ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта

Высшая школа робототехники и автоматизации

Курсовой проект

по дисциплине «Объектно-ориентированное программирование»

«*Разработка системы позиционирования мобильного робота на базе связки Raspberry Pi и Arduino с использованием QR-маркеров*»

Пояснительная записка

Выполнил студент

гр. 3331506/20401 _____ Шарифьянов А.Р.
(подпись)

Работу принял _____ Ананьевский М.С.
(подпись)

Санкт-Петербург

2025

Оглавление

ВВЕДЕНИЕ

В современных условиях автоматизации и роботизации особую актуальность приобретают автономные мобильные роботы, способные ориентироваться в пространстве и выполнять задачи без постоянного контроля со стороны человека. Разработка системы навигации на основе распознавания QR-кодов представляет собой перспективное решение, сочетающее относительную простоту реализации с высокой надежностью позиционирования. Такие системы активно внедряются в различных отраслях промышленности: в складской логистике, автомобильной промышленности, авиастроении, пищевой промышленности, металлургии, тяжелом машиностроении и многих других.

Техническое задание

Робот должен автоматически перемещаться из точки **А** в точку **Б**, следуя по черной линии и ориентируясь по QR-кодам на перекрёстках. Система состоит из двух модулей: Raspberry Pi (обработка изображений, распознавание QR-кодов, принятие решений о маршруте) и Arduino (управление моторами, считывание датчиков линии, коррекция движения).

Движение по линии с использованием датчиков линии.

Остановка на перекрёстках, распознавание QR-кода камерой (Raspberry Pi + OpenCV/ZBar).

Определение текущего положения на карте на основе данных из QR-кода (например, координаты или номер перекрёстка).

Выбор направления движения согласно заложенному алгоритму (например, по кратчайшему пути до цели).

Плавный разгон/торможение, коррекция траектории при съезде с линии.

Возможность задания маршрута через внешний интерфейс (например, передача точек А и Б по SSH).

1. Теоретические сведения

Работа будет выполняться на основе связки двух платформ – Arduino UNO и Raspberry Pi 3B+, а также на базе Omegabot. Для начала разберём функционал и особенности каждой.

1.1 Arduino Uno

Arduino Uno — это компактный микроконтроллер с простой и удобной платформой для разработки, представленная на рисунке 1. Плата умеет считывать данные с аналоговых и цифровых датчиков, управлять моторами, светодиодами и другими исполнительными устройствами, а также взаимодействовать с компьютером через USB. Главное преимущество Arduino Uno — 14 цифровых и 6 аналоговых контактов ввода/вывода, к которым можно подключать широчайший спектр периферии: от кнопок и потенциометров до сервоприводов и беспроводных модулей. Благодаря открытой архитектуре и простой среде программирования, Uno идеально подходит для быстрого прототипирования электронных устройств.

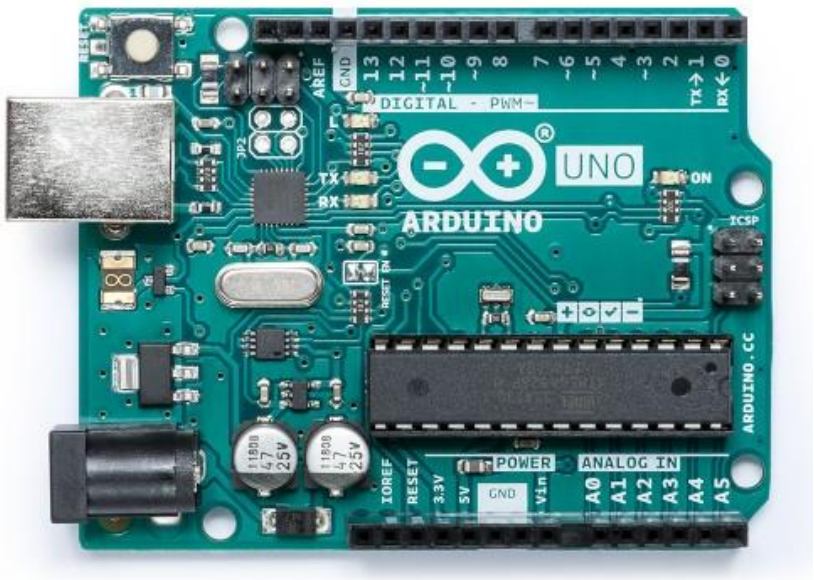

Рисунок 1 – Схема Arduino Uno

1.2 Raspberry Pi 3B+

Raspberry Pi — полноценный компьютер размером с кредитную карту (рисунок 2). Контроллер умеет выводить изображение на дисплей, работать с USB-устройствами и Bluetooth, снимать фото и видео на камеру, воспроизводить звуки через динамики и выходить в интернет. Главное преимущество Raspberry Pi — 40 контактов ввода/вывода общего назначения (GPIO). К ним возможно подключать периферию для взаимодействия с внешним миром: исполнительные устройства, любые сенсоры и всё, что работает от электричества.

Рисунок 2 – Схема Raspberry Pi 3 B+

1.3 Omegabot

Omegabot — это образовательная платформа для изучения робототехники и программирования на базе Arduino и Raspberry Pi, представлен на рисунке 3. Конструктор позволяет собирать и программировать роботов с разным уровнем сложности: от простых машинок на колёсах до автономных устройств с компьютерным зрением и ИИ. Главное преимущество Omegabot — модульная система датчиков и исполнительных

механизмов, включая ультразвуковые сенсоры, сервоприводы, Bluetooth-модули и камеры. Платформа поддерживает несколько языков программирования (C++, Python, блок-схемы) и подходит как для новичков, так и для продвинутых инженеров.

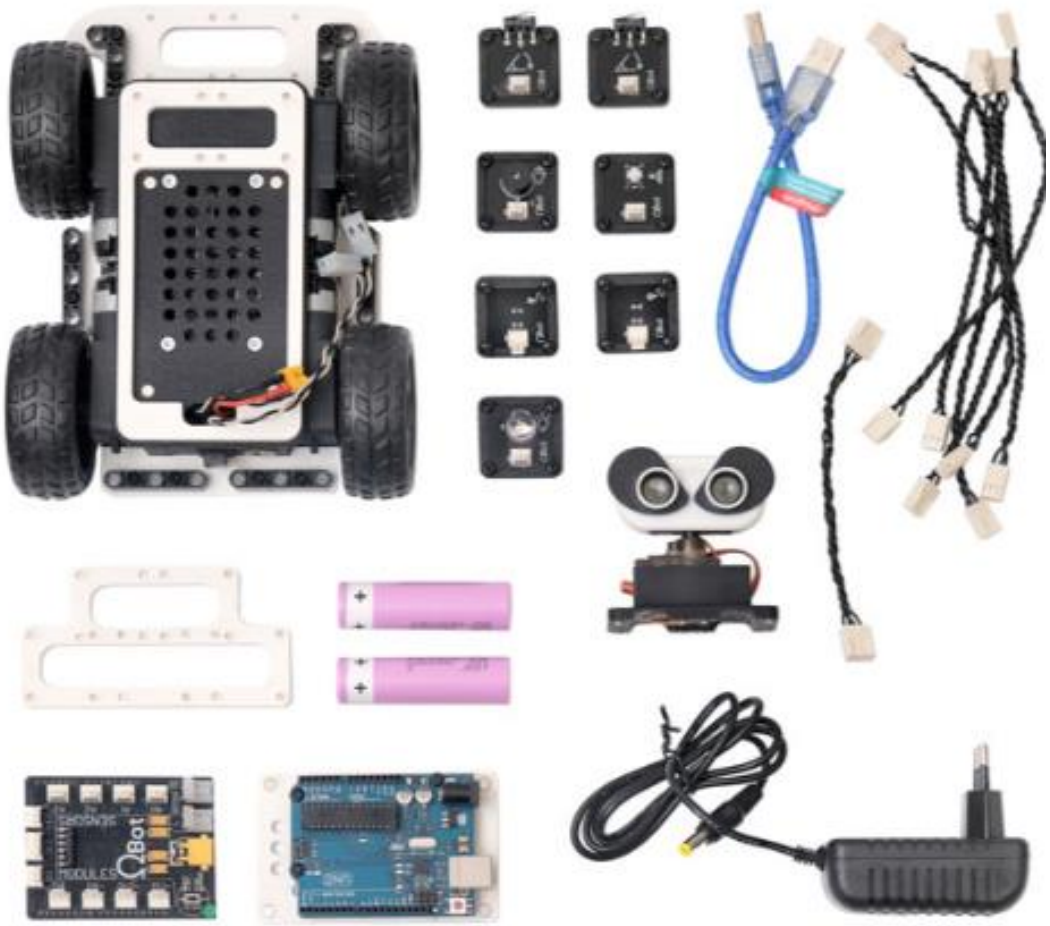

Рисунок 3 – Комплект Omegabot

2. Подготовительные работы

2.1 Выбор камеры

Перед началом работы надо определиться будет ли в качестве видеокамеры использоваться комплектная CSI-камера на штативе Omegabot, представленная на рисунке 4 или отдельная веб-камера USB, представленная на рисунке 5.

Рисунок 4 – Комплектная CSI-камера Raspberry на штативе Omegabot

Рисунок 5 – Веб-камера USB

2.2 Raspberry CSI-камера

У комплектной камеры есть свои преимущества – прямое подключение к процессору через шлейф на плате (рисунок 6), что даёт меньшую задержку, а также управляемый сервомоторами штатив. Однако для адекватной работы в программе трекинга QR-кодов потребуется очень много времени потратить на настройку libcamera (библиотека для комплектной камеры), чтобы использовать OpenCV, который понадобится для работы сканера.

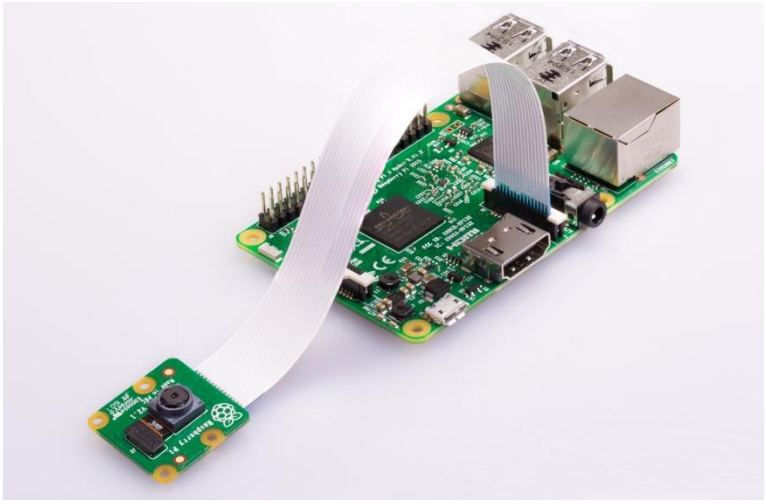

Рисунок 6 – Подключение камеры через шлейф

2.3 Веб-камера USB

В то же время обычная веб-камера поддерживается стандартными драйверами и работает почти в любой системе без дополнительных настроек, что даёт большое преимущество в случае дальнейшего использования данного проекта в будущем на более профессионально-ориентированных платформах (ROS, TensorFlow).

Таким образом, CSI-камеры Raspberry Pi демонстрируют более высокую производительность и минимальные задержки благодаря прямому подключению через аппаратный интерфейс. Однако их интеграция требует дополнительной настройки специализированного ПО (такого как libcamera),

редактирования системных конфигураций и работы с узкоспециализированными библиотеками.

Стандартные USB-камеры хоть и уступают в абсолютной производительности, обеспечивают мгновенную совместимость с большинством программных решений без необходимости сложной предварительной настройки. Для нашего проекта они подойдут больше.

2.4 Блок-схема проекта

Для более полного понимания взаимодействия компонентов в нашем проекте составим блок-схему, представленную на рисунке 7.

Далее описаны примерные

Робот должен автоматически перемещаться по заданной траектории, следуя по черной линии с помощью ИК-датчиков (TCRT5000) и корректируя направление движения. На перекрёстках система останавливается, распознаёт QR-код с помощью камеры, подключённой к Raspberry Pi, и определяет дальнейший маршрут.

Raspberry Pi отвечает за обработку изображений: захватывает видео с камеры, использует библиотеку ZBar для распознавания QR-кодов и передаёт данные на Arduino по UART (через USB). Arduino получает команды, управляет моторами через ШИМ (PWM) с помощью драйвера L298N, обрабатывает сигналы с ИК-датчиков линии и реализует основную логику движения (разгон, торможение, повороты).

Для индикации состояния робота используются светодиоды или звуковые сигналы. Питание системы раздельное: отдельный источник для моторов (чтобы избежать помех) и стабилизированное питание для Raspberry Pi и Arduino.

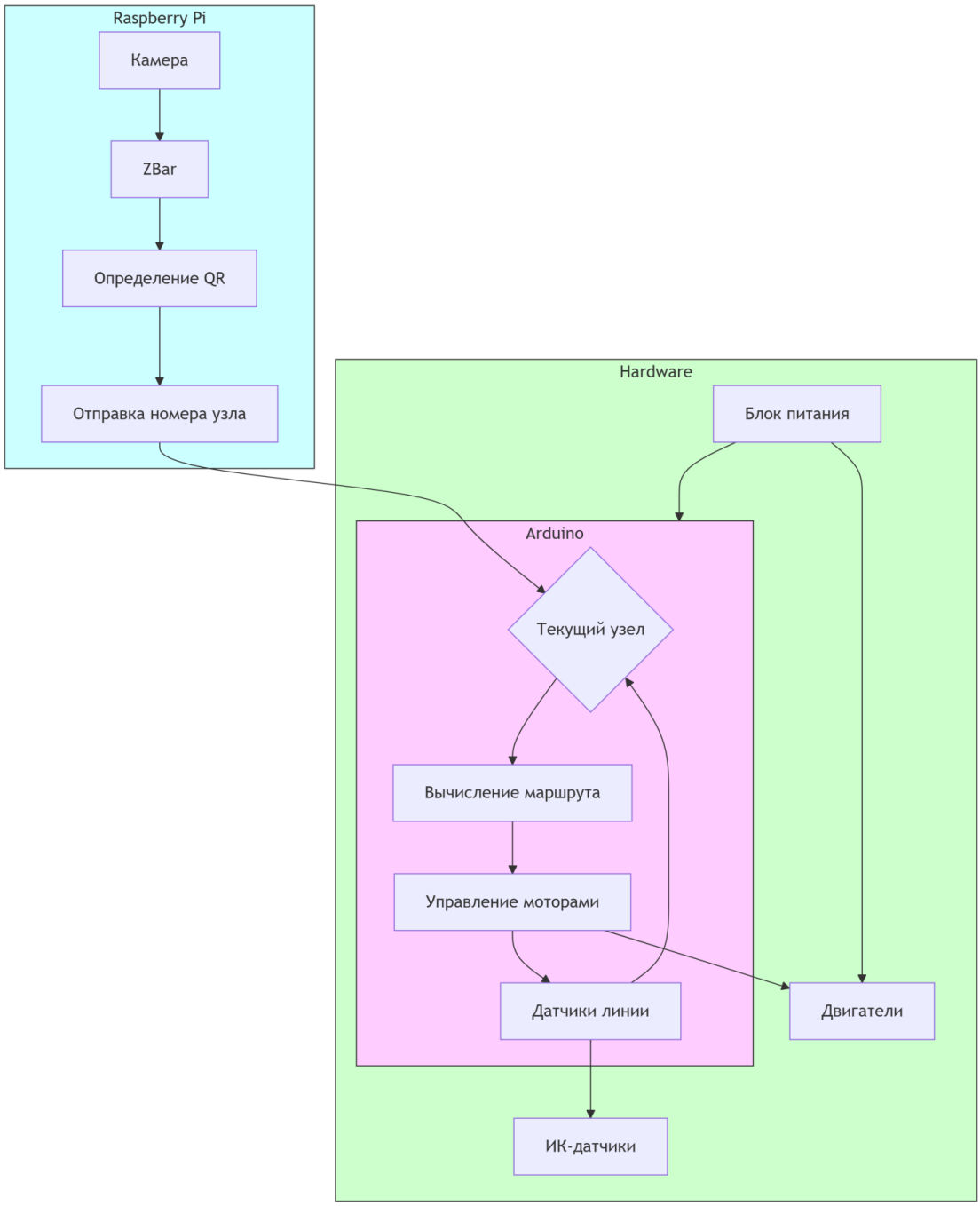

Рисунок 7 – Блок-схема проекта

3. Ход работы

Выполнение проекта будет состоять из нескольких этапов: для начала необходимо реализовать QR-сканнер на Raspberry, затем настроить передачу данных по UART в Arduino. После этого реализовать построить карту и реализовать алгоритм навигации по ней, исходя из данных QR-сканнера.

3.1 Установка ОС на Raspberry Pi

Для начала работы с Raspberry Pi необходимо записать образ операционной системы на microSD-карту. Сначала нужно подготовить карту памяти - вставить её в кард-ридер компьютера и убедиться, что на ней нет важных данных, так как в процессе записи все существующие файлы будут удалены. Затем следует загрузить подходящий образ ОС с официального сайта Raspberry Pi, где доступны различные варианты, в проекте была использована Raspberry Pi OS with Desktop. После скачивания архив нужно распакувать, чтобы получить файл образа в формате .img или .img.xz.

Далее необходимо открыть программу Balena Etcher, которая предоставляет простой и надежный способ записи образов на карты памяти. В интерфейсе программы выбираем три основных шага: сначала указываем распакованный файл образа операционной системы, затем выбираем целевую microSD-карту из списка доступных накопителей (здесь важно не ошибиться с выбором устройства, чтобы случайно не стереть данные с другого диска). После подтверждения выбора запускается процесс записи - программа автоматически форматирует карту и записывает образ, отображая прогресс операции.

По завершению можно вставлять SD карту в одноплатный компьютер и настраивать систему.

3.2 Создание QR-сканера на Raspberry

Для начала проведѐм тест камеры, которую подключим по USB. Используем команду ―ffplay - f v4l2 - i / dev / video0”. Raspberry успешно принимает сигнал с камеры.

Теперь приступим к созданию самого QR-сканера. Потребуется библиотеки OpenCV, а также zbar для работы с QR-кодами, устанавливаем их и пишем тестовый код.

3.3 Соединение с Arduino

Для передачи данных из Raspberry в Arduino по протоколу UART необходимо соединить их по USB. Соединение по USB показано на рисунке 8.

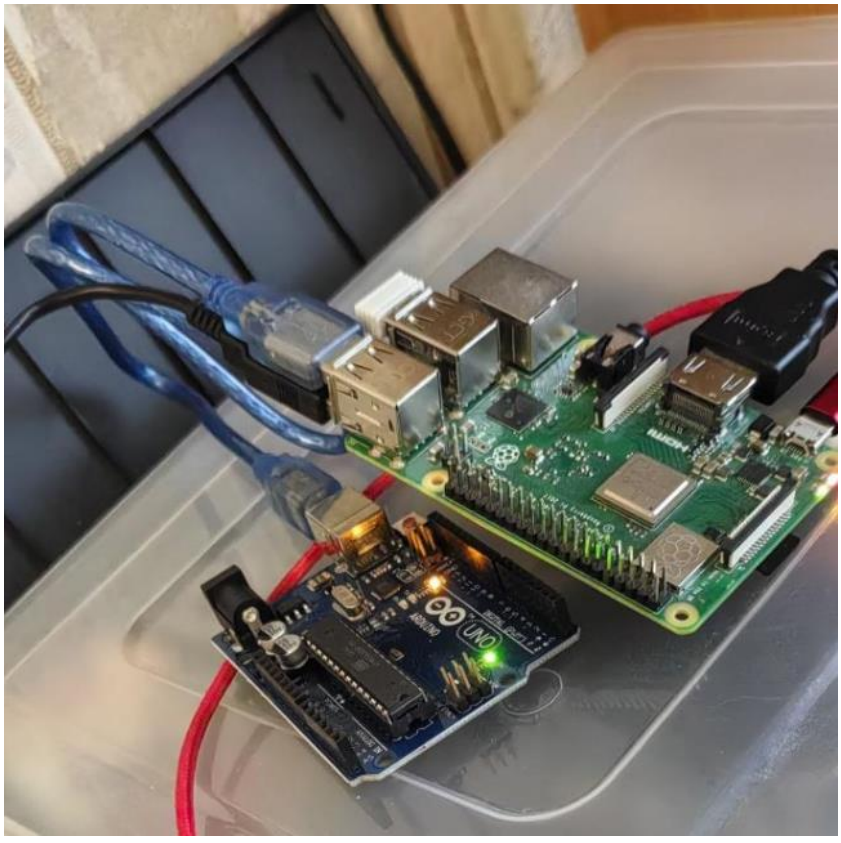

Рисунок 8 – Подключение Arduino и RaspberryPi

Далее необходимо написать код для оправки данных на Arduino (номера qr кодов). И прописать его для автозапуска в rc.local.

Полный код для считывания qr кодов и отправки из на ардуино показан в приложении 1.

3.4 Создание алгоритма навигации

Возвращаемся к поставленной задаче: робот должен передвигаться на линии, считывая QR-коды, по заранее известной карте, карта представлена на рисунке 9.

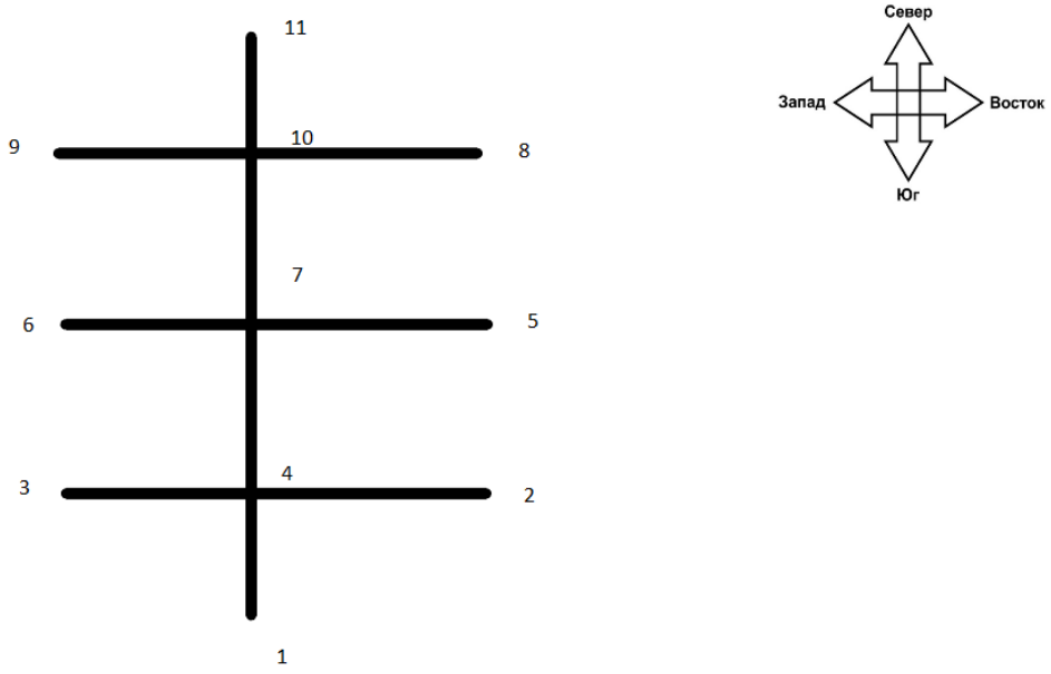

Рисунок 9 – Карта перекрестков

Алгоритм следования по линии и предварительной калибровки датчиков несложно реализовать, используем для этого код из предыдущего проекта с Omegabot. Перейдѐм сразу к самому алгоритму навигации по QR-кодам.

После анализа карты несложно заметить, что не все QR-коды имеют одинаковую сложность обработки: у нас есть перекрѐстки, на которых робот

должен принимать решение, исходя из маршрута, а также у нас есть тупики, на которых просто всегда будет необходим разворот.

Составим примерную блок-схему для необходимого алгоритма, показанную на рисунке 10

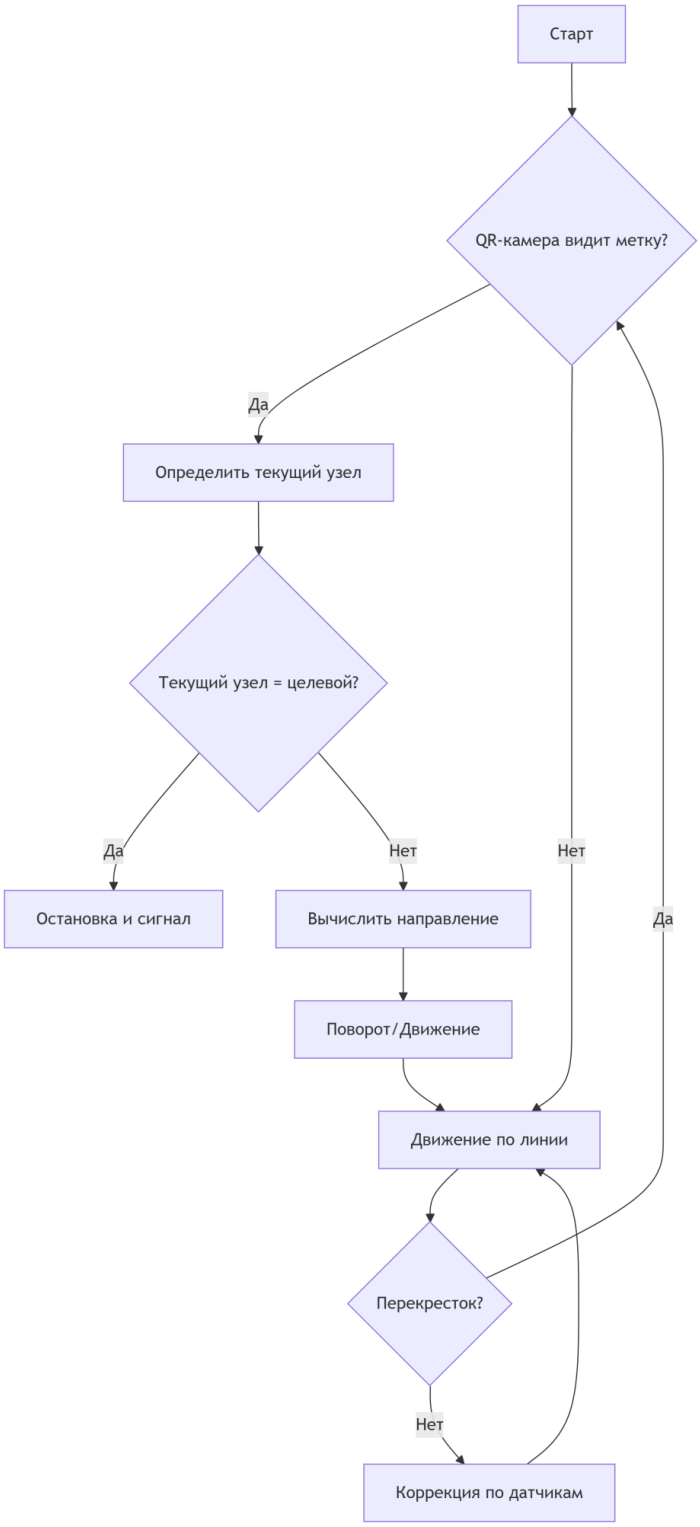

Рисунок 10 – Алгоритм навигации

4. Доработка конструкции и испытания

Прежде всего, проверяем в режиме тестирования алгоритм ориентации по QR, для этого добавим в код, чтобы значения QR, которые считывает робот, можно было также писать в консоль. Логика работает, переходим к практическим тестам. Для первого теста проверяем алгоритм следования по линии отдельно, затем задаем простой маршрут без поворотов.

Полный код для Arduino показан в приложении 2.

Проблемы: робот держит линию, однако алгоритм езды по линии мешает реализации алгоритма поворотов, а также часто робот не успевает считывать QR-коды.

Решение: введем различные состояния в коде (STATE_CALIBRATE, STATE_FOLLOW_LINE и т.д.), так будет проще обнаруживать ошибки, а также отключать следование по линии, когда робот находится в состоянии определения маршрута. Также поднимем камеру выше в конструкции, чтобы ей хватало фокусного расстояния, конструкция показана на рисунке 11.

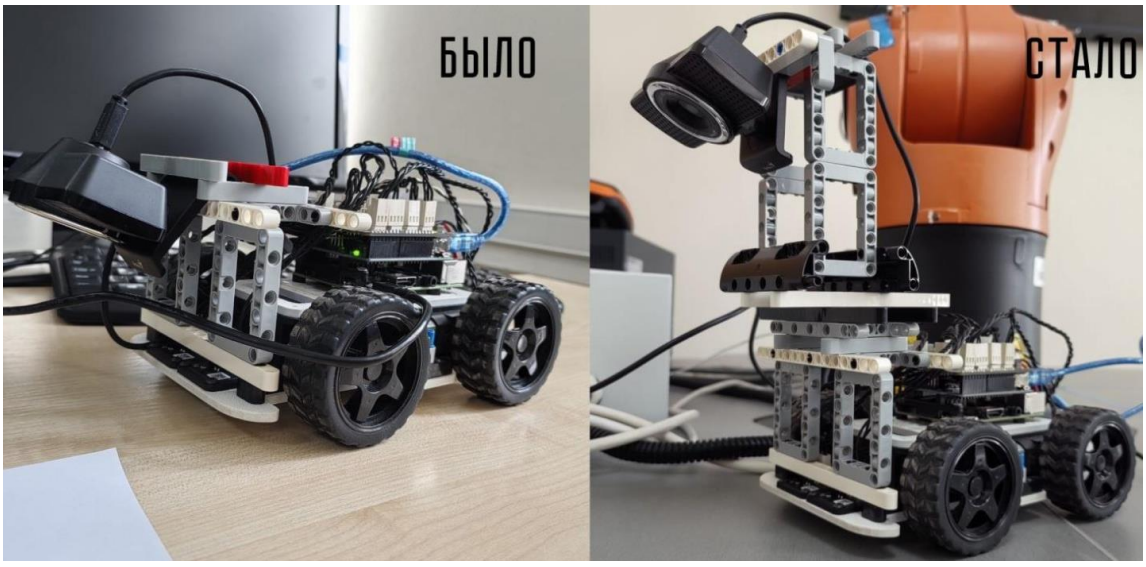

Рисунок 11 – Доработка робота

4.1 Решение проблем

После внесения изменений в конструкцию и систему навигации были проведены комплексные испытания на маршруте с несколькими

перекрестками. Система демонстрирует улучшенное распознавание QR-кодов, однако сохраняется проблема с пропуском некоторых перекрестков. Хотя следование по линии теперь не мешает работе на перекрестках, выявлена некорректная работа поворотов из-за чрезмерной скорости моторов.

Основная сложность заключается в необходимости балансировки скорости: слишком низкая скорость приводит к недостатку мощности для старта (требуется "подталкивание"), а высокая - к проблемам при поворотах. Для решения этой проблемы был разработан адаптивный алгоритм управления скоростью:

плавное замедление при приближении к перекресткам

ускорение на длинных прямых участках

полная остановка для считывания QR-кодов с последующим сбросом таймеров ускорения/замедления

После коррекции алгоритма и констурукции можно сделать следующие выводы:

На прямых участках маршрута система работает стабильно, точно останавливаясь на перекрестках и корректно считывая QR-коды. Однако при прохождении сложных маршрутов с поворотами выявлены следующие проблемы:

преждевременное начало поворотов из-за углового расположения камеры

неточное завершение поворотов

Для устранения этих недостатков планируется доработать алгоритм поворотов, добавив:

небольшой прямой проезд перед началом поворота

дополнительный прямой участок после завершения поворота

более точную калибровку положения камеры

Эти изменения позволят улучшить точность прохождения поворотов и общую надежность системы навигации.

5. Заключения

В рамках проекта была успешно создана система автономной навигации мобильного робота на основе связки Raspberry Pi и Arduino. Основная задача - организация движения по линии с распознаванием QR-кодов для позиционирования и построения маршрута - была решена с применением алгоритма поиска в ширину (BFS), который показал свою эффективность для учебного полигона с простой топологией.

Ключевые технологические решения включают:

- Реализацию компьютерного зрения на базе OpenCV и ZBar для детекции QR-кодов
- Оптимизированную UART-связь между Raspberry Pi и Arduino
- Многоуровневый алгоритм управления с состояниями калибровки, следования по линии и обработки перекрестков

Особого внимания заслуживает разработанный алгоритм плавного изменения скорости, который компенсировал ограничения моторной базы, позволяя роботу:

- Стартовать без механического воздействия
- Точно позиционироваться на перекрестках
- Сохранять стабильность движения

Для промышленного применения системы рекомендуется:

- Замена моторной базы на сервоприводы или шаговые двигатели, исключающие необходимость программной компенсации
- Переход на алгоритм Дейкстры для работы со взвешенными графами маршрутов
- Интеграция дополнительных сенсоров (лидары, GPS) для навигации вне зон с QR-разметкой
- Внедрение адаптивных алгоритмов для работы в динамически изменяющейся среде

Перспективы развития системы включают:

- Масштабирование для складских и производственных помещений

- Оптимизацию маршрутов с учетом множества факторов (длина пути, время, энергопотребление)
- Внедрение методов машинного обучения для адаптации к изменяющимся условиям

Разработанное решение представляет собой надежную основу для создания промышленных систем автономной навигации, требующую дальнейшей аппаратной и алгоритмической модернизации для работы в реальных производственных условиях.

Список литературы

1. Блум, Дж. Изучаем Arduino: инструменты и методы технического волшебства /Дж. Блум. — Санкт-Петербург : Питер, 2020. — 320 с. — ISBN 978-5-4461-1234-5.

2. Брадски, А. OpenCV 4: компьютерное зрение на Python / А. Брадски, Г. Каэлер. — Москва : ДМК Пресс, 2021. — 420 с. — ISBN 978-5-97060-789-1.

3. Монк, С. Raspberry Pi: сборка проектов и программирование / С. Монк. —Москва : Эксмо, 2019. — 256 с. — ISBN 978-5-699-87654-3.

4. Arduino Documentation [Электронный ресурс]. — URL: https://www.arduino.cc/reference/en/ (датаобращения: 10.05.2025).

5. Raspberry Pi Official Documentation [Электронный ресурс]. — URL: https://www.raspberrypi.com/documentation/ (датаобращения: 05.05.2025).

6. OpenCV Tutorials [Электронный ресурс]. — URL: https://docs.opencv.org/master/d9/df8/tutorial_root.html (датаобращения: 05.05.2025).

6. ZBar Library Documentation [Электронный ресурс]. — URL: http://zbar.sourceforge.net/ (датаобращения: 05.05.2025).

Приложение 1 – Код для парсера и отправки на Arduino

```
#include <opencv2/opencv.hpp>
#include <zbar.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <chrono>
#include <sys/ioctl.h>
#include <thread>  // Добавлен этот заголовочный файл

using namespace cv;
using namespace std;
using namespace zbar;

const string ARDUINO_PORT = "/dev/ttyACM0";
const int QR_RESEND_DELAY_MS = 10000;
const int RECONNECT_DELAY_MS = 2000;

int arduino_fd = -1;
string last_sent_qr;
chrono::steady_clock::time_point last_send_time;
bool arduino_ready = false;

bool is_arduino_connected(int fd) {
    if (fd < 0) return false;
    int status;
    return (ioctl(fd, TIOCMGET, &status) != -1);
}

int setup_serial() {
    int fd = open(ARDUINO_PORT.c_str(), O_WRONLY | O_NOCTTY);
    if (fd < 0) return -1;

    struct termios tty;
    tcgetattr(fd, &tty);
    cfsetospeed(&tty, B9600);
    tty.c_cflag &= ~PARENB;
    tty.c_cflag &= ~CSTOPB;
    tty.c_cflag |= CS8;
    tty.c_cflag &= ~CRTSCTS;
    tty.c_cflag |= CREAD | CLOCAL;
    tty.c_cc[VMIN] = 0;
    tty.c_cc[VTIME] = 5;
    tcsetattr(fd, TCSANOW, &tty);

    // Исправленная строка с использованием std::
    std::this_thread::sleep_for(std::chrono::milliseconds(2000));
    return fd;
}

void send_command(const string& cmd) {
    if (!arduino_ready) return;

    int bytes_written = write(arduino_fd, (cmd + "\n").c_str(), cmd.size() +
1);
    if (bytes_written < 0) {
        cerr << "Write error, trying to reconnect..." << endl;
        arduino_ready = false;
        close(arduino_fd);
        arduino_fd = -1;
```

```
    } else {
        fsync(arduino_fd);
    }
}

bool should_send_qr(const string& qr_data) {
    auto now = chrono::steady_clock::now();
    auto elapsed = chrono::duration_cast<chrono::milliseconds>(now -
last_send_time).count();

    if (qr_data != last_sent_qr || elapsed >= QR_RESEND_DELAY_MS) {
        last_sent_qr = qr_data;
        last_send_time = now;
        return true;
    }
    return false;
}

int main() {
    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "ERROR: Camera not found!" << endl;
        return 1;
    }
    cap.set(CAP_PROP_FRAME_WIDTH, 640);
    cap.set(CAP_PROP_FRAME_HEIGHT, 480);

    ImageScanner scanner;
    scanner.set_config(ZBAR_NONE, ZBAR_CFG_ENABLE, 1);

    cout << "Initializing..." << endl;

    arduino_fd = setup_serial();
    if (arduino_fd >= 0) {
        arduino_ready = true;
        cout << "Arduino connected successfully!" << endl;
    } else {
        cerr << "Failed to connect to Arduino" << endl;
    }

    Mat frame, gray;
    while (true) {
        static auto last_check = chrono::steady_clock::now();
        auto now = chrono::steady_clock::now();
        if (chrono::duration_cast<chrono::milliseconds>(now -
last_check).count() >= RECONNECT_DELAY_MS) {
            if (!arduino_ready) {
                arduino_fd = setup_serial();
                if (arduino_fd >= 0) {
                    arduino_ready = true;
                    cout << "Arduino reconnected!" << endl;
                }
            }
            last_check = now;
        }

        cap >> frame;
        if (frame.empty()) continue;

        cvtColor(frame, gray, COLOR_BGR2GRAY);
        Image image(gray.cols, gray.rows, "Y800", gray.data, gray.cols *
gray.rows);
```

```
        if (scanner.scan(image) > 0) {
            for (auto symbol = image.symbol_begin(); symbol !=
image.symbol_end(); ++symbol) {
                string qr_data = symbol->get_data();
                cout << "QR detected: " << qr_data << endl;

                if (should_send_qr(qr_data) && arduino_ready) {
                    send_command(qr_data);
                    cout << "Sent to Arduino: " << qr_data << endl;
                }
            }
        }

        if (waitKey(1) == 27) break;
    }

    if (arduino_fd >= 0) close(arduino_fd);
    return 0;
}
```

Приложение 2 – Код для навигации на Arduino

```
#include <SoftwareSerial.h>

// Конфигурация пинов
#define DIR_R_PIN 4
#define DIR_L_PIN 7
#define PWR_R_PIN 5
#define PWR_L_PIN 6
#define SENS_R_PIN A0
#define SENS_L_PIN A1
#define BTN_PIN 8
#define BUZZER 11

// Настройки
#define MAX_NODES 12
#define MAX_EDGES 4
#define BASE_SPEED 85
#define TURN_SPEED 120
#define TURN_DURATION 700
#define BUZZ_DURATION 200
#define STOP_DURATION 1000
#define DECELERATION_INTERVAL 300    // Интервал замедления в мс
#define DECELERATION_STEP 7          // Шаг уменьшения скорости
#define MIN_SPEED 45                 // Минимальная скорость
#define ACCELERATION_INTERVAL 200    // Интервал для обратного разгона
#define ACCELERATION_STEP 7          // Добавим шаг разгона
#define DEAD_END_TURN_DURATION 2500  // Время для разворота в тупике

// Обновленные настройки для плавных поворотов
#define TURN_RATIO 0.6f          // Соотношение скоростей моторов при
повороте
#define TURN_BASE_DURATION 700  // Базовое время поворота (на 90 градусов)
#define TURN_SPEED_MAIN 120      // Основная скорость поворота
#define TURN_SPEED_SECONDARY 75  // Вспомогательная скорость

#define TURN_SLOWDOWN_START 80     // Начальная скорость замедленного мотора
#define TURN_SLOWDOWN_MIN 45       // Минимальная скорость
#define TURN_SLOWDOWN_STEP 10      // Шаг замедления
#define TURN_SLOWDOWN_INTERVAL 50  // Интервал изменения

#define PRE_TURN_MOVE_DURATION 300  // Движение прямо перед поворотом
#define POST_TURN_MOVE_DURATION 500 // Движение прямо после поворота

// Направления
#define NORTH 0
#define EAST 1
#define SOUTH 2
#define WEST 3

// Карта маршрутов {N, E, S, W}
const int ROUTE_MAP[12][4] = {
  /* 0 */ { 0, 0, 0, 0 },
  /* 1 */ { 4, 0, 0, 0 },
  /* 2 */ { 0, 0, 0, 4 },
  /* 3 */ { 0, 4, 0, 0 },
  /* 4 */ { 7, 2, 1, 3 },
  /* 5 */ { 0, 0, 0, 7 },
  /* 6 */ { 0, 7, 0, 0 },
  /* 7 */ { 10, 5, 4, 6 },
  /* 8 */ { 0, 0, 0, 10 },
```

```
  /* 9 */ { 0, 10, 0, 0 },
  /* 10 */ { 11, 8, 7, 9 },
  /* 11 */ { 0, 0, 10, 0 }
};

enum State {
  STATE_CALIBRATE,
  STATE_WAIT_DESTINATION,
  STATE_FOLLOW_LINE,
  STATE_AT_INTERSECTION,
  STATE_STOP_AT_INTERSECTION,
  STATE_TURNING,
  STATE_DEAD_END,
  STATE_REVERSE,
  STATE_ARRIVED
};

// Глобальные переменные
State currentState = STATE_CALIBRATE;
int destination = 0;
int currentPosition = 0;
int previousPosition = 0;
int orientation = NORTH;
unsigned long turnStartTime = 0;
int turnDirection = 0;
bool lineLost = false;
unsigned long lineLostTime = 0;
int lineThreshold = 0;
int color_black = 0;
int color_white = 0;
int currentSpeed = BASE_SPEED;
unsigned long lastDecelerationTime = 0;
unsigned long lastAccelerationTime = 0;
bool acceleratingBack = false;
unsigned long turnDuration = TURN_DURATION;

// Добавляем флаг, чтобы можно было включать и отключать ускорение и
замедление
bool enableSpeedControl = false;  // Разрешение на управление скоростью

// Структура для хранения пути
struct Path {
  int next_node;
  int direction;
};

// Поиск кратчайшего пути (упрощенный BFS)
Path findShortestPath(int start, int target) {
  // Массив посещенных узлов
  bool visited[MAX_NODES] = { false };
  // Очередь для BFS: {current_node, previous_direction}
  int queue[MAX_NODES * 2];
  int front = 0, rear = 0;

  // Храним предыдущий узел и направление
  int parent[MAX_NODES] = { -1 };
  int directions[MAX_NODES] = { -1 };

  // Инициализация
  queue[rear++] = start;
  queue[rear++] = -1;  // Направление для стартового узла отсутствует
  visited[start] = true;
```

```
  // Обход в ширину
  while (front < rear) {
    int current = queue[front++];
    int prevDir = queue[front++];

    // Проверка цели
    if (current == target) {
      // Восстановление пути от цели к старту
      int node = target;
      while (parent[node] != start && node != -1) {
        node = parent[node];
      }
      if (node == -1) return { -1, -1 };
      return { target, directions[node] };
    }

    // Проверка направлений в порядке: N, E, S, W
    for (int dir = 0; dir < 4; dir++) {
      int next = ROUTE_MAP[current][dir];
      if (next != 0 && !visited[next]) {
        visited[next] = true;
        parent[next] = current;
        directions[next] = dir;
        queue[rear++] = next;
        queue[rear++] = dir;
      }
    }
  }

  return { -1, -1 };  // Путь не найден
}


void setup() {
  pinMode(DIR_R_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWR_R_PIN, OUTPUT);
  pinMode(PWR_L_PIN, OUTPUT);
  pinMode(BTN_PIN, INPUT_PULLUP);
  pinMode(BUZZER, OUTPUT);

  Serial.begin(9600);
  calibrateSensors();
  beep(1);
  Serial.println("SYSTEM READY");
}

void loop() {
  checkSerial();
  int rightSensor = analogRead(SENS_R_PIN);
  int leftSensor = analogRead(SENS_L_PIN);
  handleSensors(rightSensor, leftSensor);
  runStateMachine();
}

void checkSerial() {
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();

    Serial.print("Received: ");
    Serial.println(input);
```

```
    if (input.startsWith("GO")) {
      int newDest = input.substring(2).toInt();
      if (newDest >= 1 && newDest <= 11) {
        destination = newDest;
        Serial.print("New destination: ");
        Serial.println(destination);
        //beep(2);
        if (currentState == STATE_ARRIVED) {
          currentState = STATE_FOLLOW_LINE;
        }
      }
    } else if (input.toInt() >= 1 && input.toInt() <= 11) {
      handleQRCode(input.toInt());
    }
  }
}

void handleQRCode(int newPos) {
  previousPosition = currentPosition;
  currentPosition = newPos;

  // Сброс параметров скорости при обнаружении QR-кода
  lastDecelerationTime = millis();
  acceleratingBack = false;
  currentSpeed = BASE_SPEED;

  // Определение ориентации
  for (int dir = 0; dir < 4; dir++) {
    if (ROUTE_MAP[previousPosition][dir] == currentPosition) {
      orientation = dir % 4;  //защита от превышения
      break;
    }
  }

  // Переход в состояние перекрестка ТОЛЬКО через QR-код
  if ((currentPosition == 4 || currentPosition == 7 || currentPosition == 10)
&& currentState != STATE_WAIT_DESTINATION) {  //если нет цели, перекрёсток
или нет - неважно
    currentSpeed = BASE_SPEED;
// Сброс скорости
    currentState = STATE_STOP_AT_INTERSECTION;
    Serial.println("Intersection detected via QR");
  }

  Serial.print("Position: ");
  Serial.print(currentPosition);
  Serial.print(" | Orientation: ");
  printDirection(orientation);

  if (currentPosition == destination) {
    currentState = STATE_ARRIVED;
    Serial.println("Route completed");
    beep(3);
  }
}

void printDirection(int dir) {
  const char* directions[] = { "NORTH", "EAST", "SOUTH", "WEST" };
  Serial.println(directions[(dir % 4 + 4) % 4]);  // Гарантированный диапазон
0-3
}

void handleSensors(int rightSensor, int leftSensor) {
```

```
  // Игнорировать датчики во время поворотов и на перекрестках
  if (currentState == STATE_TURNING || currentState == STATE_AT_INTERSECTION)
return;

  bool rightOnLine = rightSensor > lineThreshold;
  bool leftOnLine = leftSensor > lineThreshold;

  switch (currentState) {
    case STATE_FOLLOW_LINE:
      // Механизм плавного замедления
      if (enableSpeedControl && millis() - lastDecelerationTime >
DECELERATION_INTERVAL) {
        if (currentSpeed > MIN_SPEED) {
          currentSpeed = max(currentSpeed - DECELERATION_STEP, MIN_SPEED);
        }
        lastDecelerationTime = millis();
      }

      // Механизм обратного разгона
      if (currentSpeed <= MIN_SPEED && !acceleratingBack) {
        acceleratingBack = true;
        lastAccelerationTime = millis();  // Сброс таймера сразу при
активации
      }

      if (enableSpeedControl && acceleratingBack) {
        if (millis() - lastAccelerationTime > ACCELERATION_INTERVAL) {
          currentSpeed = min(currentSpeed + ACCELERATION_STEP, 65);
          lastAccelerationTime = millis();
          if (currentSpeed >= BASE_SPEED) acceleratingBack = false;
        }
      }

      if (rightOnLine && leftOnLine) {
        // Оба датчика на линии - активируем управление скоростью
        enableSpeedControl = true;
        setMotors(currentSpeed, currentSpeed);
        lineLost = false;

      } else if (rightOnLine) {
        setMotors(BASE_SPEED, 0);
        enableSpeedControl = false;
        currentSpeed = 75;
        lineLost = false;

      } else if (leftOnLine) {
        setMotors(0, BASE_SPEED);
        enableSpeedControl = false;
        currentSpeed = 75;
        lineLost = false;

      } else {
        if (!lineLost) {
          enableSpeedControl = false;
          currentSpeed = BASE_SPEED;
          lineLost = true;
          lineLostTime = millis();
        }
        if (millis() - lineLostTime > 5000) {
          setMotors(BASE_SPEED, TURN_SPEED);
        }
      }
      break;
```

```
  }
}

void runStateMachine() {
  switch (currentState) {
    case STATE_CALIBRATE:
      if (lineThreshold == 0) {
        calibrateSensors();
        currentState = STATE_WAIT_DESTINATION;
        Serial.println("Calibration complete");
      } else {
        currentState = STATE_WAIT_DESTINATION;
      }
      break;

    case STATE_WAIT_DESTINATION:
      enableSpeedControl = false;  // Гарантированный сброс
      if (destination != 0) {
        currentState = STATE_FOLLOW_LINE;
        Serial.println("Starting navigation");
      }
      break;

    case STATE_FOLLOW_LINE:  // Убрана проверка позиции в FOLLOW_LINE -
перекрестки только через QR
      // При возврате в режим следования сбрасываем скорость
      if (!enableSpeedControl) {
        currentSpeed = BASE_SPEED;
        enableSpeedControl = true;
        lastDecelerationTime = millis();
      }
      break;

    case STATE_STOP_AT_INTERSECTION:
      stopMotors();
      if (millis() - turnStartTime > STOP_DURATION) {
        if (currentPosition == destination) {
          currentState = STATE_ARRIVED;
          Serial.println("Destination reached");
        } else {
          determineNextMove();
          turnStartTime = millis();   // Критичный сброс таймера
          currentSpeed = BASE_SPEED;  // Сброс скорости перед поворотом
          currentState = STATE_TURNING;
          Serial.println("Starting navigation turn");
        }
      }
      break;

    case STATE_AT_INTERSECTION:
      // Переносим логику в STATE_STOP_AT_INTERSECTION
      currentState = STATE_STOP_AT_INTERSECTION;
      turnStartTime = millis();  // Засекаем время остановки
      break;

    case STATE_TURNING:
      {
        static bool turnMessageSent = false;
        static int slowMotorSpeed = TURN_SLOWDOWN_START;
        static unsigned long lastSlowdownTime = 0;
        static bool preMoveDone = false;
        static bool postMoveDone = false;
        static unsigned long phaseStart = 0;
```

```
        // При повороте жестко фиксируем скорость
        enableSpeedControl = false;

        // Для поворотов влево/вправо добавляем фазы движения
        if (abs(turnDirection) == 1 && !preMoveDone) {
          if (millis() - turnStartTime < PRE_TURN_MOVE_DURATION) {
            // Движение прямо перед поворотом
            setMotors(BASE_SPEED, BASE_SPEED);
            return;
          } else {
            preMoveDone = true;
            phaseStart = millis();
          }
        }

        if (!turnMessageSent) {
          slowMotorSpeed = TURN_SLOWDOWN_START;  // Сброс при старте поворота
          lastSlowdownTime = millis();
          Serial.print("Turning ");
          // Четкое определение типа поворота
          switch (turnDirection) {
            case 0: Serial.println("STRAIGHT (line follow)"); break;
            case 1: Serial.println("RIGHT"); break;
            case -1: Serial.println("LEFT"); break;
            case 2: Serial.println("U-TURN"); break;
          }
          turnMessageSent = true;
          turnStartTime = millis();

          // Автоматический расчет времени поворота
          int turnMultiplier = (turnDirection == 2) ? 2 : 1;
          turnDuration = TURN_BASE_DURATION * turnMultiplier;
        }


        // Особый случай: движение прямо с использованием алгоритма
следования
        if (turnDirection == 0) {
          handleSensors(analogRead(SENS_R_PIN), analogRead(SENS_L_PIN));
          currentSpeed = BASE_SPEED;
        }
        // Управление моторами для поворотов
        else {
          // Управление моторами с плавным замедлением
          if (turnDirection == 1) {  // Поворот направо
            if (millis() - lastSlowdownTime > TURN_SLOWDOWN_INTERVAL) {
              slowMotorSpeed = max(slowMotorSpeed - TURN_SLOWDOWN_STEP,
TURN_SLOWDOWN_MIN);
              lastSlowdownTime = millis();
            }
            setMotors(-slowMotorSpeed, TURN_SPEED);

          } else if (turnDirection == -1) {  // Поворот налево
            if (millis() - lastSlowdownTime > TURN_SLOWDOWN_INTERVAL) {
              slowMotorSpeed = max(slowMotorSpeed - TURN_SLOWDOWN_STEP,
TURN_SLOWDOWN_MIN);
              lastSlowdownTime = millis();
            }
            setMotors(TURN_SPEED, -slowMotorSpeed);

          } else if (turnDirection == 2) {  // Разворот
            setMotors(TURN_SPEED_MAIN, -TURN_SPEED_MAIN);
          }
```

```
          // После основного поворота добавляем движение прямо
          if (abs(turnDirection) == 1 && millis() - phaseStart >=
TURN_BASE_DURATION && !postMoveDone) {

            if (millis() - phaseStart < TURN_BASE_DURATION +
POST_TURN_MOVE_DURATION) {
              // Движение прямо после поворота
              setMotors(BASE_SPEED, BASE_SPEED);
              return;
            } else {
              postMoveDone = true;
            }
          }
        }

        // Плавный переход к линии
        if (millis() - turnStartTime > turnDuration * 0.9) {
          // Начинаем постепенно включать следящий алгоритм
          handleSensors(analogRead(SENS_R_PIN), analogRead(SENS_L_PIN));
        }

        // Завершение поворота
        if ((millis() - phaseStart >= turnDuration) || (abs(turnDirection) ==
1 && postMoveDone)) {

          stopMotors();
          currentState = STATE_FOLLOW_LINE;
          turnMessageSent = false;
          preMoveDone = false;
          postMoveDone = false;
          currentSpeed = BASE_SPEED;
          enableSpeedControl = true;
          lastDecelerationTime = millis();
        }
        break;
      }
      break;

    case STATE_DEAD_END:
      orientation = (orientation + 2) % 4;
      currentState = STATE_REVERSE;
      turnStartTime = millis();
      Serial.println("Dead end detected");
      // Специальный разворот
      setMotors(-BASE_SPEED, BASE_SPEED);  // Разворот на месте
      break;

    case STATE_REVERSE:
      if (millis() - turnStartTime >= TURN_DURATION) {
        stopMotors();
        // После разворота ищем новый путь
        determineNextMove();
        currentState = STATE_TURNING;
        turnStartTime = millis();
        Serial.println("Dead end turn completed");
      }
      break;

    case STATE_ARRIVED:
      stopMotors();
      break;
  }
```

```
}


// Вспомогательная функция для создания отступов
String createIndent(int depth) {
  String indent = "";
  for (int i = 0; i < depth; i++) {
    indent += "  ";  // 2 пробела на уровень глубины
  }
  return indent;
}


// Модифицированная функция determineNextMove
void determineNextMove() {
  Serial.println("\n=== Планирование маршрута ===");
  Path result = findShortestPath(currentPosition, destination);

  if (result.next_node == -1) {
    turnDirection = 2;
    Serial.println("Путь не найден! Разворот.");
  } else {
    // Абсолютное направление из карты
    int requiredDir = result.direction;
    // Преобразование абсолютного направления в относительное
    turnDirection = (result.direction - orientation + 4) % 4;
    if (turnDirection == 3) turnDirection = -1;  // Налево

    Serial.print("Направление: ");
    Serial.println(result.direction);
    // Обновление ориентации
    orientation = requiredDir;
  }

  // Обновление ориентации
  //orientation = (orientation + turnDirection + 4) % 4;
  Serial.print("Требуемый поворот: ");
  switch (turnDirection) {
    case 0: Serial.println("ПРЯМО"); break;
    case 1: Serial.println("НАПРАВО"); break;
    case -1: Serial.println("НАЛЕВО"); break;
    case 2: Serial.println("РАЗВОРОТ"); break;
  }
  Serial.print("Новая ориентация: ");
  printDirection(orientation);
  Serial.println("============================");
}

void setMotors(int leftSpeed, int rightSpeed) {
  digitalWrite(DIR_L_PIN, leftSpeed > 0 ? HIGH : LOW);
  digitalWrite(DIR_R_PIN, rightSpeed > 0 ? HIGH : LOW);
  analogWrite(PWR_L_PIN, abs(leftSpeed));
  analogWrite(PWR_R_PIN, abs(rightSpeed));
}

void stopMotors() {
  setMotors(0, 0);
}

void calibrateSensors() {
  Serial.println("Calibration started");

  // 1. Белая поверхность (первая!)
```

```
  Serial.println("1. Place on WHITE surface and press button");
  while (digitalRead(BTN_PIN) == LOW)
    ;
  color_white = (analogRead(SENS_L_PIN) + analogRead(SENS_R_PIN)) / 2;
  Serial.println(color_white);
  beep(1);

  delay(1000);

  // 2. Черная линия
  Serial.println("2. Place on BLACK line and press button");
  while (digitalRead(BTN_PIN) == LOW)
    ;
  color_black = (analogRead(SENS_L_PIN) + analogRead(SENS_R_PIN)) / 2;
  Serial.println(color_black);
  beep(2);

  // Проверка калибровки
  if (color_white >= color_black || abs(color_white - color_black) < 100) {
    Serial.println("CALIBRATION ERROR!");
    while (1) {
      beep(1);
      delay(1000);
    }
  }
  lineThreshold = (color_white + color_black) / 2;

  Serial.println("Calibration results:");
  Serial.print("White: ");
  Serial.println(color_white);
  Serial.print("Black: ");
  Serial.println(color_black);
  Serial.print("Threshold: ");
  Serial.println(lineThreshold);
}

void beep(int count) {
  for (int i = 0; i < count; i++) {
    digitalWrite(BUZZER, HIGH);
    delay(BUZZ_DURATION);
    digitalWrite(BUZZER, LOW);
    if (i < count - 1) delay(BUZZ_DURATION);
  }
}

bool rightOnLine() {
  return analogRead(SENS_R_PIN) > lineThreshold;
}

bool leftOnLine() {
  return analogRead(SENS_L_PIN) > lineThreshold;
}
```