Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕНЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет энергетический
Кафедра автоматизации производственных процессов и электротехники
Направление подготовки 15.03.04 – Автоматизация технологических процессов и производств
Направленность (профиль) образовательной программы Автоматизация технологических процессов и производств в энергетике

ДОПУСТИТЬ К ЗАЩИТЕ
И.о. зав. кафедрой
____________ О.В. Скрипко
«24» 06 2024 г.

БАКАЛАВРСКАЯ РАБОТА

на тему: «Автоматизированная система управления электрошасси на базе мотор-колёс»

Исполнитель
студент группы 041-об ________ 21.06.24 ________ В. И. Бендик
(подпись, дата)

Руководитель
доцент, канд. техн. наук ________ 21.06.2024 ________ Д. А.Теличенко
(подпись, дата)

Консультант: по безопасности
и экологичности
доцент, канд. техн. наук ________ 11.06.2024 ________ А. Б. Булгаков
(подпись, дата)

Нормоконтроль
профессор, д-р техн. наук ________ 24.06.2024 ________ О. В. Скрипко
(подпись, дата)

Благовещенск 2024

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет энергетический
Кафедра автоматизации производственных процессов и электротехники

УТВЕРЖДАЮ
И. о. зав. кафедрой
_____________ О.В. Скрипко
подпись И.О. Фамилия
«__24__»__06__________ 2024 г.

З А Д А Н И Е

К выпускной квалификационной работе студента Бендик Владимира Игоревича
1. Тема выпускной квалификационной работы: Автоматизированная система управления электрошасси на базе мотор-колес
(утверждена приказом от 11.04.24 № 965-уч)
2. Срок сдачи студентом законченной работы (проекта) 27.06.2024
3. Исходные данные к выпускной квалификационной работе: 1) Открытое программное обеспечение «Hoverboard firmware hack FOC»
4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов):
1) Описание объекта управления;
2) Выбор устройств системы управления;
3) Разработка программного обеспечения контроллера электропривода;
4) Разработка программы ручного управления.
5) Разработка системы беспилотного управления электрошасси
6) Руководство пользователя
7) Безопасность и экологичность
5. Перечень материалов приложения:
Лист 1: Чертёж сборочный
Лист 2: Схема драйвера электрическая
Лист 3: Схема электрическая электрошасси
Лист 4: Схема функциональная
Лист 5: Блок схема алгоритма программы системы управления электрошасси
Лист 6: Тестирование системы проектирования маршрута
6. Консультанты по выпускной квалификационной работе:

Безопасность и экологичность – Булгаков А.Б., доцент, канд. техн. наук

7. Дата выдачи задания 05.02.2024

Руководитель выпускной квалификационной работы:

Теличенко Денис Алексеевич, доцент кафедры АППиЭ, канд. техн. наук

(фамилия, имя, отчество, должность, ученая степень, ученое звание)

Задание принял к исполнению (дата): 05.02.2024 21.06.24

(подпись студента)

РЕФЕРАТ

Бакалаврская работа содержит 70 с., 37 рисунков, 4 таблицы, 7 частей, 21 источник, 1 примечание.

ЭЛЕКТРОШАССИ, ЭЛЕКТРОПРИВОД, БДПТ, BLDC, ДРАЙВЕР УПРАВЛЕНИЯ, STM32, ST-LINK, VISUAL STUDIO CODE, ESP32, BLUETOOTH, RC-PWM, ВЕКТОРНОЕ УПРАВЛЕНИЕ, МИКРОКОНТРОЛЛЕР, ARDUINO IDE, ПРОГРАММА, АППАРАТНО-СТРУКТУРНАЯ СХЕМА, UART ИНТЕРФЕЙС, GPS, Wi-Fi, UDP.

Целью данной выпускной квалификационной работы является разработка системы автоматизированного управления электрошасси на базе мотор-колес. Для решения поставленной задачи был разработан опытный образец рамы электрошасси, проведен реинжиниринг драйвера BLDC моторов на основе микроконтроллера STM32, изучены современные системы электропривода с применение технологии BLDC моторов, проведен выбор компонентов электрошасси и написано программное обеспечение для микроконтроллеров ESP32 и STM32.

Разработанная система управления обладает двумя режимами работы: ручным и автоматическим. Ручное управление выполняется при помощи RC пульта, а автоматическое осуществляет движение по заданным GPS координатам.

Благодаря разработанному электрошасси можно проводить обучение студентов технологиям электропривода, мобильной робототехники и программированию микроконтроллеров. Также проект можно продолжать совершенствовать, интегрировав в систему управления экстероцептивные датчики. Доработанный проект будет иметь коммерческий потенциал в качестве робота служб доставки, складов или для разработки на его основе других мобильных роботов.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

BLDC (Brushless Direct Current) – бесщеточный двигатель постоянного тока.

ESP32 – микроконтроллер, разработанный компанией Espressif Systems.

STM32 – серия микроконтроллеров от STMicroelectronics, основанных на ядре ARM Cortex-M.

Arduino IDE – интегрированная среда разработки для микроконтроллеров.

Visual Studio Code – интегрированная среда разработки от Microsoft.

Smartphone IMU GPS – приложение для Android, предоставляющее доступ к датчикам смартфона.

Компас-3D – Программное обеспечение для создания 3D моделей и чертежей.

UART (Universal Asynchronous Receiver/Transmitter) – универсальный асинхронный приемник/передатчик.

GPS (Global Positioning System) – глобальная система позиционирования.

UDP (User Datagram Protocol) – протокол пользовательских датаграмм.

CSV (Comma-Separated Values) – формат представления табличных данных с использованием запятых в качестве разделителя.

PWM/ШИМ (широтно-импульсная модуляция) – способ модуляции, использующий импульсы переменной ширины.

RC (Radio Controlled) – радиоуправляемый.

ПИД-регулятор – пропорционально-интегрально-дифференциальный регулятор.

BMS (Battery Management System) – система управления аккумулятором.

GIS (Geographic Information System) – геоинформационная система.

ВВЕДЕНИЕ

Разработка мобильных робототехнических платформ, способных к автономному передвижению и выполнению заданных действий, является актуальной задачей в современной робототехнике.

Для разработки проекта были использованы следующие компоненты: 4 мотор-колеса, драйверы BLDC моторов на основе микроконтроллера STM32F103C8T6, микроконтроллер ESP32, RC-пульт дистанционного управления, Android смартфон Redmi Note 8 Pro. Также было использовано программное обеспечение Arduino IDE, Visual Studio Code, Smartphone IMU GPS и Компас-3D.

Тема автоматизированного управления электрошасси на базе мотор-колес является малоизученной и требует дальнейшего развития, особенно в контексте использования GPS-навигации и беспилотного управления. Полученный результат может быть использован для создания практических робототехнических устройств, способных решать реальные задачи в различных сферах.

В работе был реализован беспилотный режим движения электрошасси с использованием GPS-навигации и переводом её в прямоугольную систему координат X, Y. Разработан алгоритм движения по построенной траектории, основанный на вычислении угла поворота и расстояния до заданной точки назначения.

Разработанное электрошасси может быть использовано в качестве учебного стенда для обучения студентов основам робототехники, электроники и программирования микроконтроллеров.

В дальнейшем проект может быть доработан и использован для создания коммерческих роботов, способных выполнять различные задачи: доставку товаров, обслуживание складских помещений, уборку территории, и др.

1 ОБЗОР ОБЪЕКТА УПРАВЛЕНИЯ

1.1 Назначение электрошасси

Электрошасси – это платформа с электрическим приводом, на которую устанавливаются различные компоненты и модули для постройки различных видов транспортных средств и устройств мира робототехники.

Электрошасси обладают рядом преимуществ, по сравнению с классическими транспортными средствами на основе ДВС, таких как высокая эффективность и экономия энергии, плавный и тихий ход, и отсутствие продуктов сгорания, что делает их крайне удобными для городской среды, и что особенно важно при работе в закрытом пространстве, таком как склады или заводы. Все это делает электрошасси идеальным выбором для различных отраслей, включая робототехнику. На основе различных электрошасси выполняются роботы доставщики, погрузчики, сельскохозяйственные роботы, медицинские и другие.

В зависимости от конкретной модели и назначения, электрошасси могут обладать различными функциями и возможностями, такими как: системы управления беспилотным движением, устройствами сканирования пространства вокруг, установленными роботами-манипуляторами, системами повышенной безопасности и другими. Они также могут быть настроены для работы с разными типами кузовов и каркасов, что дает возможность создавать разнообразные транспортные средства, ввиду этого электрошасси играют важную роль в развитии электромобильной индустрии и сферы робототехники. Они обеспечивают более устойчивую и экологически чистую альтернативу транспортным средствам на базе двигателей внутреннего сгорания, способствуют повышению энергоэффективности, уменьшению загрязнения окружающей среды и созданию более комфортной городской среды.

1.2 Конструкция электрошасси

При изготовлении электрошасси первым этапом была разработка опытного образца рамы. Для этого был использован металлический профиль высотой

20 мм. Из него было выполнено прямоугольное основание, на котором в будущем будут размещены системы управления и полезная нагрузка. Снизу основания были добавлены профили для крепления к ним колес. Итоговый размер рамы вышел 355х700 мм.

Для надежной фиксации колес, а также упрощения сборки/разборки шасси были изготовлены крепления колес в раме. Они надежно крепятся к раме при помощи 4 болтов диаметром 6 мм и фиксируют колесо 2 винтами. Чертеж крепления и его размеры представлен на рисунке 1.1.

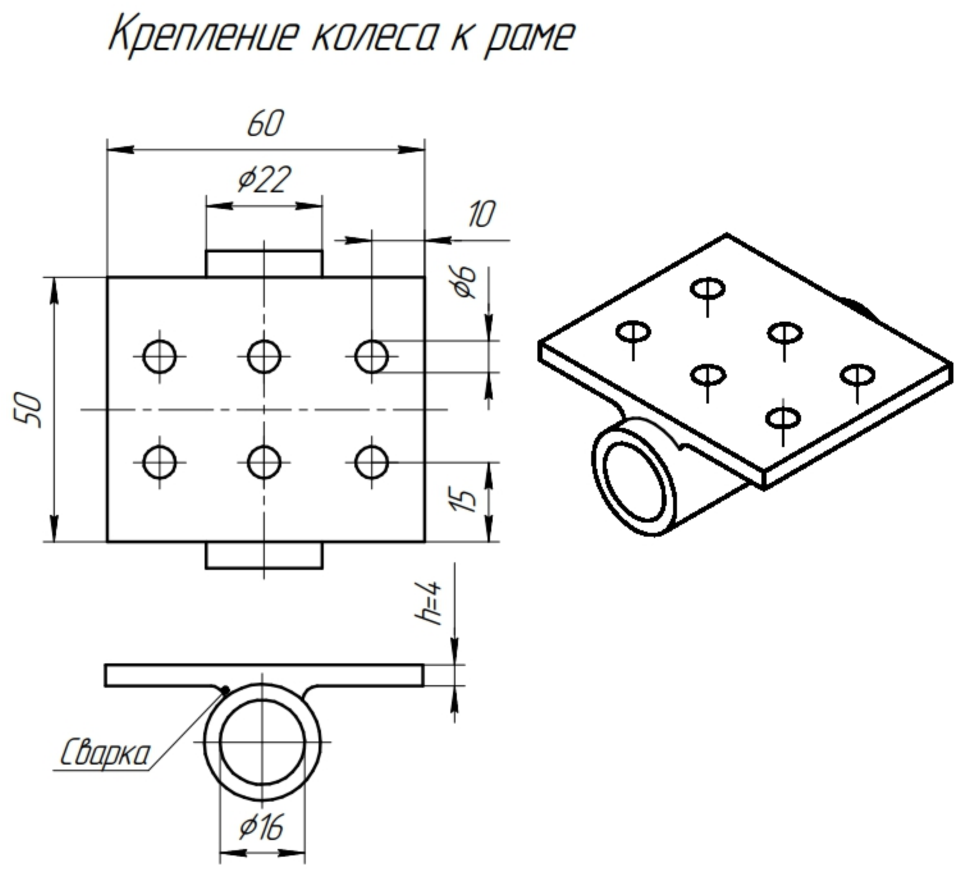


Рисунок 1.1 – Чертеж крепления колеса к раме

Детальный выбор колес будет описан в пункте 1.3, однако забегая вперед можно сказать, что в качестве колес будут выбраны тяговые BLDC моторы диаметром 16,5 см.

На основе размеров рамы, крепления колес и возможного вида будущих колес, в программе Solidworks была разработана 3D модель рамы, представленная на рисунке 1.2. Также на модели изображен отсек для установки в нем систем управления.

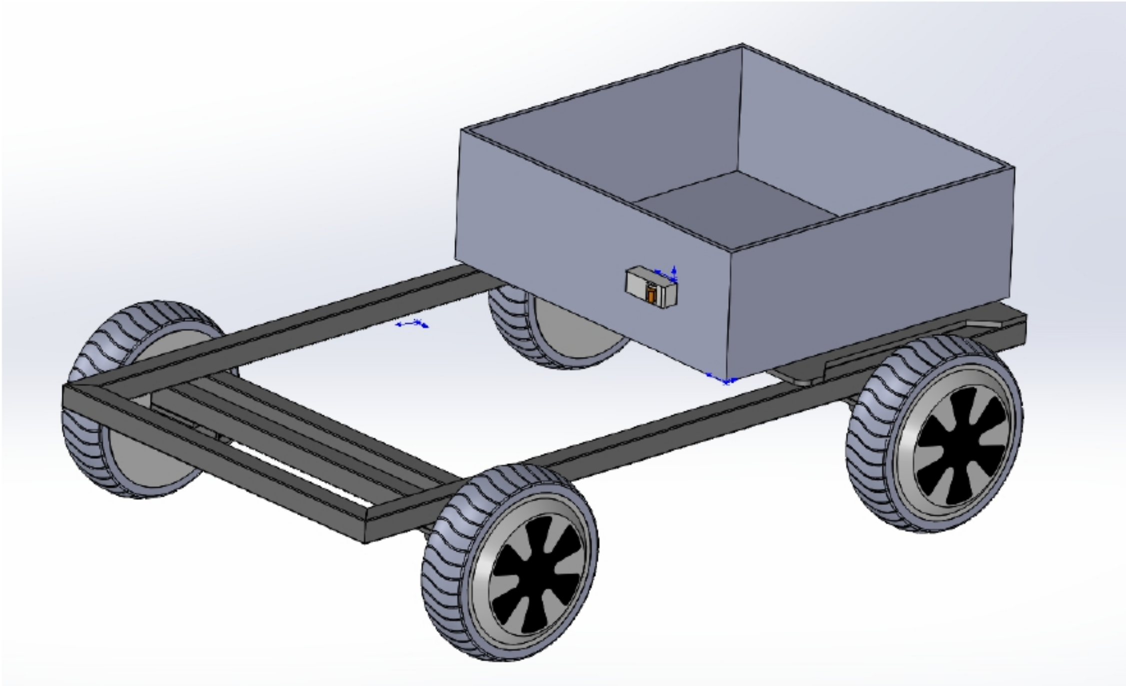


Рисунок 1.2 – 3D модель рамы

Полная схема всех деталей электрошасси и их сборочный чертеж представлены на листе 1.

1.3 Электропривод

В качестве электропривода шасси будем использовать мотор-колеса.

Мотор-колеса – это электрические приводы, которые объединяют мотор и колесо в одном компактном устройстве. В современное время для них обычно применяются BLDC моторы. Они широко используются в робототехнике, автомобильной промышленности и других областях, где требуется передвижение и маневрирование.

Мотор-колеса с BLDC мотором будут оптимальным приводом для электрошасси по следующим причинам:

а) Интеграция: Мотор-колеса предоставляют удобное и компактное решение, объединяя мотор и колесо в одном устройстве. Это упрощает интеграцию в конструкцию электрошасси позволяя не проектировать механическую трансмиссию.

б) Эффективность: Мотор-колеса разработаны с учетом высокой эффективности, что позволяет снизить потребление энергии и продлить время работы на аккумуляторе, способны работать в режиме рекуперации и электромагнитного тормоза, что позволяет экономить электроэнергию аккумулятора и отказаться от механического тормоза при проектировании шасси.

в) Надежность: Мотор-колеса не имеют щеточного узла, что увеличивает их надежность на фоне классических двигателей постоянного тока [1].

Недостатком мотор-колес можно назвать высокую сложность управления ими, о чем более подробно будет рассказано в пункте 1.4.

Для электрошасси применим мотор колеса ob20h36v применяемые в гироскутерах. За недолгое время на рынке они зарекомендовали себя высоким уровнем надежности и мощности при небольшой цене. Схема подобного колеса представлена на рисунке 1.3.

Таблица 1.1 – Характеристики выбранного мотор-колеса

Номинальное напряжение, В.	36
Максимальная мощность. Вт.	350
Рабочий ток, А.	≤16
Диаметр, см.	16.51
Крутящий момент, Нм.	10-120
Масса, Кг.	2.4
КПД, %	90

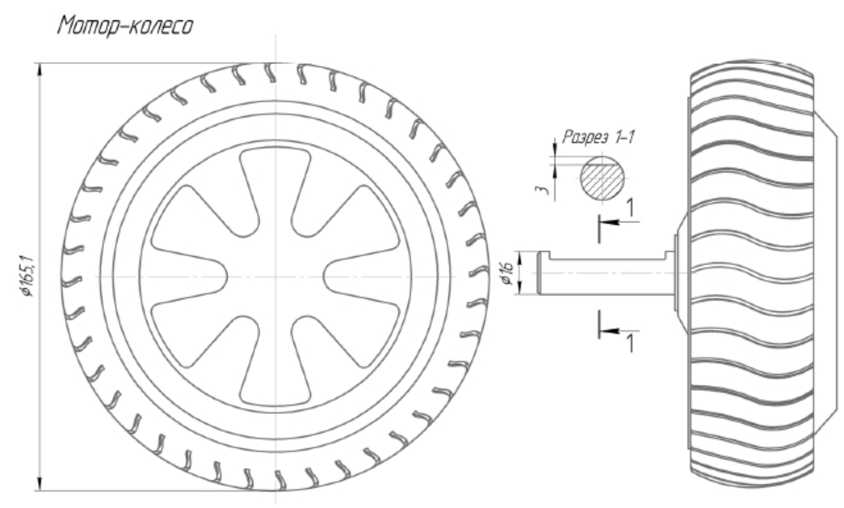


Рисунок 1.3 – Чертеж выбранного мотор-колеса

1.4 Принцип управления колесами BLDC

Мотор-колесо, как любая электрическая машина, имеет ротор и статор. В отличие от других машин, его индуктором является ротор из постоянных магнитов, а якорем – статор, состоящий из 3 фаз. Для работы BLDC мотора необходим драйвер, который выполняет функцию коммутации обмоток и располагается вне двигателя.

Магнитное поле возникает в статоре из электротехнической стали в форме многолучевой звезды. При взаимодействии с ротором из постоянных магнитов это поле вызывает вращение ротора. Обмотки на лучах статора превращаются в электромагниты при прохождении через них тока, которые притягивают магниты на роторе и запускают его вращение.

Статор имеет множество обмоток для обеспечения необходимой мощности и равномерного вращения колеса. Эти обмотки соединены в 3 группы и чередуются по окружности. На роторе расположены магниты из редкоземельных материалов, которые, взаимодействуя с электромагнитами на статоре, вызывают вращение ротора.

Напряжение подается на обмотки в нужные моменты времени, определяемые датчиками Холла на статоре. Они отслеживают расположение ротора и статора, отправляя сигналы на контроллер. С учетом этих данных контроллер, являющийся драйвером, обеспечивает подачу импульсов напряжения на обмотки статора, вызывая его вращение. Графически устройство BLDC мотора показано на рисунке 1.4 [21].

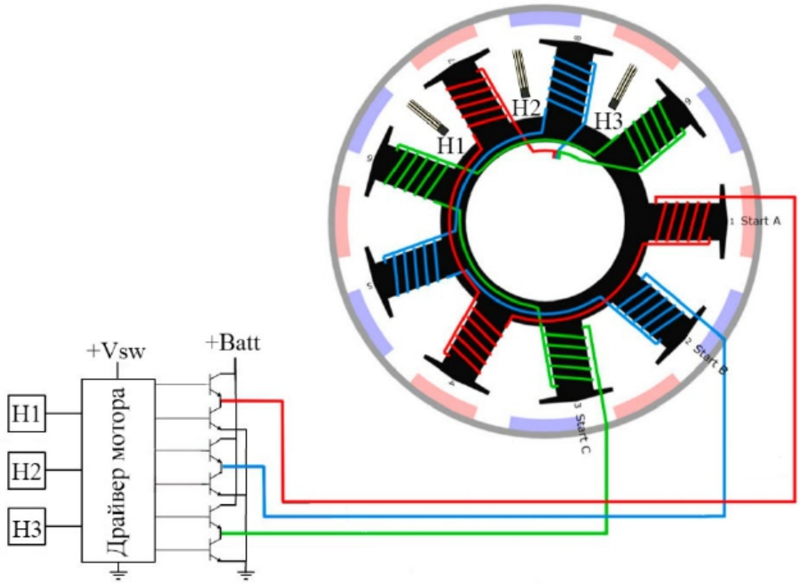


Рисунок 1.4 – Устройство BLDC мотора

Последовательность коммутации обмоток BLDC мотора зависит от конструкции двигателя и количества полюсов. Однако общая последовательность коммутации выглядит так:

1. Определяется положение ротора с помощью датчиков Холла.
2. Вычисляется необходимое напряжение и/или ток для каждой обмотки статора. Необходимое напряжение подается на обмотку за счет ШИМ сигнала.
3. Обмотки статора переключаются в нужной последовательности.
4. Повторяются шаги 1-3 для поддержания постоянного вращения ротора.

Благодаря ШИМ'у возможно создание синусоидальных форм тока в обмотках, такой метод управления называется синусоидальным. Самые продвинутые драйверы способны реализовывать векторное управление, т.к., забегая вперед, будет использоваться именно этот способ, то в пункте 1.5 будет дано описание этого метода.

1.5 Векторное управление

Векторное управление электродвигателями – это метод управления, который использует математическую модель двигателя для точного управления его крутящим моментом и скоростью. В отличие от традиционных методов управления, таких как управление по напряжению/частоте (V/f), векторное управление учитывает пространственное расположение магнитного поля двигателя и использует обратную связь по току для оптимизации его работы.

Векторное управление основано на представлении трехфазных токов и напряжений двигателя в виде пространственных векторов. Эти векторы вращаются в комплексной плоскости с угловой скоростью, пропорциональной скорости двигателя. Контроллер векторного управления использует обратную связь по току для определения положения ротора двигателя и вычисляет оптимальные значения напряжения и частоты для подачи на двигатель [3].

Преимущества векторного управления:

1. Высокий крутящий момент и динамические характеристики – векторное управление обеспечивает высокий крутящий момент при низких скоростях и быстрый отклик на изменения нагрузки.
2. Высокая точность и эффективность – при использовании векторного управления значительно повышается точность управления по скорости и крутящему моменту, что обеспечивает повышение КПД мотора, а значит выигрыш в энергоэффективной.
3. Низкий уровень шума и вибрации – векторное управление помогает снизить уровень шума и вибрации двигателя за счет оптимизации коммутации, однако, важно отметить, что низкий уровень шума и вибраций, в целом характерен для BLDC моторов.
4. Улучшенная надежность – современные системы, реализующие векторное управления, также способны обнаруживать и компенсировать неисправности двигателя, что значительно повышает надежность всей системы.

Чтобы реализовать векторное управление, необходим специальный контроллер, способный выполнять сложные математические расчеты в режиме реального времени. Для этой цели обычно применяются современные микроконтроллеры, наиболее часто используются микроконтроллеры STM32 компании STMicroelectronics на основе ядра ARM Cortex-M.

2 ВЫБОР УСТРОЙСТВ СИСТЕМЫ УПРАВЛЕНИЯ

2.1 Выбор драйвера управления колесами

Работа BLDC мотора невозможна без драйвера. Драйверы – это электронное устройств, основная задача которого обеспечивать коммутирование обмоток и контролировать скорость мотора и его направление вращения. Драйверы играют ключевую роль в эффективной работе двигателей и обеспечивают точное и стабильное управление движением.

Драйверы для BLDC моторов, условно, можно разделить на следующие группы:

а) Простые драйверы – это базовые драйверы, которые обеспечивают основные функции управления мотором колеса, такие как управление скоростью и направлением вращения, обратная связь в таком драйвере обеспечивается благодаря ЭДС. Применяются такие драйверы в моторах, не оборудованных датчиками Холла, и получили распространение в летном транспорте.

б) Драйверы с обратной связью – драйверы с возможностью получать обратную связь от датчиков положения, установленных на моторе. Это позволяет более точно контролировать скорость и положение колеса, что особенно полезно для устройств, требующих высокого вращательного. Подобные драйверы уже используются для тяговых моторов способных выдавать большой крутящий момент при небольшой скорости.

в) Драйверы с расширенными функциями – усовершенствованные драйверы с ОС, способные также предлагать дополнительные функции, такие как защита от перегрузки, торможение, рекуперативное торможение (возможность использования энергии при торможении для зарядки аккумуляторов) и другие продвинутые возможности.

Выбор драйвера для BLDC моторах происходит по следующим критериям:

а) Ток и напряжение – необходимо убедиться, что драйвер поддерживает требуемый диапазон тока и напряжения, соответствующий моторам.

б) Контроль скорости и направления по обратной связи – драйвер должен предоставлять необходимые возможности для точного и стабильного управления скоростью и направлением вращения моторов колес. Без этого ваш мотор не будет работать в полную мощность.

в) Реализация в драйвере защитных функций, таких как защита от перегрузки, короткого замыкания и температурная защита, чтобы обеспечить безопасную и надежную работу системы.

Выбор подходящего драйвера для BLDC колес важен для обеспечения эффективной и надежной работы вашей системы. Важно уделить внимание требованиям проекта и выбрать драйвер, который соответствует потребностям в управлении мотор-колесами.

В качестве драйвера управления мотор колесами электрошасси будут использоваться материнские платы устанавливаемые в гироскутерах с трехплатной системой [4]. Они способны работать с напряжением до 60В и током до 20А кратковременно. В дальнейшем основные платы будут называться просто драйверами. Внешний вид драйвера с информацией о его разъемах представлен на рисунке 2.1.

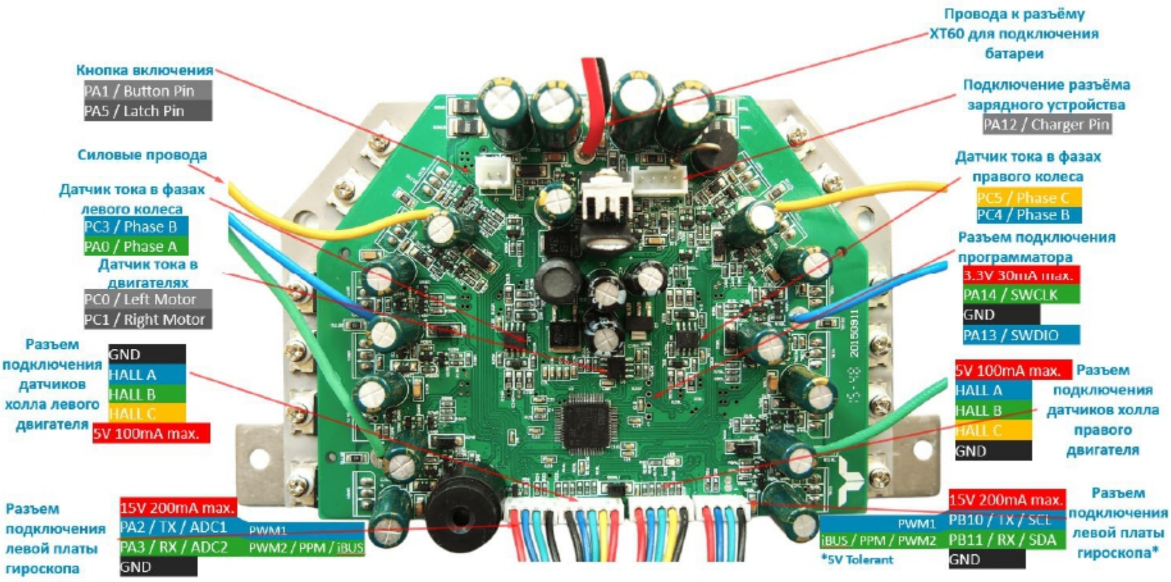


Рисунок 2.1 – Драйвер управления моторами

Полная электрическая принципиальная схема драйвера управления представлена на листе 2.

2.2 Выбор устройства ввода-вывода

В качестве устройства дистанционного управления драйверами будет использоваться микроконтроллер ESP-WROOM-32. ESP32 – это серия микроконтроллеров, разработанных компанией Espressif Systems. Микроконтроллеры ESP32 отличаются высокой производительностью, низким энергопотреблением и широкими возможностями подключения, что делает их идеальными для различных проектов Интернета вещей (IoT).

Основные характеристики ESP32:

1. Двухъядерный процессор Tensilica Xtensa LX6: работает на частоте до 240 МГц, обеспечивая высокую производительность для ресурсоемких задач.
2. Встроенный Wi-Fi и Bluetooth: поддерживает как Wi-Fi, так и Bluetooth, что позволяет устройствам подключаться к Интернету и другим устройствам без необходимости внешних модулей.
3. Низкое энергопотребление: благодаря передовым технологиям управления питанием ESP32 может работать от батареи в течение длительного времени, что делает его идеальным для устройств с питанием от батареи.
4. Встроенный контроллер заряда аккумулятора: позволяет заряжать литий-ионные аккумуляторы непосредственно через микроконтроллер, что упрощает разработку устройств с питанием от аккумулятора.
5. Множество периферийных устройств: включает в себя множество периферийных устройств, таких как таймеры, АЦП, ЦАП, SPI, I2C и UART, что позволяет подключать различные датчики, исполнительные механизмы и другие устройства [5].

Распиновка портов ESP-WROOM-32 представлена на рисунке 2.2.

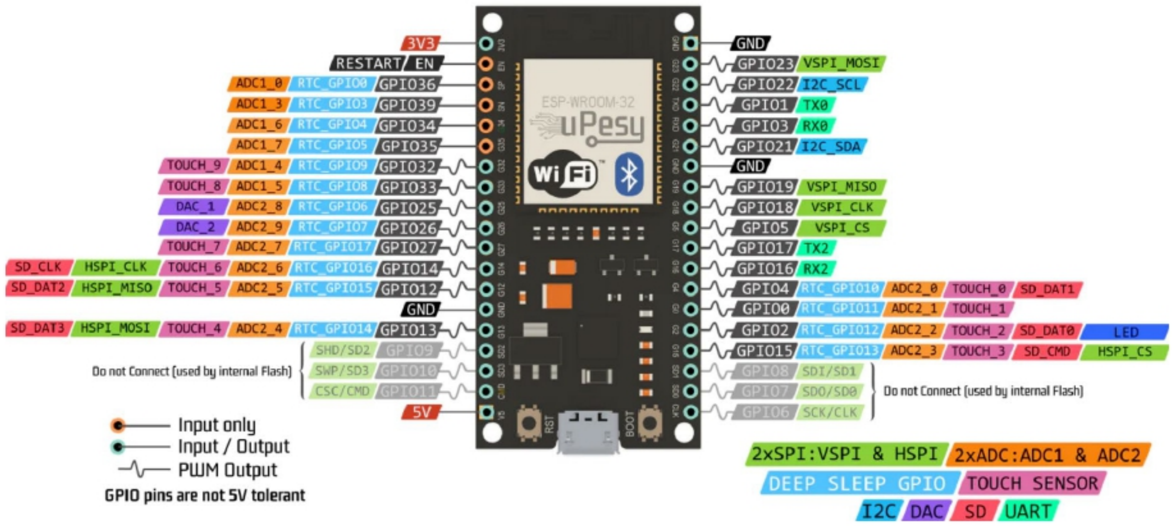


Рисунок 2.2 – Распиновка ESP-WROOM-32

Итого: система управления будет включать в себя микроконтроллер ESP32, на котором оператор задает скорость колес и угол поворота при движении при помощи внешнего устройства, выбор которого будет произведен в дальнейшем. ESP32, в свою очередь, соединен с драйвером по проводной связи через UART интерфейс. Структурная схема системы управления предоставлена на рисунке 2.3.

Рисунок 2.3 – Структурная схема управления электрошасси

Принципиальная электрическая схема соединений всех выбранных элементов представлена на листе 3. Функциональная схема всех элементов представлена на листе 4.

3 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОНТРОЛЛЕРА ЭЛЕКТРОПРИВОДА

3.1 Требования к функционалу реализуемому в ПО

Функционал, реализуемый в ПО для управления BLDC моторами, используемыми в качестве привода электрошасси, должен включать в себя следующие возможности:

1. Установка скорости вращения: ПО должно позволять пользователю устанавливать желаемую скорость вращения мотора.
2. Изменение направления вращения: ПО должно позволять пользователю изменять направление вращения колес.
3. Режимы работы: ПО может предоставлять различные режимы работы моторов, такие как постоянная скорость, постоянная мощность или позиционирование. Пользователь может выбрать нужный режим работы в зависимости от требуемых задач.
4. Защитные функции: ПО может включать в себя защитные функции, которые предотвращают перегрузку мотора, перегрев, защиту от короткого замыкания и глубокого разряда батареи.
5. Обратная связь: ПО может предоставлять обратную связь о текущей скорости вращения, токе или других параметрах мотора.
6. Интеграция с другими системами: ПО может быть способным взаимодействовать с другими системами или устройствами, такими как контроллеры движения или системы автоматизации. Это позволит в будущем интегрировать в шасси современные автопилотные системы управления.

Все выше поставленные задачи реализованы в ПО с открытым исходным кодом hoverboard firmware hack FOC опубликованным на сайте github.com [4] автором Emanuel Feru. ПО специально разработано для плат описанных в пункте 2. Данное ПО реализует векторное управление электродвигателем и представляет широкий набор пользовательских настроек: позволяет использовать различные режимы управления, а также выбрать наиболее удобный интерфейс ввода для поставленной задачи.

Режимы управления доступные в hoverboard firmware hack FOC:

1. Коммутационный (COM_CTRL)
2. Синусоидальный (SIN_CTRL)
3. Векторный (FOC_CTRL) со следующими 3 режимами управления, которые могут быть заданы в файле config.h с помощью параметра CTRL_MOD_REQ:
   a) Режим напряжения (VLT_MODE): В этом режиме контроллер подает на двигатели постоянное напряжение. Рекомендуется для робототехники или приложений, где требуется быстрый отклик двигателя.
   b) Режим постоянной скорости (SPD_MODE): В этом режиме контроллер с замкнутым контуром реализует заданное число оборотов на входе, отклоняя любое возмущение (нагрузку), подаваемое на двигатель. Рекомендуется для робототехнических систем или систем с постоянной скоростью вращения.
   c) Режим управления крутящим моментом (TRQ_MODE): В этом режиме реализуется заданный крутящий момент на входе. Этот режим позволяет двигателю "свободно вращаться", когда заданный крутящий момент равен 0. Рекомендуется для большинства приложений с сидящим человеком-водителем.

Таблица 3.1 – Сравнение различных режимов управления

Режим управления	Сложность реализации	Эффективность	Плавность	Ослабление поля	Свободное движение накатом	Удержание в состоянии покоя
1	2	3	4	5	6	7
Коммутационный	-	-	++	n.a.	n.a.	+
Синусоидальный	+	++	++	+++	n.a.	n.a.

Продолжение таблицы 3.1

1	2	3	4	5	6	7
Режим напряжения	+ +	+ + +	+ +	+ +	n.a.	+ (2)
Режим постоянной скорости	+ + +	+ + +	+	+ +	n.a.	+ + +
Режим управления крутящим моментом	+ + +	+ + +	+ + +	+ +	+ + + (1)	n.a. (2)

(1) Включив ELECTRIC_BRAKE_ENABLE в config.h, можно регулировать величину свободного хода с помощью параметра ELECTRIC_BRAKE_MAX.

(2) Функцию удержания в состоянии покоя можно принудительно активировать, включив STANDSTILL_HOLD_ENABLE в config.h.

Во всех режимах векторного управления контроллер оснащен защитой от максимальной скорости двигателя и максимального тока двигателя. Это дает большие преимущества для удовлетворения потребностей многих роботизированных устройств при сохранении безопасности работы.

Интерфейсы ввода доступные в ПО:

1. VARIANT_ADC: моторы управляются двумя потенциометрами, подключенными к левому сенсорному кабелю (длинный провод).
2. VARIANT_USART: моторы управляются по последовательному протоколу. Команды могут быть отправлены с Arduino или схожих устройств.
3. VARIANT_NUNCHUK: Wii Nunchuk позволяет управлять газом, тормозом и рулем одной рукой.
4. VARIANT_PPM: пульт дистанционного управления с сигналом PPM Sum.
5. VARIANT_PWM: RC-пульт дистанционного управления с сигналом PWM.
6. VARIANT_IBUS: RC-пульт с протоколом Flysky iBUS, подключенный к кабелю левого датчика.
7. VARIANT_HOVERCAR: моторы управляются двумя педалями тормоза и

газа. Задний ход включается двойным нажатием на педаль тормоза в состоянии покоя.

8. VARIANT_HOVERBOARD: главная плата считывает данные с двух боковых плат, оснащенных гироскопами.
9. VARIANT_TRANSPOTTER: это сборка транспоттера, который представляет собой транспортную систему на базе гироскутера.
10. VARIANT_SKATEBOARD: это сборка скейтборда, управляемого с помощью RC-пульта с ШИМ-сигналом, подключенного к правому сенсорному кабелю.

3.2 Описание файлов в hoverboard firmware hack FOC

Hoverboard firmware hack FOC содержит в себе множество файлов, краткое описание каждого предоставлено в таблице 3.2

Таблица 3.2 – Описание файлов в ПО

Название файла	Краткое описание
1	2
platformio.ini	Основной файл фреймворка PlatformIO с информацией о микроконтроллере, для которого будет компилироваться ПО.
main.c	Подключение большинства библиотек, инициализация глобальных переменных, структур, таймеров. Содержание основных инструкций и основного цикла.
control.c	Используется для управления через Pulse Position Modulation.
comms.c comms.h	Используется для формирования команд отладки, посылаемых через один из UART'ов, при соответствующей настройке.
bldc.c	Файл, реализующий FOC управление. Инструкции вызываются прерываниями от Advanced-control timers (tim1 и tim8)

Продолжение таблицы 3.2

1	2
BLDC_controller.c BLDC_controller.h BLDC_controller_data.c rtwtypes.h	Файлы автоматически сгенерированного в MatLab кода. Используются в управлении.
util.c util.h	Общие служебные процедуры, совместно используемые несколькими модулями.
eeprom.c eeprom.h	Драйвер виртуального EEPROM
hd44780.c hd44780.h	Библиотека для работы в LCD дисплеем
pcf8574.c hd44780.h	Библиотека для работы в LCD дисплеем
setup.c setup.h	Инструкции инициализации платы
stm32f1xx_it.c stm32f1xx_it.h	Библиотека прерываний для процессоров Cortex-M3
system_stm32f1xx.c system_stm32f1xx.h	
config.h	Файл пользовательской настройки ПО под индивидуальные задачи
defines.h	Дефайны

3.3 Настройка файла config и компиляция

В нашем случае устройством ввода будет USART интерфейс, а управления будет осуществляться режимом напряжения, с выключенным удержанием в состоянии покоя.

Для включения USART ввода достаточно прописать «#define VARIANT_USART» в файле «config.h» после подключения сторонних директив.

Микроконтроллер STM32 имеет на борту 3 группы USART портов, однако только одна из них является толерантной к напряжению 5 вольт, что делает ее наиболее удобной для использования с внешними микроконтроллерами. Это группа Serial3 расположенная на пинах PB_10 и PB_11. На платах драйверов эти пины вынесены на Right Sideboard, поэтому для их использования в настройках программы нужно выбрать их в разделе VARIANT_USART SETTINGS. Таким образом настройки USART имеют следующий вид представленный на рисунке 1.

```
// ############################ VARIANT_USART SETTINGS ############################
#ifdef VARIANT_USART
  // #define SIDEBOARD_SERIAL_USART2 0
 // #define CONTROL_SERIAL_USART2  0    // left sensor board cable, disable if ADC or PPM is used! For Arduino control check the hoverSerial.ino
 // #define FEEDBACK_SERIAL_USART2      // left sensor board cable, disable if ADC or PPM is used!

  // #define SIDEBOARD_SERIAL_USART3 0
   #define CONTROL_SERIAL_USART3  0    // right sensor board cable. Number indicates priority for dual-input.
   //Disable if I2C (nunchuk or lcd) is used! For Arduino control check the hoverSerial.ino
   #define FEEDBACK_SERIAL_USART3      // right sensor board cable, disable if I2C (nunchuk or lcd) is used!

  // #define DUAL_INPUTS                  //  UART*(Primary) + SIDEBOARD(Auxiliary). Uncomment this to use Dual-inputs
  #define PRI_INPUT1            3, -1000, 0, 1000, 0     // TYPE, MIN, MID, MAX, DEADBAND. See INPUT FORMAT section
  #define PRI_INPUT2            3, -1000, 0, 1000, 0     // TYPE, MIN, MID, MAX, DEADBAND. See INPUT FORMAT section
  #ifdef DUAL_INPUTS
    #define FLASH_WRITE_KEY      0x1102  // Flash memory writing key. Change this key to ignore the input calibrations
    // from the flash memory and use the ones in config.h
    // #define SIDEBOARD_SERIAL_USART2 1   // left sideboard
    #define SIDEBOARD_SERIAL_USART3 1   // right sideboard
    #define AUX_INPUT1            3, -1000, 0, 1000, 0     // TYPE, MIN, MID, MAX, DEADBAND. See INPUT FORMAT section
    #define AUX_INPUT2            3, -1000, 0, 1000, 0     // TYPE, MIN, MID, MAX, DEADBAND. See INPUT FORMAT section
  #else
    #define FLASH_WRITE_KEY      0x1002  // Flash memory writing key. Change this key to ignore the input calibrations from the flash memory and
    // use the ones in config.h
  #endif
```

Рисунок 3.1 – Настройки USART

Чтобы повысить стабильность работы USART было решено понизить его скорость с 115200 по умолчанию, до 9600 бод. Делается это в пункте «#define USART3_BAUD» раздела «UART SETIINGS».

Режим управления напряжением включен в П.О. по умолчанию поэтому редактировать в коде больше ничего не нужно и остается лишь провести его компиляцию.

Запуск компиляции осуществляется кнопкой «PlatformIO: Build» расположенной в левом нижнем углу интерфейса Visual studio code. По завершению компиляции в терминале программы появится информационное сообщение, подобное изображенному на рисунке 2, говорящее о том прошивки с каким управлением удалось реализовать, а с каким нет. Ввиду использования одних и тех же входов микроконтроллера для разных прошивок часть прошивок всегда не будет скомпилирована. В нашем случае главное, что удалось скомпилировать прошивку «VARIANT_USART».

```
Environment          Status    Duration
-------------------  --------  ------------
VARIANT_ADC          FAILED    00:00:03.952
VARIANT_USART        SUCCESS   00:00:04.310
VARIANT_NUNCHUK      FAILED    00:00:03.460
VARIANT_PPM          FAILED    00:00:03.947
VARIANT_PWM          FAILED    00:00:03.537
VARIANT_IBUS         SUCCESS   00:00:04.269
VARIANT_HOVERCAR     FAILED    00:00:03.600
VARIANT_HOVERBOARD   FAILED    00:00:03.384
VARIANT_TRANSPOTTER  SUCCESS   00:00:04.393
VARIANT_SKATEBOARD   FAILED    00:00:03.522
====7 failed, 3 succeeded in 00:00:38.372====
```

Рисунок 3.2 – Информационное сообщение после компиляции

Созданный загрузочный файл находится в каталоге «hoverboard-firmware-hack-FOC-main EFeru\.pio\build\VARIANT_USART» под именем «firmware.bin».

3.4 Загрузка прошивки в плату

Загрузка готовой прошивки в плату осуществляется программатором ST-Link V2 через программу STM32 ST-LINK Utility. Программу можно бесплатно сказать на официальном сайте st.com. Драйверы для программатора устанавливаются вместе с программой.

Подключение программатора осуществляется к выводам +3,3 V, SWDIO, GND и SWCLK к соответствующим точкам на плате, как показано на рисунке 3.3.

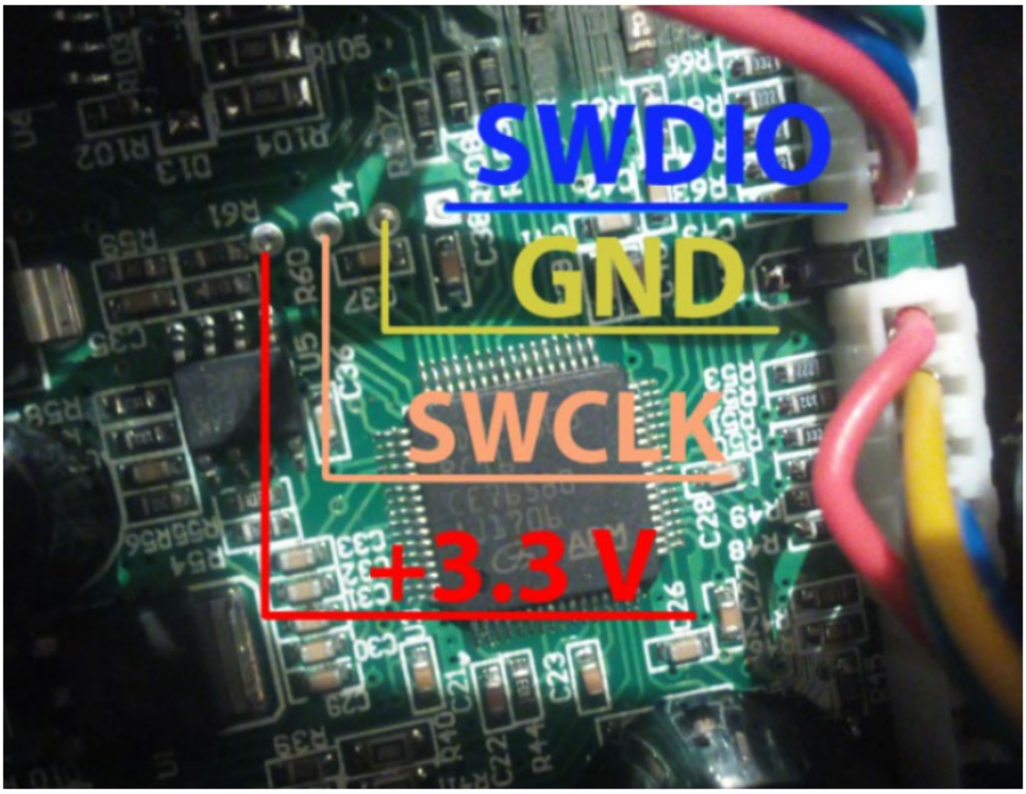


Рисунок 3.3 – Пины для подключения программатора на плате

После подключения программатора подключаемся к микроконтроллеру. Для этого в программе выбираем пункт «Target» и в нем «Connect». Если это происходит с платой впервые, то появится сообщение «Warning! Can not read memory!». Оно обозначает, что чтение загруженной в плату прошивки не доступно, так как она защищена от чтения. Для форматирования ПЗУ чипа используем пункт «Option Bytes» в разделе «Target». В поле Read Out Protection меняем Enabled на Disabled и жмем Apply. Теперь чип готов к загрузке на него пользовательской прошивки. В разделе поле «Open file» раздела «File» выбираем прошивку, скомпилированную в разделе 1.3. Для запуска процесса прошивки жмем «Target – Program & Verify…», в открывшемся окне – «Start». После удачного завершения процесса в нижнем окне увидим «Verifycation…OK» и «Programmed Memory Checksum: ******». Если загрузка прошла неудачно, то необходимо убедиться, что параметры загрузки установлены аналогично настройкам на рисунке 4.

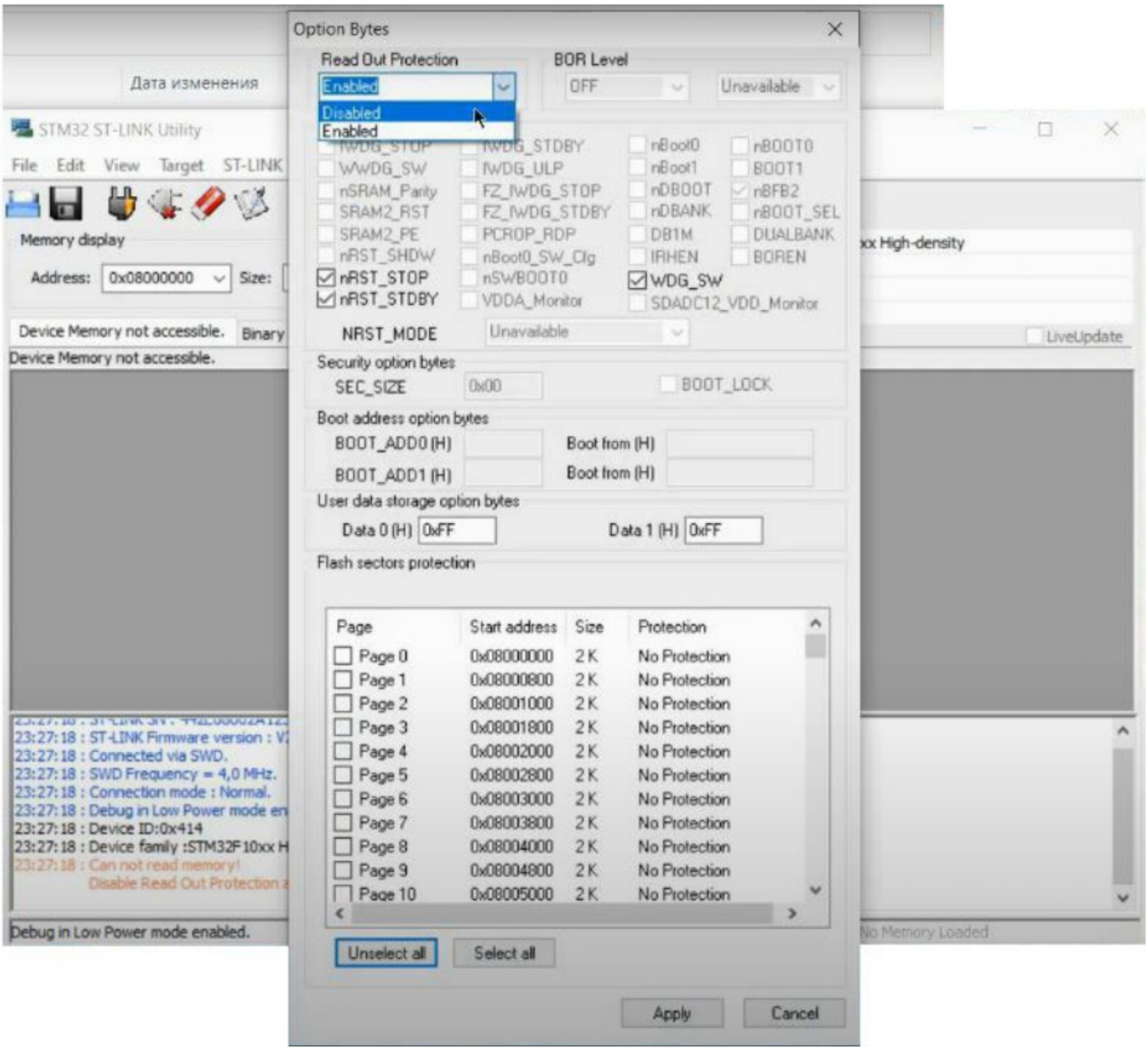


Рисунок 3.4 – Параметры загрузки в STM32 ST-LINK Utility

4 РАЗРАБОТКА ПРОГРАММЫ РУЧНОГО УПРАВЛЕНИЯ

4.1 Варианты управления

Основная задача ESP32 в электрошасси – быть устройством, связывающим оператора шасси и драйверы управления. Очевидно, что передача данных между оператором и шасси должна быть беспроводной. Исходя из этого можно использовать 2 популярных беспроводных способа передачи данных: Bluetooth и RC PWM. Каждый из вариантов имеет своими плюсами и минусами. Сравнение способов приведено в таблице 4.1.

Таблица 4.1 – Сравнение протоколов

Протокол	Плюсы	Минусы
Bluetooth	Простота использования, низкое энергопотребление, в качестве пульта управления можно использовать обычный смартфон.	Ограниченное расстояние, малые возможности: можно реализовать лишь связь с одним другим устройством.
RC PWM	Простота реализации, большое расстояние передачи данных.	Требует дополнительных устройств, невозможность реализовать двухстороннюю связь

В ходе разработки было решено реализовать оба варианта управления, чтобы опробовать их на практике.

4.2 Bluetooth

Bluetooth – это беспроводная технология ближнего радиуса действия, которая позволяет устройствам обмениваться данными на коротких расстояниях (обычно до 10 метров). Она используется для подключения различных устройств, таких как смартфоны, ноутбуки, наушники, динамики и принтеры [6].

Bluetooth использует радиоволны для передачи данных между устройствами. Устройства подключаются друг к другу, образуя сеть под названием пикосеть. Каждая пикосеть имеет одного ведущего (устройство, которое управляет соединением) и до семи ведомых (устройства, которые подключаются к ведущему).

Писать программу для выбранного микроконтроллера ESP32 было решено в среде Arduino IDE с установленным ядром arduino-esp32 [7]. ESP32 обладает встроенным Bluetooth ядром, для работы с которым удобно применять официальную библиотеку BluetoothSerial.h, кроме того, для работы с UART (обмен данными с драйверами) было решено использовать программную реализацию последовательного интерфейса при помощи библиотеки SoftwareSerial.h.

Для управления драйверами им необходимо посылать 2 переменные типа int16_t, что соответствует типу short в C++. Переменные именуются uSpeed и uSteer. Первая отвечает за скорость колес и задается в пределах от -1000 до 1000, однако в целях безопасности, ограничена через дефайн: #define SPEED_MAX_TEST 300. Вторая задаёт разницу между скоростью левого и правого колеса для регулирования поворота в движении. Работает это следующий образом: допустим uSpeed=200, uSteer =100, в таком случае на каждое колесо изначально подается скорость 200, затем uSteer делится пополам, для одного колеса полученное значение отнимается, для другого прибавляется. По итогу скорость одного колеса становится 150, а другого 250. Шасси начинает ехать в сторону медленного колеса. Если uSteer будет больше uSpeed, то колеса будут вращаться в разные стороны.

Чтобы правильно передать эти переменные и избежать ошибок необходимо иметь стартовый фрейм передачи, а также завершающий фрейм, содержащий в себе контрольную сумму переданного сообщения. Полный код функции отсылки данных представлен на рисунке 4.1.

```
// ########################## SEND FRONT ##########################
void Send_front(int16_t uSteer, int16_t uSpeed) {
  // Create command
  Command.start = (uint16_t)START_FRAME;
  Command.steer = (int16_t)uSteer;
  Command.speed = (int16_t)uSpeed;
  Command.checksum = (uint16_t)(Command.start ^ Command.steer ^ Command.speed);

  // Write to Serial
  HoverSerial_front.write((uint8_t *)&Command, sizeof(Command));
}
```

Рисунок 4.1 – Функция отсылки данных на драйвер передних колес

Функция приёма сообщений от драйвера похожа по смыслу, но выглядит значительно больше и сложнее из-за гораздо большего количества пересылаемой информации. В качестве обратной связи от драйвера ESP32 получает данные о скорости вращения колес, командах управления колесами, температуре платы и напряжении батареи.

Основной цикл программы занимается тем, что обрабатывает сообщения, поступающие по Bluetooth, в качестве посылаемого числа рассматривается числовой символ от 0 до 8, каждый символ изменяет переменные uSpeed и uSteer. Внешний вид цикла приведен на рисунке 4.2

```
void loop(void) {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    char incomingChar = SerialBT.read();
    Serial.write(incomingChar);
    if (incomingChar == '0') {
      iSpeed = 0;
      iSteer = 0;
    }
    if (incomingChar == '1') iSpeed = iSpeed + 10;
    if (incomingChar == '2') iSpeed = iSpeed - 10;
    if (incomingChar == '3') iSpeed = iSpeed + 50;
    if (incomingChar == '4') iSpeed = iSpeed - 50;
    if (incomingChar == '5') iSteer = iSteer + 10;
    if (incomingChar == '6') iSteer = iSteer - 10;
    if (incomingChar == '7') iSteer = iSteer + 50;
    if (incomingChar == '8') iSteer = iSteer - 50;
    SerialBT.print("iSpeed: ");
    SerialBT.print(iSpeed);
    SerialBT.print("  iSteer: ");
    SerialBT.println(iSteer);
  }

  if (iTest >= SPEED_MAX_TEST) iTest = SPEED_MAX_TEST;
  if (iTest <= -SPEED_MAX_TEST) iTest = -SPEED_MAX_TEST;
```

Рисунок 4.2 – Основной цикл программы

Отмечу, что для подобных конструкций грамотнее использовать конструкцию «switch case», а не «if», однако из-за невозможности использовать switch case с типом данных char пришлось бы добавлять код переводящий char в int, но при столь небольшой выборке данных if не сильно повлияет на быстродействие, поэтому было решено оставить так.

В качестве пульта управления удобно использовать смартфон на системе Android с приложением Bluetooth Terminal. Интерфейс этого приложения, во время работы, приведен на рисунке 4.3.

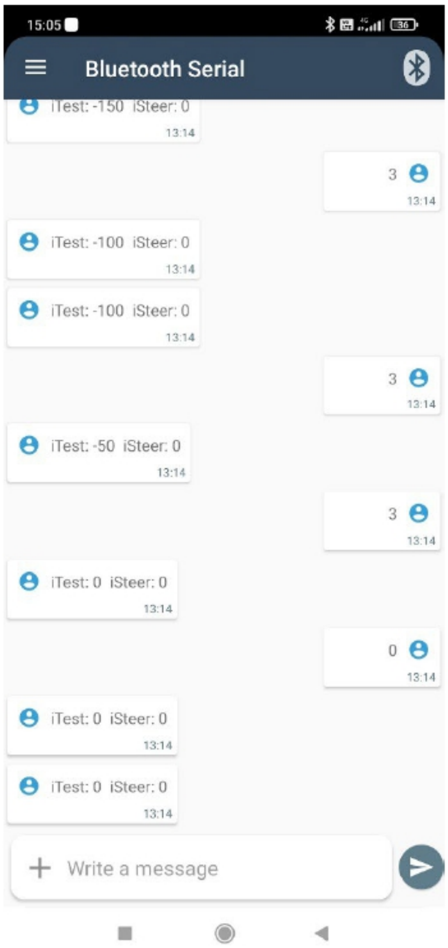


Рисунок 4.3 – Интерфейс приложения Bluetooth Terminal

4.3 RC пульт

RC PWM (широтно-импульсная модуляция) – это метод модуляции, используемый для передачи данных или управления устройствами с помощью импульсов переменной ширины. Он широко используется в радиоуправляемых (RC) системах для управления сервоприводами и регуляторами скорости двигателей [8].

RC PWM работает путем передачи серии импульсов с постоянной частотой, но с переменной шириной импульса. Ширина импульса представляет собой значение данных или команды, которую необходимо передать. Более широкие импульсы соответствуют более высоким значениям, а более узкие импульсы соответствуют более низким значениям.

Для измерения ширины импульсов используется функция pulseIn(). PulseIn() Считывает длину сигнала на заданном порту (HIGH или LOW). Например, если задано считывание HIGH функцией pulseIn(), функция ожидает пока на заданном порту не появиться HIGH. Когда HIGH получен, включается таймер, который будет остановлен, когда на порту вход/выхода будет LOW. Функция pulseIn() возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (таймаута) не был зафиксирован сигнал на порту.

Затем необходимо перенести значения из диапазона максимальной/минимальной скважности сигнала в диапазон максимальный/минимальной скорости. Это удобно сделать функцией map(). Map(value, fromLow, fromHigh, toLow, toHigh) Функция пропорционально переносит значение (value) из текущего диапазона значений (fromLow ... fromHigh) в новый диапазон (toLow ... toHigh), заданный параметрами. Итого код приема и обработки pwm сигнала имеет вид, представленный на рисунке 4.4.

```
void loop(void)
{
  duration_2 = pulseIn(CH2,HIGH);
  duration_3 = pulseIn(CH3,HIGH);

  s2_F = map(duration_2,1600,2000,speed_min,speed_max); //Правый стик вперед
  s2_B = map(duration_2,1400,1000,speed_min,-speed_max); //Правый стик назад
  s3_F = map(duration_3,1600,2000,speed_min,speed_max); //Левый стик вперед
  s3_B = map(duration_3,1400,1000,speed_min,-speed_max); //Левый стик назад
  //Serial.println(String(duration_2) + ' ' + String(duration_3));
  //unsigned long timeNow = millis();
  //if (iTimeSend > timeNow) return;
  //iTimeSend = timeNow + TIME_SEND;
```

Рисунок 4.4 – Код приёма и обработки PWM сигнала

Далее исходя из обработанный данных формируются команды пересылки заданий драйверу, с учетом дрифта курков пульта. Этот код представлен на рисунке 4.5.

```
if(duration_3 > 1590){
    Send_Left(s3_F*2, 0); //Вперед
}
else if(duration_3 < 1410 && duration_3 > 990){
    Send_Left(s3_B*2, 0); //Назад
}
else {
  Send_Left(0,0);
}
if(duration_2 > 1590){
    //Вращение вправо
    Send_Right(0, s2_F);
}
else if(duration_2 < 1410 && duration_2 > 990){
     //Вращение влево/ Умножение на 2, так как скорость распределяется на 2 колеса
    Send_Right(0, s2_B);
}
else {
  Send_Right(0,0);
}
```

Рисунок 4.5 – Код формирования и отправки задания

Тестирование показало, что управление через RC пульт удобнее и точнее, поэтому именно этот метод использован в финальной программе электрошасси.

5 РАЗРАБОТКА СИСТЕМЫ БЕСПИЛОТНОГО УПРАВЛЕНИЯ ЭЛЕКТРОШАССИ

5.1 Обзор основных элементов навигационных систем

Современные навигационные системы – это сложный комплекс технологий, работающий на стыке спутниковых коммуникаций, электроники, картографии и искусственного интеллекта. Чтобы понять, как они работают, необходимо разобраться в их основных элементах.

Источники данных:

Навигация – это, прежде всего, получение данных о нашем местоположении и окружающей среде. Для этого используются различные источники информации:

Спутниковые системы:

1. GPS (Global Positioning System): Американская система позиционирования, использующая сеть спутников для точного определения координат на Земле.
2. ГЛОНАСС (Глобальная Навигационная Спутниковая Система): Российская аналогичная система, дополняющая GPS и расширяющая доступность в различных регионах.
3. Galileo: Европейская спутниковая система, предлагающая высокую точность и доступность даже в сложных условиях.

Другие источники:

1. Радиомаяки: используются в морской и авиационной навигации для определения местоположения и курса.
2. Радарные данные: позволяют определять местоположение объектов, даже если они не видны визуально.
3. Карты: традиционные карты, электронные карты и карты в мобильных приложениях предоставляют информацию о местности, дорогах и ориентирах.
4. Сенсоры: акселерометры, гироскопы и компасы дополняют информацию

о местоположении и направлении движения, особенно в условиях ограниченного приема спутникового сигнала.

Приемники информации:

Чтобы получить данные от источника, используется специальный приемник.

1. GPS-приемник: это устройство, которое принимает сигналы от спутников GPS и обрабатывает их для определения координат.
2. ГЛОНАСС-приемник: работает аналогично GPS-приемнику, но принимает сигналы от спутников ГЛОНАСС.
3. Комбинированные приемники: способны принимать сигналы от нескольких спутниковых систем, что повышает точность позиционирования и обеспечивает стабильную работу даже в сложных условиях [9].

Устройства обработки информации:

Полученные данные от приемника не сразу становятся понятными для пользователя. Для их обработки необходим специальный процессор. Процессор анализирует полученные от приемника сигналы, фильтрует помехи и вычисляет точные координаты. Используя алгоритмы позиционирования, процессор определяет текущие координаты и строит карту местности. Исходя из заданного пункта назначения и данных о дорогах или местности, процессор вычисляет оптимальный маршрут, учитывая дорожные условия, ограничения скорости, заторы и другие факторы.

Интерфейс пользователя: взаимодействие с системой

Для комфортного взаимодействия с навигационной системой используется удобный интерфейс:

Дисплей: отображает карту, маршрут, информацию о местоположении, скорости, времени в пути, а также предупреждения и другие важные данные.

Клавиатура/сенсорный экран: позволяют вводить данные, выбирать пункт назначения, управлять настройками, масштабировать карту, изменять вид отображения.

Голосовое управление: обеспечивает удобное взаимодействие с системой, например, для задавания маршрутов, получения информации о дорожной обстановке, без отвлечения от дороги.

Для самостоятельно движения беспилотного аппарата ему необходимы данные о текущих координатах в пространстве и положении сторон света, а для более точного позиционирования при движении, необходимы элементы инерциальной навигации, такие как: акселерометры и гироскопы.

Датчики способные получить всю эту информацию расположены в любом современном смартфоне, поэтому для проекта беспилотного электрошасси в качестве навигационной системы будет использован Android смартфон Redmi note 8 pro.

5.2 Теоретические сведения о передаче навигационных данных

Существует немало приложений позволяющих получать доступ к датчикам смартфона, однако, для проекта было выбрано приложение Smartphone IMU GPS. Приложение обладает открытым исходным кодом и является полностью бесплатным. Кроме непосредственного доступа к датчикам, оно позволяет передавать информацию с них по UDP протоколу в формате CSV. Именно так ESP32, установленная в электрошасси, будет получать информацию [10].

Интерфейс приложения состоит из 2 разделов: Preference и Toggle sensors. В первом указываются настройки подключения: IP адрес устройства, на которое будет транслироваться UDP канал, ESP32 в нашем случае, и его порт. Также есть возможность изменить скорость передачи, включить разрешение приложению на работу в свернутом виде. В этом же разделе находится кнопка включения UDP стрима. В разделе Toggle sensors можно наблюдать в реальном времени за показаниями датчиком, а также включить разрешение на транслирование в стриме пользовательских данных. Это является обязательным, так как к таким данным, приложение относит геопозицию. Интерфейс приложения представлен на рисунке 5.1.

Рисунок 5.1 – Интерфейс приложения приложение Smartphone IMU GPS

Чтобы написать программу получающую информацию, необходимо разобраться с тем, что такое UDP протокол и CSV формат.

Протокол UDP (User Datagram Protocol, протокол пользовательских датаграмм) – это один из ключевых компонентов набора сетевых протоколов интернета. Он используется для отправки сообщений, или датаграмм, в сетях с использованием интернет-протокола. UDP является протоколом без установления соединения, что означает, что он не гарантирует доставку пакетов, порядок пакетов или защиту от дублирования. Особенностями работы UDP являются его быстродействие, простота реализации и ненадежность, что делает его крайне предпочтительным в задачах передачи большого количества данных, без страха, случайно передать ошибочное данное [11].

CSV (Comma-Separated Values) – это простой текстовый формат для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.

Пример CSV-текста:

Имя,Возраст,Город

Иван,25,Москва

Мария,30,Санкт-Петербург

Петр,28,Новосибирск

Преимущества CSV видны невооруженным глазом. Его крайне просто использовать, легко переносить между различными программами и удобно обрабатывать. Все это делает CSV простым и универсальным форматом для обмена данными, который подходит для многих задач, особенно для табличных данных [12].

5.3 Написание программы передачи навигационных данных

Для чтения UDP стрима на ESP32, можно использовать стандартную библиотеку WiFi.h, которая включает поддержку UDP. Для этого подключим к кому библиотеку и ее UDP раздел, а также создадим объект класса UDP.

Для транслирования данных смартфон с приложением и ESP32 должны находиться в одной Wi-Fi сети. Это удобно реализовать используя саму ESP в качестве точки доступа. Для этого в коде пропишем название и пароль будещей Wi-Fi точки. Последним шагом в области глобальных переменных будет выделение памяти под буфер входящего пакета.

В функции setup() необходимо запустить точку доступа, делается это функцией WiFi.softAP(), передаваемыми функции значениями являются название и пароль точки доступа; а также запустить чтение UDP стрима функцией Udp.begin(), передаваемым значением будет являться порт стрима.

Т.к. чтение UDP в будущем будет интегрироваться в большую программу, функцию чтения было решено написать отдельной функцией и просто вызывать ее из loop(). Особенностью работы Udp_read() является то, что она вызывается каждый цикл работы микроконтроллера, а значит, должна быть написана с как можно большим быстродействием. Это вынуждает отказаться от использования удобных string-типов данных и использовать массив типа Char созданный ранее.

Алгоритм работы функции Udp_read() следующий: сперва происходит вызов функции udp.parsePacket(), эта функция проверяет наличие UDP-пакета и сообщает его размер. Далее, если размер больше нуля, необходимо записать данные в выделенный буфер. Сделать это можно функцией Udp.read() передав ей в качестве параметров буфер и его размер. Кроме того, Udp.read() возвращает количество символов записанных в буфер, что нужно, для определения его конца нового сообщения, когда в буфере будут старые данные. Для этого, в буфер под индексом соотвествующему значению количества записанных сомволов, запишем «0». Этот ноль также называется нуль-терминатором строки. Функция Udp.read() также способна вернуть «-1», когда буфер стрима оказался пустым, поэтому для избежания ошибок, выполним проверку вернувшего ею значения на больше нуля. Для тестрирования работы выведем буфер incomingPacket в последовательный порт функцией Serial.printf(). Готовый код представлен на рисунке 5.2, результат выполнения работы программы представлен на рисунке 5.3.

```cpp
#include <WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "ESP-32-TEST";
const char* password = "PASSWORD";

WiFiUDP Udp;

char incomingPacket[512];  // Буфер для входящего пакета

void setup() {
  Serial.begin(9600);
  WiFi.softAP(ssid, password);  // Запуск точки доступа 192.168.4.1
  Udp.begin(8888);
}

void loop() {
  Upd_read();
}

void Upd_read() {
  if (Udp.parsePacket()) {
    int len = Udp.read(incomingPacket, 512);
    if (len > 0) incomingPacket[len] = 0;
    Serial.printf("UDP packet received: %s\n", incomingPacket);
    // ДАЛЕЕ НЕОБХОДИМО ОБРАБАТЫВАТЬ ПОЛУЧЕННЫЕ ЗНАЧЕНИЯ
  }
}
```

Рисунок 5.2 – Код чтения UDP стрима

```
Output    Serial Monitor ×
Message (Enter to send message to 'ESP32-WROOM-DA Module' on 'COM3')    New Line
UDP packet received: 142218.84162, 3,   0.260,  0.333,  9.589, 4,  -0.000,  0.002,  0.001, 5,  36.544, 13.500,-32.813, 81, 271.150, -1.900,  1.500, 82,  -0.012,  0.013, -0.222
UDP packet received: 142218.96099, 3,   0.262,  0.331,  9.585, 4,  -0.001, -0.000, -0.000, 5,  36.356, 13.556,-32.813, 81, 271.150, -1.900,  1.500, 82,   0.000, -0.027, -0.249
UDP packet received: 142219.25944, 3,   0.252,  0.328,  9.546, 4,  -0.001, -0.000, -0.001, 5,  36.000, 14.944,-33.356, 81, 271.150, -1.930,  1.510, 82,   0.000, -0.017, -0.249
UDP packet received: 142219.37882, 3,   0.262,  0.336,  9.573, 4,   0.000,  0.002,  0.002, 5,  36.300, 13.744,-33.056, 81, 271.150, -1.920,  1.510, 82,  -0.014, -0.002, -0.249
UDP packet received: 142219.61757, 3,   0.257,  0.333,  9.558, 4,   0.000, -0.000, -0.001, 5,  36.788, 13.613,-32.344, 81, 271.150, -1.930,  1.510, 82,   0.000,  0.009, -0.249
UDP packet received: 142219.73695, 3,   0.267,  0.331,  9.577, 4,   0.000,  0.002, -0.000, 5,  36.225, 13.781,-32.081, 81, 271.150, -1.930,  1.500, 82,  -0.012, -0.003, -0.249
UDP packet received: 142219.97570, 3,   0.260,  0.316,  9.565, 5,  36.581, 13.856,-32.513, 81, 271.150, -1.920,  1.510, 82,  -0.013, -0.016, -0.236
UDP packet received: 142220.09508, 3,   0.262,  0.319,  9.570, 4,   0.000, -0.000, -0.001, 5,  36.225, 14.213,-32.456, 81, 271.150, -1.920,  1.510, 82,   0.000, -0.002, -0.222
UDP packet received: 142220.33384, 3,   0.252,  0.336,  9.565, 4,  -0.001, -0.000,  0.002, 5,  36.225, 14.044,-33.225, 81, 271.150, -1.930,  1.510, 82,  -0.014, -0.005, -0.236
UDP packet received: 142220.45321, 3,   0.267,  0.331,  9.580, 4,  -0.000, -0.000, -0.001, 5,  36.488, 13.556,-32.381, 81, 271.150, -1.920,  1.510, 82,  -0.001, -0.016, -0.222
UDP packet received: 142220.69197, 3,   0.274,  0.350,  9.568, 4,   0.001, -0.003,  0.002, 5,  36.656, 13.556,-32.644, 81, 271.150, -1.930,  1.530, 82,  -0.016, -0.005, -0.236
UDP packet received: 142220.81135, 3,   0.267,  0.333,  9.582, 4,   0.003,  0.001, -0.000, 5,  36.356, 13.613,-32.813, 81, 271.150, -1.950,  1.530, 82,  -0.003, -0.006, -0.222
UDP packet received: 142221.05010, 3,   0.250,  0.338,  9.570, 4,  -0.001,  0.002,  0.001, 5,  36.581, 13.500,-32.381, 81, 271.150, -1.930,  1.510, 82,  -0.013, -0.019, -0.222
UDP packet received: 142221.16948, 3,   0.257,  0.336,  9.570, 4,   0.001, -0.002, -0.000, 5,  37.144, 13.556,-32.381, 81, 271.150, -1.930,  1.510, 82,   0.000, -0.018, -0.222
UDP packet received: 142221.46792, 3,   0.257,  0.350,  9.589, 4,   0.001, -0.000, -0.000, 5,  36.000, 14.213,-32.756, 81, 271.150, -1.930,  1.500, 82,  -0.011, -0.004, -0.222
UDP packet received: 142221.58730, 3,   0.264,  0.326,  9.556, 4,   0.001, -0.000, -0.000, 5,  35.925, 14.513,-32.456, 81, 271.150, -1.930,  1.500, 82,   0.000, -0.003, -0.222
UDP packet received: 142221.82606, 3,   0.267,  0.321,  9.556, 4,   0.000,  0.001,  0.002, 5,  36.188, 14.100,-33.056, 81, 271.150, -1.920,  1.510, 82,   0.000, -0.001, -0.236
UDP packet received: 142221.94544, 3,   0.262,  0.343,  9.568, 4,   0.001, -0.000,  0.001, 5,  36.225, 13.744,-32.813, 81, 271.150, -1.920,  1.510, 82,  -0.015, -0.002, -0.236
UDP packet received: 142222.06481, 3,   0.269,  0.321,  9.568, 4,  -0.001, -0.000, -0.001, 5,  36.225, 13.556,-32.156, 81, 271.150, -1.920,  1.530, 82,  -0.003, -0.002, -0.236
UDP packet received: 142222.30357, 3,   0.267,  0.345,  9.568, 5,  35.925, 14.700,-33.225, 81, 271.150, -1.900,  1.530, 82,  -0.017,  0.000, -0.236
```

Рисунок 5.3 – Результат чтения UDP стрима

Видно, четко структурируемые строки, значения в которых разделены запятыми, что и соответствует CSV формату, а значит, чтение выполняется верно.

Теперь необходимо обрабатывать полученные данные и сохранять их в память в виде уникальных переменных. Сперва создадим все нужные переменные, т.к. обращаться к ним придется из различных функций готовой программы, то сделаем их глобальными. Для навигационной системы необходимы показания GPS и ориентации. Также, для возможных в будущем опытов с Инерциальной навигацией, сохраним показания акселерометра, гироскопа, датчика магнитного поля и линейные значения акселерометра. Названия переменных в коде представлены на рисунке 5.3.

```
double GPS_Position[3];               // 1 // [0] - Широта(latitude), [1] - Долгота(longitude)
double Accelerometer_data[3];         // 3
double Gyroscope_data[3];             // 4
double Magnetic_Field_data[3];        // 5
double Orientation_data[3];           // 81 // [0] - Угол относительно севера,
double Linear_Accelerometer_data[3];  // 82
```

Рисунок 5.4 – Переменные данных с датчиков

Для обработки данных сперва необходимо полученные строки разделять на лексемы. Наиболее удобный и простой способ сделать это – функция strtok().

Strtok – функция стандартной библиотеки языка программирования Си, для поиска лексем в строке. Последовательность вызовов функции разбивает строку на лексемы, разделенные символами разделителями. Функция возвращает указатель на первую найденную лексему в строке. Если лексем не найдено, то возвращается пустой указатель [13].

Для использования функции введем указатель типа char* strtokIndx. В него и будем записывать возвращаемое функцией значение. Теперь, пока указатель не указывает на пустоту, будем переводить найденную лексему в число типа Double при помощи функции atof(). Функция atof в языке программирования Си используется для приведения (конвертации) строки в числовой вид (к числовому типу). Описанный код представлен на рисунке 5.4.

```
  if (len > 0) incomingPacket[len] = 0;
  strtokIndx = strtok(incomingPacket, ",");
  while (strtokIndx != NULL) {
    double value = atof(strtokIndx);
    // ОБРАБОТКА ЗНАЧЕНИЙ ТИПА DOUBLE, СОХРАНЕНИЕ ИХ В ПАМЯТИ
    strtokIndx = strtok(NULL, ",");
  }
}
```

Рисунок 5.5 – Обработка полученного UDP сообщения

Smartphone IMU GPS подписывает значения с датчика задавая в начале них соответствующее число. 1 – GPS, 3 – акселерометр, 4 – гироскоп, 5 – датчик магнитного поля, 81 – компас, 82 – линейный акселерометр. Таким образом задача сводится к тому, чтобы определить первое число и исходя из него, записать последующие числа в нужные переменные. Запись данных GPS представлена на рисунке 5.4, аналогично остальные.

```
if (value == 1) {
strtokIndx = strtok(NULL, ",");
GPS_Position[0] = atof(strtokIndx);
strtokIndx = strtok(NULL, ",");
GPS_Position[1] = atof(strtokIndx);
strtokIndx = strtok(NULL, ",");
GPS_Position[2] = atof(strtokIndx);
```

Рисунок 5.6 – Запись GPS данных

5.4 Передача точки назначения

Проект беспилотного электрошасси предполагает движение по заданных GPS координатам, получаемым от пользователя. На этапе разработки прототипа и его отладки для этого удобно пользоваться последовательным COM портом. Это позволит как передавать данные с ПК при работе с мастерской, так и подключить на этот порт Bluetooth модуль HC-06 и продолжать работу не меняя прошивку.

Для запуска последовательного порта достаточно прописать функцию Serial.begin(), где передаваемым значением функции будет скорость передачи данных в бод. Стандартной скоростью для ESP32 является 115200 Бод, но HC-06 работает на скорости 9600, поэтому именно такую и зададим. Далее писать код будем в отдельной функции, аналогично работе в пункте 5.3. Вызывать функцию будем при условии, что в буфере последовательного порта есть данные для чтения. Выполнить такую проверку можно, вызвав функцию Serial.available, которая возвратит количество байт находящихся в буфере и доступных для чтения.

Для работы нам нужна функция, читающая буфер порта и возвращающая значение из него в виде числа типа double. Назовем функцию readDoubleFromSerial. Её работа будет построена на использовании стандартного класса языка C++ String. String позволяет реализовывать динамические массивы типа char, что крайне удобно для работы со строками, однако, значительно медленнее, чем работа со статическими массивами, но в нашем случае, функция будет выполняться всего раз, когда у программы не задана точка назначения, поэтому отсутствие быстродействия не является проблемой [14].

Сперва создадим пустой объект класса string и назовем его dataString, теперь, пока в буфере есть данные для чтения, будем записывать их в dataString, но если будет встречен символ пробела, то цикл прервется, таким образом можно будет отправляться сразу несколько точек назначения, разделяя данные пробелом, а чтение будет происходить по порядку и по мере необходимости. После завершения чтения полученную строку необходимо преобразовать в тип double, для этого можно воспользоваться методом класса string toDouble().

Теперь осталось написать условия для вызова функции в loop(). Это должно происходить в том случае, если на текущий момент задания нет. Для этого создадим глобальный флаг No_set_point и сразу же приравняем его к единице, т.к. при запуске шасси, у него однозначно не может быть задания. Здесь же создадим переменные для хранения данных точки назначения, ее широты и долготы, в формате double: latitude_set_point_deg и longitude_set_point_deg.

В случае, когда точка назначения не назначена, необходимо задать колесам нулевую скорость, а затем уже проверять буфер. Такой подход гарантирует, что при отсутствии задания, электрошасси останется неподвижным. После прочтения задания, флаг необходимо установить в нулевое состояние. Данные необходимо передавать в порядке: сперва широта, затем долгота. Полный код написанной программы представлен на рисунке 5.7.

```
bool No_set_point = 1;
double latitude_set_point_deg = 0.0;
double longitude_set_point_deg = 0.0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (No_set_point == 1) {
    Send_Left(0, 0);
    Send_Right(0, 0);
    if (Serial.available()) {
      latitude_set_point_deg = readDoubleFromSerial();
      longitude_set_point_deg = readDoubleFromSerial();
      No_set_point = 0;
    }
  }
}

double readDoubleFromSerial() {
  String dataString = "";
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    if (inChar == ' ') {
      break;
    }
    dataString += inChar;
  }
  return dataString.toDouble();
}
```

Рисунок 5.7 – Программа передачи точки назначения

5.5 Программа проектирования маршрута

Электрошасси, выполняемое в рамках этой работы, не предназначается для полностью самостоятельного движения в условиях городской местности и автоматического проектирования маршрута. Первая задача требует огромного количества датчиков: лидаров, сенсоров, дальномеров и даже систем с компьютерным зрением. Вторая должна решаться на более высоком уровне, например на смартфоне с доступом в GIS системам. Поэтому задача, поставленная в рамках этой работы, сводится к беспилотному движению из начальной точки в конечную без каких-либо дополнительных условий. Очевидно, что наиболее оптимальной траекторией такого пути, будет прямая линия, соединяющая начальную и конечные точки. Таким образом, задача беспилотного движения сводится к нахождению этой прямой, ее угла наклона и движения с таким же углом поворота.

Первой трудностью становится то, что планета Земля не является плоской, а значит прямую линию, работая с широтой и долготой в градусах – проблематично. Исправить это можно, перейдя из системы с градусами в систему x, y. Сделать это можно несколькими способами, но для нашей задачи подойдет и наиболее простой: простейшее преобразование координат широты и долготы в плоские координаты x,y, которое основывается на приближении Земли к сфере и использует цилиндрическую равнопромежуточную проекцию.

Ключевые принципы этого преобразования:

1. Сферическая модель Земли: Земля считается идеальной сферой с определенным радиусом.
2. Цилиндрическая проекция: на Землю, как будто, накладывается цилиндр, касающийся ее в экваторе.
3. Широта (угол от экватора) масштабируется линейно по оси y цилиндра.
4. Долгота (угол от нулевого меридиана) масштабируется линейно по оси x цилиндра, но с коррекцией на косинус широты.
5. Косинусная коррекция – чтобы учесть сферичность Земли, долгота умножается на косинус широты.
6. Косинус широты уменьшается к полюсам, что приводит к сжатию меридианов на карте к полюсам. Эта коррекция частично компенсирует искажение, возникающее при цилиндрической проекции.

Преобразование широты в координату Y осуществляется по формуле 5.1.

$$Y = R_{\text{земли}} * \text{ш} = R_{\text{земли}} * \frac{\text{ш}^{\circ} * \pi}{180} \tag{5.1}$$

Преобразование долготы в координату X осуществляется по формуле 5.2.

$$X = R_{\text{земли}} * \text{д} * \cos(\text{цш}) = R_{\text{земли}} * \frac{\text{д}^{\circ} * \pi}{180} * \cos\left(\frac{\text{цш}^{\circ} * \pi}{180}\right) \tag{5.2}$$

Где: $R_{\text{земли}}$ – радиус Земли, 6378160 м.;

ш – широта в радианах;

д – долгота в радианах;

цш – центральная широта в радианах.

Центральная широта при переводе широты и долготы в координаты x, y – это ключевой параметр картографической проекции, определяющий точку или линию касания/сечения проекции с Землей, масштаб и ориентацию координатной сетки. Выбор центральной широты влияет на искажения формы, площади, расстояний и направлений на карте, поэтому для корректного перевода координат необходимо знать используемую проекцию, ее центральную широту и другие параметры. Так как Благовещенск расположен на широте около 50.3°, то для наибольшей точности при переводе координат именно в районе этого города, оптимально выбрать центральную широту 50 [15].

В программе это реализовано функциями, представленными на рисунке 5.8. Также, для этого была подключена библиотека cmath.

```
double GPS_latitude_to_Y(double latitude_deg)  // latitude - широта, longitude - долгота
{
  double latitude_rad = latitude_deg * M_PI / 180;
  double GPS_Position_Y = radius_of_the_Earth * latitude_rad;
  return (GPS_Position_Y);
}

double GPS_longitude_to_X(double longitude_deg)  // latitude - широта, longitude - долгота
{
  double longitude_rad = longitude_deg * M_PI / 180;
  double GPS_Position_X = radius_of_the_Earth * longitude_rad * cos(central_latitude_rad);
  return (GPS_Position_X);
}
```

Рисунок 5.8 – Функции преобразования градусов в декартовы координаты

Теперь можно переводить в декартовы координаты, текущие GPS координаты и координаты точки назначения. Далее найдем разницу в координатах по осям X и Y как разность между точкой назначения и текущей позицией. Имея расстояния по каждой оси, можно вычислить дистанцию до цели как гипотенузу треугольника distance = sqrt(X_error * X_error + Y_error * Y_error). Остается лишь рассчитать угол движения. Сделать это по формуле угла между двумя векторами, где одним вектором будет вектор разницы в координатах, а другой вектор будет представлять идеальное направление на север. Последним шагом будет перевод полученного угла, в угол соответствующий движению по часовой стрелке, в том случае, если точка назначения находится на 2 или 3 четверти круга, т.е. X_error меньше нуля.

Для работы с векторами в C++ используется библиотека vector. Она позволяет реализовывать динамические массивы разных видов данных, нас интересует double. Создадим функцию передаваемыми значениями которой будут разницы в координатах, а возвращаемым значением, нужный нам угол. В функции будем создавать два вышеописанных вектора и вызывать еще одну функцию, вычисляющую угол между векторами. Внешний вид этих функций представлен на рисунке 5.9.

```
double orientation_set_point(double X_error, double Y_error) {
  std::vector<double> vec1{ 0, 100 };
  std::vector<double> vec2{ X_error, Y_error };
  double angle_set_point = angleBetweenVectors(vec1, vec2);
  if (X_error < 0) angle_set_point = 360 - angle_set_point;
  return angle_set_point;
}

double angleBetweenVectors(const std::vector<double>& A, const std::vector<double>& B) {
  double dotProduct = 0.0;  // Вычисление скалярного произведения
  for (size_t i = 0; i < A.size(); ++i) {
    dotProduct += A[i] * B[i];
  }
  double magnitudeA = 0.0;  // Вычисление длин векторов
  double magnitudeB = 0.0;
  for (double element : A) {
    magnitudeA += element * element;
  }
  magnitudeA = std::sqrt(magnitudeA);
  for (double element : B) {
    magnitudeB += element * element;
  }
  magnitudeB = std::sqrt(magnitudeB);
  double cosTheta = dotProduct / (magnitudeA * magnitudeB);  // Вычисление
                                                             // косинуса угла и самого угла
  double angle = std::acos(cosTheta);                        // Угол в радианах
  return angle * 180 / M_PI;                                 // Возвращение угла (в градусах)
}
```

Рисунок 5.9 – Функции нахождения угла движения

Проведем тестирование написанных алгоритмов. Для этого зададим в качестве задания 4 разные точки, расположенные на расстоянии от 200 до 400 м. от текущего местоположения и сравним полученные значения угла и дистанции, с данными из геоинформационной системы: Яндекс Карты.

Сперва попробуем выполнить вычисления самостоятельно согласно формулам 5.1 и 5.2. Координаты начальной точки: 50.295022, 127.565703. Координаты точки назначения 50.295197, 127.566687.

$$Y_{\text{нач}} = 6378160 * \frac{50.295197 * \pi}{180} = 5598850 \text{ м.} \tag{5.3}$$

$$X_{\text{нач}} = 6378160 * \frac{127.566687 * \pi}{180} \cos\left(\frac{50 * \pi}{180}\right) = 9128040 \text{ м.} \tag{5.4}$$

$$Y_{\text{назн}} = 6378160 * \frac{50.295540 * \pi}{180} = 5598890 \text{ м.} \tag{5.5}$$

$$X_{\text{назн}} = 6378160 * \frac{127.561185 * \pi}{180} \cos\left(\frac{50 * \pi}{180}\right) = 9127650 \text{ м.} \tag{5.6}$$

$$\Delta_Y = Y_{\text{назн}} - Y_{\text{нач}} = 5598890 - 5598850 = 40 \text{ м.} \tag{5.7}$$

$$\Delta_X = X_{\text{назн}} - X_{\text{нач}} = 9127650 - 9128040 = -390 \text{ м.} \tag{5.8}$$

$$\Delta = \sqrt{{\Delta_Y}^2 * {\Delta_X}^2} = \sqrt{40^2 * -390^2} = 391.15 \text{ м.} \tag{5.9}$$

$$\cos(\alpha) = \frac{\overrightarrow{(\Delta_X, \Delta_Y)} * \overrightarrow{(0,100)}}{(\Delta_X, \Delta_Y) * (0,100)} = \frac{\overrightarrow{(-390,40)} * \overrightarrow{(0,100)}}{(-390,40) * (0,100)} = 0.11 \tag{5.10}$$

$$\cos(0.11) = 83° \tag{5.11}$$

$$\Delta_X < 0 \to \varphi = 360 - 83 = 277° \tag{5.12}$$

Таким образом, полученные значения дистанции пути и угла движения составляют: 392,04 м и 277°. Теперь зададим точку назначения в программу и проанализируем на полученные значения. Вывод программы показан на рисунке 5.10.

```
GPS position : X=50.295197, Y=127.566687, Z=107.400000
GPS position in cartesian coordinates: X = 9128040.33877490, Y=5598855.90917345
Coordinates set point X=9127646.49979385, Y=5598894.64849594
X_error: -393.84
Y_error: 38.74
Distance to point: 395.74
Angle set point: 275.62
```

Рисунок 5.10 – Вывод программы расчета дистанции пути и угла движения

Видно, что решения полученные в результате работы программы крайне близки к ручным, погрешность между ними объясняется маленьким количеством знаком после запятой при ручном решении. Теперь сравним имеющиеся данные с данными из Яндекс карт. Для этого отметим на карте начальную и конечную точки, измерим расстояние между ними инструментом линейка, а угол определим графически, перенеся карту в Компас-3D. Карта с данными представлена на рисунке 5.11.

Рисунок 5.11 – сравнение полученных значений с значениями Яндекс карт

На рисунке красным отмечена реальная линия движения, а синим – линия движения, полученная вышеописанным способом. Бесспорно, разница между ними существенная, однако, пересчет линии движения осуществляется каждый цикл работы микроконтроллера, что будет увеличивать точность, по мере движения. Главная цель первого расчета – получение примерного угла движения, что и удалось достигнуть, т.к. разница между полученным углом и реальным составляет менее 5 градусов.

Аналогичным методом проверены еще 3 точки, результаты их проверки продемонстрированы на листе 6.

5.6 Программа вычисления задания скорости движения колес

Построив маршрут, согласно, алгоритму, описанному в пункте 5.5, необходимо начать по нему движение, для этого необходимо правильно задавать правильную скорость колес. За счет выставления скорости электрошасси способно регулировать свой угол поворота и двигаться вперед и назад, поэтому задачу вычисления скорости было решено разбить на 2 функции: speed_control() и steer_control(). Обе функции входящим значением получают уставку угла движения, а возвращают число типа int. Результат работы steer_control() записывается в переменную u_steer, а результат speed_control() в u_speed. Принцип работы этих переменных, такой же как переменных uSteer и uSpeed, описанных в пункте 4.2.

Алгоритм работы steer_control() следующий: функция получает уставку угла движения и вычисляет ошибку, между уставкой и текущим углом. Если ошибка оказывается меньше, чем -180, то к ней прибавляется 360. Если ошибка больше 180, то от нее вычитается 360. За счет этого ошибка угла не может быть больше 180, что и соответствует реальной геометрии углов на круге. Затем используется ПИД-регулятор, из библиотеки GyverPID.h. На этапе прототипа, регулятор настроен как П-регулятор с уставкой 1, однако использование готового библиотечного регулятора, а не ручная реализация П-регулятора, позволит в будущем легко экспериментировать с настройками программы, добавляя в регулятор интегральную и дифференциальную составляющие. В качестве уставки регулятор получает 0, а в качестве ошибки вычисленную ранее ошибку. Далее, возвращаемое регулятором значение, будет записано в переменную функции u_steer (не путать с переменной u_steer в loop()). Если возвращаемое регулятором значение будет меньше 60, а ошибка угла больше 20, или такие же значения со знаком минус, то u_steer будет искусственно повышен до 60, т.к. это минимальное значение управления, на которое реагируют колеса. Итоговое полученное значение u_steer функция возвратит в функцию loop.

Функция установки скорости гораздо проще. Она получает ошибку углу, и если ошибка больше 20, то скорость колес определяется как 0, если ошибка меньше 20, то скорость определяется как 100, а если меньше 10, то как 200. Таким образом, если шасси не направлено в сторону движения, то оно будет осуществлять разворот на месте. Если оно почти направлено в сторону движения, то начнет медленное движение вперед, корректируя угол в движении, а если направлено почти идеально, то начнет быстрое движение к цели. Внешний вид этих описанных функций представлен на рисунке 5.12. Весь код, написанный в рамках пункта 5, представлен в приложении А. Блок-схема написанной программы представлена на листе 5.

```
int speed_control(double angle_set_point) {
  double angle_error = angle_set_point - Orientation_data[0];
  if (angle_error > 180) angle_error = angle_error - 360;
  if ((-10 < angle_error) && (angle_error < 10)) return (200);
  else if ((-20 < angle_error) && (angle_error < 20)) return (100);
  else return (0);
}

int steer_control(double angle_set_point) {
  double angle_error = angle_set_point - Orientation_data[0];
  if (angle_error < -180) angle_error = angle_error + 360;
  // если angle_error
  // положительный, то двигаться нужно против часовой стрелки,
  if (angle_error > 180) angle_error = angle_error - 360;
  // если отрицательный, то по часовой
  Serial.print("angle error: ");
  Serial.println(angle_error);
  steer_regulator.setpoint = 0;
  steer_regulator.input = (angle_error);
  // если angle_error меньше 0,
  // то u_steer будет положительным(движение по часовой стрелке),
  // если angle_error 0, то u_steer будет отрицательным(против часовой)
  int u_steer = steer_regulator.getResultTimer();
  if (u_steer < 0 && u_steer > -60 && angle_error < -20) u_steer = -60;
  else if (u_steer > 0 && u_steer < 60 && angle_error > 20) u_steer = 60;
  Serial.print("u_steer: ");
  Serial.println(u_steer);
  return (u_steer);
}
```

Рисунок 5.12 – Функции задания скорости движения

5.7 Компиляция программы и её загрузка в ESP32

После завершения написания программного обеспечения, в среде разработки Arduino IDE, его необходимо компилировать и загрузить на микроконтроллер. Сделать это можно также средствами среды.

Перед компиляцией необходимо выбрать используемый нами микроконтроллер или плату разработки. Сделать это можно в разделе «Tools», под-раздел «Board». Так как микроконтроллеры ESP не входят в семейство микроконтроллеров AVR Arduino, то для них нужно установить дополнительные файлы. Сделать это можно, используя раздел «Board manager». Для использования плат разработок на основе микроконтроллера ESP32 отлично подходит библиотека esp32 от Espessif Systems. Внешний вид менеджера плат в среде Arduino IDE 2 представлен на рисунке 5.13. Если вы используете Arduino IDE версией ниже 2, то устанавливать библиотеки микроконтроллеров придется вручную, скачав их предварительно, с GitHub страницы разработчика.

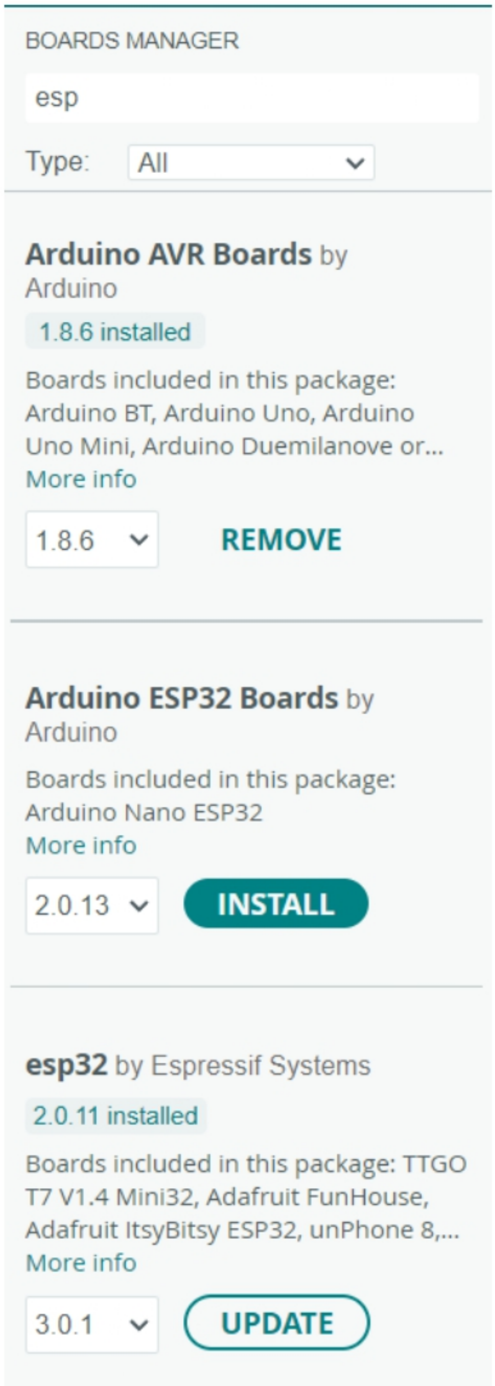


Рисунок 5.13 – Менеджер плат

Установив библиотеку esp32, необходимо выбрать нужную плату. В нашем случае это плата ESP-32-WROOM-DA-Module. Сделать это можно согласно рисунку 5.14.

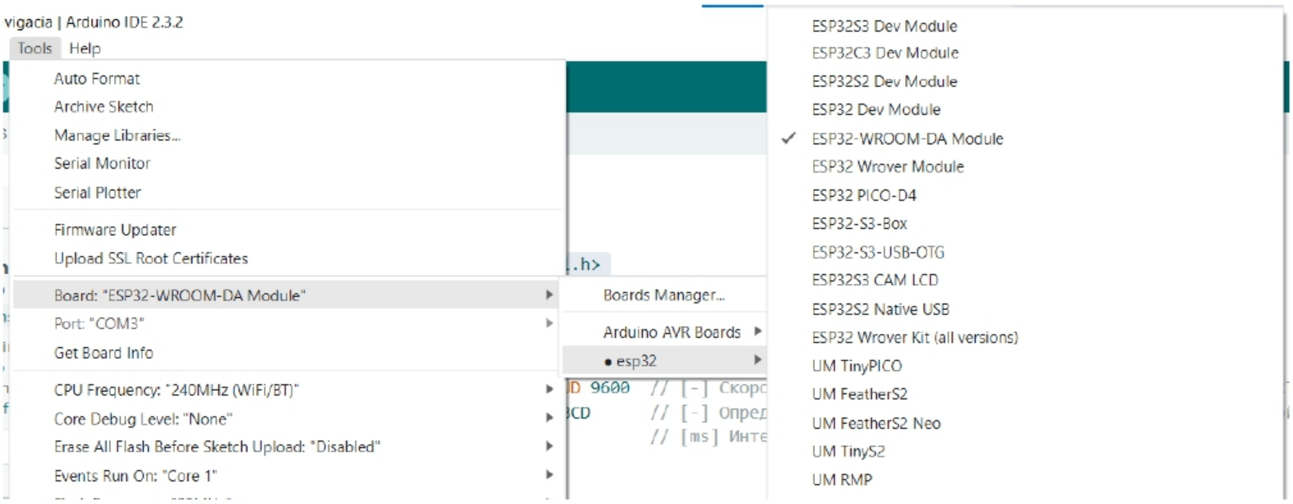


Рисунок 5.14 – Выбор платы разработки

Перед компиляцией необходимо убедиться, что все используемые в проекте библиотеки скачены на компьютер. В проекте используются следующие библиотеки: WiFi.h, WiFiUdp.h, cmath, vector, SoftwareSerial.h и GyverPID.h.. Первые 2 являются стандартными библиотеками ESP и не требуют отдельной установи. Библиотеки cmath и vector являются часть стандартного языка C++ и также не требуют установки, а вот последние 2 необходимо скачать. Сделать это можно при помощи «Library manager» в Arduino IDE 2 или вручную в более ранних версиях.

Оставшимся действием в программе является нажатие кнопки «Upload» расположенной в левом верхнем углу интерфейса. Кнопка «Upload» изображена на рисунке 5.15.

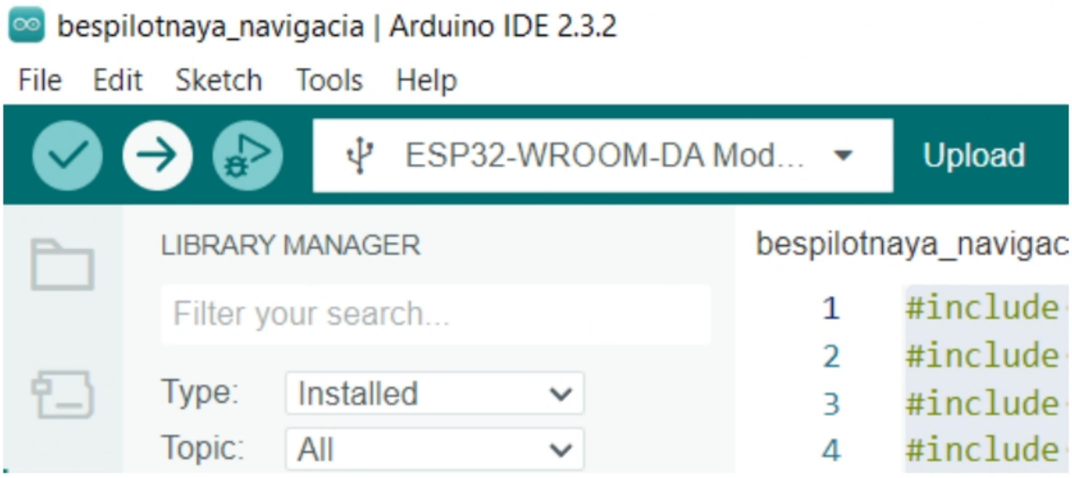


Рисунок 5.15 – Кнопка загрузки

После нажатия кнопки начнется компиляция программы, об этом будет свидетельствовать сообщение «Compiling sketch...» в правом нижнем углу интерфейса. После успешной компиляции начнется загрузка программы в микроконтроллер. Некоторые платы разработки обладают специальной кнопкой «Boot», без нажатия этой кнопки, загрузка программы не завершиться. Местонахождение кнопки на плате разработки изображено на рисунке 5.16.

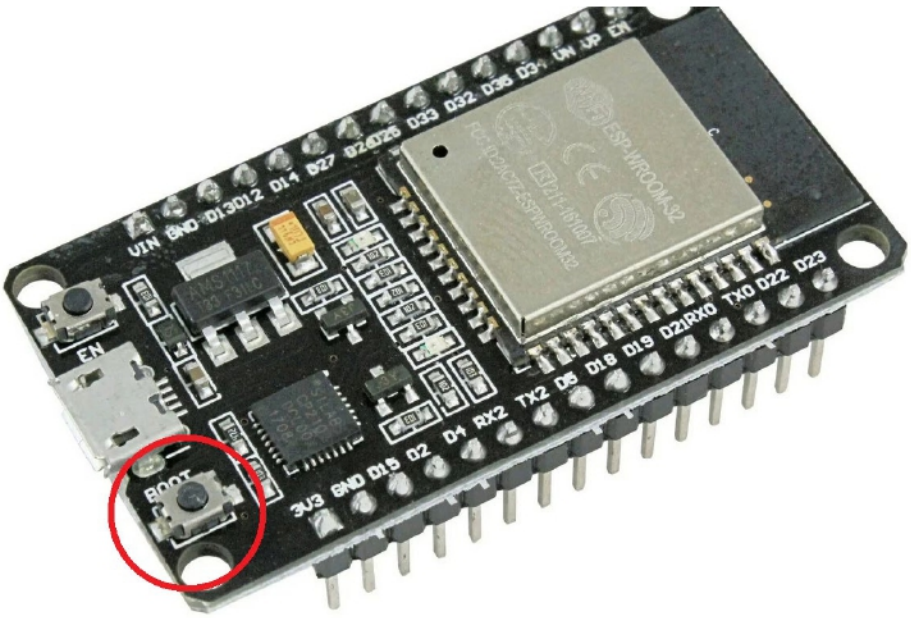


Рисунок 5.16 – Кнопка загрузки на плате

О успешности загрузки можно судить по отладочному сообщению в нижней части интерфейса программы. В случае, если при загрузке программы произошла ошибка, убедитесь, в правильности выбора последовательного порта.

Посмотреть доступные для загрузки последовательные порты можно в пункте «Port» раздела «Tools». Выбор последовательного порта изображен на рисунке 5.17.

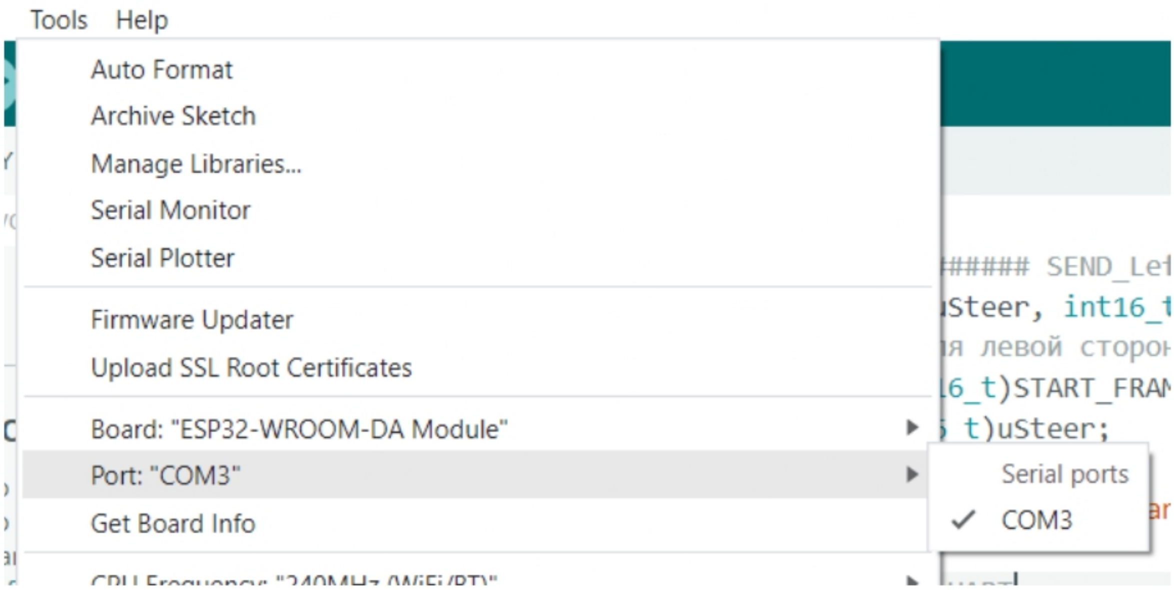


Рисунок 5.17 – Выбор последовательного порта

Если в «Port» нет подключенного устройства, то убедитесь в правильности установки драйвера для чипа USB- UART установленного на вашей плате. Дело в том, что для корректного взаимодействия устройства, подключенного к USB-порту компьютера, нужно предоставить операционной системе информацию о том, как с ним работать. Это достигается с помощью драйвера, который выполняет функцию переводчика. Например, драйвер USB-UART преобразует устройство в виртуальный COM-порт для обмена данными между устройством и компьютером. При подключении устройства через этот драйвер, чип устройства запрашивает операционную систему создать виртуальный COM-порт для общения с компьютером.

Наиболее популярные драйверы USB-UART это CH340, CP210 и CH9102. Выяснить какой драйвер установлен на вашей плате, узнать это можно на сайте производителя или напрямую изучив плату. Местонахождение драйвера USB-UART на плате ESP-32-WROOM-DA-Module показано на рисунке 5.18. Зрительно изучив выделенный чип можно установить его модель и скачать нужные драйвера. После правильной установки драйверов в разделе «Диспетчер устройств» операционной системы Windows, в пункте «Порты (COM и LPT)» можно будет найти наше устройство, аналогично рисунку 5.19.

Рисунок 5.18 – Программатор на плате разработки

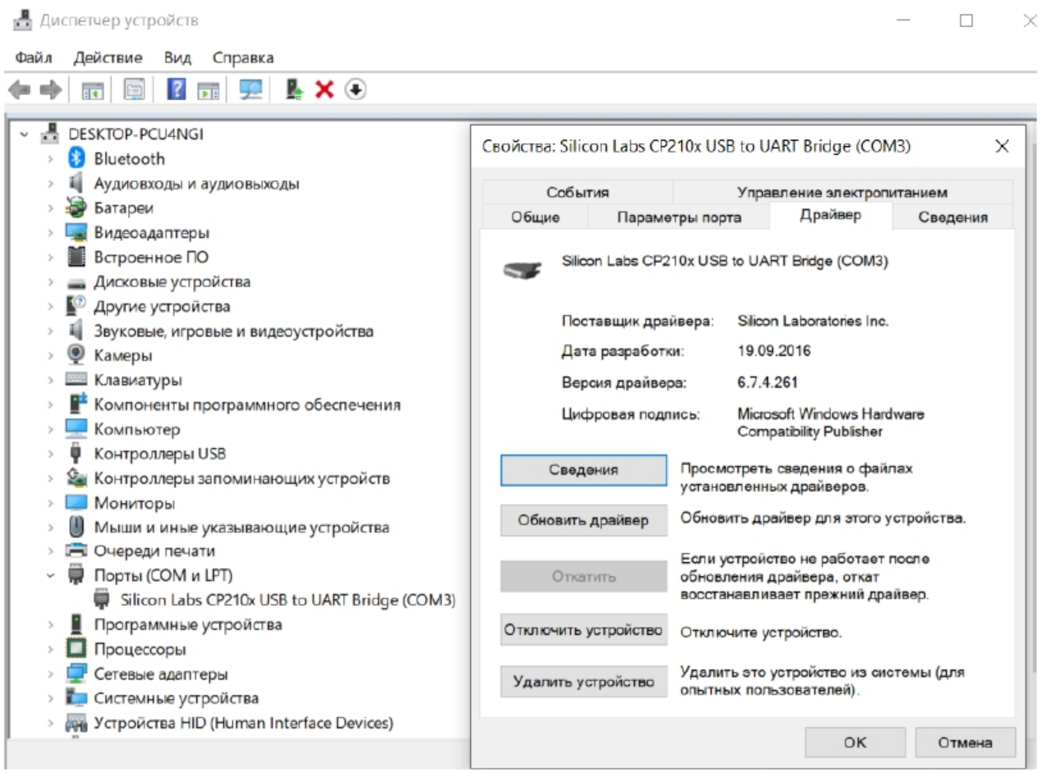


Рисунок 5.19 – Подключенная плата разработки в Диспетчере устройств

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Общие сведения по эксплуатации электрошасси

Эксплуатация электрошасси требует высокого уровня ответственности о ознакомленности с устройством. Правильная эксплуатация электрошасси не только обеспечивает эффективность работы, но и гарантирует безопасность для пользователей и окружающих. Тщательное изучение устройства и его особенностей является необходимым условием для успешного и безопасного использования.

Перед выездом электрошасси необходимо убедиться в том, что батарея заряжена. Для этого нужно открыть крышку блока управления и замерить напряжение на батареи мультиметром. Также узнать напряжение можно по показаниям вольтметра, установленного в блоке управления, но для этого необходимо включить один из драйверов управления. Внешний вид включенного мультиметра представлен на рисунке 6.1.

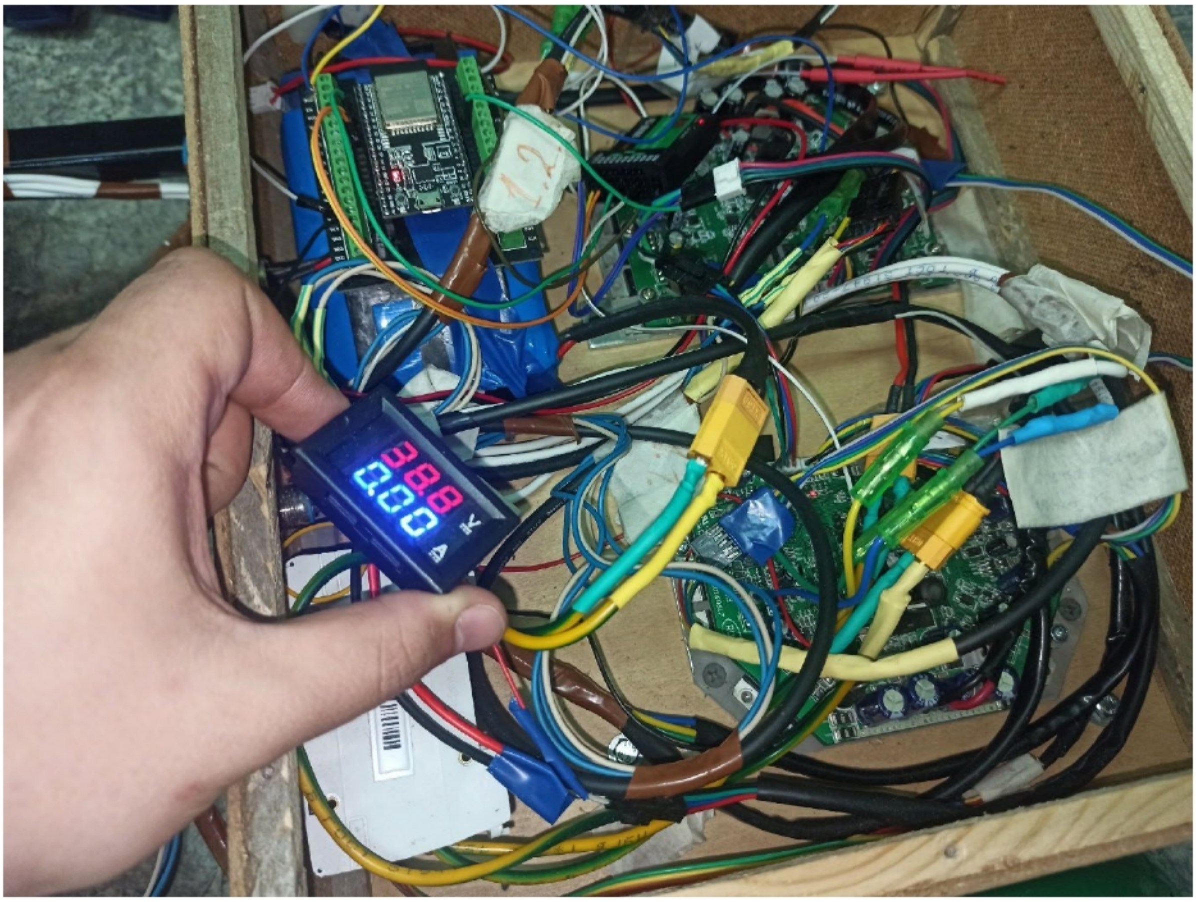


Рисунок 6.1 – Включенный мультиметр в блоке управления

Можно увидеть, что напряжение на батареи составляет 38.8 В, что соответствует практически полностью заряженной аккумуляторной батареи.

Включения драйверов осуществляется единичным нажатием на кнопки, расположенные на передней стороне блока управления. Внешний вид кнопок представлен на рисунке 6.2.

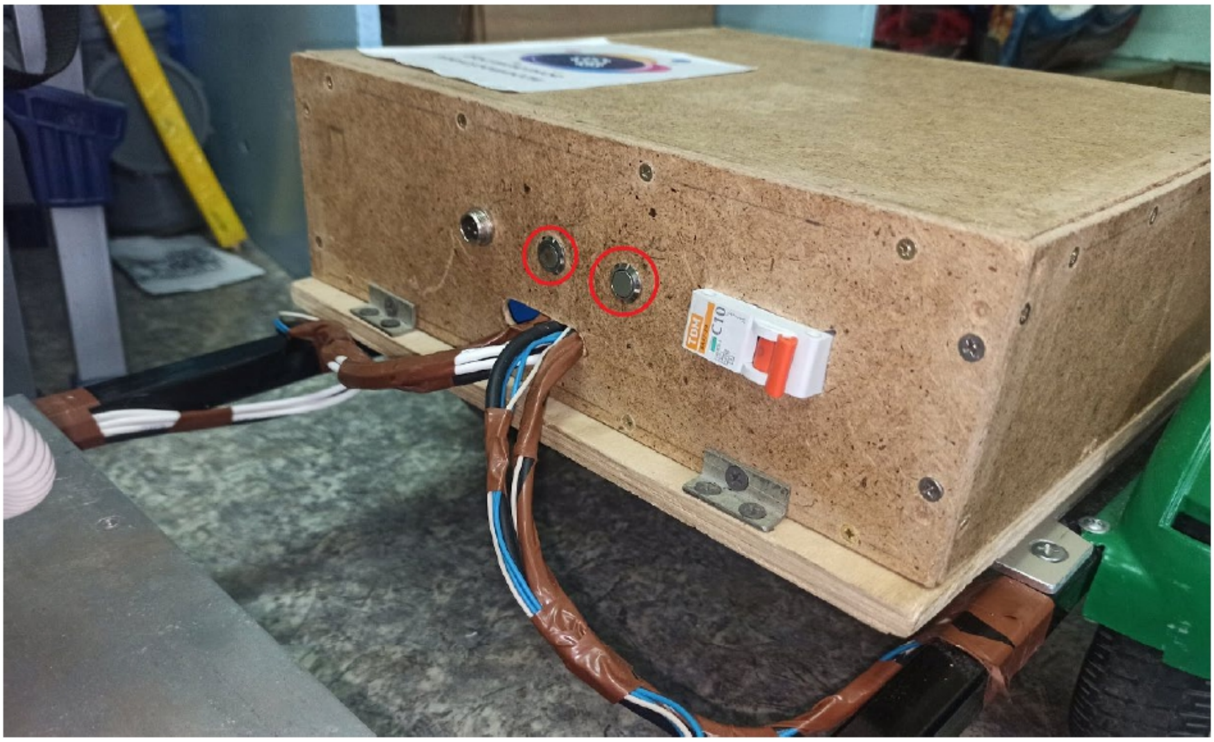


Рисунок 6.2 – Кнопки включения драйверов

Кнопка, расположенная ближе к автоматическому выключателю, включает драйвер управления колесами на левой стороне. Кнопка возле разъема разрядки, включает драйвер колес правой стороны. При включении драйвера, спикеры установленные на нем, издают непродолжительный звуковой сигнал, сигнализирующий о включении. Если включить драйверы нажатием кнопок не удалось, то проверьте, включен ли автоматический выключатель.

6.2 Работа в режиме ручного управления

После включения драйверов, необходимо включить пульт управления. Пульт получится включить только в стандартном положении всех стиков и тумблеров. Внешний вид пульта управления представлен на рисунке 6.3. Также на нем подписаны используемые стики и тумблеры и их назначение в проекте.

Рисунок 6.3 – Пульт управления

После включения пульта, необходимо удостовериться в том, что он подключился к приёмнику в электрошасси. Это можно понять по сообщениям на дисплее пульта, при соединении с приёмником будет написано его напряжение на входе, должно быть около 3.3 В. Следующим шагом необходимо, обязательно, перевести стик скорости правых колес в среднее положение. После этого можно переключить тумблер выбора режимов работы на щелчок вниз и окончательно перейти в режим ручного управления.

Теперь можно управлять электрошасси при помощи стиков на пульте. Для езды веред и назад используйте оба стика направляя их в одном направлении. Для поворота направляйте стики в противоположенные стороны.

6.3 Работа в режиме беспилотного управления

Для перехода в режим беспилотного управления необходимо перевести тумблер выбора режима в третье положение, подключиться к Bluetooth модулю со смартфона и запустить на нем приложение Bluetooth terminal. Теперь через терминал можно передать точку беспилотного назначения, передается она в виде 2 цифр, целая часть цифр от дробной отделяется точкой, цифры между собой разделяются пробелом.

С другого смартфона необходимо подключиться к точке доступа Wi-Fi, которую раздает ESP32, и запустить на нем приложение Smartphone IMU GPS, настроить работу приложения согласно рисунку 5.1 и нажать кнопку Switch stream.

Внимательно следите за работой электрошасси в этом режиме, помните, что оно не оборудовано датчиками препятствий и способно попасть в ДТП. Исходя из этого также не допускается проводить беспилотные запуски в местах скопления людей и/или автомобилей.

Помните, что в любой моменты вы можете переключиться назад в режимы ручного управления или паркинга (верхнее положение тумблера режимов).

7 БЕЗОПАСНОСТЬ И ЭКОЛОГИЧНОСТЬ

7.1 Безопасность

Безопасность беспилотного электрошасси – это комплексный вопрос, включающий в себя множество аспектов, начиная от прочности конструкции и заканчивая надежностью программного обеспечения. Необходимо обеспечить безопасность как самого электрошасси, так и людей, находящихся рядом с ним.

Первостепенное значение имеет прочная и надежная конструкция рамы. Она должна быть способна выдерживать ударные нагрузки, возникающие при столкновениях или падениях. Важно также обеспечить правильное и безопасное крепление всех элементов: двигателей, аккумуляторов, датчиков и других компонентов. Использование защитных кожухов для подвижных частей и электронных компонентов снижает риск травм при контакте с ними. Использование безопасных материалов, не поддерживающих горение, предотвратит возникновение пожара в случае аварии. Рама электрошасси выполнена из стального профиля, а все соединения надежно выполнены сваркой. Все электрические компоненты шасси установлены внутри корпуса, изолированного от окружающей среды, что защитит системы управления и батарею от внешних воздействий [16].

Ограничители скорости и крутящего момента должны быть предусмотрены для предотвращения опасных скоростей и чрезмерных усилий, которые могут привести к поломке или потере управления. В программе управления электрошасси для этого предусмотрена переменная speed_max, позволяющая быстро менять максимальную скорость движения.

Необходимо предусмотреть дублирование критически важных элементов, таких как датчики или блоки управления, чтобы обеспечить работоспособность электрошасси в случае выхода одного из них из строя. Система самодиагностики должна быть интегрирована для обнаружения ошибок и принятия безопасных действий, например, остановки движения или отправки предупреждающих сигналов оператору. На текущий момент в электрошасси не предусмотрено дублирование элементов управления, однако на программном уровне реализованы системы защиты на случай выхода из строя двигателей и перегрева плат управления.

Программное обеспечение должно быть тщательно протестировано и защищено от внешнего вмешательства, которое может привести к некорректной работе или несанкционированному управлению электрошасси, для этого точка доступа ESP32 и Bluetooth должны быть защищены надежным паролем.

Особое внимание следует уделить безопасности литий-ионных аккумуляторов, используемых в электрошасси. Они обладают высокой плотностью энергии, но при этом представляют риск пожара или взрыва при неправильной эксплуатации. Установленная аккумуляторная батарея оснащена защитами от короткого замыкания, перегрева и перезаряда.

Необходимо использовать специальные корпуса для защиты аккумуляторов от ударов, влаги и перегрева. Важно также внедрить системы управления зарядом, которые будут постоянно отслеживать температуру, ток и напряжение аккумулятора, предотвращая его перегрев или глубокий разряд. Системы охлаждения помогут отводить тепло от аккумулятора, снижая риск его перегрева. Механизмы аварийного отключения должны быть предусмотрены для отключения аккумулятора в случае превышения допустимых параметров. В качестве такого механизма в электрошасси установлен автоматический выключатель с номинальным током 10 А. Тумблер выключателя вынесен на наружную стенку корпуса и позволяет экстренно отключать электрошасси в случае неисправности.

Безопасное движение электрошасси в реальных условиях требует использования датчиков для определения препятствий. Лидары, сонары и камеры помогут распознавать объекты на пути движения и своевременно останавливать электрошасси. Подобные системы должны быть обязательно интегрированы в электрошасси на последующих стадиях его развития.

Система торможения должна быть быстрой и надежной, способной остановить электрошасси в экстренной ситуации. Алгоритмы избегания столкновений, используемые при планировании маршрута, помогут выбрать оптимальный путь и избежать опасных ситуаций. Для обеспечения безопасности пешеходов необходимо внедрить системы распознавания пешеходов и предупреждения о их появлении.

Тестирование электрошасси должно проходить только на специальных полигонах при участии подготовленного персонала, тщательно ознакомленного с пунктом 6 «Руководство пользователя».

Разработка беспилотного электрошасси требует комплексного подхода к вопросам безопасности, включающего в себя прочную конструкцию, надежное программное обеспечение и систему защиты от ошибок. Особое внимание следует уделить безопасности аккумуляторов, датчиков и элементов управления, чтобы сделать электрошасси максимально безопасным как для самого себя, так и для окружающих [17].

7.2 Экологичность

Экологичность современной техники – это не просто модный тренд, а важный аспект его разработки. Стремление к экологичности в этом случае означает минимизацию негативного воздействия на окружающую среду во всех аспектах жизненного цикла электрошасси: от производства до утилизации.

Ключевым фактором экологичности является энергоэффективность. Использование эффективных двигателей, таких как BLDC, с рекуперативным торможением позволяет сократить потребление энергии и снизить выбросы вредных веществ. Оптимизация аэродинамики для скоростных моделей также повышает энергоэффективность. Снижение массы электрошасси за счет использования легких материалов, таких как композиты, также положительно сказывается на потреблении энергии [18].

Выбор экологически чистых материалов – это важный шаг в повышении экологичности беспилотного электрошасси. Биоразлагаемые материалы, такие как пластики на основе биополимеров или растительных волокон, разлагаются без вреда для окружающей среды. Использование переработанных материалов сводит к минимуму потребление природных ресурсов и снижает количество отходов.

Бесшумные двигатели, такие как BLDC, значительно снижают шум при работе электрошасси, что делает его более экологичным, особенно в городской среде. Эффективная шумоизоляция также играет важную роль в минимизации шума.

Экологичная утилизация является заключительным этапом жизненного цикла электрошасси. Важно проектировать электрошасси таким образом, чтобы его детали легко разбирались и сортировались для переработки.

Переработка таких элементов, как аккумуляторы, пластик и металлы, позволяет свести к минимуму образование отходов и повторно использовать ценные ресурсы.

Экологичность беспилотного электрошасси – это комплексный подход, который должен быть реализован на всех этапах жизненного цикла устройства. От выбора материалов и энергоэффективных двигателей до переработки деталей и минимизации шума – все это влияет на экологичность электрошасси. При проектировании итогового варианта электрошасси, годного для выпуска на массовый рынок, необходимо учитывать все современные критерии экологичности [19].

7.3 Чрезвычайные ситуации

Несмотря на наличие защиты у литий-ионного аккумулятора, важно быть готовым к ситуации возгорания. Из-за возможности перегрева, короткого замыкания или разгерметизации батареи Li-ion, существует риск возгорания. Чтобы уменьшить вероятность возгорания, литиевые аккумуляторы оборудованы BMS-платами для мониторинга и защиты. В случае отклонения параметров от нормы, эти платы могут отключить аккумулятор от зарядного устройства или нагрузки.

Для тушения горящего лития не следует использовать воду, так как это может привести к опасному распаду воды на горючие газы. Песок, углекислотные или обычные порошковые огнетушители также не подходят. Вместо этого для тушения литиевых батарей рекомендуется использовать порошковые огнетушители класса D с наполнителями, такими как графит, нитрид бора или карборунд. При возгорании на открытом пространстве можно просто контролировать ситуацию и дождаться естественного прекращения горения.

При признаках возгорания литиевой батареи, но до появления огня, ее можно поместить в негорючую емкость и плотно закрыть. При горении лития важно избегать вдыхания выделяющихся токсичных газов. При использовании порошковых огнетушителей класса D необходимо дождаться полного выгорания и остывания аккумулятора. Соблюдая правила эксплуатации, можно минимизировать риск возгорания, а в случае необходимости тушения пожара, следует следовать инструкциям и действовать спокойно [20].

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной выпускной квалификационной работы было разработано электрошасси с ручным режимом управления через RC пульт и беспилотным режимом работы с движением по заданным GPS координатам. Разработанное устройство прошло первичные тестирования и показало работоспособность.

В ходе работы были изучены принципы работы беспроводных технологий Bluetooth, RC PWM и Wi-Fi UDP, а также особенности их применения и реализации в микроконтроллерной технике. Были разработаны аппаратная и программная части электрошасси, обеспечивающие его движение и управление, как ручное, так и в режиме автопилота.

Результаты данной ВКР могут быть использованы для дальнейшего развития и совершенствования разработанного электрошасси. Дальнейшее развитие электрошасси должно быть направлено на разработку подвески и совершенствование рамы, добавление к шасси экстероцептивных датчиков и написание алгоритмов беспилотного движения с опорой на них.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. What is a BLDC Motor in a Washing Machine? [Электронный ресурс] – Режим доступа: https://www.dumblittleman.com/bldc-motor – 25.03.24.

2. 6-шаговая коммутация BLDC моторов. [Электронный ресурс] – Режим доступа: https://habr.com/ru/articles/745372 – 25.03.24.

3. Векторное управление электродвигателем «на пальцах». [Электронный ресурс] – Режим доступа: https://habr.com/ru/companies/npf_vektor/articles/367653 – 25.03.24.

4. Hoverboard firmware hack FOC. [Электронный ресурс] – Режим доступа: https://github.com/EFeru/hoverboard-firmware-hack-FOC – 25.03.24.

5. ESP32 Series Datasheet. [Электронный ресурс] – Режим доступа: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf – 25.03.24.

6. Learn about Bluetooth. Bluetooth Technology. [Электронный ресурс] – Режим доступа: https://www.bluetooth.com/learn-about-bluetooth/tech-overview – 25.03.24.

7. Arduino core for the ESP32. [Электронный ресурс] – Режим доступа: https://github.com/espressif/arduino-esp32 – 25.03.24.

8. Read RC PWM signal with Arduino. [Электронный ресурс] – Режим доступа: https://quadmeup.com/read-rc-pwm-signal-with-arduino – 25.03.24.

9. Бородко, А. В. О навигационном обеспечении транспортного комплекса с использованием спутниковых систем ГЛОНАСС/GPS / А. В. Бородко // Геодезия и картография. – 2007. – № 8. – С. 4-11.

10. Smartphone IMU GPS. An Android App Dealing with Smartphone Sensors. [Электронный ресурс] – Режим доступа: https://sourceforge.net/projects/smartphone-imu – 25.03.24.

11. Ермолова, В. В. Архитектура системы обмена сообщений в немаршрутизируемой сети / В. В. Ермолова, Ю. П. Преображенский // Вестник Воронежского института высоких технологий. – 2010. – № 7. – С. 79-81.

12. Y. Shafranovich. Общий формат и MIME-тип для файлов значений, разделенных запятыми (CSV). SolidMatrix Technologies, Inc. Internet Society – 2005. – С. 8.

13. Рацеев, С. М., Череватенко О. И. Алгоритмы разбиения строки на лексемы в языке программирования Си // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. – 2012. – № 4 (110). – С. 6-8.

14. Магамедова, А. З. Функции для работы с символьными массивами и методы класса string в языке программирования C++ / А. З. Магамедова, А. М. Дадаев // Информатизация образования и науки. – 2016. – № 3(31). – С. 161-165.

15. Coordinate transformation tool. [Электронный ресурс] – Режим доступа: https://www.ordnancesurvey.co.uk/gps/transformation/ – 12.06.24

16. Сергеев, С. Ф. Комплексная безопасность робототехнических систем: человеческий фактор / С. Ф. Сергеев, А. С. Сергеева // Вопросы оборонной техники. Серия 16: Технические средства противодействия терроризму. – 2016. – № 11-12(101-102). – С. 31-35.

17. Басан, Е. С. Анализ безопасности систем управления робототехническими системами / Е. С. Басан, А. С. Грицынин // Перспектива-2019: Материалы VIII Всероссийской молодежной школы-семинара по проблемам информационной безопасности, Таганрог, 10–13 октября 2019 года. – Таганрог: ООО «Издательство «Лукоморье», 2019. – С. 62-76.

18. Никишин, С. А. Проблемы организации и развития инфраструктуры для экологичных видов транспорта в России / С. А. Никишин, Е. А. Сухинина, С. Ф. Дядченко // Градостроительство и архитектура. – 2022. – Т. 12, № 2(47). – С. 175-185. – DOI 10.17673/10.17673/Vestnik.2022.02.22.

19. Константинова, А. А. Экологическое проектирование / А. А. Константинова // Международный студенческий научный вестник. – 2017. – № 5. – С. 86.

20. Федеральный закон от 22.07.2008 N 123-ФЗ (ред. от 25.12.2023) "Технический регламент о требованиях пожарной безопасности". [Электронный ре-

сурс] – Режим доступа: https://10.mchs.gov.ru/uploads/resource/2023-02-10/3f7fce14bb9a26668dbb0f602a41efee.pdf – 12.06.24

21. Бендик В. И. Разработка системы управления электроприводом мотор-колеса // В. И. Бендик, В. Л. Русинов // Материалы XXXIII научной конференции Амурского государственного университета «День науки», Благовещенск, 18 апреля 2024 года. – С. 92-93.

ПРИЛОЖЕНИЕ А

Код программы

```
#include <WiFi.h>
#include <WiFiUdp.h>
#include <cmath>
#include <vector>
#include <SoftwareSerial.h>
#include <GyverPID.h>

#define HOVER_SERIAL_BAUD 9600  // [-] Скорость передачи в бодах для по-
следовательного порта UART (связь с Hoverboard)

#define START_FRAME 0xABCD      // [-] Определение начального кадра для
надежной последовательной связи (2 байта сообщения)
#define TIME_SEND 100           // [ms] Интервал опроса сообщений
#define CH2 32
#define CH3 33
#define drone_mode_switch 34

SoftwareSerial HoverSerial_Left(22, 23);   // RX, TX для левой стороны
SoftwareSerial HoverSerial_Right(18, 19);  // RX, TX для правой стороны
WiFiUDP Udp;
GyverPID steer_regulator(1, 0, 0);


// Глобальный переменные
uint8_t idx = 0;         // Индекс для нового указателя данных
uint16_t bufStartFrame;  // Буффер стартовых кадров
byte* p;                 // Объявление указателя для новых полученных
данных
byte incomingByte;       // Входящий байт
byte incomingBytePrev;   //Предыдущий вхоящий байт

unsigned long iTimeSend = 0;   //Время опроса на millis
int speed_min = 0;             // Минимальная скорость
int speed_max = 250;           //Максимальная скорость
unsigned long duration_2 = 0;  //Правый стик
unsigned long duration_3 = 0;  //Левый стик
int s2_F, s2_B, s3_F, s3_B;
//int iTest = 0;
//int iStep = SPEED_STEP;

typedef struct {
  uint16_t start;       //Старт 2 байта 0xABCD
  int16_t steer;        //Поворот
  int16_t speed;        //Скорость
  uint16_t checksum;    //Проверка суммы
```

```
} SerialCommand;        //Объект - передача данных
SerialCommand Command;  //Создание объекта

typedef struct {
  uint16_t start;             //Старт 2 байт 0xABCD
  int16_t cmd1;               //Команда 1
  int16_t cmd2;               //Команда 2
  int16_t speedR_meas;        //Измерение скорости правого колеса
  int16_t speedL_meas;        //Измерение скорости левого колеса
  int16_t batVoltage;         //Напряжение батареи
  int16_t boardTemp;          //Температура платы
  uint16_t cmdLed;            //Команда LED
  uint16_t checksum;          //Проверка суммы
} SerialFeedback;             //Объект - обратная связь
SerialFeedback Feedback;      //Создание объекта обратной связи
SerialFeedback NewFeedback;   //Создание объекта новой обратной связи

const char* ssid = "ESP-32-TEST";
const char* password = "056445870";
const double radius_of_the_Earth = 6378160;  // м
const double central_latitude_rad = 50 * M_PI / 180;

char incomingPacket[512];  // Буфер для входящего пакета

bool GPS_no_data = 0;
bool No_set_point = 1;

double GPS_Position[3];                // 1 // [0] - Широта(latitude), [1]
- Долгота(longitude)
double Accelerometer_data[3];          // 3
double Gyroscope_data[3];              // 4
double Magnetic_Field_data[3];         // 5
double Orientation_data[3];            // 81 // [0] - Угол относительно
севера,
double Linear_Accelerometer_data[3];  // 82
char* strtokIndx;                      // Указатель для strtok()


double GPS_Position_X;
double GPS_Position_Y;

double latitude_set_point_deg = 0.0;
double longitude_set_point_deg = 0.0;
double set_point_X = 0.0;
double set_point_Y = 0.0;
double angle_set_point = 0.0;
double distance = 0.0;
double X_error = 0.0;
double Y_error = 0.0;

int u_steer = 0;
```

```
int u_speed = 0;

void setup() {
  Serial.begin(9600);
  WiFi.softAP(ssid, password);  // Запуск точки доступа 192.168.4.1
  Serial.print("WiFi hot spot is ready, ESP IP address: ");
  Serial.println("192.168.4.1");
  Serial.println("UDP port to read : 8888");
  Udp.begin(8888);

  HoverSerial_Left.begin(HOVER_SERIAL_BAUD);
  HoverSerial_Right.begin(HOVER_SERIAL_BAUD);

  pinMode(CH2, INPUT);
  pinMode(CH3, INPUT);
  pinMode(drone_mode_switch, INPUT);

  steer_regulator.setDirection(REVERSE);  // направление регулирования
(NORMAL/REVERSE). ПО УМОЛЧАНИЮ СТОИТ NORMAL
  steer_regulator.setLimits(-200, 200);
}

void loop() {
  int pulse_In_drone_mode_switch = pulseIn(drone_mode_switch, HIGH);
  if (pulse_In_drone_mode_switch > 1100 && pulse_In_drone_mode_switch <
1600) hands_control();
  else if (pulse_In_drone_mode_switch > 1600) drone_mode();
  else if (pulse_In_drone_mode_switch < 1100 &&
pulse_In_drone_mode_switch > 800) {
    Send_Left(0, 0);
    Send_Right(0, 0);
    Serial.println("-----------------------");
    Serial.println("  Режим безопасности  ");
    Serial.println("-----------------------");
  } else {
    Send_Left(0, 0);
    Send_Right(0, 0);
    Serial.println("-----------------------");
    Serial.println("  Пульт не подключен  ");
    Serial.println("-----------------------");
  }
  drone_mode();
}

void drone_mode() {
  Serial.println("Беспилотное управление");
  Upd_read();
  if (GPS_Position[0] == 0 || GPS_Position[1] == 0) GPS_no_data = 1;
  else {
    GPS_no_data = 0;
    GPS_Position_Y = GPS_latitude_to_Y(GPS_Position[0]);
```

```
    GPS_Position_X = GPS_longitude_to_X(GPS_Position[1]);
  }
  if (No_set_point == 1) {
    Send_Left(0, 0);
    Send_Right(0, 0);
    if (Serial.available()) {  // Check if at least two data points are
available
      latitude_set_point_deg = readDoubleFromSerial();
      longitude_set_point_deg = readDoubleFromSerial();
      set_point_Y = GPS_latitude_to_Y(latitude_set_point_deg);
      set_point_X = GPS_longitude_to_X(longitude_set_point_deg);
      No_set_point = 0;
    } else No_set_point = 1;
  } else {
    if (GPS_no_data == 1) {
      Serial.println("----------------------");
      Serial.println("Получение геоинформации");
      Serial.println("----------------------");
      Send_Left(0, 0);
      Send_Right(0, 0);
    } else {
      X_error = (GPS_Position_X - set_point_X) * (-1);
      Y_error = (GPS_Position_Y - set_point_Y) * (-1);
      distance = sqrt(X_error * X_error + Y_error * Y_error);  // Гипоти-
нуза треугольника
      angle_set_point = orientation_set_point(X_error, Y_error);
      Serial.print("Angle set point ");
      Serial.println(angle_set_point);
      Serial.print("Orientation_data ");
      Serial.println(Orientation_data[0]);
      u_steer = steer_control(angle_set_point);
      u_speed = speed_control(angle_set_point);
      send_steer_and_speed(u_speed, u_steer);
      if ((GPS_Position[0] == latitude_set_point_deg && GPS_Position[1]
== longitude_set_point_deg) || distance < 5) {
        No_set_point = 1;
        Serial.println("---------------------------");
        Serial.println("ТОЧКА НАЗНАЧЕНИЯ ДОСТИГНУТА");
        Serial.println("---------------------------");
        delay(1000);
      }
    }
  }
  Print_data();
}

void send_steer_and_speed(int u_speed, int u_steer) {  // если u_steer >
0 - движение по часовой стрелке, если u_steer < 0 - против часовой
  int speed_left = u_speed + u_steer;
  int speed_right = u_speed - u_steer;
  Send_Left(speed_left * 2, 0);
```

```
  Send_Right(0, speed_right);
  Serial.print("speed_left: ");
  Serial.println(speed_left);
  Serial.print("speed_right: ");
  Serial.println(speed_right);
}


int speed_control(double angle_set_point) {
  double angle_error = angle_set_point - Orientation_data[0];
  if (angle_error > 180) angle_error = angle_error - 360;
  if ((-10 < angle_error) && (angle_error < 10)) return (200);
  else if ((-20 < angle_error) && (angle_error < 20)) return (100);
  else return (0);
}


int steer_control(double angle_set_point) {
  double angle_error = angle_set_point - Orientation_data[0];
  if (angle_error < -180) angle_error = angle_error + 360;  // если an-
gle_error положительный, то двигаться нужно против часовой стрелки,
  if (angle_error > 180) angle_error = angle_error - 360;   // если отри-
цательный, то по часовой
  Serial.print("angle error: ");
  Serial.println(angle_error);
  steer_regulator.setpoint = 0;
  steer_regulator.input = (angle_error);  // если angle_error меньше 0,
то u_steer будет положительным(движение по часовой стрелке),
  //                                     //если angle_error 0, то u_steer
будет отрицательным(против часовой)
  int u_steer = steer_regulator.getResultTimer();
  if (u_steer < 0 && u_steer > -60 && angle_error < -20) u_steer = -60;
  else if (u_steer > 0 && u_steer < 60 && angle_error > 20) u_steer = 60;
  Serial.print("u_steer: ");
  Serial.println(u_steer);
  return (u_steer);
}



double orientation_set_point(double X_error, double Y_error) {
  std::vector<double> vec1{ 0, 100 };
  std::vector<double> vec2{ X_error, Y_error };
  double angle_set_point = angleBetweenVectors(vec1, vec2);
  if (X_error < 0) angle_set_point = 360 - angle_set_point;
  return angle_set_point;
}

double readDoubleFromSerial() {
  String dataString = "";
  while (Serial.available()) {
```

```
    char inChar = (char)Serial.read();
    if (inChar == ' ') {
      break;
    }
    dataString += inChar;
  }
  return dataString.toDouble();
}

double GPS_latitude_to_Y(double latitude_deg)  // latitude - широта, lon-
gitude - долгота
{
  double latitude_rad = latitude_deg * M_PI / 180;
  double GPS_Position_Y = radius_of_the_Earth * latitude_rad;
  return (GPS_Position_Y);
}

double GPS_longitude_to_X(double longitude_deg)  // latitude - широта,
longitude - долгота
{
  double longitude_rad = longitude_deg * M_PI / 180;
  double GPS_Position_X = radius_of_the_Earth * longitude_rad *
cos(central_latitude_rad);
  return (GPS_Position_X);
}


void Upd_read() {
  int packetSize = Udp.parsePacket();
  if (packetSize) {
    int len = Udp.read(incomingPacket, 511);
    if (len > 0) incomingPacket[len] = 0;
    strtokIndx = strtok(incomingPacket, ",");
    while (strtokIndx != NULL) {
      double value = atof(strtokIndx);
      if (value == 3) {
        strtokIndx = strtok(NULL, ",");
        Accelerometer_data[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Accelerometer_data[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Accelerometer_data[2] = atof(strtokIndx);
      }
      if (value == 1) {
        strtokIndx = strtok(NULL, ",");
        GPS_Position[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        GPS_Position[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        GPS_Position[2] = atof(strtokIndx);
      } else if (value == 4) {
```

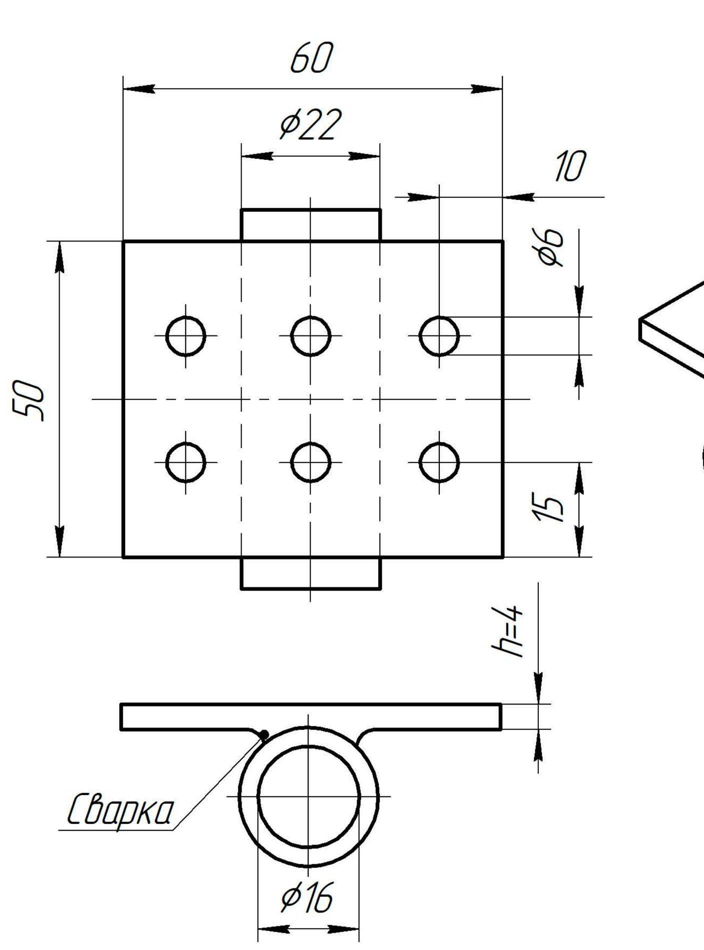
```
        strtokIndx = strtok(NULL, ",");
        Gyroscope_data[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Gyroscope_data[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Gyroscope_data[2] = atof(strtokIndx);
      } else if (value == 5) {
        strtokIndx = strtok(NULL, ",");
        Magnetic_Field_data[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Magnetic_Field_data[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Magnetic_Field_data[2] = atof(strtokIndx);
      } else if (value == 82) {
        strtokIndx = strtok(NULL, ",");
        Linear_Accelerometer_data[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Linear_Accelerometer_data[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Linear_Accelerometer_data[2] = atof(strtokIndx);
      } else if (value == 81) {
        strtokIndx = strtok(NULL, ",");
        Orientation_data[0] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Orientation_data[1] = atof(strtokIndx);
        strtokIndx = strtok(NULL, ",");
        Orientation_data[2] = atof(strtokIndx);
      }
      strtokIndx = strtok(NULL, ",");
    }
  }
}

void Print_data() {
  Serial.write(27);      // ESC command
  Serial.print("[2J");   // clear screen command
  if (GPS_no_data == 0) {
    Serial.print("GPS position : X=");
    Serial.print(GPS_Position[0], 6);
    Serial.print(", Y=");
    Serial.print(GPS_Position[1], 6);
    Serial.print(", Z=");
    Serial.print(GPS_Position[2], 6);
    Serial.println();
    Serial.print("GPS position in cartesian coordinates: X = ");
    Serial.print(GPS_Position_X, 8);
    Serial.print(", Y=");
    Serial.print(GPS_Position_Y, 8);
    Serial.println();
  } else {
    Serial.println("----------------------");
```

```
  Serial.println("GPS position not found");
  Serial.println("----------------------");
}
Serial.print("Accelerometer data : X=");
Serial.print(Accelerometer_data[0]);
Serial.print(", Y=");
Serial.print(Accelerometer_data[1]);
Serial.print(", Z=");
Serial.print(Accelerometer_data[2]);
Serial.println();
Serial.print("Gyroscope data : X=");
Serial.print(Gyroscope_data[0]);
Serial.print(", Y=");
Serial.print(Gyroscope_data[1]);
Serial.print(", Z=");
Serial.print(Gyroscope_data[2]);
Serial.println();
Serial.print("Magnetic Field data : X=");
Serial.print(Magnetic_Field_data[0]);
Serial.print(", Y=");
Serial.print(Magnetic_Field_data[1]);
Serial.print(", Z=");
Serial.print(Magnetic_Field_data[2]);
Serial.println();
Serial.print("Linear Accelerometer data : X=");
Serial.print(Linear_Accelerometer_data[0]);
Serial.print(", Y=");
Serial.print(Linear_Accelerometer_data[1]);
Serial.print(", Z=");
Serial.print(Linear_Accelerometer_data[2]);
Serial.println();
Serial.print("Orientation data : X=");
Serial.print(Orientation_data[0]);
Serial.print(", Y=");
Serial.print(Orientation_data[1]);
Serial.print(", Z=");
Serial.print(Orientation_data[2]);
Serial.println();
Serial.print("Coordinates set point X=");
Serial.print(set_point_X, 8);
Serial.print(", Y=");
Serial.print(set_point_Y, 8);
Serial.println();
Serial.print("X_error: ");
Serial.print(X_error);
Serial.println();
Serial.print("Y_error: ");
Serial.print(Y_error);
Serial.println();
Serial.print("Distance to point: ");
Serial.print(distance);
```

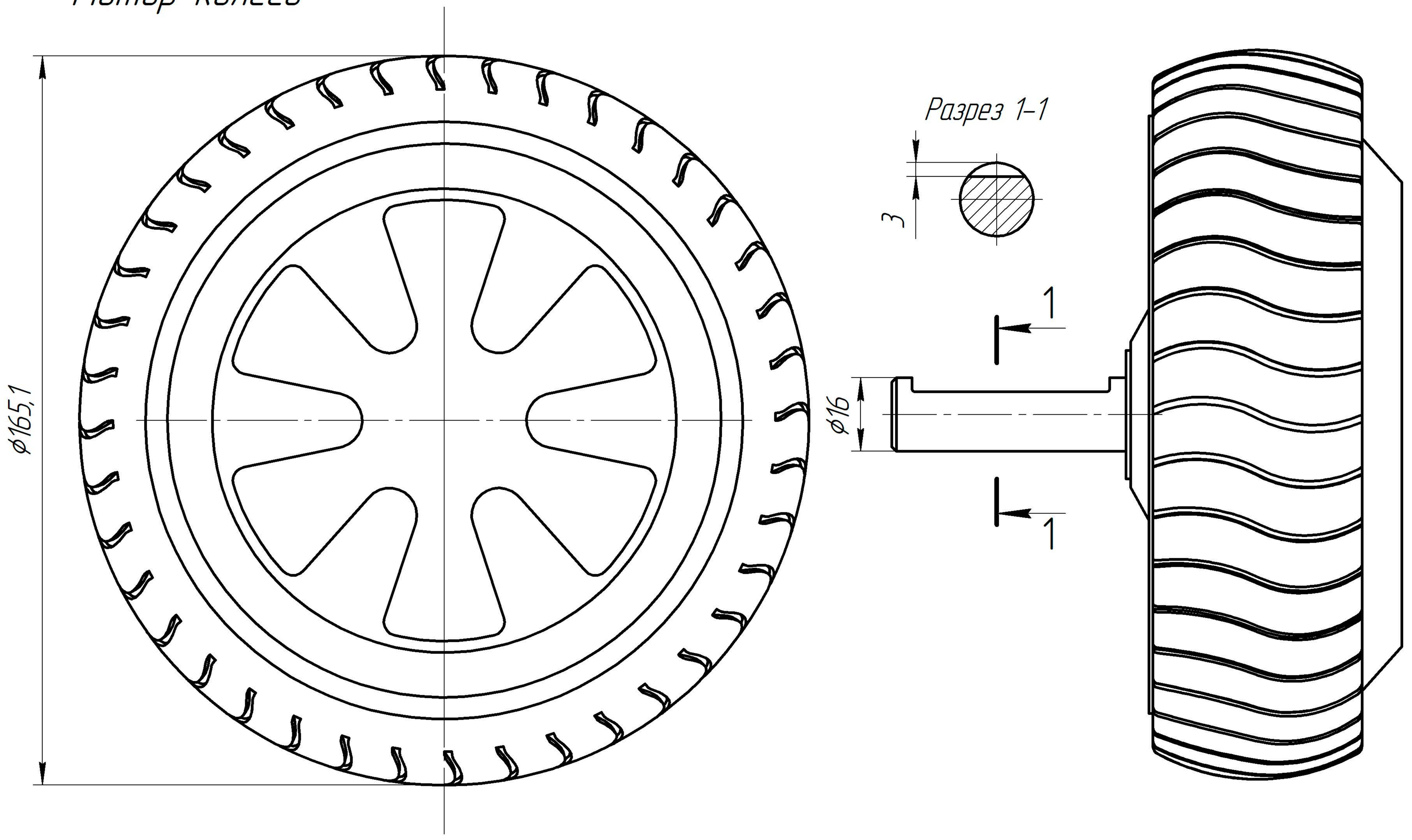
```
  Serial.println();
  Serial.print("Angle set point: ");
  Serial.print(angle_set_point);
  Serial.println();
  Serial.print("U_steer: ");
  Serial.print(u_steer);
  Serial.println();
  Serial.print("U_speed: ");
  Serial.print(u_speed);
  Serial.println();
  return;
}


double angleBetweenVectors(const std::vector<double>& A, const
std::vector<double>& B) {

  // Вычисление скалярного произведения
  double dotProduct = 0.0;
  for (size_t i = 0; i < A.size(); ++i) {
    dotProduct += A[i] * B[i];
  }
  // Вычисление длин векторов
  double magnitudeA = 0.0;
  double magnitudeB = 0.0;
  for (double element : A) {
    magnitudeA += element * element;
  }
  magnitudeA = std::sqrt(magnitudeA);
  for (double element : B) {
    magnitudeB += element * element;
  }
  magnitudeB = std::sqrt(magnitudeB);
  // Вычисление косинуса угла и самого угла
  double cosTheta = dotProduct / (magnitudeA * magnitudeB);
  double angle = std::acos(cosTheta);  // Угол в радианах
  // Возвращение угла (можно преобразовать в градусы)
  return angle * 180 / M_PI;
}

void hands_control() {
  Serial.println("Ручное управление");
  duration_2 = pulseIn(CH2, HIGH);  // продолжительность импульса, соот-
ветсвует углу отклонения // задает скорость вращения правых колес
  duration_3 = pulseIn(CH3, HIGH);  //

  s2_F = map(duration_2, 1600, 2000, speed_min, speed_max);   //Правый
стик вперед
  s2_B = map(duration_2, 1400, 1000, speed_min, -speed_max);  //Правый
стик назад
```

```
  s3_F = map(duration_3, 1600, 2000, speed_min, speed_max);   //Левый
стик вперед
  s3_B = map(duration_3, 1400, 1000, speed_min, -speed_max);  //Левый
стик назад

  if (duration_3 > 1590) {
    Send_Left(s3_F * 2, 0);  //Вперед
  } else if (duration_3 < 1410 && duration_3 > 990) {
    Send_Left(s3_B * 2, 0);  //Назад
  } else {
    Send_Left(0, 0);
  }
  if (duration_2 > 1590) {  //Вращение вправо
    Send_Right(0, s2_F);
  } else if (duration_2 < 1410 && duration_2 > 990) {
    //Вращение влево/ Умножение на 2, так как скорость распределяется на
2 колеса
    Send_Right(0, s2_B);
  } else {
    Send_Right(0, 0);
  }
}

// ########################## SEND_Left ##########################
void Send_Left(int16_t uSteer, int16_t uSpeed) {
  // Создание команды для левой стороны
  Command.start = (uint16_t)START_FRAME;
  Command.steer = (int16_t)uSteer;
  Command.speed = (int16_t)uSpeed;
  Command.checksum = (uint16_t)(Command.start ^ Command.steer ^ Com-
mand.speed);

  // Передача данных по UART
  HoverSerial_Left.write((uint8_t*)&Command, sizeof(Command));
}

// ########################## SEND_Right ##########################
void Send_Right(int16_t uSteer, int16_t uSpeed) {
  // Создание команды для правой стороны
  Command.start = (uint16_t)START_FRAME;
  Command.steer = (int16_t)uSteer;
  Command.speed = (int16_t)uSpeed;
  Command.checksum = (uint16_t)(Command.start ^ Command.steer ^ Com-
mand.speed);

  // Передача данных по UART
  HoverSerial_Right.write((uint8_t*)&Command, sizeof(Command));
}

// ########################## END RECEIVE FRONT
// ########################## END ##########################
```

Крепление колеса к раме

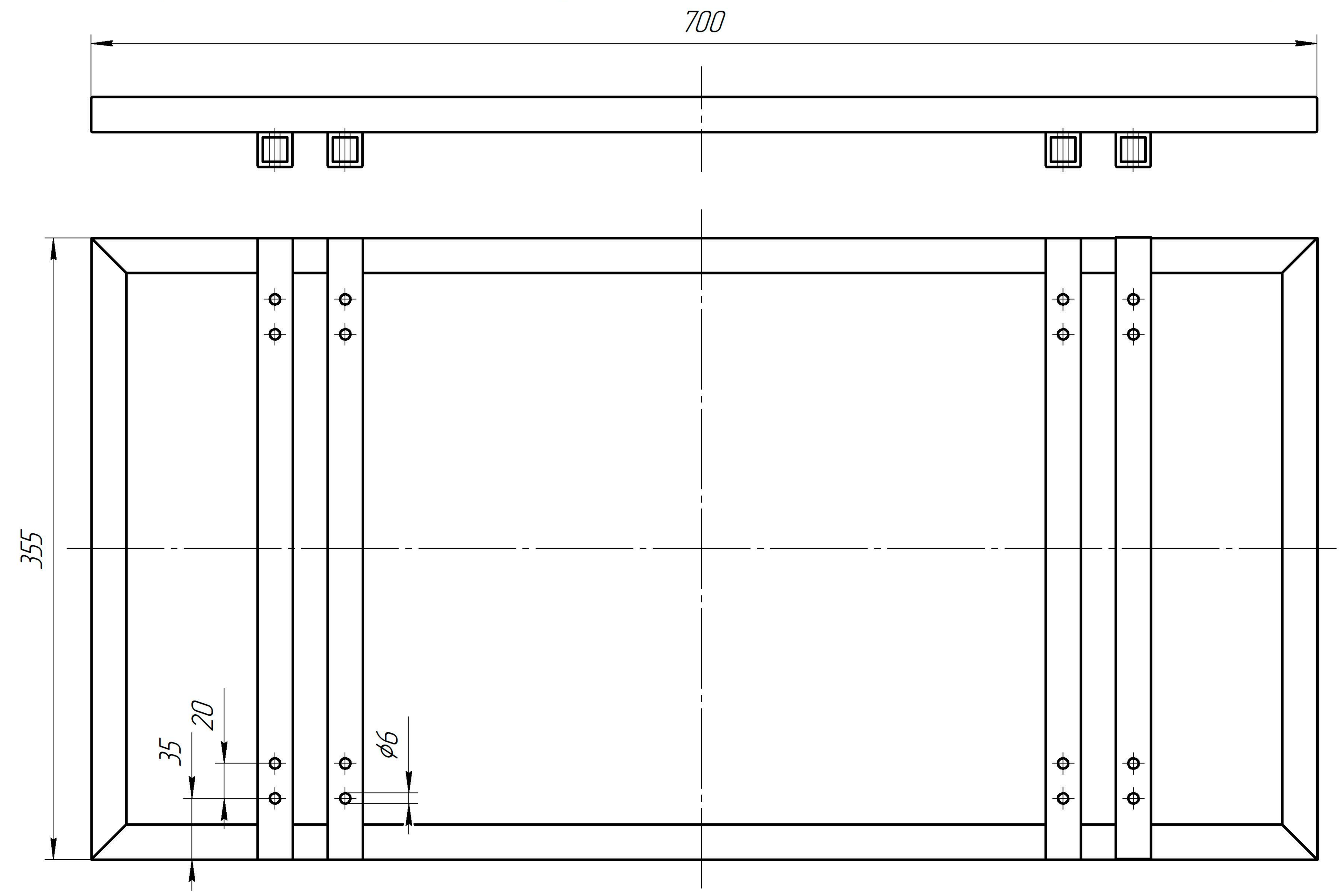


Мотор-колесо

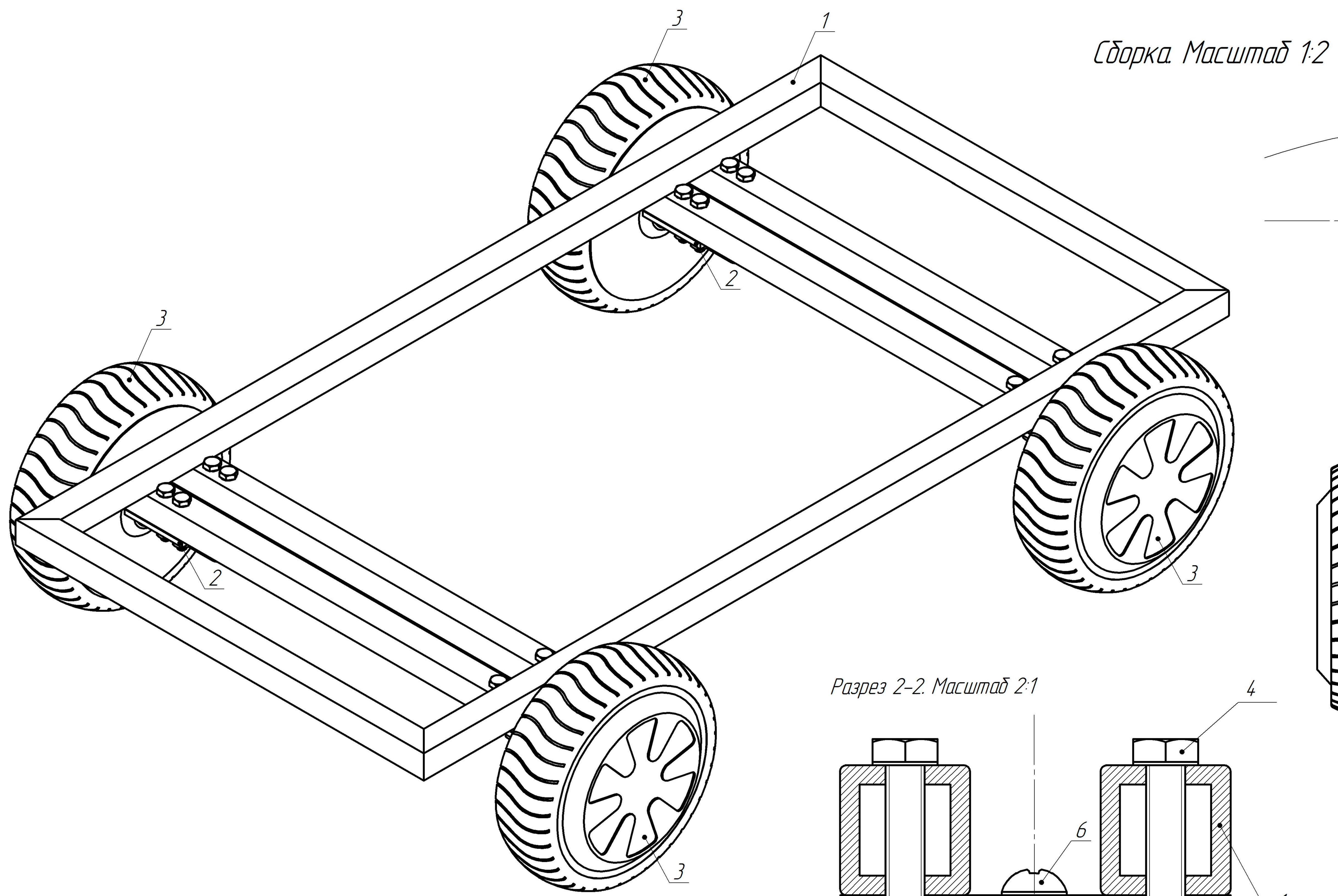


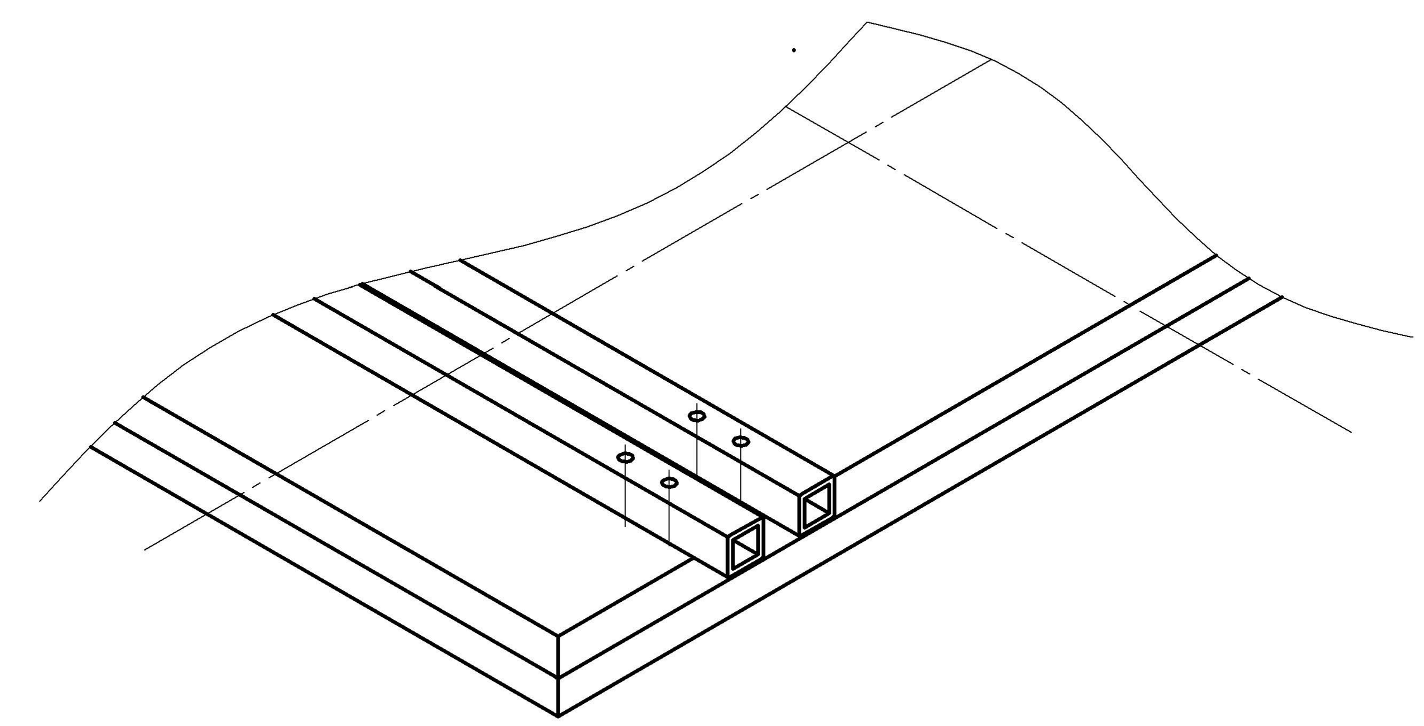
Рама. Масштаб 1:2.5

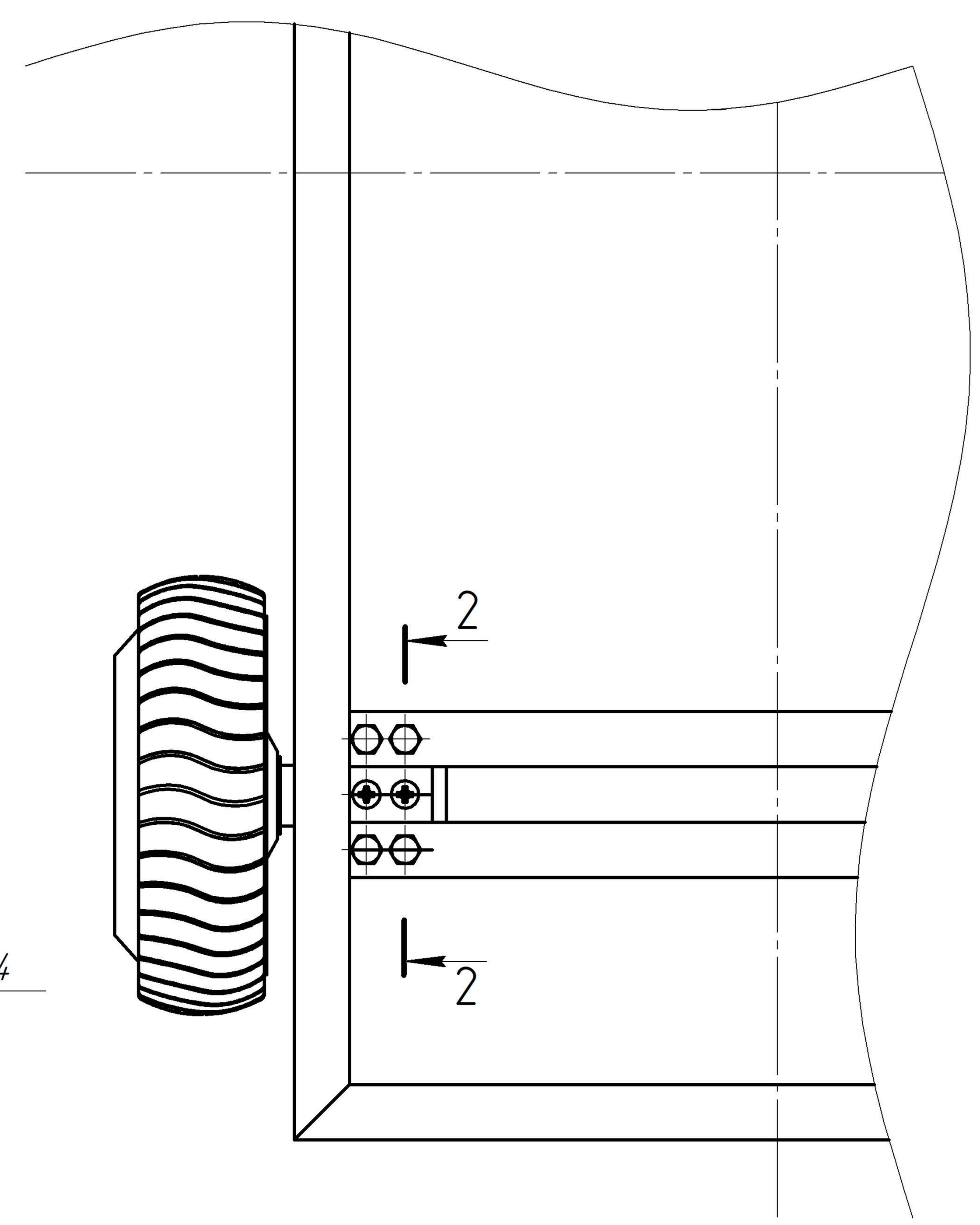
Вся рама выполняется из профилей 20мм. Все соединения выполняются сваркой.



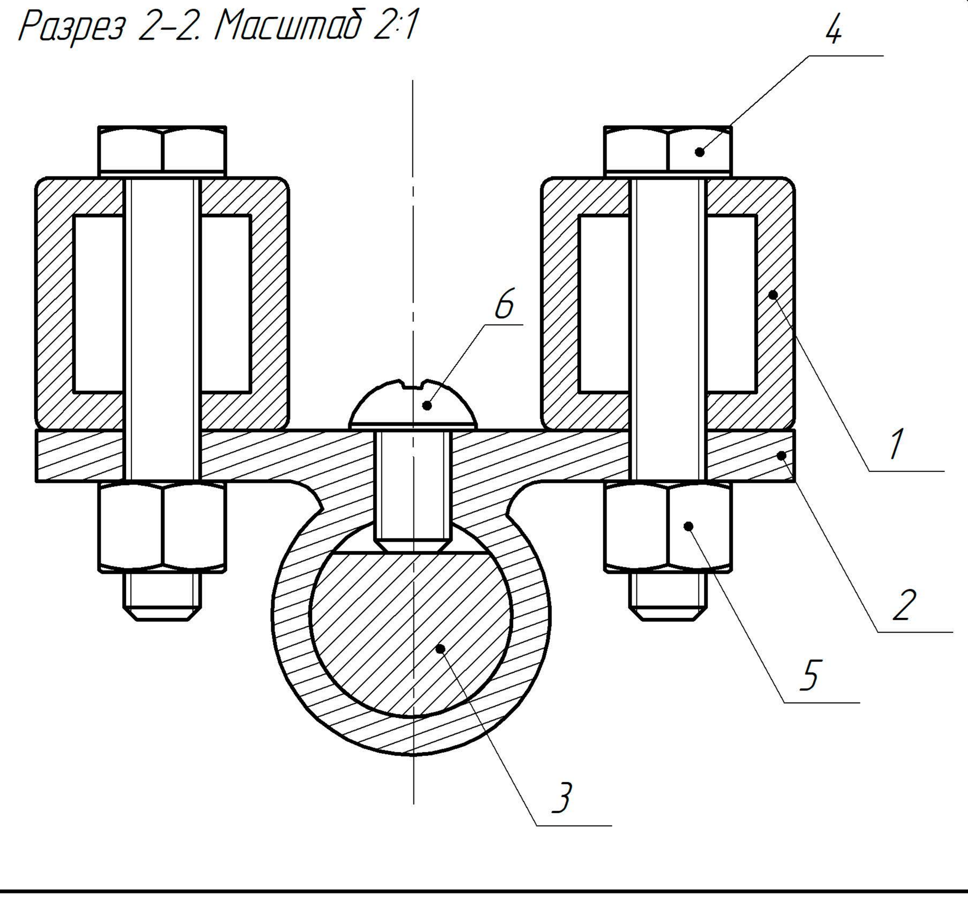
Сборка. Масштаб 1:2







Разрез 2-2. Масштаб 2:1



Формат	Зона	Поз.	Обозначение	Наименование	Кол.	Примечание
				<u>Детали</u>		
		1		Рама	1	
		2		Крепление колеса к раме	4	
		3		Мотор-колесо	4	
				<u>Стандартные изделия</u>		
		4		Болт М6х35 ГОСТ 174 73	16	
		5		Гайка М6х7.2 ГОСТ 5915	16	
		6		Винт М6х10 ГОСТ 1491-80	8	

Изм. Лист	№ докум.	Подп.	Дата	ВКР.204003.15.03.04.ЧС		
Разраб.	Бендик В. И.		21.06	Чертеж сборочный	Лит. У	Масса	Масштаб 1:1
Пров.	Теличенко Д.А.					
Т.контр.	Теличенко Д.А.				Лист 1	Листов 6
Н.контр.	Скрипко О. В.			Автоматизированная система управления электрошасси на базе мотор-колес	АмГУ гр. 041-об	
Утв.	Скрипко О. В.					

Копировал   Формат А1

					ВКР.204003.15.03.04.ОВ			
Изм	Лист	№ докум.	Подп.	Дата				
Разраб.		Бендик В. И.			Схема драйвера электрическая	Лит.	Масса	Масштаб
Пров.		Теличенко Д.А.				У		
Т.контр.		Теличенко Д.А.				Лист 2	Листов 6	
Н.контр.		Скрипко О.В.			Автоматизированная система управления электрошасси на базе мотор-колёс	АмГУ гр. 041-об		
Утв.		Скрипко О.В.						

Копировал Формат А1

Формат	Зона	Поз.	Обозначение	Наименование	Кол.	Примечание
				Двигатели		
			M1–M4	BLDC мотор ob20h36v	4	
				Элементы коммутации		
			Sb1, Sb2	PSW-6FA-B	2	
			XP1, X1, X2	Разъем XT-60	3	
				Устройства защиты		
			QF1	Автоматический выключатель C10	1	
				Схемы		
			DD1	ESP-32	1	
			DD2, DD3	Драйвер на STM-32	2	
			DD4	Bluetooth модуль HC-06	1	
			DD5	RC-приёмник FS-IA 10B	1	

ВКР.204003.15.03.04.СХ

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Бендик В. И.		21.06
Пров.		Теличенко Д.А.		
Т.контр.		Теличенко Д.А.		
Н.контр.		Скрипко О. В		
Утв.		Скрипко О. В		

Схема электрическая электрошасси

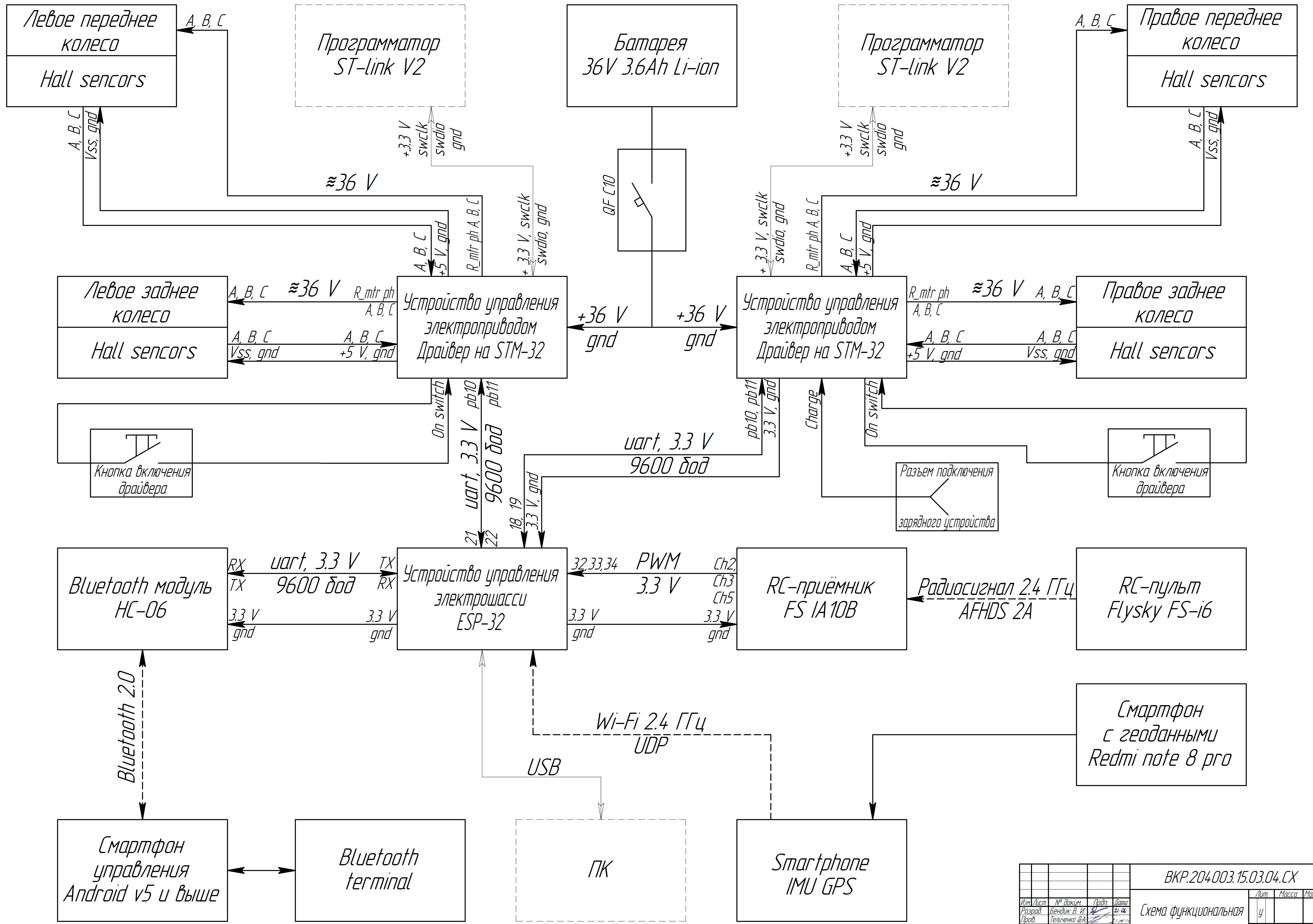
Лит.	Масса	Масштаб
У		

Лист 3 Листов 6

Автоматизированная система управления электрошасси на базе мотор-колёс

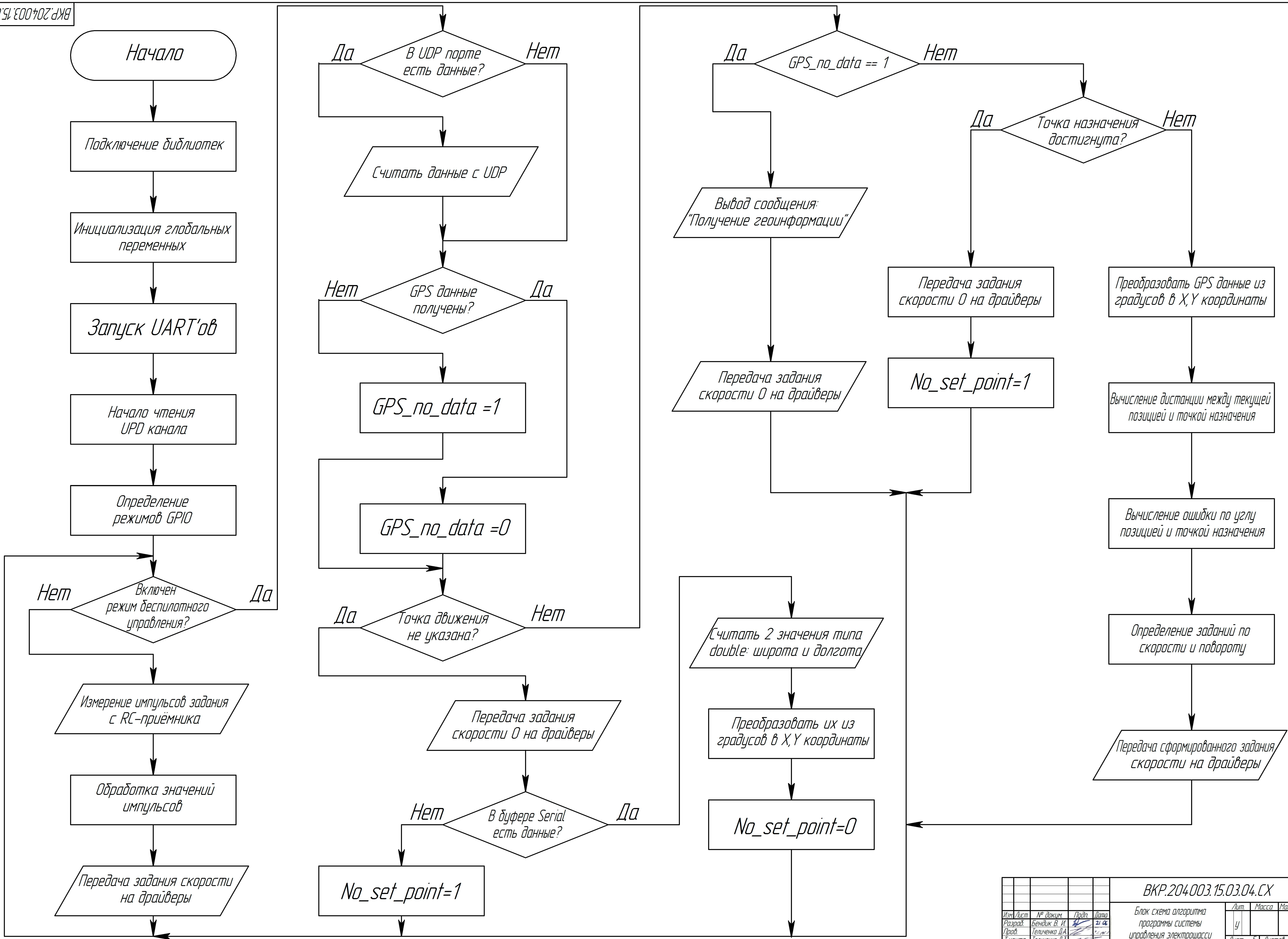
АмГУ гр. 041-об

Копировал Формат А1



					ВКР.204003.15.03.04.СХ			
Изм.	Лист	№ докум.	Подп.	Дата		Лит.	Масса	Масштаб
Разраб.		Бендик В. И.		21.06	Схема функциональная	у		
Пров.		Теличенко Д.А.						
Т.контр.		Теличенко Д.А.				Лист 4	Листов 6	
Н.контр.		Скрипко О. В.			Автоматизированная система управления электрошасси на базе мотор-колёс	АмГУ гр. 041-об		
Утв.		Скрипко О. В.						

Копировал Формат А1



ВКР.204003.15.03.04.СХ

Изм.	Лист	№ докум.	Подп.	Дата	ВКР.204003.15.03.04.СХ			
Разраб.		Бендик В. И.		21.06	Блок схема алгоритма программы системы управления электрошасси	Лит.	Масса	Масштаб
Пров.		Теличенко Д.А.				у		
Т.контр.		Теличенко Д.А.				Лист 5	Листов 6	
Н.контр.		Скрипко О. В.			Автоматизированная система управления электрошасси на базе мотор-колёс	АмГУ гр. 041-об		
Утв.		Скрипко О. В.						

Копировал Формат А1

Точка назначения 1

```
GPS position : X=50.295197, Y=127.566687, Z=107.400000
GPS position in cartesian coordinates: X = 9128040.33877490, Y=5598855.90917345
Coordinates set point X=9127646.49979385, Y=5598894.64849594
X_error: -393.84
Y_error: 38.74
Distance to point: 395.74
Angle set point: 275.62
```

	Значение из Яндекс карт	Расчетное в ручную	Расчетное программой	Погрешность между картами и программой
Дистанция до цели	329	391	395	-66
Угол поворота	280	277	275	5

Точка назначения 2

```
GPS position : X=50.295197, Y=127.566687, Z=107.400000
GPS position in cartesian coordinates: X = 9128040.33877490, Y=5598855.90917345
Coordinates set point X=9128226.09567803, Y=5599024.00221070
X_error: 185.76
Y_error: 168.09
Distance to point: 250.52
Angle set point: 47.86
```

	Значение из Яндекс карт	Расчетное в ручную	Расчетное программой	Погрешность между картами и программой
Дистанция до цели	308	261	250	58
Угол поворота	52	47	48	4

Точка назначения 3

```
GPS position : X=50.295197, Y=127.566687, Z=107.400000
GPS position in cartesian coordinates: X = 9128040.33877490, Y=5598855.90917345
Coordinates set point X=9127916.69165294, Y=5598603.21301811
X_error: -123.65
Y_error: -252.70
Distance to point: 281.33
Angle set point: 206.07
```

	Значение из Яндекс карт	Расчетное в ручную	Расчетное программой	Погрешность между картами и программой
Дистанция до цели	245	276	281	-36
Угол поворота	194	206	206	-12

Точка назначения 4

```
GPS position : X=50.295197, Y=127.565975, Z=99.000000
GPS position in cartesian coordinates: X = 9127989.39158113, Y=5598855.90917345
Coordinates set point X=9127987.03026457, Y=5599117.06564059
X_error: 6.08
Y_error: 261.05
Distance to point: 261.12
Angle set point: 1.33
```

	Значение из Яндекс карт	Расчетное в ручную	Расчетное программой	Погрешность между картами и программой
Дистанция до цели	245	272	261	-16
Угол поворота	2	11	1	1

Ручные расчеты для точки 1

$$Y_{\text{назн1}} = 6378160 * \frac{50.295540 * \pi}{180} = 5598890 \text{ м.}$$

$$X_{\text{назн1}} = 6378160 * \frac{127.561185 * \pi}{180} cos\left(\frac{50 * \pi}{180}\right) = 9127650 \text{ м}$$

$$\Delta_{Y1} = Y_{\text{назн}} - Y_{\text{нач}} = 5598890 - 5598850 = 40 \text{ м.}$$

$$\Delta_{X1} = X_{\text{назн}} - X_{\text{нач}} = 9127650 - 9128040 = -390 \text{ м}$$

$$\Delta_1 = \sqrt{{\Delta_Y}^2 * {\Delta_X}^2} = \sqrt{40^2 + (-390)^2} = 391.15 \text{ м.}$$

$$cos(\alpha)_1 = \frac{\overrightarrow{(\Delta_X,\Delta_Y)} * \overrightarrow{(0,100)}}{(\Delta_X, \Delta_Y) * (0,100)} = \frac{\overrightarrow{(-390,40)} * \overrightarrow{(0,100)}}{(-390,40) * (0,100)} = 0.11$$

$$cos(0.11) = 83°$$

$$\Delta_{X1} < 0 \rightarrow \varphi = 360 - 83 = 277°$$

Ручные расчеты для точки 2

$$Y_{\text{назн2}} = 6378160 * \frac{50.296707 * \pi}{180} = 5599030 \text{ м.}$$

$$X_{\text{назн2}} = 6378160 * \frac{127.569283 * \pi}{180} cos\left(\frac{50 * \pi}{180}\right) = 9128230 \text{ м}$$

$$\Delta_{Y2} = Y_{\text{назн}} - Y_{\text{нач}} = 5599030 - 5598850 = 180 \text{ м.}$$

$$\Delta_{X2} = X_{\text{назн}} - X_{\text{нач}} = 9128230 - 9128040 = 190 \text{ м}$$

$$\Delta_2 = \sqrt{{\Delta_Y}^2 * {\Delta_X}^2} = \sqrt{180^2 + 190^2} = 261.72 \text{ м.}$$

$$cos(\alpha)_2 = \frac{\overrightarrow{(\Delta_X,\Delta_Y)} * \overrightarrow{(0,100)}}{(\Delta_X, \Delta_Y) * (0,100)} = \frac{\overrightarrow{(190,180)} * \overrightarrow{(0,100)}}{(190,180) * (0,100)} = 0.687$$

$$cos(0.687) = 46{,}5°$$

Ручные расчеты для точки 3

$$Y_{\text{назн3}} = 6378160 * \frac{50.292927 * \pi}{180} = 5598603 \text{ м.}$$

$$X_{\text{назн3}} = 6378160 * \frac{127.564959 * \pi}{180} cos\left(\frac{50 * \pi}{180}\right) = 9127917 \text{ м}$$

$$\Delta_{Y3} = Y_{\text{назн}} - Y_{\text{нач}} = 5598603 - 5598850 = -247 \text{ м.}$$

$$\Delta_{X3} = X_{\text{назн}} - X_{\text{нач}} = 9127917 - 9128040 = -123 \text{ м}$$

$$\Delta_3 = \sqrt{{\Delta_Y}^2 * {\Delta_X}^2} = \sqrt{(-247)^2 + (-123)^2} = 275.93 \text{ м.}$$

$$cos(\alpha)_3 = \frac{\overrightarrow{(\Delta_X,\Delta_Y)} * \overrightarrow{(0,100)}}{(\Delta_X, \Delta_Y) * (0,100)} = \frac{\overrightarrow{(-252,-120)} * \overrightarrow{(0,100)}}{(-252,-120) * (0,100)} = -0.89$$

$$cos(-0.89) = 154°$$

$$\Delta_{X3} < 0 \rightarrow \varphi = 360 - 154 = 206°$$

Ручные расчеты для точки 4

$$Y_{\text{назн4}} = 6378160 * \frac{50.297543 * \pi}{180} = 5599117 \text{ м.}$$

$$X_{\text{назн4}} = 6378160 * \frac{127.565942 * \pi}{180} cos\left(\frac{50 * \pi}{180}\right) = 9127987 \text{ м}$$

$$\Delta_{Y4} = Y_{\text{назн}} - Y_{\text{нач}} = 5599117 - 5598850 = 267 \text{ м.}$$

$$\Delta_{X4} = X_{\text{назн}} - X_{\text{нач}} = 9127987 - 9128040 = -53 \text{ м}$$

$$\Delta_4 = \sqrt{{\Delta_Y}^2 * {\Delta_X}^2} = \sqrt{267^2 + (-53)^2} = 272.20 \text{ м.}$$

$$cos(\alpha)_4 = \frac{\overrightarrow{(\Delta_X,\Delta_Y)} * \overrightarrow{(0,100)}}{(\Delta_X, \Delta_Y) * (0,100)} = \frac{\overrightarrow{(-53,267)} * \overrightarrow{(0,100)}}{(-53,267) * (0,100)} = 0.98$$

$$cos(0.98) = 11°$$

	Широта	Долгота	X	Y
Точка старта	50.295210	127.566252	9128009.212	5598857.356
Точка назначения 1	50.295540	127.561185	9127646.499	5598894.648
Точка назначения 2	50.296707	127.569283	9128226.095	5599024.002
Точка назначения 3	50.292927	127.564959	9127987.030	5599117.065
Точка назначения 4	50.297543	127.565942	9127916.691	5598603.018

$$Y = R_{\text{земли}} * \text{ш} = R_{\text{земли}} * \frac{\text{ш}° * \pi}{180}$$

$$X = R_{\text{земли}} * \text{д} * cos(\text{цш}) = R_{\text{земли}} * \frac{\text{д}° * \pi}{180} * cos\left(\frac{\text{цш}° * \pi}{180}\right)$$

Где: $R_{\text{земли}}$ – радиус Земли, 6378160 м.;

ш – широта в радианах;

д – долгота в радианах;

цш – центральная широта в радианах.

$$Y_{\text{нач}} = 6378160 * \frac{50.295197 * \pi}{180} = 5598850 \text{ м.}$$

$$X_{\text{нач}} = 6378160 * \frac{127.566687 * \pi}{180} cos\left(\frac{50 * \pi}{180}\right) = 9128040 \text{ м.}$$

Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Бендик В. И.		21.06	ВКР.204003.15.03.04.ВО	
Пров.		Теличенко Д.А.			Тестирование системы проектирования маршрута	Лит. У; Масса; Масштаб
Т.контр.		Теличенко Д.А.				Лист 6; Листов 6
Н.контр.		Скрипко О. В.			Автоматизированная система управления электрошасси на базе мотор-колёс	АмГУ гр. 041-об
Утв.		Скрипко О. В.				

Копировал

Формат А1