], psi[i])  
 b\_l, b\_f, b\_r, b\_l\_dist, b\_f\_dist, b\_r\_dist, s\_l, s\_r, s\_l\_dist, s\_r\_dist = Copter.barriers(barrier\_sensor\_left, barrier\_sensor\_front,barrier\_sensor\_right,stop\_left, stop\_right)  
 x\_way\_c, y\_way\_c = Route.detour\_start(x\_way\_c, y\_way\_c, flight\_speed[0])  
 v\_coeff = 1  
  
 **if** veloc\_xy[i] > flight\_speed[0] + 0.034\*flight\_speed[0]:  
 v\_coeff = 0.9  
 x\_way\_c, y\_way\_c = Route.detour\_start(x\_way\_c, y\_way\_c, 0.4\*flight\_speed[0])  
  
 **if** s\_r == 1 **and** b\_f == 0:  
 v\_coeff = v\_for\_dif\_mission  
 x\_way\_c, y\_way\_c = Route.detour\_start(-(KS \* flight\_speed[0])/s\_r\_dist, 4, 0.005\*flight\_speed[0])  
  
 **if** s\_l == 1 **and** b\_f == 0:  
 v\_coeff = v\_for\_dif\_mission  
 x\_way\_c, y\_way\_c = Route.detour\_start(4, -(KS \* flight\_speed[0]) / s\_l\_dist, 0.005\*flight\_speed[0])  
  
 **if** b\_f == 1:  
 v\_coeff = v\_for\_dif\_mission  
 **if** b\_r == 1 **and** b\_l == 0:  
 x\_way\_c, y\_way\_c = Route.detour\_start(-(K\*flight\_speed[0])/b\_f\_dist, 4, flight\_speed[0])  
 **elif** b\_r == 0 **and** b\_l == 1:  
 x\_way\_c, y\_way\_c = Route.detour\_start(4, -(K \* flight\_speed[0]) / b\_f\_dist, flight\_speed[0])  
 **else**:  
 x\_way\_c, y\_way\_c = Route.detour\_start(4, -(2\*K \* flight\_speed[0]) / b\_f\_dist, flight\_speed[0])  
  
 Mx = Mx\_y.pid(i, Mx\_y.speed\_limit(i, x\_way\_c, x\_c), 0)  
 My = My\_x.pid(i, My\_x.speed\_limit(i, y\_way\_c, y\_c), 0)  
  
  
 **if** (start\_move == **False**):  
 trajectory\_x[i], trajectory\_y[i], trajectory\_z[i] = Route.forward\_straight(i, pos\_x[1], pos\_y[1], 1)  
 course[i] = Copter.rotation\_controll(pos\_x[i], pos\_y[i], x\_points[0], y\_points[0], psi[i])  
 err\_teta\_x, err\_fi\_y, x\_way\_c, y\_way\_c, x\_c, y\_c = Copter.horizontal\_controll(rob\_base, trajectory\_x[i], trajectory\_y[i], pos\_x[i],pos\_y[i], psi[i])  
 **if** i == 0:  
 Mx = 0  
 My = 0  
 **else**:  
 Mx = Mx\_y.pid(i, Mx\_y.speed\_limit(i, x\_way\_c, x\_c), 0)  
 My = My\_x.pid(i, My\_x.speed\_limit(i, y\_way\_c, y\_c), 0)  
 speed\_limit\_Mx = 0.0001  
 speed\_limit\_My = 0.0001  
 **if** pos\_z[i] >= 0.9:  
 start\_rotation = **True** print(**"rot"**)  
  
 **if** (start\_rotation == **True and** start\_move == **False**):  
 course[i] = Copter.rotation\_controll(pos\_x[i], pos\_y[i], x\_points[0], y\_points[0], psi[i])  
 speed\_limit\_Mx = 0.005  
 speed\_limit\_My = 0.005  
 **if** (abs(course[i]) <= 0.02):  
 stab\_rotation = stab\_rotation + 1  
 **if** stab\_rotation == 15:  
 start\_move = **True** print(**"move"**)  
  
 u\_1[i] = U1.pid(i, U1.speed\_limit(i, trajectory\_z[i], pos\_z[i]), 0)  
 u\_2[i] = Ang\_fi.pid(i, err\_fi\_y, My) *# My* u\_3[i] = Ang\_teta.pid(i, err\_teta\_x, Mx) *# Mx* u\_4[i] = U4.pid(i, course[i], 0)  
  
 **if** abs(u\_2[i]) > speed\_limit\_Mx:  
 u\_2[i] = speed\_limit\_Mx \* np.sign(u\_2[i])  
  
 **if** abs(u\_3[i]) > speed\_limit\_My:  
 u\_3[i] = speed\_limit\_My \* np.sign(u\_3[i])  
  
 **if** abs(u\_4[i]) > speed\_limit\_rotation:  
 u\_4[i] = speed\_limit\_rotation \* np.sign(u\_4[i])  
  
 w1, w2, w3, w4 = Copter.velocities\_calculator(u\_1[i], -u\_2[i], -u\_3[i], u\_4[i]) *# -u\_2[i], -u\_3[i], u\_4[i]* Copter.move(motor\_Back, motor\_Right, motor\_Front, motor\_left, w1, w2, w3, w4)  
  
 **if** App.end == 1:  
 **if** mission\_4\_saerch == **True**:  
 App.show\_cubes(cubes[0][0], cubes[0][1], cubes[1][0], cubes[1][1], cubes[2][0], cubes[2][1], cubes[3][0], cubes[3][1], cubes[4][0], cubes[4][1],  
 cubes[5][0], cubes[5][1], cubes[6][0], cubes[6][1], cubes[7][0], cubes[7][1], cubes[8][0], cubes[8][1], cubes[9][0], cubes[9][1])  
 plt.figure(figsize=(14, 10))  
 plt.suptitle(**"Motion characteristics"**)  
 plt.subplot(2, 2, 1)  
 plt.title(**"trajectiry and robot position"**)  
 plt.xlabel(**"X"**)  
 plt.ylabel(**"Y"**)  
 plt.plot(pos\_x\_graph, pos\_y\_graph, color=**'green'**)  
 plt.plot(tr\_x\_graph, tr\_y\_graph, **'r--'**)  
 plt.subplot(2, 2, 2)  
 plt.title(**"height"**)  
 plt.xlabel(**"time"**)  
 plt.ylabel(**"height"**)  
 plt.plot(time\_graph, h\_graph, color=**'black'**)  
 plt.subplot(2, 2, 3)  
 plt.title(**"speed"**)  
 plt.xlabel(**"time"**)  
 plt.ylabel(**"speed"**)  
 plt.plot(time\_graph, speed\_graph, color=**'black'**)  
 plt.subplot(2, 2, 4)  
 plt.title(**"course error"**)  
 plt.xlabel(**"time"**)  
 plt.ylabel(**"error"**)  
 plt.plot(time\_graph, course\_graph, color=**'black'**, label=**'course\_error'**)  
 plt.show()  
  
 **if** i>5:  
 pos\_x\_graph.append(pos\_x[i])  
 pos\_y\_graph.append(pos\_y[i])  
 tr\_x\_graph.append(trajectory\_x[i])  
 tr\_y\_graph.append(trajectory\_y[i])  
 **if** veloc\_xy[i] > flight\_speed[0] **and** i < 40:  
 speed\_graph.append(0)  
 **else**:  
 speed\_graph.append(veloc\_xy[i])  
 h\_graph.append(pos\_z[i])  
 course\_graph.append(course[i])  
 time\_graph.append(timer[i-1])  
  
 timer[i] = i \* step  
 i = i + 1  
 client.step()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

# РЕЗУЛЬТАТЫ РАБОТЫ

Далее проведем различные испытания для определения качества работы ИУС.

## Испытание 1

На тестовом полигоне зададим следующие параметры режима работы, следующие для проверки движения робота по прямолинейно траектории:

* Заданная скорость: ;
* Заданная высота: ;
* Координаты точки X: 80 (движение осуществляется с 0)
* Координаты точки Y: 60 (движение осуществляется с 0)
* Режим движения: движение в точку с заданной скоростью/ прямолинейное движение с заданной скоростью;

Результаты работы представлены на рисунке 10.

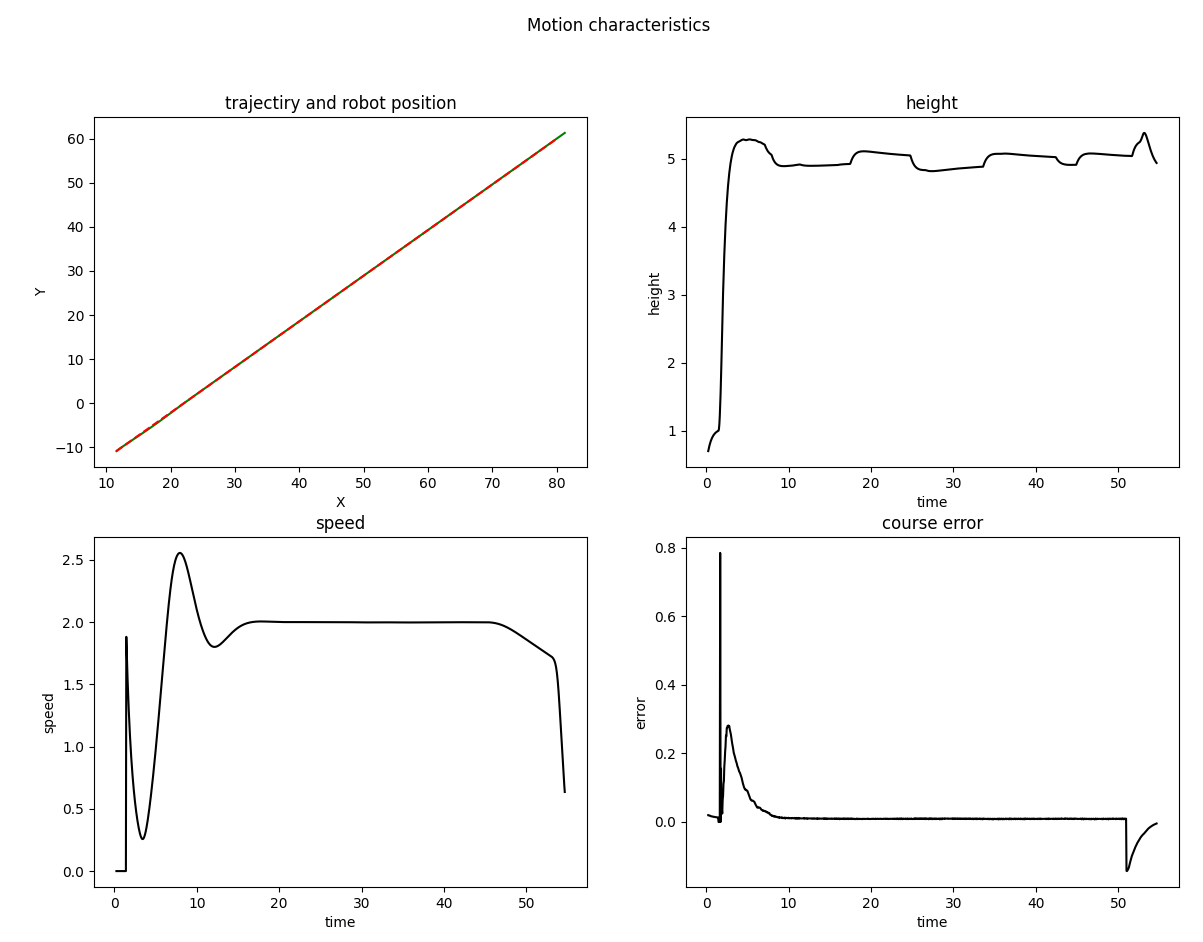


Рисунок 10 – Характеристики движения

Как видно из рисунка 10 робот полностью следует траектории, однако наблюдается некоторое перерегулирование (график в левом верхнем углу), среднее значение высоты соответствует заданному (график в правом верхнем углу), а скачки обоснованы неровным рельефом и постепенным набором, и сбросом высоты. Скорость на протяжении движения остается постоянной за исключением случае разгона и торможения (график в левом нижнем углу). Ошибка по курсу (график в правом нижнем углу) на протяжении движения по траектории остается нулевой.

## Испытание 2

На тестовом полигоне зададим следующие параметры режима работы, следующие для проверки режима движения по криволинейной траектории:

* Заданная скорость: ;
* Заданная высота: ;
* Координаты точки X: 20 50 80 100 (движение осуществляется с 0)
* Координаты точки Y: 10 -2 0 9 (движение осуществляется с 0)
* Режим движения: движение по кубическому сплайну с заданной скоростью;

Результаты работы представлены на рисунке 11.

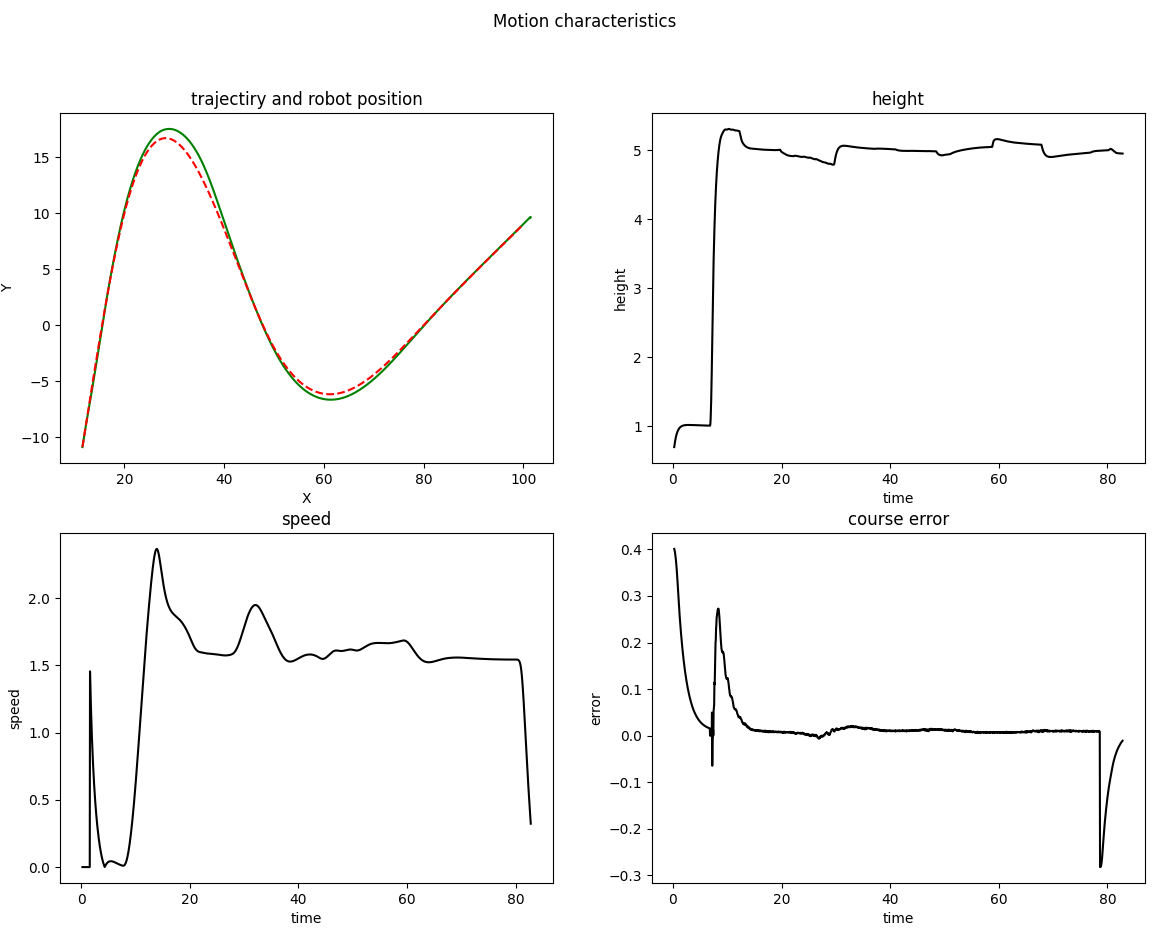


Рисунок 11 – Характеристики движения

Как видно из рисунка 12 робот полностью следует траектории, однако наблюдается влияние кориолисовых сил на поворотах (график в левом верхнем углу), среднее значение высоты соответствует заданному (график в правом верхнем углу), а скачки обоснованы неровным рельефом и постепенным набором, и сбросом высоты. Скорость на протяжении движения изменяется в районе заданной, что обосновано криволинейной траекторией, также наблюдается сильное превышение скорости на моменте разгона (график в левом нижнем углу). Ошибка по курсу (график в правом нижнем углу) на протяжении движения по траектории остается близкой к нулю.

## Испытание 3

В ходе третьего испытания проведем исследование обхода препятствий роботом. Для этого зададим следующие параметры режима работы, следующие для проверки режима движения по криволинейной траектории:

* Заданная скорость: ;
* Заданная высота: ;
* Координаты точки X: 200 (движение осуществляется с 0)
* Координаты точки Y: 200 (движение осуществляется с 0)
* Режим движения: движение по прямой заданной скоростью;

Обход препятствия изображен на рисунке 12, а характеристики движения изображены на рисунке 13.

Размер препятствия 9х9 метров.

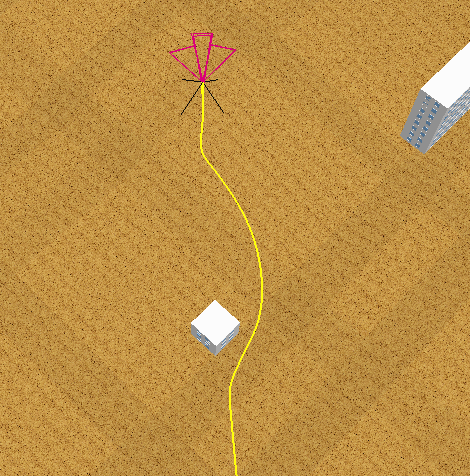


Рисунок 12 – объезд препятствия

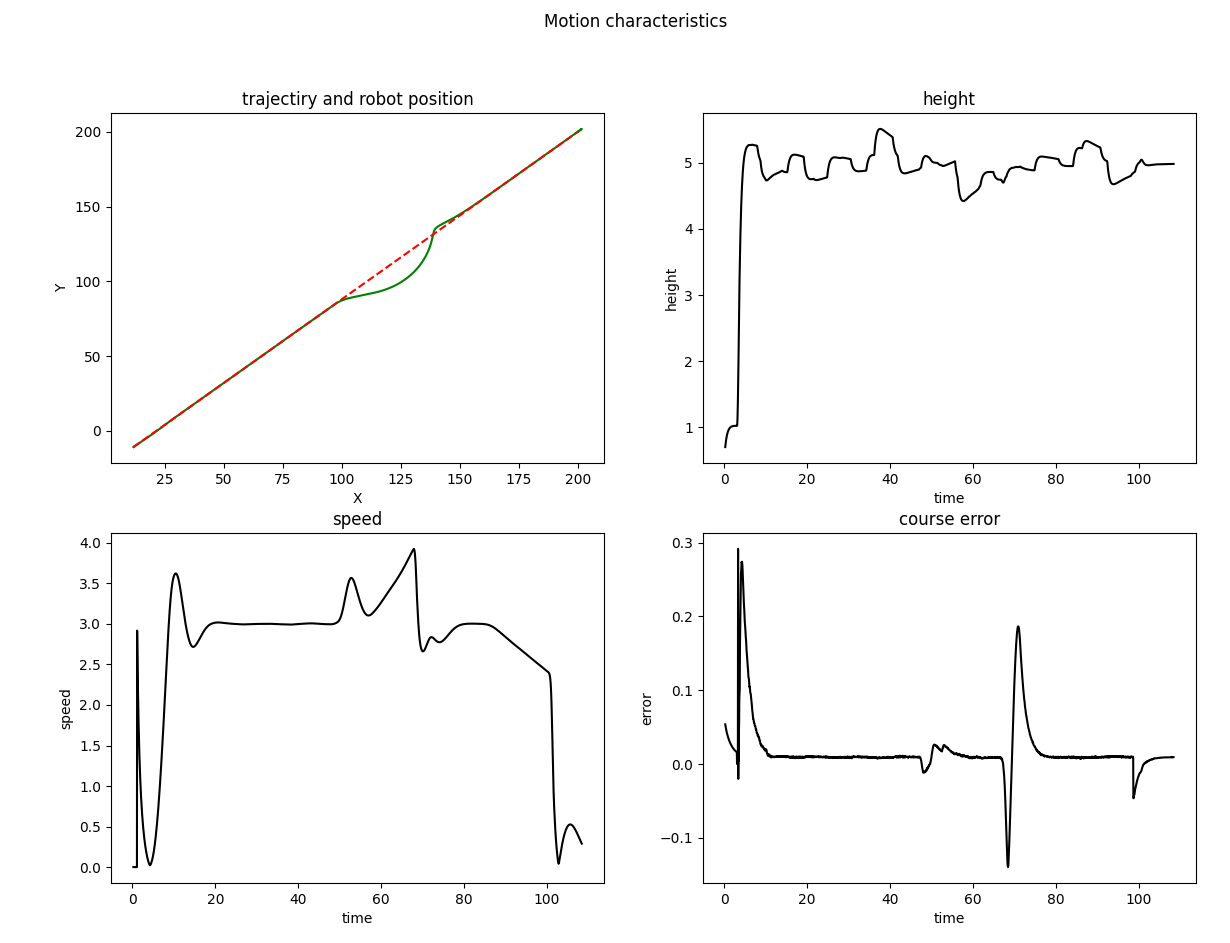


Рисунок 13 – Характеристики движения

Как видно из рисунка 13 робот возвращается на траекторию однако не быстро (график в левом верхнем углу), среднее значение высоты соответствует заданному (график в правом верхнем углу), а скачки обоснованы неровным рельефом и постепенным набором, и сбросом высоты. Скорость до препятствия соответствует заданной, во время столкновения резко возрастает, после чего с колебаниями возвращается к заданному значению (график в левом нижнем углу). Ошибка по курсу (график в правом нижнем углу) на протяжении движения по траектории остается близкой к нулю, однако на момент обнаружения препятствия резко увеличивается, что обосновано перестроением траектории движения для обхода.

## Испытание 5

В ходе пятого испытания проведем проверку движения на одной высоте над поверхностью, для направим робота на сопку, по которой он должен будет подняться, установим параметры режима работы.

* Заданная скорость: ;
* Заданная глубина: ;
* Координаты точки X: 45 (движение осуществляется с 0)
* Координаты точки Y: 45 (движение осуществляется с 0)
* Режим движения: движение с заданной скоростью.

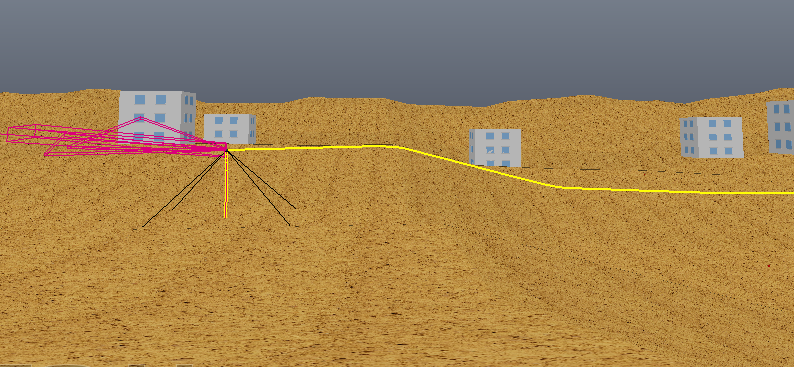


Рисунок 14 – визуальная оценка движения на заданной высоте

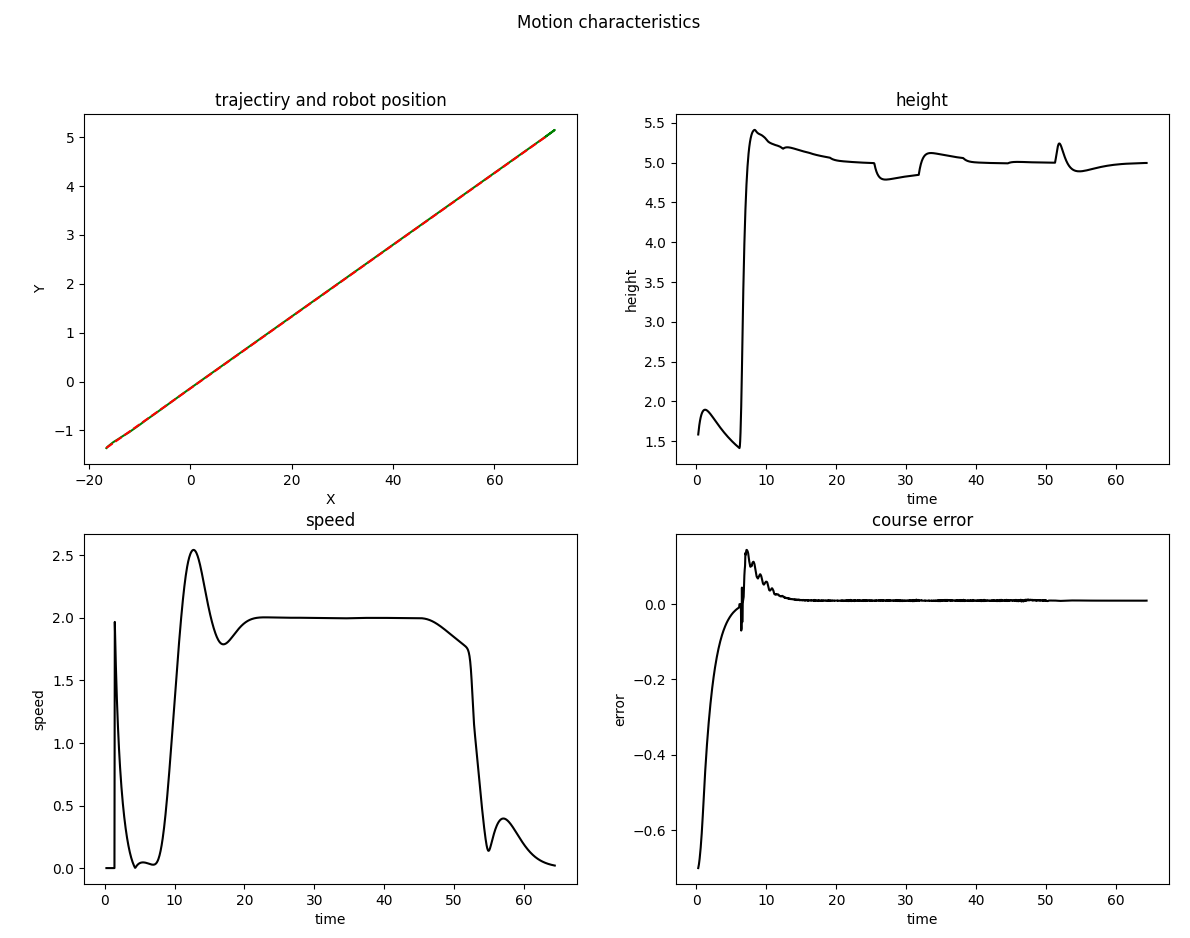


Рисунок 15 – Характеристики движения

Как видно из рисунка 15 среднее значение высоты соответствует заданному (график в правом верхнем углу), однако при начале подъема происходит потеря высоты, после чего она выравнивается по преодолении небольшого перерегулирования.

## Испытание 6

В ходе шестого испытания проведем исследование режима поиска объектов.

Установим параметры режима работы.

* Заданная скорость: ;
* Заданная глубина: ;
* Координаты точки X: 0 (движение осуществляется с 0)
* Координаты точки Y: 0 (движение осуществляется с 0)
* Режим движения: поиск.

Результаты работы приведены в таблице 3.

В данном режиме обнаружение объектов производится с использованием бортовой камеры, изображение с камеры выводится в отдельном окне (см. Рис. 16)

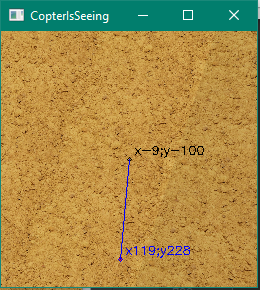


Рисунок 16 – Результат обнаружения объекта

На изображении 16 черным цветом обозначен центр камеры, синим расстояние до найденного объекта в пикселях.

На исследования рабочего поля с максимальной дистанцией до кубика (-150, 10) ушло 47 минут 33 секунды.

Результат работы программы представлен на рисунке 17. А характеристики движения на рисунке 18.

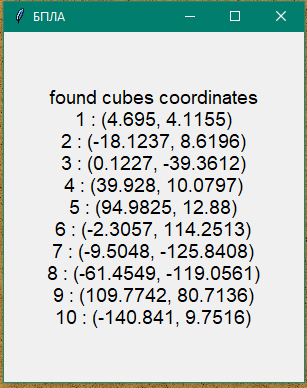


Рисунок 17 – результат работы программы

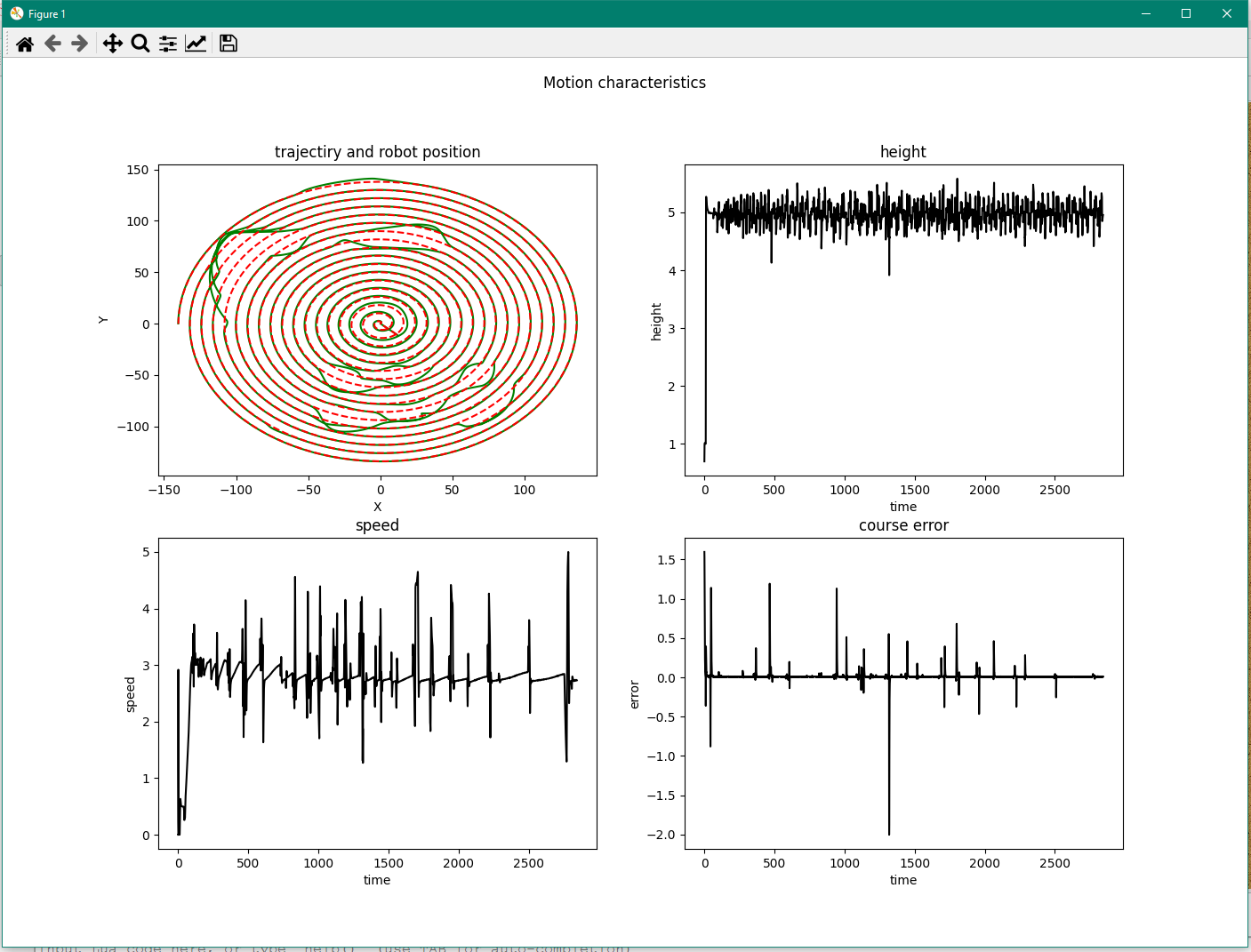


Рисунок 18 – характеристики движения

Как видно из рисунка 18 робот во время движения в силу обхода препятствий появляются слепые зоны (график в левом верхнем углу), среднее значение высоты соответствует заданному (график в правом верхнем углу), а скачки обоснованы неровным рельефом и постепенным набором, и сбросом высоты. Скорость движения постоянно изменяется, что обосновано ее постоянном набором и сбросом для объезда (график в левом нижнем углу). Ошибка по курсу (график в правом нижнем углу) на протяжении движения по траектории остается близкой к нулю, однако на момент обнаружения препятствия резко увеличивается, что обосновано перестроением траектории движения для обхода.

Таблица 3 – Результаты поиска объекта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | расчётное | расчётное | реальное | реальное |
| 1 | 4.65 | 4.11 | 5.2 | 4.7 |
| 2 | -18.1 | 8.6 | -18.5 | 9.2 |
| 3 | 0.1 | -39.36 | 0.5 | -41.28 |
| 4 | 39.9 | 10 | 39.5 | 10.7 |
| 5 | 94.98 | 12.88 | 95.3 | 14.8 |
| 6 | -2.3 | 114.2 | -5.7 | 115.6 |
| 7 | -9.5 | -125.8 | -10.58 | -127.1 |
| 8 | -61.45 | -119.05 | -62.2 | -121.01 |
| 9 | 109.7 | 80.7 | 110.6 | 81.2 |
| 10 | -140.8 | 9.7 | -141.1 | 10.6 |

По представленным результатам в таблице 3 видно, что максимальная ошибка по координатам составляет 3.4 метра. Данные значения являются допустимыми.

# ВЫВОД

В ходе выполнения курсовой работы были получены навыки создания автоматизированной информационно-управляющей системы для подводного робота, управляемого по всем степеням свободы с функцией поиска объектов, расположенных в заранее неизвестных местах. Была разработана система управления подводным роботом на языке программирования Python с использованием распределенной архитектуры. Также был разработан пользовательский интерфейс позволяющий отслеживать выполнение миссии, задавать параметры робота и миссию, которую необходимо выполнить.

Разработана следующая документация: техническое задание, описание программы, руководство системного программиста, руководство оператора и текст программы.

Было произведено тестирование полученной информационно-управляющей системы, в ходе которой она успешно справилась с поставленными задачами.