**Министерство образования и науки Российской Федерации**
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
**АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСТЕТ**
**(ФГБОУ ВПО «АмГУ»)**

Факультет энергетический

Кафедра автоматизации производственных процессов и электротехники

Направление подготовки 15.03.04 - Автоматизация технологических процессов и производств

Профиль Автоматизация технологических процессов и производств в энергетике

# КУРСОВОЙ ПРОЕКТ

на тему: Автоматизированная система управления электрошасси

По дисциплине: «Автоматизация технологических процессов и производств»

Работу выполнил _____ В. И. Бендик
(подпись, дата)

Руководитель
к.т.н. доцент _____Д. А. Теличенко
(подпись, дата)

Нормоконтроль
к.т.н. доцент _____Д. А. Теличенко
(подпись, дата)

Благовещенск 2024

**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСТЕТ**

**(ФГБОУ ВПО «АмГУ»)**

Факультет: Энергетический

Кафедра: автоматизации производственных процессов и электротехники

УКЦ «Автоматика и управление в технических системах»

**З А Д А Н И Е**

К курсовому проекту Бендик Владимира Игоревича

1. Тема курсового проекта: Автоматизированная система управления электрошасси.

2. Срок сдачи студентом законченного проекта: _____

3. Исходные данные к курсовому проекту:

Разработать и изготовить электрошасси пригодного для перевозки роботов или малогабаритных грузов. Управляемого дистанционно по беспроводной связи, а также способно функционировать автономно от источников питания не менее 2 часов.

4. Содержание курсового проекта (перечень подлежащих разработке вопросов):

1) Обзор объекта управления

2) Выбор устройств системы управления

3) Разработка программного обеспечения контроллера электропривода

4) Разработка программного обеспечения для ESP-32

5) Тестирование

6) Заключение

5. Перечень материалов приложения:

Лист 1: Электрическая схема драйвера;

Лист 2: Электрическая схема электрошасси;

Лист 3: Сборочный чертеж;

6. Дата выдачи задания: _____

Руководитель курсового проекта к.т.н. доцент Теличенко Денис Алексеевич.

Задание принял к исполнению (дата): _____.

# РЕФЕРАТ

Курсовой проект 36 с., 16 рисунков, 4 таблицы, 6 частей, 8 источников.

ЭЛЕКТРОШАССИ, ЭЛЕКТРОПРИВОД, БДПТ, BLDC, ДРАЙВЕР УПРАВЛЕНИЯ, STM-32, ST-LINK, VISUAL STUDIO CODE, ESP-32, BLUETOOTH, WI-FI, RC-PWM, ВЕКТОРНОЕ УПРАВЛЕНИЕ, МИКРОКОНТРОЛЛЕР, ARDUINO IDE, ПРОГРАММА, АППАРАТНО-СТРУКТУРНАЯ СХЕМА, UART ИНТЕРФЕЙС.

В работе описана разработка электрошасси с различными вариантами управления.

Целью курсового проекта является разработка и изготовления электрошасси способного перевозить роботов и малогабаритные грузы, отвечающее заданным техническим требованиям, обеспечивающее высокую маневренность, проходимость и надежность в различных условиях эксплуатации.

СОДЕРЖАНИЕ

Введение 6
1 Обзор объекта управления 7
1.1 Назначение электрошасси 7
1.2 Конструкция электрошасси 8
1.3 Электропривод 9
1.4 Принцип управления колесами BLDC 11
1.5 Векторное управление 13
2 Выбор устройств системы управления 15
2.1 Выбор драйвера управления колесами 15
2.2 Выбор устройства ввода-вывода 17
3 Разработка программного обеспечения контроллера электропривода 19
3.1 Требования к функционалу реализуемому в ПО 19
3.2 Описание файлов в hoverboard firmware hack FOC 22
3.3 Настройка файла config и компиляция 23
3.4 Загрузка прошивки в плату 25
4 Разработка программного обеспечения для ESP-32 27
4.1 Варианты управления 27
4.2 Bluetooth 28
4.3 RC пульт 31
4.4 Wi-Fi 33
Заключение 35
Библиографический список 36
Приложение А 37

# ВВЕДЕНИЕ

Целью данного курсового проекта является разработка электрошасси для мобильного робота. В рамках работы будет проведен выбор компонентов шасси и программного обеспечения, редактирование готового ПО в среде Visual studio code, а также самостоятельное написание программы для микроконтроллера ESP-32 на языке C++ в среде разработки Arduino IDE.

Мобильные роботы становятся все более распространенными в различных областях, включая промышленность, логистику и обслуживание клиентов. Одним из ключевых компонентов мобильного робота является его электрошасси, которое обеспечивает движение и маневренность. Разработка эффективного и надежного электрошасси имеет решающее значение для общей производительности робота.

Исследование и разработка транспорта на основе электроприводов является актуальной темой. Эта работа позволит более подробно изучить возможности разработки транспортных средств и управления электроприводом, а также научиться применять их на практике.

# 1 ОБЗОР ОБЪЕКТА УПРАВЛЕНИЯ

## 1.1 Назначение электрошасси

Электрошасси – это электрическая платформа, на которую устанавливаются различные компоненты и модули для создания различных видов транспортных средств или устройств робототехники. Она является основой для построения электрических автомобилей, автобусов, грузовиков, некоторых малогабаритных видов транспорта и мобильных роботов.

Электрошасси применяются в различных отраслях включая робототехнику. Они используются для создания экологически чистых транспортных средств с нулевыми выбросами вредных веществ и шума, что особенно важно при работе в закрытых пространствах, таких как заводы и склады. Электрошасси обладают рядом преимуществ: они обеспечивают высокую эффективность и экономию энергии, электрошасси обладают плавным и тихим ходом, что делает их особенно удобными для городской среды. Кроме того, электрический транспорт требует меньше затрат на обслуживание и имеет меньше деталей, что упрощает его конструкцию и улучшает надежность.

В зависимости от конкретной модели и назначения электрошасси могут быть оснащены различными функциями и возможностями. Они могут включать в себя системы управления аккумуляторами, системы зарядки, электронные системы безопасности, системы управления движением и другие инновационные технологии. Электрошасси также могут быть адаптированы для работы с различными видами кузовов и каркасов, что позволяет создавать разнообразные типы транспортных средств.

В целом, 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. Они способствуют повышению энергоэффективности, уменьшению загрязнения окружающей среды и созданию более комфортной городской среды.

### 1.2 Конструкция электрошасси

Первым этапом изготовления электрошасси являлось создание 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 355x700 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

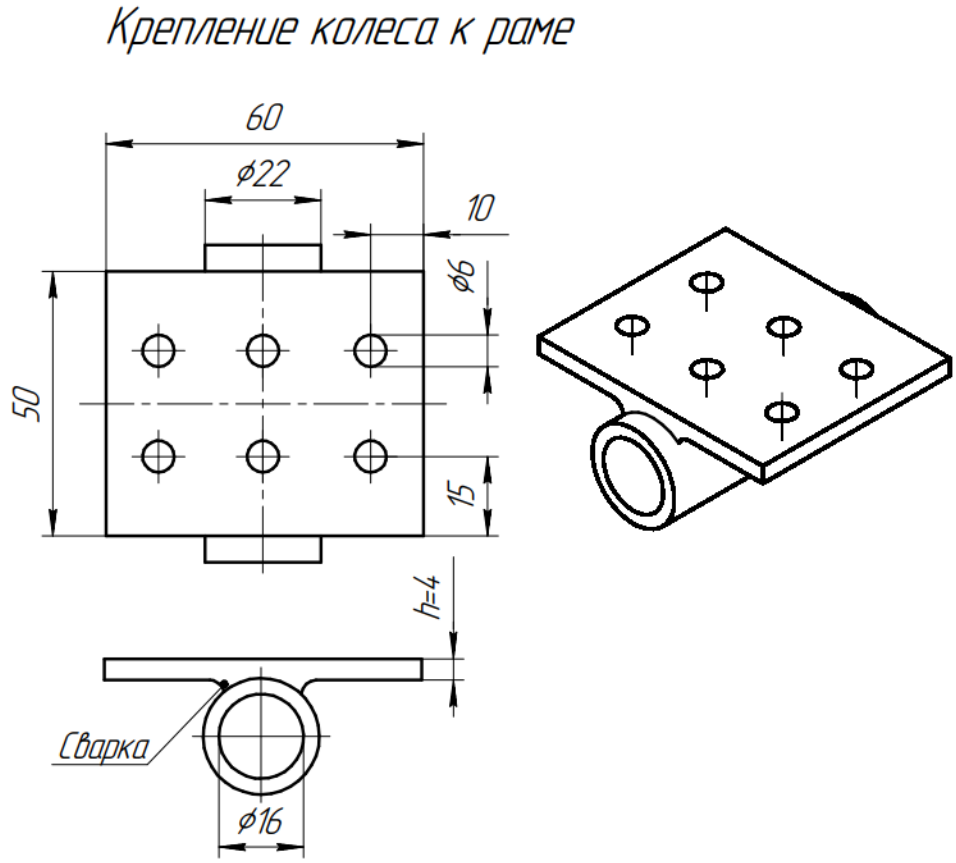



Рисунок 1.1 – Чертеж крепления колеса к раме

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.

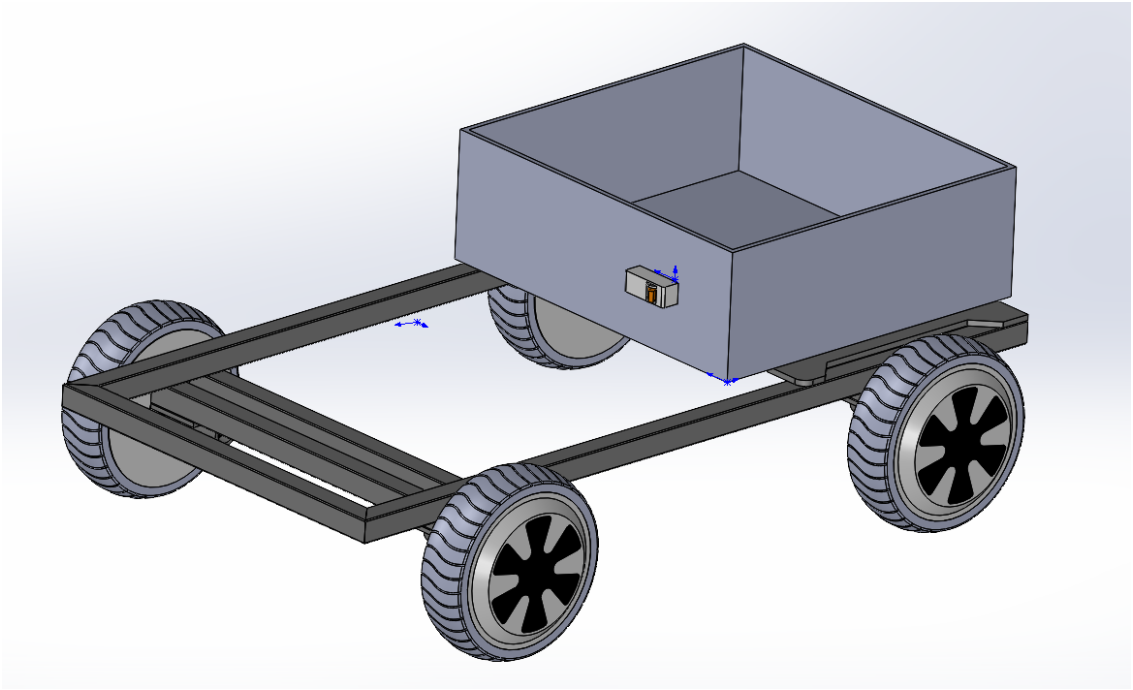


Рисунок 1.2 – 3D модель рамы

Полная схема всех деталей электрошасси и их сборочный чертеж представлены на листе 3.

### 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 интегра-

цию 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 |

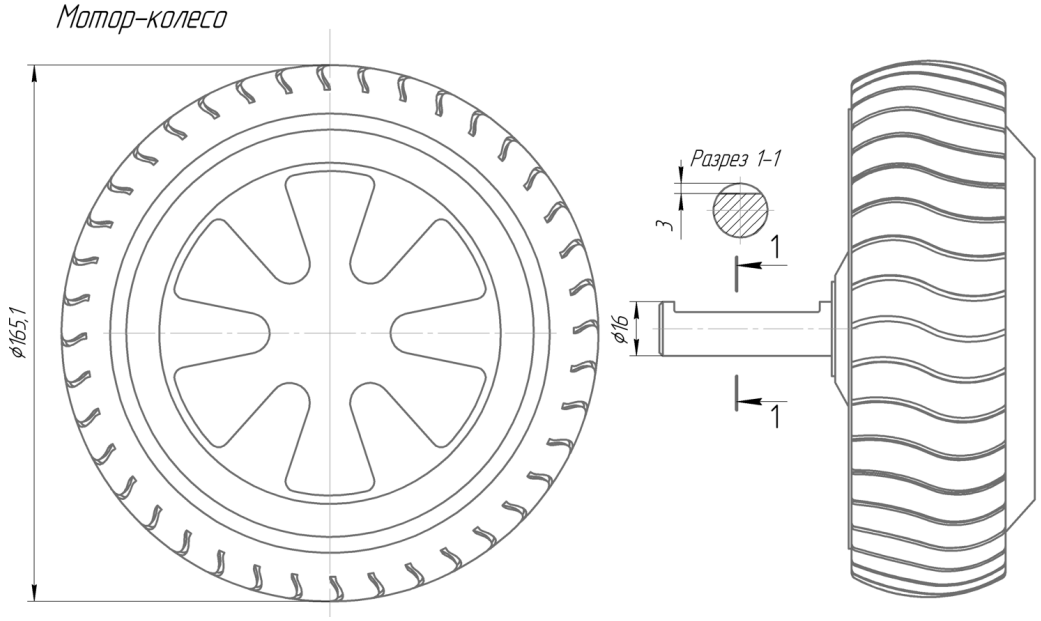



Рисунок 1.3 – Чертеж выбранного мотор-колеса

### 1.4 Принцип управления колесами BLDC

Как и любая электрическая машина мотор-колесо состоит из ротора и статора. Отличающей конструктивной чертой является то, что индуктором машины является ее ротор, выполненный из постоянных магнитов, а якорем машины является статор, состоящий из 3 фаз. Кроме того, работа BLDC мотора невозможна без драйвера, установленного вне двигателя и выполняющего функцию коммутации обмоток.

В статоре в виде многолучевой звезды из электротехнической стали появляется магнитное поле. При взаимодействии с постоянными магнитами оно инициирует вращение ротора. На лучах статора есть обмотки, и когда по ним идет ток, лучи превращаются в электромагниты. Они притягивают постоянные магниты на роторе и инициируют вращение ротора.

Для получения нужной мощности и равномерного вращения колеса статор имеет несколько десятков обмоток. Но в результате они соединяются в 3 и чередуются по окружности: 1-2-3-1-2-3. На противоположной стороне на роторе есть магниты из редкоземельных материалов. Когда на обмотки поступает напряжение, происходит активизация их магнитных качеств, взаимодействие с магнитами и вращение ротора [2].

Напряжение подается на обмотки поочередно и четко в нужные моменты

времени. Определяют эти моменты находящиеся на статоре датчики Холла. Они отслеживают взаимное расположение ротора и статора, откликаются на магнитное поле и отправляют сигналы на контроллер. На основании полученных сведений контроллер, он же драйвер, своевременно подает на обмотки статора импульсы напряжения. Обмотки превращаются в электромагниты, вступают во взаимодействие с постоянными магнитами ротора и заставляют его вращаться. Графически устройство BLDC мотора показано на рисунке 1.4.

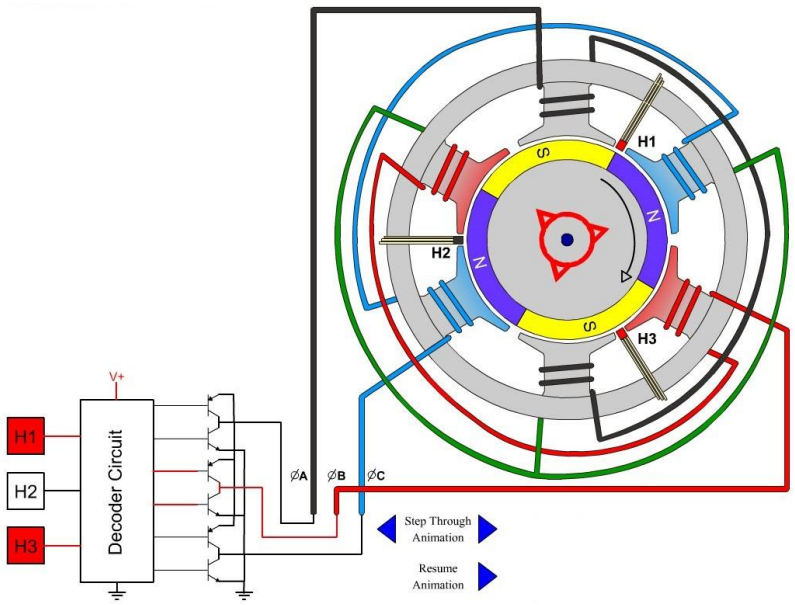



Рисунок 1.4 – Устройство BLDC мотора

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: Векторное управление позволяет точно управлять скоростью и крутящим моментом двигателя, что приводит к повышению эффективности и снижению энергопотребления.
3. Низкий уровень шума и вибрации: Векторное управление помогает снизить уровень шума и вибрации двигателя за счет оптимизации коммутации.
4. Улучшенная надежность: Векторное управление может обнаруживать и компенсировать неисправности двигателя, что повышает его надежность.

Реализация векторного управления требует специализированного контроллера, который может выполнять сложные вычисления в реальном времени.

Современные цифровые сигнальные процессоры (DSP) и микроконтроллеры (MCU) обычно используются для реализации векторного управления. Векторное управление является мощным методом управления электродвигателями, который обеспечивает высокую производительность, точность и эффективность. Он широко используется в различных областях и играет важную роль в повышении производительности и надежности систем с электроприводом.

# 2 ВЫБОР УСТРОЙСТВ СИСТЕМЫ УПРАВЛЕНИЯ

## 2.1 Выбор драйвера управления колесами

Драйверы для бесщеточных моторов — это электронные устройства, которые управляют работой моторов, обеспечивая им питание и контролируя их скорость и направление вращения. Они играют ключевую роль в эффективной работе двигателей и обеспечивают точное и стабильное управление движением.

Существует несколько типов драйверов для BLDC моторов, которые отличаются по своим характеристикам и функциональности:

а) Простые драйверы: это базовые драйверы, которые обеспечивают основные функции управления мотором колеса, такие как управление скоростью и направлением вращения. Они обычно имеют ограниченный набор функций и могут быть более доступными по стоимости.

б) Драйверы с обратной связью: Эти драйверы имеют возможность получать обратную связь от датчиков положения, установленных на роторе. Это позволяет более точно контролировать скорость и положение колеса, что особенно полезно для устройств, требующих высокого вращательного.

в) Драйверы с расширенными функциями: Некоторые драйверы могут предлагать дополнительные функции, такие как защита от перегрузки, торможение, рекуперативное торможение (возможность использования энергии при торможении для зарядки аккумуляторов) и другие продвинутые возможности. Они обычно более функциональны, но и более дороги.

При выборе драйвера для BLDC колес следует обратить внимание на следующие аспекты:

а) Ток и напряжение. Необходимо убедитесь, что драйвер поддерживает требуемый диапазон тока и напряжения, соответствующий моторам колес.

б) Контроль скорости и направления по обратной связи. Драйвер должен предоставлять необходимые возможности для точного и стабильного управления скоростью и направлением вращения моторов колес.

в) Защитные функции. Современный функциональный драйвер должен об-

ладать встроенными механизмами защиты, такими как защита от перегрузки, короткого замыкания и температурная защита, чтобы обеспечить безопасную и надежную работу системы.

е) Надежность и качество. На современном рынке существует уже множество драйверов и устройств, реализующих их функции. Многие из них зарекомендовали себя как крайне надежные устройства. Подобные драйвера применяются в брендированных гироскутерах и электросамокатах.

Выбор подходящего драйвера для BLDC колес важен для обеспечения эффективной и надежной работы вашей системы. Уделите внимание требованиям вашего проекта и выберите драйвер, который соответствует вашим 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.

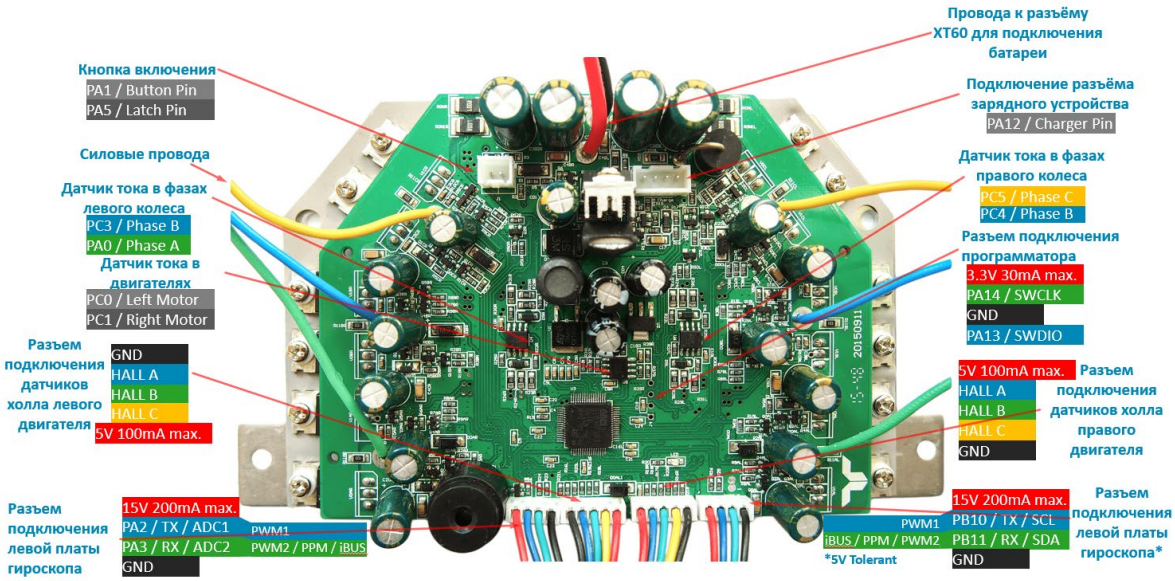



Рисунок 2.1 – Драйвер управления моторами

Полная электрическая принципиальная схема драйвера управления представлена на рисунке 1.

### 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.

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

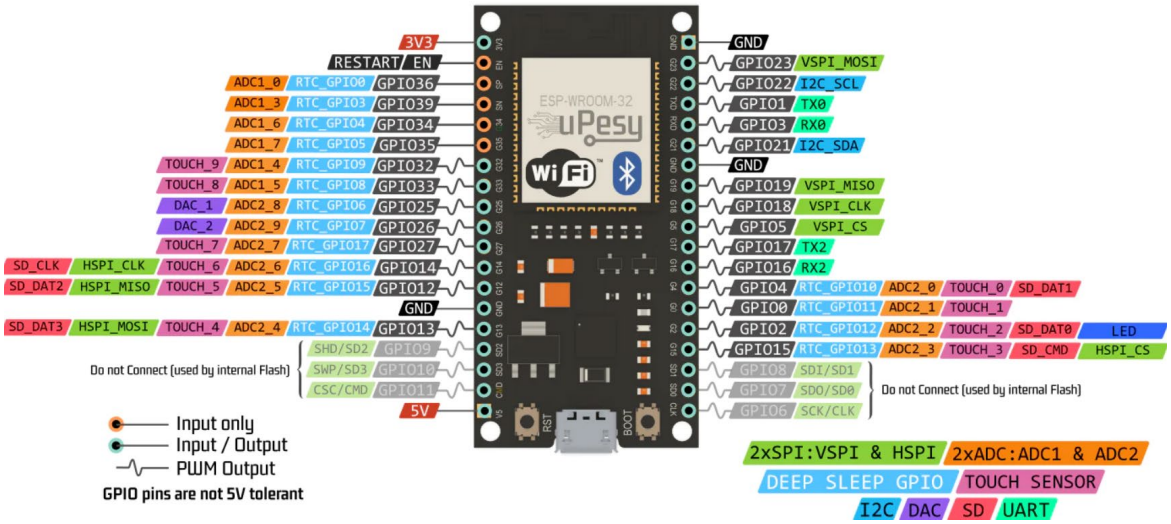



Рисунок 2.2 – Распиновка ESP-WROOM-32

Итого: система управления будет включать в себя АРМ, на котором оператор задает скорость колес и угол поворота при движении, а также наблюдает информацию о текущей скорости вращения колес и токах в них. АРМ получает и отправляет информацию на ESP32 через Wi-Fi соединение. ESP32, в свою очередь, соединен с драйвером по проводной связи через UART интерфейс. Структурная схема системы управления предоставлена на рисунке 2.3.

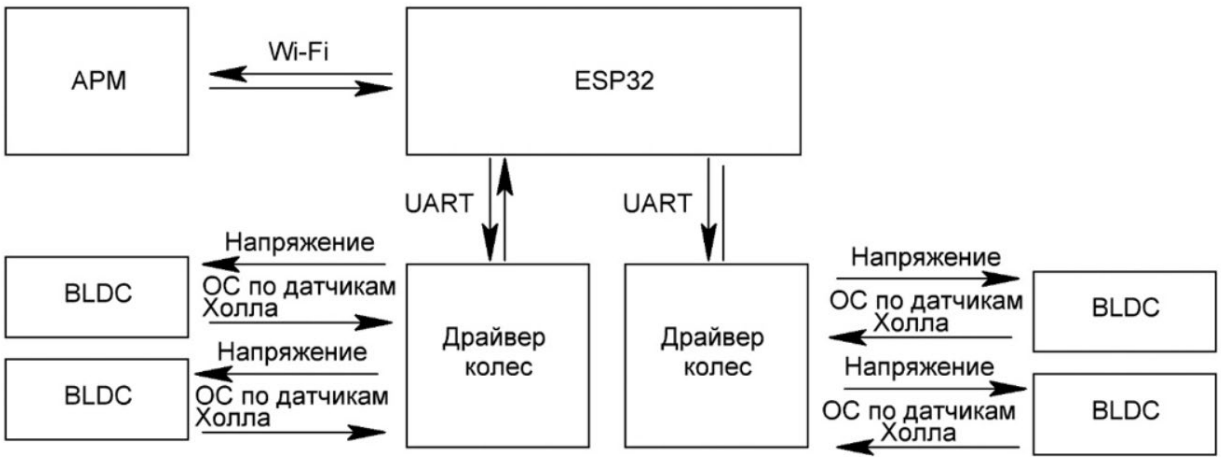



Рисунок 2.3 – Структурная схема управления электрошасси

Принципиальная электрическая схема соединений всех выбранных элементов представлена на листе 2.

# 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 |
|---|---|---|---|---|---|---|
| Коммутационный | - | - | + + | n.a. | n.a. | + |
| Синусоидальный | + | + + | + + | + + + | n.a. | n.a. |
| Режим напряжения | + + | + + + | + + | + + | n.a. | + (2) |

Продолжение таблицы 3.1

| | | | | | | |
|---|---|---|---|---|---|---|
| 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 |
|---|---|
| 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<br>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) |
| BLDC_controller.c<br>BLDC_controller.h | Файлы автоматически сгенерированного в MatLab кода. Используются в управлении. |

Продолжение таблицы 3.2

| | |
|---|---|
| BLDC_controller_data.c | |
| rtwtypes.h | |
| util.c<br>util.h | Общие служебные процедуры, совместно используемые несколькими модулями. |
| eeprom.c<br>eeprom.h | Драйвер виртуального EEPROM |
| hd44780.c<br>hd44780.h | Библиотека для работы в LCD дисплеем |
| pcf8574.c<br>hd44780.h | Библиотека для работы в LCD дисплеем |
| setup.c<br>setup.h | Инструкции инициализации платы |
| stm32f1xx_it.c<br>stm32f1xx_it.h | Библиотека прерываний для процессоров Cortex-M3 |
| system_stm32f1xx.c<br>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.

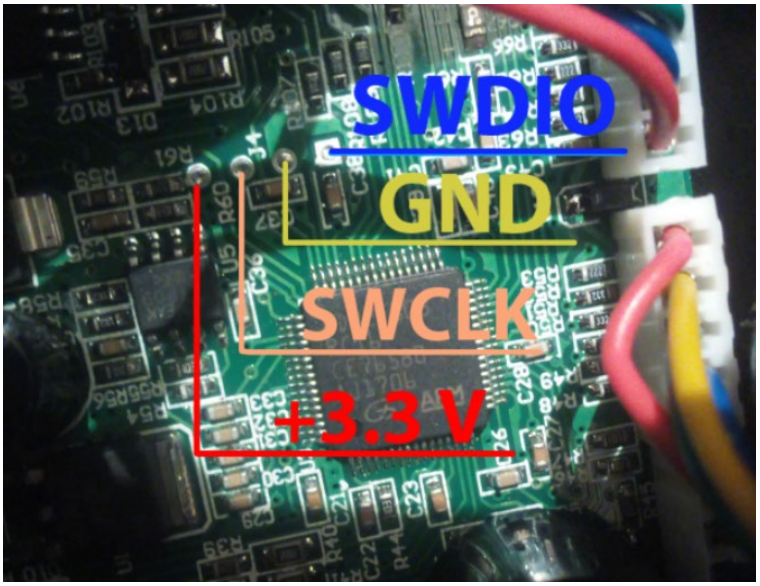



Рисунок 3.3 – Пины для подключения программатора на плате

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.

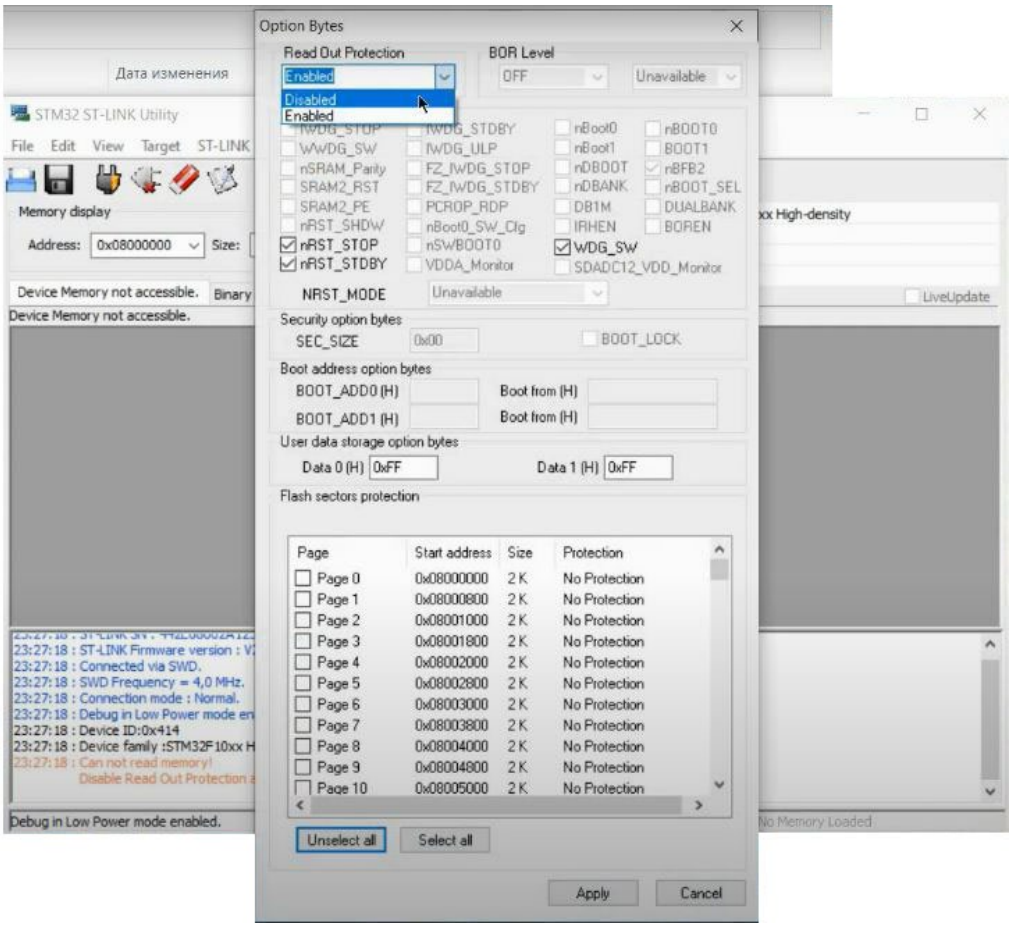



Рисунок 3.4 – параметры загрузки в STM32 ST-LINK Utility

# 4 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ESP-32

## 4.1 Варианты управления

Основная задача ESP-32 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 3 популярных беспроводных способа передачи данных: Bluetooth, Wi-Fi и RC PWM. Каждый из вариантов имеет своими плюсами и минусами. Сравнение протоколов 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 |
| Wi-Fi | Возможность общения со всеми устройствами подключенными к местной Wi-Fi сети, возможность создания полноценного web-интерфейса для оператора, в качестве пульта управления можно использовать обычный смартфон или ПК, возможность подключить внешний роутер с выходом в интернет и получить неограниченную дальность использования. | Высокая сложность реализации. Ограниченная дальность действия без внешнего роутера. |

Несмотря на то, что наибольшее количество плюсов получил Wi-Fi и он же представлен на схеме в пункте 2.2, в ходе разработки было решено реализовать все варианты 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 ESP-32 было решено в среде Arduino IDE с установленным ядром arduino-esp32 [7]. ESP-32 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 ESP-32 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.

Полный листинг кода представлен в приложении А.

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

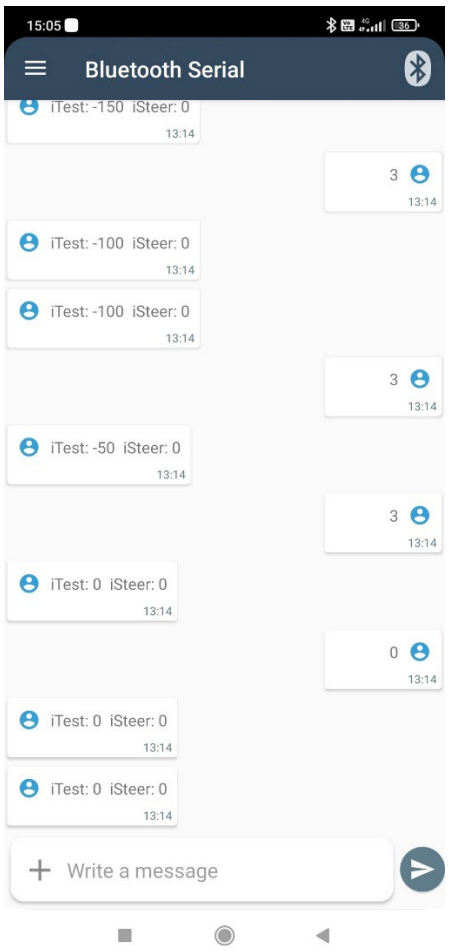



Рисунок 4.3 – Интерфейс приложения Bluetooth Terminal

### 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. 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

### 4.4 Wi-Fi

Wi-Fi (Wireless Fidelity) — это беспроводная технология, которая позволяет устройствам подключаться к сети и обмениваться данными без использования физических кабелей. Она основана на стандарте IEEE 802.11 и широко используется в домах, офисах, общественных местах и других средах.

Wi-Fi работает путем передачи данных по радиоволнам в определенном диапазоне частот. Устройства, такие как ноутбуки, смартфоны и планшеты, оснащены беспроводными сетевыми адаптерами, которые позволяют им подключаться к сети Wi-Fi.

Когда устройство подключается к сети Wi-Fi, оно устанавливает соединение с точкой доступа Wi-Fi (обычно маршрутизатором или модемом). Точка доступа передает данные на устройство по радиоволнам, а устройство отправляет данные обратно на точку доступа тем же способом.

Выбранный микроконтроллер ESP32 обладает встроенным модулем Wi-Fi, что делает его идеальным для проектов, требующих беспроводного подключения. Модуль Wi-Fi ESP32 поддерживает как режим станции (клиент), так и

режим точки доступа (хост).

В режиме станции ESP32 подключается к существующей сети Wi-Fi, как и любое другое устройство с поддержкой Wi-Fi. Для подключения к сети необходимо указать имя сети (SSID) и пароль. После подключения ESP32 может отправлять и получать данные через сеть Wi-Fi.

В режиме точки доступа ESP32 создает собственную сеть Wi-Fi, к которой могут подключаться другие устройства. Это позволяет ESP32 выступать в качестве центрального узла для других устройств, которым требуется подключение к Интернету или обмен данными между собой.

Использовать Wi-Fi на ESP32 относительно просто. Сначала необходимо инициализировать модуль Wi-Fi и настроить его для работы в режиме станции или точки доступа. Затем можно подключиться к сети Wi-Fi или создать собственную сеть. После подключения к сети можно отправлять и получать данные через сокеты или использовать протоколы более высокого уровня, такие как HTTP или MQTT.

В виду высокой сложности и малого знакомства с HTTP протоколом в рамках этой работы web-интерфейс останется не реализован и будет завершен в рамках дипломной работы по этой же теме.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсового проекта было разработано электрошасси с дистанционным управлением через Bluetooth и RC PWM. Разработанное устройство представляет собой мобильную платформу, управляемую с помощью различных беспроводных технологий.

В ходе работы были изучены принципы работы беспроводных технологий Bluetooth, Wi-Fi и RC PWM, а также особенности их применения для управления мобильными роботами. Были разработаны аппаратная и программная части электрошасси, обеспечивающие его движение и управление.

Разработанное электрошасси успешно прошло испытания и показало хорошие результаты. Устройство продемонстрировало стабильное и надежное управление, а также высокую маневренность и проходимость.

Результаты данной курсового проекта могут быть использованы для дальнейшего развития и совершенствования. Дальнейшее развитие электрошасси может быть направлено на повышение его автономности, интеллектуальности и функциональности. Возможно добавление датчиков, камер и других устройств для расширения возможностей шасси.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. What is a BLDC Motor in a Washing Machine? URL: https://www.dumblittleman.com/bldc-motor (последняя дата обращения 25.03.24).

2. 6-шаговая коммутация BLDC моторов URL: https://habr.com/ru/articles/745372 (последняя дата обращения 25.03.24).

3. Векторное управление электродвигателем «на пальцах». URL: https://habr.com/ru/companies/npf_vektor/articles/367653 (последняя дата обращения 25.03.24).

4. Hoverboard firmware hack FOC. URL: https://github.com/EFeru/hoverboard-firmware-hack-FOC (последняя дата обращения 25.03.24)

5. ESP32 Series Datasheet. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (последняя дата обращения 25.03.24)

6. Learn about Bluetooth. Bluetooth Technology. URL: Overview https://www.bluetooth.com/learn-about-bluetooth/tech-overview (последняя дата обращения 25.03.24)

7. Arduino core for the ESP32. URL: https://github.com/espressif/arduino-esp32 (последняя дата обращения 25.03.24)

8. Read RC PWM signal with Arduino. URL: https://quadmeup.com/read-rc-pwm-signal-with-arduino (последняя дата обращения 25.03.24)

## Код для ESP-32 Bluetooth

```
// ########################## DEFINES ##########################
#include "BluetoothSerial.h"
#define HOVER_SERIAL_BAUD 9600  // [-] Baud rate for HoverSerial (used to communi-
cate with the hoverboard)
#define SERIAL_BAUD 115200      // [-] Baud rate for built-in Serial (used for the
Serial Monitor)
#define START_FRAME 0xABCD      // [-] Start frme definition for reliable serial
communication
#define TIME_SEND 100           // [ms] Sending time interval
#define SPEED_MAX_TEST 300      // [-] Maximum speed for testing
#define SPEED_STEP 5            // [-] Speed step
#define LED_BUILTIN 2
//#define DEBUG_RX                        // [-] Debug received data. Prints all
bytes to serial (comment-out to disable)

#include <SoftwareSerial.h>
SoftwareSerial HoverSerial_front(22, 23);  // RX, TX
SoftwareSerial HoverSerial_rear(18, 19);   // RX, TX
BluetoothSerial SerialBT;

// Global variables
uint8_t idx = 0;          // Index for new data pointer
uint16_t bufStartFrame;  // Buffer Start Frame
byte *p;                 // Pointer declaration for the new received data
byte incomingByte;
byte incomingBytePrev;

typedef struct {
  uint16_t start;
  int16_t steer;
  int16_t speed;
  uint16_t checksum;
} SerialCommand;
SerialCommand Command;

typedef struct {
  uint16_t start;
  int16_t cmd1;
  int16_t cmd2;
  int16_t speedR_meas;
  int16_t speedL_meas;
  int16_t batVoltage;
  int16_t boardTemp;
  uint16_t cmdLed;
  uint16_t checksum;
```

```
} SerialFeedback;


SerialFeedback Feedback_front;
SerialFeedback NewFeedback_front;

SerialFeedback Feedback_rear;
SerialFeedback NewFeedback_rear;

// ########################## SETUP ##########################
void setup() {
  SerialBT.begin("Bespilotnoe_electroshasi");  //Name of your Bluetooth Signal
  Serial.begin(SERIAL_BAUD);
  Serial.println("Bespilotnoe_electroshasi_ready");

  HoverSerial_front.begin(HOVER_SERIAL_BAUD);
  HoverSerial_rear.begin(HOVER_SERIAL_BAUD);
  pinMode(LED_BUILTIN, OUTPUT);
}


// ########################## LOOP ##########################

unsigned long iTimeSend = 0;
int iTest = 0;
int iStep = SPEED_STEP;
int iSteer = 0;

void loop(void) {


  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    char incomingChar = SerialBT.read();
    Serial.write(incomingChar);
    if (incomingChar == '0') {
      iTest = 0;
      iSteer = 0;
    }
    if (incomingChar == '1') iTest = iTest + 10;
    if (incomingChar == '2') iTest = iTest - 10;
    if (incomingChar == '3') iTest = iTest + 50;
    if (incomingChar == '4') iTest = iTest - 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("iTest: ");
    SerialBT.print(iTest);
```

```
    SerialBT.print("  iSteer: ");
    SerialBT.println(iSteer);
  }

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


  unsigned long timeNow = millis();
  // Send commands
  if (iTimeSend > timeNow) return;
  iTimeSend = timeNow + TIME_SEND;

  Send_front(iSteer, iTest);
  Send_rear(iSteer, iTest);

  // Check for new received data
  Receive_front();
  Receive_rear();

  //  Serial.print("iTest: ");
  //  Serial.print(iTest);
  //  Serial.print("  iSteer: ");
  //  Serial.println(iSteer);
}

// ########################## 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));
}


// ########################## SEND REAR ##########################
void Send_rear(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_rear.write((uint8_t *)&Command, sizeof(Command));
}
```

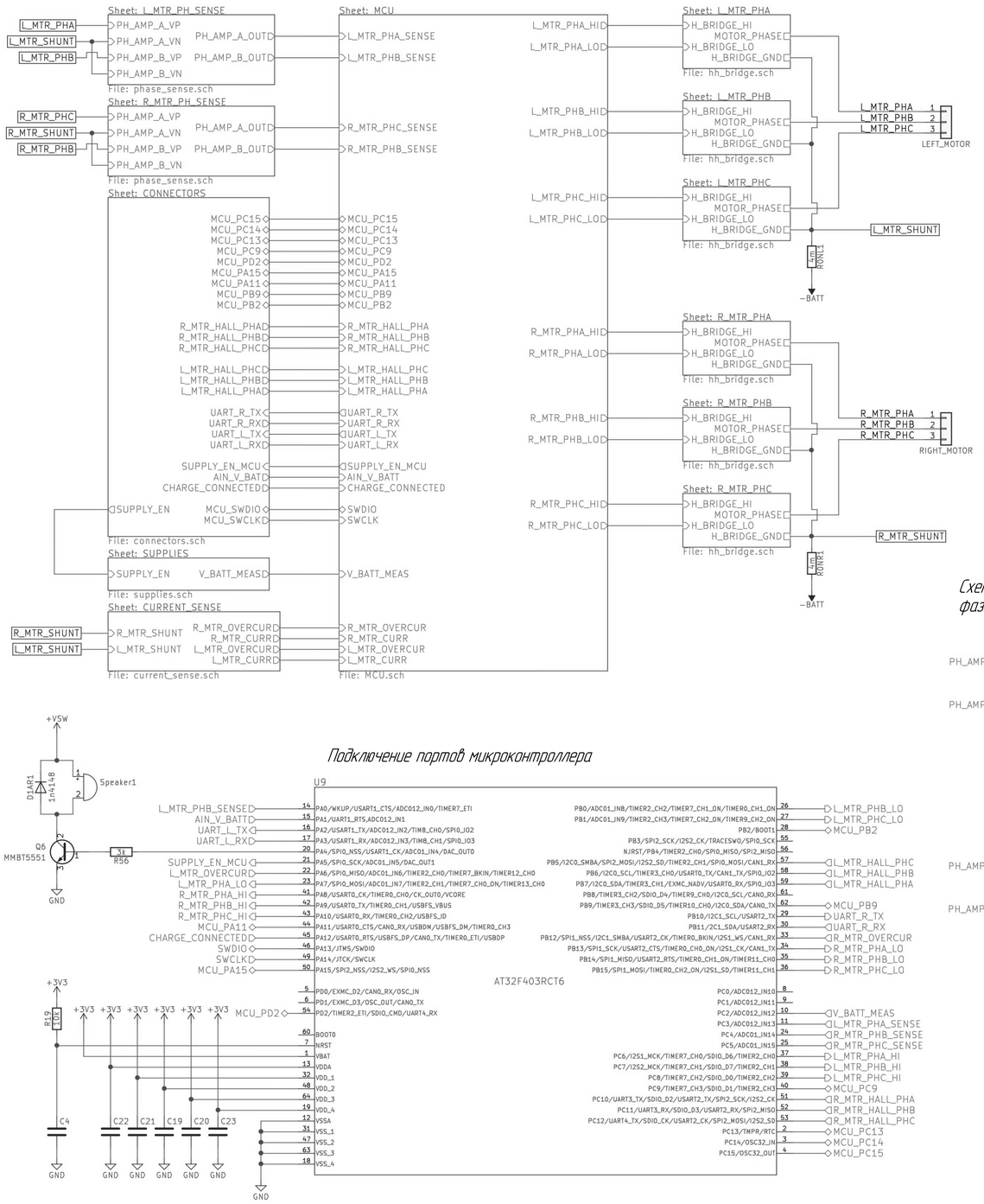
```
// ########################## RECEIVE FRONT ##########################
void Receive_front() {
  // Check for new data availability in the Serial buffer
  if (HoverSerial_front.available()) {
    incomingByte = HoverSerial_front.read();                                // Read
the incoming byte
    bufStartFrame = ((uint16_t)(incomingByte) << 8) | incomingBytePrev;  // Con-
struct the start frame
  } else {
    return;
  }


// If DEBUG_RX is defined print all incoming bytes
#ifdef DEBUG_RX
  Serial.print(incomingByte);
  return;
#endif

  // Copy received data
  if (bufStartFrame == START_FRAME) {  // Initialize if new data is detected
    p = (byte *)&NewFeedback_front;
    *p++ = incomingBytePrev;
    *p++ = incomingByte;
    idx = 2;
  } else if (idx >= 2 && idx < sizeof(SerialFeedback)) {  // Save the new received
data
    *p++ = incomingByte;
    idx++;
  }

  // Check if we reached the end of the package
  if (idx == sizeof(SerialFeedback)) {
    uint16_t checksum;
    checksum = (uint16_t)(NewFeedback_front.start ^ NewFeedback_front.cmd1 ^ New-
Feedback_front.cmd2 ^ NewFeedback_front.speedR_meas ^ NewFeedback_front.speedL_meas
                          ^ NewFeedback_front.batVoltage ^ NewFeed-
back_front.boardTemp ^ NewFeedback_front.cmdLed);

    // Check validity of the new data
    if (NewFeedback_front.start == START_FRAME && checksum == NewFeed-
back_front.checksum) {
      // Copy the new data
      memcpy(&Feedback_front, &NewFeedback_front, sizeof(SerialFeedback));

      // Print data to built-in Serial
      Serial.print("FRONT: ");
```

```
      Serial.print("1: ");
      Serial.print(Feedback_front.cmd1);
      Serial.print(" 2: ");
      Serial.print(Feedback_front.cmd2);
      Serial.print(" 3: ");
      Serial.print(Feedback_front.speedR_meas);
      Serial.print(" 4: ");
      Serial.print(Feedback_front.speedL_meas);
      Serial.print(" 5: ");
      Serial.print(Feedback_front.batVoltage);
      Serial.print(" 6: ");
      Serial.print(Feedback_front.boardTemp);
      Serial.print(" 7: ");
      Serial.println(Feedback_front.cmdLed);
    } else {
      Serial.println("Non-valid data skipped");
    }
    idx = 0;  // Reset the index (it prevents to enter in this if condition in the
next cycle)
  }

  // Update previous states
  incomingBytePrev = incomingByte;
}
// ############################ END RECEIVE FRONT ############################



// ############################ RECEIVE REAR ############################
void Receive_rear() {
  // Check for new data availability in the Serial buffer
  if (HoverSerial_rear.available()) {
    incomingByte = HoverSerial_rear.read();                                  // Read
the incoming byte
    bufStartFrame = ((uint16_t)(incomingByte) << 8) | incomingBytePrev;  // Con-
struct the start frame
  } else {
    return;
  }


// If DEBUG_RX is defined print all incoming bytes
#ifdef DEBUG_RX
  Serial.print(incomingByte);
  return;
#endif

  // Copy received data
  if (bufStartFrame == START_FRAME) {  // Initialize if new data is detected
    p = (byte *)&NewFeedback_rear;
    *p++ = incomingBytePrev;
```

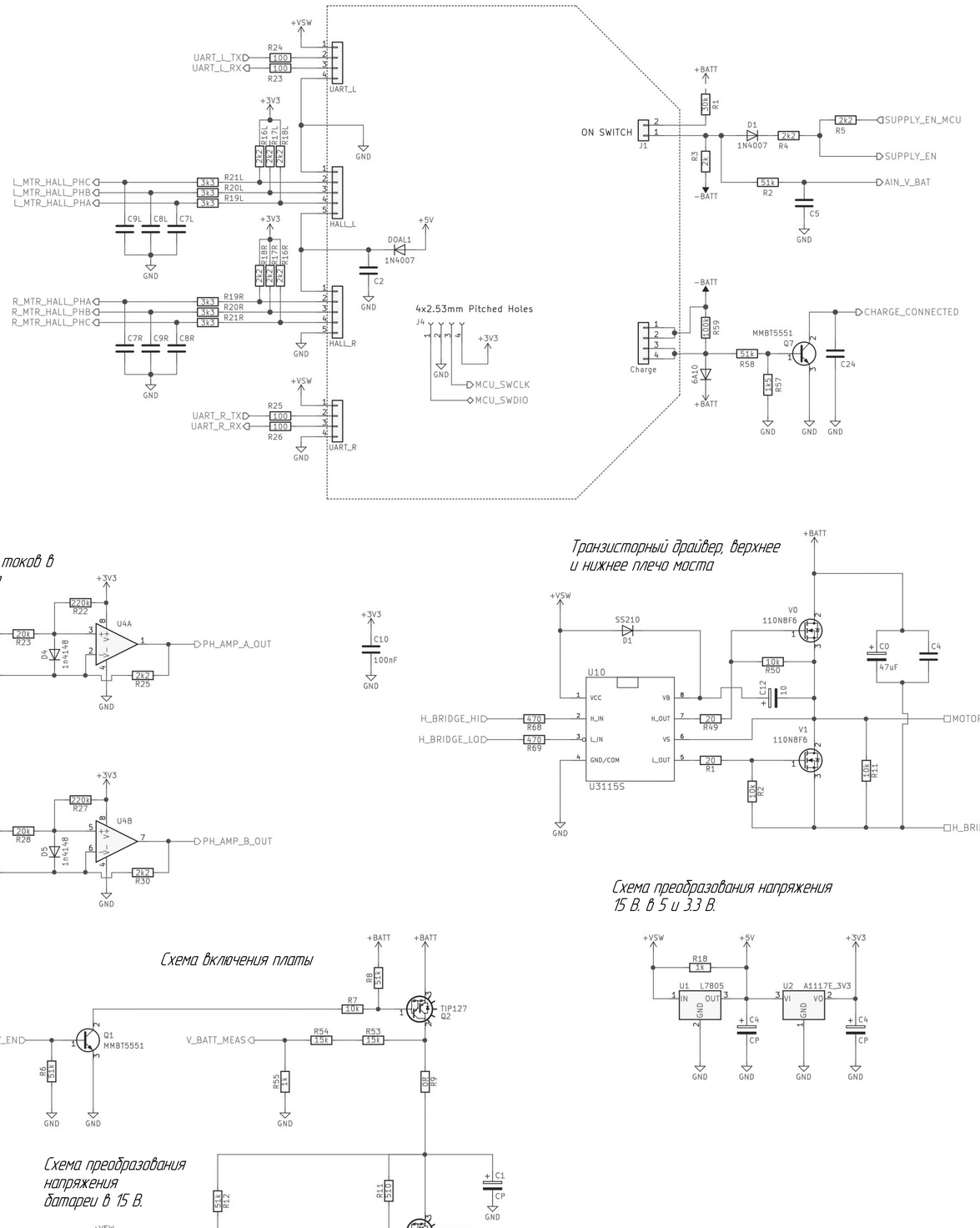
```
    *p++ = incomingByte;
    idx = 2;
  } else if (idx >= 2 && idx < sizeof(SerialFeedback)) {  // Save the new received
data
    *p++ = incomingByte;
    idx++;
  }

  // Check if we reached the end of the package
  if (idx == sizeof(SerialFeedback)) {
    uint16_t checksum;
    checksum = (uint16_t)(NewFeedback_rear.start ^ NewFeedback_rear.cmd1 ^ NewFeed-
back_rear.cmd2 ^ NewFeedback_rear.speedR_meas ^ NewFeedback_rear.speedL_meas
                          ^ NewFeedback_rear.batVoltage ^ NewFeed-
back_rear.boardTemp ^ NewFeedback_rear.cmdLed);

    // Check validity of the new data
    if (NewFeedback_rear.start == START_FRAME && checksum == NewFeed-
back_rear.checksum) {
      // Copy the new data
      memcpy(&Feedback_rear, &NewFeedback_rear, sizeof(SerialFeedback));

      // Print data to built-in Serial
      Serial.print("REAR: ");
      Serial.print("1: ");
      Serial.print(Feedback_rear.cmd1);
      Serial.print(" 2: ");
      Serial.print(Feedback_rear.cmd2);
      Serial.print(" 3: ");
      Serial.print(Feedback_rear.speedR_meas);
      Serial.print(" 4: ");
      Serial.print(Feedback_rear.speedL_meas);
      Serial.print(" 5: ");
      Serial.print(Feedback_rear.batVoltage);
      Serial.print(" 6: ");
      Serial.print(Feedback_rear.boardTemp);
      Serial.print(" 7: ");
      Serial.println(Feedback_rear.cmdLed);
    } else {
      Serial.println("Non-valid data skipped");
    }
    idx = 0;  // Reset the index (it prevents to enter in this if condition in the
next cycle)
  }

  // Update previous states
  incomingBytePrev = incomingByte;
}

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

К.П. 15.03.04.Сх

## Основные системы драйвера двигателя




## Интерфейсы подключения и их обвязка




## Схема измерения токов в фазах двигателя




## Транзисторный драйвер, верхнее и нижнее плечо моста




## Подключение портов микроконтроллера



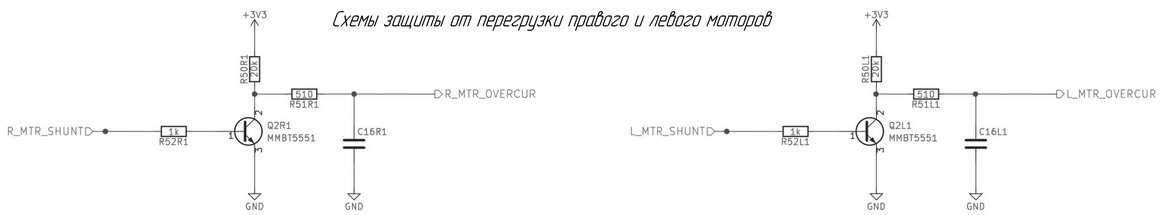

## Схема включения платы



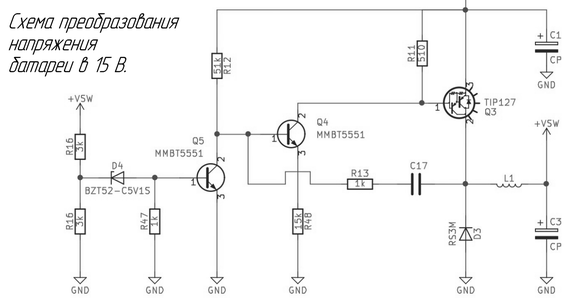

## Схема преобразования напряжения 15 В. в 5 и 3.3 В.




## Схемы защиты от перегрузки правого и левого моторов




## Схема преобразования напряжения батареи в 15 В.




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Изм. | Лист | № докум. | Подп. | Дата | К.П. 15.03.04.Сх | | |
| Разраб. | | Бендик В. И. | | | Схема драйвера электрическая | Лит. | Масса | Масштаб |
| Пров. | | Теличенко Д.А. | | | | У | | |
| Т.контр. | | | | | | Лист 1 | Листов 3 | |
| Н.контр. | | Теличенко Д.А. | | | Автоматизированная система управления электрошассси | АмГУ гр. 041-об | | |
| Утв. | | Теличенко Д.А. | | | | | | |

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

Инв. № подл.

Взам. инв. №

Инв. № дубл.

Подп. и дата

К АКБ

+ − XP1

X1

Sb1

DD2

+ − Подключение батареи

PA1 PA5 Подключение кнопки включ.

M1

Левое переднее колесо

A B C

L_MTR PHA
L_MTR PHB
L_MTR PHC
Подключение фаз левого мотора

HALL VCC 1
HALL PHA 2
HALL PHB 3
HALL PHC 4
GND 5

UART_L: +VSW | PA2 TX | PA3 RX | GND

HALL_L: +5V | HALL_A | HALL_B | HALL_C | GND (1 2 3 4 5)

+3.3V
PA14 SWCLK
GND
PA13 SWDIO

HALL_R: +5V | HALL_A | HALL_B | HALL_C | GND (1 2 3 4 5)

UART_R: +VSW | PB10 TX | PB11 RX | GND

Подключение фаз правого мотора
R_MTR PHA
R_MTR PHB
R_MTR PHC

M2

A B C

1 HALL VCC
2 HALL PHA
3 HALL PHB
4 HALL PHC
5 GND

Правое переднее колесо

DD1

+3.3V
GND
IO22
IO23
IO19
IO18
ESP-32

X2

Sb2

DD3

+ − Подключение батареи

PA1 PA5 Подключение кнопки включ.

M3

Левое заднее колесо

A B C

L_MTR PHA
L_MTR PHB
L_MTR PHC
Подключение фаз левого мотора

HALL VCC 1
HALL PHA 2
HALL PHB 3
HALL PHC 4
GND 5

UART_L: +VSW | PA2 TX | PA3 RX | GND

HALL_L: +5V | HALL_A | HALL_B | HALL_C | GND (1 2 3 4 5)

+3.3V
PA14 SWCLK
GND
PA13 SWDIO

HALL_R: +5V | HALL_A | HALL_B | HALL_C | GND (1 2 3 4 5)

UART_R: +VSW | PB10 TX | PB11 RX | GND

Подключение фаз правого мотора
R_MTR PHA
R_MTR PHB
R_MTR PHC

M4

A B C

1 HALL VCC
2 HALL PHA
3 HALL PHB
4 HALL PHC
5 GND

Правое заднее колесо

| Обозначение | Наименование | Кол. | Примечание |
|---|---|---|---|
| | Двигатели | | |
| M1–M4 | BLDC мотор ob20h36v | 4 | |
| | Кнопки | | |
| Sb1, Sb2 | PSW-6FA-B | 2 | |
| | Разъемы | | |
| XP1, X1, X2 | XT-60 | 3 | |
| | Схемы | | |
| DD1 | ESP-32 | 1 | |
| DD2, DD3 | Драйвер на STM-32 | 2 | |

| Изм. | Лист | № докум. | Подп. | Дата |
|---|---|---|---|---|
| Разраб. | | Бендик В. И. | | |
| Пров. | | Теличенко Д.А. | | |
| Т.контр. | | | | |
| Н.контр. | | Теличенко Д.А. | | |
| Утв. | | Теличенко Д.А. | | |

К.П. 15.03.04.Сх

Схема электрическая электрошассси

Автоматизированная система управления электрошассси

Лит. У | Масса | Масштаб

Лист 2 | Листов 3

АмГУ гр. 041–об

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

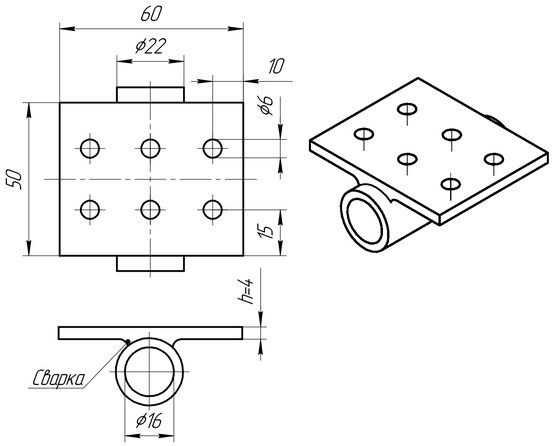
Взам. инв. №

Подп. и дата

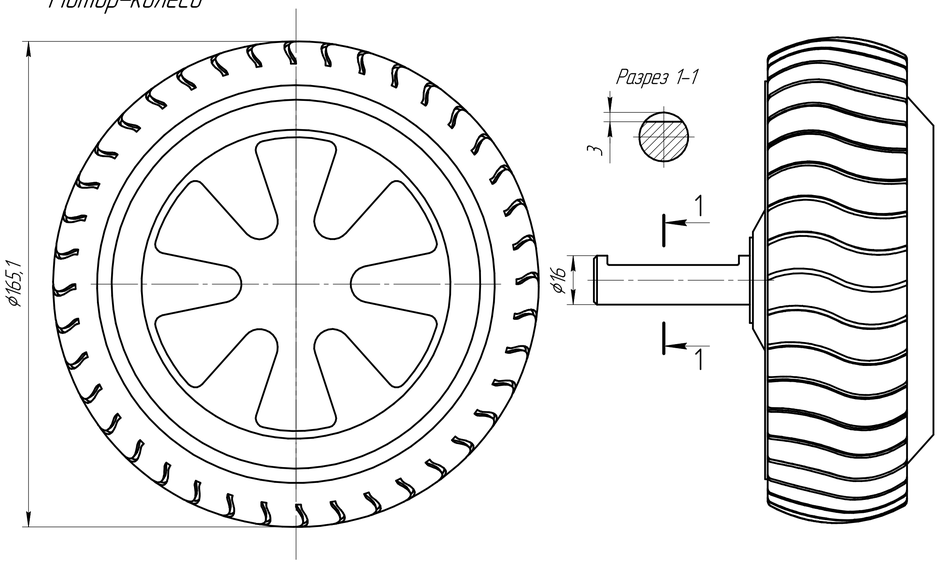
Инв. № подл.

К.П. 15.03.04.06

## Крепление колеса к раме

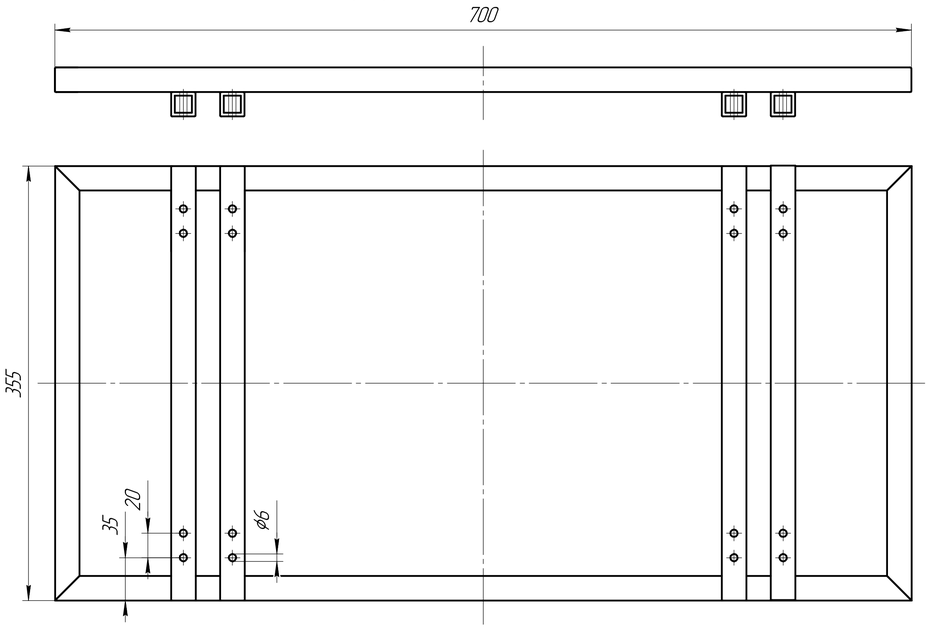



## Мотор-колесо

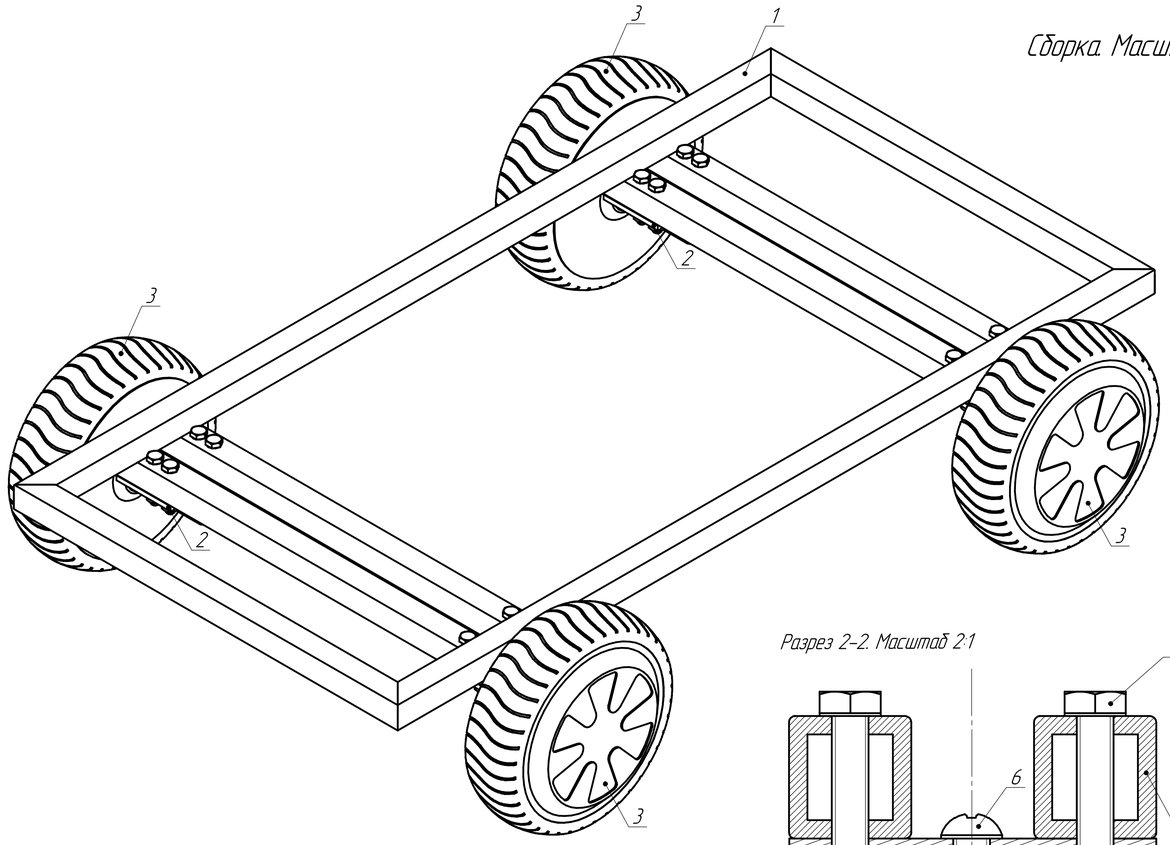


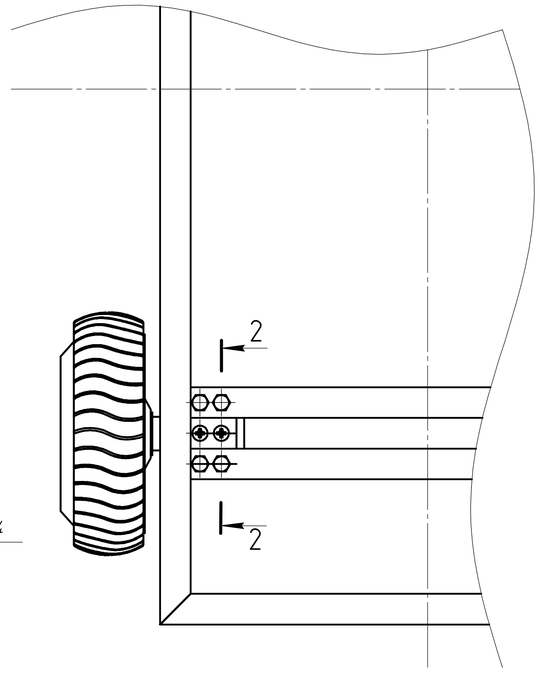

## Рама. Масштаб 1:2.5

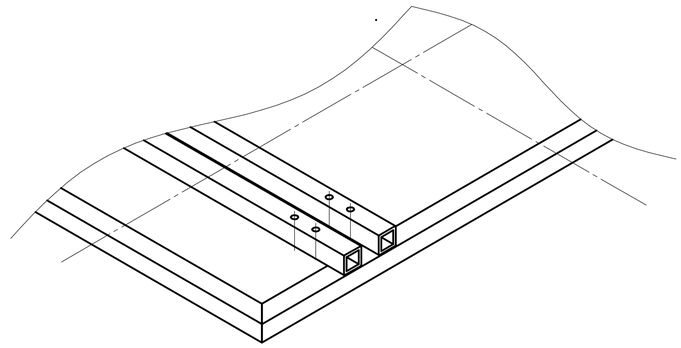
Вся рама выполняется из профилей 20мм. Все соединения выполняются сваркой.



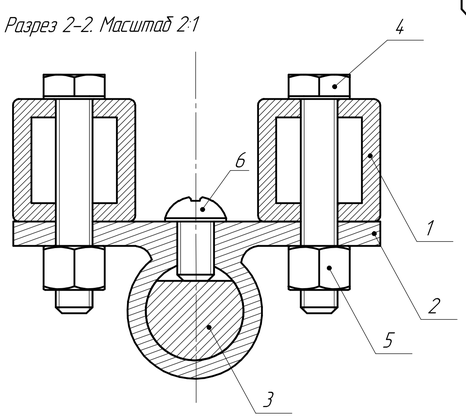

## Сборка. Масштаб 1:2



## Разрез 2-2. Масштаб 2:1




| Формат | Зона | Поз. | Обозначение | Наименование | Кол. | Примечание |
|---|---|---|---|---|---|---|
| | | | | **Детали** | | |
| | | 1 | | Рама | 1 | |
| | | 2 | | Крепление колеса к раме | 4 | |
| | | 3 | | Мотор-колесо | 4 | |
| | | | | **Стандартные изделия** | | |
| | | 4 | | Болт М6х35 ГОСТ 174 73 | 16 | |
| | | 5 | | Гайка М6х7.2 ГОСТ 5915 | 16 | |
| | | 6 | | Винт М6х10 ГОСТ 14.91-80 | 8 | |

| Изм. | Лист | № докум. | Подп. | Дата | К.П. 15.03.04.06 | | | |
|---|---|---|---|---|---|---|---|---|
| Разраб. | | Бендик В. И. | | | Чертеж сборочный | Лит. | Масса | Масштаб |
| Пров. | | Теличенко Д.А. | | | | У | | 1:1 |
| Т.контр. | | | | | | Лист 3 | Листов 3 | |
| Н.контр. | | Теличенко Д.А. | | | Автоматизированная система управления электрошассси | АмГУ гр. 041-об | | |
| Утв. | | Теличенко Д.А. | | | | | | |

Копировал Формат А1

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.