СОДЕРЖАНИЕ

[Введение 4](#_Toc27672550)

[1 Обзор технологий для конструирования системы управления колёсным роботом 5](#_Toc27672551)

[1.1 Обзор и сравнение одноплатных компьютеров 5](#_Toc27672552)

[1.2 *Arduino* как альтернатива одноплатному комьютеру 8](#_Toc27672553)

[1.3 Питание и передвижение колёсного робота 9](#_Toc27672554)

[1.4 Операционная система TinkerOS. Архитектура комплекса програм- много обеспечения 12](#_Toc27672554)

[1.5 Паттерны проектирования «*MVC*» и «*SingleTone*» 13](#_Toc27672555)

[2 Реализация клиент-серверного программного обеспечения 15](#_Toc27672556)

[2.1 Программное обеспечение на стороне одноплатного компьютера](#_Toc27672558) 15

[2.2 Клиент-серверное приложение – клиент-контроллер 18](#_Toc27672558)

[2.3 Клиент-серверное приложение – клиент-датчик 23](#_Toc27672559)

[3 Опытная эксплуатация и верификация](#_Toc27672560) 24

[3.1 Обзор интерфейса пользователя мобильных приложений](#_Toc27672561) 24

[3.2 Тестирование работы серверного программного обеспечения 27](#_Toc27672562)

[Заключение](#_Toc27672563) 30

[Список используемых источников](#_Toc27672564) 31

[Приложение А Листинг программы](#_Toc27672566) 32

[Приложение Б Схема используемого паттерна «декоратор»](#_Toc27672566) 58

[Приложение В Руководство пользователя](#_Toc27672566) 59

[Приложение Г Руководство программиста](#_Toc27672566) 61

[Приложение Д Руководство системного-программиста](#_Toc27672566) 62

## ВВЕДЕНИЕ

Сельское хозяйство, как отрасль хозяйственной деятельности, играет важную роль в экономике государства. Внедрение современных технических решений в данную отрасль положительно сказывается на качестве и количестве выпускаемой продукции. Что влечёт за собой рост экспорта и пополняемость внутреннего рынка продукцией собственного производства.

Важными задачами является защита сельскохозяйственных культур от вредителей, сбор и обработка данных о состоянии грунта. Если первая решается рассеиванием пестицидов с использованием лёгкой летающей техники, то последняя имеет статический характер. Как правило, это разнообразные датчики, измеряющие температуру на ферме, влажность, количество осадков, скорость ветра и наличие вредителей в земле. Но, в силу масштабов полей, сбор информации с данных датчиков является проблемой. Наилучшим решением может стать использование беспроводных сетей. Радиус покрытия стандартного роутера на ровной местности составляет около ста пятидесяти метров, чего недостаточно для охвата среднего по размеру сельскохозяйственного участка. Это приводит к выводу о невозможности использования статической точки, занимающейся сбором информации. Как решение, может выступать колёсный робот. Целью которого будет являться перемещение по заданной местности, сбор и доставка информации с датчиков на основную базу. На которой, в последствии будет производиться её дальнейший анализ.

Отталкиваясь от уровня интеграции технических средств в сферу сельского хозяйства, можно заметить необходимость внедрения новых решений, улучшения и пересмотр старых. Система программного обеспечения (ПО) и технических решений может стать выгодной заменой ручному труду, увеличив производительность и точность работы.

Для обеспечения своевременного сбора информации с сельскохозяйственных датчиков необходимы следующие машинно-аппаратные единицы:

* колёсный робот с одноплатным компьютером в качестве руководящего модуля;
* клиент-серверное приложение (сервер), отвечающее за сбор информации и передвижение робота;
* датчик, с необходимым для сбора и отправки информации ПО;
* клиент-серверное приложение (клиент), обеспечивающий как стандартное управление движением колёсного робота, так и управление с помощью заранее заданной траектории – карты;
* клиент-серверное приложение (клиент), осуществляющий хранение и анализ информации.

## 1 ОБЗОР ТЕХНОЛОГИЙ ДЛЯ КОНСТРУИРОВАНИЯ СИСТЕМЫ УПРАВЛЕНИЯ КОЛЁСНЫМ РОБОТОМ

## Обзор и сравнение одноплатных компьютеров

Обширность рынка одноплатных компьютеров выходит за рамки понимания. Производители, в гонке за продаваемостью, представляют новые варианты изделий. Спрос на низко-ценовой сегмент продукции особенно высок. Потому рынок имеет разнообразие дешёвых но, тем не менее, качественных представителей одноплатных компьютеров, таких как: [*Raspberry Pi*](https://www.google.com/url?q=http://linuxgizmos.com/pi-zero-tweak-adds-camera-connector-keeps-5-price/&sa=D&ust=1589321424511000&usg=AFQjCNF4SwwpkUlRonOIHyTz_TeJmX9y0w)*,* [*Orange Pi,*](https://www.google.com/url?q=http://linuxgizmos.com/headless-orange-pi-zero-sbc-has-wifi-and-ethernet-for-7/&sa=D&ust=1589321424485000&usg=AFQjCNEkntdmQAQL5xO2GsOaZf3cdpDgqg)

[*Odroid-C0*](https://www.google.com/url?q=http://linuxgizmos.com/25-dollar-odroid-c0-sbc-runs-linux-or-android-on-optional-battery/&sa=D&ust=1589321424444000&usg=AFQjCNHj_LBilnPMr6UPt6gpLRgipuQMSA)*,* [*Arduino Industrial 101*](https://www.google.com/url?q=http://linuxgizmos.com/new-arduino-srl-sbc-merges-arduino-wifi-and-linux/&sa=D&ust=1589321424269000&usg=AFQjCNGz6EeRWQvkinKr97fGlVbKPCqZlA) *и другие.* Их цена в среднем значении не превышает тридцати долларов США. Линейка *Paspberry Pi* является же самой недорогостоящей. Цены большинства плат *Paspberry Pi* находятся в пределах от десяти до двадцати долларов США [1]. Функциональность и цена – основные причины популярности *Paspberry Pi* в данном рыночном сегменте.

Однако конкурентность за долю рынка сподвигает других производителей создавать варианты одноплатных компьютеров, противопоставляемых *Paspberry Pi*. Одним из достойных конкурентов *RPi* является *Asus Tinker Board*.

Ниже приведено сравнение двух плат.

*Raspberry Pi 3 Model B* –одноплатный компьютер компании *Raspberry Pi Trading.* Явлется самым популярным в мире. Стоимость составляет 35 долларов США. Центральный процессор – *Broadcom BCM2837* с частотой 1.2 ГГц. Графический процессор – *Broadcom VideoCore IV* с частотой в 400 МГц. Оперативная память на 1 ГБ типа *SDRAM*. Оснащена модулями *WiFi* и *Bluetooth*. На изображении 1.1 представлен внешний вид *RPi*.



Рисунок 1.1 – *RaspberryPi* *3 Model B*

*Raspberry* *Pi 3 Model B* не имеет в открытом доступе полных схем и лицензии. Многие её конкуренты могут предложить более выгодные решения как в техническом так и ценовом плане. Но при этом *RPi* имеет огромное развивающееся сообщество, что является весомым преимуществом для начинающих в данной области. Все платы *Raspberry* *Pi* обратно совместимы.

*Tinker* *Board* – одноплатный компьютер, разработанный в крупной компании, производителя персональных компьютеров (ПК), – *Asus*. Стоимость составляет шестьдесят долларов США. Центральный процессор – *Rockchip* *RK3288* с частотой 1.8 ГГц. Графический процессор – *Mali-T760*. Оперативная память на 2 ГБ типа *LPDDR3* (преимущественно используется в современных мобильных и планшетных устройствах). На изображении 1.2 представлен внешний вид *Tinker Board*.



Рисунок 1.2 – *Asus* *Tinker Board*

*Tinker* *Board* – первая плата, основанная на процессоре *Rockchip* в размерах *Raspberry* *Pi*, и с 40-*pin* разъёмом. *Tinker* *Board* доступен со схемами, *2D* и *3D* чертежами и другими ресурсами открытой спецификации.

*TinkerOS* 1.8 основан на новейшем ядре *Debian* 9 и включает оптимизированный веб-браузер *Chromium*, приложения для программирования на *Python* и *Scratch*, а также медиаплеер, который был разработан совместно с *Rockchip*. *TinkerOS* в настоящее время требуется для воспроизведения в *4K*-формате.

Плата *RPi3* почти в два раза дешевле, но *Tinker* *Board* имеет преимущество в скорости (несмотря на всё те же *32bit*), имеет более мощный *GPU* и в два раза больше *RAM*, а так же *GbE* вместо *Fast* *Ethernet*. Из дополнительных преимуществ можно отметить поддержку воспроизведения видео в *4K/30fps*.

Доступен сайт сообщества с форумом и схемами, *2D* и *3D* чертежами, и другими ресурсами. Единственным доступным образом является *TinkerOS*, основанная на *Debian*, с десктопом *LXDE*, поддержка *Android* в разработке.

В таблице 1.1 представлены для сравнения некоторые технические характеристики *Tinker Board*, *Raspbery* *Pi 3 Model B* и некоторых других плат.

Таблица 1.1 – Технические характеристики одноплатных компьютеров

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Product* | *Price ($)* | *Processor* | *Cores* | *3D GPU* | *RAM* | *Storage* | *OSes* |
| [*Banana Pi BPI-M2 Ultra*](http://linuxgizmos.com/latest-banana-pi-offers-sata-and-2gb-ram/) | 40 | *Allwinner R40* | *4x A7* | *Mali-400 MP2* | *2GB* | *8GB to 64GB eMMC; SATA* | *Linux, Android* |
| [*Gizmo 2*](http://linuxgizmos.com/amd-gizmo-2-sbc-powers-up-with-g-series-soc/) | 165 | *AMD G-Series GX210HA* | *2x x86 @ 1GHz* | *Radeon 800* | *1GB* | *mSATA. 2X SATA* | *Linux* |
| [*Orange Pi PC 2*](http://linuxgizmos.com/first-64-bit-orange-pi-slips-in-under-20/) | 20 | *Allwinner H5* | *4x A53* | *Mali-450 MP2* | *1GB* | *no* | *Linux, Android* |
| [*Raspberry Pi 2 Model B*](http://linuxgizmos.com/raspberry-pi-gets-quad-core-soc-keeps-same-price/) | 35 | *Broadcom BCM2837* | *4x A53 @ 900MHz* | *VideoCore IV* | *1GB* | *no* | *Linux* |
| [*Raspberry Pi 3 Model B*](http://linuxgizmos.com/quad-core-64-bit-pi-3-is-official-with-wifi-bt-and-35-price/) | 35 | *Broadcom BCM2837* | *4x A53 @ 1.2GHz* | *VideoCore IV* | *1GB* | *no* | *Linux* |
| [*Tinker Board*](http://linuxgizmos.com/tinker-board-ships-in-u-s-at-60/) | 60 | *Rockchip RK3288* | *4x Cortex-A17 @ 1.8GHz* | *Mali-T760 GPU* | *2GB* | *no* | *Linux* |

Отталкиваясь от представленной выше информации можно сделать вывод о неоспоримом превосходстве *Tinker* *Board* в техническом плане над *RPi*. Так как разрабатываемая система приложений может быть требовательна к производительности и стабильности сетевого соединения – плата Asus Tinker Board является лучшим выбором. Для окончательного подтверждения сказанного, ниже представлены несколько графиков с официального сайта производителя, доказывающих превосходство в сетевом плане выбранной платы над другими среднестатистическими одноплатными компьютерами (*SBC* – *Session* *Border* *Controller*).

*Tinker* *Board* предлагает двухканальный *DDR3*, который обеспечивает лучшую пропускную способность памяти по сравнению с конкурирующими устройствами, которые предлагают только одноканальный *DDR2*.

Производительность сетей. *Tinker* *Board* оснащен гигабитным *Ethernet*, который предлагает значительно улучшенную пропускную способность по сравнению с конкурирующими *SBC*, которые оснащены 10/100 *Ethernet*.

Производительность *Wi*-*Fi* на плате *Tinker* *Board* выше, чем у большинства конкурирующих устройств, что позволяет улучшить прием сигнала.

Специальный контроллер *Tinker* *Board* и конструкция шины без общего доступа обеспечивают превосходную доставку и прием пакетов.

Производительность локальной сети остается неизменной на плате *Tinker* во время передачи по *USB* по сравнению с производительностью локальной сети конкурирующих *SBC*, которые во время передачи *USB* снижают скорость до 18%.

* 1. ***Arduino* как альтернатива одноплатному компьютеру**

Использовать в проекте одну плату, имеющую сразу всё необходимое – простое и логичное решение. Но существуют более дешёвые варианты, позволяющие реализовать ту же функциональность. Например заменой выбранному *Tinker* *Board* может служить тандем из платы *Arduino* и любого модуля с технологией приёма/передачи данных по радиоканалу. Это может быть не только *WIFI*, но и *BlueTooth* модуль. Так же, для решения задачи, возможно использовать обычные радио-модули, например *nrf24l01*, представленный на рисунке 1.6.

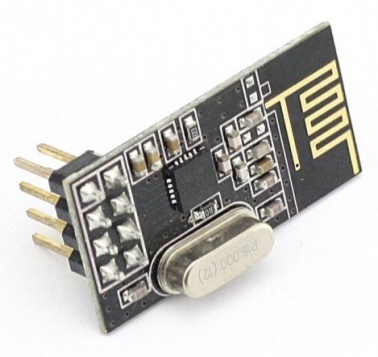


Рисунок 1.6 – Радио-модуль *nrf24l01*

Датчик *nrf24l01* имеет очень низкую стоимость и мог бы занять место в текущем проекте. Но так как для имитации сельскохозяйственных датчиков, осуществляющих сбор данных с полей, выбраны мобильные телефоны (подробнее во втором разделе) – вариант с использованием радио-датчика не применим. Существует своеобразное решение, позволяющее передавать данные с радио-датчика на любое *BlueTooth* устройство, используя технологию *Bluetooth* *Low* *Energy* (*BLE*) [2]. Но оно исключает обратную отправку данных на сам датчик. Этот факт вычёркивает датчик из возможных решений, уступая путь не столь дешёвым *Wifi* и *BlueTooth* модулям.

Платы Arduino часто используют для реализации простых проектов, таких как, например, автоматизированный полив растений, умная подсветка и прочего. На рисунке 1.7 представлен внешний вид *Arduino* *Nano*.

Настоящий проект предполагает сбор и хранение информации роботом до тех пор, пока она не будет доставлена на базу для дальнейшей обработки. С учётом масштабов полей и количества возможных собирающих датчиков, предоставляемой *Arduino* *Nano* или *Arduino* *Uno* *EEPROM* памяти (*Electrically* *Erasable* *Programmable* *Read*-*Only* *Memory*) может не хватить. Так как её размер составляет всего 1000 байт (1Кб).

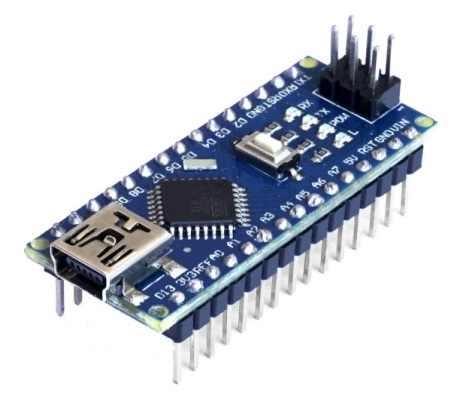


Рисунок 1.7 – *Arduino* *Nano*

Несмотря на преимущество огромного сообщества *Arduino* и её стоимости, решение использовать *Arduino* и *WiFi* модуль, несомненно, проигрывает одноплатному компьютеру в рамках реального проекта, реализованного для сельскохозяйственной местности.

* 1. **Питание и передвижение колёсного робота**

Движение – неотъемлемая часть настоящего проекта. Для организации передвижения возможно использовать один из существующих видов электродвигателей. Самым простым и дешёвым является коллекторный электродвигатель постоянного тока, его часто можно встретить в детских игрушках. Он прост в управлении, маломощен и быстро приходит в негодность. Модель, представленная на рисунке 1.8 (слева), выдерживает максимальное напряжение в 6В.

Наиболее полезными являются два других рассматриваемых варианта –шаговый двигатель и сервопривод, которые находятся на рисунке 1.8 в центре и справа соответственно.

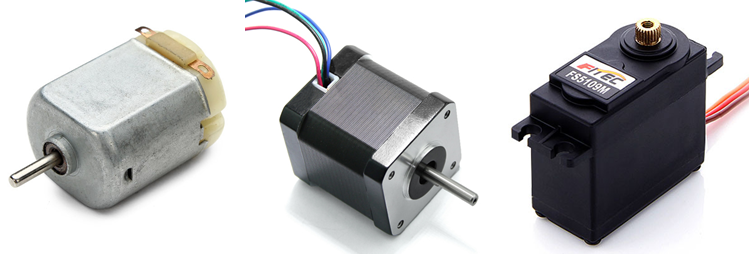


Рисунок 1.8 – Коллекторный, шаговый и серво двигатели (слева на право)

Шаговый двигатель – это бесщеточный электромотор, работающий в паре плата-драйвер. Как правило, шаговые двигатели имеют несколько фаз (обмоток), поочередно включаемых драйвером. Двигатель поворачивается за счет подачи короткого импульса на одну из обмоток статора, в результате чего в движение приводится магнитный ротор. Шаговые двигатели встречаются, например, в *3D* принтерах.

Сервопривод – это обычный мотор с дополнительно установленным датчиком контроля, выполняющим функцию обратной связи. При работе мотор будет удерживаться в заданном положении с помощью контроллера. Такой принцип взаимосвязи позволяет добиться высокой скорости и точности оборудования вплоть до одного микрона. Чтобы зафиксировать движение в одном положении и при этом не заставить его двигаться в обратном, контроллер должен постоянно переключать ток двигателя на противоположенный, пока не поступит следующая команда. При таком подходе пропуск шагов исключен, так как отклонения вала отслеживаются и корректируются [3].

У каждого из представленных двигателей есть своя ниша применения, от чего они не являются конкурентами.

Для реализации дешёвого прототипа колёсного робота достаточно двух обычных коллекторных моторов, для поворотов и, непосредственно, заднего привода. Для управления двигателями постоянного тока, рассчитанными на большее напряжение, чем 5В, необходимо управлять через полевой (биполярный) транзистор или через драйвер *L298N*. Распиновка и внешний вид модуля, содержащего микросхему *L298N*, представлены на рисунке 1.9.

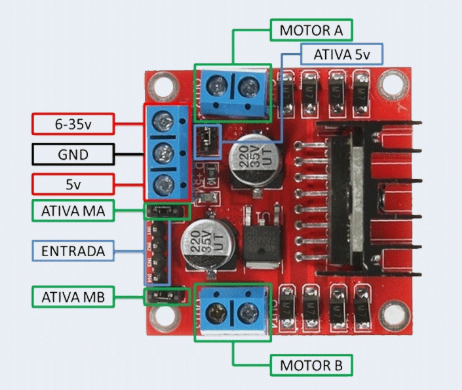


Рисунок 1.9 – Модуль *L298*

Драйвер двигателей позволяет управлять скоростью и направлением вращения мотора. А так же даёт возможность питать двигатели отдельно от главной составляющей проекта. Так как, при старте и завершении работы, двигатели постоянного тока создают выбросы тока, превышающие 40мА, что может негативно сказаться на пинах управляющего устройства.

Одной из наиболее мощных функций одноплатных компьютеров является интерфейс общего назначения *GPIO*. Эти пины используются для подключения и контроля внешних устройств, путём программирования.  *Tinker* *Board* имеет сорок контактов, двадцать восемь из которых являются контактами общего назначения. Они совмещены с контактами шин SPI/*UART*/I2C, причем шина *SPI* может работать с двумя ведомыми чипами. Распиновку можно найти на официальном сайте Asus или в документации.

Модуль *L298* имеет 4 логических выхода и 2 выходя под ШИМ для управления двумя коллекторными моторами или одним шаговым. Логические выходы контролируют направление вращения, а ШИМ отвечает за подаваемое напряжение, то есть скорость двигателей.

Совместив данные возможности управления с выходами программируемой платы, можно получить простого робота с регулируемой скоростью передвижения.

Есть два варианта использования имеющихся двух выходов под моторы постоянного тока. Первый – использовать один двигатель как основной способ передвижения, а другой направить на работу с поворотом передних колёс. Второй – разместить оба двигателя на задней части робота и использовать как движущие и управляющие поворотами детали одновременно. В таком варианте прокручивание одного и отсутствие движения у другого мотора обеспечат поворот.

Схема подключения драйвера, моторов и одноплатного компьютера представлена на рисунке 1.10.

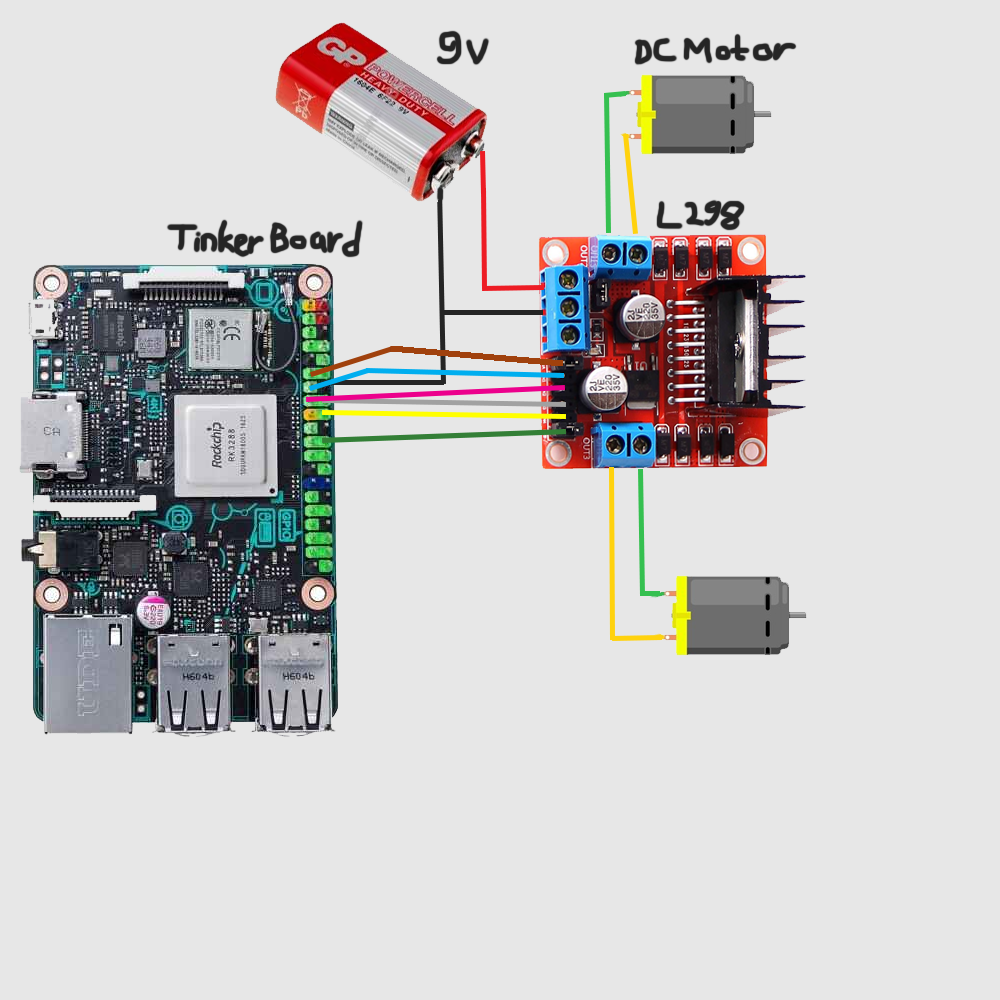


Рисунок 1.10 – Схема подключения частей колёсного робота

*Tinker* *Board* требует для питания 5В при потреблении тока 2-3A. Моторы питаются отдельно от 9В. Так как робот не является стационарным – для питания *Tinker* *Board* можно использовать, например, литиевый аккумулятор, соблюдающий необходимый уровень напряжения и способный выдавать требуемое количество тока. Более дешёвым вариантом является использование кроны на 9B с понижающим напряжение элементом. Путём несложных вычислений можно прийти к цифре в двадцать минут до полного разряда батареи (при активном режиме одноплатного компьютера). Этого достаточно для проведения полевых испытаний прототипа колёсного робота.

* 1. **Операционная система *TinkerOS*. Архитектура комплекса программного обеспечения**

Несмотря на рекомендации производителя, использовать языки *Python* и *C* для написания программ на *Tinker* *Board*, так же существует возможность писать на *Java* и *C*#. Но при этом последние два не имеют официальных библиотек для управления контактами *GPIO*, а лишь любительскую реализацию.

Для программирования, под выбранным одноплатным компьютером, надёжнее, проще, а также быстрее в разработке будет использовать язык *Python*. Он достаточно гибок, чтобы вместить простой сервер в десять строк своего кода. Так же *Asus* предоставляет готовые примеры скриптов на *python*.

С архитектурной точки зрения выгодным решением будет организовать сервер на стороне робота. А так же создать несколько типов клиентов, что схематично изображено на рисунке 1.11.

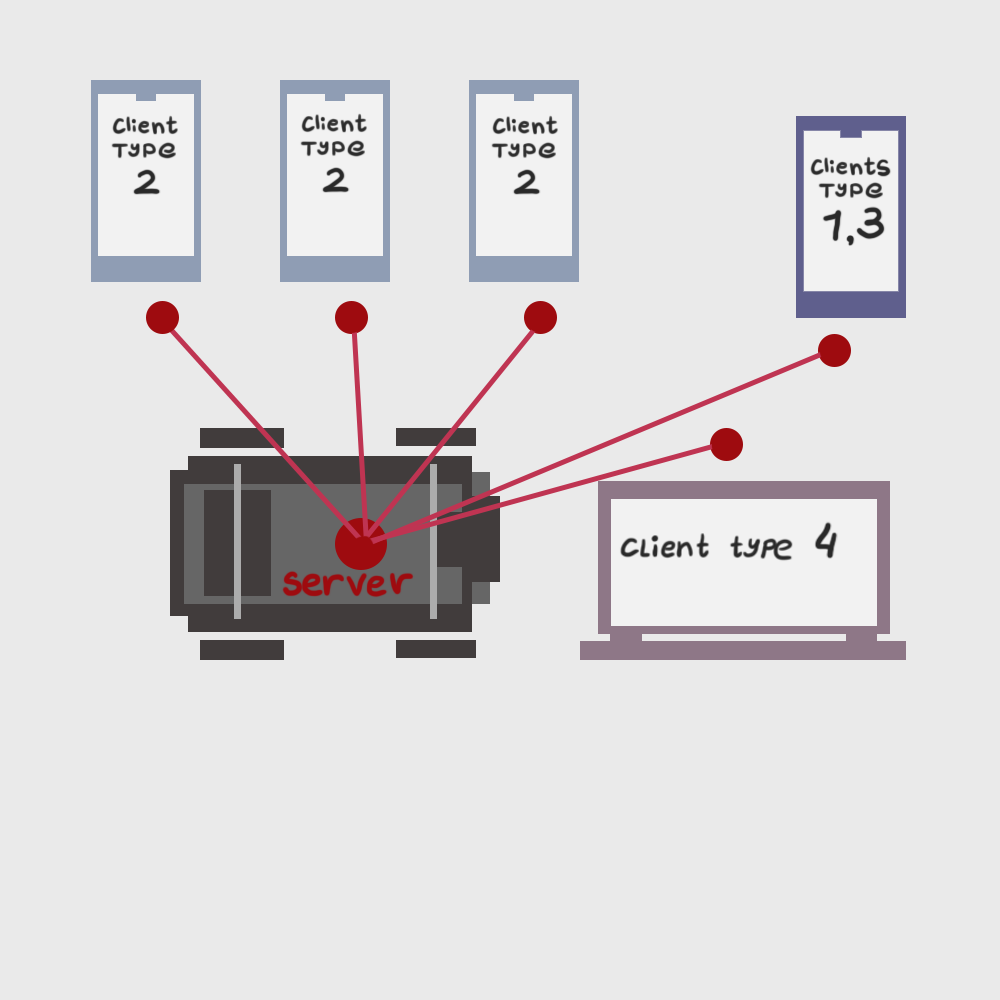


Рисунок 1.11 – Схема подключения частей колёсного робота

Первым типом клиента выступает контроллер движений робота посредством кнопок вперёд, назад, вправо, влево и кнопки стоп. Клиент предназначен для ручного управления роботом, которое может быть полезно в случае сбоя в работе системы ориентирования по карте. А так же довольно удобен при тестировании.

Второй тип клиента имитирует сельскохозяйственный датчик. Он располагает встроенным таймером и примитивной логики подключения, отправки данных и отключения от сервера.

Третий тип – усовершенствованный контроллер. Представляет редактор двухмерной карты, задающей траекторию движения робота, с возможностью загружать уже готовые карты из стороннего источника. Путь, указанный на изображении, анализируется на самом клиенте и передаётся серверу в виде последовательности примитивных команд.

Клиент четвёртого типа – это, непосредственно, база. Он принимает собранные роботом данные, хранит и обрабатывает их. Имеет средства анализа информации, в виде графиков.

Комплекс выше перечисленных приложений дополняется консольным клиентом, необходимым для отладки и проверки работоспособности сервера. Так как подключение периферийных устройств к одноплатному компьютеру не является удобным с точки зрения пользователя и его времени.

Система данных приложений обеспечивает слаженный подход к решению поставленной проблемы.

* 1. **Паттерны проектирования «*MVS*» и «*SingleTone*»**

В начале проектирования следует разделять будущее приложение на три основных блока: модель, представление и контроллер. Данный подход – это паттерн проектирования *MVC* (*Model-View-Controller*). Он необходим для разделения логики приложения, данных и интерфейса в три самостоятельные сущности, взаимодействующие между собой. Независимость сущностей позволяет легко интегрировать в приложение ту или иную функциональность, а также изменять уже существующую.

Модель – статические данные. Это может быть база данных или совокупность классов, отображающих сущности проектируемой области.

Контроллер – сущность, которая отслеживает действия пользователя. Может иметь логику взаимодействия классов модели. Отвечает за управление.

Представление – содержит логику того как будут отображаться данные модели. Как правило, представление неразрывно связано с интерфейсом пользователя или является им [4].

Паттерн *MVC* имеет два основных варианта.

В первом, как отображено на рисунке 1.12, модель взаимодействует с представлением.

Цикл работы приложения следующий: действия пользователя поступают на контроллер, контроллер изменяет данные модели в соответствии с действиями пользователя, модель передаёт представлению собственные данные, представление отображает полученную информацию.

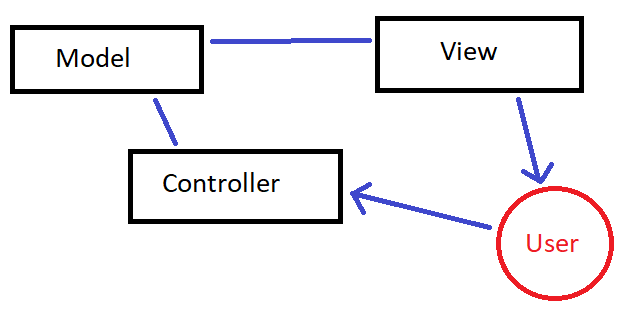


Рисунок 1.12 – Первый вариант реализации паттерна *MVC*

Второй вариант характерен тем, что модель является полностью статичной и задача её отображения переходит к контроллеру, как показано на рисунке 1.13. Второй вариант применим для веб-приложений. Он способен уменьшить общее число зависимостей.

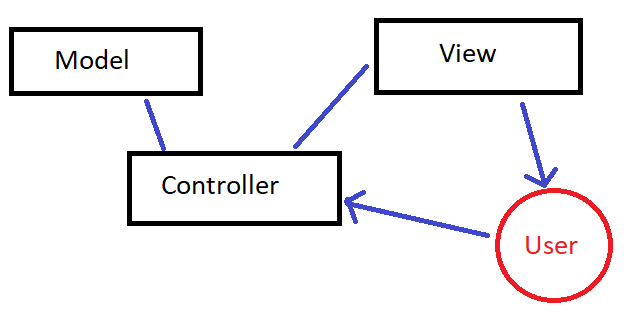


Рисунок 1.13 – Второй вариант реализации паттерна *MVC*

Тесная совместимость с данным шаблонным решением прослеживается в разработке приложений на платформах направленных на использование графического интерфейса, таких как *WPF* и WinForms в *C#*, *Android* приложениях, а также направленных на веб, например, *ASP*.*NET* *Core* и других.

Шаблон проектирования *SingleTone*, несмотря на его нарушение принципа единственной ответственности (*Single* *responsibility* *principle* – *SRP*) решает две проблемы. Первая – гарантия наличия единственного экземпляра класса. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных или сокету, используемому в нескольких частях приложения.

Вторая – предоставляет глобальную точку доступа. И, в отличие, от обычных глобальных переменных (создание которых невозможно в некоторых языках программирования, например, таких как *C#*) экземпляр защищён от записи.

Два представленных шаблона часто используются и незаменимы для некоторых задач.

1. **РЕАЛИЗАЦИЯ КЛИЕНТ-**СЕРВЕНОГО ПРОГРАММНОГО **ОБЕСПЕЧЕНИЯ**
   1. **Программное обеспечение на стороне одноплатного компьютера.**

Клиент-серверное приложение, находящееся на платформе одноплатного компьютера *Tinker* *Board* играет роль ключевого звена во всей системе приложений – сервера. Он представляет из себя скрипт с набором инструкций, которые следует выполнять при подключении того или иного типа клиента.

После запуска сервер становится в позицию ожидания подключений. Он не поддерживает многопользовательность. От того в один момент времени сервер может принять подключение у одного клиента и работать с ним до тех пор, пока тот не отключится. После отключения клиента сервер вновь переходит в режим ожидания подключений.

Данное решение просто в своей реализации и делает приложение сервера более лёгким и читабельным. При этом эффективность работы всего программного комплекса не снижается.

В силу невозможности быстрого решения появившейся ошибки, серверное приложение максимально застраховано от возможного падения. Так как в противном случае пришлось бы перезапускать одноплатный компьютер, что привело бы к потере собранной информации. Архитектурное решение системы приложений не предусматривает хранение собранных данных в постоянной памяти робота, вся собранная информация находится в оперативной памяти до тех пор, пока не будет передана на базовое приложение, которым будет помещено в базу данных.

Сервер имеет консольное отображение информации, преимущественно используемое для отладки и тестирования. Запускается со стартом самого Tinker Board.

Для подключения клиента к серверу необходимо указать *IP*-адрес (*Internet Protocol Address*) и порт. Порт задан постоянной переменной и не изменен при каждом новом запуске.

Существует несколько вариантов узнать *IP*-адрес одноплатного ПК. Одним из представительных решений может оказаться широковещательная рассылка. Более простой способ – использовать статический *IP* [5].

После подтверждения удачного подключения клиент отправляет серверу свой тип – число от одного до четырёх.

На стороне сервера есть 4 основных метода для работы с соответствующими типами клиентов. Методы, находящиеся в классе *Server* можно увидеть на рисунке 2.1. Методы, обслуживающие клиентов имеют названия: *messagin*\_*base*, *map*\_*controller*, *controller* и *get*\_*data*.

Первый тип клиента отвечает за непосредственный контроль движениями робота. Приняв подключение первого клиента, сервер переходит в режим считывания отправляемых им сообщений (подробнее о клиенте первого типа написано в главе 2.2).

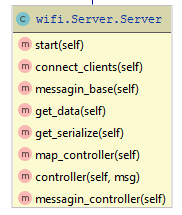


Рисунок 2.1 – Значения выходов

Сообщения приходят в виде строка-число. Если это команда направления или остановки движения – число отвечает за количество секунд выполнения данной операции. Если строкой является команда поворота, то число здесь – это градусы угла поворота. Есть команды, в которых присутствие числа игнорируется – это команды регулирования скорости (они заданы жёстко)

Таблица 2.1 – Описание команд управления движениями робота

|  |  |  |
| --- | --- | --- |
| Команда | Обозначение | Применение |
| *Forward* | *f* | *f2* |
| *Backward* | *b* | *b1* |
| *Stop* | *s* | *s5* |
| *Rotate right* | *rr* | *rr90* |
| *Rotate left* | *rl* | *rl30* |
| *Speed high* | *h* | *h* |
| *Speed medium* | *m* | *m* |
| *Speed low* | *l* | *l* |

Второй тип клиента является имитацией датчика. Приняв подключение от второго клиента, сервер переходит в режим ожидания одного сообщения в виде собранных данных. После получения необходимой информации разрывает соединение с клиентом. Подробнее о клиенте второго типа написано в подразделе 2.3.

Третий тип клиента представляет собой карту. Его задача – принять от пользователя изображение маршрута передвижения робота, обработать его и отправить последовательность команд, перечисленных в таблице 2.1, для дальнейшего исполнения их роботом (подраздел 2.2). Сервер, как и в случае со вторым клиентом, переходит в режим ожидания принятия сообщения. Когда сообщение доставлено сервер отключает клиента и начинает последовательное исполнение команд. Команда остановки в данном случае означает ожидание подключения датчика.

Четвёртым типом является базовое приложение для настольного ПК. При соединении с этим типом сервер отправляет имеющиеся данные клиенту и разрывает соединение.

Сервер использует в работе классы, которые имеют прямой доступ к выходам *GPIO* одноплатного компьютера. Для управления пинами используется класс *GPIO* библиотеки *ASUS*.*GPIO*. На рисунке 2.2 указаны задействованные для контроля двигателей пины. Значения «*en*» отвечают за скорость (путём регулировки подаваемого напряжения), а значения «in» являются логическими и отвечают за направление вращения двигателя. Соответственно «*enA*», «*in1*» и «*in2*» контролируют первый мотор, а «*enB*», «*in3*» и «*in4*» – второй.

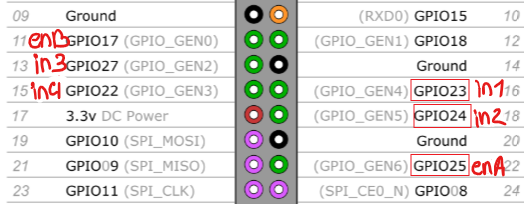


Рисунок 2.2 – Значения выходов *GPIO* одноплатного компьютера *Tinker* *Board*

Был разработан класс, использующий методы стандартного класса *GPIO* для реализации простых функций, таких как: движение вперёд, движение назад, остановка, изменение скорости вращения мотора. На изображении 2.3 представлены поля и методы класса *Controller*, управляющего каждым мотором в отдельности и работающего непосредственно с выходами *GPIO*, и класса *Moove*, который реализует повороты робота, при которых должны быть задействованы оба двигателя, так как существует необходимость точного контролирования градусов поворота робота.

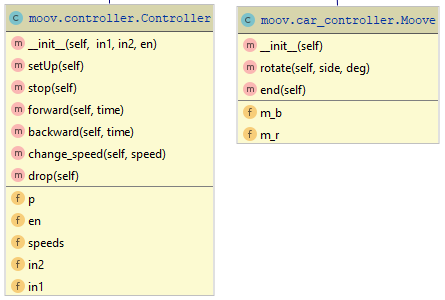


Рисунок 2.3 – Классы *Controller* и *Moove*

Также класс *Moove* завершает работу с пинами в методе end, вызывая методы *drop* у каждого из двух экземпляров класса *Controller* (один экземпляр принадлежит одному мотору).

На этом программное обеспечение на стороне одноплатного компьютера завершается. На изображении 2.4 указана архитектура расположения скриптов.

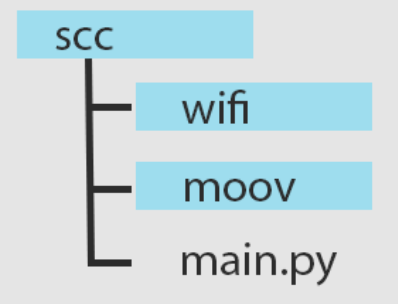


Рисунок 2.4 – Архитектура каталогов ПО на одноплатном компьютере

Скрипт *main.py* приводит в запуск классы, хранящиеся в директориях *wifi* и *moov*. Запуск осуществляется последовательным введением команд: *cd scc* и *python main.py* в консольное окно *TinkerOS*.

* 1. **Клиент-серверное приложение – клиент-контроллер**

Руль управления движениями робота должен находиться на переносимом устройстве, что добавит гибкости управлению на открытых пространствах. Это стало причиной разработки мобильного приложения. Платформой разработки выбран *Android*. Языком программирования – *java*.

Для старта движения роботу необходимо задать карту передвижения, либо управлять им вручную – разработанный мобильный контроллер решает две вышеперечисленные задачи.

На изображении 2.5 отображена архитектура разработанных классов и каталогов, в которых они находятся. Схему взаимодействия классов мобильного приложения-контроллера можно увидеть в приложении Б.

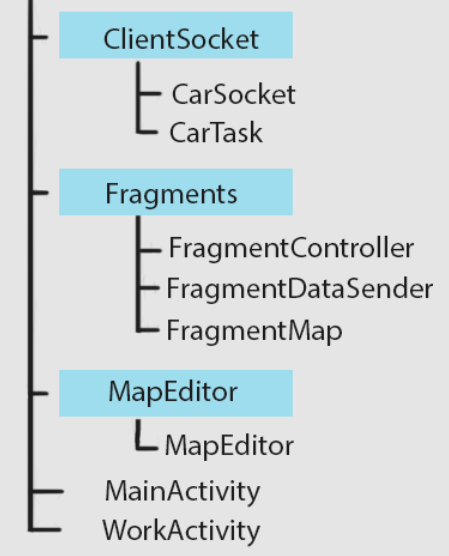


Рисунок 2.5 – Архитектура каталогов мобильного клиента-контроллера

Приложение имеет два основных окна: окно подключения (*MainActivity*), которое состоит из двух полей для ввода параметров подключения и кнопки «*connect*», и рабочее окно (*WorkActivity*), которое становится доступным после успешного подключения к серверу.

Как только пользователь ввёл все необходимые данные для подключения и нажал соответствующую кнопку, создаётся новый экземпляр асинхронного класса *CarTask*, который является наследником базового класса *AsynkTask*.

*Android* не позволяет выполнять масштабные операции в основном потоке, например, такие как: работа с сетью или операции с базой данных. Тяжёлые операции в основном потоке, который работает с интерфейсом, приведут к отказу отклика последнего. Это может негативно сказаться на пользовательском опыте. Избежать подобной ситуации можно используя потоки, например класс *Thread* или, имплементировав интерфейс *Runnable*, реализовать исполнение тяжёлого кода в методе *Run*.

Существует ограничение – параллельным потокам запрещён доступ к элементам управления. И если от результата исполнения тяжёлой операции зависит представление пользовательского интерфейса, уместней использовать класс, разработанный специально для этих нужд – *AsynkTask*. Он является абстрактным и требует собственной реализации от разработчика. Основными методами можно назвать: *doInBackground* (в нём необходимо реализовать код тяжёлой операции), *onPostExecute* и *onPreExecute* (методы, исполняющиеся до и после операции, в основном потоке, то есть имеющие доступ к интерфейсу пользователя). Так же существуют три реже используемых метода: *onProgressUpdate* (для промежуточного вывода результатов в основном потоке) и *onCanselled* (реализующий код отмены операции). Конструктор *AsynkTask* принимает на вход неограниченное число параметров заданного в объявлении класса типа. Также можно задать типы параметров, возвращаемых при завершении операции или при промежуточных выводах.

Выше описанные составляющие доказывают удобство использования методов данного класса для решения задач с сетевым соединением.

В классе *CarTask* создаётся сокет подключения, принимается сообщение от сервера, подтверждающее успешность подключения и возвращается результат операции в виде булевского значения.

Во время подключения в *MainActivity* создается сокет подключения, позволяющий обмениваться данными между устройствами. Класс отвечающий за сокетное соединение – *CarSoket*, находящийся в директории *ClientSoket*.

Задача передачи сокета между *Activity* решается паттерном SingleTone, используемом на классе *CarSoket*. Единственный экземпляр класса храниться в классе и доступ к нему зависит от модификатора доступа к классу. Такое решение заменяет глобальную переменную, а также ограничивает создание новых экземпляров класса без необходимости.

Класс *CarSoket*, помимо реализации *SingleTone*, имеет четыре основных рабочих метода: *connectToDevice*, *unconnectToDevice*, *sendMessage* и *getMessage*. Методы и поля класса *CarSoket* представлены на рисунке 2.6.

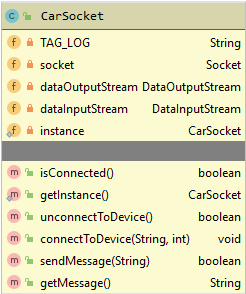


Рисунок 2.6 – Методы и поля класса *CarSocket*

В методе *connectToDevice* вызывается метод Connect у объекта сокета. Метод *unconnectToDevice* закрывает все открытые соединения, в том числе и те, что были использованы для передачи или получения данных. Метод *sendMessage* создаёт или использует уже имеющийся объект класса *DataOutputStream*, используемый для отправки сообщения на сервер. Метод *getMessage* создаёт или использует уже имеющийся объект класса *DataInputStream*, используемый для получения сообщения от сервера [6].

После успешного подключения к серверу происходит переход на *WorkActivity*, которая располагает четырьмя кнопками: «*unconnected*» «*controller*» «*data*\_*sender*» «*map*». Первая из списка кнопка отправляет на сервер числовое значение ноль, что является признаком завершения сеанса с пользователем. Клиентское же приложение снова переходит в окно подключения.

Остальные три кнопки отвечают за тип клиента, при их нажатии происходит отправка на сервер сообщения с номером типа клиента, а также создаётся и отображается соответствующий фрагмент. Для определения типа фрагмента использован фабричный метод [7], возвращающий необходимый класс, по числу. Для выполнения операций по отправки сообщений на сервер создан вложенный класс *ClientTypeTask*, наследующий *AsynkTask*. Он принимает входным значением число, идентифицирующее тип клиента. Отправляет его значение на сервер и, в случае успешной отправки, создаёт и открывает соответствующий фрагмент.

Приложение имеет три класса фрагментов: *FragmentController*, *FragmentDataSender* и *FragmentMap*. Операции, исполняемые в фрагменте *FragmentDataSender* в последствии были вынесены в отдельное приложение и описаны в подразделе 2.3.

*FragmentController* содержит кнопки, отвечающие за передвижение робота: «*Forward*», «*Backward*», «*Right*», «*Left*» и «*Stop*». А так же кнопки регулировки значений скорости и градуса поворота. Класс содержит вложенный асинхронный класс – *ControllerTask*. Его задачей является отправка на сервер строки, переданной в конструктор в качестве параметра.

Внутренняя архитектура класса устроена так, что при нажатии управляющей кнопки, создаётся новый объект класса *ControllerTask*. И, несмотря на то, что они пользуются одним и тем же объектом класса *DataOutputStream*, создание очередного нового потока может сказаться на производительности. Учитывая управление закрытием потоков самим *Android*.

Класс *FragmentMap* отвечает за работу с картой. Существует два варианта – это создать карту в предложенном редакторе или загрузить изображение формата *bmp* из галереи телефона. Задача данного класса – расшифровать путь, указанный на карте, и отправить последовательность команд на сервер для дальнейшего их воспроизведения.

Редактор карты состоит из выстроенной, по указанным пользователем размерам, белой сетки. Каждый элемент которой является пустым элементом *TextView*, сгенерированным в цикле программного кода.

Для примера взяты размеры шесть на шесть пикселей. Каждый, из которых, имеет свой цвет. На рисунке 2.7 чёрным обозначен примитивный путь, зелёным – точка старта, а красным – присутствие датчика на этом участке. Путь должен быть замыкающим. Так как поля больше подобны квадрату, нежели вытянуты в линию, логичней строить маршрут, начиная в одной стороны и замыкая в том же месте, но с другой. В отличии от реализации возвращения на базу путём обхода в другом направлении уже пройденного пути.

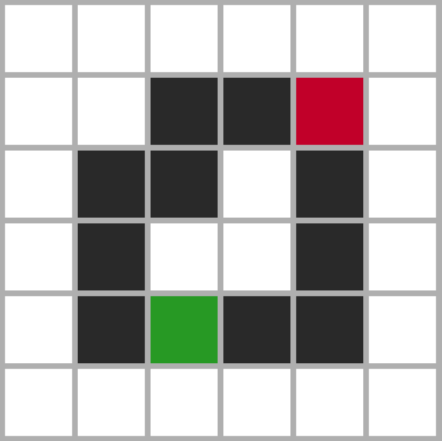


Рисунок 2.7 – Элементарный путь на карте

Задача извлечения последовательности команд, таких как: «*f*», «*rr*», «*rl*» и «*s*», решается проверкой нескольких условий. Так как карта представляется двоичным массивом и по ней будет осуществляться поиск, разумней представить её элементы в виде цифр и записать в примитивный тип или же использовать класс *ArrayList* для хранения объектов *TextView*, который подразумевает упорядоченное хранение объектов в памяти, что ускорит операции поиска по индексу.

В начале алгоритма преобразования указатель находит место расположения на карте стартовой точки и принимает значение его координат. Начало движения зафиксировано и начинается справа. Дальше следует проверка на прямолинейность, которая изображена на рисунке 2.8.

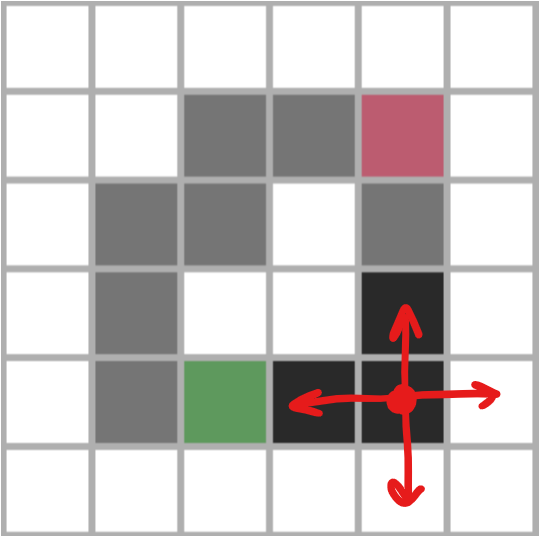
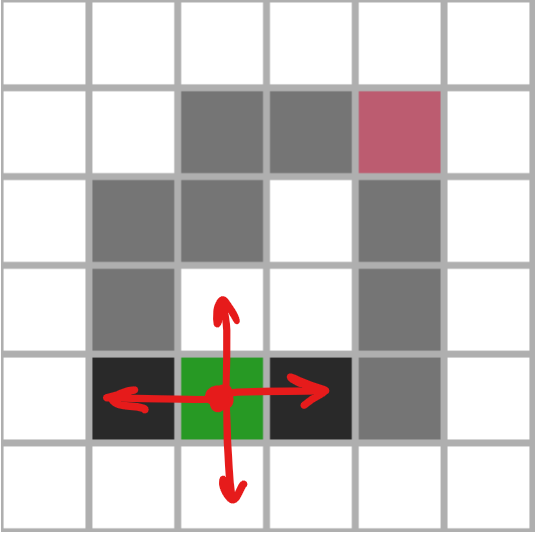


Рисунок 2.8 – Проверка на наличие угла на карте

Если с двух сторон текущего местоположения, либо вертикально, либо горизонтально, имеются цветные области – следующей командой будет передвижение вперёд. На углах эта проверка не сработает, что является показателем поворота. Алгоритм делает шаг вправо, то есть увеличивает индекс по столбцам на единицу, строковый индекс остаётся с тем же значением.

Сторона поворота находится, основываясь на направлении движения. Направление можно узнать, зная текущее местоположение и местоположение, которое было шаг назад. Например, на рисунке 2.9 указан путь и два направления движений по нему. В первой части рисунка каждый из обозначенных поворотов будет поворотом налево, а во второй – поворотом направо. Несмотря на то, что внешнее представление улов (справа на рисунке 2.9) абсолютно одинаково в обоих случаях и абсолютно разное, если рассматривать относительно одного случая.

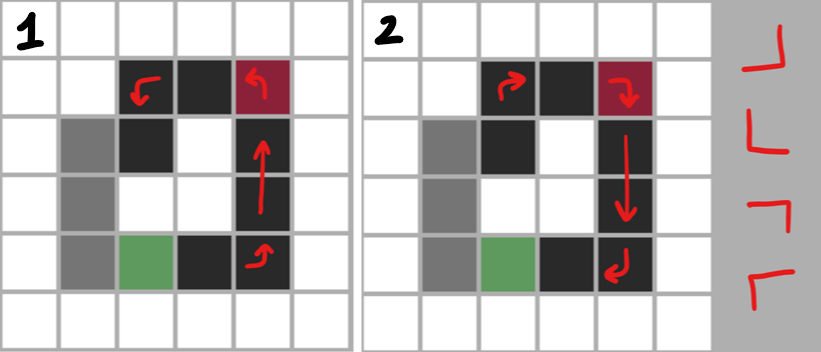


Рисунок 2.9 – Распознавание стороны поворота на карте

Алгоритм не предусматривает обработку границ, что обязывает пользователя оставлять их пустыми.

Завершающей частью алгоритма является проверка конкретного значения цветной клетки. Если цвет красный или зелёный, то помимо основной команды с движением добавляется команда стоп, символизирующая остановку. Со стороны сервера это означает наличие клиента в данном месте.

Все команды записываются в массив строк, но отправляются на сервер в виде единой строки. Там происходит их обратное преобразование в массив и добавление каждой необходимого числового значения. Как правило, это градус поворота для углов и секунды исполнения движущих операций.

* 1. **Клиент-серверное приложение – клиент-датчик**

Для имитации реальных условий необходимо приложение, исполняющее роль сельскохозяйственного датчика. В его задачи входит отправление по заданному заранее *IP* и портуимитированных данных в определённый момент времени.

Данный вид клиента был вынесен в отдельное приложение для удобства пользователей и логического разделения.

Расположения датчиков в пространстве строго зафиксированы для каждой карты. У каждого из датчиков установлен личный оригинальный идентификатор. Данные с датчика передаются в виде «*id;date;data*». Так как сокеты передают информацию в байтовом виде, вышеупомянутая строка преобразуется в массив байт. А затем, сервер преобразует их обратно в строку. После, регулярные выражения получают необходимые части строки, из которых создаётся новый объект данных.

Существует более удобный (в достаточных масштабах класса) способ передачи информации – сериализация и десериализация. Принцип работы, которой заключается в записывании полной структуры класса в последовательность битов. Для десериализации необходимым условием является наличие идентичного класса на принимающей стороне. Выполнение условия может оказаться затруднительным, если сервер и клиент написаны на разных языках.

Приложение датчика состоит из двух основных экранов – экран подключения, и рабочий экран. Экран подключения принимает данные: *IP,* номер порта, идентификатор датчика, и передаваемые данные. Рабочий экран устанавливает таймер, по истечении которого будет совершена попытка подключения и передачи данных.

Для удобства пользователя приложение датчика отображает результат отправки данных, изменяя цвет фона приложения (красный – не удалось отправить, зелёный – отправка произведена успешно).

Внутренняя архитектура приложения схожа с приложением-контроллером. Отличие состоит в упрощённой функциональности и наличии отложенного подключения по таймеру.

Схему взаимодействия классов мобильного приложения-датчика можно увидеть в приложении Б.

## 3 ОПЫТНАЯ ЭКСПЛУАТАЦИЯ И ВЕРИФИКАЦИЯ

* 1. **Обзор интерфейса пользователя мобильных приложений**

Сервер начинает работу с отображения в консольном окне *IP* адреса, на котором он запущен, и порта. Прослушивает сервер любые подключения из вне.

На рисунке 3.1 представлены снимки экрана мобильного приложения «*CarClient*», являющегося представителем первого и третьего типа клиентов.

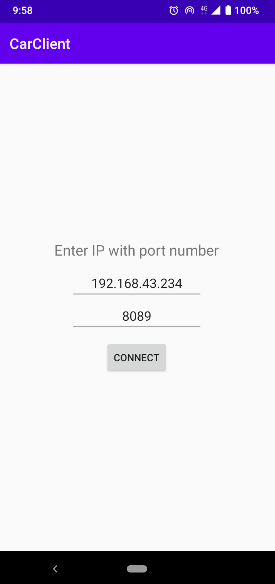
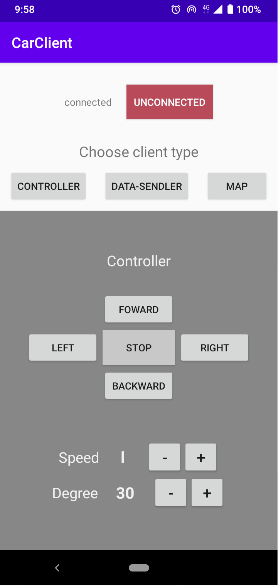
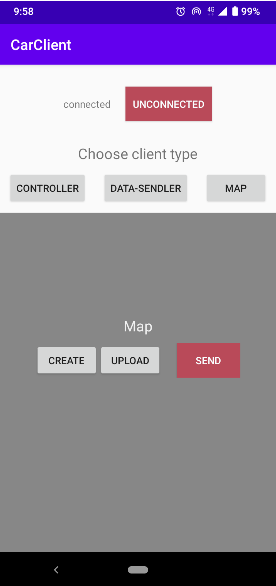
  

Рисунок 3.1 – Основные рабочие окна приложения-контроллера

Слева изображено окно подключения. Если подключение прошло успешно экран перейдёт в рабочую область, часть которой выделена под отображение фрагментов и является пустой до тех пор, пока не будет выбран тип клиента, нажатием одной из трёх серых клавиш. Кнопка «*unconnected*» символизирует отключение клиента. Если тип клиента не был выбран, но кнопка «*unconnected*» нажата – сервер закончит работу с безымянным клиентом и перейдёт в режим прослушивания новых подключений.

После выбора типа клиента, на сервер будет отправлено сообщение с его указанием, а приложение откроет один из двух фрагментов.

Клик по кнопке «*Controller*» (рисунок 3.1 центральная часть) приводит к отображению фрагмента с джойстиком из кнопок. Также есть возможность настройки параметров скорости и градуса поворота.

Клик по кнопке «*Map*» (рисунок 3.1 справа) открывает панель работы с картой. Здесь пользователь имеет выбор – загрузить готовое изображение в формате *bmp* из галереи или создать его во встроенном редакторе. Нажатие на кнопку «*Send*» преобразует путь на карте в последовательность команд и отправляет в виде строки на сервер, где происходит дальнейшая обработка.

На рисунке 3.2 представлена работа с редактором карты. После нажатия на кнопку «*Create*» станет видимой панель, где пользователю будет необходимо ввести размеры будущей карты и нажать кнопку «*Ok*», после чего панель снова станет скрытой, а в нижней области будет сгенерировано поле для рисования карты.

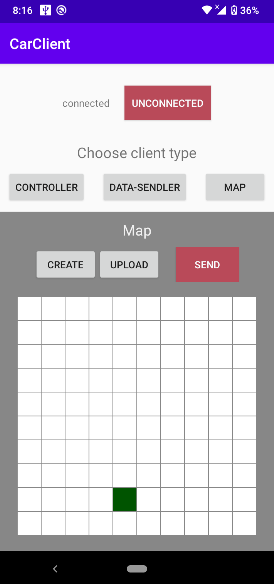
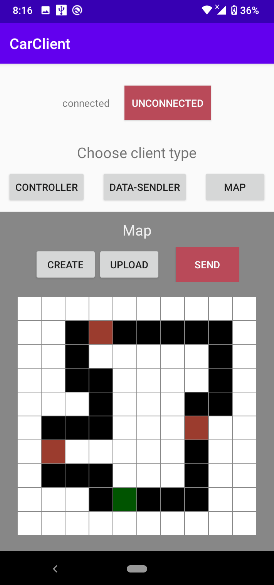
 

Рисунок 3.2 – Работа с редактором карты

На рисунке 3.2 слева представлена пустая область с заранее заданным расположением точки старта, а справа изображена готовая к отправке на сервер карта. При нажатии на белый квадрат, его цвет меняется на чёрный. При долгом нажатии на него же – цвет становится красным, что символизирует наличие датчика в данной области. Путь, указанный на карте, должен быть замкнут, не содержать пересечений, двойных линий, незавершённых тупиков. А также не должен находиться на границах карты. В противном случае путь может быть распознан неверно либо быть нераспознанным вовсе.

На изображении 3.3 представлены основные окна приложения «*DataSender*», которое представляет собой второй тип клиента.

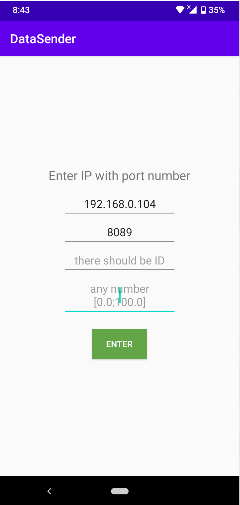
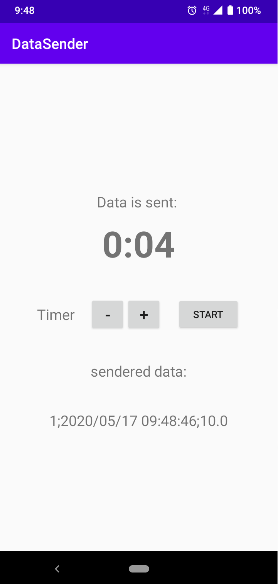
 

Рисунок 3.3 – Интерфейсы приложения-датчика

Слева на рисунке 3.3 представлен снимок начального экрана. Он требует ввода *IP* и порта подключения, а также идентификационного номера конкретного датчика (*id*) и данных, что будут отправлены. В итоге приложение отправляет три значения: id датчика, собравшего данные; сами данные; дата, когда эти данные были получены.

Нажатие кнопки «Enter» позволяет перейти в рабочее окно с таймером (рисунок 3.3 справа). Пользователь устанавливает значение таймера используя предоставленные кнопки, после нажатия кнопки «*Start*» происходит запуск таймера. Как только установленное время выйдет, приложение совершит попытку подключения к серверу и передачи данных. Если она будет удачной и данные будут отправлены на сервер – фон рабочей области сменит цвет на зелёный, если попытка подключения окончится провалом – фон рабочей области станет красным. Решение наглядного отображения результата работы датчика связано с удобством тестирования в реальных условиях.

Все значимые процессы, происходящие в клиентских приложениях залогированы. Пример логов при подключении к серверу и отправке типа клиента представлен на рисунке 3.4. Ни изображении 3.5 представлены логи, выводящиеся при завершении работы с сервером.

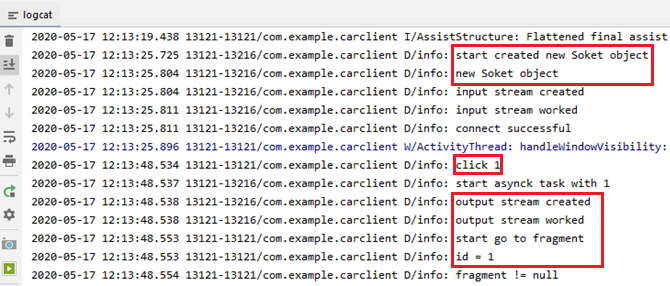


Рисунок 3.4 – Вывод логов при подключении к серверу

Для точного отображения происходящих операций, логи представлены до и после создания объекта сокета. Так же регистрируется создание и работа объектов классов, ответственных за отправление и приём сообщений.

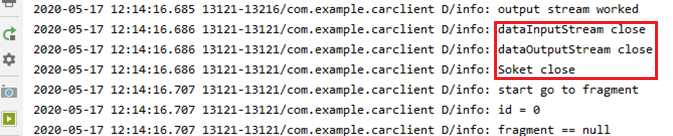


Рисунок 3.5 – Вывод логов при отключении от сервера

Логирование помогает отследить места происхождения ошибок или предотвратить их возможное появление. А также наладить стабильность и правильность в работе приложения.

* 1. **Тестирование работы серверного программного обеспечения**

Со стороны сервера запуск и подключение клиентов отслеживается и отображается консольно.

После запуска сервер выводит строку с *IP* и номером порта, которые необходимо ввести клиентам для подключения. Дальше начинает прослушивать входящие подключения. Как только один из клиентов подключается, сервер выводит в консоль номер подключённого клиента и его *IP*. Номер подключения – это число отображающее общее количество подключённых клиентов, которые принял сервер за время работы. Оно ограничено и по достижении максимального значения сервер завершает свою работу.

На рисунке 3.6 выделен вывод *IP* сервера и поступающих команд от первого типа клиента (клиента-контроллера).

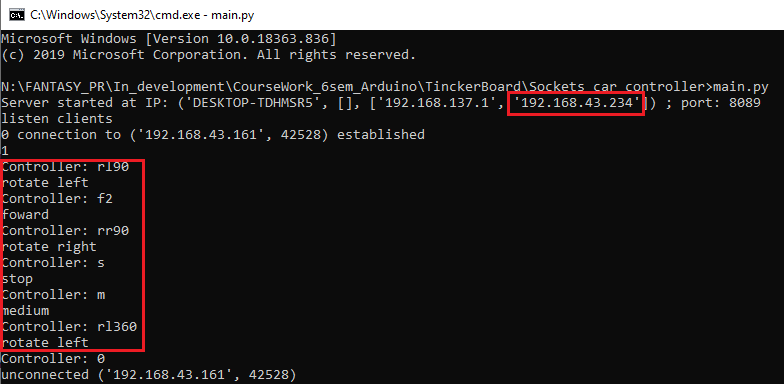


Рисунок 3.6 – Консольное отображение запуска сервера и работы с первым

типом клиента

Подключение и отключение клиента также регистрируются выводом на консоль информации о сокете клиента. После подключения следует цифра, обозначающая *id* клиента. На рисунке выше за цифрой выводятся команды, отправляемые контроллером, после которых происходит соответствующее движение робота.

Завершение работы символизируется значением ноль, после получения которого сервер прекращает работу с данным клиентом.

На рисунке 3.7 представлена работа с клиентом второго типа (клиентом-датчиком). После успешного получения данных, их строковое представление выводится в консоль, а соединение с клиентом разрывается.

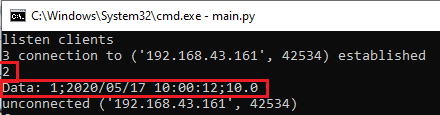


Рисунок 3.7 – Работа сервера со вторым типом клиента

На рисунке 3.8 отражена работа с клиентом третьего типа (клиентом-картой).

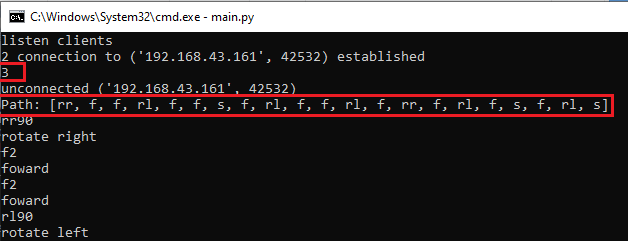


Рисунок 3.8 – Работа сервера с третьим типом клиента

Клиент отправляет путь передвижения робота в строковом виде. Путь для верификации выводится в консоль.

Следующим шагом сервера является проигрывание очерёдности команд, записанных через запятую в представлении. Сервер устанавливает свои значения для каждого из числовых параметров и выводит на экран название команды, а также её расшифровку, что символизирует исполнение метода соответствующего команде.

Интервал воспроизведения очередной команды выставлен в четыре секунды. Это время рассчитано на исполнение роботом предыдущей команды. Если команда символизирует наличие датчика, сервер запускает метод ожидания подключений клиентов. При этом консольный вывод сводится к изображению на рисунке 3.7, с учётом подключения клиента второго типа.

## ЗАКЛЮЧЕНИЕ

При выполнении поставленной задачи первоначально был проведен аналитический обзор технологий, в области разработки роботизированных устройств с использованием одноплатных компьютеров. Аналитическим образом выявлены преимущества одноплатного компьютера *Tinker* *Board* в решении данной задачи. Также решение разработки управляющих приложений на мобильных устройствах даёт неоспоримое преимущество в гибкости и переносимости системы программного обеспечения. Подобные решения наиболее успешны в эксплуатировании на открытой местности, на которой и будет работать робот и прилегающие к нему приложения.

Языком программирования на стороне *Tinker* *Board* выбран *python*, так как он довольно компактен, что сокращает количество кода, а следовательно и время затраченное на его написание. А так же кроссплатформенность языка *python* даёт возможность отлаживать и разрабатывать приложение на ПК, а затем испытывать непосредственно на одноплатном компьютере.

Для разработки мобильных приложений была выбрана платформа *Android*, как наиболее распространённая. А языком программирования – *java*. Несомненно, молодой и перспективный язык *kotlin* имеет свои преимущества. Однако для разработки сложного ПО, требующего обязательной отладки и обработки ошибок – язык *java* подходит больше.

Результатом разработки стали два приложения и колёсный робот.

Приложения и робот предназначены для работ в сельскохозяйственном поле, преимущественно для сбора информации с датчиков, расположенных на местности.

Робот и прилагаемое к нему программное обеспечение призваны облегчить труд фермерам и людям работающим в сельском хозяйстве. Они позволят автоматизировать человеческий труд, улучшат количество и качество исполняемой работы.

К особым достоинствам разработанной системы можно отнести продуманность пользовательского взаимодействия, а также интуитивно понятный интерфейс, требующий минимального количества нажатий для старта работы системы. Недостатками же является не застрахованность робота от возможных физических повреждений, потере собранных данных при аварийном завершении работы сервера. Также значительной проблемой может стать отклонение робота от заданной траектории во время обхода. Но она может быть устранена добавлением корректировки движений, посредством отслеживания местоположения робота системой *GPS*.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Сравнение 98 одноплатных компьютеров [Электронный ресурс] – Режим доступа: [*http://linuxgizmos.com/catalog-of-98-open-spec-hacker-friendly-sbcs/*](http://linuxgizmos.com/catalog-of-98-open-spec-hacker-friendly-sbcs/)– Дата доступа 17.05.2020.
2. Отправка данных с радио датчика на телефон Android с использованием технологии *Bluetooth* *Low* *Energy* [Электронный ресурс] – Режим доступа: [*http://digitrode.ru/computing-devices/mcu\_cpu/2274-otpravka-dannyh-s-datchika-na-telefon-android-s-ispolzovaniem-arduino-i-nrf24l01-cherez-bluetooth-ble.html*](http://digitrode.ru/computing-devices/mcu_cpu/2274-otpravka-dannyh-s-datchika-na-telefon-android-s-ispolzovaniem-arduino-i-nrf24l01-cherez-bluetooth-ble.html) – Дата доступа 17.05.2020.
3. Основная разница между сервоприводом и шаговым двигателем [Электронный ресурс] – Режим доступа: [*https://arduinoplus.ru/otlichia-servoprivoda-ot-shagovogo-dvigatelya/*](https://arduinoplus.ru/otlichia-servoprivoda-ot-shagovogo-dvigatelya/) – Дата доступа 17.05.2020.
4. Приемы объектно-ориентированного программирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб: Питер, 2001. – 368 с.
5. *Python Cookbook / David B., Brian K. J. – Sebastopol: O’Reilly Media,* 2013. – 441 c.
6. *Java*. Полное руководство, 10-е изд. : Пер. с англ. -СПб. ООО "Альфакнига'; 2018. - 1488 с. : ил. - Парал. тит. англ. – 907 c.
7. Фабричный метод [Электронный ресурс] – Режим доступа: *https://metanit.com/sharp/patterns/2.1.php* – Дата доступа: 02.12.2019.

## ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программы**

**Код *CarClient/MainActivity.java***

**package** com.example.carclient;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.content.Intent;  
**import** android.os.Bundle;  
  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.Toast;  
  
**import** com.example.carclient.ClientSocket.CarTask;  
**import** java.util.concurrent.ExecutionException;  
  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 **private** Button **b\_enter**;  
 **private** EditText **port**;  
 **private** EditText **ip**;  
  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 **b\_enter** = findViewById(R.id.***b\_connect***);  
 **port** = findViewById(R.id.***car\_port***);  
 **ip** = findViewById(R.id.***car\_ip***);  
  
  
 **b\_enter**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 String p = **port**.getText().toString();  
 String ip\_str = **ip**.getText().toString();  
  
 CarTask carTask = **new** CarTask();  
 carTask.execute(ip\_str, p);  
  
 **boolean** connected = **false**;  
 **try** {  
 *//изза ожидания skipped >100 frames* connected = carTask.get();  
  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 Toast.*makeText*(getApplicationContext(),  
 String.*valueOf*(connected) ,Toast.***LENGTH\_SHORT***).show();  
  
 **if**(connected){  
 Intent intent2 = **new** Intent(getApplicationContext(), WorkActivity.**class**);  
 startActivity(intent2);  
 }  
  
 }  
 });  
  
 }  
  
}

**Код *CarClient/WorkActivity.java***

**package** com.example.carclient;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.fragment.app.Fragment;  
**import** androidx.fragment.app.FragmentManager;  
  
**import** android.content.Intent;  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.example.carclient.ClientSocket.CarSocket;  
**import** com.example.carclient.fragments.FragmentController;  
**import** com.example.carclient.fragments.FragmentDataSender;  
**import** com.example.carclient.fragments.FragmentMap;  
  
**import** java.util.concurrent.ExecutionException;  
  
**public class** WorkActivity **extends** AppCompatActivity {  
  
 **private** Button **b\_controller**;  
 **private** Button **b\_map**;  
 **private** Button **b\_data\_sender**;  
  
 **private** Button **b\_unconnected**;  
  
 **protected** TextView **tv\_connected**;  
  
 **private** CarSocket **carSocket**;  
*// private ClientTypeTask clientTypeTask;* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_work***);  
  
 **tv\_connected** = findViewById(R.id.***tv\_connected***);  
  
 **b\_unconnected** = findViewById(R.id.***b\_unconnected***);  
 **b\_controller** = findViewById(R.id.***b\_controller***);  
 **b\_map** = findViewById(R.id.***b\_map***);  
 **b\_data\_sender** = findViewById(R.id.***b\_data\_sender***);  
  
 **carSocket** = CarSocket.*getInstance*();  
*// clientTypeTask = new ClientTypeTask();* **b\_unconnected**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 ClientTypeTask c = **new** ClientTypeTask();  
 c.execute(0);  
  
 **boolean** b = **false**;  
 **try** {  
 b = c.get();  
  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **if**(b){  
 **if**(**carSocket**.unconnectToDevice()){  
 **tv\_connected**.setText(**"unconnected"**);  
 Intent intent = **new** Intent(getApplicationContext(), MainActivity.**class**);  
 startActivity(intent);  
 Toast.*makeText*(getApplicationContext(),**"unconnected"** ,Toast.***LENGTH\_SHORT***).show();  
  
 }**else**{  
 Toast.*makeText*(getApplicationContext(),**"can't unconnected"** ,Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 }  
 });  
  
  
 *//прежде чем переходить на другой тип клиента необходимо убедиться- закончил ли работу другой тип  
 //после того как клиент выполняет свою задачу- он разрывает соединение* **b\_controller**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *//отправляю серверу необходимое сообщение, указывая на тип подключаемого клиента* **if**(**tv\_connected**.getText().equals(**"unconnected"**)){  
 Log.*d*(**"info"**, **"click 1"**);  
 **tv\_connected**.setText(**"connected"**);  
 **new** ClientTypeTask().execute(1);  
 }  
 }  
 });  
  
 **b\_data\_sender**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**tv\_connected**.getText().equals(**"unconnected"**)){  
 **tv\_connected**.setText(**"connected"**);  
 **new** ClientTypeTask().execute(2);  
 }  
 }  
 });  
  
 **b\_map**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(tv\_connected.getText().equals(**"unconnected"**)){  
 tv\_connected.setText(**"connected"**);  
 **new** ClientTypeTask().execute(3);  
 }  
 }  
 });  
  
 }  
  
  
  
 *//единственная задача-отправить серверу сообщение о том, какой тип контроллера выбран  
 //и открывать соответствующий фрагмент* **class** ClientTypeTask **extends** AsyncTask<Integer, Void, Boolean> {  
  
 **private int id**;  
  
 *//маленький фабричный метод* **private** Fragment getFragment(**int** id){  
 **switch** (id){  
 **case** 0: **return null**;  
 **case** 1: **return new** FragmentController();  
 **case** 2: **return new** FragmentDataSender();  
 **case** 3: **return new** FragmentMap();  
 }  
 **return null**;  
 }  
  
  
 @Override  
 **protected** Boolean doInBackground(Integer ... params) {  
 Log.*d*(**"info"**, **"start asynck task with 1"**);  
 **try** {  
  
 **if**(**carSocket**.sendMessage(params[0].toString())) {  
 **id** = params[0];  
 **return true**;  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return false**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Boolean result) {  
 **super**.onPostExecute(result);  
  
 **if**(result){  
 *//подключаю нужный фрагмент* Log.*d*(**"info"**, **"start go to fragment"**);  
 Log.*d*(**"info"**, **"id = "**.concat(String.*valueOf*(**id**)));  
 Fragment dv = getFragment(**id**);  
 **if**(dv !=**null**){  
 Log.*d*(**"info"**, **"fragment != null"**);  
 FragmentManager fragmentManager = getSupportFragmentManager();  
 fragmentManager.beginTransaction()  
 .replace(R.id.fragment\_data\_list, dv).commit();  
 }**else** {  
 Log.d(**"info"**, **"fragment == null"**);  
 }  
  
 }**else** {  
 Toast.makeText(getApplicationContext(),**"can't send message to server"** ,Toast.LENGTH\_LONG).show();  
 }  
  
 }  
  
 }  
  
}

**Код *CarClient/ClientSocket/CarSocket.java***

**package** com.example.carclient.ClientSocket;  
  
**import** android.util.Log;  
  
**import** java.io.BufferedInputStream;  
**import** java.io.BufferedOutputStream;  
**import** java.io.DataInputStream;  
**import** java.io.DataOutputStream;  
**import** java.io.IOException;  
**import** java.net.InetSocketAddress;  
**import** java.net.Socket;  
**import** java.net.UnknownHostException;  
  
*//singletone***public class** CarSocket {  
  
 **private** String **TAG\_LOG** = **"info"**;  
  
 **private** Socket **socket**;  
 **private** DataOutputStream **dataOutputStream** = **null**;  
 **private** DataInputStream **dataInputStream** = **null**;  
  
 **public boolean** isConnected(){  
 **return socket**.isConnected();  
 }  
  
 **private** CarSocket(){  
 **socket** = **new** Socket();  
 }  
  
 **private static** CarSocket *instance*;  
  
 **public static** CarSocket getInstance(){  
 **return** (*instance* == **null**) ? *instance* = **new** CarSocket() : *instance*;  
 }  
  
  
 **public boolean** unconnectToDevice()  
 {  
 **try** {  
 **if**(**dataInputStream** != **null**) **dataInputStream**.close();  
 Log.*d*(**TAG\_LOG**, **"dataInputStream close"**);  
 **if**(**dataOutputStream** != **null**) **dataOutputStream**.close();  
 Log.*d*(**TAG\_LOG**, **"dataOutputStream close"**);  
 **if**(**socket** != **null**) **socket**.close();  
 Log.*d*(**TAG\_LOG**, **"Soket close"**);  
  
 *instance* = **null**;  
 **return true**;  
  
 }  
 **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 **return false**;  
  
 }  
  
 **public void** connectToDevice(String ip, **int** port)  
 {  
 **try** {  
 Log.*d*(**TAG\_LOG**, **"start created new Soket object"**);  
 **socket**.connect(**new** InetSocketAddress(ip,port));  
 Log.*d*(**TAG\_LOG**, **"new Soket object"**);  
  
 }  
 **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 *instance* = **null**;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 *instance* = **null**;  
 }  
  
 }  
  
 **public boolean** sendMessage(String str)  
 {  
 **try** {  
 **if**(**dataOutputStream** == **null**){  
 **dataOutputStream** = **new** DataOutputStream(**new** BufferedOutputStream(**socket**.getOutputStream()));  
 Log.*d*(**TAG\_LOG**, **"output stream created"**);  
 }  
 **dataOutputStream**.write(str.getBytes());  
*// dataOutputStream.writeUTF(str);* **dataOutputStream**.flush();  
  
  
*// printWriter = new PrintWriter(  
// new OutputStreamWriter(socket.getOutputStream()));  
// printWriter.print(str);  
// printWriter.flush();  
// printWriter.close();* Log.*d*(**TAG\_LOG**, **"output stream worked"**);  
 **return true**;  
  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 **return false**;  
 }  
  
 **public** String getMessage()  
 {  
 String s = **null**;  
  
 **try** {  
 **if**(**dataInputStream** == **null**){  
 **dataInputStream** = **new** DataInputStream(**new** BufferedInputStream(**socket**.getInputStream()));  
 Log.*d*(**TAG\_LOG**, **"input stream created"**);  
 }  
  
 **byte**[] b = **new byte**[13];  
 **int** i = **dataInputStream**.read(b);  
  
 **if**(i > 0){  
 s = **new** String(b);  
 }  
  
 Log.*d*(**TAG\_LOG**, **"input stream worked"**);  
 }  
 **catch** (NullPointerException e){  
 e.printStackTrace();  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 **return** s;  
 }  
  
  
  
}

**Код *CarClient/ClientSocket/CarTask.java***

**package** com.example.carclient.ClientSocket;  
  
**import** android.os.AsyncTask;  
**import** android.util.Log;  
  
**public class** CarTask **extends** AsyncTask<String, Void, Boolean> {  
  
 @Override  
 **protected** Boolean doInBackground(String ... params) {  
 **try** {  
 CarSocket carSocket = CarSocket.*getInstance*();  
 carSocket.connectToDevice(params[0], Integer.*parseInt*(params[1]));  
  
 **if**(carSocket.isConnected()){  
 String messageFromServer = carSocket.getMessage();  
  
 **if**(messageFromServer != **null**){  
 Log.*d*(**"info"**, **"connect successful"**);  
  
 **return true**;  
 }  
  
 }**else** {  
 Log.*d*(**"info"**, **"Can't connected"**);  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return false**;  
 }  
  
}

**Код *CarClient/Fragments/FragmentController.java***

**package** com.example.carclient.fragments;  
  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
  
**import** androidx.fragment.app.Fragment;  
  
**import** android.util.Log;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.example.carclient.ClientSocket.CarSocket;  
**import** com.example.carclient.R;  
  
**public class** FragmentController **extends** Fragment {  
  
*// private Button b\_controller\_exit;* **private** Button **b\_stop**;  
 **private** Button **b\_forward**;  
 **private** Button **b\_backward**;  
 **private** Button **b\_left**;  
 **private** Button **b\_right**;  
  
 **private** Button **b\_speed\_deny**;  
 **private** Button **b\_speed\_add**;  
  
 **private** Button **b\_degree\_deny**;  
 **private** Button **b\_degree\_add**;  
  
 **private** TextView **tv\_speed**;  
 **private** TextView **tv\_degree**;  
  
 **private** CarSocket **carSocket**;  
 **private** ControllerTask **controllerTask**;  
  
 **private** String[] **speeds**;  
 **private int speedID**;  
  
 **private** String[] **degrees**;  
 **private int degreeID**;  
 **private int time**;  
  
 **private boolean sended** = **true**;  
  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
  
 View v = inflater.inflate(R.layout.***fragment\_controller***, container, **false**);  
  
 **carSocket** = CarSocket.*getInstance*();  
 **controllerTask** = **new** ControllerTask();  
  
 **speeds** = **new** String[] {**"l"**, **"m"**, **"h"**};  
 **degrees** = **new** String[] {**"30"**, **"90"**, **"180"**, **"360"**};  
 **degreeID** = 1;  
 **time** = 2;  
  
  
 *//инициализируем кнопки  
// b\_controller\_exit = v.findViewById(R.id.b\_controller\_exit);* **b\_stop** = v.findViewById(R.id.***b\_stop***);  
 **b\_forward** = v.findViewById(R.id.***b\_forward***);  
 **b\_backward** = v.findViewById(R.id.***b\_backward***);  
 **b\_left** = v.findViewById(R.id.***b\_left***);  
 **b\_right** = v.findViewById(R.id.***b\_right***);  
  
  
 **b\_speed\_deny** = v.findViewById(R.id.***b\_speed\_deny***);  
 **b\_speed\_add** = v.findViewById(R.id.***b\_speed\_add***);  
  
 **b\_degree\_deny** = v.findViewById(R.id.***b\_degree\_deny***);  
 **b\_degree\_add** = v.findViewById(R.id.***b\_degree\_add***);  
  
 **tv\_speed** = v.findViewById(R.id.***tv\_speed***);  
 **tv\_degree** = v.findViewById(R.id.***tv\_degree***);  
  
  
 **tv\_speed**.setText(**speeds**[0]);  
 **tv\_degree**.setText(**degrees**[0]);  
  
 *//обработчики нажатий  
 //на поворотах градусы  
 //на направлении время  
  
// b\_controller\_exit.setOnClickListener(new View.OnClickListener() {  
// @Override  
// public void onClick(View v) {  
// new ControllerTask().execute("0", "");  
// Log.d("info", "exit");  
// }  
// });* **b\_stop**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**){  
 **sended** = **false**;  
 **new** ControllerTask().execute(**"s"**, **""**);  
 }  
 }  
 });  
  
 **b\_forward**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**) {  
 **sended** = **false**;  
 ControllerTask c = **new** ControllerTask();  
 c.execute(**"f"**, String.*valueOf*(**time**));  
 }  
*//  
// if(c.getStatus() != AsyncTask.Status.FINISHED){  
// Toast.makeText(getContext(),"asynctask status finishhed" ,Toast.LENGTH\_LONG).show();  
// }  
  
// controllerTask.doInBackground("f", String.valueOf(time));  
  
// MyThread myThread = new MyThread("f", String.valueOf(time));  
// myThread.start();* }  
 });  
  
 **b\_backward**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
*// controllerTask.doInBackground("b", String.valueOf(time));* **if**(**sended**) {  
 **sended** = **false**;  
 ControllerTask c = **new** ControllerTask();  
 c.execute(**"b"**, String.*valueOf*(**time**));  
 }  
  
*// MyThread myThread = new MyThread("b", String.valueOf(time));  
// myThread.start();* }  
 });  
  
 **b\_left**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**) {  
 **sended** = **false**;  
 **new** ControllerTask().execute(**"rl"**, String.*valueOf*(**degrees**[**degreeID**]));  
 }  
 }  
 });  
  
 **b\_right**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**) {  
 **sended** = **false**;  
 **new** ControllerTask().execute(**"rr"**, String.*valueOf*(**degrees**[**degreeID**]));  
 }  
 }  
 });  
  
 *//на кнопки параметров* **b\_speed\_deny**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**) {  
 **if**(**speedID** > 0) {  
 **sended** = **false**;  
 **speedID**--;  
 Log.*d*(**"info"**, **"speed --"**);  
 *//****todo перенести логику в метод setSpeedID* tv\_speed**.setText(**speeds**[**speedID**]);  
 **new** ControllerTask().execute(**speeds**[**speedID**], **""**);  
 }  
 }  
 }  
 });  
  
 **b\_speed\_add**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**) {  
 Log.*d*(**"info"**, **"speed.len "**.concat(String.*valueOf*(**speeds**.**length**)));  
 Log.*d*(**"info"**, **"speedID+1= "**.concat(String.*valueOf*(**speedID** + 1)));  
 **if**(**speeds**.**length** > **speedID**+1) {  
 **sended** = **false**;  
 **speedID**++;  
 Log.*d*(**"info"**, **"speed ++"**);  
 **tv\_speed**.setText(**speeds**[**speedID**]);  
 **new** ControllerTask().execute(**speeds**[**speedID**], **""**);  
 }  
 }  
 }  
 });  
  
 **b\_degree\_deny**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Log.*d*(**"info"**, **"degreeID= "**.concat(String.*valueOf*(**degreeID**)));  
 **if**(**degreeID** > 0){  
 **degreeID**--;  
 Log.*d*(**"info"**, **"degree --"**);  
 **tv\_degree**.setText(**degrees**[**degreeID**]);  
 }  
 }  
 });  
  
 **b\_degree\_add**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Log.*d*(**"info"**, **"degrees.len "**.concat(String.*valueOf*(**degrees**.**length**)));  
 Log.*d*(**"info"**, **"degreeID+1= "**.concat(String.*valueOf*(**degreeID** + 1)));  
 **if**(**degrees**.**length** > **degreeID**+1){  
 Log.*d*(**"info"**, **"degree ++"**);  
 **degreeID**++;  
 **tv\_degree**.setText(**degrees**[**degreeID**]);  
 }  
 }  
 });  
  
 **return** v;  
 }  
  
*// public class MyThread extends Thread {  
//  
// String str;  
// String ss;  
//  
// MyThread(String str, String ss){  
// this.str = str;  
// this.ss = ss;  
// }  
//  
// public void run() {  
// carSocket.sendMessage(str.concat(ss));  
// Log.d("info", "send ".concat(str));  
// }  
// }* **class** ControllerTask **extends** AsyncTask<String, Void, Void> {  
  
 @Override  
 **protected** Void doInBackground(String ... params) {  
 **try** {  
 **carSocket**.sendMessage(params[0].concat(params[1]));  
 Log.*d*(**"info"**, **"send "**.concat(params[0]));  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 Log.*d*(**"info"**, **"task ended"**);  
 **sended** = **true**;  
 }  
 }  
}

**Код *CarClient/Fragments/FragmentMap.java***

**package** com.example.carclient.fragments;  
  
**import** android.content.Intent;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.graphics.drawable.ColorDrawable;  
**import** android.net.Uri;  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
  
**import** androidx.fragment.app.Fragment;  
  
**import** android.os.Environment;  
**import** android.util.Log;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.ImageView;  
**import** android.widget.LinearLayout;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.example.carclient.ClientSocket.CarSocket;  
**import** com.example.carclient.R;  
**import** com.example.carclient.mapEditor.MapEditor;  
  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.io.InputStream;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
  
**public class** FragmentMap **extends** Fragment {  
  
 View **v**;  
 CarSocket **carSocket**;  
  
 Button **b\_size\_ok**;  
 EditText **et\_map\_x**;  
 EditText **et\_map\_y**;  
  
 Button **b\_create**;  
 Button **b\_send**;  
 Button **b\_upload**;  
 LinearLayout **ll\_map\_size**;  
 LinearLayout **ll\_map\_square**;  
  
 ArrayList<ArrayList<TextView>> **map** = **new** ArrayList<>();  
 **int x**;  
 **int y**;  
  
 **private int Pick\_image** = 1;  
  
  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 **v** = inflater.inflate(R.layout.***fragment\_map***, container, **false**);  
  
 **carSocket** = CarSocket.*getInstance*();  
  
 **b\_size\_ok** = **v**.findViewById(R.id.***b\_size\_ok***);  
 **et\_map\_x** = **v**.findViewById(R.id.***te\_size\_x***);  
 **et\_map\_y** = **v**.findViewById(R.id.***te\_size\_y***);  
  
 **b\_create** = **v**.findViewById(R.id.***b\_map\_create***);  
 **b\_upload** = **v**.findViewById(R.id.***b\_map\_upload***);  
 **b\_send** = **v**.findViewById(R.id.***b\_map\_send***);  
  
 **ll\_map\_size** = **v**.findViewById(R.id.***ll\_map\_size***);  
 *//основной лайаут по размещению карты* **ll\_map\_square** = **v**.findViewById(R.id.***ll\_map\_square***);  
  
 **b\_upload**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 File file = **new** File(Environment.  
 *getExternalStoragePublicDirectory*(Environment.*DIRECTORY\_DOWNLOADS*),**"a.bmp"**);  
 Bitmap bitmap = BitmapFactory.*decodeFile*(file.getAbsolutePath());  
 Log.*d*(**"log"**, String.*format*(**"bitmap size = %sx%s, byteCount = %s"**,  
 bitmap.getWidth(), bitmap.getHeight(),  
 (**int**) (bitmap.getByteCount() / 1024)));  
  
  
*// ByteArrayOutputStream stream = new ByteArrayOutputStream();  
// bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream);  
// byte[] bytes = stream.toByteArray();  
  
  
//  
// //Получаем URI изображения, преобразуем его в Bitmap  
// //объект и отображаем в элементе ImageView нашего интерфейса:  
// final Uri imageUri = intent.getData();  
// final InputStream imageStream = getContext().getContentResolver().openInputStream(imageUri);  
// Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);  
//  
// Toast.makeText(getContext(), "map uploaded" ,Toast.LENGTH\_SHORT).show();  
//  
// //переводим в массив байт  
//// ByteArrayOutputStream stream = new ByteArrayOutputStream();  
//// selectedImage.compress(Bitmap.CompressFormat.PNG, 100, stream);  
//// byte[] bytes = stream.toByteArray();* }  
  
 });  
  
 **b\_create**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **ll\_map\_size**.setVisibility(View.***VISIBLE***);  
 }  
 });  
  
 **b\_size\_ok**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 **x** = Integer.*parseInt*(**et\_map\_x**.getText().toString());  
 **y** = Integer.*parseInt*(**et\_map\_y**.getText().toString());  
  
 **map** = CreateMap(**x**, **y**);  
  
 **ll\_map\_size**.setVisibility(View.***GONE***);  
  
 }  
 });  
  
  
 **b\_send**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *//переводим в стандартный вид и отправляем* **double**[][] d = TranslateToArray(**map**, **x**, **y**);  
  
 *//сделать метод расширяющий границы массива* String[] str = MapEditor.*TranslateArrayToPath*(d);  
  
 **if**(str != **null**){  
 Toast.*makeText*(getContext(), Arrays.*toString*(str),Toast.***LENGTH\_LONG***).show();  
 **new** MapTask().execute(Arrays.*toString*(str));  
 }  
 **else** {  
 Toast.*makeText*(getContext(), **"can't translate map"** ,Toast.***LENGTH\_SHORT***).show();  
 }  
  
 }  
 });  
  
 **return v**;  
  
 }  
  
 **private double**[][] TranslateToArray(ArrayList<ArrayList<TextView>> map, **int** x, **int** y){  
  
 **double**[][] map\_d = **new double**[x][y];  
  
 **for**(**int** i = 0; i< x; i++){  
 **for**(**int** j = 0; j< y; j++){  
  
 **if** (map.get(i).get(j).getBackground() **instanceof** ColorDrawable) {  
 ColorDrawable cd = (ColorDrawable) map.get(i).get(j).getBackground();  
 **int** colorCode = cd.getColor();  
  
 *//0xFFFFFFFF - белый* **if**(colorCode == 0xFFFFFFFF){  
 map\_d[i][j] = 0;  
 }  
 **else if**(colorCode == 0xFF000000){  
 map\_d[i][j] = 1;  
 }  
 **else if**(colorCode == 0xFF005400){  
 map\_d[i][j] = 0.1;  
 }  
 **else if**(colorCode == 0xFF9B3C2E){  
 map\_d[i][j] = 0.5;  
 }  
  
 *//очищение* map.get(i).get(j).setBackgroundColor(0xFFFFFFFF);  
  
 }  
  
 }  
 }  
  
 *//изменяем базовую кнопку* map.get(x-2).get((**int**)(y/2)-1).setBackgroundColor(0xFF005400);  
  
  
 **for** (**double**[] doubles : map\_d) {  
 System.***out***.println(Arrays.*toString*(doubles));  
 }  
  
 **return** map\_d;  
  
 }  
  
 **private** ArrayList<ArrayList<TextView>> CreateMap(**int** ii, **int** jj){  
  
 ArrayList<ArrayList<TextView>> map = **new** ArrayList<>();  
  
 **ll\_map\_square** = **v**.findViewById(R.id.***ll\_map\_square***);  
  
 **for**(**int** i = 0; i< ii; i++){  
  
 ArrayList<TextView> mm = **new** ArrayList<>();  
  
 LinearLayout ll\_map\_line = **new** LinearLayout(getContext());  
 ll\_map\_line.setOrientation(LinearLayout.***HORIZONTAL***);  
 ll\_map\_line.setLayoutParams(**new** LinearLayout.LayoutParams(  
 LinearLayout.LayoutParams.***WRAP\_CONTENT***,  
 LinearLayout.LayoutParams.***WRAP\_CONTENT***));  
  
 **for**(**int** j = 0; j< jj; j++){  
  
 LinearLayout.LayoutParams lParams = **new** LinearLayout.LayoutParams(  
 LinearLayout.LayoutParams.***WRAP\_CONTENT***,  
 LinearLayout.LayoutParams.***WRAP\_CONTENT***);  
 lParams.setMargins(1,1,1,1);  
  
 TextView tv = **new** TextView(getContext());  
 *//0xFFFFFFFF - белый* tv.setBackgroundColor(0xFFFFFFFF);  
 tv.setHeight(80- (ii\*2));  
 tv.setWidth(80- (ii\*2));  
  
 tv.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 v.setBackgroundColor(0xFF000000);  
  
 }  
 });  
 tv.setOnLongClickListener(**new** View.OnLongClickListener() {  
 @Override  
 **public boolean** onLongClick(View v) {  
 v.setBackgroundColor(0xFF9B3C2E);  
 **return true**;  
 }  
 });  
  
 ll\_map\_line.addView(tv, lParams);  
  
 mm.add(tv);  
  
 }  
  
 **ll\_map\_square**.addView(ll\_map\_line);  
  
 map.add(mm);  
  
 }  
  
 *//изменяем базовую кнопку* map.get(ii-2).get((**int**)(jj/2)-1).setBackgroundColor(0xFF005400);  
 map.get(ii-2).get((**int**)(jj/2)-1).setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 }  
 });  
 map.get(ii-2).get((**int**)(jj/2)-1).setOnLongClickListener(**new** View.OnLongClickListener() {  
 @Override  
 **public boolean** onLongClick(View v) {  
 **return false**;  
 }  
 });  
  
  
 **return** map;  
  
 }  
  
  
 **class** MapTask **extends** AsyncTask<String, Void, Void> {  
  
 @Override  
 **protected** Void doInBackground(String ... params) {  
 **try** {  
 **carSocket**.sendMessage(params[0]);  
 Log.*d*(**"info"**, **"send "** + params[0]);  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 Log.*d*(**"info"**, **"task ended"**);  
 Toast.*makeText*(getContext(), **"path sanded"** ,Toast.***LENGTH\_SHORT***).show();  
*// sended = true;* }  
 }  
  
}

**Код *DataSender/WorkActivity.java***

**package** com.example.datasender;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.graphics.drawable.ColorDrawable;  
**import** android.os.Bundle;  
**import** android.os.CountDownTimer;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.LinearLayout;  
**import** android.widget.TextView;  
  
**import** com.example.datasender.SocketPart.CarSocket;  
**import** com.example.datasender.DataSenderData.DataPart;  
**import** com.example.datasender.SocketPart.DataSenderTask;  
  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.concurrent.ExecutionException;  
  
**public class** WorkActivity **extends** AppCompatActivity {  
  
 LinearLayout **ll\_work**;  
  
 DataPart **dataPart**;  
  
 TextView **timer**;  
 TextView **status**;  
 TextView **sendered\_data**;  
  
 Button **b\_timer\_deny**;  
 Button **b\_timer\_add**;  
 Button **b\_timer\_enter**;  
  
 CountDownTimer **countDownTimer**;  
 **int time** = 5000;  
 **boolean timer\_start** = **false**;  
  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_work***);  
  
 **ll\_work** = findViewById(R.id.***ll\_work***);  
  
 **dataPart** = DataPart.*getInstance*();  
  
 **timer** = findViewById(R.id.***timer***);  
 **status** = findViewById(R.id.***tv\_status***);  
 **sendered\_data** = findViewById(R.id.***tv\_sendered\_data***);  
  
 **b\_timer\_add** = findViewById(R.id.***b\_timer\_add***);  
 **b\_timer\_deny** = findViewById(R.id.***b\_timer\_deny***);  
 **b\_timer\_enter** = findViewById(R.id.***b\_timer\_enter***);  
  
 **sendered\_data**.setText(**dataPart**.toString());  
  
 updateTimer();  
  
  
  
 **b\_timer\_enter**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(!**timer\_start**){  
 **timer\_start** = **true**;  
 **b\_timer\_enter**.setText(**"stop"**);  
 *//60 00 00 = 10 мин =>* **countDownTimer** = **new** CountDownTimer(**time**, 1000) {  
 **public void** onTick(**long** millisUntilFinished) {  
 **time** -= 1000;  
 updateTimer();  
 }  
 **public void** onFinish() {  
 **timer**.setText(**"0:0"**);  
  
 *//пытается подключиться и передать данные* ConnectToCar();  
 }  
 };  
  
 **countDownTimer**.start();  
  
 }**else** {  
 **countDownTimer**.cancel();  
 **timer\_start** = **false**;  
 **b\_timer\_enter**.setText(**"start"**);  
 }  
 }  
 });  
  
  
 **b\_timer\_add**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(!**timer\_start**){  
 **time** += 1000;  
 updateTimer();  
 }  
 }  
 });  
  
 **b\_timer\_deny**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(!**timer\_start**){  
 **time** -= 1000;  
 updateTimer();  
 }  
 }  
 });  
  
 }  
  
  
 **public void** updateTimer(){  
 **int** minutes = (**int**)**time** / 60000;  
 **int** seconds = (**int**)**time** % 60000 / 1000;  
  
 String timeText = minutes + **":"**;  
 **if**(seconds< 10){  
 timeText += 0;  
 }  
 timeText += seconds;  
  
 **timer**.setText(timeText);  
  
 }  
  
  
 **public void** ConnectToCar(){  
  
*// String name = getApplicationInfo().dataDir + "/data.dat";  
  
 //передаём данные в виде строки- без использования сериализации* DataSenderTask dataSenderTask = **new** DataSenderTask();  
 dataSenderTask.execute(CarSocket.*getInstance*().**ip**, CarSocket.*getInstance*().**port**, **dataPart**.toString()+**"]"**);  
  
  
 **boolean** connected = **false**;  
 **try** {  
 *//изза ожидания skipped >100 frames* connected = dataSenderTask.get();  
  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
*// Toast.makeText(getApplicationContext(),  
// String.valueOf(connected) ,Toast.LENGTH\_SHORT).show();* **status**.setText(String.*valueOf*(connected));  
 **if**(connected){  
 **ll\_work**.setBackgroundColor(0xFF82B21E);  
 }**else** {  
 **ll\_work**.setBackgroundColor(0xFF9B3C2E);  
 }  
 }  
}

**Код *DataSender/DataPart.java***

**package** com.example.datasender.DataSenderData;  
  
**import** androidx.annotation.NonNull;  
  
**import** java.io.Serializable;  
**import** java.text.DateFormat;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Calendar;  
**import** java.util.Date;  
  
**public class** DataPart **implements** Serializable {  
  
 **private static** DataPart *instance*;  
  
 **public static** DataPart getInstance(){  
 **return** (*instance* == **null**) ? *instance* = **new** DataPart() : *instance*;  
 }  
  
 **public int ID**;  
 **private double Data**;  
 **private** String **Time**;  
  
 **public void** setData(**int** id, **double** data){  
 **ID** = id;  
 **Data** = data;  
  
 DateFormat dateFormat = **new** SimpleDateFormat(**"yyyy/MM/dd HH:mm:ss"**);  
 Calendar cal = Calendar.*getInstance*();  
 **Time** = dateFormat.format(cal.getTime());  
 }  
  
 @NonNull  
 @Override  
 **public** String toString() {  
 **return ID** + **";"** + **Time** + **";"** + **Data**;  
 }  
}

**Код *DataSender/DataSenderTask.java***

**package** com.example.datasender.SocketPart;  
  
**import** android.os.AsyncTask;  
**import** android.os.Build;  
**import** android.util.Log;  
  
**import** androidx.annotation.RequiresApi;  
  
**public class** DataSenderTask **extends** AsyncTask<String, Void, Boolean> {  
  
 @RequiresApi(api = Build.VERSION\_CODES.***KITKAT***)  
 @Override  
 **protected** Boolean doInBackground(String ... params) {  
 **try** {  
 CarSocket carSocket = CarSocket.*getInstance*();  
 Log.*d*(**"info"**, **"start connection"**);  
 carSocket.connectToDevice(params[0], Integer.*parseInt*(params[1]));  
  
 **if**(carSocket.isConnected()){  
 String messageFromServer = carSocket.getMessage();  
  
 **if**(messageFromServer != **null**){  
 Log.*d*(**"info"**, **"connect successful"**);  
  
 **if**(carSocket.sendMessage(**"2"**)) {  
 *//отправляем сериализованные данные* Thread.*sleep*(500);  
 carSocket.sendMessage(params[2]);  
  
*// Log.d("info", "start serialize operation");  
// SerializeObject.Serialize( params[2], DataPart.getInstance());  
// Log.d("info", "start sended file operation");  
// Boolean b = carSocket.sendObject(params[2]);* carSocket.unconnectToDevice();  
  
 **return true**;  
*// return carSocket.sendMessage(DataPart.getInstance().toString());* }  
 }  
  
 }**else** {  
 Log.*d*(**"info"**, **"Can't connected"**);  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return false**;  
 }  
}

**Код *wifi/Server.py***

**import** socket  
**import** pickle  
  
**from** time **import** sleep  
*# from moov.car\_controller import Moove***class** Server:  
  
 max\_clients = 10  
 was\_clients = 0  
 port = 8089  
  
  
 map = **""** data = []  
  
  
 **def** start(self):  
  
 self.s= socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 ip\_address = socket.gethostbyname\_ex(socket.gethostname())  
 self.s.bind((**''**, self.port))  
 self.s.listen(self.max\_clients)  
  
 print(**"Server started at IP: "** + str(ip\_address) + **" ; port: "** + str(self.port))  
  
 **while** self.was\_clients < self.max\_clients:  
 self.connect\_clients()  
  
 self.s.close()  
  
  
  
 **def** connect\_clients(self):  
 print(**"listen clients"**)  
 self.clt, self.adr = self.s.accept()  
 print(str(self.was\_clients) + **" connection to "** + str(self.adr) + **" established"**)  
 *# msg = self.clt.recv(64).decode("utf-8")  
 #print(msg)* self.clt.send(bytes(**"You connected"**, **"utf-8"**))  
 *# self.clt.send(bytes("You connected"))* self.was\_clients += 1  
  
 **while True**:  
 *#try:* msg = int(self.clt.recv(1024).decode(**"utf-8"**))  
 print(msg)  
 **if** (msg == 1):  
 self.messagin\_controller()  
 self.clt.close()  
 print(**"unconnected "** + str(self.adr))  
 **break  
 elif**(msg == 2):  
 *# l = self.get\_serialize()* l = self.get\_data()  
 self.data.append(l)  
 self.clt.close()  
 print(**"unconnected "** + str(self.adr))  
 **break  
 elif**(msg == 3):  
 self.map\_controller()  
 *# self.map = self.get\_serialize()  
 # print("unconnected " + str(adr))* **break  
 elif** (msg == 4):  
 self.messagin\_base()  
 self.clt.close()  
 print(**"unconnected "** + str(self.adr))  
 **break  
 elif** (msg == 0):  
 print(**"unconnected "** + str(self.adr))  
 **break  
 else**:  
 print(**"incorrect key. Try again"**)  
  
 *#except:  
 # print("unconnected (unsafe) " + str(adr))  
 # break* **def** messagin\_base(self):  
  
 self.data\_str = **","  
  
 for** d **in** self.data:  
 self.data\_str = self.data\_str + d + **","** self.data\_str = self.data\_str[1:-1]  
 print(self.data\_str)  
  
 b = bytes((str)(self.data\_str), **"utf-8"**)  
  
 self.clt.send(bytes((str)(b), **"utf-8"**))  
  
  
  
 **def** get\_data(self) -> str:  
 string = self.clt.recv(1024).decode(**"utf-8"**)  
  
 string = string.split(**']'**)  
 string = string[0]  
  
 print(**"Data: "** + string)  
  
 **return** string  
  
  
 **def** get\_serialize(self):  
 all\_data = bytearray()  
  
 **while True**:  
 data = self.clt.recv(4096)  
 **if not** data:  
 **break** all\_data += data  
  
  
 obj = pickle.loads(all\_data)  
 print(**'Obj:'** + str(obj))  
 print(**'Obj:'** + str(all\_data))  
  
 **return** obj  
  
  
 **def** map\_controller(self):  
 string = self.clt.recv(2024).decode(**"utf-8"**)  
 self.clt.close()  
 print(**"unconnected "** + str(self.adr))  
  
 string = string.split(**']'**)  
 string = string[0] + **"]"** print(**"Path: "** + string)  
  
 string = string[1: -1]  
 string = string.replace(**" "**, **""**)  
 string = string.split(**','**)  
  
 **for** val **in** string:  
 **if**(val == **"f"**):  
 val = val+**"2"  
 elif**(val == **"s"**):  
 val = val+**"5"  
 else**:  
 val = val+**"90"** print(val)  
 self.controller(val)  
 **if**(val == **"s5"**):  
 print(val)  
 self.connect\_clients()  
 sleep(4)  
  
  
 **def** controller(self, msg):  
 command = **''**.join(x **for** x **in** msg **if** x.isalpha())  
 time = **''**.join(x **for** x **in** msg **if** x.isdigit())  
  
 **if** (command == **"s"**):  
 print(**"stop"**)  
 **elif** (command == **"f"**):  
 print(**"foward"**)  
 *# m.m\_b.forward(time)* **elif** (command == **"b"**):  
 print(**"backward"**)  
 *# m.m\_b.backward (time)* **elif** (command == **"l"**):  
 print(**"low"**)  
 *# m.m\_b.change\_speed(m.m\_b.speeds[1])* **elif** (command == **"m"**):  
 print(**"medium"**)  
 *# m.m\_b.change\_speed(m.m\_b.speeds[2])* **elif** (command == **"h"**):  
 print(**"high"**)  
 *# m.m\_b.change\_speed(m.m\_b.speeds[3])* **elif** (command == **"rl"**):  
 print(**"rotate left"**)  
 *# m.rotate(1, time)* **elif** (command == **"rr"**):  
 print(**"rotate right"**)  
 *# m.rotate(0, time)* **def** messagin\_controller(self):  
  
 start = **True** *# m = Moove()  
 # m.m\_b.setUp()  
 #  
 # m.m\_r.setUp()* **while** start:  
 msg = self.clt.recv(1024).decode(**"utf-8"**)  
  
 print(**"Controller: "** + msg)  
  
 **if** (msg == **"" or** msg == **"0"**):  
 *# m.end()* start = **False  
 else**:  
 self.controller(msg)

**Код *moov/Controller.py***

**import** ASUS.GPIO **as** GPIO  
**from** time **import** sleep  
  
**class** Controller:  
  
 **def** \_\_init\_\_(self, in1, in2, en):  
 self.in1 = in1  
 self.in2 = in2  
 self.en = en  
  
 self.speeds = [10, 25, 50, 75]  
  
  
 **def** setUp(self):  
 *# GPIO.setmode(GPIO.BCM)* GPIO.setup(self.in1, GPIO.OUT)  
 GPIO.setup(self.in2, GPIO.OUT)  
 GPIO.setup(self.en, GPIO.OUT)  
 GPIO.output(self.in1, GPIO.LOW)  
 GPIO.output(self.in2, GPIO.LOW)  
 self.p = GPIO.PWM(self.en, 1000)  
  
 self.p.start(self.speeds[1])  
  
  
 **def** stop(self):  
 GPIO.output(self.in1, GPIO.LOW)  
 GPIO.output(self.in2, GPIO.LOW)  
 print(**"stop"**)  
  
  
 **def** forward(self, time):  
 print(**"forward"**)  
 GPIO.output(self.in1, GPIO.HIGH)  
 GPIO.output(self.in2, GPIO.LOW)  
 sleep(time)  
 self.stop()  
  
  
 **def** backward(self, time):  
 print(**"backward"**)  
 GPIO.output(self.in1, GPIO.LOW)  
 GPIO.output(self.in2, GPIO.HIGH)  
 sleep(time)  
 self.stop()  
  
  
 **def** change\_speed(self, speed):  
 self.p.ChangeDutyCycle(self.speeds[speed])  
 print(**"speed: "** + str(self.speeds[speed]))  
  
  
 **def** drop(self):  
 GPIO.cleanup()  
 print(**"GPIO Clean up"**)

**Код *moov/Car\_controller.py***

**from** moov.controller **import** Controller  
**import** ASUS.GPIO **as** GPIO  
  
**class** Moove():  
  
 **def** \_\_init\_\_(self):  
 self.m\_b = Controller(24, 23, 25)  
 self.m\_r = Controller(27, 22, 17)  
  
  
 GPIO.setmode(GPIO.BCM)  
 self.m\_b.setUp()  
 *# self.m\_r.setUp()* **def** rotate(self, side, deg):  
  
 **if**(side == 1):  
 self.m\_r.forward(0.2)  
 **else**:  
 self.m\_r.backward(0.2)  
  
 self.m\_b.backward(deg/45)  
  
  
 **if** (side == 1):  
 self.m\_r.backward(0.1)  
 **else**:  
 self.m\_r.forward(0.1)  
  
  
  
 **def** end(self):  
 self.m\_b.drop()  
 self.m\_r.drop()

## ПРИЛОЖЕНИЕ Б

(обязательное)

**Схема архитектуры созданной системы программного обеспечения**

## ПРИЛОЖЕНИЕ В

(обязательное)

**Руководство пользователя**

1. Введение

Руководство пользователя обеспечивает получение пользователем базовых навыков по эксплуатации приложения «*CarController*».

Разработанное программное обеспечение предназначено для дистанционного управления колёсным роботом.

Приложение обладает следующим функционалом:

* непосредственный контроль над колёсным роботом;
* контроль путём создания в редакторе и установки карты передвижения;
* контроль путём добавления заранее созданной карты передвижения.

Для использования программного обеспечения пользователь должен быть ознакомлен со следующими руководствами:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

2. Назначение и условия применения

Приложение предназначено для контроля передвижением сельскохозяйственного робота. Назначения ему необходимого пути обхода местности, с указанием мест остановок для сбора данных.

Не рекомендуется эксплуатировать робота во время грозы и плохих погодных условий.

Для корректной работы мобильного приложения пользователь должен обладать мобильным устройством с операционной системой Android версии 4.1 и выше.

Минимальные технические требования к мобильному устройству:

– доступ к сети *WiFi*;

– диагональ экрана от 4.3 дюйма;

– не менее 5 МБ свободной внутренней памяти;

3. Подготовка к работе

Для установки приложения необходимо загрузить на мобильное устройство файл с расширением *apk* и запустить процесс установки, нажав на него. При необходимости установить в настройках телефона разрешение на установку приложений из неизвестных источников. Узнайте IP адрес и порт сервера.

4. Аварийные ситуации

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования. Прежде чем завершить работу с приложением разорвите соединение с сервером, путём нажатия клавиши «*Unconnect*».

5. Правила работы с приложением

При запуске мобильного приложения введите *IP* адрес сервера и порт, нажмите клавишу «*Connect*». После успешного подключения откроется рабочее окно. Выберете тип контроля – джойстик или карта, нажав соответствующую названиям кнопку.

Для джойстика станет доступно окно управления стандартными движениями: «вперёд», «назад», «вправо», «влево» и «стоп». Также можно регулировать скорость и угол поворота.

Для карты будет представлено два варианта – загрузка готового изображения из галереи или создание собственного во встроенном редакторе. Нажмите кнопку соответствующую вашему выбору. Создайте или загрузите изображение и нажмите кнопку «*Send*».

По завершении работы с роботом нажмите кнопку «*Unconnect*» и выйдете из приложения.

6. Рекомендации по освоению.

Изучить базовые навыки работы с мобильными устройствами, при необходимости.

## ПРИЛОЖЕНИЕ Г

(обязательное)

**Руководство программиста**

1. Назначение и условия применения

Система программного обеспечения предназначена для эффективного управления сельскохозяйственным роботом.

Программное клиент-серверное обеспечение выполняет задачу связи с роботом посредством сети *WiFi*.

Для корректной работы мобильных приложений рекомендуется использовать устройства с операционной системой *Android* версии 4.1 и выше.

Для корректной работы сервера и его успешного запуска необходимо наличие на персональном компьютере *python* версии 2.х и сети *WiFi* на обоих устройствах.

Минимальные технические требования к компьютеру:

* оперативная память 2 ГБ;
* не менее 20 МБ свободной памяти на жестком диске;

Минимальные технические требования к мобильному устройству:

* диагональ экрана от 4.3 дюйма;
* не менее 5 МБ свободной внутренней памяти;

2. Характеристика программного обеспечения

Сервер работает автоматически и не требует стороннего вмешательства. Мобильные приложения нуждаются в действиях и настройке пользователя.

3. Обращение к программному комплексу

Для запуска сервера необходимо открыть консоль, перейти в директорию *scc/*, расположенную в основном рабочем пространстве, и запустить скрипт *main.py*.

Для запуска мобильного приложения – установить его на мобильное устройство и ввести требуемые для подключения IP адрес и порт.

4. Входные и выходные данные

Данные поступающие на вход серверу – это команды управления от мобильных приложений, либо информация, собранная датчиком. Выходные данные – та информация, которую накопил сервер за один проход указанного маршрута.

5. Сообщения

Код мобильных приложений оснащён логированием, которое можно просмотреть в режиме отладки.

## ПРИЛОЖЕНИЕ Д

(обязательное)

**Руководство системного-программиста**

1. Общие сведения о программном комплексе.

Программный комплекс разработан для сбора данных с сельскохозяйственных датчиков, посредством автоматизированного робототехнического передвижного устройства. Робот контролируется мобильным приложением.

Для корректной работы сервера на стороне робота необходимо наличие *python* версии 2.x.

Минимальные технические требования к компьютеру:

– возможность подключаться к беспроводной сети *WiFi*;

– оперативная память 1 ГБ;

– не менее 20 МБ свободной памяти на жестком диске;

1. Структура программного обеспечения

Сервер представлен в виде нескольких классов, отвечающих за управление движениями робота, и класса, работающего непосредственно с сетью и обрабатывающего запросы четырёх типов клиентов.

Мобильное приложение состоит из двух Activity – регистрационного и главного. На главном может располагаться один из трёх фрагментов, в зависимости от выбранного типа клиента.

1. Настройка программного обеспечения

Со стороны сервера есть возможность изменить максимальное количество подключений, которое по умолчанию – десять. Также можно настроить область прослушивания клиентских подключений.

Со стороны мобильных приложений никаких дополнительных настроек нет.

1. Проверка программного обеспечения

Программный комплекс не вызовет затруднений в работе, если подключение клиента к серверу длилось менее двух секунд. В противном случае связь может быть нестабильна.

1. Дополнительные возможности.

Возможность контролировать робота, управляя его движениями напрямую.

1. Сообщения системному программисту.

Некоторые ситуации, например, аварийное завершении работы клиента, могут вызвать сбой в работе сервера. В такое случае необходимо его перезапустить.