Полуавтоматическая турель «Lutic-S»

Инструкция по программированию

Содержание

[**Java** 2](#_Toc195013828)

[**1. Обзор проекта** 2](#_Toc195013829)

[**2. Настройка и запуск** 3](#_Toc195013830)

[**2.1 Запуск робота** 3](#_Toc195013831)

[**2.2 Режимы работы:** 3](#_Toc195013832)

[**3. Работа с подсистемами** 3](#_Toc195013833)

[**3.1 AutoTrain:** 3](#_Toc195013834)

[**3.2 OMS:** 4](#_Toc195013835)

[**3.3 UDP:** 4](#_Toc195013836)

[**4. Управление с геймпада** 4](#_Toc195013837)

[**5. Автономный режим** 4](#_Toc195013838)

[**6. Телеуправление** 4](#_Toc195013839)

[**6.1 Teleop.java:** 4](#_Toc195013840)

[**6.2 TeleopOMS.java:** 5](#_Toc195013841)

[**7. UDPCommands.java** 5](#_Toc195013842)

[**8. SimpleAuto.java** 6](#_Toc195013843)

[**9. Auto.java** 6](#_Toc195013844)

[**10. AutoCommand.java** 7](#_Toc195013845)

[**11. Shuffleboard** 7](#_Toc195013846)

[**12. Советы и рекомендации.** 7](#_Toc195013847)

[**Python** 9](#_Toc195013848)

[**13. Назначение** 9](#_Toc195013849)

[**14. Зависимости** 9](#_Toc195013850)

[**15. Ключевые компоненты** 9](#_Toc195013851)

[**15.1 YOLO-модель:** 9](#_Toc195013852)

[**15.2 UDP-клиент:** 9](#_Toc195013853)

[**15.3 Обработка видео:** 9](#_Toc195013854)

[**15.4 Многопоточность:** 9](#_Toc195013855)

[**16. Настройка** 10](#_Toc195013856)

[**16.1 Модель YOLO:** 10](#_Toc195013857)

[**16.2 UDP-сервер:** 10](#_Toc195013858)

[**16.3 Источник видео:** 10](#_Toc195013859)

[**17. Пример использования** 10](#_Toc195013860)

[**18. Важные замечания** 10](#_Toc195013861)

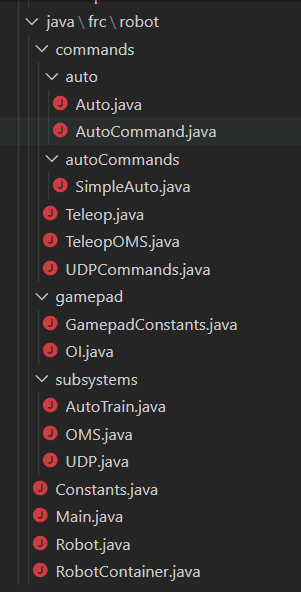
[**19. Дополнительные возможности** 10](#_Toc195013862)

# 

# **Java**

## **1. Обзор проекта**

Проект состоит из 15 java файлов.



**Main.java**: Точка входа программы, запускает турель.

**Robot.java**: Главный класс турели, управляет режимами (автономный, телеуправление и т.д.).

**RobotContainer.java**: Контейнер для подсистем, команд и настройки кнопок.

**Constants.java**: Хранит константы (ID двигателей, параметры энкодеров и т.д.).

**UDP.java**: Подсистема для работы с UDP-сервером.

**OMS.java**: Подсистема для управления механизмами.

**AutoTrain.java**: Подсистема для управления движением и датчиками.

**OI.java**: Управление вводом с геймпада.

**GamepadConstants.java**: Константы для кнопок геймпада.

**Teleop.java**: Команда для управления турели в режиме телеуправления.

**TeleopOMS.java**: Команда для управления подсистемой OMS в режиме телеуправления, с обработкой ввода с геймпада.

**UDPCommands.java**: Команды для обработки UDP-пакетов и управления подсистемой на основе полученных данных.

**SimpleAuto.java**: Базовая автономная команда для тестирования механизмов, содержащая простые последовательности действий.

**Auto.java**: Конкретная реализация автономной команды, использующая **SimpleAuto** с заданными параметрами для выполнения стандартного автономного режима.

**AutoCommand.java**: Абстрактный базовый класс для создания сложных автономных команд, поддерживающий последовательное выполнение действий.

## **2. Настройка и запуск**

### **2.1 Запуск робота**

- Программа начинается с **Main.java**, который вызывает **RobotBase.startRobot(Robot::new).**

Класс **Robot** инициализирует **RobotContainer** и управляет режимами работы.

### **2.2 Режимы работы:**

**robotInit()**: Инициализация при запуске.

**autonomousInit()**: Начало автономного режима.

**teleopInit()**: Начало телеуправления.

**testInit()**: Начало тестового режима**.**

## **3. Работа с подсистемами**

### **3.1 AutoTrain:**

Управляет движением (моторы, энкодеры, гироскоп).

Пример использования:



### **3.2 OMS:**

Управляет подъемом и сервоприводом.

Пример:



### **3.3 UDP:**

Принимает данные по **UDP** (координаты, статус дрона).

Пример:



## **4. Управление с геймпада**

Класс **OI** обрабатывает ввод с геймпада.

Пример:



Константы кнопок определены в **GamepadConstants.java**.

## **5. Автономный режим**

**Настройка автономных команд:**

В **RobotContainer** используется **SendableChooser** для выбора автономного режима.

Пример:



**Реализация команд:**

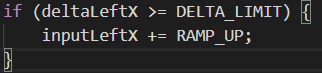
Команды автономного режима должны наследовать от **AutoCommand** или **CommandBase**.

## **6. Телеуправление**

### **6.1 Teleop.java:**

Обрабатывает ввод с геймпада и управляет движением.

Использует рампу для плавного изменения скорости:



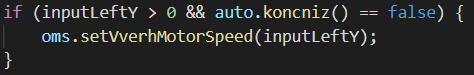
### **6.2 TeleopOMS.java:**

Управляет подсистемой **OMS** (например, подъемом и сервоприводом) в режиме телеуправления.

**Управление стволом:**

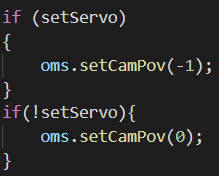
Использует входы с геймпада (**inputLeftY**) для перемещения ствола вверх/вниз.

Проверяет концевики (**koncniz**, **koncverh**), чтобы избежать перегрузки.



**Управление сервоприводом камеры:**

Реагирует на нажатие кнопки (**setServo**).



**Рампа для плавного изменения скорости:**

Применяется к входам с геймпада для обеспечения плавных движений.

**Советы:**

- Убедитесь, что концевики правильно подключены и настроены.

- Проверьте направление движения ствола и сервопривода.

## **7. UDPCommands.java**

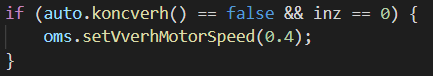
Обработка **UDP-команд** и взаимодействие с подсистемами.

**Инициализация:**

Закрывает клапан при запуске (**udp.clopen(false)**).

**Автоматическое перемещение ствола:**

Метод **nullVerf()** перемещает ствол до срабатывания концевиков.



**Советы:**

- Для отладки **UDP** используйте логирование данных.

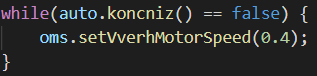
- Проверьте работу концевиков перед использованием автономных команд.

## **8. SimpleAuto.java**

Простая автономная команда для тестирования базовых функций.

**Автоматическое перемещение лифта:**

Метод **nullposition()** перемещает ствол вниз до срабатывания концевика **koncniz**.



**Советы:**

- Используйте **Timer.delay()** для временных задержек, но избегайте блокирующих вызовов в основном цикле.

- Добавьте проверки на прерывание команды.

## **9. Auto.java**

Реализация автономной команды, наследуемой от **AutoCommand**.

**Использование SimpleAuto**:

Создает экземпляр **SimpleAuto** с параметрами (**x=0.0, y=0.5, z=0.0**).



**Советы:**

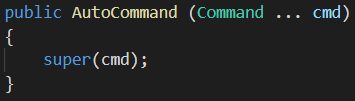
- Параметры **SimpleAuto** можно настроить для конкретных задач.

- Добавьте больше команд в **SequentialCommandGroup** для сложных автономных сценариев.

## **10. AutoCommand.java**

Базовый класс для автономных команд, использующий **SequentialCommandGroup**.

Позволяет комбинировать несколько команд в последовательность.



**Советы:**

- Наследуйте от этого класса для создания сложных автономных режимов.

- Используйте **ParallelCommandGroup** для параллельного выполнения команд.

## **11. Shuffleboard**

Данные выводятся на вкладке "**Turel**":

- Давление,

- Энкодеры,

- Гироскоп.

Пример добавления данных:





## **12. Советы и рекомендации.**

**Отладка:**

- Используйте **Shuffleboard** для мониторинга данных.

- Логируйте ключевые события (например, получение **UDP-пакетов**).

**Безопасность:**

- Всегда проверяйте пределы значений (например, скорость моторов).

- Используйте концевики для предотвращения перегрузки механизмов.

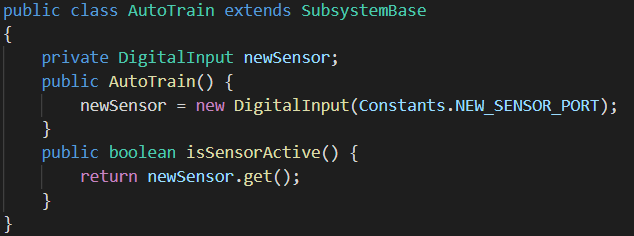
**Оптимизация:**

- Избегайте блокирующих операций в основном цикле (например, используйте потоки для **UDP**).

**Документация:**

- Комментируйте код, особенно сложные алгоритмы (например, голономное управление).

**Пример добавления датчика:**



Следуя этой инструкции, вы сможете эффективно работать с кодом, добавлять новые функции и отлаживать существующие.

Для полного ознакомления с кодом смотреть **Приложение 7**.

# **Python**

## **13. Назначение**

Файл **drone\_ai.py** реализует систему обнаружения дронов с использованием модели **YOLO** (You Only Look Once) и передает координаты обнаруженных объектов через **UDP-клиент**. Основные функции:

- Обнаружение дронов на видео или с камеры в реальном времени.

- Передача координат и статуса обнаружения (**isDrone**) на сервер по **UDP**.

- Визуализация результатов обнаружения.

## **14. Зависимости**

Убедитесь, что установлены следующие библиотеки:

## **15. Ключевые компоненты**

### **15.1 YOLO-модель:**

Загружает предобученную модель **best558.pt**.

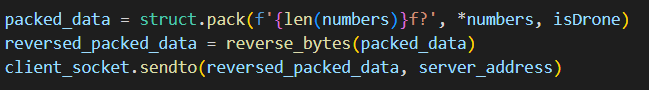
Настраивает порог уверенности (**conf=0.75**).



### **15.2 UDP-клиент:**

Отправляет координаты (**x, y**) и флаг **isDrone** на сервер (**10.12.34.2:8080**).

Данные упаковываются в байты и переворачиваются для совместимости.



### **15.3 Обработка видео:**

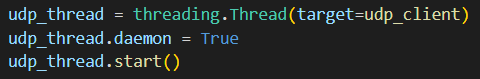
Чтение видео с камеры (**cv2.VideoCapture(1)**) или файла.

Визуализация **bounding boxes** для обнаруженных объектов.



### **15.4 Многопоточность:**

**UDP-клиент** запускается в отдельном потоке для асинхронной работы.



## **16. Настройка**

### **16.1 Модель YOLO:**

Замените **best558.pt** на путь к вашей модели.

Настройте **threshold\_confidence** для фильтрации ложных срабатываний.

### **16.2 UDP-сервер:**

Измените IP и порт сервера в переменных **ipRasp** и **server\_address**.



### **16.3 Источник видео:**

Для использования камеры укажите индекс (**cv2.VideoCapture(0)**).

Для видеофайла укажите путь (**video\_path = "путь/к/файлу.mp4"**).

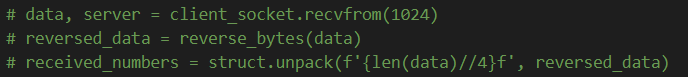
## **17. Пример использования**

1) Запустите скрипт:



2) Нажмите **q** для выхода.

3) Для отладки **UDP** раскомментируйте строки:



## **18. Важные замечания**

**Производительность**: Для ускорения работы используйте **GPU (CUDA)**. Убедитесь, что **PyTorch** с поддержкой **CUDA** установлен.

**Безопасность**: При работе с **UDP** убедитесь, что сервер доступен и данные корректно интерпретируются.

**Логирование**: Для отладки добавьте логирование в файл или консоль.

## **19. Дополнительные возможности**

**Сохранение видео**: Используйте **cv2.VideoWriter** для записи результатов.

**Кастомизация**: Добавьте обработку других классов объектов, изменив **drone\_class\_id**.

Для полного ознакомления с кодом смотреть **Приложение 7**.