

Правительство Санкт-Петербурга
Комитет по образованию
Государственное бюджетное общеобразовательное учреждение
“Президентский физико-математический лицей №239”

ВСЕРОССИЙСКАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Технология. Профиль “Робототехника”

ПРОЕКТ

РОБОТ “НАЯТ”

Работу выполнил

Балакирский Аркадий
учащийся 11 класса

Руководители:

Романько Павел Николаевич
Устинов Илья Дмитриевич
Викторов Борис Викторович
педагоги дополнительного образования

Санкт-Петербург

2023

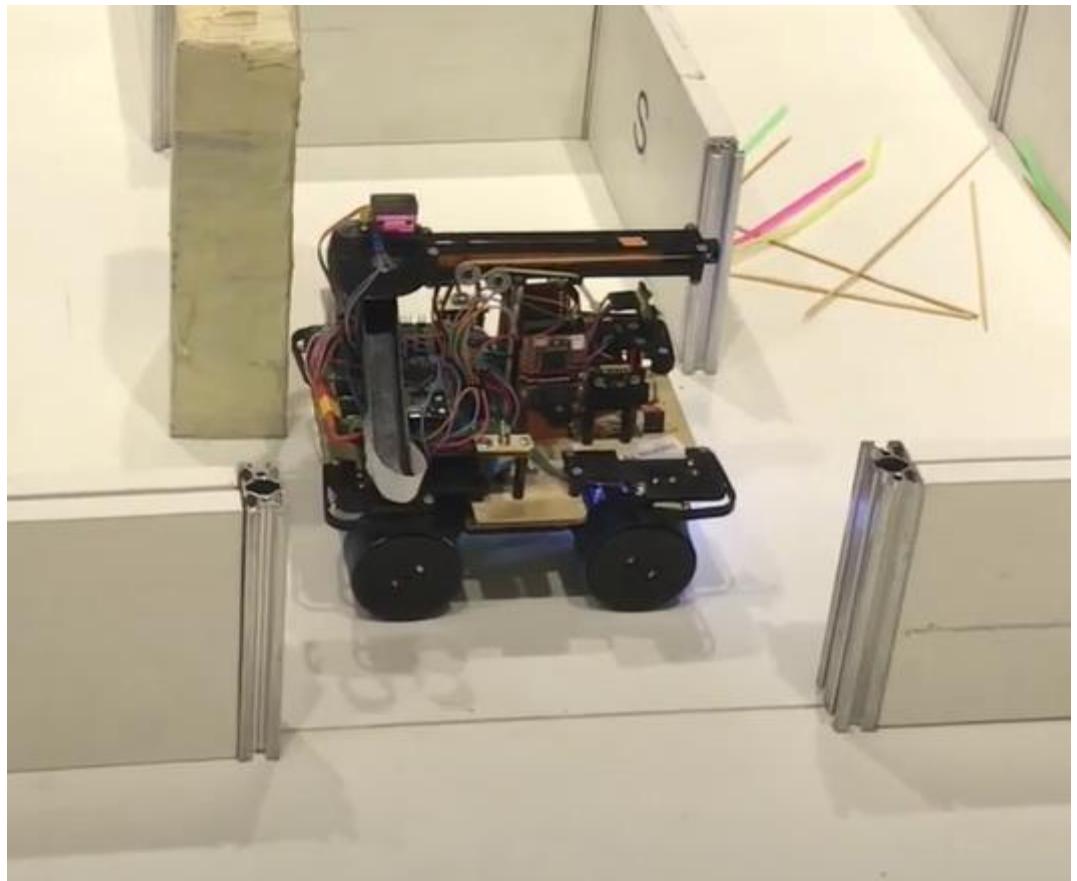
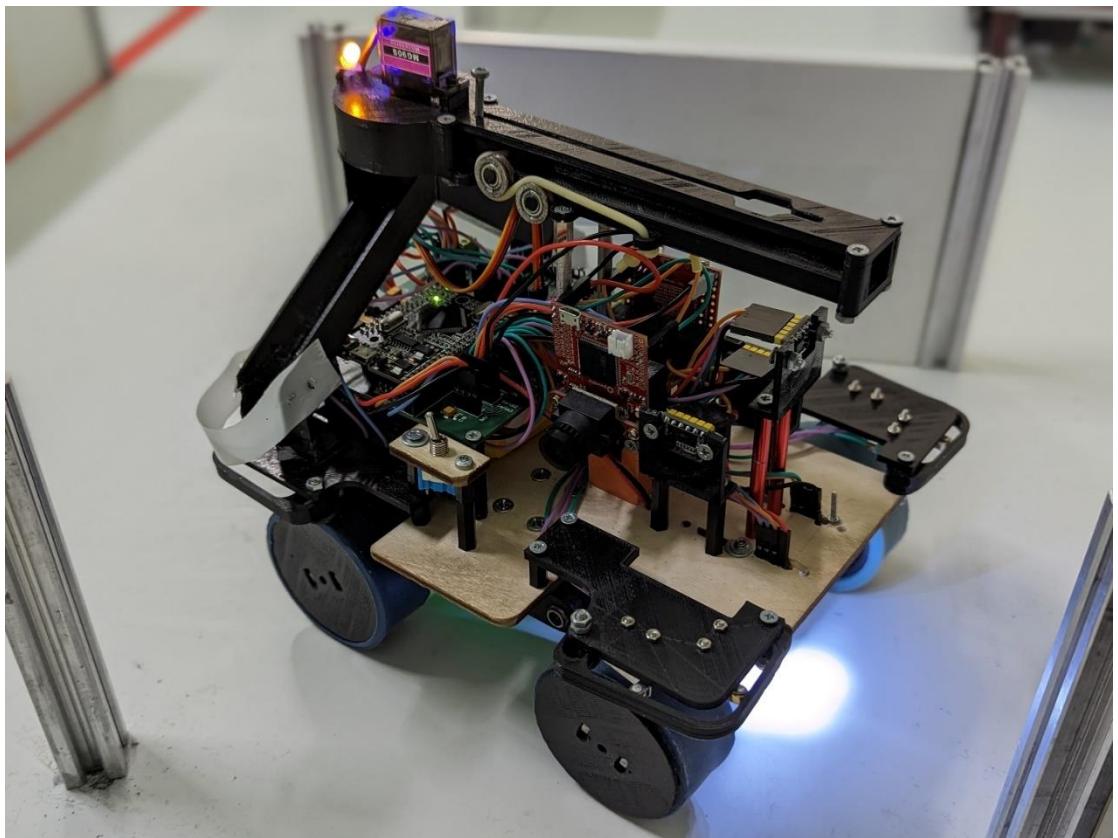


Рисунок 1 – Робот “Hayat”

СОДЕРЖАНИЕ

1 ПОИСКОВО-ИССЛЕДОВАТЕЛЬСКИЙ ЭТАП	5
1.1 Актуальность	5
1.2 Цель проекта.....	6
1.3 Задачи проекта.....	7
1.4 Анализ аналогов	7
1.5 Техническое задание.....	9
1.6 Концепция робота	10
2 КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКИЙ ЭТАП	10
2.1 Подбор материалов и проектирование конструкции.....	10
2.2 Список использованного электронного оборудования	11
2.3 Структурная и принципиальная схемы робота	13
2.4 Выбор сред разработки и языков программирования. Общие сведения.....	14
2.5 Компьютерное зрение.....	14
2.6 Алгоритм обхода лабиринта по левой и правой руке.....	17
2.7 Алгоритм поиска в ширину (BFS).....	19
2.8 Алгоритм Дейкстры	23
2.9 Сравнение различных алгоритмов поиска пути	23
2.10 Хранение графа лабиринта	24
2.10 Структура программы основного микроконтроллера	26
2.11 Распознавание цветов на полу лабиринта	28
2.12 Проезд вперед между двумя соседними ячейками.....	29
2.13 Отладка и модификации	31
3 ЗАКЛЮЧИТЕЛЬНЫЙ ЭТАП	34

3.1 Креативность и новизна проекта.....	34
3.2 Результат.....	34
4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35
ПРИЛОЖЕНИЕ А	36
ПРИЛОЖЕНИЕ Б.....	37

1 ПОИСКОВО-ИССЛЕДОВАТЕЛЬСКИЙ ЭТАП

1.1 Актуальность

В наше время происходит достаточно много природных и техногенных катастроф по всему миру. Поэтому одним из важнейших направлений современной робототехники является экстремальная робототехника, которая занимается разработкой роботов, способных работать в экстремальных природных и производственных условиях (например, на рисунке 2 изображено полуразрушенное промышленное предприятие), полностью заменяя человека.



Рисунок 2. Полуразрушенное здание

Спасение людей с помощью роботов – актуальная задача современности. Для привлечения молодых людей, студентов и школьников, к решению этих актуальных задач, по всему миру проводятся робототехнические соревнования, моделирующие реальную ситуацию спасения пострадавших из загроможденной после катастрофы местности, для этого собирают полигоны со сложным рельефом, а перед участниками ставится задача создать такую робототехническую систему, которая могла бы их исследовать и преодолеть, доставить грузы или выяснить нахождения объектов. Примером таких соревнований является чемпионат RoboCup. RoboCup — это научный и

культурный проект по продвижению искусственного интеллекта, робототехники и других связанных областей науки и техники посредством организации и проведения робототехнических соревнований. Участвуя в таких состязаниях, школьники и студенты всего мира, приобщаются к решению актуальных, еще до конца не решенных, проблем экстремальной робототехники.

1.2 Цель проекта

Было принято решение создать робота, с помощью которого можно будет отлаживать и тестировать алгоритмы для навигации по лабиринту (замкнутому пространству). Эти алгоритмы можно будет использовать для создания робота, способного оказывать помощь людям уже в реальном полуразрушенном здании. Своего робота я решил сделать на основе регламента соревнований RoboCupJunior Rescue Maze. В них роботу необходимо обследовать как можно большую часть лабиринта с препятствиями: горками, лежачими полицейскими, мусором на полу лабиринта, лесенками. В лабиринте роботу нужно доставить спасательные наборы (представлены кубиками 1x1x1 см) жертвам в зависимости от их состояния (стабильное, непораженное и пораженное). Также в лабиринте могут быть расположены прямоугольные блоки, мешающие проезду робота по лабиринту. Жертвы представляют из себя метки на стенах: буквы “Н”, “С”, “У” (визуальные), а также прямоугольные квадраты разных цветов (красного, зеленого и желтого), являющихся цветовыми жертвами. Также на полу лабиринта могут располагаться черные клетки (половина проекции робота не может быть на этой клетке) и синяя клетка (робот должен остановиться на 5 с на клетке). На рисунке 3 приведен пример лабиринта для RoboCupJunior Rescue Maze:

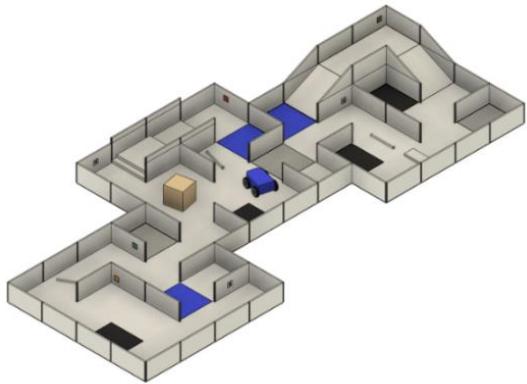


Рисунок 3. Пример лабиринта из регламента Rescue Maze

1.3 Задачи проекта

- 1 На основе анализа составить концепцию робота для Rescue Maze.
- 2 Сконструировать и изготовить робота.
- 3 Написать основную программу для езды по лабиринту.
- 4 Написать код для распознавания жертв с помощью компьютерного зрения.
- 5 Проверить работоспособность и внести необходимые улучшения в конструкцию и программу робота.

1.4 Анализ аналогов

В качестве аналогов были рассмотрены роботы других команд, участвовавших ранее в соревнованиях RoboCupJunior Rescue Maze. Информация об аналогах представлена в таблице 1.

Таблица 1 – Сравнение роботов других участников

Название	Изображение робота	Краткое описание
МК		Для навигации по лабиринту используется алгоритм BFS. На роботе закреплены 4 колеса. Имеет 6 датчиков расстояния (для выравнивания по стенкам лабиринта), 1 RGB-датчик, 2 камеры OpenMV. Для распознавания

		жертв используется собственный линейный алгоритм. Система выдачи спасательных наборов происходит с помощью сервомотора.
MZCAT		Для навигации по лабиринту используется алгоритм BFS. На роботе закреплены 4 колеса (4 мотора с энкодерами). Имеет 6 датчиков расстояния, 1 датчик освещенности, 2 камеры OpenMV. Для распознавания жертв используется собственный линейный алгоритм. Не имеет подвески. Система выдачи спасательных наборов происходит с помощью пружинного механизма
Робот хорватской команды		Для навигации по лабиринту используется алгоритм BFS. Для компьютерного зрения используются две MIPI камеры (код камеры написан с помощью собственного линейного алгоритма).

Проанализировав представленные аналоги, были сделаны выводы:

- 1 С помощью BFS возможно выполнить задание RCJ Rescue Maze.
- 2 Камеры OpenMV и MIPI являются достаточно хорошими инструментами компьютерного зрения для распознавания жертв в контексте регламента Rescue Maze.
- 3 Собственные линейные алгоритмы для компьютерного зрения работают лучше, чем встроенные алгоритмы в OpenMV.

- 4 Проходить препятствия робот может как с независимой подвеской, так и без нее.

1.5 Техническое задание

Требования к работе для Rescue Maze:

- 1 Робот должен удовлетворять регламенту RoboCupJunior Rescue Maze: исследовать неизвестный лабиринт, преодолевать препятствия (лежачие полицейские, горки, лесенки, мусор на полу), определять черный и синий цвет пола ячеек, определять тип жертвы с помощью компьютерного зрения, уметь возвращаться на стартовую ячейку после исследования всего лабиринта.
- 2 Робот не должен превышать 210 мм по ширине и длине в геометрических размерах.
- 3 Масса робота не должна превышать 3 кг.
- 4 Робот должен перемещаться между ячейками лабиринта быстрее, чем 0,1 м/с.
- 5 Алгоритм продвижения робота по лабиринту должен гарантировать прохождение роботом всех ячеек лабиринта.
- 6 Каркас робота должен состоять из фанеры или алюминия.
- 7 Потребляемая мощность не больше 46 Вт.
- 8 На роботе должны быть как минимум 2 камеры для распознавания меток слева и справа от робота.

Заметим, что у этого устройства будет 3 степени подвижности (оно перемещается по плоскости), также с помощью датчиков расстояния, цвета, камер будет обеспечено взаимодействие устройства с окружающей средой, с помощью микроконтроллера Arduino mega2560 будет обеспечена автономность. Таким образом, мое устройство будет соответствовать определению робота по ГОСТ.

1.6 Концепция робота

Пунктами, включающими в себя концепцию робота, являются:

- 1 Для анализа окружающей обстановки на роботе должно быть закреплено минимум 4 датчика расстояния (слева, спереди, справа, сзади).
- 2 Для выравнивания по стенкам лабиринта на роботе должны быть закреплены минимум 2 микропереключателя (спереди). Желательно, но необязательно также наличие двух микропереключателей сзади для выравнивания по задней стенке.
- 3 Для ровных поворотов на роботе должен быть закреплен датчик-гироскоп. Также этот датчик нужен для определения горки. То есть робот должен уметь определять свое рысканье и тангаж.
- 4 Для синхронизации моторов при езде вперед на роботе должны быть закреплены как минимум 2 датчика-энкодера.
- 5 Определение расстояния, на которое нужно проехать роботу, чтобы проехать ровно одну ячейку, должно происходить с помощью датчиков расстояния, а не энкодеров.

2 КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКИЙ ЭТАП

2.1 Подбор материалов и проектирование конструкции

Каркас робота сделан из фанеры, вырезанной на лазерном станке. Такой вариант дает достаточную свободу в процессе разработки, материал легкодоступен и при наличии мелких недоработок можно поправить после резки вручную, после внеся исправления в модель. Также достаточно много

деталей были напечатаны на 3D-принтере. Конструкция робота была спроектирована в Autodesk Inventor. Пример детали (рисунок 4):

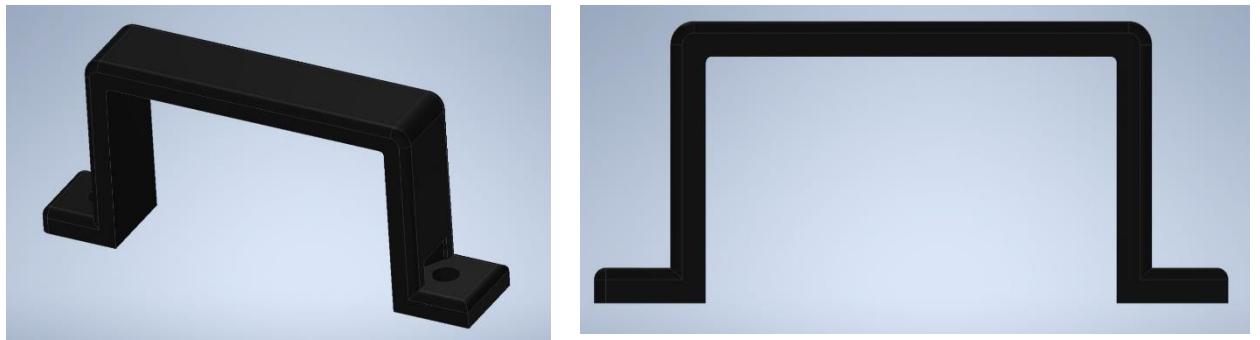


Рисунок 4. Крепление для аккумулятора, спроектированное в Autodesk Inventor

Для соединения деталей используются винты-гайки M1, M3, M4, M5; пластиковые и металлические стойки различной длины.

2.2 Список использованного электронного оборудования

В таблице 2 приведено описание электронных компонентов, закрепленных на роботе.

Таблица 2 – Описание электронных компонентов, закрепленных на роботе

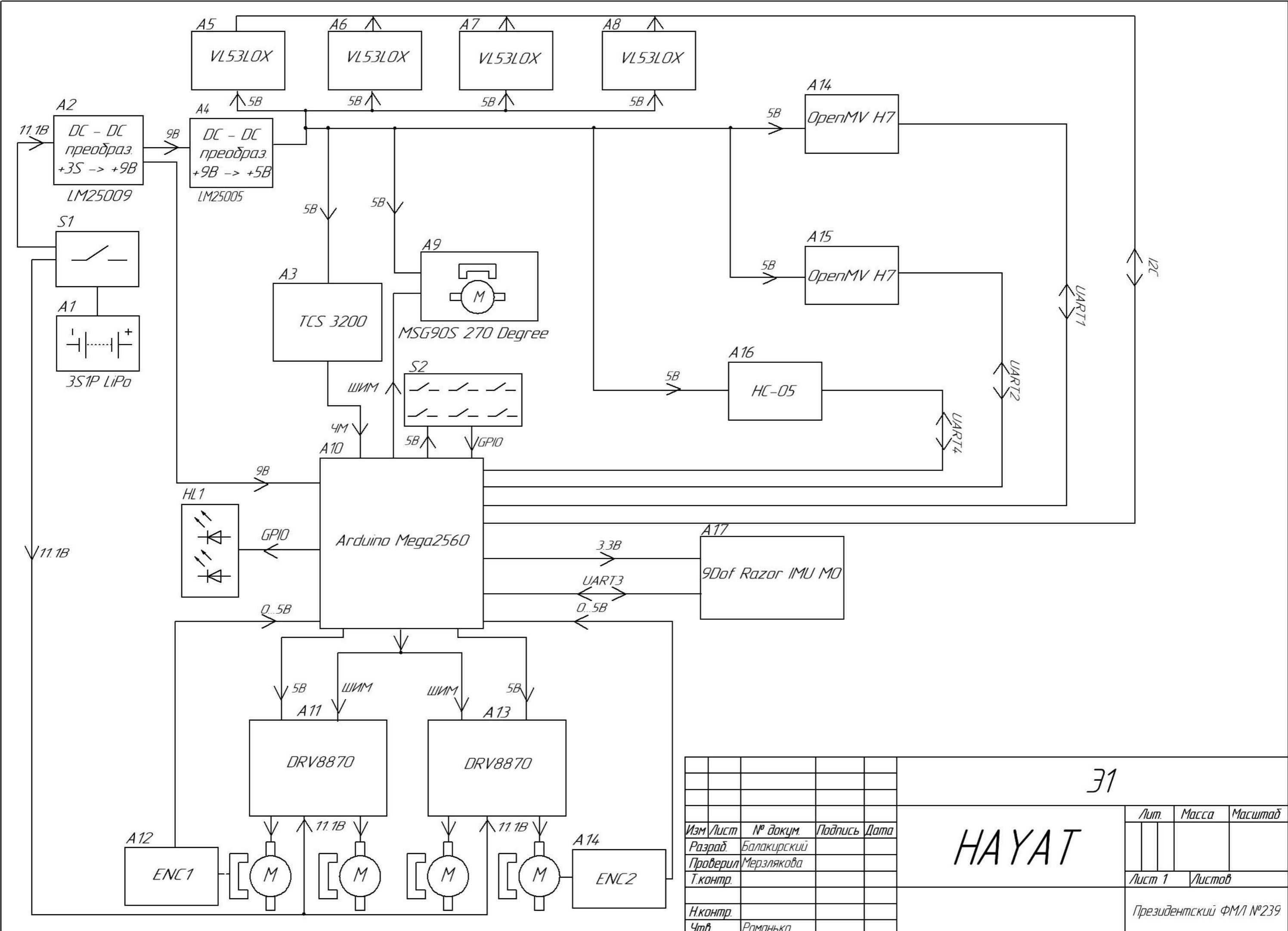
Оборудование	Электронный компонент и его характеристики	Обоснование выбора данного компонента
Управляющий контроллер	Arduino ATmega2560 Наличие 4 физически реализованных портов UART, 1 порт I ² C, 16 портов ADC, 54 цифровых пина. Частота 16 МГц.	Высокая частота, легкость в программировании, наличие множества пинов для передачи данных.
Электродвигатель	CHP-20GP-180 Мотор потребляет 6-12В, скорость 780 об/мин. Вес 80 г.	Высокая скорость, не большие размеры.

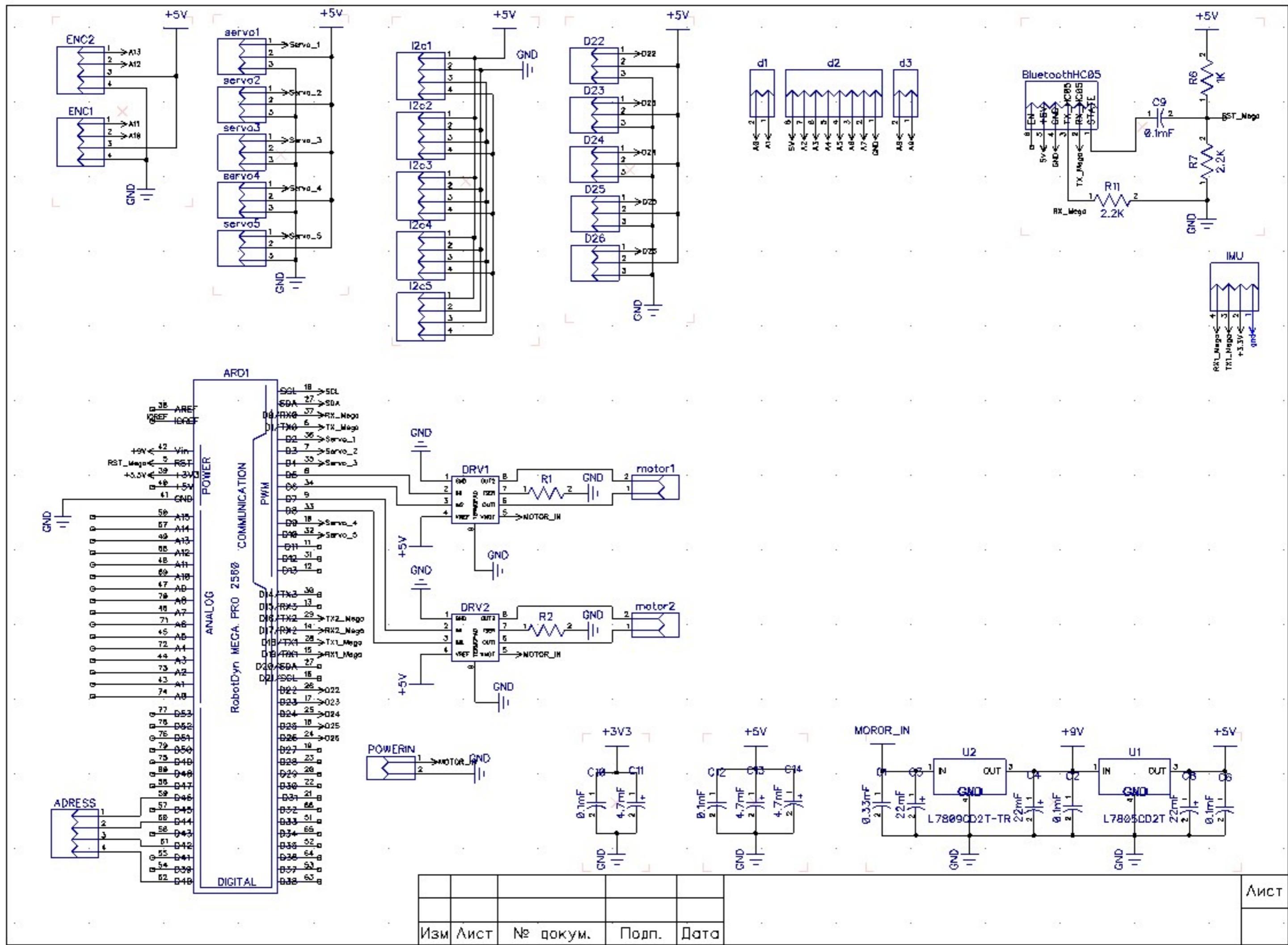
Датчик расстояния	VL53L0X Максимальная дальность – 2 м.	Небольшие размеры, наличие выводов под I ² C, наличие XSHUT- пина.
Датчик-гирокоп	9DoF Razor IMU M0 Автоматическая фильтрация, наличие 1 физически реализованного порта UART и SPI	Наличие своего МК, который программируется отдельно, высокая точность
Камера	OpenMV H7 Наличие 2 физически реализованного порта UART, 2 I ² C, 1 SPI. Наличие 1 RGB светодиода. Программируется на MicroPython.	Наличие своего МК для программирования камеры. Высокая частота для UART.
Датчик цвета	TCS3200 Питание от 2.7В до 5В.	Небольшие размеры, вывод цвета в RGB- формате (в виде 3 чисел).
Микропереключа- тели	MSW-13 Вес 2.7 г, Размеры 19.8 мм x 17.5 мм x 6.4 мм.	Небольшие размеры, большая прочность, чем у аналогов.
Сервомотор	MG90D 270 Degree Крутящий момент 3.2 кг * см.	Максимальный угол отклонения – 270 градусов.
Модуль связи (Bluetooth- модуль)	HC-05 Размеры 12.7 мм x 27 мм. Поддерживается UART.	Легкость в использовании и подключении, небольшие размеры.

Платы управления двигателями	DRV8870 Выходной ток 3.6 А.	Плата управления двигателем соответствует характеристикам электродвигателя.
Аккумулятор	SERC LiPo batteries Трехбаночный аккумулятор, емкость 1100 мА/ч, номинальное напряжение 11.1 В, масса 190 г.	Небольшие размеры, высокая токоотдача, небольшая масса.

2.3 Структурная и принципиальная схемы робота

На следующих страницах приведены схемы Э1 и Э3 робота соответственно.





На плате предусмотрены разъемы для подключения датчиков по таким шинам передачи данных, как I²C и UART. Также на плате есть разъемы для использования ШИМ-сигнала. На плате расположены стабилизаторы напряжения и драйвера моторов. Также есть возможность использования Bluetooth-модуля для беспроводной заливки программы в микроконтроллер.

2.4 Выбор сред разработки и языков программирования. Общие сведения

Для программирования основного микроконтроллера ATmega2560 (на базе Arduino mega2560) было решено использовать среду Arduino IDE (язык C++), так как в нее встроена библиотека Arduino.h, упрощающая программирование микроконтроллеров, поддерживающих загрузчик от Arduino. Для программирования модуля SparkFun IMU (гироскоп) также выбрана Arduino IDE (из-за наличия официальной документации от SparkFun по программированию этого устройства в Arduino IDE). А для программирования модуля камеры целесообразно использовать OpenMV IDE и язык MicroPython, предназначенную специально для данного модуля камеры. Соответственно, камера и гироскоп являются полностью самостоятельными модулями, программируемыми отдельно от основного микроконтроллера и отправляющими ему данных с помощью протокола UART. Камера отправляет либо численное значение буквы (“H”, “S”, “U”) по таблице ASCII, либо значение цвета (“Y” – желтый, “R” – красный, “G” – зеленый), либо число 0, которое отправляется если камера не обнаружила жертв, при том, что в предыдущем цикле жертва была обнаружена. Гироскоп отправляет по UART сначала старт-бит (число 255), потом значения рысканья (yaw), тангажа (pitch) и значение контрольной суммы (CRC8) от значений yaw и pitch.

2.5 Компьютерное зрение

В настройках программы камеры используется изображение размерами 160x120 формата RGB. Цветовые жертвы определяются с помощью встроенной функции find_blobs(), определяющей связанные области пикселей цвета из заданного диапазона в заданной области изображения (блобы). С

помощью вкладки Threshold Editor подбираются пороговые значения в цветовом пространстве LAB для каждого из необходимых цветов (желтого, красного, зеленого и черного). Для идентификации букв сначала находится самая большая связанные черная область с помощью функции `find_blobs()`. Позднее эта область делиться на 3 равных области по вертикали, в каждой из которых находится количество черных блобов. Это количество черных блоб будем называть вхождениями. На рисунке 5 (в случае буквы “Н”): 2 вхождения сверху, 1 по центру, 2 снизу. После этих действий изначальная черная блоба делится на 3 равные части по горизонтали. В случае с буквой “Н” получается, что у буквы: 1 вхождение слева, 1 посередине, 1 сверху. Также на рисунке 6 приведен пример разделения буквы “S” на вхождения. После анализа количества вхождений по вертикали и горизонтали определяется, какую букву видит камера (“Н”, “С” или “У”).

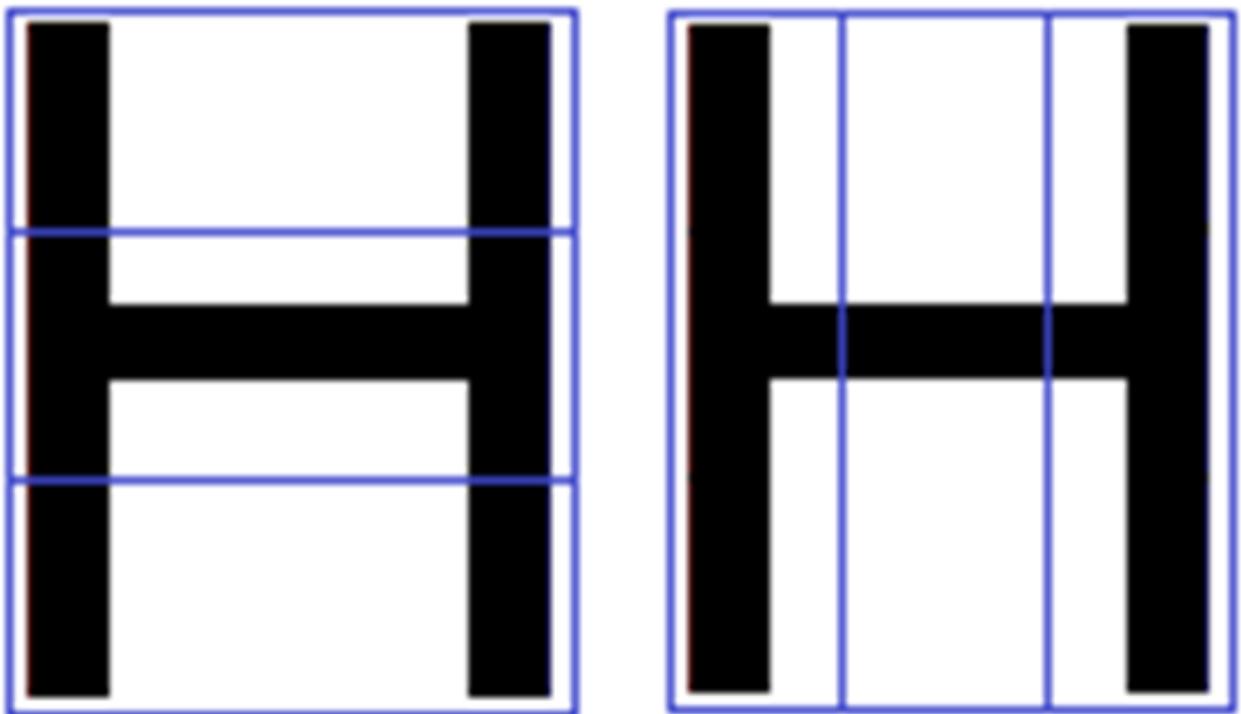


Рисунок 5. Пример разделения "Н" на области по вертикали и горизонтали соответственно.

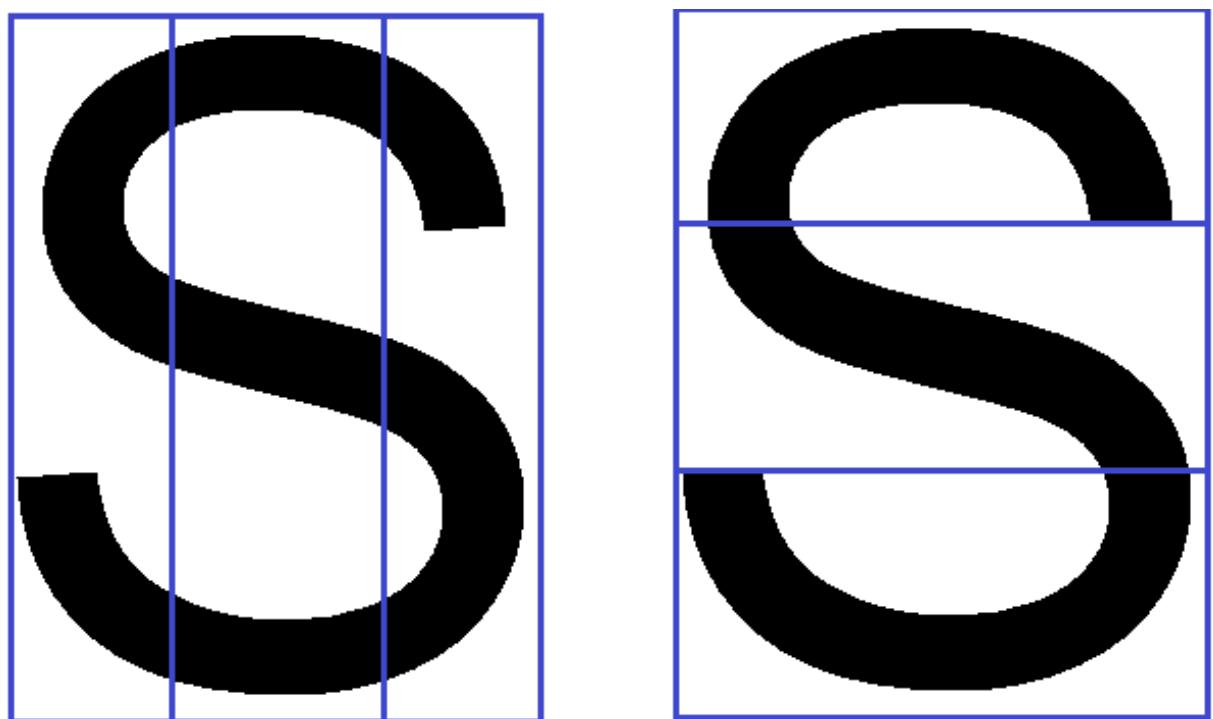


Рисунок 6. Пример разделения “S” на области по вертикали и горизонтали соответственно.

2.6 Алгоритм обхода лабиринта по левой и правой руке

Одним из самых простых алгоритмов для прохода по лабиринту является алгоритм левой/правой руки. Рассмотрим суть алгоритма левой руки. Если слева от робота нет стенки, то робот поворачивается налево и проезжает одну ячейку вперед. Если слева от робота есть стенка, а спереди свободно, то робот проезжает одну ячейку вперед. Если и спереди, и слева стенка, то робот поворачивается направо. И эти условия повторяются, пока робот не выедет из лабиринта. Для правой руки алгоритм аналогичен, однако изначально просматривается, есть ли стенка справа (а не слева). Рассмотрим блок-схемы каждого из алгоритмов, они приведены на рисунке 7.

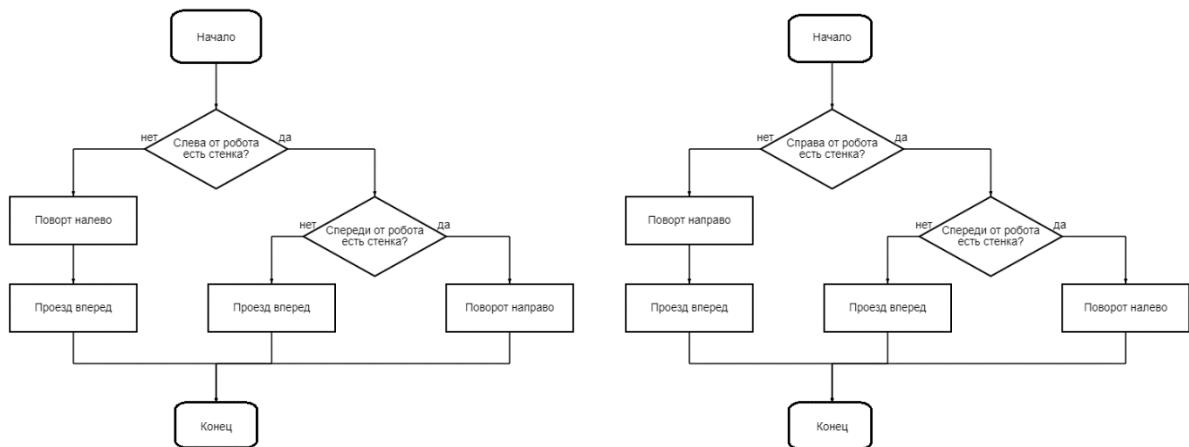


Рисунок 7. Блок-схемы алгоритмов левой и правой рук.

Как можно заметить, алгоритмы левой и правой руки не дают гарантии проезда каждой ячейки лабиринта. Например, на рисунке 8 приведен пример лабиринта и его проезда по правилам левой и правой рук.

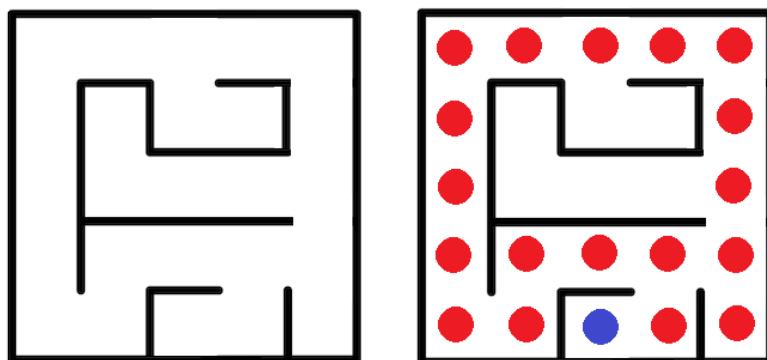


Рисунок 8. Пример лабиринта и пример его проезда по левой/правой руке.

Синий круг – стартовая ячейка (изначально робот смотрит направо). Красными кругами обозначены ячейки, через которые проедет робот по данному алгоритму. Как мы можем заметить при проезде лабиринта остаются не пройденные ячейки. Чтобы решить эту проблему обратимся к более сложным алгоритмам, гарантирующим проезд лабиринта целиком.

2.7 Понятие графа

Чтобы использовать более совершенные алгоритмы важно научиться сохранять в памяти робота конфигурацию части лабиринта, которую он уже проехал. Для этого используем понятие графа:

Граф – это абстрактная структура данных, которая состоит из узлов (вершин) и ребер (связей или дуг), соединяющих эти узлы (вершины). В графе узлы представляют собой точки данных, а рёбра показывают отношения между этими точками. В случае RCJ Rescue Maze узлами графа можно считать ячейки лабиринта, а ребрами – переходы между двумя соседними ячейками, неотделенными стенкой. На рисунке 9 приведен пример графа, созданного на основе лабиринта.

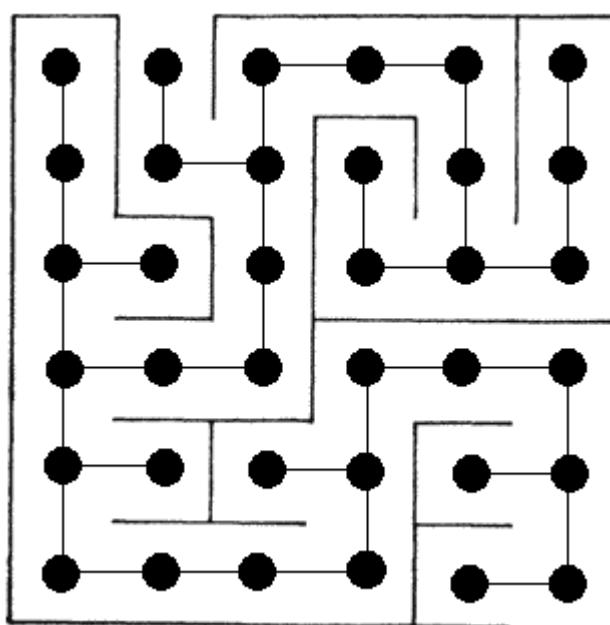


Рисунок 9. Пример графа, созданного на основе лабиринта.

На этом рисунке черные круги являются узлами графа, которые соединяются прямыми (ребрами). Для дальнейшей работы нам потребуется еще два определения:

- 1 Начальный узел – узел, с которого начинается процесс поиска или обхода графа (в случае Rescue Maze – стартовая ячейка).
- 2 Целевой узел – узел, являющийся целью поиска в графе (ячейка, в которую роботу нужно проехать).

Также заметим, что в нашем случае, граф лабиринта будет неориентированным, так как из любой ячейки, до любой связанной с ней другой ячейки, робот может двигаться в обоих направлениях.

2.7 Алгоритм поиска в ширину (BFS)

BFS (breadth-first search) является одним из основных алгоритмов поиска пути в графе. Его основная идея заключается в том, чтобы посетить все узлы графа в порядке от ближайших к начальному узлу к более отдаленным. Процесс работы алгоритма BFS начинается с выбора стартового узла графа. В нашем случае это стартовая клетка, с которой робот начинает движение. Затем все смежные с ним узлы добавляются в очередь. После этого каждый из этих узлов проверяется на наличие смежных узлов, которые еще не были посещены, и они добавляются в очередь. Таким образом, алгоритм распространяется от узла к его ближайшим соседям, затем ко второму слою соседей и так далее. Рассмотрим работу алгоритма на примере небольшого лабиринта RCJ Rescue Maze (изображен на рисунке 10):

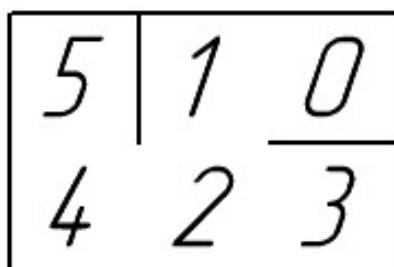


Рисунок 10. Пример лабиринта для рассмотрения BFS.

Узлы этого графа (ячейки) пронумерованы от 0 до 5, узел 0 – начальный узел. Теперь будем рассматривать каждый шаг BFS. Сначала выбирается начальный узел (его номер записывается в очередь). После чего начинается основной цикл алгоритма. Из очереди выбирается первый элемент (в нашем случае узел 0) и сразу же удаляется оттуда. Потом просматриваются все не посещенные узлы, связанные с ним, после чего эти узлы добавляются в очередь. В нашем случае узел 0 связан с только с узлом 1, он не посещенный, поэтому этот узел будет добавлен в очередь. На рисунке 11 приведена визуализация первой итерации этого цикла.

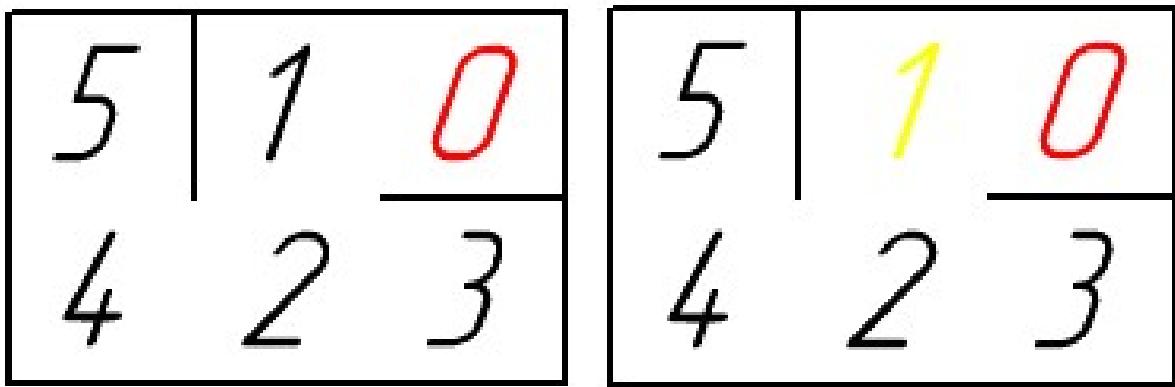


Рисунок 11. Первая итерация цикла BFS.

Здесь 0 обозначен красным, потому что мы сразу же объявляем его посещенным. Из нуля мы просматриваем узел 1 (поэтому узел 1 на рисунке обозначен желтым).

Рассмотрим следующую итерацию основного цикла. Из очереди достается ее первый элемент (узел 1), после чего он сразу же удаляется из нее. Потом просматриваются не посещенные узлы, связанные с ним. В нашем случае заметим, что с узлом 1 связан только начальный узел, который уже посещен, и узел 2 (он будет добавлен в очередь). На рисунке 12 приведена визуализация второй итерации этого цикла.

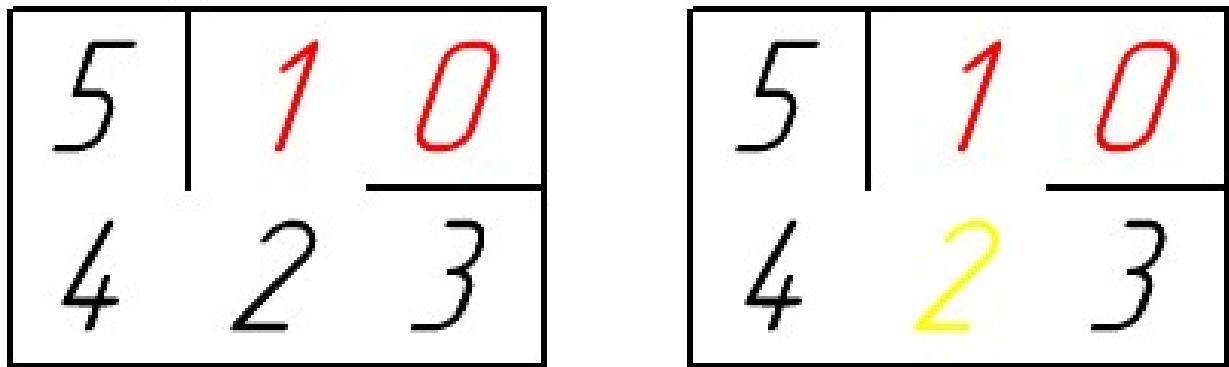


Рисунок 12. Вторая итерация цикла BFS.

Рассмотрим следующую (третью) итерацию основного цикла. Из очереди достается ее первый элемент (узел 2) и сразу же удаляется из нее. Потом просматриваются не посещенные узлы, связанные с ним. С узлом 2 связаны первый, третий и четвертый узлы. Из них узлы 3 и 4 не посещены, то есть эти узлы будут добавлены в очередь. На рисунке 13 приведена визуализация третьей итерации основного цикла.

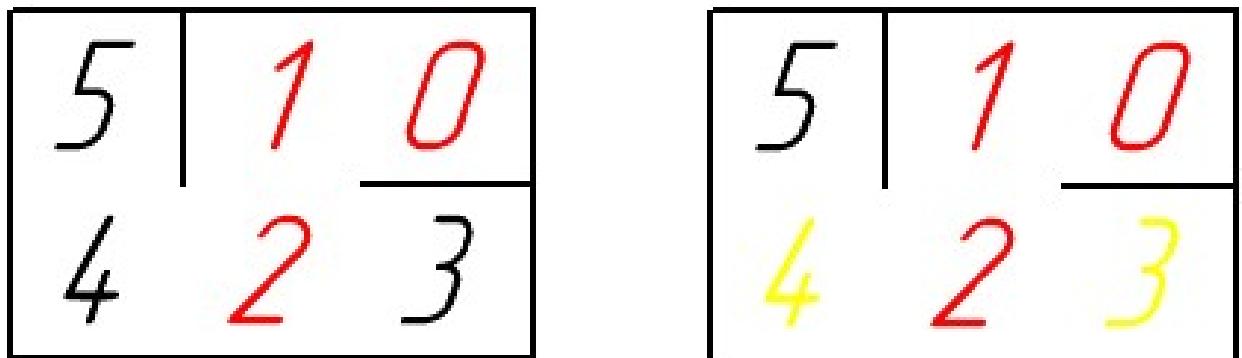


Рисунок 13. Третья итерация цикла BFS.

Приведу последующие итерации основного цикла в графической форме.

Четвертая итерация (приведена на рисунке 14):

5	1	0
4	2	3

Рисунок 14. Четвертая итерация цикла BFS.

Пятая итерация (приведена на рисунке 15):

5	1	0
4	2	3

5	1	0
4	2	3

Рисунок 15. Пятая итерация цикла BFS.

Шестая (последняя) итерация (приведена на рисунке 16):

5	1	0
4	2	3

Рисунок 16. Шестая итерация цикла BFS.

Таким образом, в ходе работы алгоритма будут пройдены все узлы графа, что и является одним из ключевых преимуществ BFS (если путь к целевому узлу существует, то алгоритм гарантированно найдет кратчайший путь до этого узла). Отметим также, что кроме прохождения всех вершин необходимо

искать длину кратчайшего пути до данной вершины и путь до нее. Как это сделать указано в статье [8].

2.8 Алгоритм Дейкстры

Для алгоритма Дейкстры нам понадобится определение взвешенного графа.

Взвешенный граф – граф, каждому ребру которого поставлено в соответствие некоторое значение (вес ребра). Его алгоритм во многом схож с BFS, однако в нем при просмотре не посещенных узлов (к некоторому данному) в первую очередь просматриваются узлы, ближайшие к начальному. Более подробно алгоритм расписан в статье [9].

2.9 Сравнение различных алгоритмов поиска пути

Преимуществами алгоритма левой/правой руки являются простота их реализации и достаточная высокая эффективность (с помощью них обычно робот способен обследовать большую часть ячеек). Их главным недостатком является отсутствие гарантии обследования всего лабиринта целиком. Также отмечу, что регламентом специально предусмотрены ячейки, которые невозможно из старта проехать по алгоритмам левой/правой руки. За нахождение жертв на этих ячейках предусмотрены дополнительные баллы.

Преимуществами алгоритма BFS является гарантированность обследования всего лабиринта и его относительная простота. Его недостатком является необходимость постоянного обновления и сохранения части лабиринта, которую робот проехал. Это является недостатком в связи с тем, что при возникновении каких-либо ошибок при проезде робота, при которых граф будет записываться неправильно. Например, если робот своим передом столкнется с боковой поверхностью стенки у него будут возникать ошибки с записью графа. При этом алгоритм левой/правой руки относительно устойчив к подобным ситуациям. Асимптотика BFS – $O(n + m)$, где n – число вершин, m – число ребер.

Алгоритм Дейкстры, также как и BFS, гарантирует обследование всего лабиринта, однако он сложнее в реализации, чем BFS. Асимптотика алгоритма Дейкстры – $O(n^2+m)$, где n – число вершин, m – число ребер (что означает, что BFS работает быстрее алгоритма Дейкстры). Также заметим, что алгоритм Дейкстры имеет смысл использовать только во взвешенном графе, однако лабиринт в целом представляет собой невзвешенный граф, из-за чего BFS работает более эффективно, чем алгоритм Дейкстры.

В итоге для выполнения задания RoboCupJunior Rescue Maze было принято решение использовать алгоритм BFS.

2.10 Хранение графа лабиринта

В программе каждая ячейка (узел графа) представляет собой элемент класса Trio:

```
class Trio {
public:
    int8_t first; //Первая координата (X)
    int8_t second; //Вторая координата (Y)
    int8_t floor; //Этаж

    Trio(int8_t first, int8_t second, int8_t floor) { //Задание всех параметров класса
        this->first = first;
        this->second = second;
        this->floor = floor;
    }

    Trio() {
        this->first = 0;
        this->second = 0;
        this->floor = 0;
    }

    bool equalXYFloor(Trio t) { //Два элемента класса Trio полностью одинаковы
        return this->first == t.first && this->second == t.second && floor == t.floor;
    }

    bool equalXY(Trio t) { //У двух элементов класса Trio одинаковы X и Y координаты
        return this->first == t.first && this->second == t.second;
    }

    bool slideDetection(Trio t1) { //Если хоть в один элемент (из двух) класса Trio содержит горку
        return t1.floor == CLIMB || t1.floor == DESCENT || this->floor == CLIMB || this->floor == DESCENT;
    }
};
```

Такой класс включает в себя 3 числа: координата робота по оси X, координата по оси Y и этаж, на котором находится робот. Также в переменную этажа может быть записаны числа CLIMB (1) и DESCENT (2), что означает подъем и спуск с горки. Место подъема/спуска по горке в графе представляется

как отдельная ячейка. Также значение этажа может быть равно FIRST_FLOOR (0) и SECOND_FLOOR (128). В SECOND_FLOOR специально записывается значение 128, чтобы возможно было модифицировать код программы и добавить больше двух этаже: в случае возникновения этажа между первым и вторым, его этаж будет записываться в Trio, как 64; а в случае возникновения этажа выше, второй – как 256, 384, 512 и так далее.

Сам график записывается с помощью двухмерного вектора класса Trio (назовем этот двумерный вектор – maze), в котором каждый нулевой элемент строки описывает координату, из которой ведется наблюдение, а остальные ячейки в строке – те, которые напрямую связаны с этой нулевой. Стартовой координатой робота считается (100; 100; FIRST_FLOOR). Таким образом составляется общий график лабиринта. Рассмотрим пример сохранения лабиринта в памяти робота (рисунок 17):

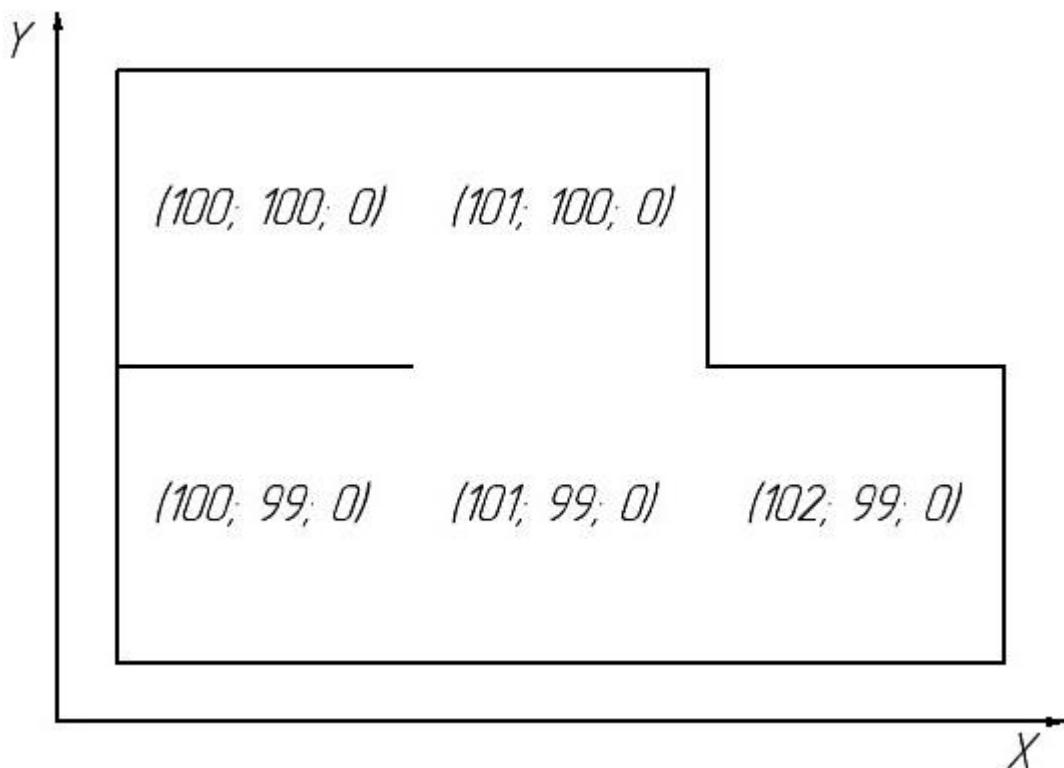


Рисунок 17. Пример лабиринта, сохраненного в maze

На этом рисунке каждой ячейке лабиринта присваивается координата (слева и снизу от рисунка подписано направление осей X и Y). Ячейка с

координатами (100; 100; 0) – стартовая ячейка. В maze, в случае езды робота от стартовой ячейки до момента, пока не будет исследован весь лабиринт, этот лабиринт будет записан следующим образом:

maze[0]: (100; 100; FIRST_FLOOR), (101; 100; FIRST_FLOOR)

maze[1]: (101; 100; FIRST_FLOOR), (101; 99; FIRST_FLOOR)

maze[2]: (101; 99; FIRST_FLOOR), (100; 99; FIRST_FLOOR), (102; 99; FIRST_FLOOR)

maze[3]: (100; 99; FIRST_FLOOR), (101; 99; FIRST_FLOOR)

maze[4]: (101; 99; FIRST_FLOOR), (100; 99; FIRST_FLOOR), (102; 99; FIRST_FLOOR)

maze[5]: (102; 99; FIRST_FLOOR), (101; 99; FIRST_FLOOR)

Пока робот едет по лабиринту, в maze постоянно добавляются новые элементы, каждый из которых однозначно характеризует клетку. Также отмечу, что элементы в maze могут повторяться (на примере вышеперечисленного лабиринта 2 и 4 элементы maze полностью идентичны) в случае проезда роботом одной и той же ячейки несколько раз, что не нарушает работу BFS. Также с помощью этой структуры записи ячеек лабиринта достаточно легко понять, какие ячейки не посещены, так как все элементы maze[i][0] (i может принимать значения от 0 включительно до числа, равному количеству пройденных роботом ячеек не включительно) являются посещенными, соответственно все ячейки, не совпадающие с ними по координате, являются не посещенными. То есть при такой структуре нет необходимости в создании еще одного массива или вектора, в котором будут записываться все посещенные ячейки.

2.10 Структура программы основного микроконтроллера

Заметим, что с помощью BFS ищется ближайшая не посещенная ячейка в лабиринте и строится путь к ней. Также отмечу, что в случае, если программа

не обнаружила ни одной не посещенной ячейки, то такой лабиринт считается полностью исследованным и в таком случае робот (по регламенту RCJ Rescue Maze) должен вернуться в стартовую ячейку, чтобы получить еще больше баллов. На рисунке 18 представлена общая блок-схема кода робота “Hayat”:

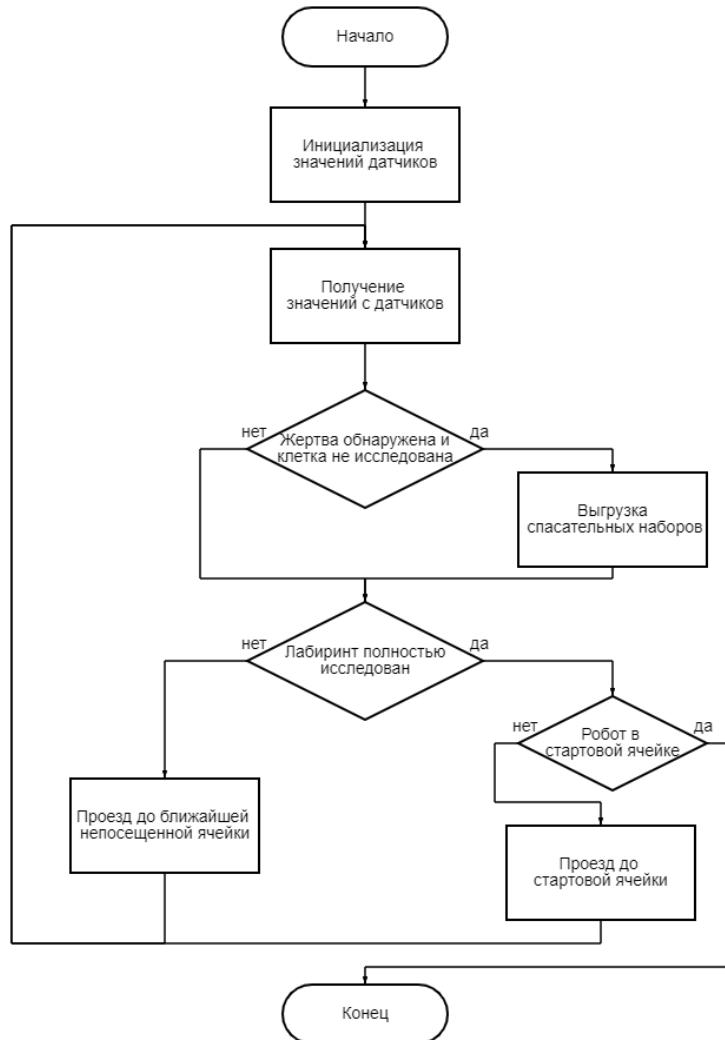


Рисунок 18. Блок-схема всей программы робота

Данная структура создается с помощью оператора `switch case` и переменной `state`, хранящей в себе номер выполняющегося состояния, то есть в виде конечного автомата. Ниже приведена часть кода в процедуре `loop()` в Arduino IDE, в которой происходит чтение с датчиков, обработка значения `state`, и вызов процедур соответствующего состояния:

```

readVLX(); //Чтение значения с датчика расстояния
readAngles(); //Чтение значений с датчика-гироскопа
readRGB(); //Чтение значения с датчика цвета
readMVRRight(); //Чтение значения с правой OpenMV
readMVLLeft(); //Чтение значения с левой OpenMV

switch (state) { //Номер состояния
    case 0: //Поворот на 90 градусов налево
        leftTurn();
        break;
    case 1: //Поворот на 90 градусов направо
        rightTurn();
        break;
    case 2: //Проезд на ячейку вперед
        forward();
        break;
    case 3: //Уточнение в какую ячейку ехать и в каком направлении двигаться
        question();
        break;
    case 4: //Простаивание в ячейке на 700 миллисекунд
        waitInCell();
        break;
    case 5: //Выравнивание по стенке вперед
        alignment(FORWARD);
        break;
    case 6: //Действия при обнаружении жертвы: мигание светодиодом и выброс наборов
        victims();
        break;
    case 7: //Действия при синей ячейке
        blueColorFunction();
        break;
    case 8: //Действия при черной ячейке
        blackColorFunction();
        break;
    case 9: //Проезд вперед после горки
        afterSlideForward();
        break;
    case 10: //Объезд справа (если нажался левый микропереключатель)
        detour(RIGHT);
        break;
    case 11: //Объезд слева (если нажался правый микропереключатель)
        detour(LEFT);
        break;
    case 12: //Выравнивание по стенке назад
        alignment(BACKWARD);
        break;
}

```

2.11 Распознавание цветов на полу лабиринта

Так как на полу лабиринта могут находиться черные ячейки (которые нельзя посещать) и синие ячейки (на которых нужно остановиться на не менее, чем 5 секунд), есть необходимость в использовании датчика цвета, смотрящего вниз. Этим датчиком в роботе является TCS3200, который способен выдавать

видимый им цвет в виде 3 чисел по осям red (r), green (g) и blue (b) в системе RGB. Черная клетка определяется с помощью суммы значений r, g, b. Это значение суммы должно быть меньше определенного значения. Синяя клетка же определяется с помощью нахождения угла между нынешним вектором цвета (с координатами (r; g; b) в трехмерном цветовом пространстве) и искомым значением вектора на синей клетке (он определяется перед попыткой). Этот угол равен арккосинусу отношения скалярного произведения этих двух векторов и произведения длин этих векторов.

2.12 Проезд вперед между двумя соседними ячейками

Для ровного проезда робота вперед используются датчики-расстояния (VL53L0X) по бокам (для выравнивания по стенкам по бокам), датчик-гироскоп (9DoF Razor IMU M0) для корректирования угла, под которым едет робот, и энкодеры для синхронизации моторов. Ниже приведена процедура, рассчитывающая значения, которые необходимо подать на моторы для реализации езды из одной ячейки в другую:

```
void goForward_Enc_IMU_VLX(int speed, int setPointYaw) {
    float uIMU, uVLX, uEnc, u; //Объявление переменных

    //Нахождение управляющего воздействия для гироскопа
    uIMU = -8 * differenceYaw(setPointYaw, yaw) * signYaw(yaw, setPointYaw);
    //Здесь функция differenceYaw(int, int) возвращает модуль разницы двух углов с учетом периода
    //То есть, например, для при вызове функции differenceYaw(10, 350), она выдаст значение 20
    //Функция signYaw(int, int) показывает больше ли первый угол, чем второй с учетом периода

    if (abs(rightVLX - leftVLX) < 40 && leftVLX < 175 && rightVLX < 175) { //Если оба датчика расстояния видят стенку
        uVLX = -12 * (float)(rightVLX - leftVLX); //Считаем управляющее воздействие для датчиков, как разность их показаний
    } else if (rightVLX > leftVLX && leftVLX < 175) { //Если только левый датчик расстояния видит стенку
        //Считаем управляющее воздействие, как разность желаемого расстояния и показания левого датчика
        uVLX = -4.1f * (float)(115 - leftVLX);
    } else if (leftVLX > rightVLX && rightVLX < 175) { //Если только правый датчик расстояния видит стенку
        //Считаем управляющее воздействие, как разность желаемого расстояния и показания правого датчика
        uVLX = 4.1f * (float)(115 - rightVLX);
    } else { //Если оба датчика расстояния не видят стенку
        uVLX = 0; //Считаем управляющее воздействие датчиков расстояния за 0
    }

    //Если поднимаемся или спускаемся с горки уменьшаем управляющее воздействие датчиков расстояния
    if (floors == CLIMB || floors == DESCENT) uVLX /= 2;

    //Расчет управляющего воздействия для энкодеров
    uEnc = (float)((rightEnc - rightInCellStart) - (leftEnc - leftInCellStart)) * 3.5f;
    //Здесь rightEnc и leftEnc - показания энкодеров на правом и левом моторе соответственно
    //rightInCellStart и leftInCellStart - показания энкодеров в начале ячейки из которой совершает проезд вперед робот

    //Расчет суммарного управляющего воздействия
    if (uVLX != 0) u = uVLX * 0.5 + uIMU * 0.35 + uEnc * 0.15;
    else u = uIMU * 0.7 + uEnc * 0.3;

    //Расчет подаваемого значения на моторы
    leftMotorValue = speed - u;
    rightMotorValue = speed + u;
}
```

Заметим, что в этой процедуре используются пропорциональные регуляторы для расчета управляющего воздействия для каждого из датчиков.

Теперь рассмотрим расчет расстояния, которое нужно проехать роботу (двигаясь вперед), чтобы переместиться в соседнюю ячейку. Для решения этой проблемы для начала были просмотрены значения выдаст передний датчик расстояния при дистанции от передней стенки в 0, 1, 2, 3 ячейки (при большей дистанции от передней стенки датчик начинает показывать неточные значения, поэтому они не учитываются), значения при этих дистанциях были записаны в массив. Аналогичные действия были проведены с задним датчиком расстояния. В итоге получилось 2 массива:

```
int distForward[] = {60, 455, 830, 1050}; //Массив для переднего датчика расстояния  
int distBackward[] = {70, 330, 635, 1010}; //Массив для заднего датчика расстояния
```

Будем руководствоваться принципом: если ближайшая стенка спереди находится менее, чем расстоянии $distForward[3] + 70$ (то есть на расстоянии не большем 3 ячеек с запасом), то нужно найти такой элемент i , что число $|centralVLX - distForward[i]|$ – минимально, где $centralVLX$ – показание переднего датчика расстояния. После нахождения такого i нужно двигаться вперед до тех пор, пока $centralVLX$ не станет меньше или равен $distForward[i - 1]$. Практически аналогичная логика работает и для заднего датчика расстояния, однако ближайшая стенка сзади должна находиться на расстоянии, не большем, чем $distBackward[2] + 70$, и нужно двигаться вперед пока $backwardVLX$ ($backwardVLX$ – показание заднего датчика расстояния) не станет больше и равен $distBackward[i + 1]$. Заметим, что из-за специфики датчиков расстояния (а именно из-за нелинейности показаний датчиков в лабиринте) невозможно создать линейную зависимость показаний переднего или заднего датчика от количества ячеек до ближайшей передней или задней клетки соответственно (именно из-за этого создается массив со значениями). Еще отмечу, что если расстояния спереди и сзади не проходят вышеописанные условия, то робот должен есть вперед по количеству градусов энкодеров.

2.13 Отладка и модификации

Изначально был сделан прототип будущего робота (изображен на рисунке 19).

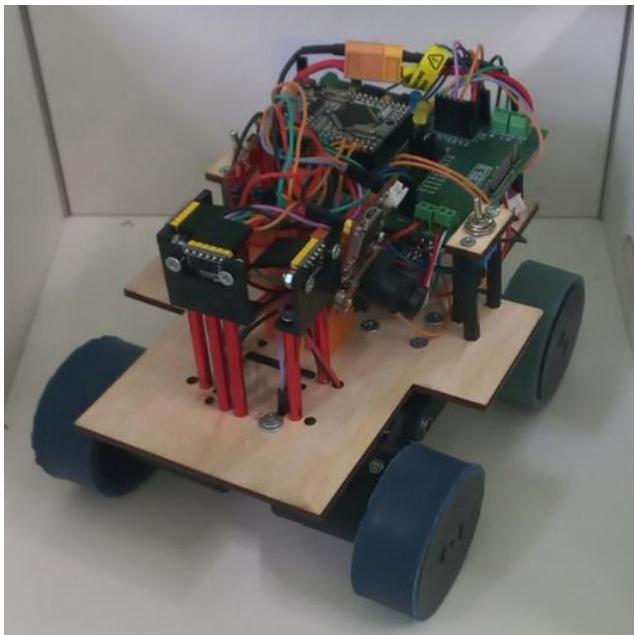


Рисунок 19. Первая версия робота.

Если рассматривать конструкционную часть робота, то отмечу, что на нем закреплены 4 мотора (двою из которых имеют датчики-энкодеры), на которых закреплено 4 колеса. В роботе используется независимая подвеска. Основной каркас робота вырезан из фанеры. На роботе закреплены 2 камеры OpenMV (они смотрят по бокам), 1 RGB датчик TCS3200, 1 датчик-гироскоп 9DoF Razor IMU

M0, 4 датчика расстояния VL53L0X (1 датчик смотрит вперед, 1 назад, 1 вправо и 1 влево). Также на роботе закреплены 2 светодиода. Робот проезжает между ячейк исключительно по энкодерам. Для реализации компьютерного зрения была использована встроенная функция `find_template()` в OpenMV.

Серьезными недостатками этого прототипа являются:

- 1 Отсутствие системы выдачи спасательных наборов (на тот момент она не была окончательно продумана и сделана).
- 2 Несовершенство алгоритма обхода лабиринта по левой руке.
- 3 Вставание робота на дыбы при столкновении с передней и задней стенками (из-за этого невозможно было выравнивание по стенкам).

Учитывая эти проблемы, было принято решение изменить робота. Нынешний вариант робота изображен на рисунке 20.

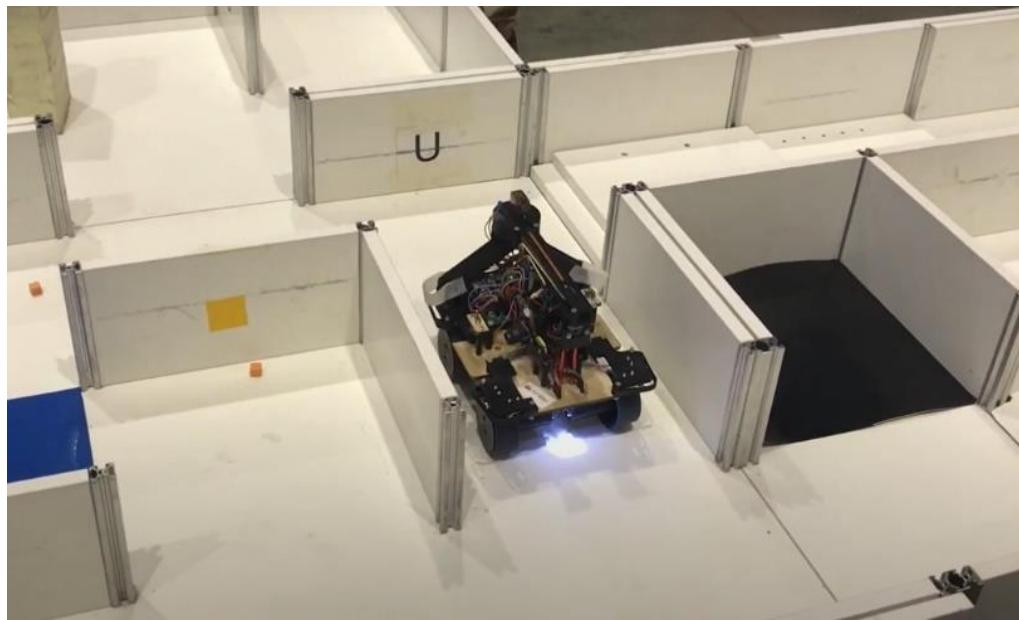


Рисунок 20. Текущая версия робота “Hayat”.

Для начала отметить конструкционные изменения в двух версиях робота. На роботе появились 6 микропереключателей (4 спереди и 2 сзади). Стало заметно, что они сильно помогают во время выравниваний по стенке спереди и сзади, а также во время обнаружения препятствий. В изначальной модели не было системы для выкидывания спасательных наборов, после чего она была добавлена. Она была сделана на основе системы выдачи команды “МК”, ранее участвовавшей в Rescue Maze. Система выдачи робота “Hayat” представлена на рисунке 21:

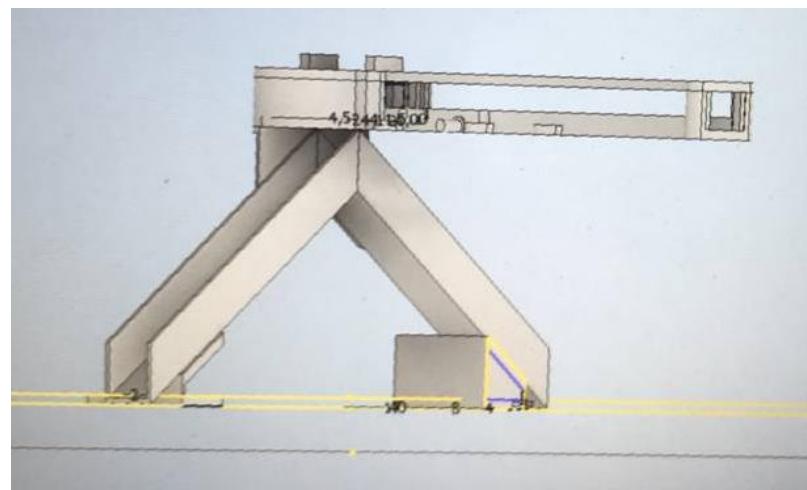


Рисунок 21. Система выдачи спасательных наборов команды “Hayat”

Достаточно много времени понадобилось, чтобы разработать кубики так, чтобы после выкидывания они не отскакивали от жертв на расстояние больше 15 сантиметров (в таком случае по регламенту RCJ Rescue Maze считается, что спасательные наборы не выданы). В итоге было принято решение создать кубики с дробями внутри для уменьшения их инертности, что помогло решить эту проблему (это решение было реализовано в октябре 2023).

С точки зрения программирования, робот стал сохранять конфигурацию лабиринта в граф в свою память, был реализован BFS (вместо алгоритма обхода лабиринта по левой руке, как это было раньше). Робот стал ехать из одной ячейки в другую, используя датчик-гирокоп и датчики-расстояния, а не только энкодеры. Для реализации компьютерного зрения был придуман и написан собственный алгоритм.

Основная отладка робота происходила на полигоне RCJ Rescue Maze в “Президентском” ФМЛ №239 (изображен на рисунке 22).



Рисунок 22. Полигон Rescue Maze в ПФМЛ №239

3 ЗАКЛЮЧИТЕЛЬНЫЙ ЭТАП

3.1 Креативность и новизна проекта

Проект является оригинальной разработкой. Конструкция робота разработана полностью самостоятельно, создан алгоритм именно для данного робота с этим набором датчиков, моторов и другой периферии.

3.2 Результат

В результате проделанной работы была достигнута поставленная цель. Был создан робот, с помощью которого можно отлаживать и тестировать алгоритмы для навигации по лабиринту. Алгоритмы были протестированы на полигоне RoboCupJunior Rescue Maze. В процессе создания робота я научился работать с компьютерным зрением и аппаратной платформой Arduino. Улучшил свои знания в 3D-моделировании в Autodesk Inventor. Также отмечу, что робот “Hayat” был представлен на соревнованиях “Открытые состязания Санкт-Петербурга по робототехнике 2023” (2 место), “РобоКап Россия 2023” (1 место) и “Фестиваль Робофинист 2023” (2 место). На рисунке 23 приведены фотографии с данных соревнований.



Рисунок 23. Робот "Hayat" на фестивале "Робофинист" и чемпионате "РобоКап Россия 2023" соответственно

В приложении 1 приложен диплом с состязания “РобоКап Россия 2023”, в приложении 2 – диплом с “Фестиваля Робофинист 2023”.

4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Официальный регламент RoboCupJunior Rescue Maze [Электронный ресурс] – <https://junior.robocup.org/wp-content/uploads/2024/01/RCJRescueMaze2024-final.pdf>

2 Официальный сайт OpenMV [сайт] –
<https://openmv.io/>

3 Документация для программирования OpenMV [сайт] –
<https://docs.openmv.io/>

4 Datasheet ATmega2560 [Электронный ресурс] –
https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

5 Datasheet датчика расстояния [Электронный ресурс] –
<https://www.st.com/resource/en/datasheet/vl53l0x.pdf>

6 Руководство пользователя 9Dof Razor IMU M0 [сайт] –
<https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide/all>

7 Datasheet микропереключателей [Электронный ресурс] –
<https://static.chipdip.ru/lib/364/DOC014364385.pdf>

8 Статья, в которой рассматривается нахождение кратчайших расстояний и путей по BFS [сайт] –
<https://brestprog.by/topics/bfs/>

9 Статья, в которой более подробно рассматривается алгоритм Дейкстры [сайт] –
<https://prog-cpp.ru/deikstra/>

ПРИЛОЖЕНИЕ А



ТУСУР
TUSUR UNIVERSITY

SMART
TECHNOLOGIES TOMSK

центра инноваций и
инициативы
Томской области

2023 ГОД ПЕДАГОГА
И НАСТАВНИКА

ДИПЛОМ

Балакирский Аркадий Владимирович

участник команды

Hayat

ГБОУ "Президентский ФМЛ №239", Санкт-

Петербург

награждается за I место

в лиге Роботы-спасатели - лабиринт, Старшая

группа (RCJ Rescue Maze Secondary)

Открытый Российский чемпионат по

робототехнике РобоКап Россия 2023

Томск

11-14 мая 2023 года

Б. М. Рулевский
Ректор ТУСУР
Председатель
Оргкомитета

Е. С. Шандаров
Российский
представитель
RoboCup Junior



ПРИЛОЖЕНИЕ Б



Награждается участник команды

ФИО Балакирский Аркадий Владимирович

Команда Hayat

ГБОУ Президентский ФМЛ №239

(название организации)

За II место в категории

Спасатели в лабиринте RoboCupJunior Rescue Maze

Международного фестиваля робототехники РобоФинист 2023
г. Санкт-Петербург

Заместитель председателя
комитета по образованию
Борщевский А.А.

Директор ГБОУ
«Президентский ФМЛ №239»
Пратусевич М.Я.

Главный судья соревнований
Филиппов С.А.

Президент
Благотворительного Фонда «Финист»
Аминджанов Т.А.

Руководитель
организационного комитета
Бородачева О.В.



29 октября – 1 ноября 2023

000-104-055

РОБОФИНИСТ

StarLine

