ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

МОБИЛЬНАЯ СИСТЕМА ДЛЯ ОПТИМАЛЬНОГО ПРОХОЖДЕНИЯ ЛАБИРИНТА НА МК ARDUINO

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.03.01
Математика и компьютерные науки
очной формы обучения, группы 07001403
Яцынюк Дарьи Александровны

Научный руководитель
к.т.н., доцент
Чашин Ю.Г.

БЕЛГОРОД 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время интерес к мобильным автономным системам возрастает. Активно ведутся разработки в области робототехники – изобретают новые механизмы, новые микроконтроллеры и т.д. Известные университеты мира, такие крупные компании как Бостон Дайнемикс и другие занимаются изобретением различных автономных систем. С появлением на рынке микроконтроллеров Ардуино многократно возрос интерес к созданию автономных систем среди обывателей. Их используют для обучения, для обустройства системы «умного дома», а также во многих других сферах человеческой жизни. Такие системы можно разделить на два класса: полностью автономные системы, способные принять самостоятельные решения, и системы управляемые дистанционно человеком. Второй способ требует непрерывного наблюдения за системой, а также имеет ряд других недостатков, например, необходимость организации непрерывной поддержки канала связи между человеком и роботом(кабельная связь или радиосвязь). В связи с этим, особенно востребованы мобильные системы с возможностью принятия решений. Такие системы требуют значительных вычислительных ресурсов, а также сложных математических моделей, чтобы система могла принимать решения для ситуаций в реальном времени, поэтому данная тема будет актуальной еще долгое время.

Наиболее распространенными системами принятия решений являются мобильные транспортные системы. Зачастую они применяются для изучения и получения информации о местах, которые человек неспособен изучить в силу своих физических размеров, или же в связи с неприсоболенностью среды для человека.

В общем случае, такие системы можно разделить на два класса – наземные и воздушные транспортные системы. К первому типу относят различные колесные средства передвижения, ко второму – различные виды летательных аппаратов.

Для наиболее адекватной оценки роботом среды, в которой он находится, ему необходимо анализировать данные телеметрии. Для этого используют, например, датчики расстояния, камеры и многое другое.

Для одного и другого типа систем принцип оценки окружающей среды для передвижения основан на выявлении препятствий и средств их обхода. Единственное различие в том, что данные с летательных аппаратов можно получить в трехмерном пространстве, а с наземных, основную часть, в двухмерном.

Основными являются показатели получаемые с датчиков расстояния, ведь именно они позволяют создать адаптивную систему, которая сможет перемещаться без боязни, того что робот разобьется о препятствие, а также это наиболее простой инструмент для адаптации к окружающей среде, не требующий больших вычислительных ресурсов.

В данной работе рассматриваются задачи изучения лабиринта и нахождения оптимального пути в нем. Часто необходимо исследовать комнату или же некое другое пространство, в которое человек не сможет поместиться. В общем виде данное пространство можно представить в виде лабиринта. Данное представление является наиболее удобным для решения поставленной задачи, поскольку для их изучения уже существуют методы решения, подходящие для представления его в памяти вычислительного устройства.

Решение лабиринта применяется в олимпиадных соревнованиях по скоростному прохождению лабиринта, а также для вышеупомянутой задачи. Данная задача уже давно известна и на данный момент имеет ряд решений. Однако, предложенные решения не вполне удовлетворяют условию универсальности. К примеру, в олимпиадном программировании заранее известны определенные структуры лабиринта, соответственно при малейшем изменении структуры робот теряет возможность быстро определять структуру, что сказывается на скорости прохождения. В иных случаях, используются математические модели, которые накладывают ограничения на исследуемый участок.

Предлагается изучить лабиринт и представить его в виде графа с вершинами и связями, а также применить алгоритм Дейкстры для нахождения кратчайшего пути из любой заданной точки.

В качестве прототипа будет использоваться мобильная система на базе микроконтроллера Ардуино. Это обусловлено простотой программирования данного контроллера, а также его широкой распространенностью.

Целью данной работы является создание мобильной системы для изучения лабиринта, определения оптимального алгоритма нахождения кратчайшего пути, подходящего для лабиринта, и применения его в мобильной системе, а также изучение получившейся комплексной системы.

В данной работе поставлены следующие задачи:

1. изучение существующих решений для построения карты лабиринта;

2. сравнение и определение аппаратных модулей, необходимых для сборки мобильной системы;

3. построение математической модели для построения карты и нахождения оптимального пути;

4. разработка графического интерфейса для пользователя;

5. разработка системы хранения изученного лабиринта.

Первая глава данной работы является обзором существующих аппаратных модулей для решения поставленной задачи, а также посвящена теоретическим сведениям о мобильной вычислительной системе.

Вторая глава данной работы посвящена проектированию математической модели решаемой задачи.

Третья глава содержит описание реализации алгоритмов предложенных во второй главе.

Четвертая глава посвящена комплексному тестированию реализованной системы.

Заключение содержит информацию о проделанной работе и анализе недостатков системы.

Данная работа содержит 67 страниц, 4 главы, 22 рисунка, 1 таблицу.

ГЛАВА 1. ОБЗОР АППАРАТНОЙ ЧАСТИ И ОБЩИЕ СВЕДЕНИЯ О МОБИЛЬНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

1.1. Обзор архитектуры контроллера Arduino Robot

Платформа Arduino – это аппаратно-программные средства для построения простых систем автоматизации и робототехники, разработанные итальянской компанией «Smart Projects». Arduino Robot - первая официальная версия Ардуино, в конструкции которого предусмотрены колеса[14]. Робот состоит из двух плат, каждая из которых содержит свой микропроцессор. Плата приводов (Motor Board) контролирует работу двигателей, в то время, как управляющая плата (Control Board) считывает показания датчиков и принимает решения о дальнейших операциях. Каждая из двух плат представляет собой полноценное устройство Ардуино, программируемое с помощью среды разработки Arduino IDE. Схема входов представлена на рисунке 1.1.

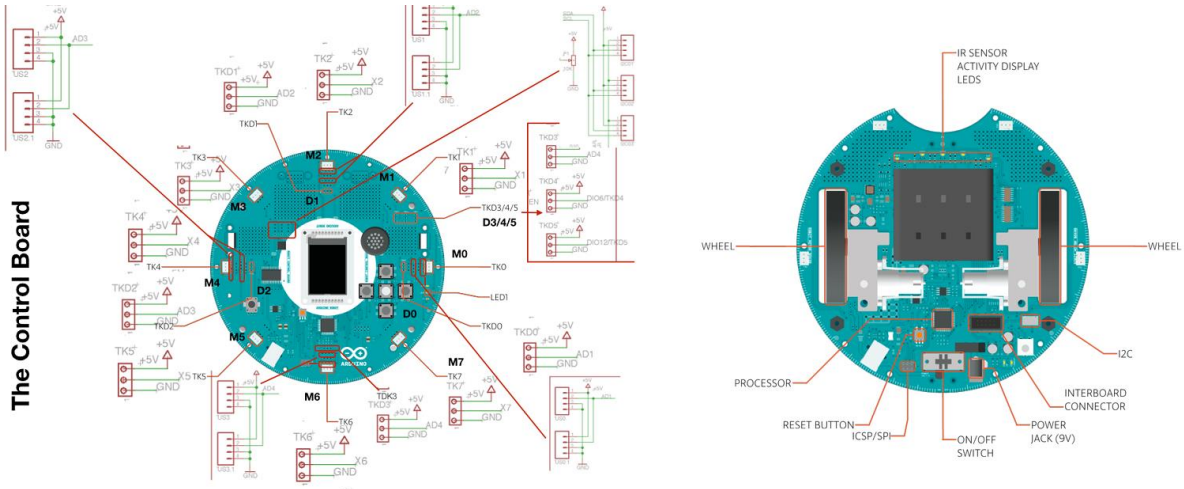

Рис.1.1. Схема входов управляющей и моторной плат

Обе платы построены на базе микроконтроллера ATmega32U4, выводы которого связаны с различными приводами и датчиками на плате. Характеристики управляющей платы:

- микроконтроллер - ATmega32u4;

- рабочее напряжение – 5В;
- напряжение питания – 5В через шлейф;
- цифровые входы/выходы - 5;
- каналы ШИМ - 6;
- аналоговые входы – 4 (на цифровых выводах) ;
- аналоговые входы(мультиплексируемые) - 8;
- максимальный ток одного вывода – 40 мА;
- flash память – 32 КБ (ATmega32u4) из которых 4 КБ используются загрузчиком;
- SRAM – 2.5 КБ (ATmega32u4) ;
- EEPROM (внутренняя) – 1 КБ (ATmega32u4) ;
- EEPROM (внешняя) – 512 Кбит (I2C) ;
- тактовая частота – 16 МГц;
- клавиатура – 5 кнопок;
- полноцветный LCD-экран – через интерфейс SPI;
- SD-картридер – для карт, отформатированных в FAT16;
- динамик – 8 Ом;
- цифровой компас – показывает отклонение от географического севера в градусах;
- I2C распаечные площадки - 3;
- области для прототипирования – 4.

Поскольку оперативная память содержит 32 КБ, из которых доступно лишь 28 КБ, то для хранения графа и расчетов кратчайшего пути этой вычислительной мощности недостаточно. В связи с этим для решения подобных задач необходимо иметь некую стороннюю вычислительную систему, например, смартфон под управлением мобильной операционной

системы связанный с транспортной системой посредством беспроводного соединения.

Данный контроллер был сделан на базе микроконтроллера Leonardo[23]. Соответствие входов приведено в таблице 1.1.

Таблица 1.1

Соответствие входов управляющей платы

Leonardo	Robot Control	Atmega	Функция	Регистр
D0	RX	PD2	RX	RXD1/INT2
D1	TX	PD3	TX	TXD1/INT3
D2	SDA	PD1	SDA	SDA/INT1
D3#	SCL	PD0	PWM8/SCL	OC0B/SCL/INT0
D4	MUX_IN_A6	PD4		ADC8
D5#	BUZZ	PC6		OC3A/#OC4A
D6#	MUXA/TKD4_A7	PD7	FastPWM	#OC4D/ADC10
D7	RST_LCD	PE6		INT6/AIN0
D8	CARD_CS A8	PB4		ADC11/PCINT4
D9#	LCD_CS A9	PB5	PWM16	OC1A/#OC4B/ADC12/PCINT5
D10#	DC_LCD A10	PB6	PWM16	OC1B/OC4B/ADC13/PCINT6
D11#	MUXB	PB7	PWM8/16	OC0A/OC1C/#RTS/PCINT7
D12	MUXC/TKD5_A11	PD6		T1/#OC4D/ADC9
D13#	MUXD	PC7	PWM10	CLK0/OC4A
A0	KEY D18	PF7		ADC7
A1	TKD0 D19	PF6		ADC6
A2	TKD1 D20	PF5		ADC5
A3	TKD2 D21	PF4		ADC4
A4	TKD3 D22	PF1		ADC1
A5	POT D23	PF0		ADC0
MISO	MISO D14	PB3		MISO, PCINT3
SCK	SCK D15	PB1		SCK, PCINT1
MOSI	MOSI D16	PB2		MOSI, PCINT2
SS	RX_LED D17	PB0		RXLED, SS/PCINT0
TXLED	TX_LED	PD5		
HWB		PE2		HWB

Однако данная плата содержит макетные площадки, которые можно использовать для расширения входов.

1.2. Изучение беспроводной связи Bluetooth на платформе Arduino

Для платформы Arduino разработаны различные аппаратные модули, предназначенные для реализации беспроводной связи. Это Bluetooth, WI-FI, связь через оптические и радиоканалы, мобильная связь в стандарте GSM[16].

Рассмотрим более подробно Bluetooth-модули. Реализованы различные аппаратные модули блютуз. Наиболее популярно семейство модулей HC.

Существуют такие типы модулей:

- **HC-03**, **HC-04**(HC-04-M, HC-04-S) на чипе BC417143 – для промышленного применения;

- **HC-05**, **HC-06**(HC-06-M, HC-06-S) на чипе BC417143 – для коммерческого применения;

- **HC-05-D, HC-06-D** (с отладочной платой для оценки и тестирования);

- **HC-07** – модуль с чипом CSR 41C6, предназначен для замены HC-06 (полностью с ним совместимый);

- **HC-08** – модуль с ультранизким энергопотреблением и протоколом Bluetooth 4.0;

- **HC-09** – самый новый модуль, предназначенный для замены HC-06 и HC-07.

Краткие характеристики модулей:

- чип Bluetooth – BC417143 производства CSR company (Cambridge Silicon Radio);

- протокол связи – Bluetooth Specification v2.0+EDR;

- радиус действия – до 10 метров (уровень мощности 2);

- совместимость со всеми Bluetooth-адаптерами, которые поддерживают SPP;

- объем flash-памяти (для хранения прошивки и настроек) – 8 Мбит;

- частота радиосигнала – 2.40 .. 2.48 ГГц;

- хост-интерфейс – USB 1.1/2.0 или UART;

- энергопотребление – ток в течение связи составляет 30-40 мА. Среднее значение тока около 25 мА. После установки связи потребляемый ток 8 мА Режим сна отсутствует.

Все Bluetooth-модули могут работать в режиме master(ведущий) или slave(ведомый). В режиме master Bluetooth-модуль является инициатором подключения, а в режиме slave ожидает запрос на соединение[27].

Практически все модули поддерживают оба этих режима, кроме HC-04 и HC-06, где режим функционирования определяется производителем, но изменение этих параметров возможно путем перепрограммирования микрокода ПЗУ[6].

Для решения поставленной задачи достаточно использовать модуль HC-04, работащий в режиме slave, поскольку инициатором соединения будет выступать смартфон.

1.3. Обзор существующих датчиков расстояния

Существует множество различных датчиков расстояния. Это и ультразвуковые датчики, и инфракрасные, а также лазерные. Они различаются по принципу работы, по максимальному и минимальному измеряемому расстоянию, а также по возникающей погрешности. Каждый имеет свои достоинства и недостатки, выбор зависит от конкретной решаемой задачи. Наиболее популярными являются ультразвуковые датчики, особенно модуль HC-SR04, в силу своей относительно малой стоимости, а также наименьшему минимальному измеряемому расстоянию. Рассмотрим его характеристики:

- напряжение питания – 4.8 В-5.5 В;
- потребляемый ток – 15 мА;
- угол обзора - <15 градусов;
- измеряемое расстояние – 2см-400см;
- точность – 0.3 см;
- рабочая температура - 0°C-60°C;
- размер - 45мм x 20мм x 15мм.

Ультразвуковой датчик HC-SR04 использует точно такую же технологию, что и летучие мыши (ультразвук). Если не вдаваться в подробности, то описать принцип работы можно, датчик посылает звуковые импульсы частотой 40 кГц и прослушивает эхо. В отличии от других датчиков,

HC-SR04 не реагирует на солнечный свет или черные предметы, но может давать ложные показания от ткани или тонких предметов.
На передней части HC-SR04 расположено два ультрозвуковых датчика, первый с надписью T (Transmiter) — это передатчик ультрозвуковых волн (TCT40-16T), а второй с надписью R (Receive) — это приемник отраженных ультрозвуковых волн (TCT40-16R), по центру расположен выводной кварцевый генератор на 27 МГц [1].

С другой стороны датчика HC-SR04, расположена электрическая обвязка, в которой выделяется три основных микросхемs и электрическая обвязка. Для взаимодействия с контроллером Arduino установлен четырех выводной разъем.

Используя сервопривод для поворота датчика можно расширить радиус действия, что позволит получать больше данных без передвижения самого робота.

1.4. Обзор интегрированной среды разработки Arduino

Для программирования контроллера Arduino используется интегрированная среда разработки(IDE) Arduino[2].

Среда разработки Arduino состоит из встроенного текстового редактора программного кода, области сообщений, окна вывода текста(консоли), панели инструментов с кнопками часто используемых команд и нескольких меню. Для загрузки программ и связи среда разработки подключается к аппаратной части Arduino. Программа, написанная в среде Arduino, называется скетч. Скетч пишется в текстовом редакторе, имеющем инструменты вырезки/вставки, поиска/замены текста. Во время сохранения и экспорта проекта в области сообщений появляются пояснения, также могут отображаться возникшие ошибки. Окно вывода текста(консоль) показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию[3].

Данная среда разработки содержит встроенные методы для работы с библиотеками предназначеными для упрощения работы с микроконтроллерами. В частности, библиотека Arduino Robot Library – предоставляющая функции для работы со всеми датчиками платы. Также среду можно расширить своими разработанными методами.

1.5. Обзор операционной системы Android

ОС Android – это операционная система, которая основана на ядре Linux и собственной реализации виртуальной машины Java от Google, созданная компанией Android Inc. Для реализации приложений под Android используется среда разработки от компанией Google, Android Studio. Для написания приложений используется язык программирования Java[20]. Некоторые обозреватели отмечают, что Android проявляет себя лучше одного из своих конкурентов, Apple iOS, в ряде особенностей, таких как веб-сёрфинг, интеграция с сервисами Google Inc. и прочих. Также Android, в отличие от iOS, является открытой платформой, что позволяет реализовать на ней больше функций[26].

Несмотря на изначальный запрет на установку программ из «непроверенных источников» (например, с карты памяти), это ограничение отключается штатными средствами в настройках аппарата, что позволяет устанавливать программы на телефоны и планшеты без интернет-подключения (например, пользователям, не имеющим Wi-Fi-точки доступа и не желающим тратить деньги на мобильный интернет, который обычно стоит дорого), а также позволяет всем желающим бесплатно писать приложения для Android и тестировать на своём аппарате[25].

Android доступен для различных аппаратных платформ, таких как ARM, MIPS, x86.

Существуют альтернативные Google Play магазины приложений: Amazon Appstore, Opera Mobile Store, Yandex.Store, GetUpps!, F-Droid.

В версии 4.3 введена поддержка многопользовательского режима.

1.6. Изучение взаимодействия ОС Android с аппаратными модулями носимых устройств

Для расширения функциональности носимых устройств используются: микроконтроллеры (МК) и микропроцессоры для управления устройством, концентраторы датчиков и системы управления ими, графические и сенсорные интерфейсы, модули Bluetooth и Wi-Fi для беспроводной связи, а также блоки обеспечения безопасности для аутентификации и защиты от подделки. Для работы с ОС Android используется стандартная библиотека android, содержащая интерфейсы программирования для работы с датчиками, модулями беспроводной связи и т.п. Такие как: bluetooth, hardware и media. Интерфейсы программирования обеспечивают набор готовых классов, процедур, функций, структур и констант для использования во внешних программных продуктах[13].

1.7. Обзор интегрированных сред разработки приложений на Android

Для разработки приложений на Android используются различные интегрированные среды разработки(IDE), такие как: Android Studio, NetBeans, Eclipse и IntelliJ.

NetBeans – свободная IDE для программирования на языках Java, Python, PHP, JavaScript, C, C++ и ряда других.

Eclipse - свободная IDE модульных кроссплатформенных приложений. Основным преимуществом является то, что любой разработчик может расширить Eclipse своими модулями. Также Eclipse является платформо-независимым продуктом, так как он написан на Java.

IntelliJ IDEA - коммерческая IDE программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Android Studio - это IDE для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O[24]. IDE основана на программном обеспечении IntelliJ IDEA от компании JetBrains. Основные особенности - реализована возможность вёрстки в реальном времени, доступно множество вариантов размеров и разрешений экранов. Присутствует раздел справки. Встроены инструменты улучшения качества приложений и монетизации. Имеются инструменты для отслеживания эффективности рекламных объявлений. Добавлено средство взаимодействия с бета-тестерами[4].

1.8. Изучение особенностей программирования Android-приложений

Java — это основной язык для разработки приложений на платформе Android. Имеется также возможность написания приложений на различных скриптовых языках. Разработка приложения для android целиком на C/C++, по крайней мере официально, не поддерживается (хотя есть программы, которые преобразуют C/C++ код в java код). Структура андроид-приложения: класс MainActivity.java, файлы activityMain.xml(разметка приложения), AndroidManifest.xml(содержит описание проекта), различные xml-файлы, содержащие ресурсы проекта, библиотеки, используемые приложением[26].

ГЛАВА 2. ПРОЕКТИРОВАНИЕ МАТЕМАТИЧЕСКОЙ МОДЕЛИ И ФУНКЦИОНАЛЬНОЙ СХЕМЫ

2.1. Алгоритм прохождения лабиринта

Существует 2 алгоритма для прохождения лабиринта: «правило одной руки» и алгоритм Люка-Тремо.

Тремо предлагает следующие правила: выйдя из любой точки лабиринта, надо сделать отметку на его стене (крест) и двигаться в произвольном направлении до тупика или перекрестка; в первом случае вернуться назад, поставить второй крест, свидетельствующий, что путь пройден дважды - туда и назад, и идти в направлении, не пройденном ни разу, или пройденном один раз; во втором - идти по произвольному направлению, отмечая каждый перекресток на входе и на выходе одним крестом; если на перекресте один крест уже имеется, то следует идти новым путем, если нет - то пройденным путем, отметив его вторым крестом. Данный алгоритм является универсальным и подходит для всех типов лабиринтов, однако для представления пройденных путей в вычислительной машине требуется большой объем оперативной памяти, что представляет невозможным создание мобильной системы, которая могла бы принимать свои решения.

«Правило одной руки» заключается в том, что нужно выбрать одну из стен, прикоснуться к ней рукой и следовать вдоль неё. Данный алгоритм является самым простым и не требует хранения уже пройденных вершин, хотя и подходит не для всех типов лабиринтов – возможна ситуация, когда часть лабиринта не будет изучена. Для решения этой проблемы можно воспользоваться правилом другой руки, возможно это расширит изученное пространство.

Однако, оба метода не гарантируют нахождение выхода из лабиринта. Для решения поставленой задачи было решено использовать «правило одной

руки», т.к. необходимо разработать мобильную систему с возможностью принятия самостоятельных решений.

Опишем алгоритм для робота. В начале своей работы робот должен найти стену, по которой он будет следовать. Для этого он может просто двигаться вперед, пока не упрется в преграду. После того как робот наткнулся на препятствие, он начинает передвигаться в соответствии с правилом "правой руки". Двигаясь вдоль стены, робот следит, есть ли проход справа. Если проход есть, робот должен идти по нему, чтобы не оторваться от стены справа. Если прохода нет - впереди стена - робот поворачивает налево. Если прохода снова нет, он еще раз поворачивает налево, таким образом разворачиваясь на 180 градусов, и идет в обратном направлении.

Блок-схема алгоритма приведена на рисунке 2.1.

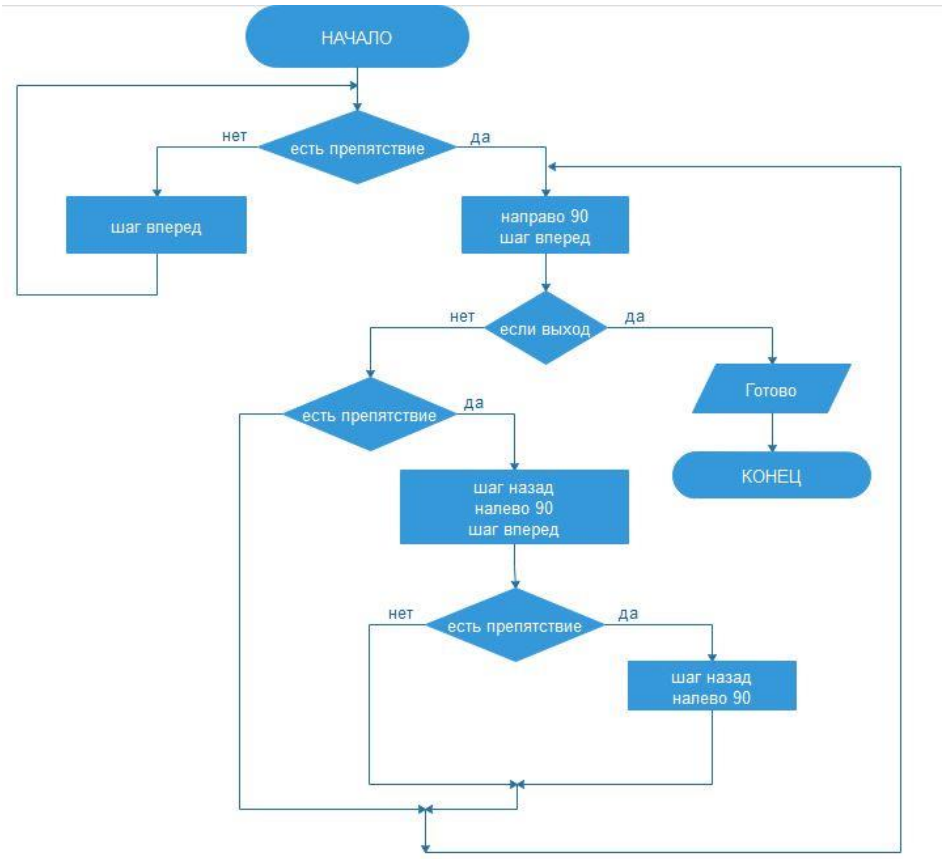

Рис. 2.1. Блок-схема алгоритма для изучения лабиринта

На блок-схеме (см. рис. 2.1) проиллюстрировано «правило правой руки», при котором требуется придерживаться правой стены.

2.2. Алгоритм нахождения оптимального пути

Поскольку лабиринт можно представить в виде графа, то поиск оптимального пути сводится к нахождению кратчайшего пути в графе[9]. Проведем сравнительный анализ основных существующих алгоритмов.

Первый алгоритм – алгоритм Флойда-Уоршелла[10]. Находит расстояние от каждой вершины до каждой за количество операций порядка **n^3**. Веса могут быть отрицательными, но не может быть циклов с отрицательной суммой весов рёбер. Поскольку расстояние между точками для лабиринта представляется в виде положительного значения веса ребра, а эффективность данного алгоритма не самая лучшая, то целесообразно использовать другой алгоритм.

Волновой алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные[11]. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь. Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости. Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути. Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки. Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке. Для решения данной задачи волновой алгоритм не подходит –

необходимо четко знать размеры лабиринта/комнаты и «накладывать сетку» посещенных и непосещенных точек, что накладывает ограничения на структуру лабиринта, а также замедляет процесс изучения лабиринта.

Алгоритм Дейкстры - находит расстояние от одной вершины (дадим ей номер 0) до всех остальных за количество операций порядка n^2[18]. Все веса неотрицательны. В простейшей реализации для хранения чисел $d[i]$ можно использовать массив чисел, а для хранения принадлежности элемента множеству U — массив булевых переменных[7].

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бóльшим максимального возможного пути в графе). Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины. Если в них расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается, когда флаги всех вершин становятся равны 1.

Данный алгоритм подходит для решения поставленной задачи, однако можно заметить, что в случае с лабиринтом возможное количество ребер относительно мало, что дает нам право утверждать, что любой лабиринт можно представить разреженным графом. Для таких графов существует модифицированный алгоритм Дейкстры – с использованием двоичной кучи.

Двоичная куча – это такое двоичное дерево, для которого выполнены три условия:

1. значение в любой вершине не меньше, чем значения её потомков;
2. глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой;
3. последний слой заполняется слева направо без «дырок».

Для разреженных графов (то есть таких, для которых m много меньше n^2) непосещённые вершины можно хранить в двоичной куче, а в качестве ключа использовать значения $d[i]$(расстояние от начальной вершины до i-ой), тогда время удаления вершины из множества посещенных вершин станет $\log n$ при том, что время модификации $d[i]$ возрастёт до $\log n$. Так как цикл выполняется порядка n раз, а количество релаксаций (смен меток) не больше m, время работы такой реализации - $O(n \log n + m \log n)$.

2.3. Функциональная схема взаимодействия устройства на Arduino с устройством на Android

Данный алгоритм должен реализовывать следующую схему взаимодействия устройства на Arduino с устройством на Android:

- устройство на Android посылает команду, определяющую режим (изучение или перемещение по кратчайшему пути) на контроллер Arduino;
- устройство на Arduino непрерывно посылает данные о лабиринте, если определен режим изучения;
- устройство на Android посылает пачку команд, содержащую указатели для передвижения.

Схему взаимодействия иллюстрирует рис.2.2.

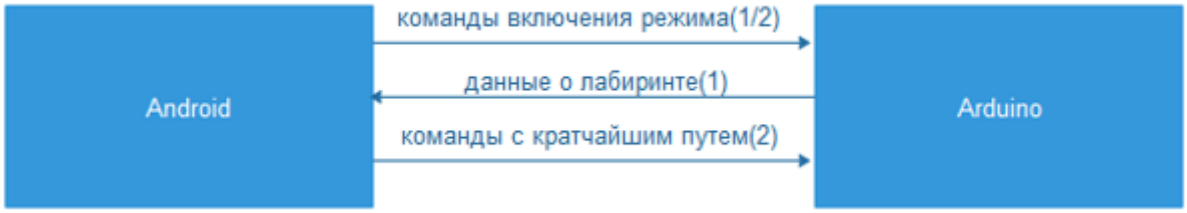

Рис. 2.2. Схема взаимодействия между Android устройством и Arduino

Схема передачи данных по Bluetooth соединению, используя режим изучения лабиринта, должна содержать следующее:

- подключение к устройству Arduino по Блютуз соединению;

- отправка команды на устройство;
- получение данных о лабиринте.

Данная схема показана на рис.2.3.

Рис.2.3.Схема передачи данных между устройствами при режиме изучения

Схема запроса телеметрии реализована в следующем виде:

- подключение к устройству Arduino по Блютуз соединению;
- отправка команды о выбранном режиме на устройство;
- отправка команд с кратчайшим путем.

Данная схема проилюстрирована на рис.2.4.

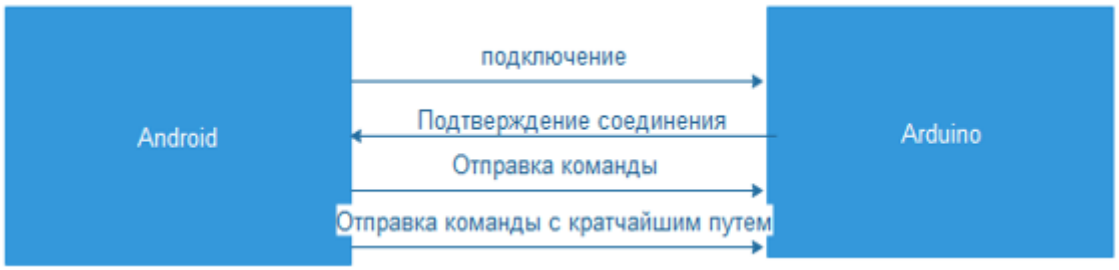

Рис.2.4. Схема передачи данных между устройствами при режиме перемещения

Поскольку выбранный микроконтроллер имеет некоторые проблемы с подключением Bluetooth модуля, то было принято решение использовать в качестве переходного устройства для передачи данных другой микроконтроллер - Uno. А также было решено перенести управление сервоприводом. Схема их взаимодействия выглядит следующим образом:

- Отправить команду о начале движения.
- Передать точки графа.
- Подтвердить получение.

- Получить команду о повороте сервопривода.
- Подтвердить поворот.

Данная схема представлена на рис.2.5.

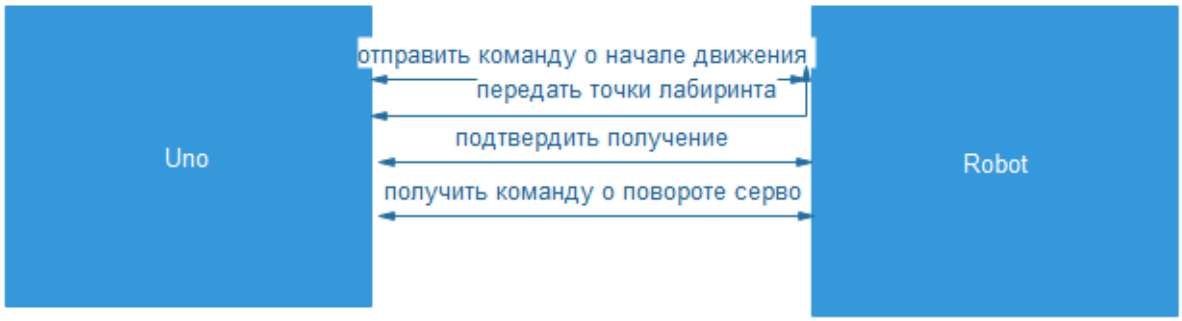

Рис.2.5. Схема взаимодействия между Arduino устройствами

Для реализации вышеперечисленных функциональных схем были разработанны алгоритмы, отдельные для каждой из систем.

2.4. Алгоритм для робота

Завершив проектирование математической модели и определив функциональное взаимодействие межу всеми частями системы, были разработаны алгоритмы для каждой части. Начнем с робота.

Алгоритм для робота должен выглядеть следующим образом:

1. Подключение к смартфону по беспроводной связи.
2. Получение команды о начале изучения лабиринта.
3. Считать расстояние впереди.
4. Считать расстояние слева.
5. Ехать вперед пока слева нет прохода или впереди нет препятствия.
6. Отослать координаты точки на смартфон.

7. Если слева проход – повернуть налево, иначе направо.

8. Повторить шаг 6.

9. Если выход не найден – повторить шаги 3-8 Иначе – отослать информацию об оканчании изучения лабиринта на смартфон.

10. Если получена команда о переходе из одной точки в другую, то выполнить команды со смартфона. Блок-схема на рис.2.6.

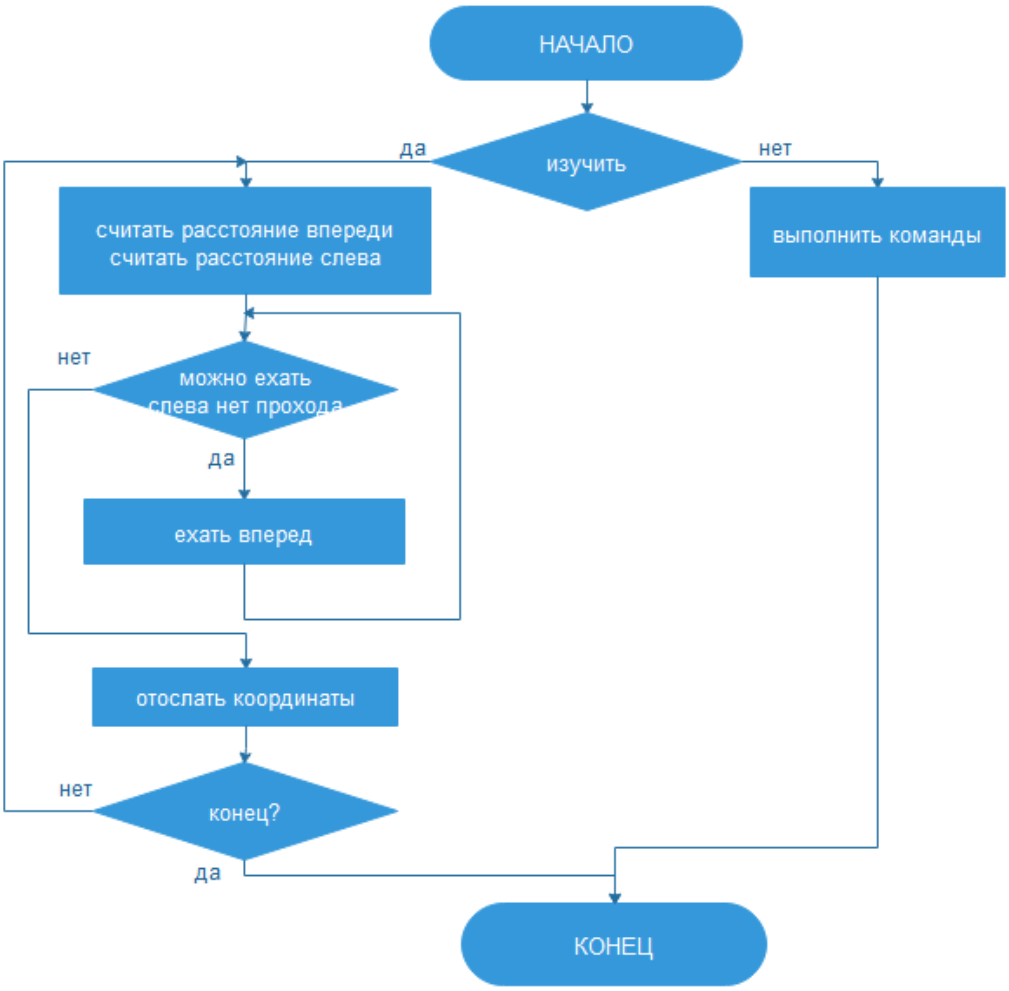

Рис.2.6. Блок-схема для алгоритма робота

Используя данный алгоритм робот сможет в любом случае найти выход из лабиринта. Для более точного изучения можно применить этот алгоритм 2 раза – с левой и правой стеной.

2.5. Алгоритм для смартфона

Поскольку смартфон является не просто хранилищем данных, но и вычислительным ресурсом, то для него тоже необходимо разработать алгоритм.

Смартфон должен выступать инициатором беспроводного bluetooth соединения. В смартфоне необходимо разработать графический интерфейс, который бы упростил работу с мобильной системой и соответствовал современным стандартам современного дизайна приложений.

Поскольку правило одной руки не гарантирует точного изучения лабиринта, то предлагается давать возможность пользователю решить достаточно ли изучен данный лабиринт или следует применить правило другой руки, или же просто продолжить изучение, если был найден выход.

А также одной из функций предложенного решения является хранение уже изученых маршрутов, чтобы данная система имела некое прикладное применение.

В качестве расширения функционала для пользователя, приложение должно визуализировать исследованную территорию, чтобы человек мог проверить визуально полученный результат.

Алгоритм должен выглядеть следующим образом:

1. Подключиться к роботу по беспроводной связи.
2. Отослать команду роботу о начале изучения лабиринта.
3. Получить координаты точки.
4. Занести точку в локальное хранилище.
5. Повторить шаги 3-4 пока не получена информация о завершении изучения лабиринта.
6. Получить все точки из локального хранилища.

7. Посчитать расстояния между связанными точками.

8. Преобразовать связанные точки в дуги графа.

9. Расчитать кратчайшее расстояние и путь.

10. Отправить команды роботу.

Блок-схема предложенного алгоритма приведена на рис.2.7.

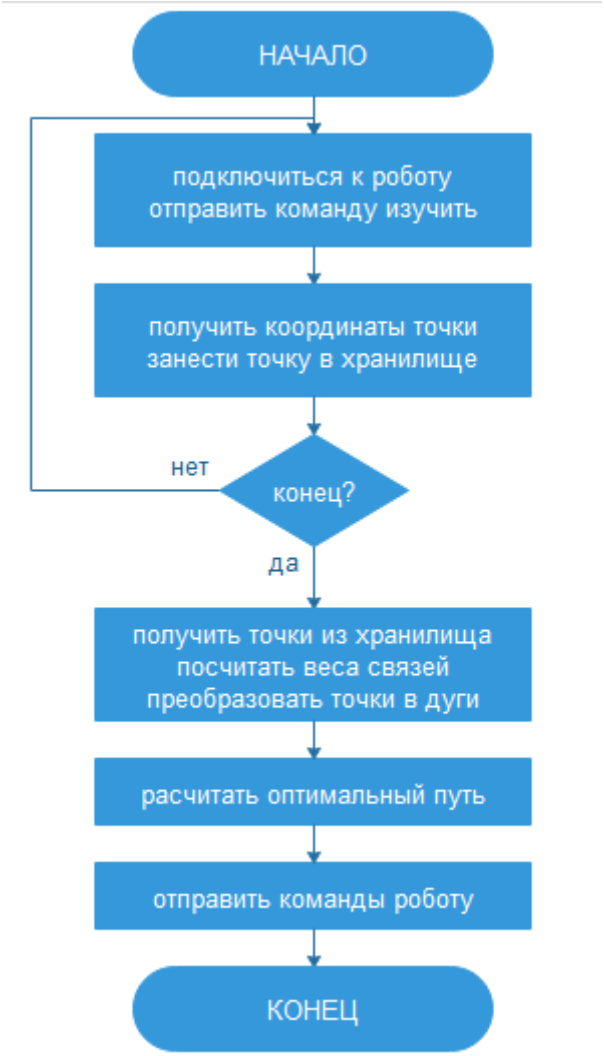

Рис. 2.7. Блок-схема алгоритма для смартфона

В целом, смартфон является инициатором процесса изучения и звеном, вычисляющим оптимальный путь.

2.6. Проектирование базы данных

Предложенное решение предусматривает использование локального хранилища. Необходимо разработать логическую и физическую схему базы данных для хранения лабиринта.

База данных должна содержать следующие сущности: Вершины графа, Связи, Пути. Вершины графа должны иметь следующие свойства: координаты X и У, номер вершины в данном пути, номер пути. Сущность Связи содержит следующие поля: номер связи, номер родительской вершины и номер вершины-потомка. Хранить вес связи не обязательно ведь мы имеем координаты родительской и дочерней вершины, а значит можем провести вычисления используя Евклидово расстояние. Сущность Пути была добавлена для удобства пользователя, чтобы не пришлось изучать заново уже пройденную территорию. В ней содержится номер пути и название(см. рис.2.8).

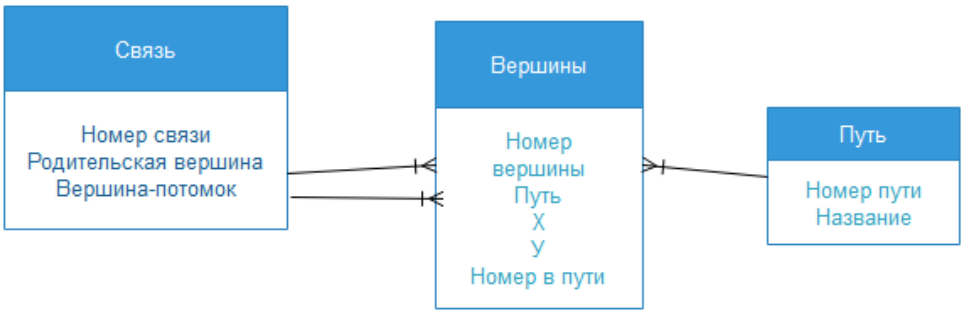

Рис.2.8. Логическая схема базы данных

Логически база данных должна следовать следующим правилам: каждый путь может содержать множество вершин, каждая вершина может иметь множество связей.

Любая база данных должна быть приведена хотя бы к третьей нормальной форме.

Данная база данных находится в первой нормальной форме, так как все атрибуты являются простыми, а также не существует повторений записей.

Также она находится во второй нормальной форме, поскольку находится в первой нормальной форме и каждый неключевой атрибут неприводимо зависит от первичного ключа.

Также она находится в третьей нормальной форме, так как она находится во второй нормальной форме и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. То есть все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы вынесено в отдельные таблицы.

ГЛАВА 3. РЕАЛИЗАЦИЯ АЛГОРИТМОВ ДВИЖЕНИЯ, ПЕРЕДАЧИ ДАННЫХ И НАХОЖДЕНИЯ КРАТЧАЙШЕГО ПУТИ

3.1. Реализация алгоритма на Arduino устройстве

Структура Android-приложения должна соответствовать шаблону, приведенному на рис.3.1.

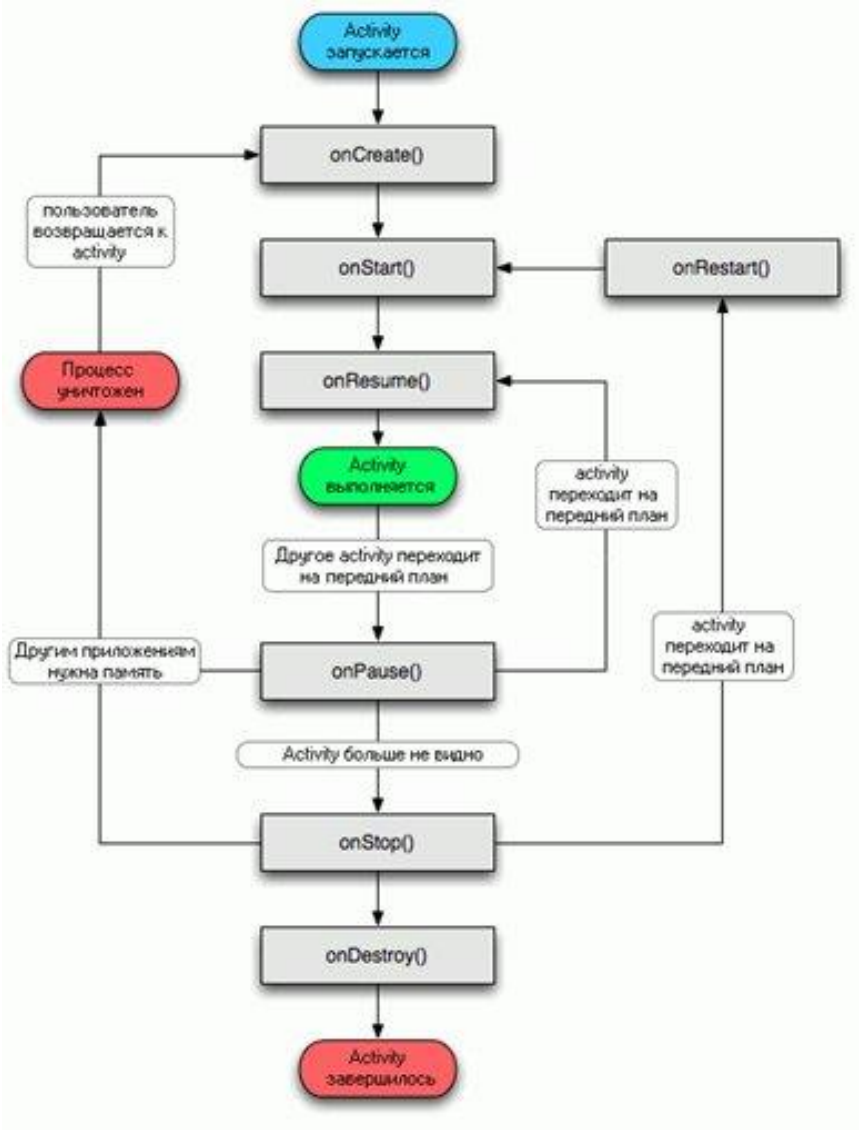

Рис.3.1. Структура Android-приложения

Данное приложение содержит следующие функциональные модули: модуль для взаимодействия по беспроводной связи, модуль для работы с

локальным хранилищем данных, а также модуль для расчета кратчайшего пути. Рассмотрим каждый модуль в отдельности.

Начнем с модуля обмена данными. Данный модуль базируется на классе Thread. Для обработки данных следует написать свою реализацию методов отправки сообщения, отмены отправки, а также метод для чтения данных[25].

Был реализован свой класс ConnectedThred наследуемый от класса Thread. Он содержит все обязательные для реализации методы. Метод sendData() (для отправки данных, см.листинг 1) преобразовывает строковое сообщение в массив байтов, для последующей пересылки данных используя потоки.

Листинг 3.1. Метод для отправки данных

```
 public void sendData(String message) {
byte[] msgBuffer = message.getBytes();

try {
   OutStrem.write(msgBuffer);
} catch (IOException e) {}
}
```

Метод cancel()(для отмены) закрывает сокет для передачи данных.

И метод run() (для чтения данных, см.листинг 3.2). Он считывает данные из потока и инициирует обработку сообщения handler-ом.

Листинг 3.2. Метод для чтения данных

```
      public void run(){
byte[] buffer = new byte[1024];
int bytes;
while(true){
   try{
      bytes = InStrem.read(buffer);
      h.obtainMessage(ArduinoData, bytes, -1, buffer).sendToTarget();
   }catch(IOException e){break;}}
```

При работе с обменом сообщений создается поток для передачи данных.

Для приема данных использовался поток Handler – механизм, который позволяет работать с очередью сообщений(см. листинг 3.3). Исходный код см.Приложение 2.

Листинг 3.3. Handler для работы с очередью сообщений

```
  h = new Handler() {
 public void handleMessage(android.os.Message msg) {
    switch (msg.what) {
       case ArduinoData:
          byte[] readBuf = (byte[]) msg.obj;
          String strIncom = new String(readBuf, 0, msg.arg1);
          mytext.setText("Данные от Arduino: " + strIncom);
          break;
    }
  }
};
```

Функциональный модуль для работы с локальным хранилищем представляет собой набор методов для работы с локальной базой данных. Как уже было сказано, смартфоны под управлением ОС Android позволяют работать с компактной СУБД SQLite.

В соответствии с разработанной структурой необходимо реализовать внедрение базы данных в приложение.

В языке программирования Java есть специальная обертка для работы с базой данных SQLite - Room. Она позволяет уменьшить количество кода, а также избежать лишнего добавления сущностей за счет внедрения шаблона проектирования Dependency Injection[12].

Были разработаны программные модели соответствующие описанной структуре, используя аннотации Entity. Были определены типы даных и внешние связи(ForeignKey), первичные ключи(PrimaryKey).

После определения моделей был реализован объект DAO – Data Access Object. Он содержит все необходимые методы для получения данных из БД. С помощью аннотаций Insert, Updated, Delete, Query. Написанные строковые

представления запросов к БД преобразовуются в запросы с соответсвующими параметрами. Пример приведен в листинге 3.4.

Листинг 3.4. Пример DAO - ConnectionDAO

```
@Dao
public interface ConnectionDAO {
   @Insert(onConflict = OnConflictStrategy.REPLACE)
   public void insertConnection(Connection con);

   @Update
   public void updateConnection(Connection con);

   @Delete
   public void deleteConnection(Connection con);

   @Query("SELECT * FROM Connection")
   public List<Connection> getConnectionsList();

   @Query("SELECT * FROM Connection WHERE Id= :id")
   public Connection getConnectionById(int id);

   @Query("SELECT * FROM Connection WHERE ParentId= :id")
   public List<Connection> getConnectionForPoint(int id);

   @Query("SELECT COUNT(Id) FROM Connection WHERE ParentId= :parentId and
ChildId= :childId")
   public int checkIfConnectionExists(int parentId, int childId);
}
```

На данном этапе подготовительные этапы БД окончены. Следующим шагом было добавление подготовленной структуры в основное приложение.

С помощью аннотации Database определим вспомагательный класс, который дает доступ к сущности базы данных. Данный класс содержит метод для получения контекста БД вместе с DAO объектами(см. листинг 3.5)

Листинг 3.5. Внедрение контекста БД в основное приложение

```
        public static MazeDatabase getMazeDatabase(Context) {
  if (INSTANCE == null) {
     INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
              MazeDatabase.class, "MazeDB.db")
              .build();
  }
  return INSTANCE;}
```

После этого класс MazeDatabase можно применять в основном классе MainActivity.

Последний модуль содержит реализацию алгоритма Дейкстры. Так как было определено, что необходимо использовать двоичную кучу для реализации алгоритма, а Java не содержит готовых методов, то была самостоятельно реализована структура.

Для представления лабиринта в виде графа было решено использовать список смежности, где каждой вершине графа соответствует список, состоящий из соседей этой вершины.

Поскольку вершины хранятся в локальной базе данных, то необходимо получить сначала все вершины соответствующие данному пути, затем вычислить расстояние между вершинами для инициализации вершины в списке смежности.

После проведения расчетов необходимо получить кратчайший путь, состоящий из номеров вершин, занесенный в массив. Эти данные необходимо затем передать роботу для его дальнейшего перемещения.

3.2. Сборка прототипа системы

Как уже было сказано, в качестве системы перемещения была выбрана платформа Arduino. В первой главе были описаны выбранные датчики и микроконтроллеры для решения задачи.

Контроллер Arduino Robot был выбран из-за того, что для него уже написана основная библиотека для перемещения, а также есть распаянные основные входы. Однако, для подключения большого количества датчиков и чтобы оставить данный контроллер универсальным необходимо распаять контактные площадки.

Основная проблема выбранного микроконтроллера Arduino Robot состоит в невозможности подключения Bluetooth модуля, из-за несоответствия входов[15]. Для решения данной проблемы было решено выбрать в качестве переходного звена для передачи данных другой микроконтроллер – Arduino

Uno. Для связывания двух контроллеров использовалась технология I2C/TWI. Для этого в коде используется библиотека Wire[17].

Поскольку плате перемещения требуется постоянно отслеживать показатели датчиков расстояния слева, для того чтобы система могла вовремя остановить движение и не пропустить поворот налево, а передача данных через провода занимает некоторое количество времени и возможно возникновение потери или искажения передаваемых данных, то было решено оставить подключение и считывание данных с датчика расстояния на плате движения.

Для разгрузки платы движения, было принято решение перенести функцию поворота сервопривода на вспомогательную плату. Это стало возможным, так как поворот не является постоянной операцией и не требует моментального реагирования.

С помощью приложения fritzing была разработана принципиальная схема для решения данной задачи, учитывая все возможные проблемы с аппаратной частью, представленная на рисунке 3.2.

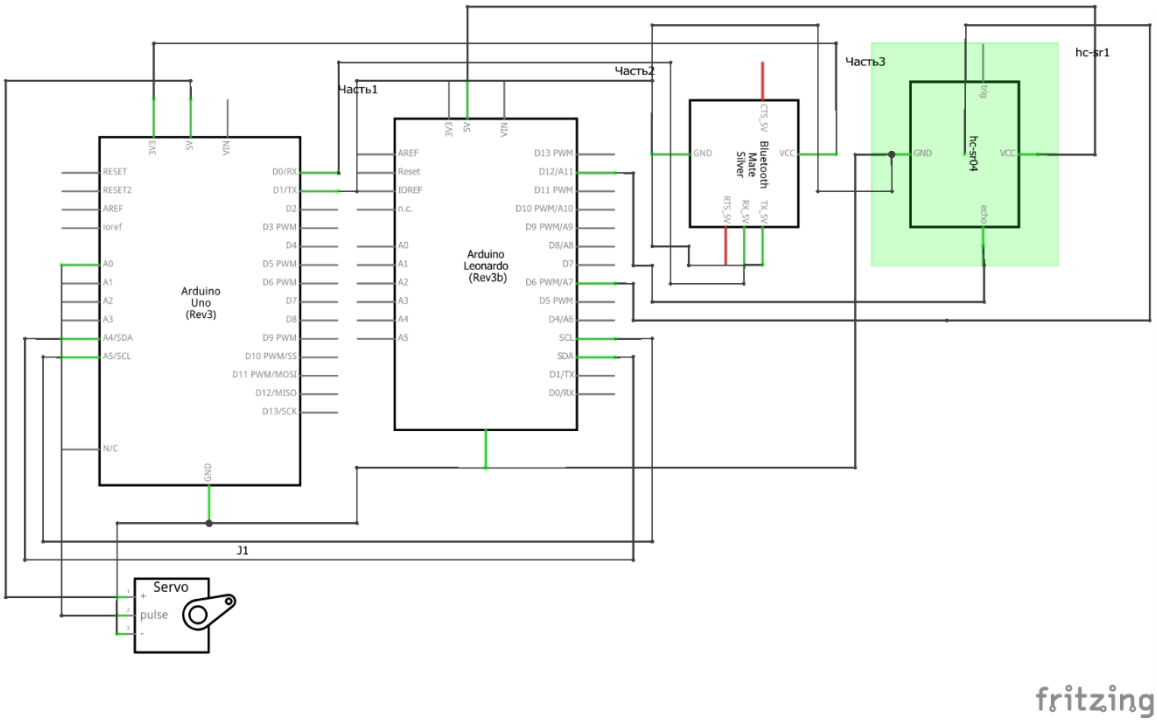

Рис.3.2. Схема принципиальная

Поскольку плата Arduino Robot сделана на базе Arduino Leonardo, то для более компактного представления схем использовалась плата Leonardo, все входы соответствуют тем, что приведены в карте соответствий.

В соответствии с разработанной схемой принципиальной было собрано устройство, представленное на рис.3.3.

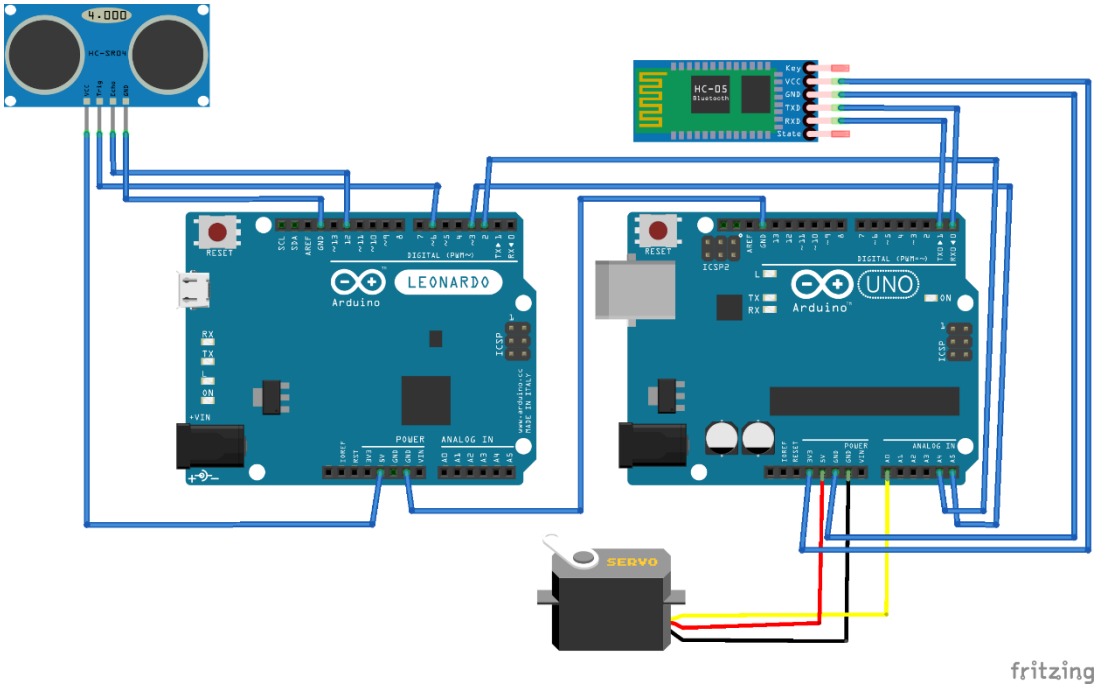

Рис.3.3. Устройство

Подключив все необходимые датчики, был реализован алгоритм поведения робота и его взаимодействия с вычислительной системой.

3.3. Реализация алгоритма на Arduino устройстве

Алгоритм условно содержит 2 части – часть для МК Robot и часть для МК Uno.

Поскольку МК Uno обеспечивает беспроводную связь, то необходимо реализовать функционал для получения и отправки данных. Для этого в Ардуино применяется библиотека Serial.h. Также необходимо реализовать передачу данных на другой микроконтроллер. Поскольку соединение проводное, то используется библиотека Wire.

Программы в Arduino представляют собой «скетчи», содержащие команды которые выполняются непрерывно в методе loop().[5] Данный метод циклически ведет опрос необходимых датчиков.

В случае с переходной платой Uno циклически идет опрос последовательного порта – на него приходят данные от смартфона. Чтение данных происходит только, если пришли данные на этот порт. Как только приходят данные с телефона, соответствующие выбору режима или команды пути, с этой платы по проводному соединению передаются команды на плату движения. Код приведен в листинге 3.6.

Листинг 3.6. Получение данных по беспроводному соединению

```
while(Serial.available()) {
 character = Serial.read();
 if(character == 's' || character == 'b') {
   Wire.beginTransmission(ROBOT_ADDRESS);
   Wire.write(character);
   Wire.endTransmission();
 }
}
```

Также как и для андроид приложения, в ардуино приложении для получения данных используется handler, чтобы не происходило чтения лишних данных. В данном случае Uno ожидает от Robot либо данные о лабиринте, либо сигнал об окончании изучения лабирнта, либо команду поворота сервопривода. Пример приведен в листинге 3.7.

Листинг 3.7. Взаимодействие с платой движения

```
 void receiveEvent(int bytes) {
// Get the message when available
if(message[0]=='#'){
Serial.println(message);
}
if(message[0]=='l')
{
 Servo1.write(179);
}
if(message[0]=='f')
{
 Servo1.write(90);
}
}
```

Поскольку сервопривод очень чувствителен к граничным значениям(0 и 180), то при повороте налево/направо было решено использовать ближайшие к граничным, а не граничные значения(см. приложение 2)

Рассмотрим реализацию алгоритма для платы движения. Для упрощения решаемой задачи, робот будет перемещаться строго по вертикальным и горизонтальным линиям. Это позволит создать простейшую локальную систему координат.

Примем начальное положение робота как начальную точку локальной системы координат – (0;0). В качестве локальной системы координат было принято решение использовать Декартову систему координат на плоскости. Для расчета точек в данной системе необходимо знать направление робота. Для этого введем две переменные – global_X и global_Y – которые будут показывать направление изменения координат. Таким образом, при движении прямо эти переменные должны принять значения (0;1), при движении направо – (1;0), при движении вниз – (0;-1), а при движении налево – (-1;0). Для корректировки данных значений была написана функция, представленная в листинге 3.8.

Листинг 3.8. Функция корректировки направления

```
 //изменить указатели направления робота
void changeGlobals(int side){
  if(global_X==0){
    if(global_Y==1 && side==1){
      global_X=-1;global_Y=0;
    }else if (global_Y==1 && side==-1){
      global_X=-1;global_Y=0;
    }else if (global_Y==-1 && side==1){
      global_X=-1;global_Y=0;
    }else{global_X=-1;global_Y=0;
    }
  } else if(global_Y==0){
    if(global_X==1 && side==1){
      global_X=0;global_Y=-1;
    }else if(global_X==1 && side==-1){
      global_X=0;global_Y=1;
    }else if(global_X==-1 && side==1){
      global_X=0;global_Y=1;
    }else if(global_X==-1 && side==-1){
      global_X=0;global_Y=-1;  } }}
```

Данная функция вызывается каждый раз, когда робот совершил поворот. Переменные используются для изменения координат, характеризующих вершину графа (проход в лабиринте).

Используя формулы 3.1 и 3.2 мы всегда можем без лишних вычислений узнать нужную координату:

$$X += global\_X * distanse \quad (3.1)$$

$$Y += global\_Y * distanse \quad (3.2)$$

Пройденное растояние обычно вычисляют используя оптопару - электронный прибор, состоящий из излучателя света (обычно — светодиод, в ранних изделиях — миниатюрная лампа накаливания) и фотоприёмника (биполярных и полевых фототранзисторов, фотодиодов, фототиристоров, фоторезисторов), связанных оптическим каналом и, как правило, объединённых в общем корпусе. Принцип работы оптрона заключается в преобразовании электрического сигнала в свет, его передаче по оптическому каналу и последующем преобразовании обратно в электрический сигнал. Однако физическое устройство Arduino Robot не позволяет использовать данную технологию. Поэтому было решено использовать встроенные возможности по замерам времени.

Первое, что было сделано, это определено расстояние, которое робот проезжает с заданной скоростью за 1 секунду. Практически была проверена и определена скорость движения робота равная 31см/с. В Arduino встроена функция, которая возвращает время от начала работы скетча. Вычислив разницу между начальным временем и конечным временем между перемещениями, можно узнать время, за которое робот проехал определенное растояние. В совокупности оба этих показателя дают возможность вычислять пройденный путь, по всем известной формуле, где растояние есть ни что иное, как время умноженное на скорость.

Также используя этот метод можно избавить робота от лишних поворотов сервопривода и считываний информации. Поскольку необходимо постоянно следить за левой/правой стеной и в то же время не разбиться о препятствие впереди, то можно перед началом движения расчитать расстояние, затем расчитать время, которое робот может ехать до препятствия впереди, а данные при движении получать только о расстоянии по левой стене. В таком случае в условиях для продвижения робота вперед необходимо учитывать пройденное время от начала движения, а также изменение расстояния от левой/правой стены.

Для определения завершил ли изучение лабиринта робот необходимо ввести некий флаг. Поскольку Robot содержит инфракрасные датчики линий, позволяющие определить пересек ли робот линию, то в связи с этим было решено в качестве флага окончания изучения выбрать черную линию. Данные показатели также необходимо учитывать в условиях для дальнейшего продвижения. Реализация приведена в листинге 3.9.

Листинг 3.9. Вычисление расстояния

```
    leftDistance=getDistance();
  startPointTime=micros()/1000000;
  currentDistance=getDistance();
  Robot.updateIR();
  while((micros()/1000000)<(startPointTime+timeToPass)&&
(currentDistance>=(leftDistance-5) && currentDistance<=(leftDistance+5)))
  {
   Robot.motorsWrite(127,127);
   currentDistance=getDistance();
   if(Robot.IRarray[3]==0)
   {
    break;
   }
   Robot.updateIR();
  }
  Robot.motorsWrite(0,0);
  endPointTime=micros()/1000000;
  delay(1000);
  float realTime=endPointTime-startPointTime;
  float passedDistance=realTime*31.0;
```

Данный контроллер не должен действовать в цикле. Он также как и Uno содержит функцию – handler, которая при вызове того или иного режима выполняет свои функции(см. приложение 3).

В ходе реализации алгоритма возник вопрос о проходимости робота. Было необходимо решить, как определить проходит ли робот в лабиринте и что делать, если робот не помещается.

Было предложено следующее решение – использовать датчик расстояния для определения расстояния до прохода слева и справа (угловые значения), определить угол и затем посчитать по теореме косинусов третью сторону:

$$a^2 = b^2 + c^2 - 2bc\cos\alpha \qquad (3.3)$$

После извлечения корня квадратного из a, полученную величину следует сравнить с диаметром робота. Если робот проходит, то алгоритм продолжается в обычном режиме, если же нет, то робот воспринимает это как препятствие и продолжает работу алгоритма.

ГЛАВА 4. ТЕСТИРОВАНИЕ РЕАЛИЗОВАННОЙ СИСТЕМЫ

4.1. Тестирование беспроводного соединения

В соответствии с современными требованиями к дизайну мобильных приложений, в среде разработки Arduino Studio был разработан графический интерфейс представленный на рис.4.1.

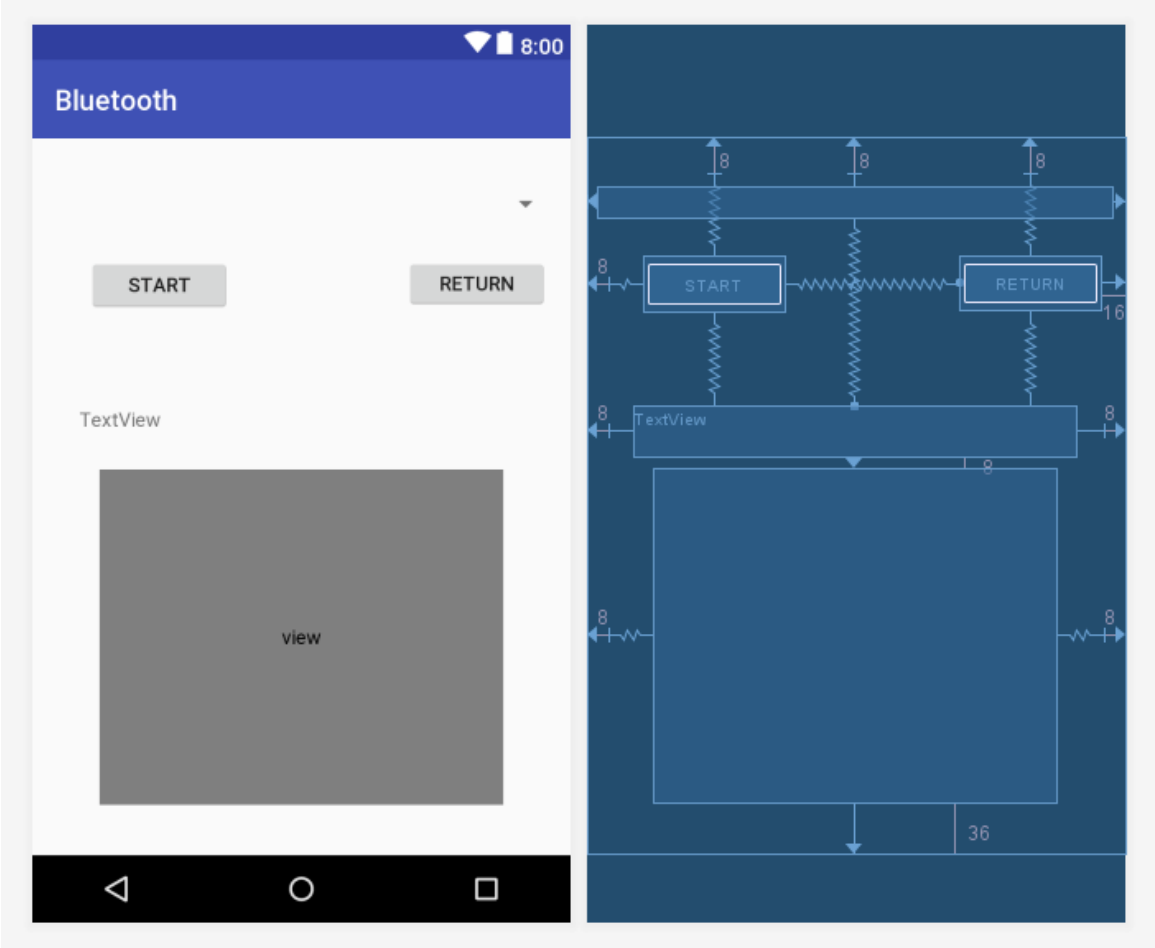

Рис. 4.1. Графический интерфейс приложения

Были добавлены 2 кнопки – Start и Return. По нажатии на них вызываются соответствующие события – Изучение и Возврат.

Также был добавлен компонент Spinner, позволяющий выбрать уже существующий путь.

Для индикации работы алгоритма было добавлено текстовое поле, показывающее подключен ли Bluetooth, сопряжены ли устройства,

закончилась фаза изучения лабиринта или нет, а также любую другую возможную информацию.

Для визуального отображения изученного лабиринта был добавлен компонент View, содержащий в себе кастомный класс для прорисовки точек графа.

Физическое изображение собранной мобильной системы представлено на рисунке 4.2.

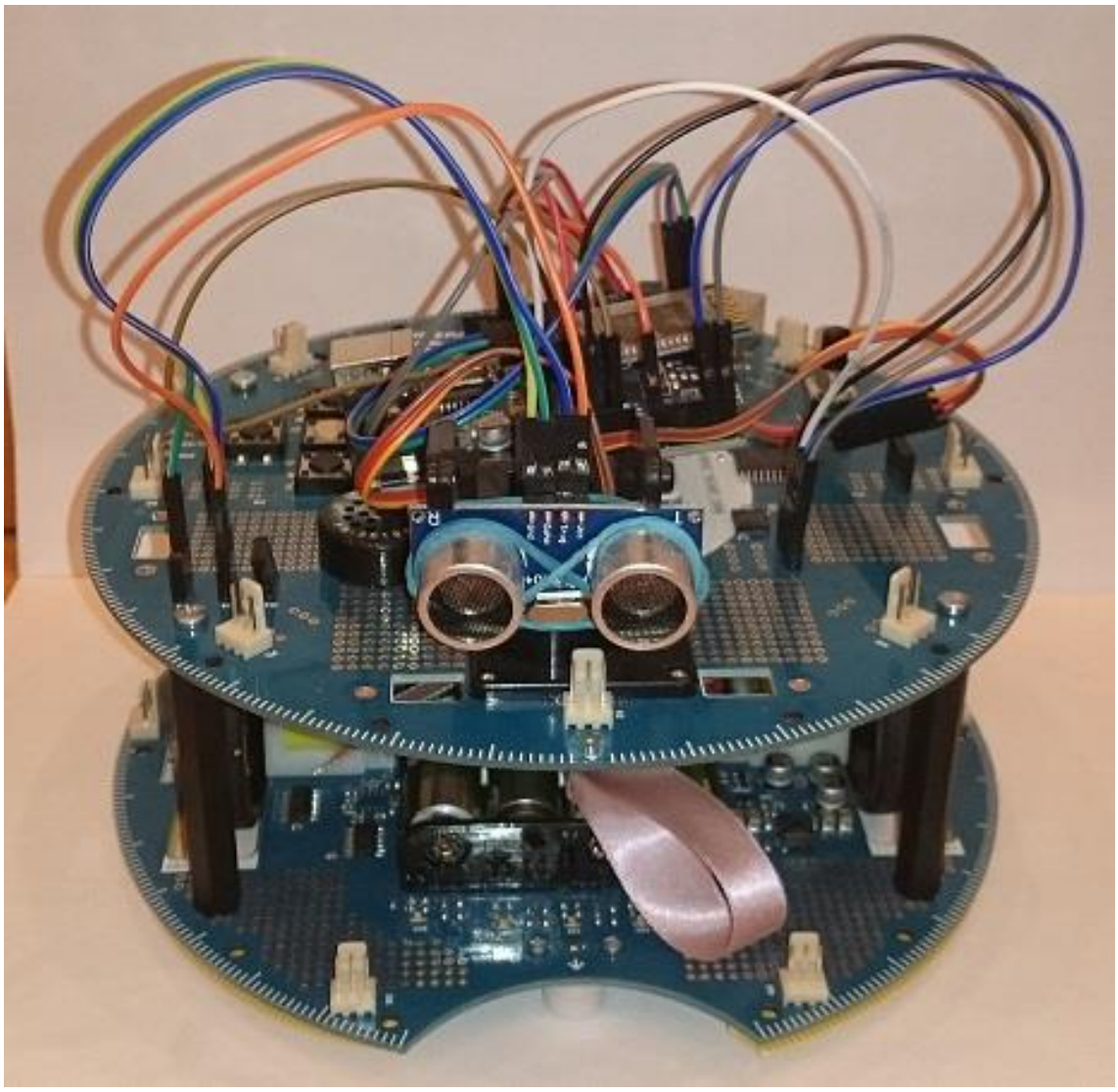

Рис 4.2. Физическое устройство системы

Для сборки данной конструкции использовались все компоненты описанные ранее.

При подключении к Bluetooth адаптеру модуль беспроводного соединения перестает моргать датчиком соединения. При проведении

тестирования, после успешного подключения к телефону, датчик перестал моргать, что свидетельствует о верной реализации подключения.

4.2. Тестирование системы изучения лабиринта

Тестирование данной системы заключается в проверке данных присылаемых с мобильной системы движения, поведения робота в лабиринте, а также адекватной реакции робота на команды.

Для проверки корректности данных использовалось стороннее приложение терминал, поскольку основное реализованное приложение не содержит данного функционала.

При включении робота, поведение соответствовало ожидаемому – робот ожидал сопряжения со смартфоном и поступления дальнейшей команды. Далее было запущено терминальное приложение, позволившее подключиться к мобильному устройству и считывать получаемые по Bluetooth соединению данные.

При тестировании поведения робота, выяснилось, что датчик расстояния имеет достаточно большую погрешность и это сказывается на поведении робота. Иногда возникают случаи, когда необходимо выяснить расстояние до «углового» препятствия, где время за которое «возвращается» сигнал не соответсвует действительному, поскольку отраженный сигнал не попал в датчик и он определил расстояние как недостижимое. В таком случае робот может повернуть и пропустить проход или же не «увидеть» препятствие. Существует 2 решения данной проблемы – заменить ультразвуковой датчик на лазерный (или инфракрасный) или производить поворот на незначительное расстояние, чтобы можно было найти препятствие. Первый вариант является наиболее простым решением, второй же требует некоторых затрат на изменение алгоритма. Однако при реализации поворота решение станет более универсальным с любым используемым датчиком.

Ожидаемые результаты должны показывать строку, содержащую координаты робота в лабиринте, разделенные знаком «|», и начинающуюся со знака «#». Такой формат сообщения позволил отсеять лишнюю информацию, которая могла попадать в потоке данных.

Полученные результаты представлены на рис.4.3.

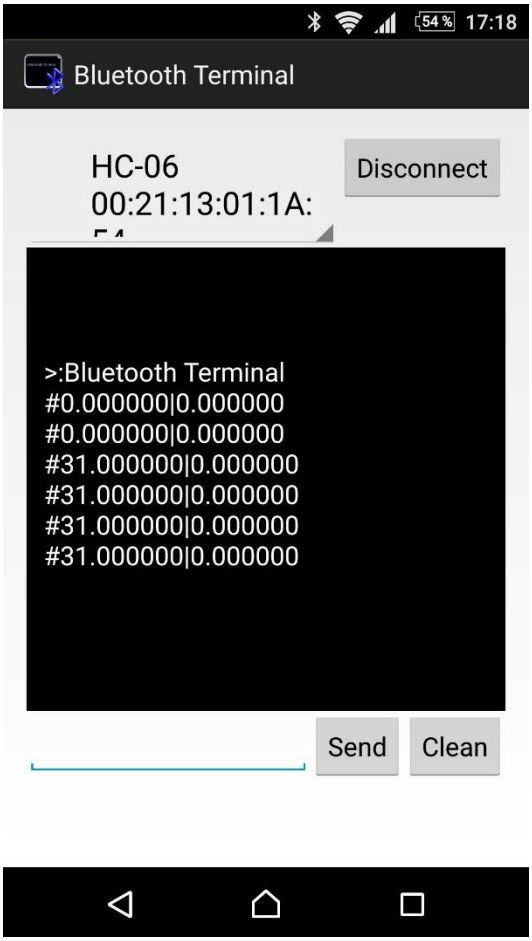

Рис.4.3. Тестирование передачи данных

Полученные данные соответствуют ожидаемым. Проанализировав полученные данные можно определить движение робота – при неизменности координат робот не перемещается, при изменении любой из координат робот производит движение в ту или иную сторону по соответствующим осям. Данная логика была заложена в построение карты лабиринта по известным точкам.

Также, было замечено, что габариты реализованной системы достаточно велики, что является существенным недостатком при изучении лабиринта с

проходами небольшого размера. Хотя система и предусматривает проверку на проходимости, но все же некторые участки остаются не исследованными.

В целом, система работает верно, за исключением незначительных недочетов в аппаратной части системы, которые можно устранить без значительных усилий.

4.3. Тестирование алгоритма нахождения кратчайшего пути

Прежде чем использовать реализованный алгоритм для решения задачи, необходимо провести анализ адекватности вычислений. Для тестирования был создан класс Main, для вызова консольного приложения, который содержит вызов функции нахождения кратчайшего пути.

Проведем расчеты вручную. Будем использовать граф представленный на рис.4.4.

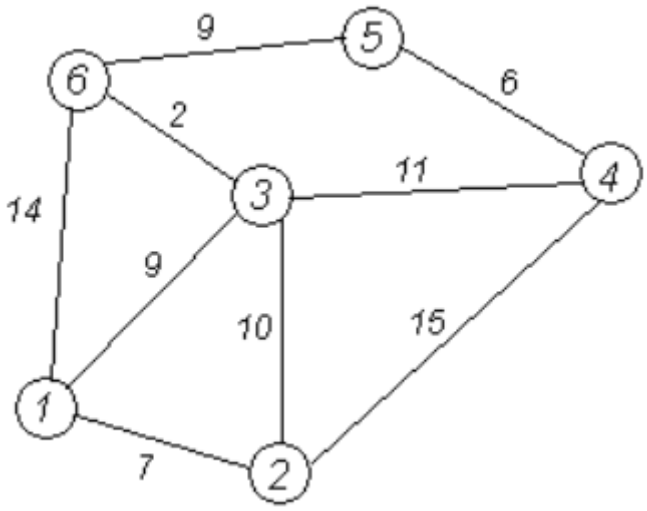

Рис.4.4. Исходный граф

Проставим метки всем вершинам равные бесконечности (гораздо большему числу, чем суммарно возможное расстояние для всего графа). Начнем с вершины 1, так как она имеет наименьшую метку равную 0 (начальная вершина имеет метку с расстоянием 0). Проверим всех соседей данной вершины – если расстояние равное сумме метки текущей вершины и

веса связи меньше метки соседней вершины, то следует изменить значение метки соседа. Данную процедуру следует повторить до тех пор пока все связи не будут просмотрены и все вершины не отмечены как посещенные.

После того как алгоритм завершил свою работу, метки всех вершин выглядели так как показано на рис.4.5.

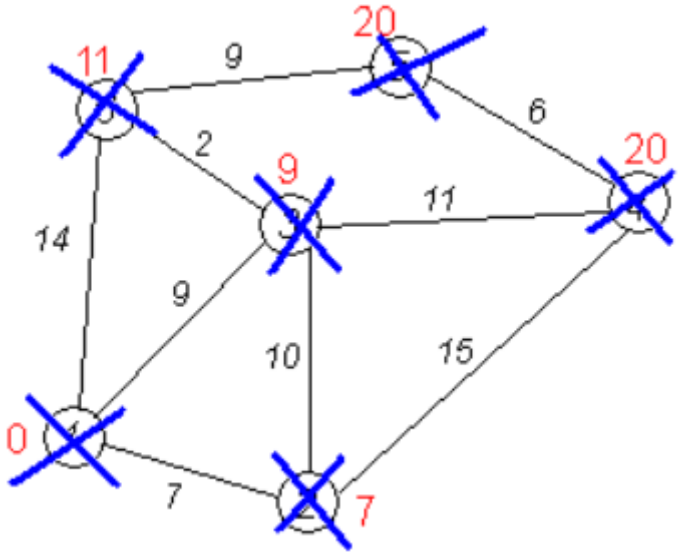

Рис.4.5. Завершение алгоритма

Теперь известны расстояния до любой вершины от вершины номер 1.

Для проверки адекватности вычислений предлагается использовать тот же граф с теми же условиями.

После введения данных в программу, был запущен алгоритм, результаты вычислений которого приведены на рис.4.6.

```
Tree: [-1, 0, 0, 2, 5, 2,
Dist: [0.0, 7.0, 9.0, 20.0, 20.0, 11.0,
```

Рис.4.6. Результаты программной реализации алгоритма

При сравнении результатов с расчетами видно, что алгоритм дает правильные значения, что свидетельствует об адекватности вычислений.

Данный алгоритм удобен тем, что он не зависит от исследуемой системы – для расширения возможостей системы это является несомненным преимуществом, изменения могут быть применены лишь к физическому устройству и принципу позиционирования, но не к математической модели.

Даже с учетом недочетов в аппаратной части системы, данный алгоритм сможет выдать кратчайший путь в известном лабиринте.

4.4. Тестирование системы в комплексе

После тестирования всех разработанных модулей взятых в отдельности, необходимо провести комплексное тестирование всей системы в совокупности.

Для этого необходимо разработать схему лабиринта. Поскольку «правило одной руки» накладывает ограничения на лабиринт, то необходимо исключить участки лабиринта, стенки которых не имеют связи с наружными его стенами. Конечно, это не говорит о том, что такой лабиринт не проходим используя предложенный алгоритм, это свидетельствует лишь о том, что данный участок не будет изучен.

На рисунке 4.7 представлен разработанный лабиринт для прохождения.

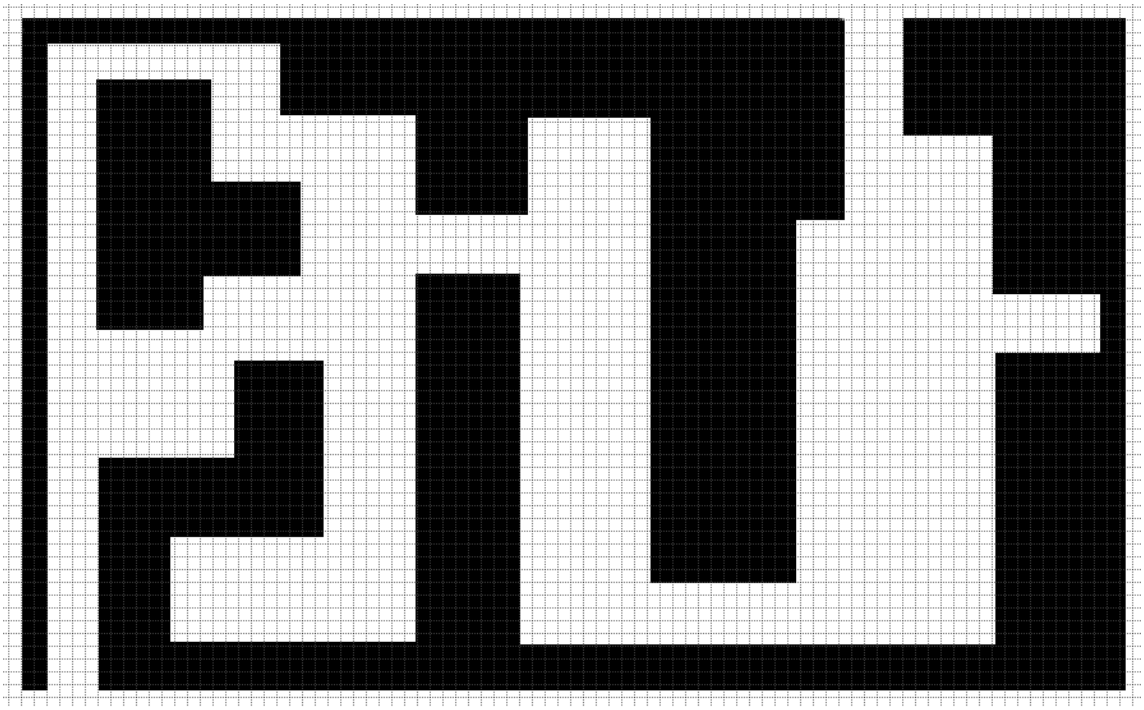

Рис.4.7. Лабиринт для прохождения

Как мы можем видеть, лабиринт имеет циклический элемент, но он может быть изучен, если не остановить алгоритм при обнаружении выхода.

Для изучения данного лабиринта было выбрано правило «правой руки». На рисунке 4.8 приведена траектория, по которой должен пройти робот при изучении лабиринта.

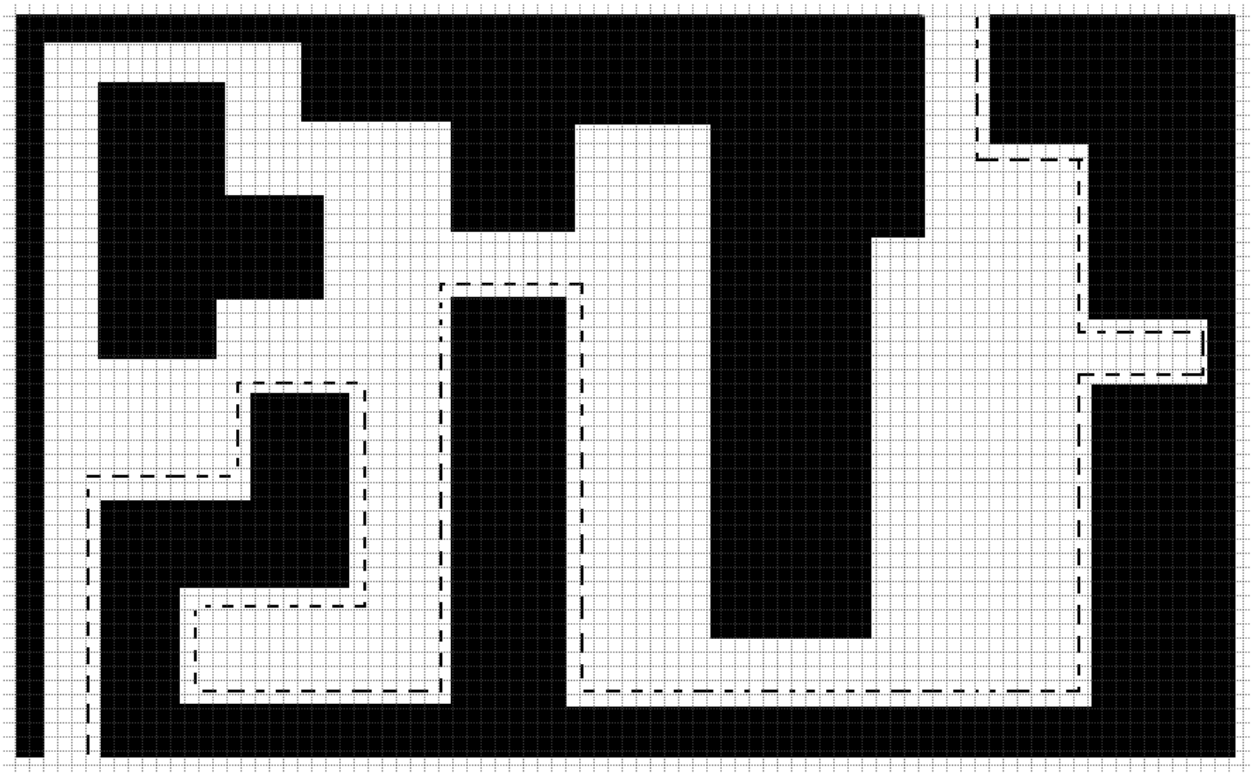

Рис.4.8. Траектория движения робота

В конце своего пути робот обнаружил выход. В этом случае пользователь выбирает дальнейшие действия – продолжить изучение или оптимизировать изученный путь. В первом случае робот изучит оставшийся маршрут и это может дать лучший результат при оптимизации пути. Во втором случае робот вернется в исходную точку по неполному оптимизированному пути. Для тестирования работоспособности системы предлагается выбрать первый вариант.

При изучении роботом лабиринта, он отправляет на сервер «контрольные точки» - места, где робот нашел препятствие или проход справа. Поскольку оптимизированный путь «отсекает» тупики и лишние маршруты, то, для дальнейшего построения графа, «контрольные точки» сохраняются в некоторой окрестности, что позволяет избежать дублирования точек и возникновения лишних связей.

На рисунке 4.9. представлен граф, который построила система после изучения лабиринта.

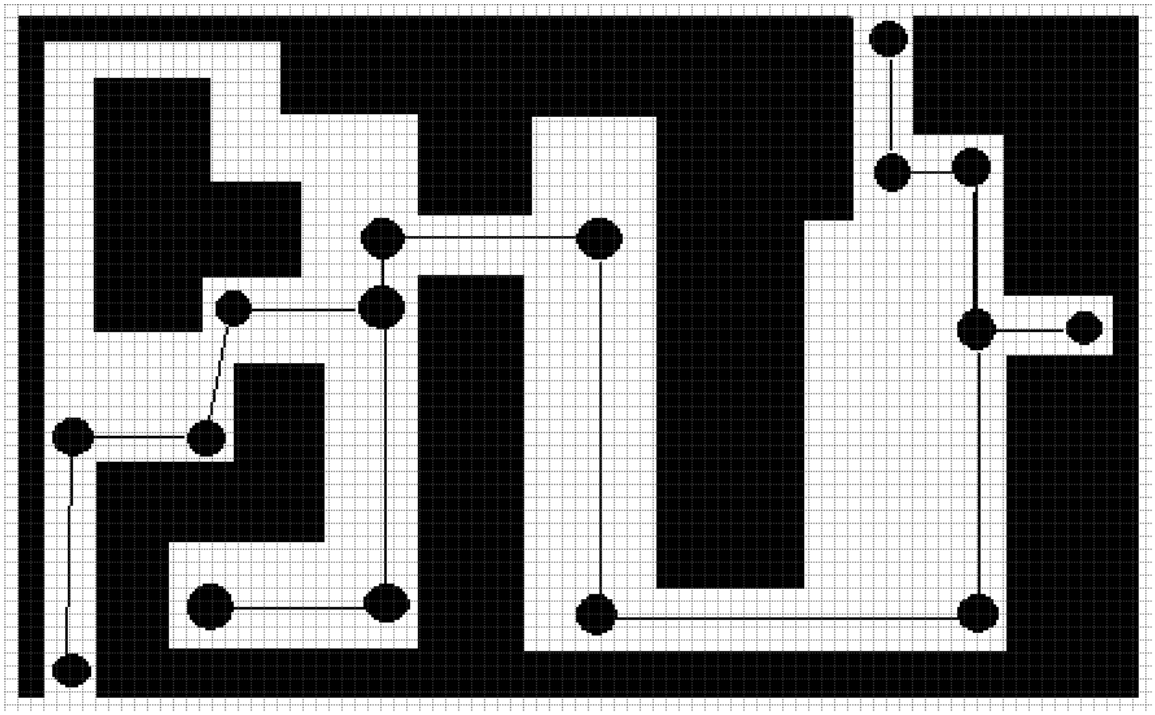

Рис.4.9. Граф изученного пути

После проведения оптимизации граф принял вид представленный на рисунке 4.10.

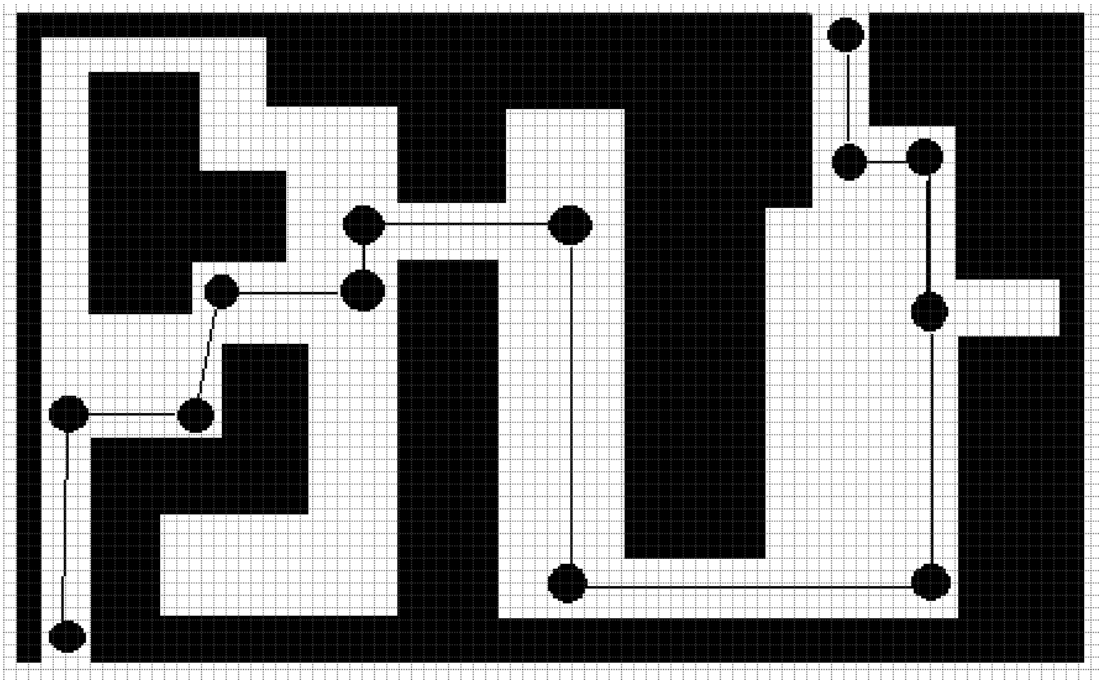

Рис. 4.10. Оптимизированный граф

В целом, по известным, точкам путь построен верно.

После проведения вышеизложенных расчетов необходимо проверить работу самой системы. После включения и успешного соединения мобильной вычислительной системы с роботом, робот прошел по ожидаемой траектории, нашел выход и передал данные на мобильную систему, система запросила дальнейшие действия, вычислила оптимальной маршрут по заданным точкам и робот вернулся в исходную точку уже по оптимизированному маршруту.

На рисунке 4.10 можно увидеть, что граф может быть оптимизирован лучше – предпоследняя точка может быть соединена напрямую с точкой, без привязки к стене. Это является недостатком системы и связано с тем, что робот проходит лишь по «стене» и не изучает внутреннее пространство. Для решения этой проблемы можно добавить проверки в коде и дополнительные проверки физических препятствий.

После проведения комплексного тестирования была подтверждена правильность реализованных алгоритмов, были выявлены недостатки системы и предложен возможный вариант решения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была создана мобильная система для изучения лабиринта, реализован алгоритм нахождения оптимального пути в лабиринте, а также проведено тестирование созданной системы.

В ходе анализа реализованной мобильной системы были выявлены некоторые недостатки системы, связанные с используемой аппаратной платформой, однако построенная математическая модель может использоваться в дальнейших иссследованиях. К примеру, расширив систему локальных координат, используя Декартову систему в пространстве, а также расширив функциональность мобильной системы новыми датчиками и возможностью перемещения не только на поверхности, но и в пространстве (к примеру, используя квадрокоптер), можно применить те же методы для изучения лабиринта не только в двухмерном пространстве, но и в трехмерном.

В данной работе были выполнены следующие задачи:

1. были изучены существующие решения для построения карты лабиринта;

2. было проведено сравнение и определение необходимых аппаратных модулей;

3. была построена математическая модель для построения карты;

4. был проведен анализ решений для нахождения кратчайшего пути, а также определен алгоритм подходящий для лабиринта;

5. был собран прототип мобильной системы для тестирования математической модели;

6. ОС Android была определена в качестве вычислительного ресурса;

7. было реализовано android-приложение и база даных.

После проведения тестирования, выяснили, что для более точного построения лабиринта следует применять лазерные дальномеры, поскольку у них меньшая погрешность в показателях. Также для полной автономности можно использовать микроконтроллер с большим объемом оперативной памяти, например, Arduino Mega или одноплатный компьютер Raspberry PI, а также использовать карту памяти для хранения уже пройденных маршрутов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Как работает ультразвуковой дальномер [Электронный ресурс] – URL: http://robotosha.ru/electronics/how-works-ultrasound-meter.html (дата обращения: 13.05.2018)
2. Среда разработки Arduino [Электронный ресурс] – URL: http://arduino.ru/Arduino_environment (дата обращения: 05.05.2018).
3. Arduino IDE [Электронный ресурс] – URL: https://www.arduino.cc/en/Main/Software (дата обращения: 10.04.2018)
4. Android Studio [Электронный ресурс] – URL: https://developer.android.com/studio/index.html?hl=i (дата обращения: 10.04.2018
5. Описание основных функций языка Arduino [Электронный ресурс] – URL: http://www.freeduino.ru/arduino/lang.html (дата обращения: 07.05.2018).
6. Организация беспроводной связи по стандарту BLUETOOTH [Электронный ресурс] – URL: http://taketop.ru/articles/informatika/progobesp/besprov-set (дата обращения: 29.04.2018
7. Алгоритм Дейкстры. Электронный ресурс. Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры (дата обращения: 12.06.2018)
8. Лазерные приборы для измерения высоты и расстояний: [Электронный документ] - (http://mylektsii.ru/1-87002.html) . (дата обращения: 12.06.2018)
9. Задача о кратчайшем пути. Электронный ресурс. Режим доступа: https://ru.wikipedia.org/wiki/Задача_о_кратчайшем_пути.(дата обращения: 12.06.2018)

10. Алгоритм Флойда-Уоршелла. Электронный ресурс. Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Флойда-Уоршелла (дата обращения: 12.06.2018)
11. Алгоритм Ли (Волновой алгоритм). Электронный ресурс. Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_Ли (дата обращения: 12.06.2018)
12. Основы программирования для Android [Электронный ресурс] – URL: http://metanit.com/java/android/2.1.php (дата обращения: 10.05.2018).
13. Разработка под Android [Электронный ресурс] – URL: http://developer.alexanderklimov.ru/android/ (дата обращения: 10.05.2018)
14. Машина-робот на базе Arduino [Электронный ресурс] – URL: https://habrahabr.ru/sandbox/82341/ (дата обращения: 09.05.2018
15. Сайт вопросов и ответов для программистов [Электронный ресурс] – URL: http://stackoverflow.com/ (дата обращения: 14.05.2018)
16. Беспроводные технологии передачи данных для создания систем управления и персональной информационной поддержки [Электронный ресурс] – URL: http://window.edu.ru/resource/177/56177/files/62331e1-st18.pdf (дата обращения: 29.04.2018)
17. *Boxall Lohn.* Arduino Workshop / Lohn Boxall. ISBN-13: 978-1-59327-448-1
18. *Dijkstra, E. W.* A note on two problems in connection with graphs / E. W. Dijkstra// Numerische Mathematik. - 1959. - 1. - С. 269-271
19. *Fox, D.* The Dynamic Window Approach to Collision Avoidance [Текст]/ D. Fox, W. Burgard, S.Thrun // IEEE Robotics and Automation Magazine. - 1997. - С. 23– 33.
20. *Friesen Jeff.* Learn Java for Android Development / Jeff Friesen. ISBN-13: 978-1-4302-5722-6
21. *Kamon, I.* A new Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots [Текст]/ I. Kamon, E. Rivlin, E. Rimon//

Minneapolis: - Robotics and Automation. Proceedings on IEEE International Conference - 1995.T1.-C. 429-435.

22. *McRoberts Michael*. Beginning Arduino / Michael McRoberts. ISBN-13: 978-1-4302-3420-7
23. *Schmidt Maik*. Arduino.A Quick-Start Guide / Maik Schmidt. ISBN-13: 978-1-934356-66-1
24. *Zapata Belen Cruz*. Android Studio Application Development / Belen Cruz Zapata. ISBN 978-1-78328-527-3
25. Android. Программирование для профессионалов. 2-е изд / Харди Б., Филлипс Б., Стюарт К., Марсикано К. — СПб.: Питер, 2016. — 640 с.: ил. — (Серия «Для профессионалов»).
26. *Дейтел П.* Android для разработчиков / П. Дейтел, Х. Дейтел, А. Уолд. СПб.: Питер, 2016. — 512 с.: ил. — (Серия «Библиотека программиста»).
27. *Маслов В.А*., Финогеев А.А., Финогеев А.Г. Известия ВУЗов. Методика идентификации и событийного управления мобильными устройствами на основе технологии bluetooth: 2-е изд, 2008. – 65 с.
28. *Петин В. А*. Проекты с использованием контроллера Arduino / В.А. Петин. - СПб.: БХВ-Петербург, 2014. - 400 с.
29. *Платонов, А.К*. Метод потенциалов в задаче выбора пути: история и перспективы [Текст]/ А.К. Платонов, М.А. Колганов , А.А. Кирильченко. – М.: Препринты ИПМ им. М. В. Келдыша, 2001, 4 с.
30. *Тусеева, И.Б.* Алгоритм динамического окна для навигации автономных подводных аппаратов [Текст]/ И.Б. Тусеева, Д.Б. Тусеева, Юн-Ги Ким// Искусственный интеллект и принятие решений. - 2013. - №3. - С. 66-67

ПРИЛОЖЕНИЕ 1. Листинг Android-приложения

```
package com.example.dasha.bluetooth;

import android.annotation.SuppressLint;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;

import com.example.dasha.bluetooth.Entities.Connection;
import com.example.dasha.bluetooth.Entities.GraphPoint;
import com.example.dasha.bluetooth.Entities.Path;

import java.io.IOException;
import java.util.List;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {
    Button Start;
    Button Return;

    public TextView MyText;

    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    final String LOG_TAG = "myLogs";
    private static String MacAddress = "00:21:13:01:1A:54"; // MAC-адрес БТ
модуля
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

    private ConnectedThred MyThred = null;

    private GraphPoint parentPoint;

    private Path pathInfo;

    public MazeDatabase database =
MazeDatabase.getMazeDatabase(MainActivity.this);

    private int globalNumberPoint=0;

    CustomDrawableView mCustomDrawableView;

    private String startMove="s";

    Handler h;

    @SuppressLint("HandlerLeak")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //mCustomDrawableView = new CustomDrawableView(this);
```

```
        //setContentView(mCustomDrawableView);


        Start = (Button) findViewById(R.id.btnStart);
        Return = (Button) findViewById(R.id.btnReturn);


        MyText = (TextView) findViewById(R.id.statusText);

        btAdapter = BluetoothAdapter.getDefaultAdapter();

        final int path=1;

        if(btAdapter!=null) {
            pathInfo=new Path();
            pathInfo.setPathName("testPath");

            database.pathDao().insertPath(pathInfo);


pathInfo.setPathId(database.pathDao().getPathIdByName(pathInfo.getPathName())
);

            parentPoint=new
GraphPoint(pathInfo.getPathId(),globalNumberPoint, 0.0f, 0.0f);
            database.graphPointDao().insertPoint(parentPoint);
            globalNumberPoint++;


            Start.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    MyThred.sendData(startMove);
                    MyText.setText("Отправлены данные: начать движение");
                }
            });

            Return.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    MyText.setText("Выполняется поиск оптимального пути...");
                    List<GraphPoint>
allPoints=database.graphPointDao().getPointsFromPath(pathInfo.getPathId());
                    Graph g=new Graph(allPoints.size());
                    for (GraphPoint p:allPoints
                         ) {
                        List<Connection>
allConnectionsForPoint=database.connectionDao().getConnectionForPoint(p.getPo
intId());
                        for (Connection c:allConnectionsForPoint
                             ) {
                            GraphPoint
pPoint=database.graphPointDao().getPointById(c.getParentId());
                            GraphPoint
cPoint=database.graphPointDao().getPointById(c.getChildId());
                            float weight= (float)
Math.sqrt(Math.pow(pPoint.getX()-cPoint.getX(), 2)+Math.pow(pPoint.getY()-
cPoint.getY(), 2));

g.addArc(pPoint.getPointNumber(),cPoint.getPointNumber(),weight);
                        }
                    }
                    GraphService gService=new GraphService(g);
                }
```

```
            });

            h = new Handler() {
                public void handleMessage(android.os.Message msg) {
                    byte[] readBuf = (byte[]) msg.obj;
                    String strIncom = new String(readBuf, 0, msg.arg1);
                    String[] data = strIncom.substring(1).split("|");
                    if(data.length>1){
                        GraphPoint newPoint=new
GraphPoint(pathInfo.getPathId(),globalNumberPoint,Integer.parseInt(data[0]),
Integer.parseInt(data[1]));

if(database.graphPointDao().checkIfPointExists(newPoint.getX(),
newPoint.getY())==0) {
                            database.graphPointDao().insertPoint(newPoint);
                        }

newPoint.setPointId(database.graphPointDao().getPointIdByCoord(newPoint.getX(
),newPoint.getY()));

if(database.connectionDao().checkIfConnectionExists(parentPoint.getPointId(),
newPoint.getPointId())==0){
                            Connection newCon=new
Connection(parentPoint.getPointId(),newPoint.getPointId());

database.connectionDao().insertConnection(newCon);
                        }
                        parentPoint=newPoint;
                    }else{
                        MyText.setText("Лабиринт изучен.");
                        Start.setVisibility(View.INVISIBLE);
                        Return.setVisibility(View.VISIBLE);
                    }
                }
            };
        }
        MyText.setText("Bluetooth не включен.");
    }

    @Override
    public void onResume() {
        super.onResume();
        if(btAdapter!=null) {
            BluetoothDevice device = btAdapter.getRemoteDevice(MacAddress);
            try {
                btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
            } catch (IOException e) {
                MyError("Fatal Error", "В onResume() Не могу создать сокет: "
+ e.getMessage() + ".");
            }

            btAdapter.cancelDiscovery();
            try {
                btSocket.connect();
            } catch (IOException e) {
                try {
                    btSocket.close();
                } catch (IOException e2) {
                    MyError("Fatal Error", "В onResume() не могу закрыть
сокет" + e2.getMessage() + ".");
                }
            }
```

```
            MyText.setText("Connected");
            MyThred = new ConnectedThred(btSocket, h);
            MyThred.start();
        }
    }

    @Override
    public void onPause() {
        super.onPause();

        Log.d(LOG_TAG, "...In onPause()...");

        if (MyThred.status_OutStrem() != null) {
            MyThred.cancel();
        }

        try     {
            btSocket.close();
        } catch (IOException e2) {
            MyError("Fatal Error", "В onPause() Не могу закрыть сокет" +
e2.getMessage() + ".");
        }
    }

    private void MyError(String title, String message){
        Toast.makeText(getBaseContext(), title + " - " + message,
Toast.LENGTH_LONG).show();
        finish();
    }
}
```

ПРИЛОЖЕНИЕ 2. Листинг программы на Arduino UNO

```
#include <Wire.h>
#include <Servo.h>

const int UNO_ADDRESS=8;
const int ROBOT_ADDRESS=9;
const int SERVO_PIN=A0;
char character;
String message = "";
int i=1;

Servo Servo1;

void setup() {
  // Start bluetooth serial
  Serial.begin(9600);
  // Begin wire connection
  Wire.begin(UNO_ADDRESS);
  Wire.onReceive(receiveEvent);
  Servo1.attach(SERVO_PIN);
}

void receiveEvent(int bytes) {
  // Get the message when available
  message="";
  while(Wire.available()) {
    char c = Wire.read();
    //Serial.println(c);
    message += c;
  }
```

```
  if(message[0]=='#'){
  Serial.println(message);
  }
  if(message[0]=='l')
  {
    Servo1.write(179);
    delay(1000);
    message="";
  }
  if(message[0]=='f')
  {
    Servo1.write(90);
    delay(1000);
    message="";
  }
}

void loop() {
  while(Serial.available()) {
    character = Serial.read();
    if(character == 's') {
      // Write the character to Robot Control
      Wire.beginTransmission(ROBOT_ADDRESS);
      Wire.write(character);
      Wire.endTransmission();
    }
   }
}
```

ПРИЛОЖЕНИЕ 3. Листинг программы на Arduino Robot

```
#include <Ultrasonic.h>
#include <ArduinoRobot.h>
#include <Wire.h>
const int UNO_ADDRESS=8;
const int ROBOT_ADDRESS=9;
//time begin end
unsigned long startPointTime;
unsigned long endPointTime;
// This will keep the message we get
String message = "";
int echoPin = D5;
int trigPin = D4;

//distance forward and left
float forwardDistance=0;
float leftDistance=0;

//coordinates
float X=0;
float Y=0;

//direction
float global_X=0;
float global_Y=1;

void setup() {
  // Start Robot modules
  Robot.begin();
  Robot.beginSpeaker();
```

```
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Create a receive event
  Wire.begin(ROBOT_ADDRESS);
  Wire.onReceive(receiveEvent);
}

void receiveEvent(int bytes) {
  // Get the message when available
  message="";
  while(Wire.available()) {
    char c = Wire.read();
    Serial.println(c);
    message += c;
  }
  if(message[0]=='s'){
   while(true){
    //Повернуть сервопривод вперед(90)
    Wire.beginTransmission(UNO_ADDRESS);
    Wire.write('f');
    byte result=Wire.endTransmission();
    delay(2000);
    //получить расстояние впереди
    float currentDistance=getDistance()-5.0;
    //получить время которое можно ехать вперед
    float timeToPass=currentDistance/31.0;
```

```
  delay(30);

  //повернуть сервопривод влево (179)
  Wire.beginTransmission(UNO_ADDRESS);
  Wire.write('l');
  result=Wire.endTransmission();
  delay(1100);

  //получить расстояние слева
  leftDistance=getDistance();
  //начальный отсчет времени
  startPointTime=micros()/1000000;

  currentDistance=getDistance();
  Robot.updateIR();
  //ехать вперед пока есть возможность(впереди нет препятствий и слева нет прохода)
  while((micros()/1000000)<(startPointTime+timeToPass) && (currentDistance>=(leftDistance-5) && currentDistance<=(leftDistance+5)))
  {
    Robot.motorsWrite(127,127);
    currentDistance=getDistance();
    if(Robot.IRarray[3]==0)
    {
      break;
    }
    Robot.updateIR();
  }
  //остановить движение с тормозным путем
  Robot.motorsWrite(0,0);
```

```
//конечный отсчет времени
endPointTime=micros()/1000000;
delay(1000);

//получить пройденное время
float realTime=endPointTime-startPointTime;

//получить пройденное расстояние
float passedDistance=realTime*31.0;

//корректировка координат
X=X+global_X*passedDistance;
Y=Y+global_Y*passedDistance;

//Отправить данные на смартфон
Wire.beginTransmission(UNO_ADDRESS);
Wire.print('#'+String(X,6)+'|'+String(Y,6));
result=Wire.endTransmission();

//проверить стену прямо
forwardDistance=getDistance();

//проверить стену слева
Wire.beginTransmission(UNO_ADDRESS);
Wire.write('l');
result=Wire.endTransmission();
delay(2000);
//получить расстояние слева
leftDistance=getDistance();
```

```
int turns=0;

//выполнить не более 2 поворотов вправо
while(turns<2){
  //повернуть робота
  if(leftDistance>30){
    Robot.motorsWrite(-80,80);
    delay(1000);
    Robot.motorsStop();
    changeGlobals();
    turns=2;
  }else if(forwardDistance<5){
    Robot.motorsWrite(80,-80);
    delay(1000);
    Robot.motorsStop();
    changeGlobals();
    turns=turns+1;
  }

  if(turns==1){
  //проверить стену слева
  leftDistance=getDistance();

  //проверить стену прямо
  Wire.beginTransmission(UNO_ADDRESS);
  Wire.write('f');
  result=Wire.endTransmission();
  delay(2000);
  //получить расстояние слева
```

```
      forwardDistance=getDistance();
      }
    }
   }
  }
}

void loop() {

}
//изменить указатели направления робота
void changeGlobals(){
  if(global_X==0){
    if(global_Y==1){
      global_X=-1;
      global_Y=0;
    }else{
      global_X=1;
      global_Y=0;
    }
  } else if(global_Y==0){
    if(global_X==1){
      global_X=0;
      global_Y=1;
    }else{
      global_X=0;
      global_Y=-1;
    }
  }
}
```

```
float getDistance() {

  int duration;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  float distance_cm = (duration/2) / 29.1;
  return distance_cm;
}
```