

## ВВЕДЕНИЕ

Среди роботов различных типов нужно выделить отдельную группу мобильных роботов.

Мобильный робот перемещается для решения тех или иных задач, получает данные с внешних датчиков, и должен постоянно обрабатывать информацию, чтобы управлять своим движением. Все эти процессы происходят непрерывно и тесно взаимосвязаны друг с другом.

Мобильные роботы, включающие в себя чувствительные элементы, исполнительные механизмы, компьютеры и обладающие элементами искусственного интеллекта, представляют весьма удобный объект для постановки, изучения и нахождения решений современных проблем мехатроники.

Их создание во многом ещё требует поиска нестандартных решений в разработке их конструкций, алгоритмического, сенсорного и программного обеспечения.

Для всех роботов этой группы свойственны общие признаки, а именно, все они используют: движитель, способный обеспечить передвижение робота в заданной среде; набор внутренних датчиков информации, обеспечивающих в системе управления робота возможность регулирования состояния его систем и формирование требуемого движения его приводных механизмов; локальные и дистанционные средства определения характеристик опорных и/или профильных характеристик окружения робота для автономного исполнения процессов принятия решений о требуемом или возможном движении; интерфейс для взаимодействия (пульт управления).

Целью данного проекта является разработка платформы для тестирования алгоритмов позиционирования робота через внешний блок управления.

В соответствии с поставленной целью были определены следующие задачи:

- моделирование и создание подвижной платформы;
- построение алгоритма управления платформой;
- разработка дистанционного взаимодействия с платформой;
- построение алгоритма позиционирования робота в пространстве в соответствии с внешними данными;
- анализ и реализация алгоритмов обработки полученных данных;
- разработка программного средства.

При проектировании мобильного робота требуется учесть ряд ограничений: габаритных — по компоновке узлов и агрегатов робота, массовых — по весу робота и энергетических — по общему потреблению энергии мобильного робота в активном и неподвижном режимах. Исходя из требований по изучению задач управления, а также указанных ограничений

аппаратная часть мобильного робота включает в себя:

- подвижную платформу;
- четыре колеса;
- четыре редукторных мотора;
- два дальномерных модуля;
- модуль беспроводной связи;
- драйвер управления двигателями;
- внешний блок питания.

Программная часть будет состоять из приложения, которое будет предоставлять следующие функции:

- получение информации с датчиков расстояния;
- управление моторами по отдельности;
- регулирование скорости платформы;
- позиционирование платформы по заданным координатам.

При разработке системы перемещения робота необходимо учитывать следующие моменты:

- скорость или ускорение движения;
- точность позиционирования;
- гибкость и надежность при различных условиях;
- эффективность (низкое энергопотребление).

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор существующих аналогов

На этапе проектирования системы были тщательно изучены существующие аналоги. Одним из наиболее приближенных примеров является мобильный робот «Варан» (рисунок 1.1).

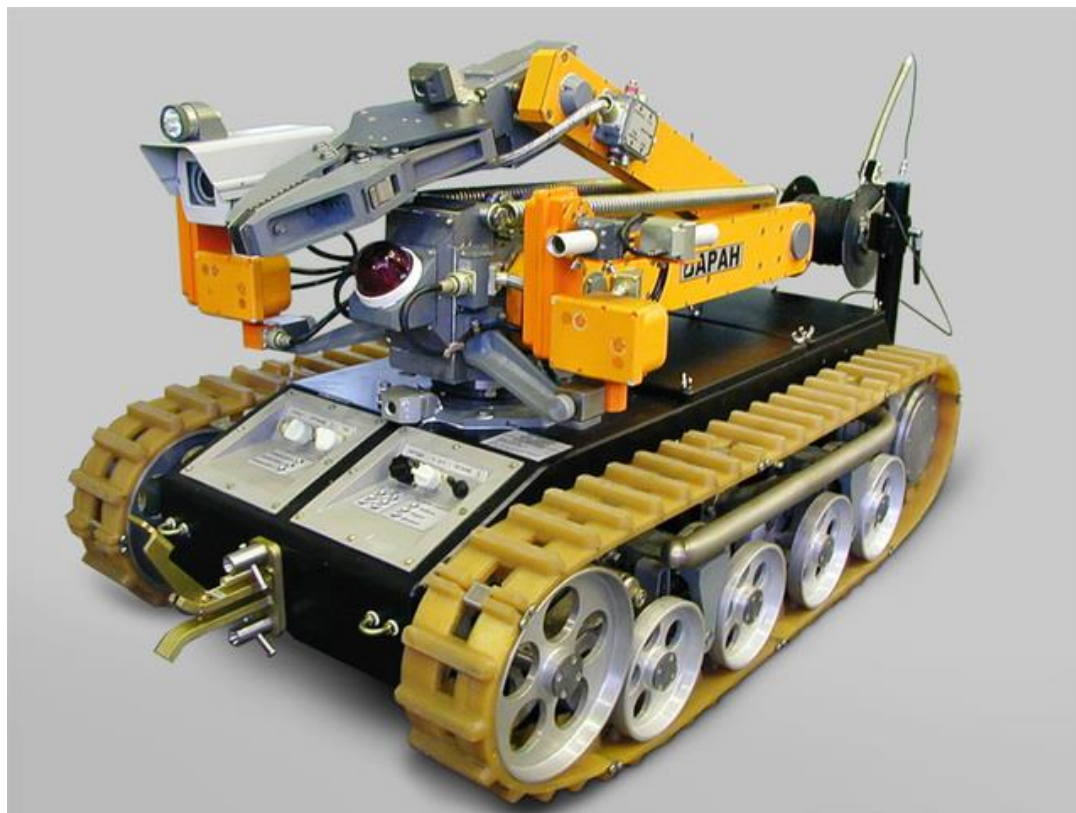


Рисунок 1.1 – Мобильный робот «Варан» [1]

Данный проект – это подвижная, дистанционно управляемая платформа для выявления, обезвреживания и уничтожения взрывных устройств. Мобильный робот «Варан», помимо обнаружения, обезвреживания, уничтожения на месте или доставки в специальном контейнере в безопасное место взрывных устройств, способен также выполнять такие задачи, как ведение разведки в городских или полевых условиях и работы в опасных для здоровья и жизни человека местах (в условиях радиационного, химического и биологического заражения). Он может работать как в управляемом удаленным оператором режиме, так и в автономном режиме, по заранее введенной в него программе. На гусеничную платформу «Варана», в зависимости от поставленной задачи, может устанавливаться различное рабочее оборудование. Например, двухпальцевый манипулятор, системы видеонаблюдения или водомет, служащий для уничтожения взрывных устройств.

Также, к основным недостаткам проекта относятся:

- высокая цена;
- достаточно сложное подключение;
- огромный функционал.

Еще один аналог – мобильный робот «РобоРовер M1 Education» (рисунок 1.2).

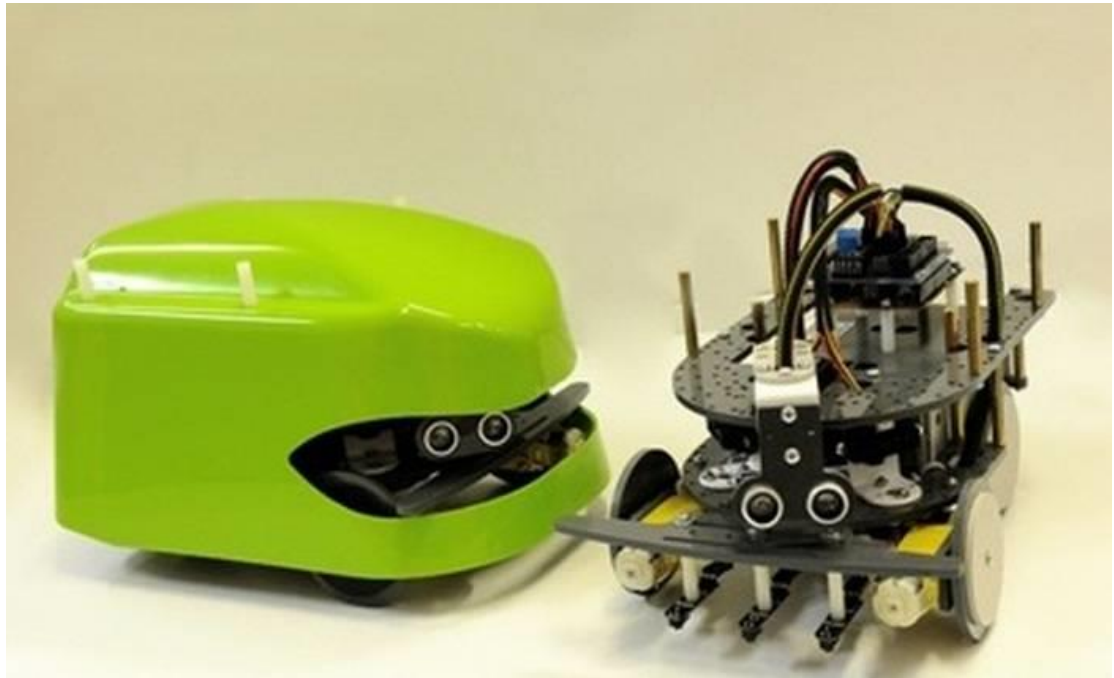


Рисунок 1.2 – Мобильный робот «РобоРовер M1 Education» [2]

Это четырехколесный образовательный робот для практического и нескучного изучения программирования, робототехники и электроники. Робот поставляется полностью собранным и настроенным к работе. В комплекте с роботом поставляется все необходимое для первого запуска: инструкция, аккумулятор, зарядное устройство, отвертка, мини-поле для движения по линии. Робот имеет небольшие размеры для комфортной с ним работы учеником. Робот оснащен двумя оптическими датчиками расстояния Sharp, тремя датчиками линии, одним ультразвуковым датчиком расстояния на поворотном сервоприводе.

К роботу разработана графическая среда программирования под названием РоверБлок. В программе используются блоки, чтобы запрограммировать робота. Каждый блок отвечает за считывание показаний с определенного датчика или за действие при помощи электродвигателя или сервопривода.

Данный проект также не лишен недостатков:

- управление роботом осуществляется только по Bluetooth;
- нет мобильного приложения.

## 1.2 Микроконтроллеры

Современную микроэлектронику трудно представить без такой важной составляющей, как микроконтроллеры. Микроконтроллеры незаметно завоевали весь мир. В последнее время на помощь человеку пришла целая армия электронных помощников. Одно и то же устройство, которое раньше собиралось на традиционных элементах, будучи собрано с применением микроконтроллеров, становится проще, не требует регулировки и меньше по размерам. Кроме того, с применением микроконтроллеров появляются практически безграничные возможности по добавлению новых потребительских функций и возможностей к уже существующим устройствам.

Сегодня на рынке существует множество фирм-производителей, выпускающих различные микроконтроллеры. Рассмотрим несколько из них.

### 1.2.1 Семейства ARM

Микроконтроллеры семейства ARM — семейства лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited.

В основном процессоры семейства завоевали сегмент массовых мобильных продуктов (сотовые телефоны, карманные компьютеры) и встраиваемых систем средней и высокой производительности (от сетевых маршрутизаторов и точек доступа до телевизоров). Отдельные компании заявляют о разработках эффективных серверов на базе кластеров ARM процессоров, но пока это только экспериментальные проекты с 32-битной архитектурой.

Архитектура ARM обладает следующими особенностями RISC:

- архитектура загрузки/хранения;
- нет поддержки нелинейного (не выровненного по словам) доступа к памяти;
- равномерный 16x32-битный регистровый файл;
- фиксированная длина команд (32 бита) для упрощения декодирования за счет снижения плотности кода. Позднее режим Thumb повысил плотность кода;
- одноцикловое исполнение.

### 1.2.2 Микроконтроллеры Arduino

Стоит обратить внимание на микроконтроллеры Arduino. На них нет операционной системы, как на Raspberry Pi, они не сложны в изучении и подойдут как для новичков, так и для более продвинутых пользователей.

Отладочная плата Arduino Uno построена на микроконтроллере Atmega328P.

Платформа пользуется огромной популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду. Устройство программируется через USB без использования программаторов.

Основные преимущества данной платы:

- кроссплатформенность;
- простая среда программирования;
- открытый исходный код;
- открытые спецификации и схемы оборудования.

Наиболее распространенные версии плат: Uno, Leonardo, Nano, Mini, Mega (рисунок 1.3).

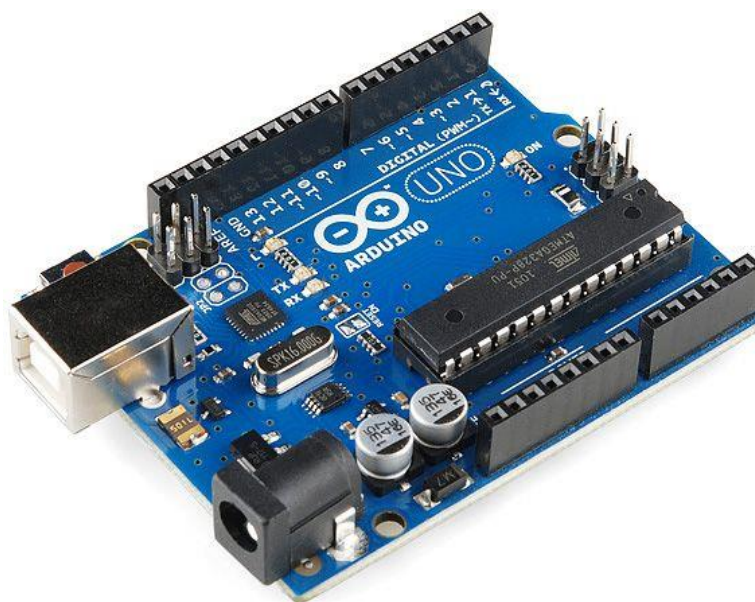


Рисунок 1.3 – Микроконтроллер Arduino Uno [3]

### 1.2.3 Микроконтроллеры AVR

За последние годы микроконтроллеры AVR приобрели большую популярность, привлекая разработчиков достаточно выгодным соотношением показателей «цена/быстродействие/энергопотребление», удобными режимами программирования, доступностью программно-аппаратных средств поддержки и широкой номенклатурой выпускаемых кристаллов. Микроконтроллеры этой серии представляют собой удобный инструмент для создания современных высокопроизводительных и экономичных встраиваемых контроллеров многоцелевого назначения. В частности, они используются в автомобильной электронике, бытовой технике, сетевых

картах и материнских платах компьютеров, в мобильных телефонах нового поколения и т.д.

В рамках единой базовой архитектуры AVR микроконтроллеры подразделяются на три семейства:

- Classic AVR – базовая линия микроконтроллеров;
- Mega AVR - микроконтроллеры для сложных приложений, требующих большого объема памяти программ и данных;
- Tiny AVR — низкостоимостные микроконтроллеры в 8-выводном исполнении.

AVR Classic – самая обширная производственная линия среди других Flash-микроконтроллеров корпорации Atmel. Также это модемы различных типов, современные зарядные устройства, спутниковые навигационные системы для определения местоположения автомобилей на трассе, материнские платы компьютеров. Atmel представила первый 8-разрядный Flash-микроконтроллер в 1993 году и с тех пор непрерывно совершенствует технологию.

Основные особенности микроконтроллеров данного семейства:

- возможность вычислений со скоростью до 1 MIPS/МГц;
- FLASH-память программ объемом от 1 до 8 Кбайт (число циклов стирания/записи не менее 1000);
- память данных на основе статического ОЗУ (SRAM) объемом до 512 байт;
- программирование в параллельном (с использованием программатора) либо в последовательном (непосредственно в системе через последовательный SPI-интерфейс) режимах;
- наличие нескольких режимов пониженного энергопотребления.

AVR Tiny (рисунок 1.4) – это интеллектуальные автомобильные датчики различного назначения, игрушки, игровые приставки, материнские платы персональных компьютеров, контроллеры защиты доступа в мобильных телефонах, зарядные устройства, детекторы дыма и пламени, бытовая техника, разнообразные инфракрасные пульты дистанционного управления.

AVR Mega (рисунок 1.5) – имеют наиболее развитую периферию, наибольшие среди всех микроконтроллеров AVR объемы памяти программ и данных. Предназначены для использования в мобильных телефонах, в контроллерах различного периферийного оборудования (такого как принтеры, сканеры, современные дисковые накопители, приводы CD-ROM/DVD-ROM и т. п.), в сложной офисной технике и т. д.

Микроконтроллеры семейства Mega поддерживают несколько режимов пониженного энергопотребления, имеют блок прерываний, сторожевой таймер и допускают программирование непосредственно в готовом устройстве.



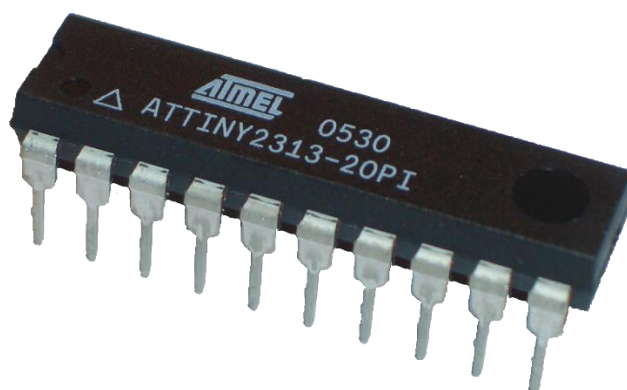


Рисунок 1.4 – Микроконтроллер Attiny2313 [4]

Стоит отметить главную особенность всех вышеперечисленных устройств: все они имеют единую архитектуру, и это позволяет с легкостью переносить код с одного микроконтроллера на другой.

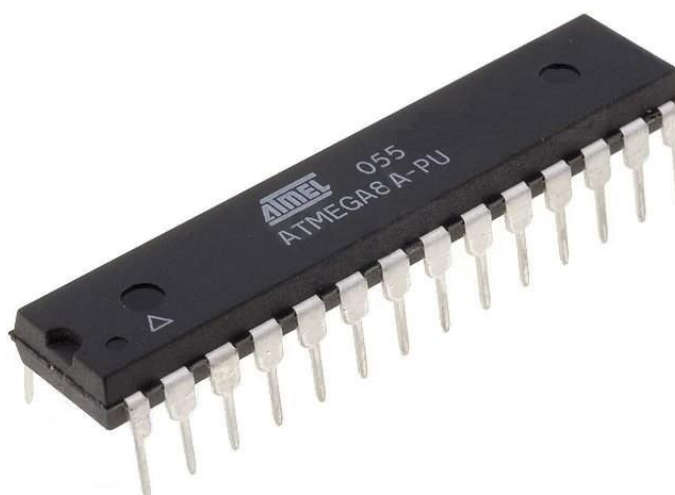


Рисунок 1.5 – Микроконтроллер Atmega8 [5]

### 1.3 API микроконтроллера

В качестве среды разработки для API (application programming interface) микроконтроллера была выбрана технология Microsoft Visual C++. Данная среда содержит большинство компонентов, необходимых для построения программы, а также для подключения и работы с микроконтроллерами. Разработка модуля осуществляется в среде Microsoft Visual Studio 2017.

Для разработки API микроконтроллера необходимо владеть базовыми знаниями о языке C++ и иметь общее представление о принципах объектно-ориентированного программирования.

Также, для удобной разработки API было использован плагин для среды Microsoft Visual Studio 2017 – Visualmicro.

Достоинства Visualmicro:



- поддержка IntelliSense — технология автодополнения Microsoft;
- поддержка нескольких проектов или скетчей в одном решении;
- возможность мониторинга с помощью графических окон о нескольких COM-портов;
- возможность разработки программы с использованием принципов объектно-ориентированного программирования и других сложных конструкций.

## 1.4 Аналитический обзор

Для проектирования мобильного робота был выбрана плата Arduino Uno. Arduino Uno является более оптимальным вариантом решения для проекта, так как у платы есть все необходимые компоненты для обеспечения работы микроконтроллера. Достаточно подключить USB кабель к компьютеру и подать питание. Микроконтроллер установлен на колодке, что позволяет легко заменить его в случае выхода из строя.

Также важным критерием выбора платы послужила энергоемкость, память и количество входов/выводов разного типа.

Arduino Uno может получать питание через подключение USB или от внешнего источника питания. Источник питания выбирается автоматически.

Внешнее питание (не USB) может подаваться через преобразователь напряжения AC/DC (блок питания) или аккумуляторной батареей. Платформа может работать при внешнем питании от 6 В до 20 В. При напряжении питания ниже 7 В, вывод 5V может выдавать менее 5 В, при этом платформа может работать нестабильно. При использовании напряжения выше 12 В регулятор напряжения может перегреться и повредить плату. Рекомендуемый диапазон от 7 В до 12 В.

Arduino Uno контроллер построен на микроконтроллере ATmega328, который располагает 32 кБ флэш памяти, из которых 0.5 кБ используется для хранения загрузчика, а также 2 кБ ОЗУ (SRAM) и 1 Кб EEPROM.

Также плата имеет на борту 6 аналоговых входов, 14 цифровых выводов общего назначения что позволяет подключать дополнительные модули, не задумываясь о нехватке выводов.

В качестве связи с платой Arduino был выбран недорогой Wi-Fi модуль NodeMCU LoLin ESP8266 (рисунки 1.6).

В основу платформы загружена стандартная прошивка Node MCU, в которую встроен интерпретатор языка Lua. При помощи Lua-команд можно выполнять следующие действия:

- подключение к Wi-Fi точке доступа;
- работа в роли Wi-Fi точки доступа;
- переход в режим глубокого сна для уменьшения потребления энергии;
- включение или выключения светодиода на выходе GPIO 16;

- выполнение различные операции с файлами во флэш-памяти;
- поиск открытой Wi-Fi сети, подключение к ней;
- вывод MAC адреса;
- управление пользовательскими таймерами.

Для удобной разработки API была выбрана среда Visual Studio 2017 с дополнительным расширением для разработки программ под микроконтроллеры Arduino на языке программирования C++ и также прошивки самого микроконтроллера. В качестве среды разработки программного средства был выбран язык программирования Python.

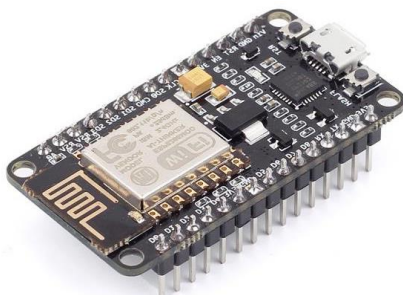


Рисунок 1.6 – Wi-Fi модуль NodeMCU LoLin ESP8266 [6]

### 1.5 Обзор Wi-Fi модулей

В настоящее время большинство пользователей Arduino больше не беспокоятся о цене таких устройств, хотя 3 года назад модуль arduino Wi-Fi считался роскошью. Wi-Fi jammer esp8266, благодаря всем производителям, которые внедрили совершенно новый продукт на рынок, которые внесли значительный вклад и сделали конкуренцию в этой области, довольно дешевы, что довольно дешево, что влияет на его производительность.

Таким образом, arduino wi-fi esp8266 теперь считается самым удобным модулем на рынке, как и все его аналоги, цена иностранной платформы начинается от 2 долларов, что позволяет покупать пакеты модулем и обновлять их тысячи раз, чтобы сохранить эффективность.

Далее представлен список Wi-Fi модулей, их особенностей, а также изображений.

На рисунке 1.7 представлен Wi-Fi модуль модели «ESP» под номером 1.

Для Wi-Fi модуля ESP-01: антенна PCB, после согласования расстояния до открытых 400 метров, проста в использовании.

Питание ESP-01 нужно строго 3.3V, иногда приходится воспользоваться DC-DC преобразователем.

Для Wi-Fi модуля типа ESP-02: SMD для ограничения подачи, антенну можно нарисовать с помощью корпуса заголовка IPX.

Рассмотрим ESP-03. Для данного модуля предусмотрено SMD, керамическая антенна, разведены все доступные GPIO.



ESP-01

Рисунок 1.7 – Wi-Fi модуль ESP-01 [12]

Использование Wi-Fi модуль типа ESP-04 позволит получить ожидаемый результат, но без антенны.

На рисунке 1.8 представлен Wi-Fi модуль модели «ESP» под номером 2.



ESP-02

Рисунок 1.8 – Wi-Fi модуль ESP-02 [12]

В модуле ESP-05 также идет отдельно антенна и из преимуществ можно выделить: разведены только VCC33, GND, TX, RX, RST, миниатюрная антенна.

В ESP-06 из особенностей можно выделить: контактные площадки, расположенные снизу, сверху металлический экран.

На рисунке 1.9 представлен Wi-Fi модуль модели «ESP» под номером 3.



ESP-03

Рисунок 1.9 – Wi-Fi модуль ESP-03 [12]

На рисунке 1.10 изображен Wi-Fi модуль модели «ESP» под номером 4.



ESP-04

Рисунок 1.10 – Wi-Fi модуль ESP-04 [12]

Из преимуществ использования Wi-Fi модулей марки ESP можно выделить дистанционную перепрошивку. С помощью Wi-Fi можно обновить прошивку рабочего устройства. Для этого отделили флэш-память программ на несколько частей. Один присваивается диспетчеру прошивки, остальные два - в пользовательскую программу. Когда они хотят обновить прошивку, новое изображение загружается в свободную часть флэш-памяти. После тщательной проверки целостности недавно загруженного изображения диспетчер прошивок переключает флажок, после чего освобождается область памяти со старой прошивкой, а исполнение кода происходит из новой области.

На рисунке 1.11 изображен Wi-Fi модуль модели «ESP» под номером 5.



ESP-05

Рисунок 1.11 – Wi-Fi модуль ESP-05 [12]

На рисунке 1.12 изображен Wi-Fi модуль модели «ESP» под номером 6.



ESP-06

Рисунок 1.12 – Wi-Fi модуль ESP-06 [12]

Из особенностей для ESP-07 выделяются: керамическая антенна и разъем для внешней антенны, металлический экран.

Кроме того, мощность процессорного ядра достаточна для работы сложных пользовательских приложений для обработки цифрового сигнала. Модуль ESP-07 оснащен встроенным кварцевым резонатором, который полностью обеспечивает работу процессорного ядра и периферийных устройств при подаче питания. ESP-07 связывается с внешними устройствами через 16 краевых выводов, расположенных вдоль двух противоположных краев модуля. Рядом с каждым выходом модуля имеется сквозное монтажное отверстие для пайки линейки штифтов.

На рисунке 1.13 изображен Wi-Fi модуль модели «ESP» под номером 7.



ESP-07

Рисунок 1.13 – Wi-Fi модуль ESP-07 [12]

О модуле ESP-08 можно сказать, что флэш-памяти объемом 1 МБ достаточно для хранения полноценных программных приложений, управляемых обширным набором текстовых AT-команд, но недостаточно для реализации сложных алгоритмов шифрования и аутентификация на основе сертификатов безопасности WPA2-Enterprise.

На рисунке 1.14 изображен Wi-Fi модуль модели «ESP» под номером 8.



ESP-08

Рисунок 1.14 – Wi-Fi модуль ESP-08 [12]

На рисунке 1.15 изображен Wi-Fi модуль модели «ESP» под номером 9.

Рассмотрим особенности модуля Wi-Fi «ESP-09»: поддержка беспроводного стандарта 802.11 b / g / n; Поддержка двух режимов Wi-Fi Direct (P2P), soft-AP; интегрированный стек TCP / IP; встроенный TR-переключатель, усилитель и сетевой координатор; интегрированные PLL, регуляторы, блок управления питанием; выходная мощность в режиме 802.11b: + 19.5dBm; поддержка подключения нескольких TCP-клиентов; ток утечки при отключении питания <10 мкА; встроенный 32-битный микроконтроллер; интерфейсы SDIO 1.1 / 2.0, SPI, UART; STBC, 1 × 1 MIMO, 2 × 1 MIMO; A-MPDU и A-MSDU и защитный интервал 0,4 мс; прием / отправка пакетов <2 мс; Потребляемая мощность в режиме ожидания <1,0 мВт (DTIM3).



ESP-09

Рисунок 1.15 – Wi-Fi модуль ESP-09 [12]

Рассмотрим ESP-10. WiFi-модуль на базе однокиповой системы (SoC) ESP8266. Он мал по сравнению с другими модулями и ULP-технологией. Модуль специально разработан для создания мобильных устройств и Интернета вещей (IoT).

На рисунке 1.16 изображен Wi-Fi модуль модели «ESP» под номером 10.



ESP-10

Рисунок 1.16 – Wi-Fi модуль ESP-10 [12]

В таблице 1.1 отображены особенности Wi-Fi модуля ESP-11.

Таблица 1.1 – Особенности Wi-Fi модуля ESP-11.

Обозначение	Особенность	Чип	Flash	Корпус
ESP-11	8-выводный patch-интерфейс и встроенная керамическая антенна	ESP8266 EX	1 МБ	16x18.5x3 мм

На рисунке 1.17 изображен Wi-Fi модуль модели «ESP» под номером 11.



ESP-11

Рисунок 1.17 – Wi-Fi модуль ESP-11 [12]



## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

### **2.1 Определение структуры комплекса**

Целью проекта является разработка подвижной платформы и управление ей с помощью программного средства.

В разрабатываемой системе можно выделить следующие блоки:

- центральный контроллер;
- блок обмена данными;
- блок беспроводной связи;
- блок управления;
- блок отображения данных;
- блок управления моторами;
- блок питания;
- блок определения местоположения.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.050 С1.

Из данных требований можно сделать вывод о необходимых элементах структурной схемы.

Рассмотрим каждый блок отдельно.

### **2.2 Центральный контроллер**

Центральный контроллер представляет собой контроллер на плате Arduino Uno. Основной задачей данного блока является управление всеми нижеперечисленными блоками: обработка приходящих от них сигналов, принятие решения об изменении состояния системы и само изменение состояний проектируемого объекта.

Управляющий блок взаимодействует напрямую практически со всеми остальными блоками, т.к. является основным вычислительным модулем. На корпусе управляющего блока будет находиться также модуль беспроводного соединения, благодаря которому устройство может быть управляемо по локальной сети с компьютера.

Управляющий блок будет оснащен картой памяти, которая будет кэшировать приходящую информацию, находящейся вне самого устройства.

Также интересной особенностью может являться подключение к устройству напрямую к управляющему блоку для перепрошивки и других настроек. Тем не менее это будет работать и по локальной сети.

### **2.3 Блок обмена данными**

Блок обмена данными необходим для обмена данными между аппаратной частью API микроконтроллера и пользовательской программой.

Является связующим звеном между блоком беспроводной связи и блоками отображения данных и управления. Данный блок предоставляет удобный процесс обмена данными.

## 2.4 Блок беспроводной связи

Блок беспроводной связи необходим для дистанционного управления платформой с помощью программного средства на компьютере.

Используется модуль ESP8266. Управляющее устройство общается с ESP8266 через UART (Serial-порт) с помощью набора AT-команд. Поэтому работа с модулем тривиальна для любой платы с UART-интерфейсом: использовать можно Arduino, Raspberry Pi, что душе угодно.

Работа над приёмом и передачей данных выглядит, как взаимодействие с сырым TCP-сокетом или с serial-портом компьютера.

Более того, модуль можно перепрошивать. Программировать и загружать прошивки можно через Arduino IDE, точно так же, как при работе с Arduino. Реакция на AT-команды — это просто функция штатной прошивки, устанавливаемой на заводе. А вы можете написать свою собственную, если того требует проект. Поскольку на модуле есть 2 порта ввода-вывода общего назначения, вы можете обойтись вовсе без управляющей платы: просто подключите периферию непосредственно к ним.

Для того, чтобы среда Arduino IDE научилась прошивать ESP8266 достаточно добавить директорию с конфигурацией платформы в папку со своими скетчами.

Для физического соединения при прошивке вам понадобится USB-Serial адаптер или плата Arduino/Iskra, настроенная в режим USB-моста.

ESP8266 может работать как в роли точки доступа так и оконечной станции. При нормальной работе в локальной сети ESP8266 конфигурируется в режим оконечной станции. Для этого устройству необходимо задать SSID Wi-Fi сети и, в закрытых сетях, пароль доступа. Для первоначального конфигурирования этих параметров удобен режим точки доступа. В режиме точки доступа устройство видно при стандартном поиске сетей в планшетах и компьютерах. Остается подключиться к устройству, открыть HTML страничку конфигурирования и задать сетевые параметры. После чего устройство штатно подключится к локальной сети в режиме оконечной станции.

В случае исключительно местного использования возможно всегда оставлять устройство в режиме точки доступа, что снижает необходимые усилия пользователя по его настройке.

После подключения к Wi-Fi сети устройство должно получить IP-параметры локальной сети. Эти параметры можно задать вручную вместе с параметрами Wi-Fi либо активизировать какие-либо сервисы автоматического конфигурирования IP-параметров (например, DHCP).

После настройки IP параметров обращение к серверу устройства в локальной сети обычно осуществляется по его IP адресу, сетевому имени (в случае если имена поддерживаются какой-либо технологией, например NBNS) или сервису (в случае если поддерживается автоматический поиск сервисов, например через протокол SSDP).

## **2.5 Блок управления**

Для настройки устройства под конкретного пользователя необходим удобный интерфейс. Блок управления будет представлять собой набор инструментов для программного средства, с которым взаимодействие происходит по локальной сети. Пользователь всегда сможет зайти в программу, подключиться и начать взаимодействовать с платформой, посылая ей команды. Команды управления будут представлять собой набор команд для получения интересующей пользователя информации, а также возможность управления устройством. Например, задавать направления движения платформы, получать координаты местоположения платформы и прочее.

Каждая команда сформирована таким образом, чтобы данный блок сразу преобразовал ее в предпочтительный для себя вид и в соответствии с тем, что пришло, выдает конкретные команды. Тем самым является обработчиком приходящих команд от блока обмена данными.

## **2.6 Блок отображения данных**

Блок отображения данных необходим для отображения данных, которые приходят от блока обмена данными, если блок управления распознает, что это результат пришедшей команды, предназначенной для отображения на экран. Данный блок будет зависеть от блока управления, так как блок управления определяет тип команды, пришедшей от блока обмена данными, и проверяет, следует ли ее передавать на блок отображения данными.

## **2.7 Блок управления моторами**

Блок управления моторами предназначен для регулирования скорости моторов, следит за корректной работой отдельных моторов и т.д.

Блок управления моторами выполняет крайне важную роль в проектах ардуино, использующих двигатели постоянного тока или шаговые двигатели. В данном проекте использовался популярный драйвер двигателей на базе микросхемы L298N.

Как известно, плата Arduino имеет существенные ограничения по силе тока присоединенной к ней нагрузки. Для платы это 800 mA, а для каждого отдельного вывода — и того меньше, 40mA. Мы не можем подключить

напрямую к Arduino Uno, Mega или Nano даже самый маленький двигатель постоянного тока. Любой из этих двигателей в момент запуска или остановки создаст пиковые броски тока, превышающие этот предел.

Модуль используется для управления шаговыми двигателями с напряжением от 5 до 35 В. При помощи одной платы L298N можно управлять сразу двумя двигателями. Наибольшая нагрузка, которую обеспечивает микросхема, достигает 2 А на каждый двигатель. Если подключить двигатели параллельно, это значение можно увеличить до 4 А.

## 2.8 Блок питания

Блок питания предназначен для обеспечения корректного и стабильного питания для функционирования платформы.

Большинство плат требует наличие питания в диапазоне от 4.5 до 9 вольт через разъем внешнего питания и 4.5-5 вольт через USB. Однако в инструкции написано 7-12 вольт, то есть будем считать, что оптимальным вариант это 9 вольт.

Питание от 5 вольт. Этот вариант питания от компьютера. Реализовать такое питание можно также от зарядного устройства телефона или купив преобразователь. Если же уже все сделано и плата прошита, то напряжение 5 вольт будет недостаточно. В этом случае при значительных нагрузках на выходы, возможны провалы в работе. Относительно данного проекта 5 вольт недостаточно.

Питание от 9 вольт. Arduino можно запитать от батарейки "Крона" или блока пальчиковых батареек. На холостом ходу или с минимальной нагрузкой она проработает не один месяц. А вот уже с небольшим увеличением нагрузки время автономной работы быстро сойдет на нет. Если увеличить нагрузку на батарейку в виде датчиков требующих больше мощности и светодиодов индикации, то батарейки может хватить совсем ненадолго.

## 2.9 Блок определения местоположения

Блок определения местоположения отвечает за определение местоположения относительно заданной системы координат.

Пока основной проблемой всех ныне существующих мобильных аппаратов, перемещающихся самостоятельно, без управления со стороны человека, остается навигация. Для успешной навигации в пространстве бортовая система робота должна уметь строить маршрут, управлять параметрами движения (задавать угол поворота колес и скорость их вращения), правильно интерпретировать сведения об окружающем мире, получаемые от датчиков, и постоянно отслеживать собственные координаты.

Полноценный робот должен определять собственные координаты и выбирать направление движения только на основании показателей бортовых датчиков, поэтому системы искусственного интеллекта, создаваемые для

автономных машин, ориентированы на поддержку непрерывного цикла "опрос датчиков – принятие оперативного решения об изменении маршрута". Таких циклов может быть несколько – один ответственен за следование по основному маршруту, другой – за обход препятствий и т. д. Кроме того, на аппаратном уровне каждый цикл может поддерживаться датчиками разных типов и разных принципов действия, формирующих потоки данных разного объема и интенсивности.

В результате робот начинает теряться в сложной обстановке и на длинных маршрутах, когда надо не просто обходить мелкие препятствия и уклоняться от опасностей на относительно прямом пути, а планировать долгосрочные действия на стратегическом уровне и выполнять ряд вспомогательных задач, которые весьма трудоемки сами по себе. Поэтому современные системы навигации объединяют механизмы как низкоуровневого управления, так и высокоуровневого планирования.

Проблемы, непосредственно связанные с движением на текущем коротком отрезке маршрута, решаются путем простого реагирования на особенности внешней среды, а глобальная система следит за соблюдением общего плана, модифицируя его в случае необходимости, и синхронизирует работу всех подчиненных структур управления.

### 3 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ

В разделе разработки функциональной схемы рассматривается проектирование устройства на функциональном уровне, в соответствии с полученной ранее структурной схемой.

Данный дипломный проект включает в себя 2 части: аппаратную и программную, которые будут рассматриваться в следующих пунктах подробно.

Стоит отметить, что для обмена данными и командами между аппаратной и программной частью были использованы сокеты. Сокеты позволяют подключаться к адресу и порту в локальной сети, на который настроен wi-fi модуль и общаться посредством отправки строк через консоль.

#### 3.1 Аппаратная часть

Аппаратная часть представляет собой некоторый набор из блоков, которые были определены в ходе анализа структурной схемы, а именно:

- центральный контроллер. Представляет собой микроконтроллер Arduino Uno.

- блок беспроводной связи. Представляет собой wi-fi модуль ESP8266. Имеет плату расширения для возможности подключения нескольких устройств для взаимодействия.

- блок управления моторами – драйвер моторов Motor Shield L298N, подключается к центральному контроллеру.

- блок питания. Используется для питания устройства и представляет собой бокс с 3-мя аккумуляторами.

- блок определения местоположения – хорошо известный датчик определения расстояния HC-SR04, который напрямую подключен к центральному контроллеру.

Рассмотрим подключение данных блоков, которые и составляют основу функциональной схемы.

##### 3.1.1 Подключение и назначение плат и элементов

Рассмотрим основные характеристики платы Arduino Uno.

В таблице 3.1 представлены основные характеристики главного контроллера.

Чтобы знать, как подключать те или иные платы и элементы к центральному контроллеру, нужно знать распиновку для этого контроллера и какой пин за что отвечает, а также изображение данного контроллера для дальнейшего понимания и понимания расположения пинов.

В таблице 3.2 представлена распиновка микроконтроллера Arduino Uno.

В последующем описании подключения блоков будет использоваться терминология и обозначения из данной таблицы.

Таблица 3.1 – Основные характеристики Arduino Uno.

Микроконтроллер	ATmega328
Рабочее напряжение, В	5
Входное напряжение (рекомендуемое), В	7-12
Входное напряжение (предельное), В	6-20
Цифровые Входы/Выходы	14
Аналоговые входы	6
Постоянный ток через вход/выход, мА	40
Постоянный ток для вывода 3.3 В, мА	50
Флеш-память, Кб	32
ОЗУ, Кб	2
EEPROM, Кб	1
Тактовая частота, МГц	16

Таблица 3.2 – Распиновка микроконтроллера Arduino Uno

Пин ардуино	Адресация на плате	Специальное назначение
1	2	3
Цифровой пин 0	0	RX
Цифровой пин 1	1	TX
Цифровой пин 2	2	Ввод для прерываний
Цифровой пин 3	3	
Цифровой пин 4	4	
Цифровой пин 5	5	
Цифровой пин 6	6	
Цифровой пин 7	7	
Цифровой пин 8	8	



*Продолжение таблицы 3.2*

1	2	3
Цифровой пин 9	9	
Цифровой пин 10	10	SPI(SS)
Цифровой пин 11	11	SPI(MOSI)
Цифровой пин 12	12	SPI(MISO)
Цифровой пин 13	13	SPI(SCK)
Аналоговый пин A0	A0 или 14	
Аналоговый пин A1	A1 или 15	
Аналоговый пин A2	A2 или 16	
Аналоговый пин A3	A3 или 17	
Аналоговый пин A4	A4 или 18	I2C (SCA)
Аналоговый пин A5	A5 или 19	I2C (SCL)
VIN	VIN	
GND	GND	Вывод земли
5V	5V	Запитывание устройства (5В)
3.3V	3.3V	Запитывание устройства (3.3В)

Рассмотрим расположение пинов для платы центрального контроллера в графическом варианте. Сделано это для более подробного представления и ознакомления с разработкой функциональной схемы.

На рисунке 3.1 представлено расположение пинов на Arduino Uno.

После подробного ознакомления с Arduino Uno можно знакомиться глубже с другими блоками и их подключением.

Рассмотрим самый главный блок по значимости после центрального контроллера – блок беспроводной связи.

Блок беспроводной связи – wi-fi модуль с платой расширения. Является ключевым компонентом дипломного проекта и функциональной схемы. Представляет собой соединительное звено между аппаратной частью и программной. Этот модуль имеет возможность подключения к любой

открытой wi-fi точке доступа, возможность конфигурирования и переподключения в активном режиме работы.

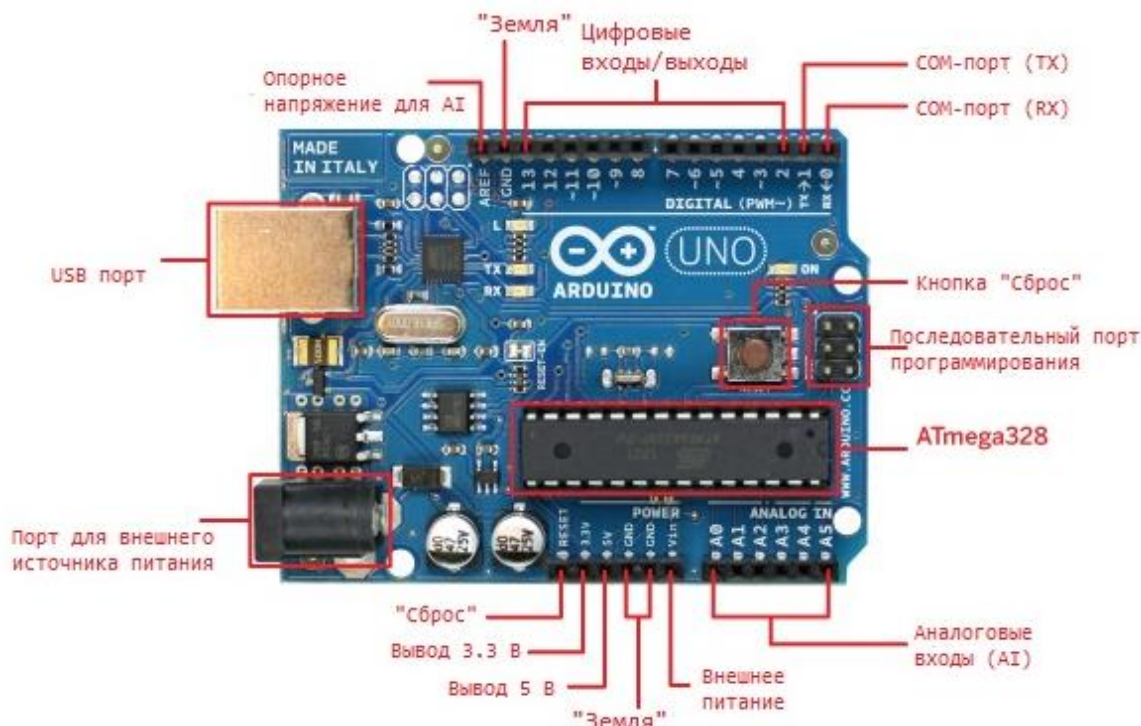


Рисунок 3.1 – Расположение пинов на Arduino Uno [7]

В своем арсенале возможностей модуль беспроводной связи имеет способность напрямую связываться и передавать данные центральному контроллеру, Arduino Uno. Происходит это посредством использования пинов RX, TX на центральном контроллере и соответственно на wi-fi модуле. Запитывание wi-fi модуля может происходить через USB-кабель, а также через пины 5V и GND.

В связке с беспроводным модулем идет плата расширения. Она позволяет, не задумываясь, подключать несколько устройств напрямую к модулю и обеспечивать корректную связь и передачу данных. Также на плате расширения помимо пинов, через которые идет связь – RX, TX, присутствует большой набор пинов 5V и GND, что в свою очередь позволяет не только обеспечивать связь в передачи данных, но и поддерживать работоспособность устройств, т.е. обеспечивать дополнительное запитывание.

На рисунке 3.2 представлена подробная распиновка wi-fi модуля ESP8266 NodeMCU v.3 Lua.

На рисунке 3.3 показано соединение платы расширения и wi-fi модуля. В дальнейшем будет рассматриваться именно такая связка.

Нужно обратить внимание, что обеспечение связи между модулем беспроводной связи и центральным контроллером происходит через

подключение RX и TX пинов на центральном контроллере к TX и RX пирам на wi-fi модуле. Только так можно наладить обмен данными напрямую.

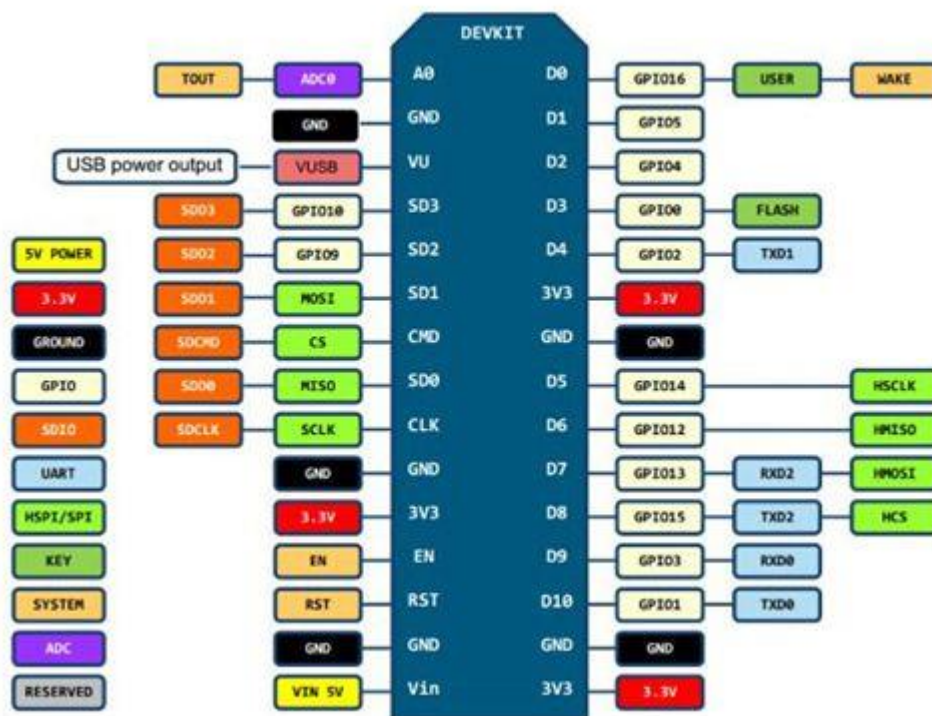


Рисунок 3.2 – Расположение пинов на ESP8266 NodeMCU v.3 Lua [8]

Следующим для рассмотрения возьмем блок, отвечающий за управление моторами. Максимальное количество моторов, которые можно подключить к плате L298N равняется 4. В данном дипломном проекте как раз и использовалось 4 мотора. Для удобной реализации движения и поворотов платформы было решено объединить по 2 мотора с правой и левой стороны вместе. Это решает проблему поиска простого решения управления моторами. В случае поворота в левую сторону активными остаются моторы с правой стороны платформы, в случае поворота направо – с левой. В случае движения платформы вперед или назад активными остаются все 4 мотора, которые, соответственно, вращаются в нужном направлении.

Рассмотрев логику объединения моторов, можно глубже ознакомиться с распиновкой модуля управления моторами и, конечно, подключением его к центральному контроллеру.

В случае подключения платы обратимся к рисунку выше. Из него следует, что плата L298N обладает слотами для запитывания и земли.

Для максимальной производительности платы и активного поведения моторов нужно использовать питание в диапазоне 5-12В. Отдельно про блок, отвечающий за питание устройства, будет рассмотрено далее.

Подключение происходит довольно просто – от пина «5V» центрального контроллера протягиваем провод «папа-папа» к слоту с

запитыванием платы L298N. С пином «GND» поступает точно таким же образом.

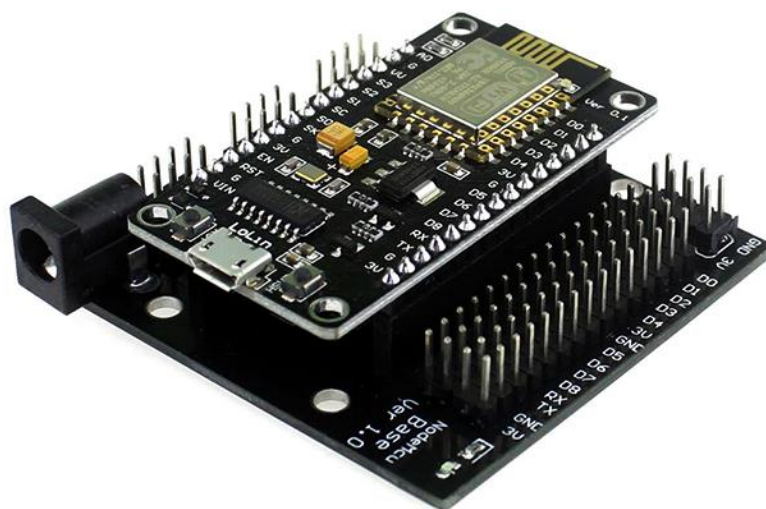


Рисунок 3.3 – Wi-fi модуль в связке с платой расширения [9]

На рисунке 3.4 представлена распиновка платы L298N.

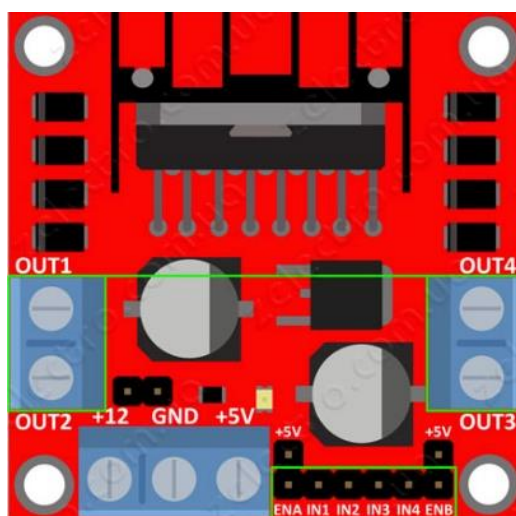


Рисунок 3.4 – Распиновка модуля управления моторами

На данный момент плата, отвечающее за управление моторами запитано от центрального контроллера. Теперь нужно обеспечить взаимосвязь и прослушивание команд от центрального контроллера на плате L298N.

Обращаясь к рисунку 3.4 можно заметить, что пины «IN1» – «IN4» необходимо подключить к цифровым входам центрального контроллера. Пины «IN1» – «IN4» отвечают за слоты, куда подключаются моторы, а также обеспечивают обмен и прослушивание команд посылаемых от центрального контроллера.

Итак, плата L298N успешно подключена и запитана.

Перейдем к рассмотрению блока питания.

Данный блок представляет собой обычный бокс с 4-мя аккумуляторами, дающий на выходе 9В и обеспечивающий надежным и долгим питанием все устройство.

На рисунке 3.5 изображен бокс с 4-мя аккумуляторами и разъемом для подключения к Arduino uno.



Рисунок 3.5 – Блок питания

Блок питания имеет разъем для подключения к Arduino Uno.

Следующим блоком является блок определения местоположения. Данный блок представляет собой датчик расстояния HC-SR04 и способен в течение очень малого промежутка времени отсылать инфракрасные лучи для определения препятствий, вспомогательных стен и т.д.

Как происходит взаимодействие датчика с центральным контроллером можно понять если углубиться в его распиновку.

На рисунке 3.6 представлена распиновка модуля, отвечающего за определение местоположения платформы.

Vcc – отвечает за запитывание датчика. Требуемое напряжение равняется 5В.

Ground – обеспечивает землю и подключается к земле на центральном контроллере.

Trigger – вход, импульс 10 мкс уровень TTL.

Echo – выход, сигнал уровень TTL – ШИМ длительность от 150 мкс до бесконечности если нет эха.

Подключение датчика происходит напрямую к центральному контроллеру. Пины «VCC», «GND» подключаются напрямую к таким же пинам.

Пины «Echo» и «Trigger» датчика определения местоположения обеспечивают обмен информацией, получаемой датчиком и сообщением ее центральному контроллеру. Подключение данных пинов происходит так же



напрямую. Пины «Echo» и «Trigger» датчика подключаются к цифровым пинам Arduino Uno.

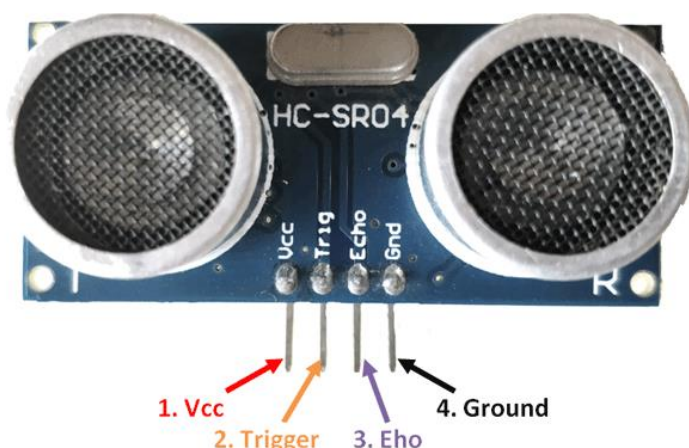


Рисунок 3.6 – Распиновка датчика HC-SR04

Датчик HC-SR04 питается от центрального контроллера и voltage равняется 5В, что в свою очередь обеспечивает корректную и надежную работу датчика.

Следует отметить, что сам датчик следует располагать осторожно на платформе и направить его следует ровно прямо.

Так же происходят иногда сбои в показаниях датчика. Это обуславливается близким расположением предметов или препятствий.

Центральный контроллер в свою очередь обрабатывает входящую информацию с датчика и посредством алгоритма определения местоположения устанавливает местоположение платформы.

Объединение данных модулей рассмотрим в пункте 3.1.2

А теперь остановимся подробнее на функционировании аппаратной части, рассмотрим методы и свойства.

Исходный код аппаратной части делится на 2 раздела:

- главный контроллер;
- wi-fi модуль.

Рассмотрим методы главного контроллера.

Метод *setup* представляет собой точку входа.

В нем инициализируются пины устройства, активируется серийный порт и wi-fi модуль.

Метод *loop* представляет бесконечный цикл программы, в котором выполняются методы. Как и предыдущий метод является основным методом для работы программы.

В данном методе происходит прием входящих команд от клиента, а именно когда клиент подключен напрямую к Arduino Uno.

Метод *handleCmd* является обработчиком входящих команд от

контроллера.

В таблице 3.3 представлен перечень команд для настройки wi-fi модуля, которые выполняются на контроллере.

Таблица 3.3. – Перечень команд для главного контроллера.

Команда	Назначение
GET_IP_ADDRESS	Команда для получения IP-адреса, к которому подключен wi-fi модуль
SET_SSID	Команда для установления SSID. SSID – имя wi-fi точки
GET_SSID	Команда для получения SSID
GET_LOCAL_SSID	Команда для получения SSID в случае, если wi-fi модуль автоматически подключился к wi-fi точке
SET_PASSWORD	Команда для установления пароля для подключения к wi-fi точке
GET_PASSWORD	Команда для получения пароля wi-fi точки
CONNECT_TO_WIFI	Команда, которая использует данные SSID и password, находит данную wi-fi точку по SSID и инициирует подключение
WIFI_SESSION	Команда, которая активирует wi-fi сессию. Wi-fi сессия используется для работы с wi-fi модулем, напрямую посылая команды ему с пользовательского компьютера с помощью сокетов

Метод *movementMenu* предоставляет пользователю, который запрашивает данные о возможностях контроллера, меню движения.

В данном меню можно выбрать и установить движение платформы. Как и в обычном управлении роботом, можно выбрать 4 стороны направление: прямо, назад, влево, вправо. Также возможно команда «стоп» и выход из меню.

За движение робота отвечают методы:



- moveForward;
- moveBack;
- moveRight;
- moveLeft;
- moveStop.

Для осуществления движения в основной библиотеке Arduino предусмотрены методы установления высокого и низкого напряжений.

Для примера возьмем движение вперед. Для этого нужно сделать следующее:

```
digitalWrite (RIGHT_DOWN, LOW);
digitalWrite (RIGHT_UP, HIGH);
digitalWrite (LEFT_UP, HIGH);
digitalWrite (LEFT_DOWN, LOW);
```

где RIGHT\_DOWN – пин на модуле управления моторами, отвечающий за движение правой стороны вниз,

RIGHT\_UP – пин на модуле управления моторами, отвечающий за движение правой стороны вверх,

LEFT\_DOWN – пин на модуле управления моторами, отвечающий за движение левой стороны вниз,

LEFT\_UP – пин на модуле управления моторами, отвечающий за движение левой стороны вверх.

В случае движения в другие стороны и полной остановки используется тот же принцип.

### 3.1.2 Межсетевое взаимодействие на аппаратном уровне

Межсетевое взаимодействие - это структура связи, которая заключается в подключении к локальной и глобальной сети, ее основная функция - быстро запрашивать информацию в соответствии с запросом и в полной концентрации. Современные пользователи все больше зависят от своей сети. Хаос, который возникает в офисе после сбоя серверов узлов или групп.

После прибытия факсимильного аппарата люди перестали просить его, но только начали просить их номер. Теперь интернет широко доступен как владение факсом. Это означает, что для поддержания своей эффективности, на основе которой она модернизируется на основе сети, с этого дня (или даже в то же время) необходимо эффективно управлять:

- изображением и графическими образами;
- файлы с большими размерами;
- приложение клиент-сервер;
- чрезвычайно сильно связанный с сетью трафик.

Чтобы удовлетворить эти потребности, пользователи объединенной части должны предоставить следующее:

- увеличение пропускной способности;

- полосу пропускания по запросу;
- низкая латентность;
- возможности хранения данных, звука и видео.

Кроме того, нужно быть готовы к созданию современной сети, которая легко адаптируется для завтрашних приложений. В ближайшем будущем сеть должна быть готова к обработке:

- высокоточных графиков;
- полноразмерных видео;
- цифрового звучания.

По сути, для осуществления своей цели, можно объединить разные сети, чтобы объединиться для обслуживания зависимых от них организаций. И это не должно происходить без учета физической среды, связанной с этой средой. Компании, которые расширяют свою сеть, должны преодолевать пределы физической и географической юрисдикции. Интернет работает как модель для этого роста.

Сетевые устройства были разработаны для работы в ограниченных географических зонах локальной сети, таких как строительный этаж или независимое здание. Локальные сети объединяют персональные компьютеры в один, чтобы они могли получить доступ к сетевым ресурсам, таким как принтеры и файлы. LAN подключает физически подключенные компьютеры к сетевой среде или кабелю. Некоторые сетевые устройства включают в себя повтор, пул, концентратор, коммутатор, маршрутизатор и шлюз.

Повторители. Повторитель восстанавливает и распределяет эти устройства из одного сегмента сети в другой. Они не меняют адрес или данные; Повторители не могут фильтровать пакеты, даже если повторители не могут фильтровать пакеты, поэтому повторители могут расширить размер сети, восстановив потерянный сигнал, помните, что использование ретранслятора позволяет только объединить два сегмента сети в одной сети. На рисунке 3.7 изображен повторитель в сети.

Иногда поставщики хранятся в исходных компьютерах и компьютерах-получателях для компенсации ухудшенного сигнала. Приводит ко времени ожидания: временная задержка, необходимая для перемещения сигнала с конечного компьютера на конечный компьютер.

Мосты. Эти устройства также смогли восстановить сигнал, но они более интеллектуальны, чем устройства повторителей. Пул может прочитать аппаратные адреса из кадра назначения или фрейма данных и решить, находится ли целевой компьютер в локальной области - том, с которого был обрамлен фрейм, или в другом разделе сети. Если конечный компьютер находится в локальной области, рамка моста не продвигается вперед. Если конечный компьютер не находится в локальной области, пул отправляет кадры во всех других сегментах сети. На рисунке 3.8 изображена работа моста в локальной сети.

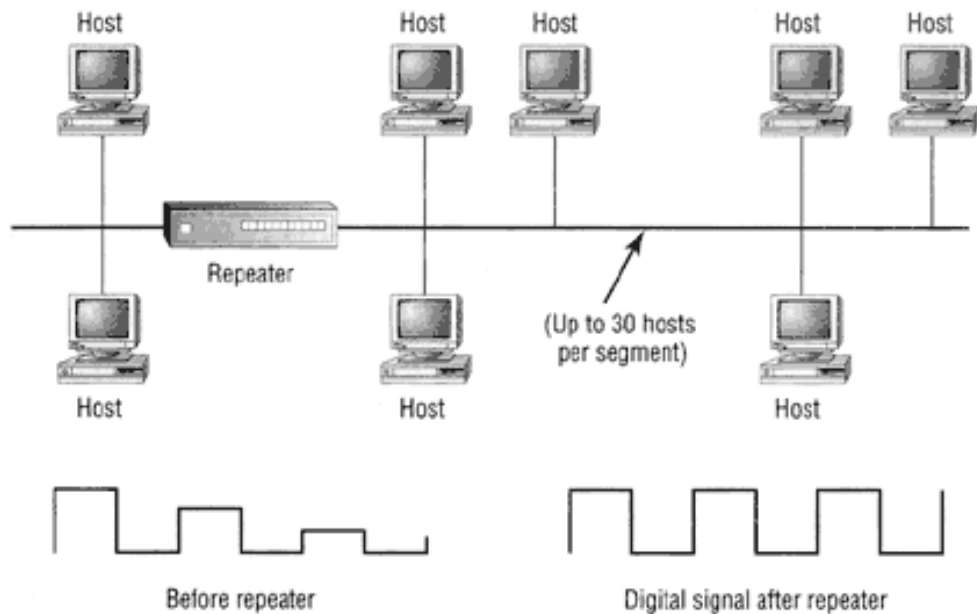


Рисунок 3.7 – Повторитель в сети

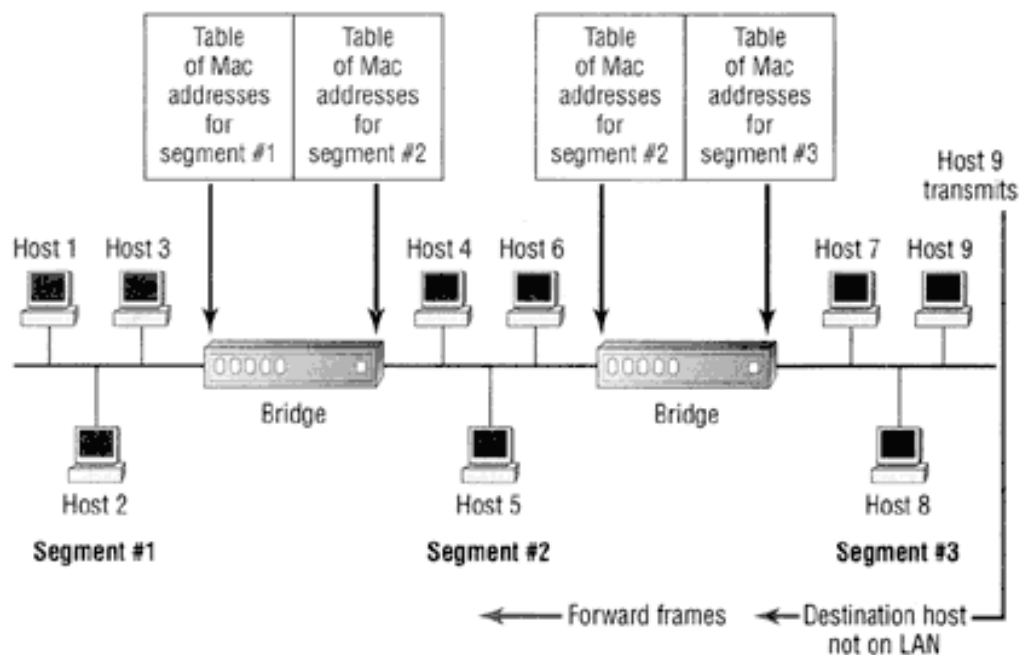


Рисунок 3.8 – Мост в сети

Сегментация локальной сети Ethernet с использованием мостов вместо повторителей, может дать вам увеличение пропускной способности для пользователей, так как мост перемещает данные к меньшему числу пользователей на сегмент. Но, опять-таки, вы можете столкнуться с испытанной проблемой увеличения времени ожидания до 20-30%, вызванной обработкой и фильтрацией пакетов. К тому же, поскольку мосты отправляют широковещательные пакеты во все подключенные сегменты, передача таких

пакетов в сети может иметь результатом широковещательные штормы. Широковещательный шторм - это событие в сетевом сегменте при котором широковещательный пакет посылается в бесконечном цикле пока не переполняет сегмент.

Концентраторы. Хабы соединяют сочленения всех компьютеров локальной сети в одно устройство или концентратор. Концентраторы можно считать многопортовыми повторителями. Персональные компьютеры могут быть связаны с использованием коаксиального кабеля, или кабеля витых пар, или даже радио-частоты. Когда один компьютер передает сигнал в сетевую среду, сигнал ретранслируется во все сегменты, которые подключены к концентратору. Рисунок 3.9 демонстрирует типичный концентратор, работающий в локальной сети.

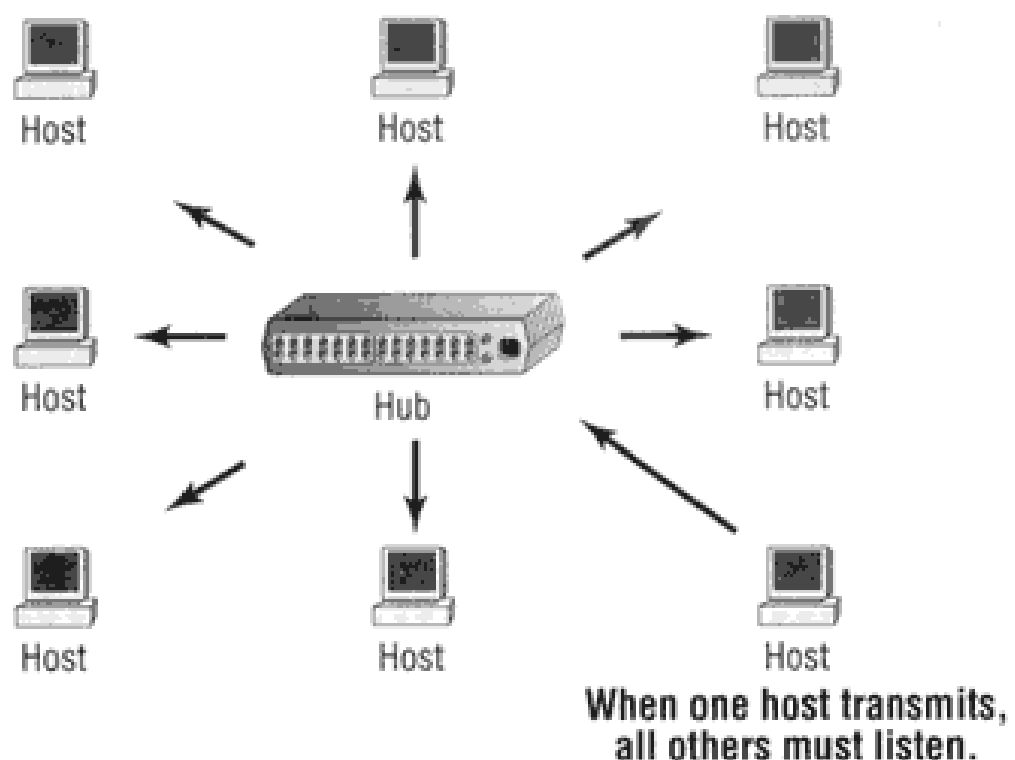


Рисунок 3.9 – Концентратор в локальной сети

Коммутаторы. Как и концентратор, коммутаторы могут работать в полном двойном режиме. Это означает, что оба компьютера и коммутаторы могут одновременно передавать и принимать данные. Коммутатор - это только определенный порт, где получатель отправляет их на MAC-адрес, когда есть разница между коммутатором и центром, когда компьютерный центр отправляет цифровые сигналы, а затем тот факт, что все порты, подключенные к концентратору, соединяются. Представьте, что каждый порт коммутатора является очень быстрым портом с несколькими портами.

Маршрутизаторы. Эти устройства являются шагом вперед от мостов. Мосты фильтруют по MAC-адресам, а маршрутизаторы могут фильтровать как аппаратные, так и сетевые (IP) адреса. Когда мост препроводжает пакет, он отправляет его всю книгу для конфигурации. Маршрутизаторы экономично предотвращают ненужный сетевой трафик от посылки через сетевые сегменты посредством открытия пакета данных и прочтения сетевого адреса, прежде чем препроводить этот пакет адресату. Рисунок 3.10 показывает локальную сеть, сегментированную с помощью маршрутизатора.

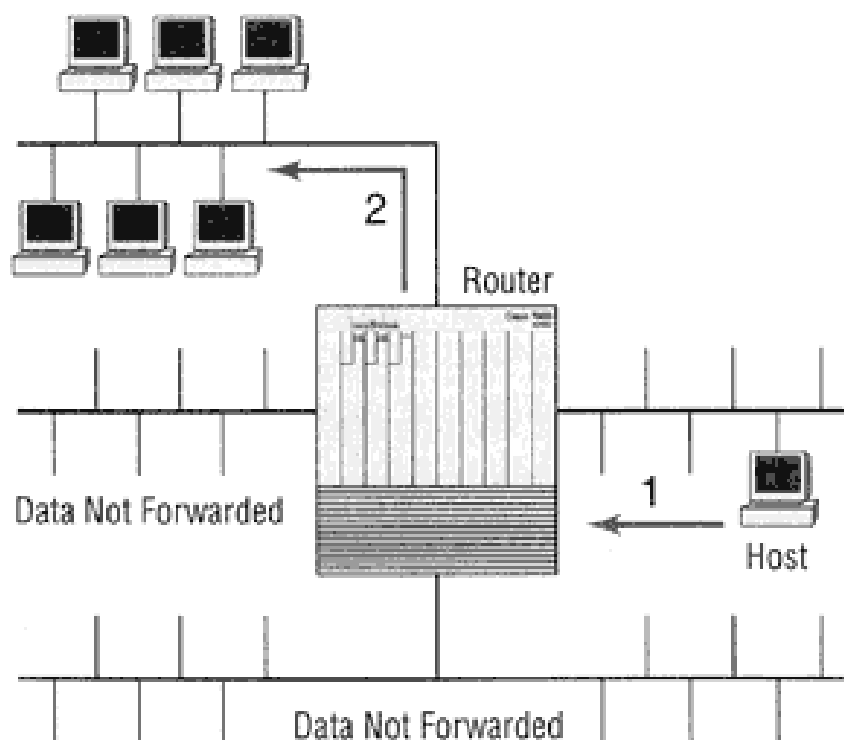


Рисунок 3.10 – Маршрутизатор в локальной сети

Существует множество библиотек для работы с TCP-соединением и передачей данных по wi-fi. Одной из них является библиотека «ESP8266».

Эта библиотека позволяет Arduino получить доступ к карте расширения WiFi в Интернете. Это позволяет Arduino действовать как клиент, который подключается к серверу и подключается к удаленному серверу. Библиотека поддерживает методы личного шифрования WEP и WPA2 (WPA2 Enterprise не поддерживается). Вы должны принять, что карта расширения учетной записи может быть подключена только к общедоступной сети SSID.

Коммуникационная плата Arduino и WiFi-расширения, цифровой выход, выполняемый на шинах SPI 11, 12 и 13 между Arduino, Uno и 50, 51, 52 сочетается в себе - Arduino Mega. S-продукт на обеих S-платах использует 10-выводное оборудование S (53) Arduino Mega не используется, но нормальный интерфейс SPI должен быть настроен как продукт для него.

Digital Pin 7 Arduino используется для запуска карты расширения Wi-Fi при подключении, поэтому ее нельзя использовать для других целей.

Работа библиотеки аналогична функциям библиотеки Ethernet и часто является одним и тем же составом.

Рассмотрим схематичное подключение обычного wi-fi модуля к Arduino.

На рисунке 3.11 изображено схематичное подключение wi-fi модуля к Arduino uno.

На рисунке сделать уклон в сторону контактов RX и TX.

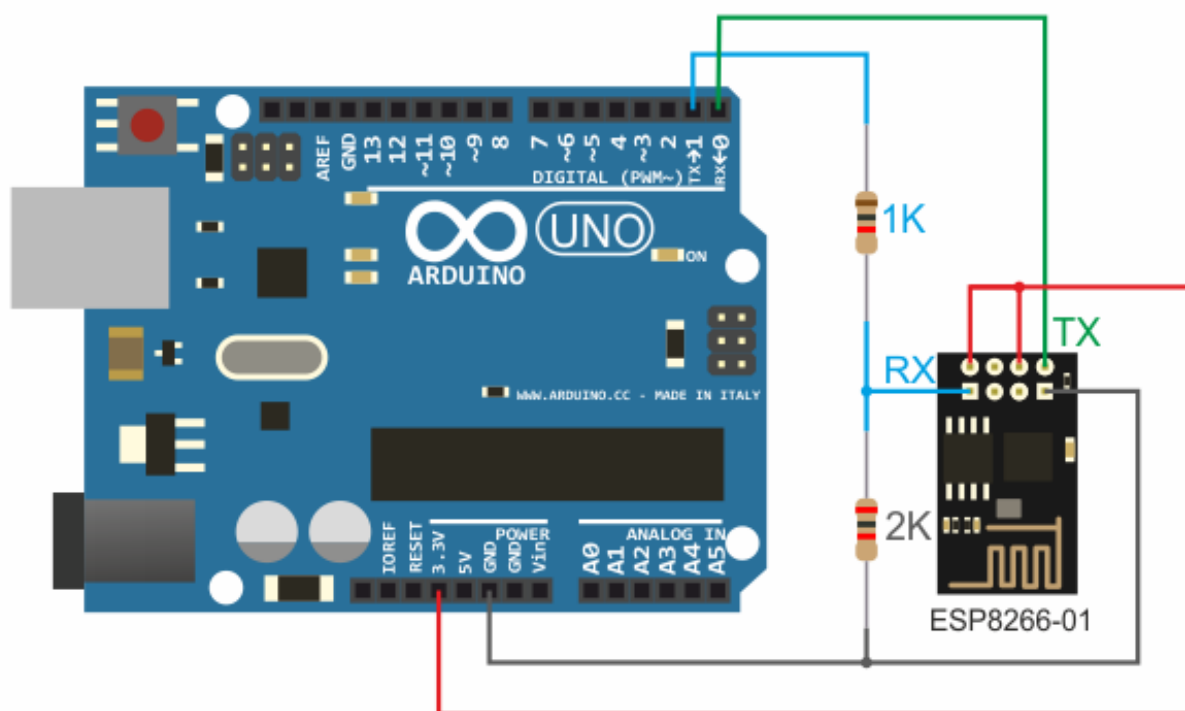


Рисунок 3.11 – Схематичное подключение wi-fi модуля к Arduino

### 3.1.3 Архитектура аппаратной части

Архитектура аппаратной части представлена на рисунке 3.12

Клиентом в данной архитектуре выступает пользователь, который подключается к главному контроллеру через USB-кабель. Далее на компьютере он выбирает конкретный порт, к которому присоединено устройство. После того, как пользователь подключился – он может конфигурировать устройство. Конфигурация происходит через терминал или другие программные средства.

Остальные модули и их взаимодействие с главным контроллером были рассмотрены выше.

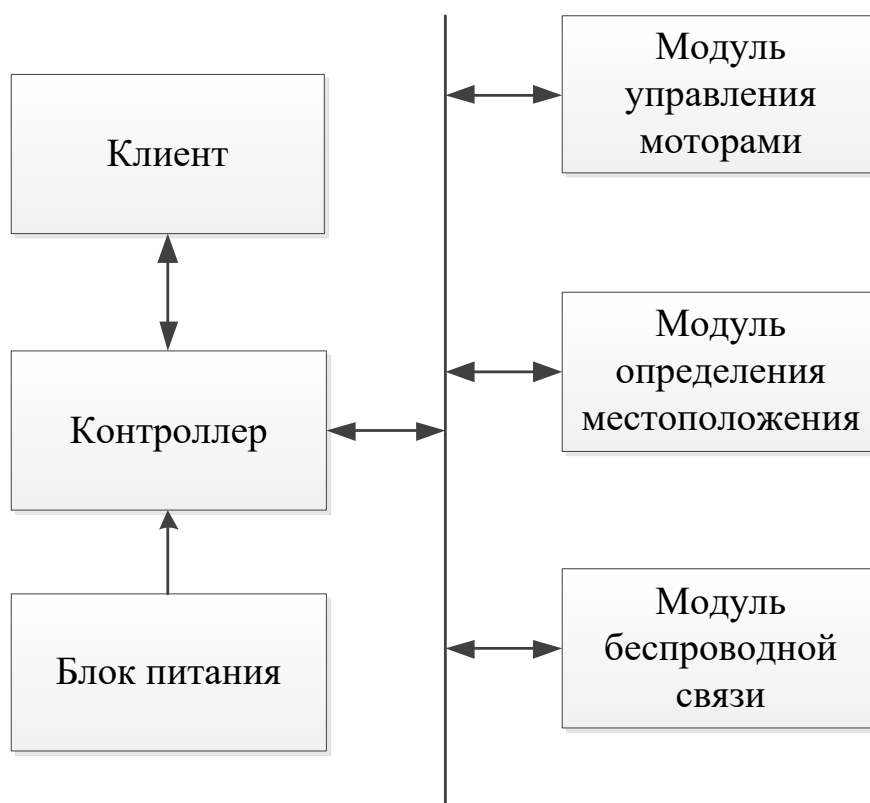


Рисунок 3.12 – Архитектура аппаратной части

### 3.2 Программная часть

Программная часть представляет собой остальной набор модулей, которые были представлены в структурной схеме, а именно:

- модуль обмена данными. Используется для обмена и обработки данных между программной и аппаратной частью.
- модуль управления. Представляет собой модуль, который отвечает за ввод команд на стороне клиента.
- модуль отображения данных. Используется для вывода получаемой информации от центрального контроллера.

Рассмотрим модуль обмена данными.

Использование данного модуля является ключевым звеном в объединении в одно целое аппаратной и программной части. Также является связующим модулем в программной части.

Модуль обмена данными выполняет роль распределения входящей информации от модуля управления и модуля беспроводной связи, а также отправления данных на модуль отображения, если это необходимо.

Модуль обмена данными для корректного обмена с модулем беспроводной связи использует технологию взаимодействия, именуемая как сокеты.

На рисунке 3.13 изображено классическое взаимодействие с помощью сокета клиента и сервера.



Сокеты позволяют подключаться к любой открытой wi-fi точке, тем самым создавая ТСР-окно.

Подробнее о влиянии сокетов в разработке проекта рассмотрим в пункте 3.2.2

В данном дипломном проекте можно рассматривать аппаратную часть, как серверную часть, а программное средство – клиентская часть.

Рассмотрим следующий модуль – модуль управления.

Данный модуль представляет собой программный модуль, написанный как и вся клиентская часть, на языке Python. Использование данного блока позволяет отправлять команды модулю обработки данных, который в свою очередь определяет к чему относится данная команда – к модулю отображения или к аппаратной части.

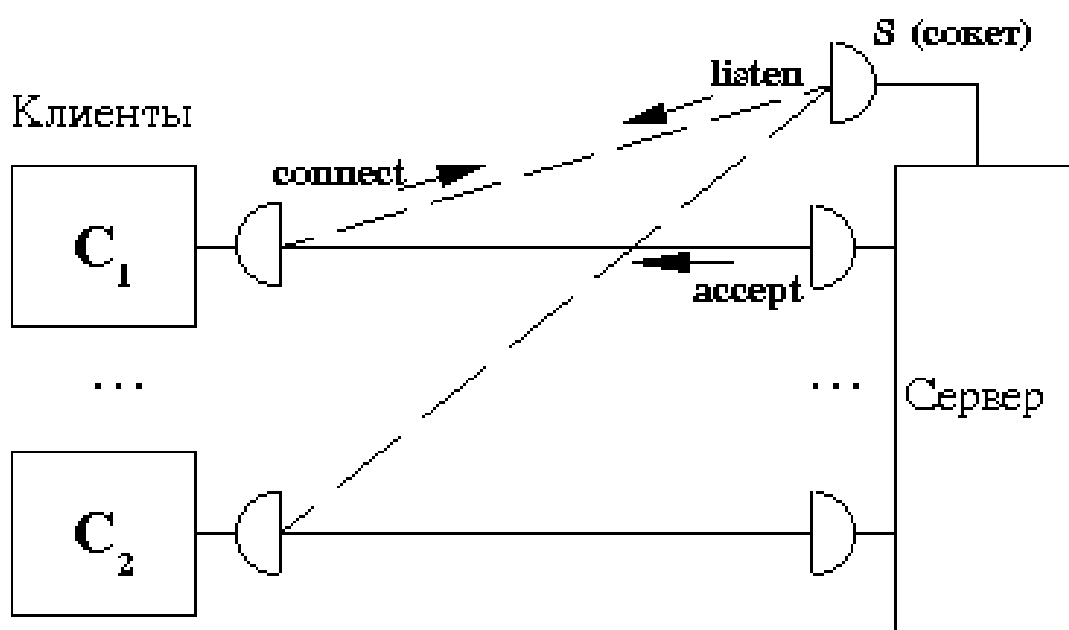


Рисунок 3.13 – Взаимодействие клиента и сервера через сокет

Последний модуль – модуль отображения.

Использование данного модуля необходимо для отображения данных, чтобы пользователь понимал, что происходит и что команда, введенная им, повлияла на процесс работы устройства.

Теперь рассмотрим более детально функционирование программной части, рассмотрим значение методов и свойств исполняемых файлов.

- *Main.py* представляет собой главный исполняемый файл на клиентской части. К нему подключаются основные модули, а также дополнительные библиотеки.

- *Connection.py* представляет собой исполняемый файл, который отвечает за подключение клиента к модулю беспроводной связи. Также выполняет проверку IP-адреса и наличие сети.

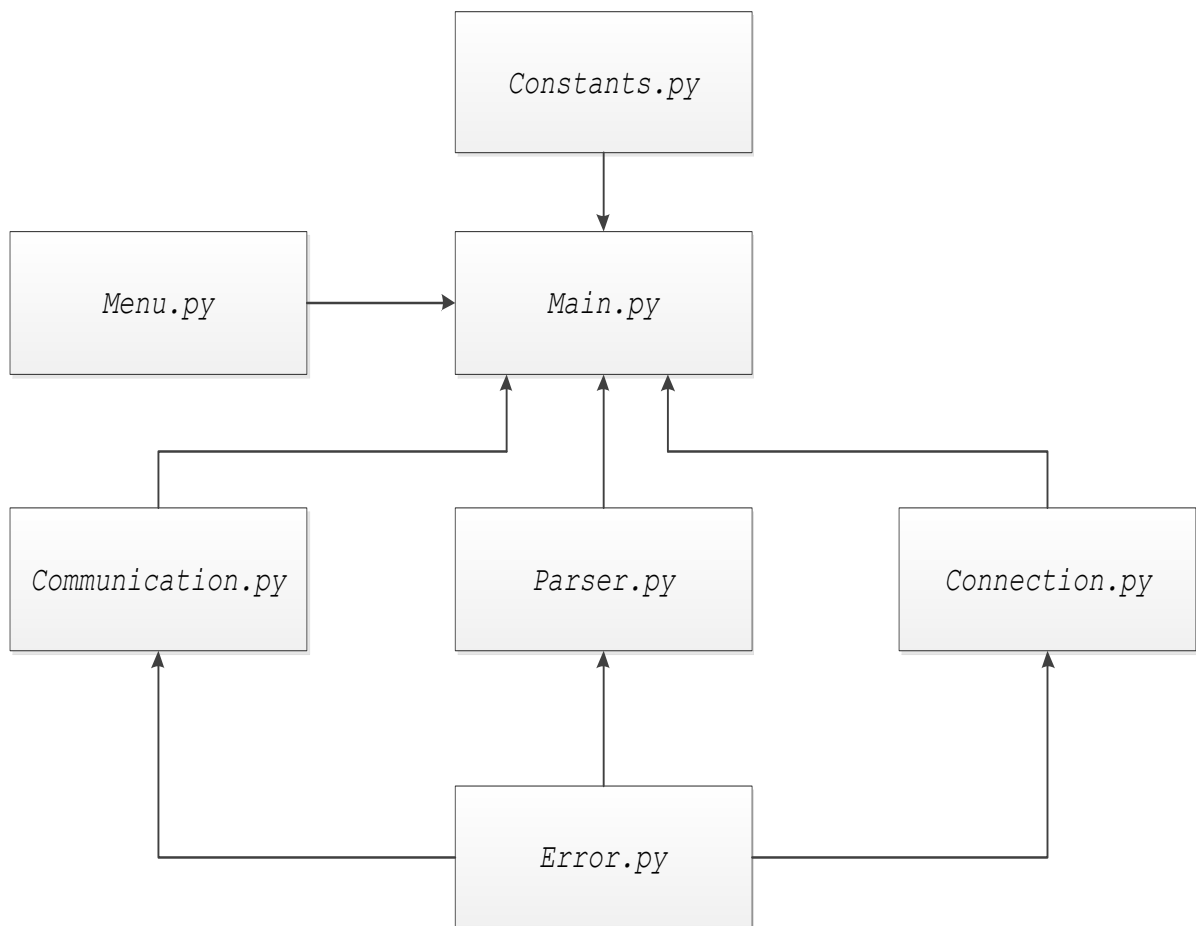


Рисунок 3.14 – Связь между исполняемыми файлами

– *Communication.py* представляет собой исполняемый файл, в котором происходит общение между аппаратной и клиентской частью. Выполняет проверку валидности отправляемых данных и дополнительную конвертацию в пригодный для чтения вид.

– *Error.py* представляет собой модуль обработки ошибок, которые могут возникать в ходе выполнения программы.

– *Menu.py* представляет модуль, который отвечает за представление дополнительной графической информации и за оповещение клиента о предоставляемых ему возможностях в ходе выполнения программы.

– *Executor.py* представляет собой модуль, который выполняет команды, отправленные клиентом.

– *Parser.py* представляет собой модуль конвертации приходящих и исходящих команд.

– *Constants.py* представляет собой модуль с нужными константами для работы клиентской программы.

Используя такое разбиение на модули, обеспечивается ясная и последовательная картина исполнения команд, их обработки и отображения.

В случае непредвиденных ошибок мы используем дополнительно модуль обработки ошибок.

Связь между исполняемыми файлами программной части представлена на рисунке 3.14

### 3.2.1 Архитектура программной части

Архитектура программной части изображена на рисунке 3.15

Сервером является аппаратная часть. Выше было указано, что такое упрощение оправдывает себя.

Модуль обмена данными, как и говорилось выше, является связующим звеном в построении аппаратно-программной архитектуры.

Клиентом является пользователь, который запустил главный исполняемый файл, отвечающий за подключение к IP-адресу, на который настроен wi-fi модуль.

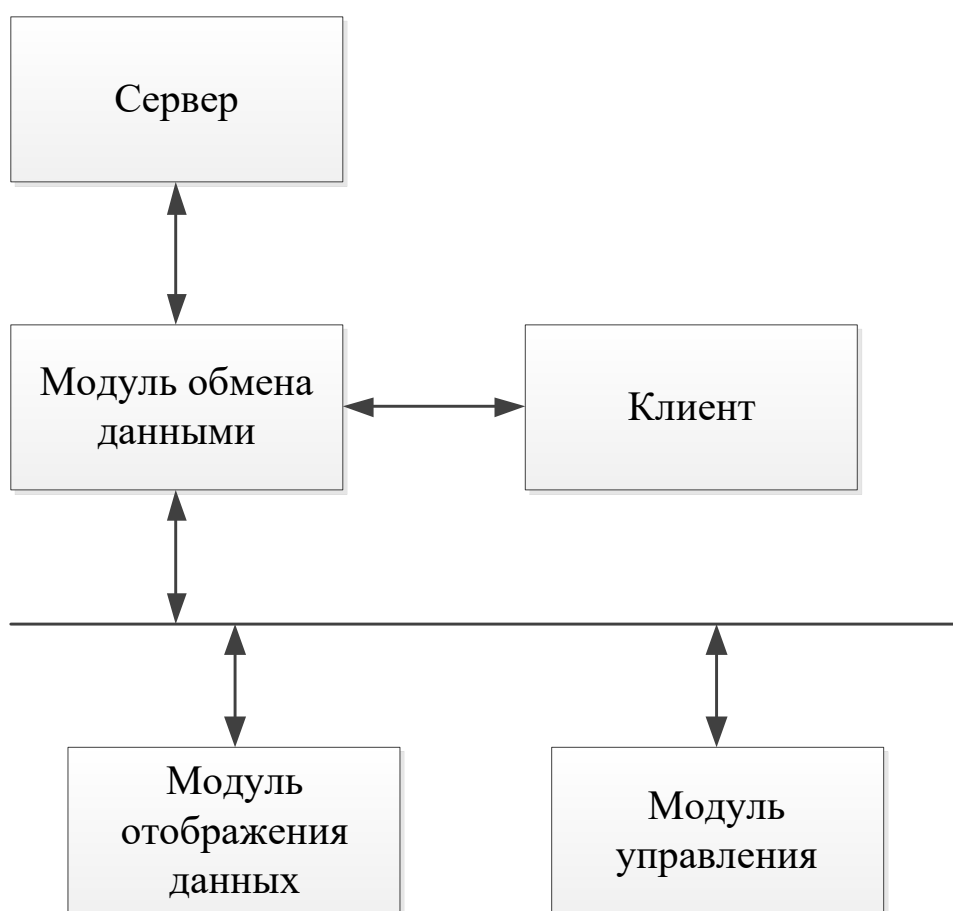


Рисунок 3.15 – Архитектура программной части

### 3.2.2 Использование сокетов и TCP-соединения

Сокет — программный интерфейс для обеспечения обмена данными между процессами. Процессы при таком обмене могут выполняться как на одной компьютере, так и на различных, связанных между собой сетью.

Также сокет это абстрактный объект, представляющий конечную точку соединения.

Существуют клиентские и серверные сокеты.

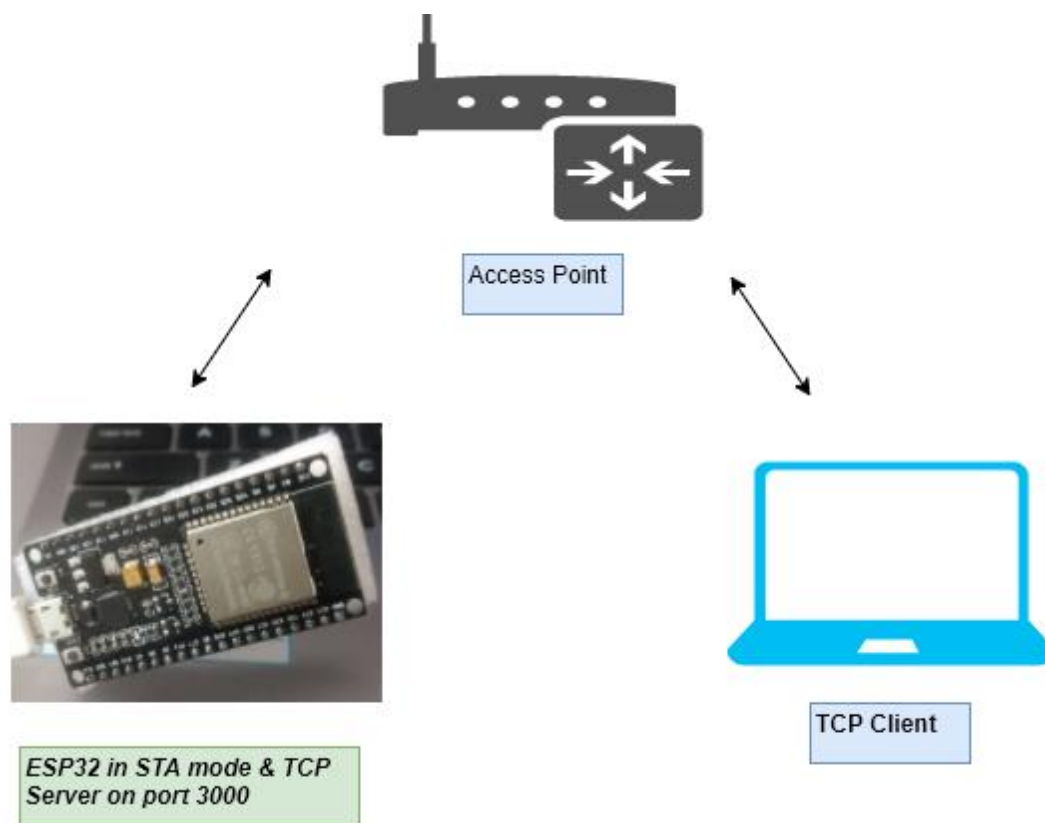


Рисунок 3.16 – Цикл обмена данными между пользователем и wi-fi модулем

Клиентские сокеты можно сравнить с конечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (например, браузер) использует только клиентские сокеты, а серверное (например, веб-сервер, которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

Как известно, для взаимодействия между машинами с помощью стека протоколов TCP/IP используются адреса и порты. Первое на текущий момент представляет собой 32-битный адрес (для протокола IPv4, 128-битный для IPv6), наиболее часто его представляют в символьной форме `mmm.nnn.ppp.qqq` (адрес, разбитый на четыре поля, разделённых точками, по одному байту в поле). Номер порта в диапазоне от 0 до 65535 (для протокола TCP).

Эта пара и есть сокет («гнездо», соответствующее адресу и порту).

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя.

Каждый процесс может создать «слушающий» сокет (серверный сокет) и привязать его к порту операционной системы.

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т. д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут общаться любые локальные процессы путём чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта.

Обычно клиент явно «подсоединяется» к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером.

В исполняемом файле *«Connection.py»* выполняется подключение к IP-адресу, на который настроен wi-fi модуль.

В данном модуле изначально происходит проверка на валидность введенного IP-адреса.

Регулярное выражение для проверки IP-адреса выглядит так:

```
'^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'
```

С помощью Python библиотеки *«re»* мы производим проверку.

Теперь подробнее рассмотрим установление TCP-соединения между клиентом и модулем беспроводной связи, а именно wi-fi модулем.

Для этого нужно обратить внимание на библиотеку *«socket»*, которая является основной в программной части.

Датчик беспроводной связи можно использовать в качестве как и клиента, так и сервера.

На рисунке 3.16 изображен цикл обмена данными между wi-fi модулем и конечным пользователем.

### 3.3 Архитектура проекта

На рисунке 3.17 представлена архитектура данного проекта, состоящего из аппаратной и программной части.

Как можно заметить Клиент является главным звеном.

Клиент может управлять как и аппаратной частью через USB-соединение на любой ЭВМ, а также с помощью программного средства подключаться к wi-fi модулю и выполнять дополнительные команды.

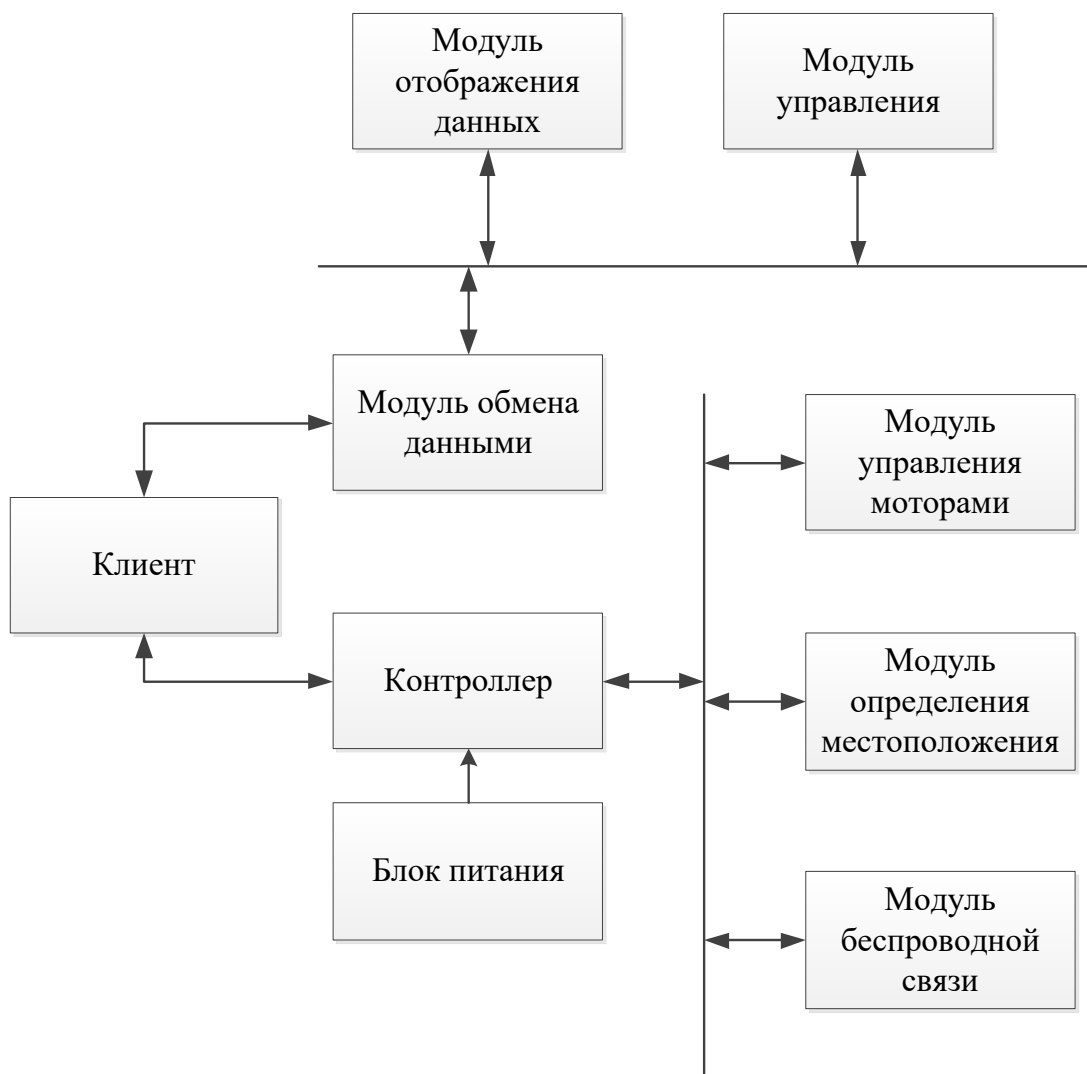


Рисунок 3.17 – Архитектура проекта

## 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

### 4.1 Аппаратная часть

Разработка аппаратной части началась с разработки основного корпуса. В соответствии требованиям к реализации аппаратной части следовало разработать подвижный корпус, который мог бы складываться на угол не меньший 30 градусов.

Разработка корпуса производилась в программе AutoCAD. Данная среда для разработки корпуса была выбрана, т.к. она предоставляет широкий спектр решения задач по визуализации, моделирования и скорости разработки устройства.

На рисунках 4.1 и 4.2 представлены макеты подвижного корпуса, выполненный в 3D виде:

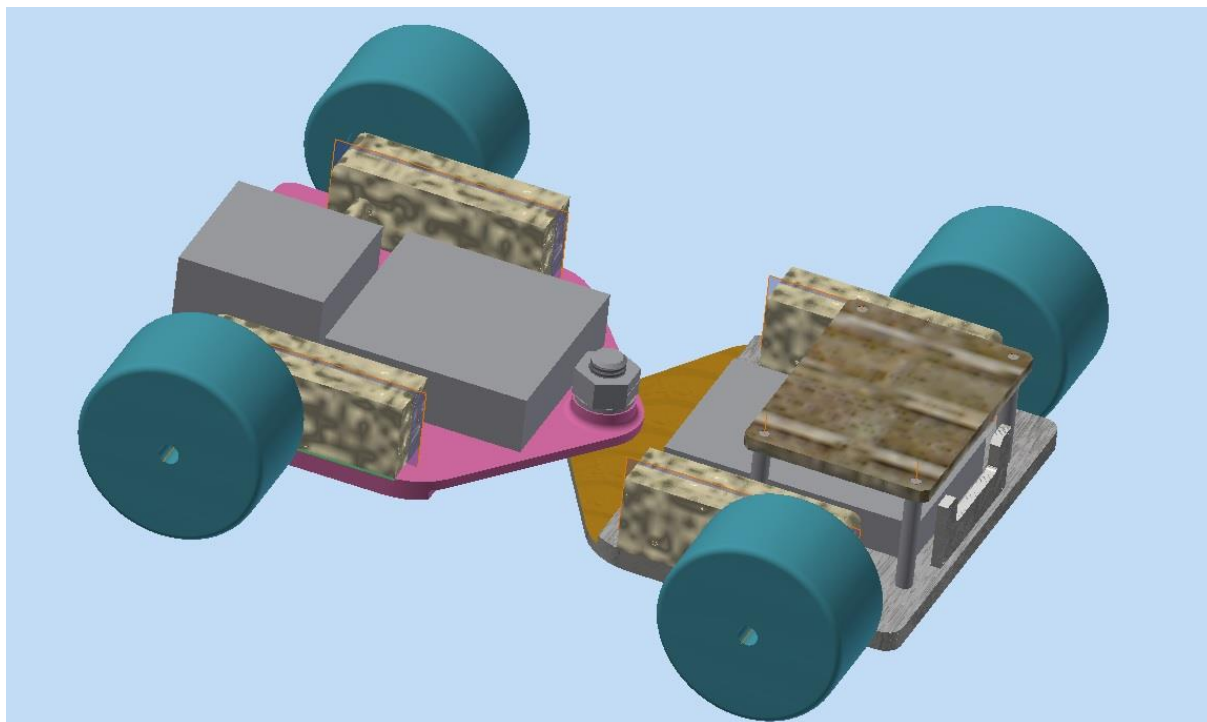


Рисунок 4.1 – Макет подвижного корпуса

На передней части корпуса располагается главный контроллер – Arduino Uno и датчик расстояния.

На задней части корпуса поместились wi-fi модуль и модуль управления моторами.

Данный корпус имеет дополнительно 2й этаж для размещения блока питания на передней части.

Для печати корпуса был выбран 3D принтер. Для распечатывания использовалась программа 3D Max.

Рассмотри подробнее расположение деталей, плат на данном корпусе.

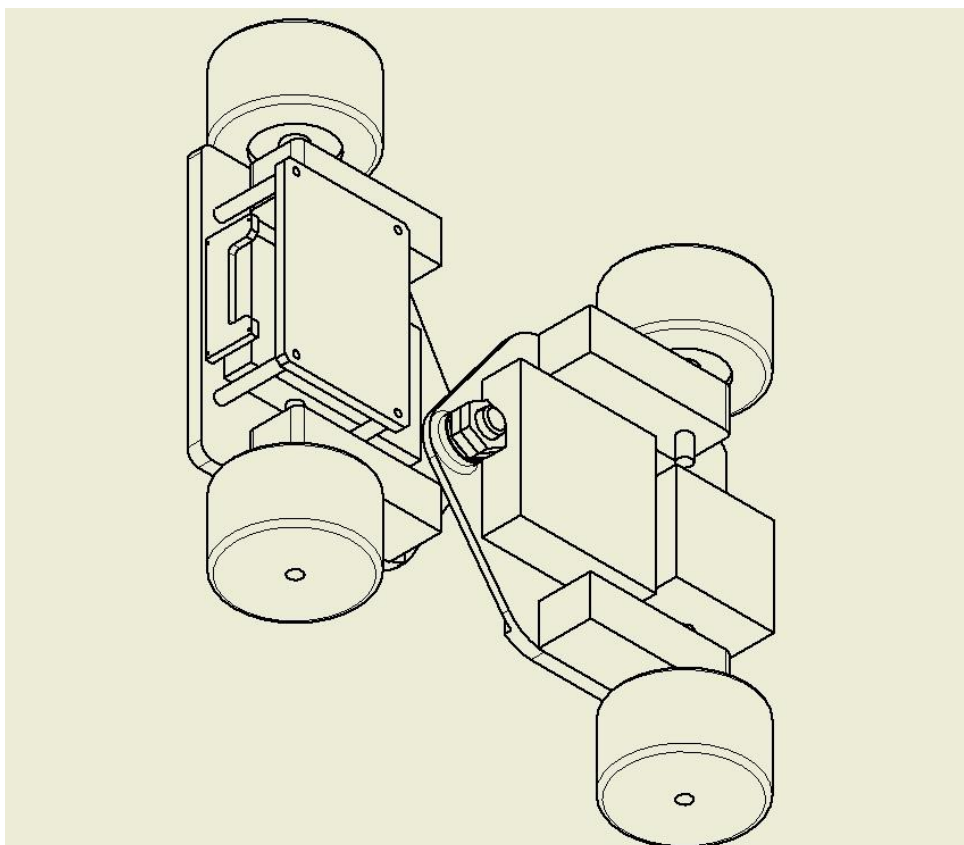


Рисунок 4.2 – Макет подвижного корпуса

На рисунке 4.3 указаны размеры корпуса и некоторых плат.

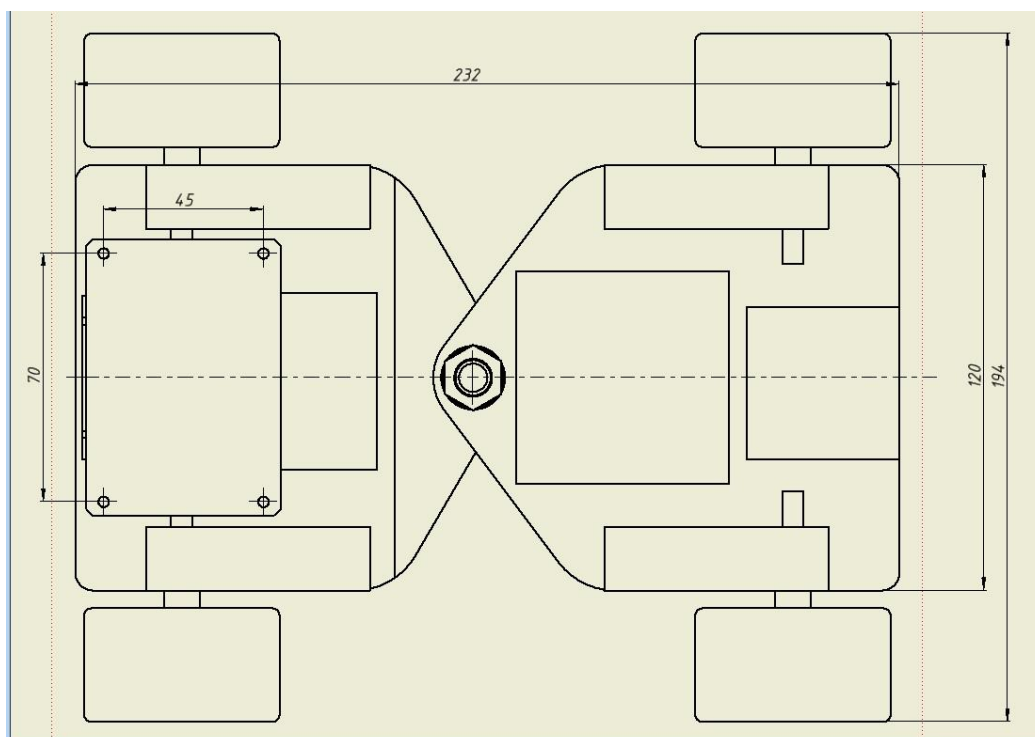


Рисунок 4.3 – Размеры деталей корпуса и плат



Теперь рассмотрим реализацию программы для главного контроллера и wi-fi модуля.

Прошивка для обоих устройств производилась с помощью USB-порта. Далее следовало выбрать нужный COM-порт и указать для чего производится компиляция: для Arduino или для wi-fi модуля.

На рисунке 4.4, 4.5 представлены скриншоты настройки для компиляции исходного кода для Arduino.

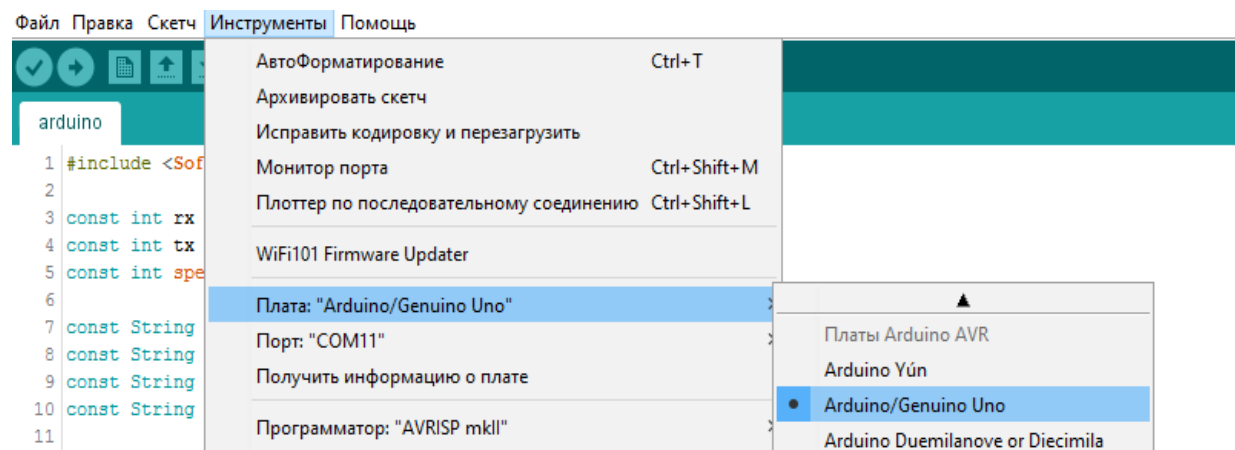


Рисунок 4.4 – Выбор платы Arduino Uno для компиляции

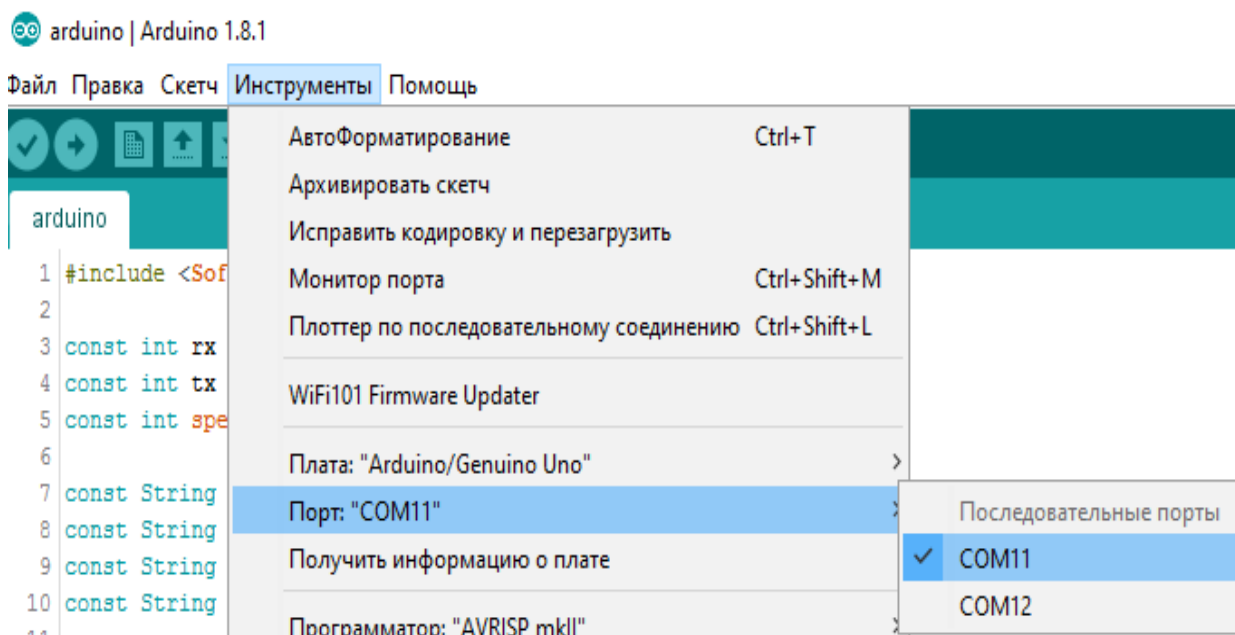


Рисунок 4.5 – Выбор порта, к которому подключено Arduino Uno

На рисунке 4.6, 4.7 представлены скриншоты настройки для компиляции исходного кода для wi-fi модуля.

Приступим к разбору исходного кода и методов для главного контроллера и wi-fi модуля.

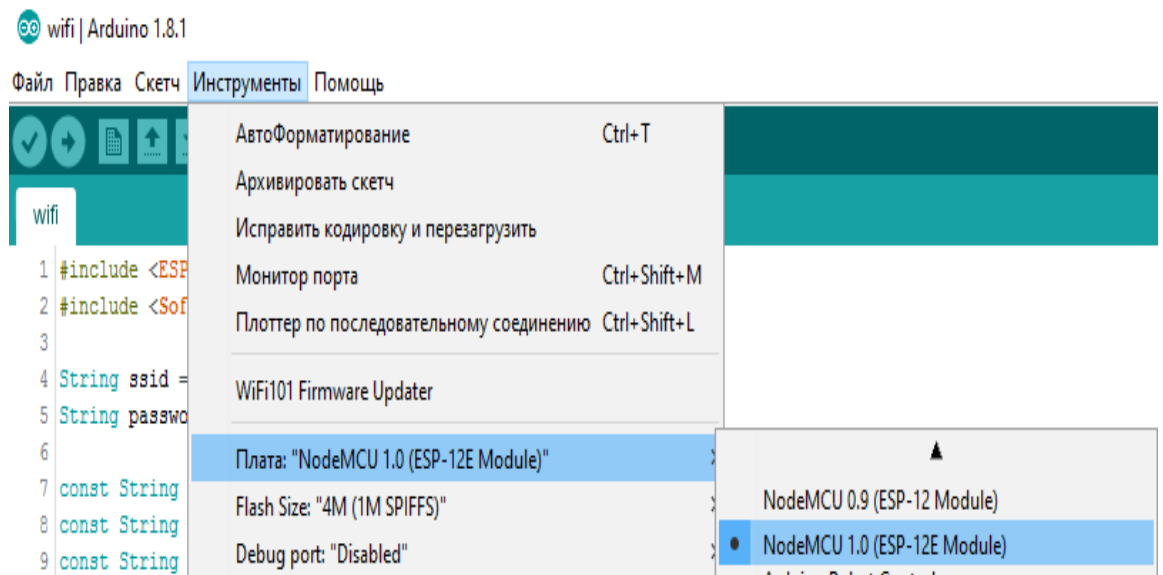


Рисунок 4.6 – Выбор платы NodeMCU для wi-fi модуля

Начинается все с того, что происходит инициализация плат, пинов, скорости серийных портов, моторов и т.д.

Инициализация для главного контроллера происходит в методе `setup`, реализация которого представлена ниже:

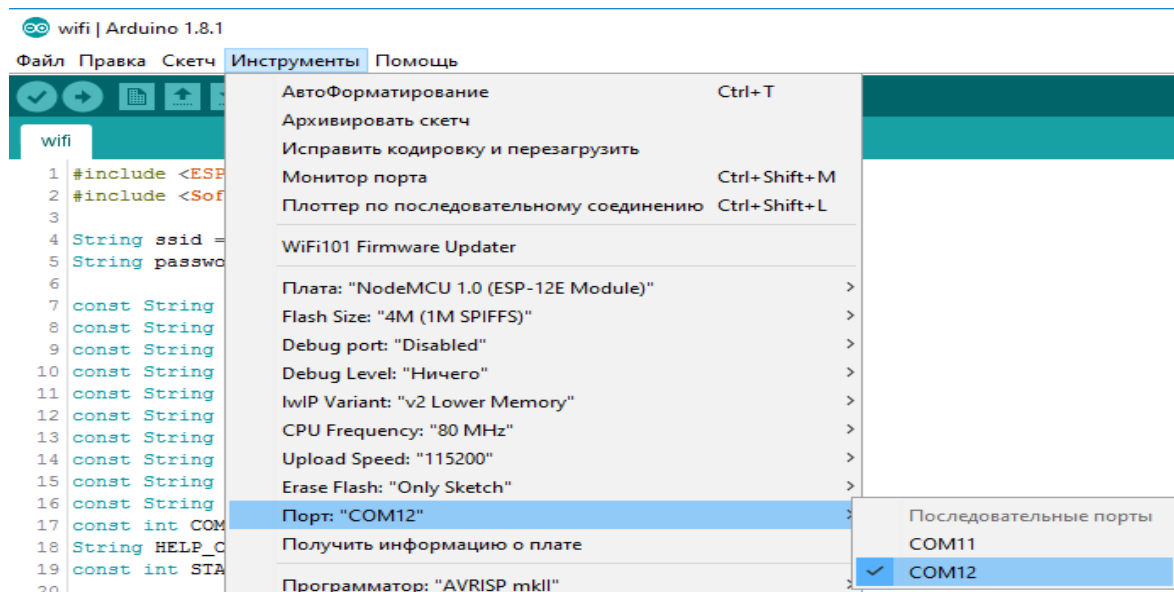


Рисунок 4.7 – Выбор порта, к которому подключен wi-fi модуль

```
void setup() {
    Serial.begin(speed);
    wifiModule->begin(speed);
    wifiModule->listen();
    pinMode (LEFT_DOWN, OUTPUT);
}
```

```

pinMode (LEFT_UP, OUTPUT);
pinMode (RIGHT_DOWN, OUTPUT);
pinMode (RIGHT_UP, OUTPUT);
},

```

где speed = 9600.

С помощью функции pinMode привязывается пин, к которому подключен мотор к OUTPUT.

Аналогично, для wi-fi модуля используется метод setup, в котором происходит инициализация связи с главным контроллером и wi-fi сервера.

```

SoftwareSerial* controller = new SoftwareSerial(D9, D10);
WiFiServer wifiServer(80);

void setup() {
    controller->begin(9600);
    controller->listen();
    wifiServer.begin();
}

```

Далее начинается работа бесконечного цикла, который реализован по умолчанию, в функции loop.

Для главного контроллера в данном методе происходит изначально считывание команд со стороны серийного порта. Т.е., пользователь с помощью USB-провода для Arduino выбирает порт и начинает его «прослушивать». Также он может вводить команды, которые главный контроллер изначально слушает.

Основными командами, которые пользователь может отправлять с помощью терминала на COM-порт главного контроллера являются:

```

const String GET_IP_ADDRESS = "GET_IP_ADDRESS";
const String SET_IP_ADDRESS = "SET_IP_ADDRESS";
const String SET_SSID = "SET_SSID";
const String GET_SSID = "GET_SSID";
const String GET_LOCAL_SSID = "GET_LOCAL_SSID";
const String SET_PASSWORD = "SET_PASSWORD";
const String GET_PASSWORD = "GET_PASSWORD";
const String CONNECT_TO_WIFI = "CONNECT_TO_WIFI";
const String WIFI_SESSION = "WIFI_SESSION";
const String HELP = "HELP";

```

Часть логики, отвечающая за прослушивание и выполнение команд в функции loop представлен ниже:

```

void loop() {
    String cmd = Serial.readString();
    cmd = cmd.substring(0, cmd.length() - 2);
}

```

```

    if (cmd == WIFI_SESSION) {
        wifiSession = true;
    }
    if (cmd.length() > 0) {
        sendToWifiModule(cmd);
        Serial.println("From serial: " + cmd);
    }
    if (wifiModule->available()) {
        String cmd = wifiModule->readString();
        Serial.println("From wifiModule: " + cmd);

        if (wifiSession) {
            Serial.println("wifi session enabled \n");
            startWifiSession();
        }
    }
}

```

Стоит заметить, что команды, которые пользователь посылает главному контроллеру напрямую, они все предназначены для настройки wi-fi модуля, т.к. wi-fi модуль настроить можно только через его взаимодействие с главным контроллером.

Обработка и отправка введенной команды происходит в функции `sendToWifiModule`:

```

void sendToWifiModule(String data) {
    wifiModule->print("*" + data);
}

```

В сторону wi-fi модуля отправляются данные и стартовый символ, который обозначает, что это команда и она пришла от главного контроллера.

В свою очередь ожидание и конвертация команды со стороны wi-fi модуля происходит в функции `readFromController`.

```

String readFromController() {
    String data = controller->readString();
    while (data[0] != '*') {
        data = controller->readString();
    }
    return data.substring(1, data.length());
}

```

В данном методе происходит постоянно считывание из сокета и как только приходит команда от главного контроллера данный метод возвращает конвертированную команду на обработку.

Для wi-fi модуля функция `loop` не представляется сложной. Основная задача wi-fi модуля – прием, конфигурация и отправка данных контроллеру для дальнейшей обработки и выполнения.

Ниже представлен метод `loop` для wi-fi модуля:

```

void loop() {
    String cmd = readFromController();
    handleCommand(cmd);

    delay(100);
}

```

Ключевой командой является WIFI\_SESSION, которая переводит wi-fi модуль в постоянное прослушивание команд со стороны клиента, который подключился к wi-fi модулю.

После того, как была введена эта команда, становится актуальным данный кусок функции loop:

```

if (wifiSession) {
    startWifiSession();
}

```

И, соответственно, сам метод startWifiSession:

```

void startWifiSession() {
    while(wifiSession) {
        if (wifiModule->available()) {
            String cmd = wifiModule->readString();
            Serial.println("From wifiModule: " + cmd);

            handleCmd(cmd);
        }
    }
}

```

Как можно заметить, цикл, который выполняется в данном методе будет выполняться до тех пор, пока пользователь не прервет wi-fi сессию.

В методе handleCmd происходит выполнение команд, который пользователь прислал через программу по wi-fi. И как раз в данном методе и происходит отлов, когда произойдет остановка wi-fi сессии.

Ниже представлен исходный код функции handleCmd:

```

void handleCmd(String cmd) {
    if (cmd == "HELP") {
        sendToWifiModule("- " + MOVEMENT + "\n- " + SENSORS);
    }
    else if (cmd == MOVEMENT) {
        movementMenu();
        sendToWifiModule(MOVEMENT + " exit");
    }
    else if (cmd == SENSORS) {
        sensorMenu();
        sendToWifiModule(SENSOR + " exit");
    }
}

```

```

    }
    else if (cmd == "stop") {
        wifiSession = false;
        sendToWifiModule("WiFi session stopped");
    }
    else {
        sendToWifiModule("Unknown command: " + cmd);
    }
}

```

Остановить wi-fi сессию можно с помощью команды «stop» или же, если произойдет обрыв соединения с устройством. Тогда wi-fi модулю отправится уведомление о том, что нужно прекратить слушать пользователя и вернуться в исходное состояние.

Со стороны wi-fi модуля переход и функционирование wi-fi сессии представлен отрывком из кода функции `handleCommand`, в которой отлавливается команда `WIFI_SESSION` и вызов функции `wifiSession`:

```

void handleCommand(String cmd) {
    if (cmd == WIFI_SESSION) {
        approve(cmd);
        notifyController(STATE_OK,
            "WIFI_SESSION",getIpAddress());
        delay(200);
        wifiSession();
    }
}

void wifiSession() {
    while(1) {
        WiFiClient client = wifiServer.available();
        String wifiCmd = "";
        if (client) {
            while (client.connected()) {
                while (client.available() > 0) {
                    char c = client.read();
                    if (c == '\n') {
                        sendToController(wifiCmd);
                        String data = readFromController();
                        client.write(data.c_str());

                        if (wifiCmd == "stop") {
                            client.stop();
                            return;
                        }
                    }
                    wifiCmd += c;
                }
            }
        }
        else {
            wifiCmd += c;
        }
    }
}

```

```

    }
    delay(10);
  }
}
client.stop();
}
}

```

В методе `wifiSession` используется бесконечный цикл, который прерывается той самой командой «stop» от пользователя.

Блок-схема данного метода представлена на рисунке 4.8

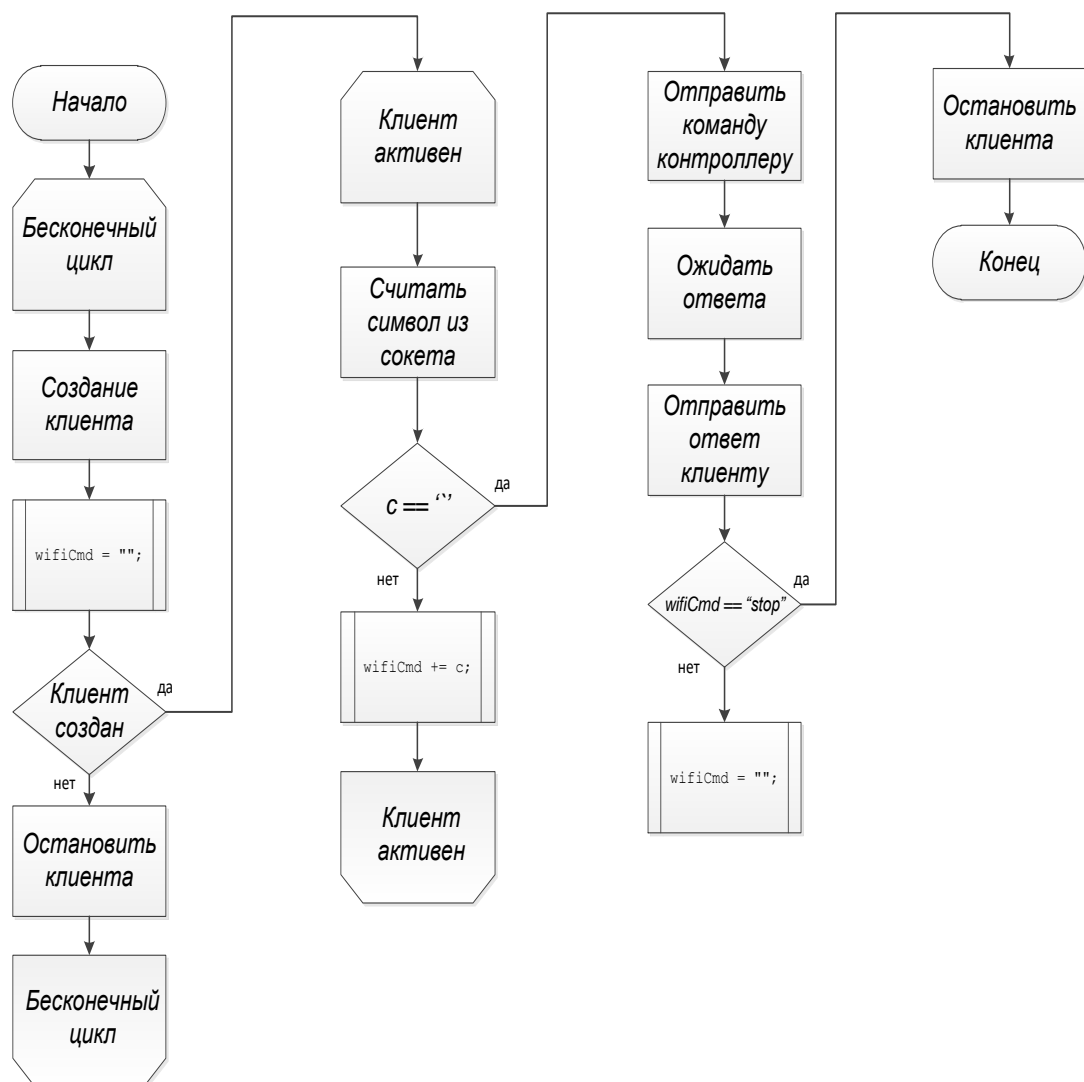


Рисунок 4.8 – Блок-схема работы функции `wifiSession`

## 4.2 Программная часть

Программное средство для управления подвижной системой-комплексом представляет из себя приложение, разработанное на языке

Python. Данный язык был выбран в качестве основного, так как представляется несложным в понимании и предоставляет большое количество возможностей для гибкой разработки приложений любых форматов. Данный язык программирования поддерживает несколько парадигм программирования, в том числе объектно-ориентированное, функциональное, императивное и аспектно-ориентированное.

Также одним из аспектов в выборе Python в качестве основного языка для программного средства послужила динамическая типизация, автоматическое управление памятью и удобный механизм обработки исключений.

Пространство имен и модульность повлияли на условие расширяемости программного средства. С этими двумя факторами Python отлично справляется. В ходе разработки проекта это очень помогло.

Начинается работа программы с того, что пользователь должен ввести IP-адрес, к которому должна подключиться программа, чтобы наладить контакт с устройством. Получение IP-адреса, к которому нужно подключаться, описано в предыдущем пункте.

Ниже приведен код для ввода и проверки IP-адреса:

```
REGULAR_IP = '^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'
regex = re.compile(REGULAR_IP)

while (is_valid_address == False):
    addr = input("\nInput host address: ")
    if (regex.match(addr)):
        is_valid_address = True
        HOST = addr
```

Как можно заметить `REGULAR_IP` является регулярным выражением для проверки валидности введенного IP-адреса.

Программа будет предлагать ввести IP-адрес до тех пор, пока он не пройдет основную валидацию.

Далее приведена настройка клиентского сокета:

```
wifi_module = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
wifi_module.connect((HOST, PORT))
```

После подключения к wi-fi модулю пользователю предлагается ввести команду. За данную функциональность отвечает данный блок кода:

```
while True:
    try:
        cmd = input("Type command: ")
        if validate_command(cmd):
            execute(cmd)
```



В данном блоке происходит сначала проверка валидности команды. Проверка представляет собой соответствие введенной команды на адекватность и наличие данной команды в списке команд.

После прохождения проверки введенная команда готова к исполнению. В методе `execute` происходит выполнение, а именно:

```
def execute(cmd):
    if is_menu_cmd(cmd):
        menu.run(cmd)
    else:
        communication.send_data(cmd)
        communication.print_data()
```

Сначала нужно определить является ли введенная команда командой для меню программы. В случае, если команда ориентирована на взаимодействие с wi-fi модулем, то она отправляется в блок `else`, в котором команда с помощью модуля `Communication`, который в свою очередь использует модуль `CommandParser`, где происходит конвертация запроса или ответа, и отправляется модулю обработки данных, а далее беспроводному модулю.

Далее ожидается ответ.

Ответ может быть одним из 3-х вариантов:

- Null – в случае отсутствия данных;
- Unknown command – в случае, когда была введена неизвестная команда для контроллера;
- Любые, адекватные данные, пришедшие со стороны контроллера.

Рассмотрим форматирование и конвертацию данных при получении и отправке. Для этого рассмотрим методы модуля `Communication`:

- `get_data`;
- `send_data`.

```
def get_data():
    return comandParser.unparse(wifi_module.recv(
        BUFFER_SIZE))
BUFFER_SIZE по умолчанию равен 1024.
```

В данном методе происходит прослушивание сокета и ожидание, когда будут посланы данные с другой стороны – со стороны wi-fi модуля.

```
def send_data(data):
    comandParser.parse(wifi_module.send(str(data+STOP_SYMBOL))
```

В данном методе происходит отправка данных wi-fi модулю.

Рассмотрим парсинг команд, который происходит в модуле `CommandParser`. С помощью методов `parse` и `unparse` происходит парсинг входящих и отправляющихся команд.

```
def parse(data):
    return data.encode('utf-8')

def unparse(data):
    return data.decode('utf-8')
```

Данные кодируются в `utf-8` – для более компактного хранения и передачи.

Как можно заметить к данным конкатенируется дополнительный стоп-символ. Это сделано для того, чтобы контроллер мог распознать закончена команда или нет. В качестве стоп-символа можно использовать любой символ, только следует заранее указать его в контроллере.

Использование обработчика ошибок и исключений является обязательным в разработке приложений с межсетевым взаимодействием.

В модуле `Error` предусмотрены обработки таких ошибок как:

- ошибка сети, сокета;
- прерывание клавиатуры.

```
except KeyboardInterrupt:
    print("KeyboardInterrupt was handled")
    wifi_module.close()
    os._exit(1)
except socket.error as e:
    print("WiFi session was stoped")
    wifi_module.close()
    os._exit(1)
```

Также дополнительные команды для клиентской части можно узнать с помощью вызова метода `help` у модуля `Menu`:

```
def help:
    for i in HELP_COMMANDS:
        print(i, ": ", HELP_COMMANDS[i])
```

`HELP_COMMANDS` – команды, доступные пользователю.

Описание данного объекта представлено ниже:

```
HELP_COMMANDS = {
    "TEST": "test connection between wi-fi module and
programm",
    "MOVEMENT": "request from controller a movement menu",
    "SENSOR": "request from controller a sensor menu",
    "HELP": "shows main menu",
```

```
    "EXIT": "exit from programm"  
}
```

Стоит заметить, что существует недостаток во взаимодействии пользователя и устройства. В связи с тем, что аппаратная часть запрограммирована на обработку запросов только от одного пользователя, то и программное средство может быть использовано только одним пользователем. Связано это со сложностью реализации многопоточности на устройствах типа Arduino, а также это повлекло бы за собой огромное количество утечек памяти и замедлению работоспособности и отзывчивости устройства, что не является приемлемым для данного дипломного проекта. Также это повлекло бы повышение сложности реализации аппаратной части, а соответственно и стоимости устройства в целом.

## 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя действия по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов.

Виды тестирования ПО:

1. Стресс-тестирование — оценивает надёжность и устойчивость системы в условиях превышения пределов нормального функционирования. Необходимо для «критически важного» ПО, однако также используется и для остального ПО. Обычно обнаруживает устойчивость, доступность и обработку исключений системой под большой нагрузкой, чем-то, что считается корректным поведением в нормальных условиях.

2. Регрессионное тестирование проводят по результатам исправления выявленных на этапе эксплуатации программного продукта ошибок и дефектов. Цель регрессионного тестирования: доказать, что программный продукт по-прежнему соответствует всем заявленным ранее требованиям.

3. Нагрузочное тестирование используется для выявления характеристик функционирования ПО при изменении нагрузки.

4. Тестирование частей ПО с целью проверки правильности реализации алгоритмов.

5. Функциональное тестирование подсистем и ПО в целом с целью проверки степени выполнения функциональных требований к ПО.

В данном дипломном проекте производится функциональное тестирование, которое в свою очередь, разделено на критическое и углубленное.

Критическое тестирование – это процесс поиска ошибок в программе при стандартной ее работе.

Углубленное тестирование – это процесс поиска ошибок в программе в нестандартных, непредвиденных ситуациях.

### 5.1 Аппаратная часть

Тестирование аппаратной части производилось на основе отправки команд главному контроллеру, который в свою очередь должен вернуть статус выполнения или предложить ввести дополнительную информацию. Программное обеспечение для тестирования аппаратной части было запущено на Windows 10 и на самом контроллере – Arduino Uno и wi-fi модуле. В таблице 5.1 приведены тесты и результаты тестов, проведенные над аппаратной частью.

Таблица 5.1 – Тестирование программ для аппаратной части

№	Название теста	Описание	Ожидаемые результаты	Тест пройден
1	Проверка связи с wi-fi модулем	Отправить тестовый пакет	test	Да
2	Обработка неизвестных команд	Отправка неизвестной команды	Unknown command и название команды	Да
3	Конфигурация wi-fi модуля	Получение SSID после прошивки	NULL	Да
4	Конфигурация wi-fi модуля	Установление SSID	SET_SSID_OK и название SSID	Да
5	Конфигурация wi-fi модуля	Установление пароля	SET_PASSWORD_OK и значение пароля	Да
6	Конфигурация wi-fi модуля	Подключение к сети	CONNECT_TO_WIFI_OK	Да
7	Конфигурация wi-fi модуля	Получение IP-адреса	IP-адрес сети	Да
8	Переход к wi-fi сессии	Установление wi-fi сессии	WIFI_SESSION_OK и IP-адрес подключения	Да

На рисунках ниже представлено подтверждение выполнения данных тестов.

На рисунке 5.1 представлен результат отправки тестовых данных.

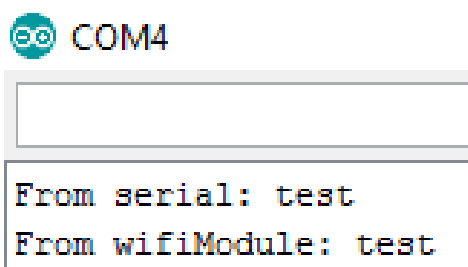
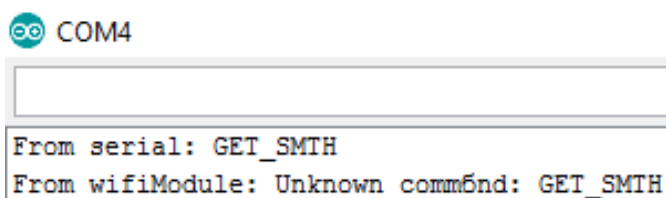


Рисунок 5.1 – Отправка тестового пакета

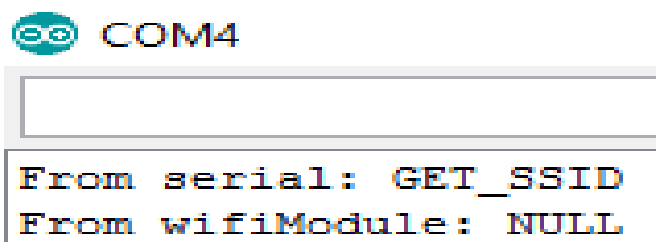
На рисунке 5.2 представлен результат отправки неизвестной команды.



```
COM4
From serial: GET_SMTH
From wifiModule: Unknown command: GET_SMTH
```

Рисунок 5.2 – Отправка неизвестной команды

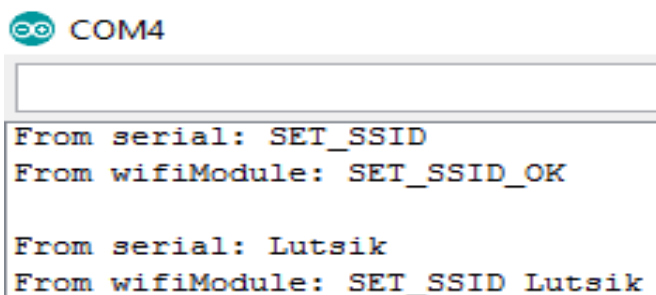
На рисунке 5.3 представлен результат получения SSID после прошивки.



```
COM4
From serial: GET_SSID
From wifiModule: NULL
```

Рисунок 5.3 - Получение SSID после прошивки

На рисунке 5.4 представлен результат установления SSID.

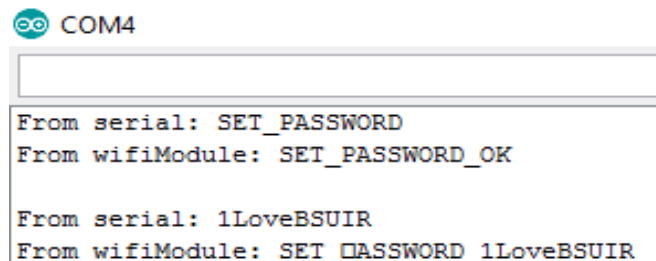


```
COM4
From serial: SET_SSID
From wifiModule: SET_SSID_OK

From serial: Lutsik
From wifiModule: SET_SSID Lutsik
```

Рисунок 5.4 – Установление SSID

На рисунке 5.5 представлен результат установления пароля.

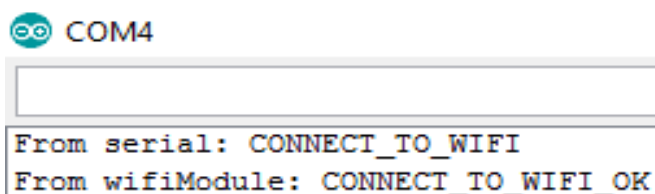


```
COM4
From serial: SET_PASSWORD
From wifiModule: SET_PASSWORD_OK

From serial: 1LoveBSUIR
From wifiModule: SET_PASSWORD 1LoveBSUIR
```

Рисунок 5.5 – Установление пароля

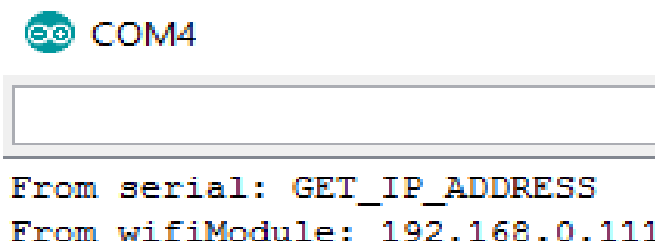
На рисунке 5.6 представлен результат подключения к сети.



```
COM4
From serial: CONNECT_TO_WIFI
From wifiModule: CONNECT_TO_WIFI_OK
```

Рисунок 5.6 – Подключение к сети

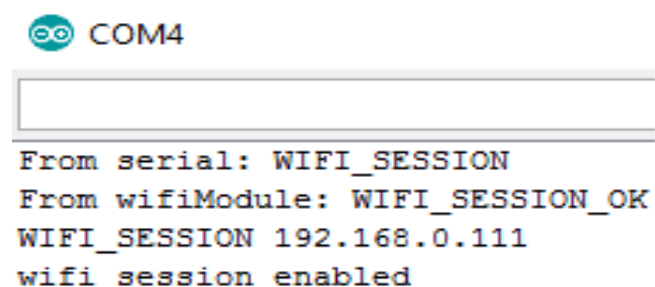
На рисунке 5.7 представлен результат получения IP-адреса.



```
COM4
From serial: GET_IP_ADDRESS
From wifiModule: 192.168.0.111
```

Рисунок 5.7 – Получение IP-адреса

На рисунке 5.8 представлен результат установления wi-fi сессии.



```
COM4
From serial: WIFI_SESSION
From wifiModule: WIFI_SESSION_OK
WIFI_SESSION 192.168.0.111
wifi session enabled
```

Рисунок 5.8 – Установление wi-fi сессии

## 5.2 Программная часть

Тестирование программной части происходило на основе подключения программы к wi-fi модулю, когда он находится в wi-fi сессии к IP-адресу, который был выведен в консоль при конфигурации.

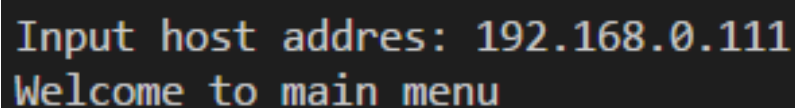
В таблице 5.2 приведены тесты и результаты тестов, проведенные над программной частью. Также эти тесты затрагивают и аппаратную часть, поэтому их можно считать дополнительной проверкой к тестированию аппаратной части.

На рисунках ниже представлено подтверждение выполнения данных тестов.

На рисунке 5.9 представлен результат проверки входа в программу.

Таблица 5.2 – Тестирование программы для программной части

№	Название теста	Описание	Ожидаемые результаты	Тест пройден
1	Проверка входа в программу	1. Запускаем программу 2. Вводим IP-адрес	Надпись приветствия	Да
2	Отображение доступных команд	1. Войти в главное меню 2. Ввести команду HELP	Список доступных команд	Да
3	Отображение команд Movement меню	1. Войти в главное меню 2. Ввести команду MOVEMENT	Список доступных команд	Да
4	Выполнение одной из команд движения	1. Войти в MOVEMENT меню 2. Ввести одну из предложенных команд	Изменение работы моторов и вывод команды на экран	Да
5	Проверка ввода невалидной команды	1. Войти в главное меню 2. Ввести любую команду не из списка возможных	Unknown command: название команды	Да
6	Проверка валидности IP-адреса	1. Запускаем программу 2. Вводим невалидный IP-адрес	Сообщение о невалидности введенного IP-адреса	Да
7	Обработка исключения прерывания клавиатуры	1. Запускаем программу 2. Вводим IP-адрес 3. Начинаем вводить команду 4. Нажимаем комбинацию CTRL+C	Сообщение об отлове исключительной ситуации	Да



```
Input host address: 192.168.0.111
Welcome to main menu
```

Рисунок 5.9 – Проверка входа в программу



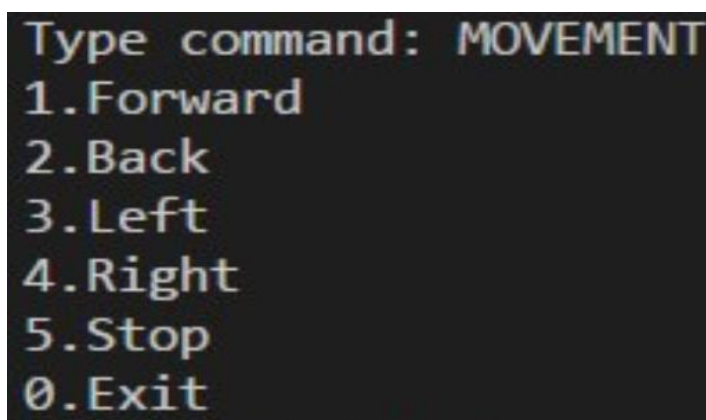
На рисунке 5.10 представлен результат отображения доступных команд.



```
Type command: HELP
- MOVEMENT
- SENSORS
```

Рисунок 5.10 - Отображение доступных команд

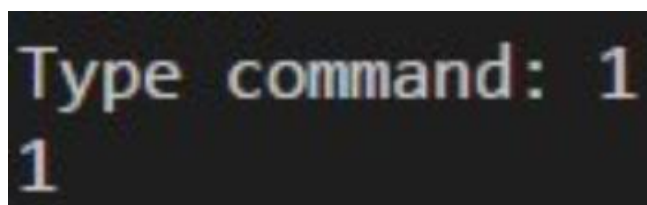
На рисунке 5.11 представлен результат отображения Movement меню.



```
Type command: MOVEMENT
1.Forward
2.Back
3.Left
4.Right
5.Stop
0.Exit
```

Рисунок 5.11 – Отображение команд Movement меню

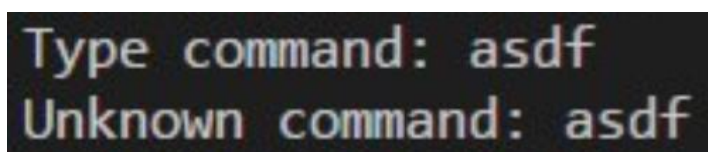
На рисунке 5.12 представлен результат выполнения одной из команд.



```
Type command: 1
1
```

Рисунок 5.12 – Выполнение одной из команд движения

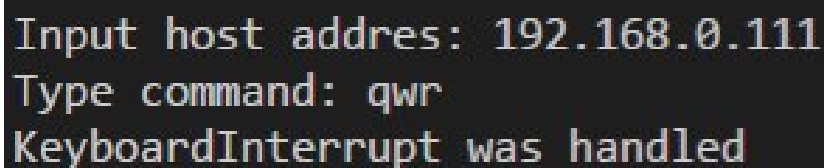
На рисунке 5.13 представлен результат проверки ввода невалидной команды.



```
Type command: asdf
Unknown command: asdf
```

Рисунок 5.13 – Проверка ввода невалидной команды

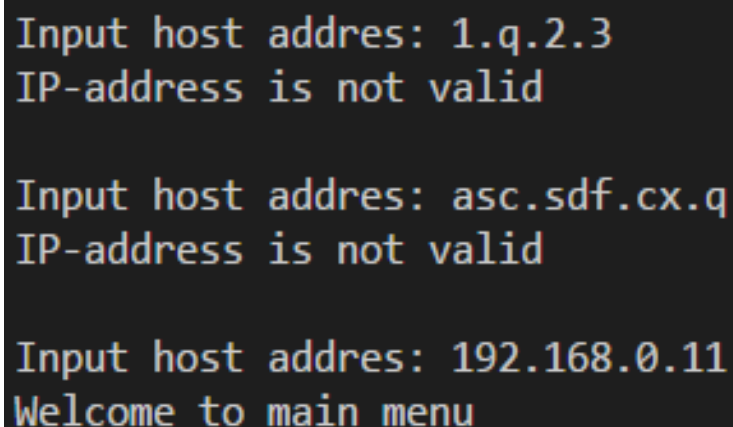
На рисунке 5.14 представлен результат обработки исключения прерывания клавиатуры.



```
Input host address: 192.168.0.111
Type command: qwr
KeyboardInterrupt was handled
```

Рисунок 5.14 – Обработка исключения прерывания клавиатуры

На рисунке 5.15 представлен результат проверки валидности IP-адреса.



```
Input host address: 1.q.2.3
IP-address is not valid

Input host address: asc.sdf.cx.q
IP-address is not valid

Input host address: 192.168.0.11
Welcome to main menu
```

Рисунок 5.15 – Проверка валидности IP-адреса

В ходе тестирования программной и аппаратной частей было установлено, что все тест-условия были успешно выполнены, практически все возможные варианты взаимодействия пользователя и программы были смоделированы и описаны. Для тестирования программной части была использована консоль операционной системы Windows, где и происходило подключение и вывод информации с wi-fi модуля. Программная и аппаратная части справились с тестами на хорошем уровне, что говорит о высокой работоспособности устройства в целом.

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данный раздел является отправной точкой для конечных пользователей данного устройства. Пошаговое описание подключения, взаимодействия с конечным устройством и подробное обоснование использования программного средства позволит уменьшить время для ознакомления с платформой.

Ориентирование на пользователя является основополагающей частью в принципе в разработке и продаже устройства. Ключевыми факторами реализации и использования данного проекта пользователем являются:

- простота в понимании;
- экономия времени пользователя;
- скорость ответа;
- минимальный набор дополнительных компонентов;
- открытый исходный код;
- возможность обратной связи и технической поддержки.

Данный раздел разбит на 2 части: аппаратную и программную, в которых будет подробно описаны соответствующие компоненты.

Стоит отметить, что одним из списка преимуществ данного дипломного проекта является минимальный порог вхождения или ознакомления пользователя с платформой. Забота о времени и терпении пользователя являются ключевыми факторами в разработке данного руководства.

### 6.1 Аппаратная часть

Для начала работы с аппаратной частью необходимо подключить данную платформу к аккумуляторам. Бокс с аккумуляторами имеет специальный разъем для подключения к Arduino Uno – главному контроллеру устройства.

После того, как устройство было подключено нужно убедиться, что оно подключено к вашей локальной сети.

Если устройство ранее не использовалось, то с большой вероятностью придется настроить wi-fi точку доступа самостоятельно.

Конфигурирование wi-fi модуля происходит через главный контроллер и занимает всего 3 простых шага:

1. Подключить USB-кабель для прошивки Arduino Uno в разъем компьютера;
2. Определить к какому COM-порту подключено устройство. На рисунке 6.1 показано, где можно посмотреть список подключенных устройств к вашему персональному компьютеру;
3. С помощью программы для работы с COM-портами подсоединиться к выбранному COM-порту, к которому подключено устройство. В таблице 6.1 представлены способы подключения или программы, с помощью которых можно начать общение с COM-портом.

В дальнейших примерах будет использован способ подключения через Arduino IDE Serial Monitor.

Для получения списка команд для конфигурирования wi-fi модуля нужно ввести команду *HELP*.

В таблице 6.2 представлен список возможных команд для конфигурирования wi-fi модуля.

Таблица 6.1 – Способы взаимодействия с COM-портами

№	Способ взаимодействия
1	Программа PUTTY
2	Arduino IDE Serial Monitor
3	Программа Advanced Serial Port Monitor
4	Программа Serial and Net Tools

После подключения можно приступать к конфигурированию.

Таблица 6.2 – Список команд для конфигурирования wi-fi модуля

Команда	Назначение
1	2
GET_IP_ADDRESS	Команда для получения IP-адреса, к которому подключен wi-fi модуль
SET_SSID	Команда для установления SSID. SSID – имя wi-fi точки
GET_SSID	Команда для получения SSID

*Продолжение таблицы 6.2*

1	2
GET_LOCAL_SSID	Команда для получения SSID в случае, если wi-fi модуль автоматически подключился к wi-fi точке
SET_PASSWORD	Команда для установления пароля для подключения к wi-fi точке
GET_PASSWORD	Команда для получения пароля wi-fi точки
CONNECT_TO_WIFI	Команда, которая использует данные SSID и password, находит данную wi-fi точку по SSID и инициирует подключение
WIFI_SESSION	Команда, которая активирует wi-fi сессию. Wi-fi сессия используется для работы с wi-fi модулем, напрямую посылая команды ему с пользовательского компьютера с помощью сокетов

В случае, если устройство еще не было использовано, то следует ввести команду SET\_SSID, после нее ввести SSID желаемой сети. Далее следует ввести команду SET\_PASSWORD и соответственно пароль. После ввести команду CONNECT\_TO\_WIFI, с помощью которой происходит подключение к сети с настройками, которые ввел пользователь.

На рисунке 6.2 представлен ход выполнения подключения к новой сети.

Для работы с программной частью, которая будет рассмотрена в следующем пункте, следует активировать wi-fi сессию. Wi-fi сессия предназначена для беспроводного общения с программным средством и выполнение команд, входящих от него.

Данный подход был встроен для того, чтобы пользователь всегда знал, в каком режиме настройки или использования он находится. И, естественно, однонаправленная обработка данных является ключом для увеличения производительности и однонаправленности данных, что в свою очередь хорошо влияют на тестирование и, если пользователь владеет навыками разработки программного обеспечения и разбирается в написании программ для микроконтроллеров Arduino, то поправки в открытый исходный код и перепрошивка будут требовать минимум времени для осознания кода.

*Продолжение таблицы 6.2*

1	2
GET_LOCAL_SSID	Команда для получения SSID в случае, если wi-fi модуль автоматически подключился к wi-fi точке
SET_PASSWORD	Команда для установления пароля для подключения к wi-fi точке
GET_PASSWORD	Команда для получения пароля wi-fi точки
CONNECT_TO_WIFI	Команда, которая использует данные SSID и password, находит данную wi-fi точку по SSID и инициирует подключение
WIFI_SESSION	Команда, которая активирует wi-fi сессию. Wi-fi сессия используется для работы с wi-fi модулем, напрямую посылая команды ему с пользовательского компьютера с помощью сокетов

В случае, если устройство еще не было использовано, то следует ввести команду SET\_SSID, после нее ввести SSID желаемой сети. Далее следует ввести команду SET\_PASSWORD и соответственно пароль. После ввести команду CONNECT\_TO\_WIFI, с помощью которой происходит подключение к сети с настройками, которые ввел пользователь.

На рисунке 6.2 представлен ход выполнения подключения к новой сети.

Для работы с программной частью, которая будет рассмотрена в следующем пункте, следует активировать wi-fi сессию. Wi-fi сессия предназначена для беспроводного общения с программным средством и выполнение команд, входящих от него.

Данный подход был встроен для того, чтобы пользователь всегда знал, в каком режиме настройки или использования он находится. И, естественно, однонаправленная обработка данных является ключом для увеличения производительности и однонаправленности данных, что в свою очередь хорошо влияют на тестирование и, если пользователь владеет навыками разработки программного обеспечения и разбирается в написании программ для микроконтроллеров Arduino, то поправки в открытый исходный код и перепрошивка будут требовать минимум времени для осознания кода.

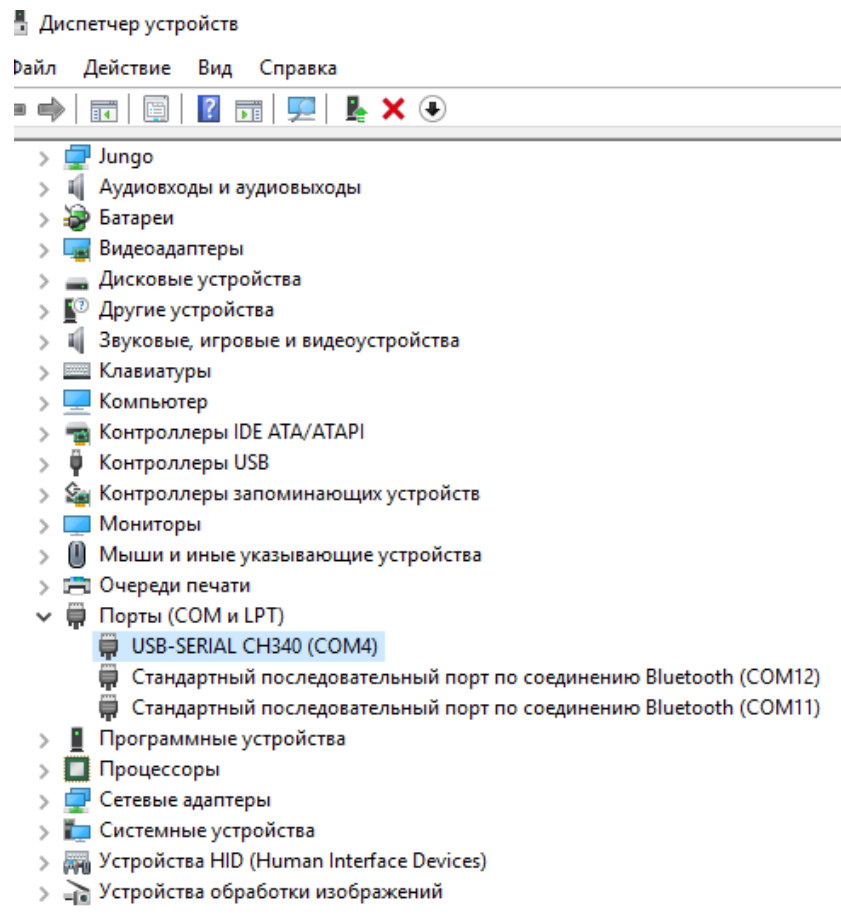


Рисунок 6.1 – Определения COM-порта, к которому подключена платформа

```

From serial: SET_SSID
From wifiModule: SET_SSID_OK

From serial: Lutsik
From wifiModule: SET_SSID Lutsik
From serial: SET_PASSWORD
From wifiModule: SET_PASSWORD_OK

From serial: 1LoveBSUIR
From wifiModule: SET_PASSWORD 1LoveBSUIR
From serial: CONNECT_TO_WIFI
From wifiModule: CONNECT_TO_WIFI_OK

```

Рисунок 6.2 – Подключение к wi-fi точке

## 6.2 Программная часть

С программной частью разобраться намного проще, чем с аппаратной. Во-первых, все настройки для плат и соединения с СОМ-портами были уже выполнены, и все, что остается сделать, это попросту запустить исполняемый файл на компьютере, ввести корректные данные и наслаждаться работой с устройством.

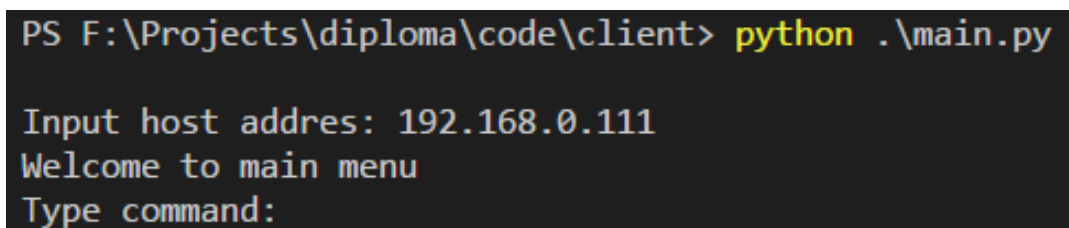
Итак, рассмотрим начало взаимодействия пользователя с программой, а именно – запуск исполняемого файла `Main.py`.

Но стоит помнить, что если устройство не было настроено на пользовательскую wi-fi сеть, то нужно вернуться к этому этапу и произвести настройку.

При старте программы пользователю предлагается ввести IP-адрес сети, к которой нужно подключиться. В случае некорректного IP-адреса пользователю будет доступна возможность сделать это еще раз.

Подключившись к wi-fi точке, программа поприветствует пользователя.

На рисунке 6.3 показано окно приветствия после успешного входа в систему.



```
PS F:\Projects\diploma\code\client> python .\main.py

Input host address: 192.168.0.111
Welcome to main menu
Type command:
```

Рисунок 6.3 – Запуск программы и окно приветствия

Общение с устройством происходит по мере отправки команд. Пользователю предлагается ввести команду «Type command». В случае, если пользователь не осведомлен с работой программы, ему нужно ввести команду «HELP».

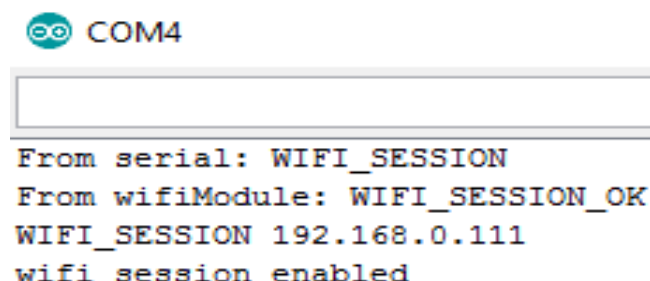
После ввода команды «HELP» будет предложен выбор действий-команд, которые пользователь может совершить. В аппаратной части реализовано управление и обмен данными с контроллером движения и датчиками. В программе за контроль за движением отвечает команда «MOVEMENT», за датчики – «SENSORS».

Рассмотрим такую ситуацию. Пользователь ведет общение с wi-fi модулем, но ему нужно изменить настройки wi-fi модуля на другие или же изменить точку беспроводной связи. В данном случае пользователь должен остановить работу wi-fi модуля, а именно wi-fi сессии и подключиться к главному контроллеру напрямую и изменить настройки, как было рассмотрено в пункте «Аппаратная часть».



Далее, чтобы войти обратно в wi-fi сессию для общения с программным средством, нужно ввести в терминале или программе, с которой работает главный контроллер, команду «WIFI\_SESSION».

Устройство уведомит пользователя об успешном старте и можно дальше продолжать управлять устройством удаленно. На рисунке 6.4 показан успешный вход в wi-fi сессию.

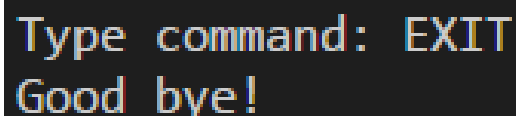


The screenshot shows a terminal window with a title bar that includes a blue icon with two white circles and the text "COM4". Below the title bar is a text area containing the following output:

```
From serial: WIFI_SESSION
From wifiModule: WIFI_SESSION_OK
WIFI_SESSION 192.168.0.111
wifi session enabled
```

Рисунок 6.4 – Успешная установка wifi сессии

Выход из программы осуществляется по вводу команды «EXIT». На рисунке 6.5 представлено успешное завершение программы.



The screenshot shows a terminal window with a black background and yellow text. The text displayed is:

```
Type command: EXIT
Good bye!
```

Рисунок 6.5 – Успешное завершение программы

Освоение данной программы не займет большого количества времени у пользователя, что является безусловно отличной новостью для него и соответственно для производителя данного устройства.

## 7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ АППАРАТНО-ПРОГРАММНОГО КОМПЛЕКСА МОБИЛЬНОГО РОБОТА

### 7.1 Характеристика аппаратно-программного комплекса

Целью данного проекта является разработка аппаратно-программного комплекса мобильного робота. Причиной разработки данного комплекса стала необходимость тестирования алгоритмов позиционирования робота через внешний блок управления.

Для удобства управления данным устройством разработано программное средство, с помощью которого можно управлять платформой и получать данные с датчиков.

Также данный проект предоставляет следующие возможности:

- подвижная, складывающаяся платформа;
- энергоэкономичные аппаратные модули;
- дистанционное управление;
- позиционирование платформы в пространстве;
- вычисление кратчайшего пути до точки назначения;
- удобное управление со стороны программного средства;
- понятное отображение выполнения запрошенных команд со стороны программного средства;
- обработка входящих запросов от программного средства.

Для оценки экономической эффективности разработанного аппаратно-программного проекта проводится расчет затрат на разработку системы, оценка прибыли от продажи одной такой системы и расчет показателей эффективности инвестиций в разработку аппаратно-программного комплекса.

### 7.2 Расчет стоимостной оценки затрат

Расчет затрат на заработную плату разработчиков проектной документации аппаратно-программного комплекса мобильного робота представлен в таблице 7.1.

Таблица 7.1 – Расчет основной заработной платы исполнителей

Исполнитель	Количество исполнителей, чел.	Оклад, руб.	Трудовое количество, мес.	Заработная плата по тарифу, руб.
1	2	3	4	5
1. Руководитель проекта	1	750	1	750

Продолжение таблицы 7.1

1	2	3	4	5
2. Программист 2к	1	730	1	730
Всего	2	-	-	1480
Премия, 30%	-	-	-	444
Всего основная заработная плата	-	-	-	1924

Расчет затрат на разработку проектной документации представлен в таблице 7.2

Таблица 7.2 – Расчет затрат на разработку проектной документации

Наименование статьи затрат	Расчет	Значение, руб.
1. Основная заработная плата разработчиков	табл. 7.1	1924
2 Дополнительная зарплата	$1924 \cdot 20/100$	384,8
3. Отчисления на социальные нужды	$(1924 + 384,8) \cdot 34,6/100$	798,84
4. Всего	-	3107,64

Расчет основной заработной платы на разработку программной части представлен в таблице 7.3.

Таблица 7.3 – Расчет основной заработной платы исполнителей программной части

Категория исполнителя	Эффективный фонд времени работы, дн.	Дневная тарифная ставка, руб.	Тарифная заработная плата, руб.
1	2	3	4

Продолжение таблицы 7.3

1	2	3	4
1. Программист 2к	10	37	370
Премия, 30%	-	-	111
Основная заработная плата	-	-	481

Расчет затрат на разработку программной части представлен в таблице 7.4.

Таблица 7.4 – Расчет затрат программной части

Наименование статьи затрат	Расчет	Значение, руб.
1. Основная заработная плата разработчиков	481	481
2. Дополнительная зарплата	$481 \cdot 20/100$	96,2
3. Отчисления на социальные нужды	$(481+96,2) \cdot 34,6 / 100$	199,71
Всего	-	776,91

Расчет затрат на оборудование для аппаратно-программного комплекса мобильного робота представлен в таблице 7.5

Таблица 7.5 – Расчет затрат на оборудование

Наименование покупных комплектующих изделий и аппаратных модулей	Количество на изделие, шт.	Цена за единицу руб.	Сумма, руб.
1	2	3	4
1. Arduino UNO	1	24	24

Продолжение таблицы 7.5

1	2	3	4
2. Мотор-драйвер L298N	1	35	35
3. Мотор-редуктор	4	16	64
4. Wi-Fi модуль NodeMcu Lua v.3	1	39	39
5. Аккумуляторы	3	12	36
6. Бокс для аккумуляторов	1	11	11
7. Провода	20	0,19	3,8
Всего			212,8
Всего с учетом транспортных расходов (20%)			255,36

Расчет затрат на материалы, необходимые для монтажа аппаратной части аппаратно-программного комплекса мобильного робота, представлен в табл. 7.6

Таблица 7.6 – Расчет затрат на материалы

Наименование материала	Единица измерения	Норма расхода на единицу продукции	Оптовая цена за единицу, руб.	Сумма, руб.
1	2	3	4	5
1. Припой	кг	0,15	36	5,4
2. Канифоль	кг	0,08	73,1	5,9

Продолжение таблицы 7.6

1	2	3	4	5
3. Флюс глицериновый	л	0,25	4,96	1,24
4. Лист ДВП	м <sup>2</sup>	0,7	5	3,5
5. Скотч двухсторонний	м	0,3	0,7	0,21
Всего				16,25
Всего с учетом транспортных расходов (20%)				19,5

Расчет заработной платы на монтаж аппаратно-программного комплекса мобильного робота представлен в табл. 7.7.

Таблица 7.7 – Расчет основной заработной платы на монтаж

Исполнитель	Количество исполнителей, чел.	Трудоемк ость, мес.	Оклад, руб.	Заработная плата по тарифу, руб.
1. Инженер по наладке и испытаниям	1	0,3	510	153
2. Техник по наладке и испытаниям	1	0,3	480	144
Всего	4	-	-	297
Премия, 20%	-	-	-	59,4
Основная заработная плата	-	-	-	356,4

Расчет затрат на монтаж аппаратно-программного комплекса мобильного робота представлен в таблице 7.8.

Таблица 7.8 – Расчет затрат на монтаж

Наименование статьи затрат	Расчет	Значение, руб.
1. Затраты на оборудование	См. табл. 7.5	255,36
2. Затраты на материалы	См. табл. 7.6	19,5
3. Основная заработная плата	См. табл. 7.7	356,4
4. Дополнительная зарплата	$356,4 \cdot 20/100$	71,28
5. Отчисления на социальные нужды	$(356,4 + 71,28) \cdot 34,6 / 100$	147,97
Всего	-	850,51

Капитальные вложения на разработку и изготовление аппаратно-программного комплекса мобильного робота представлены в табл. 7.9

Таблица 7.9 – Капитальные вложения на разработку и изготовление аппаратно-программного комплекса мобильного робота

Наименование статьи затрат	Расчет	Значение, руб.
1	2	3
1. Затраты на разработку проектной документации	См. табл. 7.2	3107,64
2. Затраты на программной части	См. табл. 7.4	776,91
3. Затраты на разработку аппаратной части	См. табл. 7.8	850,51
Всего	-	4735,07

Продолжение таблицы 7.9

1	2	3
4. Накладные расходы (50%)	$4735,07 \cdot 50/100$	2367,53
Всего затрат на разработку	$4735,07 + 2367,53$	7102,6
5. Прибыль (50 %)	$7102,6 \cdot 50/100$	3551,3
6. Отпускная цена	$7102,6 + 3551,3$	10653,91
7. Налог на добавленную стоимость (20 %)	$10653,91 \cdot 20/100$	2130,78
8. Отпускная цена с НДС	$10653,91 + 2130,78$	12784,69

### 7.3 Расчет экономической эффективности разработки аппаратно-программного комплекса

Экономическим эффектом у предприятия - разработчика системы является чистая прибыль, остающаяся в распоряжении организации, которая составит:

$$П = 3551,3 - \frac{3551,3 \cdot 18}{100} = 2912,07 \text{ руб.},$$

Рентабельность затрат на разработку данной системы для организации-разработчика составит:

$$Р = \frac{2912,07}{7102,6} \cdot 100\% = 41\%,$$

На основании полученных результатов экономического обоснования +комплекса мобильного робота являются экономически эффективными для предприятия-разработчика. После выполнения работ предприятие-разработчик получает чистую прибыль в размере 2912,07 руб., при этом рентабельность разработки составит 41%.



## ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования был разработан аппаратно-программный комплекс мобильного робота. Были рассмотрены существующие аналоги, микроконтроллеры, принцип работы устройства и его использование.

Для проектирования аппаратно-программного комплекса мобильного робота был выбран микроконтроллер Arduino Uno, исходя из его достоинств и так как он является оптимальным решением для проекта, целенаправленное программное обеспечение такое как Arduino IDE, дополнительные библиотеки для работы с компонентами Arduino Uno, а так же удобная и быстрая прошивка через USB-кабель.

В качестве среды разработки для программной части был выбран язык программирования Python, который обеспечил легкость и устойчивость к реализации данного программного средства. Пространство имен и модульность повлияли на условие расширяемости программного средства. С этими двумя факторами Python отлично справляется. В ходе разработки проекта это очень помогло.

В процессе работы с программной частью были изучены библиотеки для работы с сокетами, регулярными выражениями и системными взаимодействиями.

Для разработки корпуса устройства были использованы программы AutoCAD и 3D Max. Данные программы без проблем позволили реализовать задуманные идеи. Решение использовать 3D-принтер оправдало свои ожидания.

Для разработки программы для аппаратной части был использован плагин для среды Microsoft Visual Studio 2017 – Visualmicro.

Разработанная система предоставляет следующие возможности:

1. Подвижный, складывающийся корпус;
2. Дистанционное управление с помощью ПК;
3. Распознавание межсетевых команд;
4. Конфигурирование отдельных аппаратных модулей;
5. Программное средство для управления и отображения данных платформы;
6. Позиционирование устройства в пространстве.

В дальнейшем планируется добавить возможность управления с помощью мобильного приложения, добавить отдельный Bluetooth модуль. А также разработать новый дизайн для устройства, уменьшить количество соединений с главной платой.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] РобоРовер M1 Education - образовательный робот для студентов. [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://mrobot.by/blog/79-roborover-m1-education-obrazovatelnyj-robot-dlya-studentov-i-shkolnikov-dlya-studentov-i-shkolnikov/>
- [2] Робот «Варан» — роботехническая платформа. [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://3dnews.ru/563129>
- [3] Белов, А. В. Самоучитель по микропроцессорной технике / А. В. Белов – [2-е изд.] – СПб.: Наука и техника, 2007. – 240 с.
- [4] Евстифеев А.В. Микроконтроллеры AVR Семейства Tiny. / Евстифеев А.В. – М.: Издательский дом «Додэка-XXI», 2007. – 432 с.
- [5] Евстифеев А.В. Микроконтроллеры AVR Семейства Mega. / Евстифеев А.В. – М.: Издательский дом «Додэка-XXI», 2007. – 592 с.
- [6] Wi-Fi модуль ESP8266. – Электронные данные. – Режим доступа: <https://esp8266.ru>
- [7] Arduino Uno – распиновка и подключение. – Электронные данные. – Режим доступа: <https://arduinoplus.ru>
- [8] Работа с ESP8266 NodeMcu v3 Lua. – Электронные данные. – Режим доступа: <https://arduinomaster.ru>
- [9] Адаптер плата NODE MCU ESP8266. – Электронные данные. – Режим доступа: <https://arduinomania.in.ua>
- [10] Сокет (программный интерфейс). – Электронные данные. – Режим доступа: [https://ru.wikipedia.org/wiki/Сокет\\_\(программный\\_интерфейс\)](https://ru.wikipedia.org/wiki/Сокет_(программный_интерфейс))
- [11] Введение в межсетевое взаимодействие. – Электронные данные. – Режим доступа: [http://support.mdl.ru/pc\\_compl/doc/cisco](http://support.mdl.ru/pc_compl/doc/cisco)
- [12] ESP-модули и их классификации. – Электронные данные. – Режим доступа: <http://ulran.ru>

## ПРИЛОЖЕНИЕ А (обязательное)

### Исходный текст аппаратной части

#### Файл arduino.ino:

```
#include <SoftwareSerial.h>

const int rx = 10;
const int tx = 11;
const int speed = 9600;

const String HELP = "HELP";
const String MOVEMENT = "MOVEMENT";
const String SENSORS = "SENSORS";
const String WIFI_SESSION = "WIFI_SESSION";

int RIGHT_UP = 7;
int RIGHT_DOWN = 6;
int LEFT_UP = 5;
int LEFT_DOWN = 4;
bool wifiSession = false;

SoftwareSerial* wifiModule = new SoftwareSerial(rx, tx);

void setup() {
  Serial.begin(speed);
  wifiModule->begin(speed);
  wifiModule->listen();
  pinMode (LEFT_DOWN, OUTPUT);
  pinMode (LEFT_UP, OUTPUT);
  pinMode (RIGHT_DOWN, OUTPUT);
  pinMode (RIGHT_UP, OUTPUT);
}

void loop() {
  String cmd = Serial.readString();

  cmd = cmd.substring(0, cmd.length() - 2);
  if (cmd == WIFI_SESSION) {
    wifiSession = true;
  }
  if (cmd.length() > 0) {
    sendToWifiModule(cmd);
    Serial.println("From serial: " + cmd);
  }
  if (wifiModule->available()) {
    String cmd = wifiModule->readString();
    Serial.println("From wifiModule: " + cmd);

    if (wifiSession) {
      Serial.println("wifi session enabled \n");
      startWifiSession();
    }
  }

  delay(100);
}

void startWifiSession() {
  while(wifiSession) {
    if (wifiModule->available()) {
      String cmd = wifiModule->readString();
      Serial.println("From wifiModule: " + cmd);

      handleCmd(cmd);
    }
  }
}

void sendToWifiModule(String data) {
```

```

    wifiModule->print(" " + data);
}

void handleCmd(String cmd) {
    if (cmd == "HELP") {
        sendToWifiModule("- " + MOVEMENT + "\n- " + SENSORS);
    }
    else if (cmd == MOVEMENT) {
        movementMenu();
        sendToWifiModule(MOVEMENT + " exit");
    }
    else if (cmd == SENSORS) {
    }
    else if (cmd == "stop") {
        wifiSession = false;
        sendToWifiModule("WiFi session stopped");
    }
    else {
        sendToWifiModule("Unknown command: " + cmd);
    }
}

void movementMenu() {
    sendToWifiModule("1.Forward\n2.Back\n3.Left\n4.Right\n5.Stop\n0.Exit");
    while (1) {
        if (wifiModule->available()) {
            String cmd = wifiModule->readString();
            if (cmd == "1") {
                moveForward();
            }
            else if (cmd == "2") {
                moveBack();
            }
            else if (cmd == "3") {
                moveLeft();
            }
            else if (cmd == "4") {
                moveRight();
            }
            else if (cmd == "5") {
                moveStop();
            }
            else if (cmd == "0") {
                return;
            }
            sendToWifiModule(cmd);
        }
    }
}

void moveForward() {
    digitalWrite (RIGHT_DOWN, LOW);
    digitalWrite (RIGHT_UP, HIGH);
    digitalWrite (LEFT_UP, HIGH);
    digitalWrite (LEFT_DOWN, LOW);
}

void moveBack() {
    digitalWrite (RIGHT_DOWN, HIGH);
    digitalWrite (RIGHT_UP, LOW);
    digitalWrite (LEFT_UP, LOW);
    digitalWrite (LEFT_DOWN, HIGH);
}

void moveLeft() {
    digitalWrite (RIGHT_DOWN, LOW);
    digitalWrite (RIGHT_UP, HIGH);
    digitalWrite (LEFT_UP, LOW);
    digitalWrite (LEFT_DOWN, LOW);
}

void moveRight() {
    digitalWrite (RIGHT_DOWN, LOW);
    digitalWrite (RIGHT_UP, LOW);
    digitalWrite (LEFT_UP, HIGH);
    digitalWrite (LEFT_DOWN, LOW);
}

```

```

void moveStop() {
    digitalWrite (RIGHT_DOWN, LOW);
    digitalWrite (RIGHT_UP, LOW);
    digitalWrite (LEFT_UP, LOW);
    digitalWrite (LEFT_DOWN, LOW);
}

```

## Файл wifi.ino:

```

#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>

String ssid = "";
String password = "";

const String GET_IP_ADDRESS = "GET_IP_ADDRESS";
const String SET_IP_ADDRESS = "SET_IP_ADDRESS";
const String SET_SSID = "SET_SSID";
const String GET_SSID = "GET_SSID";
const String GET_LOCAL_SSID = "GET_LOCAL_SSID";
const String SET_PASSWORD = "SET_PASSWORD";
const String GET_PASSWORD = "GET_PASSWORD";
const String CONNECT_TO_WIFI = "CONNECT_TO_WIFI";
const String WIFI_SESSION = "WIFI_SESSION";
const String HELP = "HELP";
const int COMMANDS_LENGTH = 8;
String HELP_COMMANDS[]={ GET_IP_ADDRESS, SET_SSID, GET_SSID, GET_LOCAL_SSID, SET_PASSWORD,
GET_PASSWORD, CONNECT_TO_WIFI, WIFI_SESSION };
const int STATE_OK = 4;
IPAddress ip(192, 168, 1, 229);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 0, 0);

SoftwareSerial* controller = new SoftwareSerial(D9, D10);
WiFiServer wifiServer(80);
void setup() {
    controller->begin(9600);
    controller->listen();
    wifiServer.begin();
}

void loop() {
    String cmd = readFromController();
    handleCommand(cmd);

    delay(100);
}

void handleCommand(String cmd) {
    if (cmd == "test") {
        sendToController("test");
    }
    else if (cmd == SET_IP_ADDRESS) {
        approve(cmd);
        String newIp = readFromController();
        int state = setIpAddress(newIp);
        notifyController(state, cmd, getIpAddress());
    }
    else if (cmd == GET_IP_ADDRESS) {
        String ipAddress = getIpAddress();
        sendToController(ipAddress);
    }
    else if (cmd == SET_SSID) {
        approve(cmd);
        delay(100);
        String newSSID = readFromController();
        int state = setSSID(newSSID);
        notifyController(state, cmd, getSSID());
    }
    else if (cmd == GET_SSID) {
        String ssid = getSSID();
        sendToController(ssid);
    }
    else if (cmd == GET_LOCAL_SSID) {
        String ssid = getLocalSSID();
        sendToController(ssid);
    }
}

```

```

    }
    else if (cmd == SET_PASSWORD) {
        approve(cmd);
        String newPassword = readFromController();
        int state = setPassword(newPassword);
        notifyController(state, cmd, getPassword());
    }
    else if (cmd == GET_PASSWORD) {
        String password = getPassword();
        sendToController(password);
    }
    else if (cmd == CONNECT_TO_WIFI) {
        approve(cmd);
        int state = connectToHotspot();
        sendToController(getIpAddress());
        notifyController(state, cmd, getIpAddress());
    }
    else if (cmd == WIFI_SESSION) {
        approve(cmd);
        notifyController(STATE_OK, "WIFI_SESSION", getIpAddress());
        delay(200);
        wifiSession();
    }
    else if (cmd == HELP) {
        help();
    }
    else {
        controller->print("Unknown command: " + cmd);
    }
}

void wifiSession() {
    while(1) {
        WiFiClient client = wifiServer.available();
        String wifiCmd = "";
        if (client) {
            while (client.connected()) {
                while (client.available() > 0) {
                    char c = client.read();
                    if (c == '\n') {
                        sendToController(wifiCmd);
                        String data = readFromController();
                        client.write(data.c_str());

                        if (wifiCmd == "stop") {
                            client.stop();
                            return;
                        }
                        wifiCmd = "";
                    }
                    else {
                        wifiCmd += c;
                    }
                }

                delay(10);
            }
            client.stop();
        }
    }
}

void notifyController(int state, String cmd, String payload) {
    state == STATE_OK
    ? sendToController(cmd + " " + payload)
    : reportError(cmd);
}

void reportError(String cmd) {
    sendToController("Error when " + cmd);
}

void approve(String cmd) {
    sendToController(cmd + "_OK\n");
}

void sendToController(String data) {
    controller->print(data.length() > 0 ? data : "NULL");
}

```

```

}

String readFromController() {
    String data = controller->readString();
    while (data[0] != '*') {
        data = controller->readString();
    }
    return data.substring(1, data.length());
}

int setSSID(String newSSID) {
    ssid = newSSID;
    return STATE_OK;
}

String getSSID() {
    return ssid;
}

String getLocalSSID() {
    return WiFi.SSID();
}

int setPassword(String newPassword) {
    password = newPassword;
    return STATE_OK;
}

String getPassword() {
    return password;
}

int setIpAddress(String newIp) {
    WiFi.config(ip, gateway, subnet);
    return STATE_OK;
}

String getIpAddress() {
    return WiFi.localIP().toString();
}

int connectToHotspot() {
    WiFi.begin(ssid.c_str(), password.c_str());
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        sendToController("Connection to " + ssid);
        controller->flush();
    }
    sendToController("Connected to: " + ssid);
    return STATE_OK;
}

void help() {
    for (int i = 0; i < COMMANDS_LENGTH; i++) {
        sendToController("\n" + HELP_COMMANDS[i]);
    }
}

```

## ПРИЛОЖЕНИЕ Б (обязательное)

### Исходный текст программной части

#### Файл main.py:

```
import socket
from commands import client_commands
import os
import os.path
import sys
import errno
import time
import re

HOST = ''
PORT = 9001

BUFFER_SIZE = 1024
TIMEOUT = 20

OK_STATUS = 200

def wait_ok():
    while (client.recv(2).decode('utf-8') != "OK"):
        print("wait for OK")

def send_ok():
    client.send("OK".encode('utf-8'))

def get_data():
    return client.recv(BUFFER_SIZE).decode('utf-8')

def send_data(data):
    client.send(str(data).encode('utf-8'))

def handle_input_request(request):
    command = request.split()
    name_command = command[0]

    if (len(command) == 2):
        body = command[1]

    if (client_commands.get(name_command) == "echo"):
        send_data(request)
        if (wait_for_ack(name_command) == False):
            return
        echo(body)

    if (client_commands.get(name_command) == "time"):
        send_data(request)
        if (wait_for_ack(name_command) == False):
            return
        get_time()

    if (client_commands.get(name_command) == "download"):
        send_data(request)
        if (wait_for_ack(name_command) == False):
            return
        download(body, request)

    if (client_commands.get(name_command) == "upload"):
        if (is_file_exist(body)):
            send_data(request)
            if (wait_for_ack(name_command) == False):
                return
            upload(body, request)
        else:
            show_error_message("No such file exists")

    if (client_commands.get(name_command) == "delete"):
        send_data(request)
```



```

        if (wait_for_ack(name_command) == False):
            return
        delete(body, request)

    if (client_commands.get(name_command) == "exit"):
        send_data(request)
        if (wait_for_ack(name_command) == False):
            return
        client.close()
        os._exit(1)

def wait_for_ack(command_to_compare):
    while True:
        response = client.recv(BUFFER_SIZE).decode('utf-8').split(" ", 2)

        if not response:
            return False

        sent_request = response[0]
        status = response[1]

        if (len(response) > 2):
            message = response[2]
        else:
            message = None

        if (command_to_compare == sent_request and int(status) == OK_STATUS):
            return True
        elif (message):
            print(message)
            return False
        else:
            return False

def is_server_available(request, command):
    global client

    client.close()

    i = TIMEOUT

    while(i > 0):
        try:
            client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client.connect((HOST, PORT))
            client.send(request.encode('utf-8'))
            wait_for_ack(command)
            return True

        except socket.error as er:
            sys.stdout.write("Waiting for a server: %d seconds \r" % i)
            sys.stdout.flush()

            i -= 1
            time.sleep(1)

    sys.stdout.flush()
    print("\nServer was disconnected")
    sys.stdout.flush()
    return False

def is_file_exist(file_name):
    return os.path.exists(file_name)

def echo(body):
    send_data(body)
    print(get_data())

def get_time():
    print(get_data())

def download(file_name, request):
    size = int(get_data()) # 1

    send_data(0) # 3

    data_size_recv = int(get_data()) # 4

```

```

if (data_size_rcv == 0):
    f = open(file_name, "wb")
else:
    f = open(file_name, "rb+")

while (data_size_rcv < size):
    try:
        data = client.recv(BUFFER_SIZE)
        f.seek(data_size_rcv, 0)
        f.write(data)
        data_size_rcv += len(data)

        progress = (data_size_rcv / (size)) * 100
        sys.stdout.write("Download progress: %d%% \r" % progress)
        sys.stdout.flush()

    except socket.error as e:
        if(is_server_available(request, "download")):
            size = int(get_data())
            send_data(data_size_rcv)
            data_size_rcv = int(get_data())
            print("\n")
        else:
            f.close()
            client.close()
            os._exit(1)

    except KeyboardInterrupt:
        print("KeyboardInterrupt was handled")
        f.close()
        client.close()
        os._exit(1)

f.close()
print("\n" + file_name + " was downloaded")

def upload(file_name, request):
    f = open(file_name, "rb+")

    size = int(os.path.getsize(file_name))

    send_data(size) # 1

    wait_ok() # 2

    send_data(0) # 3

    data_size_rcv = int(get_data()) # 4

    wait_ok() # 5

    f.seek(data_size_rcv, 0)

    while (data_size_rcv < size):
        try:
            data_file = f.read(BUFFER_SIZE)
            client.send(data_file)
            received_data = get_data()

            progress = (data_size_rcv / size) * 100
            sys.stdout.write("Upload progress: %d%% \r" % progress)
            sys.stdout.flush()

        except socket.error as e:
            if(is_server_available(request, "upload")):
                send_data(size)
                wait_ok()
                send_data(data_size_rcv)
                data_size_rcv = int(get_data())
                wait_ok()
                print("\n")
            else:
                f.close()
                client.close()
                os._exit(1)

    except KeyboardInterrupt:
        print("KeyboardInterrupt was handled")

```

```

        f.close()
        client.close()
        os._exit(1)

    if (received_data):
        data_size_recv = int(received_data)
        f.seek(data_size_recv, 0)

    f.close()
    print("\n" + file_name + " was uploaded")

def delete(file_name):
    pass

def exit():
    pass

def check_valid_request(request):
    command = request.split()
    if (len(command) == 0):
        return False
    else:
        return True

def show_status():
    pass

def show_error_message(error):
    print(error)

def show_start_message():
    print("\nWelcome to client cli!")

is_valid_address = False

REGULAR_IP = '^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'
regex = re.compile(REGULAR_IP)

while (is_valid_address == False):
    addr = input("\nInput host address: ")
    if (regex.match(addr)):
        is_valid_address = True
        HOST = addr
    else:
        try:
            HOST = socket.gethostbyname(addr)
            is_valid_address = True
        except socket.error:
            print("Please, input valid address")
            is_valid_address = False

show_start_message()

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))

while True:
    try:
        request = input()
        if (check_valid_request(request)):
            handle_input_request(request)
    except KeyboardInterrupt:
        print("KeyboardInterrupt was handled")
        client.close()
        os._exit(1)

```

ПРИЛОЖЕНИЕ В  
(обязательное)

Схема электрическая функциональная

ПРИЛОЖЕНИЕ Г  
(обязательное)

Схема электрическая принципиальная

ПРИЛОЖЕНИЕ Д  
(обязательное)

Спецификация проекта

ПРИЛОЖЕНИЕ Е  
(обязательное)

Ведомость документов