ZARNITZA
ROBOTICS

Производственное объединение «Зарница»

КОМПЛЕКТ ДЛЯ ИЗУЧЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ И СИСТЕМ УПРАВЛЕНИЯ АВТОНОМНЫХ МОБИЛЬНЫХ РОБОТОВ OPTIMA-DRIVE

Методическое пособие

РОБОТОТЕХНИКА

ZARNITZA
ROBOTICS

Разрабатываем и производим
высокотехнологичное учебное
оборудование для любых специальностей

УП9147

Образовательный набор для изучения
многокомпонентных робототехнических систем
и манипуляционных роботов
Приказ 838 Минпросвещения РФ

УП9145

Базовый робототехнический набор
для конструирования,
изучения электроники и микропроцессоров
и информационных систем и устройств Z.Robo-2
Приказ 838 Минпросвещения РФ

УП6738

Стол для робототехники
Приказ 838 Минпросвещения РФ

Телефон 8-800-775-37-97
www.zarnitza.ru, zakaz@zrnc.ru

Производственное объединение «Зарница»

КОМПЛЕКТ ДЛЯ ИЗУЧЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ И СИСТЕМ УПРАВЛЕНИЯ АВТОНОМНЫХ МОБИЛЬНЫХ РОБОТОВ OPTIMA-DRIVE

Методическое пособие

2025

СОДЕРЖАНИЕ

МОБИЛЬНЫЙ РОБОТ OPTIMA-DRIVE. ЗНАКОМСТВО, УСТРОЙСТВО И СОСТАВ КОМПЛЕКТА OPTIMA-DRIVE

Optima-Drive – это мобильная платформа, оснащенная 4 независимыми всенаправленными колесами, приводимыми в движение с помощью индивидуальных моторов. Управление осуществляется с помощью связки контроллер + микрокомпьютер – это Arduino-совместимый контроллер + микрокомпьютер на базе Raspberry Pi 4 (или аналог). Мобильная платформа оснащена набором датчиков:

- 8 ультразвуковых датчиков
- Гироскоп + акселерометр + магнитометр
- Датчик линии Octoliner v2 8-канальный
- Лидар
- Курсовая видеокамера
- Pan-tilt камера (модуль технического зрения)

Габаритные размеры и конструктив мобильной платформы позволяет подключение и установку на нее робота-манипулятора Optima EDU*, в результате чего можно получить мобильный 5-осевой робот-манипулятор.

Общий вид мобильной платформы, а также вид с установленным на ней роботом-манипулятором на рисунках 1–3.

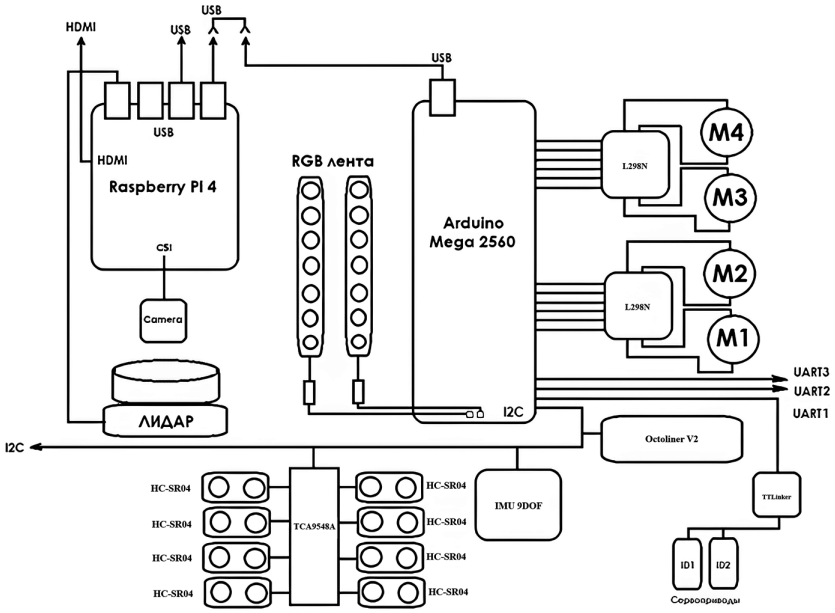
Рисунок 1.

*Базовый набор учебного манипулятора Optima EDU, производство ПО «Зарница».

Рисунок 2.

Рисунок 3. Модуль технического зрения:
1 – механизм крепления камеры Go-pro; 2 – модуль камеры; 3 – разъем интерфейса UART;
4 – разъем интерфейса I²C

В данной версии мобильной платформы Optima-Drive в качестве операционной системы используется связка Raspbian Buster + ROS Noetic. Дополнительно установлен пакет OpenCV.



Функциональная схема Optima-Drive

К шине I²C контроллера Arduino Mega 2560 подключены IMU 9DOF, Octoliner и ультразвуковые сенсоры, доступ к которым возможен через мультиплексор TCA9548A.

Адреса модулей:

Octoliner..... ID = 0x2A

IMU 9DOF..... ID =0x19 (A), 0x69 (G), 0x13 (M)

TCA9548A..... ID = 0x70

Адресная лента RGB (2 шт. по бокам платформы) состоит из 18 светодиодов, подключена к пину 44 (вторая к пину 46) Arduino Mega 2560.

Для управления двигателями мобильного робота Optima-Drive используются следующие цифровые пины контроллера Arduino Mega 2560:

Motor 1: IN1 = 3 IN2 = 4 ENA = 2

Motor 2: IN3 = 5 IN4 = 6 ENB = 7

Motor 3: IN1_1=9 IN1_2 = 10 ENA_1 = 8

Motor 4: IN1_3 = 11 IN1_4 = 12 ENB_1 = 13

ВНИМАНИЕ! Все приведенные скетчи лабораторных работ и библиотеки, необходимые для их работы, прилагаются на электронном носителе (флеш-накопителе) в папке «ПО», а также на рабочем столе Raspbian микрокомпьютера Raspberry Pi в папке python_code\.

ЧАСТЬ 1. ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

В современном мире встраиваемые системы становятся все более распространенными, и их применение охватывает широкий спектр областей – от бытовой электроники до сложных промышленных автоматизированных систем. В таких системах часто требуется выполнение множества задач одновременно, что ставит перед разработчиками серьезные вызовы.

Так, например, в современных автомобилях используются сложные системы управления, такие как ABS (антиблокировочная система тормозов), системы контроля устойчивости и адаптивные круиз-контроли. Эти системы требуют мгновенной реакции на изменения в окружающей среде для обеспечения безопасности.

Или, например, в системах управления полетом и навигации требуется высокая степень надежности и предсказуемости. Жизненно необходимо обрабатывать данные с датчиков и управлять системами в реальном времени, что критично для безопасности полетов.

И даже в столь популярной области, стремительно развивающейся сегодня – интернет вещей (англ. Internet of things, IoT), устройства IoT часто взаимодействуют друг с другом и требуют быстрой реакции на изменения в окружающей среде. Необходимо обеспечить эффективное управление ресурсами и связь между устройствами, чтобы сделать их более функциональными.

Здесь на помощь приходит операционная система реального времени (RTOS), которая обеспечивает эффективное управление ресурсами и временем выполнения задач. В этих областях использование RTOS становится не просто предпочтительным, а необходимым условием для успешной работы систем.

Таким образом, есть два основных вида операционных систем: общего назначения и реального времени. Системы общего назначения – это обычные операционные системы в привычном нам смысле: Windows, Linux и Mac OS. Они решают много задач сразу, но иногда могут зависать или тормозить. Но есть устройства и задачи, где на всё нужна моментальная реакция без права на ошибку или зависание. В них используют операционные системы реального времени.

Одной из ключевых характеристик RTOS является способность обрабатывать события в строго определенные временные рамки. В отличие от традиционных операционных систем, которые могут не гарантировать время отклика на события, RTOS обеспечивает предсказуемость и надежность. Это особенно важно в критически важных приложениях, где задержка в обработке данных может привести к серьезным последствиям.

Кроме того, RTOS позволяет разработчикам организовывать код в виде независимых задач, каждая из которых может выполняться параллельно. Это упрощает структуру программы и делает ее более модульной. Например, в системе умного дома одна задача может отвечать за управление освещением, другая – за мониторинг температуры, а третья – за взаимодействие с пользователем через мобильное приложение. Такой подход не только улучшает читаемость кода, но и облегчает его отладку и тестирование.

Еще одним важным аспектом является управление ресурсами. RTOS предоставляет механизмы для синхронизации задач и управления доступом к общим ресурсам, что предотвращает конфликты и обеспечивает стабильную работу системы. Использование семафоров, очередей и других средств синхронизации позволяет избежать состояния гонки и гарантирует корректное выполнение задач.

С точки зрения производительности, RTOS оптимизирует использование процессорного времени и памяти. Это особенно актуально для устройств с ограниченными ресурсами, где каждая милли-

секунда имеет значение. Благодаря эффективному планированию задач и минимизации накладных расходов на переключение контекста RTOS позволяет добиться высокой производительности даже на простых микроконтроллерах.

RTOS является достаточно надежной системой, лишенной некоторых недостатков обычной ОС. Например, в системе перемкнуло датчик управления, и одна из функций, которая за него отвечает, ушла в бесконечный цикл. В обычной ОС это привело бы к тому, что зависла бы программа или весь компьютер. В случае с RTOS это решается так:

1. Планировщик выполняет по очереди команды и доходит до команды с бесконечным циклом;
2. Выполняет одну команду оттуда, ставит эту задачу на паузу и идёт к другим задачам.

Получается, что бесконечный цикл тоже выполняется по очереди с другими – это значит, что он не будет мешать работе остальных задач. При этом, если планировщик достаточно умный, в какой-то момент он поймёт, что функция работает.

Итак, RTOS – это операционная система, разработанная для обслуживания приложений реального времени, которые обрабатывают данные по мере их поступления, как правило, без задержек буферизации. Она обеспечивает выполнение критических задач в течение определенного временного ограничения.

Операционные системы реального времени можно условно разделить на:

- системы жёсткого реального времени;
- системы мягкого реального времени.

ОС жёсткого реального времени – используются для решения задач, требующих выполнения за строго определённое время и не более, так как превышение этого времени может вызвать серьёзные критические последствия. Например, часть систем автопилотов в авиации работает на таких системах – они должны укладываться в норматив по времени срабатывания в любых нештатных ситуациях.

ОС мягкого реального времени – допускают выполнение задач с превышением указанного времени, то есть если оно не превышает определённых значений (например, 0,3 секунды).

В качестве примеров конкретной реализации RTOS можно привести VxWorks, QNX, Win CE, pSOS, Nucleus® RTOS, FreeRTOS™, Zephyr™ Project.

Таким образом, если вы разрабатываете достаточно сложную систему, которая обладает множеством функций, и эти функции являются достаточно разнородными: отображение информации на экране, межсетевой обмен, постоянный опрос ряда датчиков и пользовательских клавиатур и так далее, вам сможет помочь RTOS.

Одной из самых популярных операционных систем реального времени (RTOS) является **FreeRTOS** и имеет несколько преимуществ, которые делают её предпочтительным выбором для многих разработчиков. Вот некоторые из причин, почему FreeRTOS может быть лучшим выбором:

Легковесность: FreeRTOS спроектирована так, чтобы занимать минимальное количество ресурсов, что делает её идеальной для встраиваемых систем с ограниченными ресурсами. Размер ядра

FreeRTOS составляет всего 4–9 кбайт, в зависимости от типа платформы и настроек ядра. FreeRTOS написана на языке Си (исходный код ядра представлен в виде всего лишь четырех Си-файлов).

Открытый исходный код: FreeRTOS является проектом с открытым исходным кодом, что позволяет разработчикам изучать, изменять и адаптировать код под свои нужды без лицензионных ограничений.

Поддержка множества платформ: FreeRTOS поддерживает широкий спектр микроконтроллеров и архитектур, что делает её универсальным решением для различных проектов.

Простота использования: FreeRTOS имеет понятный API и хорошо документированную библиотеку, что облегчает процесс разработки и ускоряет время выхода на рынок.

Сообщество и поддержка: У FreeRTOS есть активное сообщество разработчиков и множество ресурсов, включая форумы, учебные пособия и примеры кода, что упрощает решение возникающих проблем.

Модульность: FreeRTOS позволяет разработчикам выбирать только те компоненты, которые им нужны, что помогает оптимизировать размер и производительность приложения.

Поддержка многозадачности: FreeRTOS предоставляет мощные механизмы для управления задачами, включая приоритеты задач, синхронизацию и межзадачное взаимодействие.

Интеграция с другими библиотеками: FreeRTOS легко интегрируется с другими библиотеками и инструментами разработки, такими как TCP/IP стеки или библиотеки для работы с сенсорами.

Коммерческая поддержка: Хотя FreeRTOS является проектом с открытым исходным кодом, существует возможность получения коммерческой поддержки от Amazon Web Services (AWS), что может быть полезно для крупных проектов или предприятий.

Совместимость с IoT: FreeRTOS имеет специальные расширения для работы с IoT-устройствами, что делает её отличным выбором для разработки приложений в этой области.

Эти факторы делают FreeRTOS привлекательным выбором для многих проектов встраиваемых систем и приложений реального времени.

Программирование Arduino с использованием операционных систем реального времени (RTOS), таких как FreeRTOS, позволяет разработчикам создавать более сложные и многозадачные приложения. Основная концепция заключается в том, чтобы использовать возможности RTOS для управления задачами, синхронизации и взаимодействия между ними. Ниже приведены основные понятия и модули, которые стоит учитывать.

Основные понятия

Задача (Task): В RTOS задача – это основной элемент выполнения программы. Каждая задача выполняется в своем контексте и может иметь свой приоритет. Задачи могут выполняться параллельно, что позволяет более эффективно использовать ресурсы микроконтроллера.

Приоритеты задач: Каждой задаче можно назначить приоритет, что позволяет системе определять порядок выполнения задач. Задачи с более высоким приоритетом будут выполняться раньше задач с более низким приоритетом.

Планировщик (Scheduler): Это компонент RTOS, который управляет выполнением задач на основе их приоритетов и состояния (готова к выполнению, заблокирована и т. д.).

Синхронизация: Для взаимодействия между задачами используются механизмы синхронизации, такие как **семафоры** и **мьютексы**. Они помогают избежать конфликтов при доступе к общим ресурсам.

Очереди (Queues): Очереди позволяют передавать данные между задачами безопасным образом. Одна задача может помещать данные в очередь, а другая – извлекать их.

Таймеры: RTOS предоставляет возможность работы с таймерами для выполнения задач через определенные интервалы времени или для создания задержек.

Основные модули

FreeRTOS: Это библиотека RTOS, которую можно интегрировать в проекты Arduino. Она предоставляет API для создания задач, управления ими и синхронизации.

Task Management: Модуль управления задачами позволяет создавать, удалять и управлять состоянием задач.

Inter-task Communication: Модули для очередей и семафоров обеспечивают безопасное взаимодействие между задачами.

Memory Management: FreeRTOS включает механизмы управления памятью для динамического выделения памяти под задачи и другие объекты.

Timers and Delays: Модули таймеров позволяют выполнять задачи по расписанию или через заданные интервалы времени.

Задачи в FreeRTOS представляют собой функции, которые могут выполняться одновременно благодаря планировщику. Каждая задача имеет свой приоритет, и планировщик определяет, какая задача должна выполняться в конкретный момент времени. Это позволяет управлять многозадачностью, выделяя время для выполнения каждой задачи на основе её приоритета. Создание задачи осуществляется с помощью функции, в которой указываются параметры, такие как имя задачи, приоритет, объём выделяемой памяти и другие важные характеристики.

Планировщик является основным компонентом FreeRTOS, который управляет переключением между задачами. Чтобы дать возможность другим задачам выполняться, разработчик должен использовать функции задержки, такие как vTaskDelay(). Эта функция приостанавливает выполнение текущей задачи на определённое время, освобождая процессор для выполнения других задач. Это особенно важно в условиях ограниченных ресурсов микроконтроллера, чтобы обеспечить плавную и эффективную работу всех задач.

Для организации взаимодействия между задачами в FreeRTOS используются очереди. Очереди позволяют передавать данные между задачами, что способствует синхронизации их работы. Это особенно полезно, когда одна задача генерирует данные, а другая должна их обработать. Очереди помогают аккумулировать данные и обеспечивать их корректную обработку в нужный момент, предотвращая потерю информации.

Для синхронизации задач в FreeRTOS также применяются семафоры. Семафоры бывают двух типов: бинарные и счётные. Бинарные семафоры используются для управления доступом к ресурсам, когда только одна задача может получить доступ к ним в определённый момент. Это особенно полезно для синхронизации между задачами и прерываниями. Счётные семафоры позволяют учитывать ко-

личество произошедших событий, что полезно в случаях, когда нужно обработать несколько событий, произошедших за определённый период времени.

Мьютексы в FreeRTOS применяются для предотвращения одновременного доступа нескольких задач к общим ресурсам. Это особенно важно, когда задачи могут конфликтовать при совместном использовании ресурсов, таких как периферийные устройства. Мьютексы позволяют одной задаче заблокировать ресурс, чтобы другие задачи не могли его использовать, пока первая задача не завершит свою работу. Это обеспечивает целостность данных и предотвращает возможные ошибки, вызванные конкурентным доступом.

На микроконтроллерах ESP32, которые имеют два ядра, можно распределять задачи между ядрами для повышения общей производительности системы. FreeRTOS позволяет создавать задачи, привязанные к конкретному ядру, что позволяет эффективно использовать ресурсы контроллера и выполнять задачи параллельно на обоих ядрах. Это особенно полезно в случаях, когда необходимо выполнить вычислительно сложные операции или обеспечить одновременное выполнение нескольких задач с высокой приоритетностью.

Лабораторная работа № 1 с кодом для Arduino демонстрирует создание двух задач с использованием FreeRTOS.

ЛАБОРАТОРНАЯ РАБОТА

№ 1 ДЕМОНСТРАЦИЯ ПРЕИМУЩЕСТВ FREERTOS – ЗАПУСК ДВУХ ЗАДАЧ «ОДНОВРЕМЕННО»

Цель работы: Протестировать возможность запуска двух задач на контроллере Arduino одновременно.

Задачи: Создадим с помощью кода две задачи для контроллера: одна должна мигать светодиодом, а другая должна отправлять сообщения в последовательный порт.

Объяснение кода

Инициализация: В функции setup() мы инициализируем пин для светодиода и создаем две задачи с помощью xTaskCreate(). Каждая задача имеет свой стек и приоритет.

Для создания задачи мы используем следующую команду:

xTaskCreate(TaskFunction_t pvTaskCode, const char * const pcName, uint16_t usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pxCreatedTask);

Данная функция имеет 6 аргументов:

1. **pvTaskCode** – указатель на функцию, которая будет выполнять эту задачу.
2. **pcName** – описательное имя для задачи, нигде не используется системой FreeRTOS.
3. **usStackDepth** – количество памяти, выделяемой под данную задачу. Одно слово занимает 4 байта. Таким образом, число слов, указанных в данной строке, необходимо умножить на 4 байта для определения выделяемой памяти. Количество выделяемой памяти очень актуально, если использовать микроконтроллеры с очень маленькой оперативной памятью.
4. **pvParameters** – входной параметр задачи, при отсутствии параметров пишем NULL.
5. **uxPriority** – приоритет задачи, от 0 (наименьший) и выше.
6. **pxCreatedTask** – используют для того, чтобы пропустить обработчик созданной задачи или для изменения приоритета задачи или ее удаления. Если нам это не нужно, пишем NULL.

Для того, чтобы планировщик мог распределять работу между программами, необходимо в программу вставить задержку, которая приостанавливает выполнения данной программы на определенное количество тиков:

vTaskDelay(const TickType_t xTicksToDelay);

Так как время тиков в разных случаях может быть разным, то для задания точного времени блокировки выполнения задачи воспользуемся следующей записью:

vTaskDelay(N / portTICK_PERIOD_MS);

где N – время в миллисекундах, на которое необходимо сделать паузу.

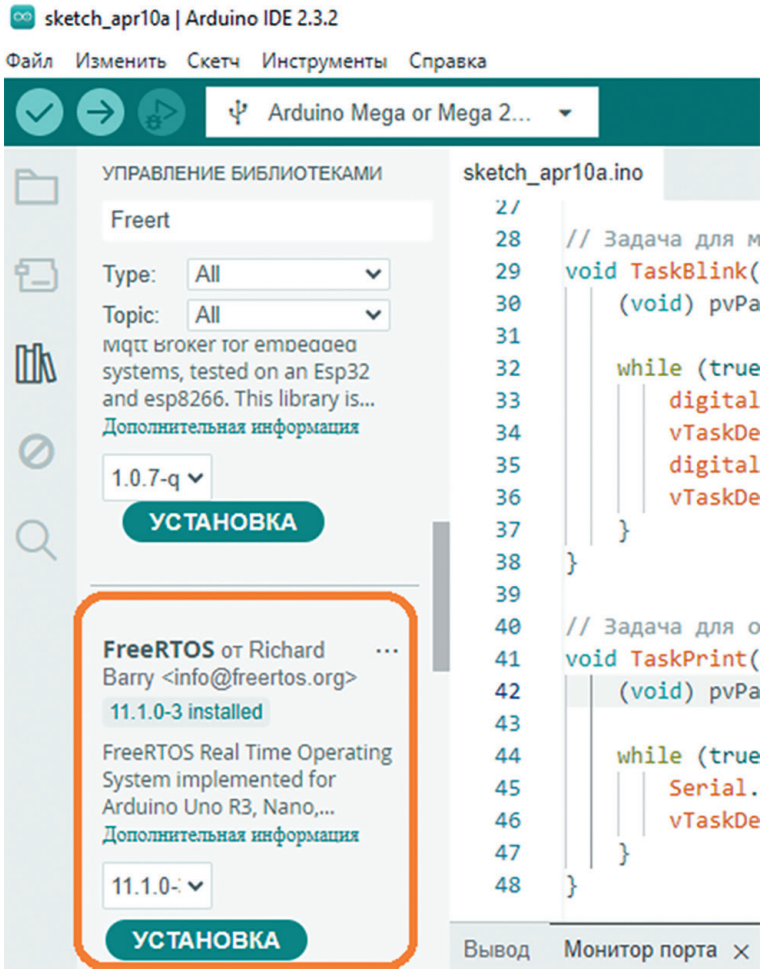
Задача TaskBlink: Эта задача мигает светодиодом с интервалом в 500 мс. Она использует vTaskDelay() для приостановки выполнения на заданное время.

Задача TaskPrint: Эта задача отправляет сообщение в последовательный порт с интервалом в 500 мс. Она также использует vTaskDelay() для управления временем выполнения.

Планировщик: После создания задач мы запускаем планировщик FreeRTOS с помощью vTaskStartScheduler(). Основной цикл loop() остается пустым, так как управление передается задачам.

Порядок выполнения работы

Для начала необходимо установить библиотеку FreeRTOS:



Далее загрузить код программы в контроллер Arduino

```
#include <Arduino_FreeRTOS.h> // подключаем установленную библиотеку

// Определяем пин для светодиода
const int ledPin = 13;

// Объявление функций для задач
void TaskBlink(void *pvParameters);
void TaskPrint(void *pvParameters);
```

```
void setup() {
  Serial.begin(9600);
  // Инициализация пина светодиода
  pinMode(ledPin, OUTPUT);

  // Создание задач
  xTaskCreate(TaskBlink, "Blink", 128, NULL, 1, NULL);
  xTaskCreate(TaskPrint, "Print", 128, NULL, 1, NULL);

  // Запуск планировщика FreeRTOS
  vTaskStartScheduler();
}

void loop() {
  // Основной цикл не используется в FreeRTOS
}

// Задача для мигания светодиодом
void TaskBlink(void *pvParameters) {
  (void) pvParameters; // Убираем предупреждение о неиспользуемом параметре

  while (true) {
    digitalWrite(ledPin, HIGH);   // Включаем светодиод
    vTaskDelay(500 / portTICK_PERIOD_MS); // Ждем 500 мс
    digitalWrite(ledPin, LOW);    // Выключаем светодиод
    vTaskDelay(500 / portTICK_PERIOD_MS); // Ждем 500 мс
  }
}

// Задача для отправки сообщений в последовательный порт
void TaskPrint(void *pvParameters) {
  (void) pvParameters; // Убираем предупреждение о неиспользуемом параметре

  while (true) {
    Serial.println("Hello from FreeRTOS!");
    vTaskDelay(500 / portTICK_PERIOD_MS); // Ждем 500 мс
  }
}
```

Теперь, запустив Monitor в Arduino IDE, можно убедиться, что задачи выполняются практически одновременно.

ЛАБОРАТОРНАЯ РАБОТА

№ 2 СЕМАФОРЫ

Цель работы: Смоделировать на практике и изучить процесс синхронизации между задачами и обработчиком прерывания с помощью семафора.

Задачи: Написать код программы для Arduino на примере представленного, реализующий алгоритм синхронизации между двумя задачами.

Порядок выполнения работы

• Для начала изучите приведённый пример.

• На базе представленного примера напишите код программы - реализовать систему, которая управляет несколькими светодиодами с различными режимами мигания:

- задача для управления каждым светодиодом (например, мигание с разной частотой).

- задача для обработки пользовательского ввода (например, кнопки для изменения режима мигания).

Используемые элементы:

Семафоры для синхронизации доступа к общим ресурсам (например, состоянию светодиодов).

Очереди для передачи сообщений между задачами (например, изменение режима мигания).

1. Добавление зависимостей:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
```

Подключаются заголовочные файлы Arduino_FreeRTOS.h и semphr.h для работы с FreeRTOS и семафорами. FreeRTOS позволяет реализовать многозадачность, а семафоры используются для синхронизации.

2. Объявление глобальных переменных:

```
long d_time = 150;
volatile unsigned long last_micros;
SemaphoreHandle_t interruptSemaphore;
```

• d_time – время (в миллисекундах) для задержки между срабатываниями прерывания.

• last_micros – переменная для хранения времени последнего срабатывания прерывания.

• interruptSemaphore – семафор, который используется для синхронизации между задачами и обработчиком прерывания.

3. Функция setup:

```
void setup() {
  pinMode(2, INPUT_PULLUP);
  xTaskCreate(TaskLed, "Led", 128, NULL, 0, NULL);
```

```
  xTaskCreate(TaskBlink, "LedBlink", 128, NULL, 0, NULL);
  interruptSemaphore = xSemaphoreCreateBinary();
  if (interruptSemaphore != NULL) {
    attachInterrupt(digitalPinToInterrupt(2), Interrupt, LOW);
  }
}
```

Настраивается пин 2 как вход с подтяжкой (INPUT_PULLUP), чтобы реагировать на внешнее событие.

Создаются две задачи:

• TaskLed – задача для управления светодиодом.

• TaskBlink – задача для мигания другим светодиодом.

Создается бинарный семафор interruptSemaphore.

Если семафор успешно создан, к пину 2 подключается внешнее прерывание, которое срабатывает на низком уровне (LOW).

4. Функция loop:

```
void loop() {}
```

Поскольку FreeRTOS управляет задачами, функция loop остается пустой.

5. Обработчик прерывания interruptHandler

```
void interruptHandler() {
  xSemaphoreGiveFromISR(interruptSemaphore, NULL);
}
```

Эта функция вызывается при возникновении прерывания, и она «освобождает» (дает) семафор, используя функцию xSemaphoreGiveFromISR, чтобы задача могла быть разблокирована.

6. Задача TaskLed:

```
void TaskLed(void *pvParameters) {
  (void) pvParameters;
  pinMode(8, OUTPUT);
  for (;;) {
    if (xSemaphoreTake(interruptSemaphore, portMAX_DELAY) == pdPASS) {
      digitalWrite(8, !digitalRead(8));
    }
  }
}
```

Настраивает пин 8 как выход. Задача ожидает семафор с помощью функции xSemaphoreTake с параметром portMAX_DELAY, что означает, что она будет ждать семафор бесконечно. После получения семафора переключает состояние светодиода на пине 8.

7. Задача TaskBlink:

```
void TaskBlink(void *pvParameters) {
      (void) pvParameters;
      pinMode(9, OUTPUT);
      for (;;) {
                  digitalWrite(9, HIGH);
                  vTaskDelay(500 / portTICK_PERIOD_MS);
                  digitalWrite(9, LOW);
                  vTaskDelay(500 / portTICK_PERIOD_MS);
            }
}
```

Настраивает пин 9 как выход. В бесконечном цикле мигает светодиодом на пине 9: 500 мс включено, 500 мс выключено, используя функцию vTaskDelay для задержки.

8. Обработчик прерывания Interrupt:

```
 void Interrupt() {
     if ((long)(micros() - last_micros) >= d_time * 1000) {
         interruptHandler();
         last_micros = micros();
     }
 }
```

Проверяет, прошло ли достаточное количество времени (d_time в микросекундах) с момента последнего вызова прерывания, чтобы избежать дребезга контактов.

Если условие выполняется, вызывает interruptHandler() и обновляет переменную last_micros текущим временем.

ЛАБОРАТОРНАЯ РАБОТА

№ 3 МЬЮТЕКСЫ

Цель работы: Убедиться, что мьютекс позволяет выполнять критические секции задач последовательно, предотвращая одновременный доступ и предотвращая ошибки в работе с последовательным портом и светодиодами.

Задачи: Изучить и закрепить понимание необходимости использования мьютексов.

Порядок выполнения работы

• Изучить пример кода, представленного ниже.

• Написать программу, реализующую предотвращение одновременного доступа к общему ресурсу.

Необходимо создать систему, в которой несколько задач будут пытаться записывать данные в общий буфер. Для предотвращения конфликтов при записи в буфер нужно использовать мьютексы, то есть допустим, что есть несколько задач, которые генерируют данные и записывают их в общий буфер. Чтобы избежать одновременной записи в буфер (что может привести к повреждению данных), необходимо использовать мьютекс для синхронизации доступа к этому ресурсу.

Общий ресурс: Необходимо создать общий массив buffer и переменную count, чтобы отслеживать количество записанных данных.

Мьютекс: создать мьютекс xMutex с помощью xSemaphoreCreateMutex(), который будет использоваться для защиты доступа к общему буферу.

Задача генерации данных (vDataGeneratorTask):
Эта задача генерирует данные (просто увеличивает счетчик).
Перед записью данных в буфер она захватывает мьютекс с помощью xSemaphoreTake().
После записи она освобождает мьютекс с помощью xSemaphoreGive().

Задача обработки данных (vDataProcessorTask):
Эта задача читает данные из буфера.
Она также захватывает мьютекс перед чтением и освобождает его после завершения операции.

Планировщик: В конце программы запускается планировщик FreeRTOS с помощью vTaskStartScheduler().

Код программы для примера

1. Добавление зависимостей:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
```

Подключаются библиотеки Arduino_FreeRTOS.h и semphr.h для работы с многозадачностью и семафорами.

2. Объявление глобальных переменных:

```
SemaphoreHandle_t mutex;
```

Создается mutex – это мьютекс (семафор взаимного исключения), используемый для синхронизации доступа к общим ресурсам между несколькими задачами.

3. Функция setup:

```
void setup() {
  Serial.begin(9600);
  mutex = xSemaphoreCreateMutex();
  if(mutex == NULL){
    Serial.println("Mutex can not be created");
  }
  xTaskCreate(Task1,"Task1",128, NULL,1,NULL);
  xTaskCreate(Task2,"Task2",128, NULL,1,NULL);
  xTaskCreate(Task3,"Task3",128, NULL,1,NULL);
  xTaskCreate(TaskBlink,"TaskBlink",128, NULL,1,NULL);
}
```

Настраивается последовательный порт для вывода сообщений на скорость 9600 бод.

Создается мьютекс mutex с помощью функции xSemaphoreCreateMutex(). Если мьютекс создать не удалось, выводится сообщение об ошибке Mutex can not be created.

Создаются четыре задачи (Task1, Task2, Task3, TaskBlink), каждая из которых получает 128 байт памяти и приоритет 1.

4. Функция loop:

```
void loop() {}
```

Поскольку FreeRTOS управляет задачами, функция loop остается пустой.

5. Задача Task1:

```
void Task1(void *pvParameters) {
  pinMode(10, OUTPUT);
  while(1)
  {
    xSemaphoreTake(mutex,portMAX_DELAY);
    Serial.println("TASK1");
    digitalWrite(10, HIGH);
    vTaskDelay( 500 / portTICK_PERIOD_MS );
    digitalWrite(10, LOW);
    xSemaphoreGive(mutex);
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}
```

Настраивает пин 10 как выход. Бесконечно выполняет следующие действия:

- Получает мьютекс (xSemaphoreTake(mutex, portMAX_DELAY)), блокируя ресурс для других задач;
- Выводит сообщение TASK1 в последовательный порт;
- Включает светодиод на пине 10 на 500 мс, затем выключает его;
- Освобождает мьютекс (xSemaphoreGive(mutex));
- Задержка перед следующим циклом – 1000 мс.

6. Задача Task2:

```
void Task2(void *pvParameters) {
  pinMode(9, OUTPUT);
  while(1)
  {
    xSemaphoreTake(mutex,portMAX_DELAY);
    Serial.println("TASK2");
    digitalWrite(9, HIGH);
    vTaskDelay( 500 / portTICK_PERIOD_MS );
    digitalWrite(9, LOW);
    xSemaphoreGive(mutex);
    vTaskDelay(pdMS_TO_TICKS(500));
  }
}
```

Настраивает пин 9 как выход. Бесконечно выполняет:

- Получение мьютекса;
- Вывод сообщения TASK2 в последовательный порт;
- Включение светодиода на пине 9 на 500 мс, затем выключение;
- Освобождение мьютекса;
- Задержка перед следующим циклом – 500 мс.

7. Задача Task3:

```
void Task3(void *pvParameters) {
  pinMode(8, OUTPUT);
  while(1)
  {
    xSemaphoreTake(mutex,portMAX_DELAY);
    Serial.println("TASK3");
    digitalWrite(8, HIGH);
    vTaskDelay( 500 / portTICK_PERIOD_MS );
    digitalWrite(8, LOW);
    xSemaphoreGive(mutex);
    vTaskDelay(pdMS_TO_TICKS(2000));
  }
}
```

Настраивает пин 8 как выход. Бесконечно выполняет:

- Получение мьютекса;
- Вывод сообщения TASK3 в последовательный порт;
- Включение светодиода на пине 8 на 500 мс, затем выключение;
- Освобождение мьютекса;
- Задержка перед следующим циклом – 2000 мс.

8. Задача TaskBlink:

```
void TaskBlink(void *pvParameters) {
  (void) pvParameters;
  pinMode(LED_BUILTIN, OUTPUT);
  for (;;)
  {
    digitalWrite(LED_BUILTIN, HIGH);
    vTaskDelay( 500 / portTICK_PERIOD_MS );
    digitalWrite(LED_BUILTIN, LOW);
    vTaskDelay( 500 / portTICK_PERIOD_MS );
  }
}
```

Настраивает встроенный светодиод (LED_BUILTIN) как выход. В бесконечном цикле мигает встроенным светодиодом с интервалом 500 мс (включение и выключение).

9. Работа с мьютексом:

Использование мьютекса (mutex) позволяет исключить конкуренцию за доступ к общим ресурсам, такими как последовательный порт и выводы.

xSemaphoreTake(mutex, portMAX_DELAY) блокирует выполнение задачи до тех пор, пока мьютекс не станет доступным.

xSemaphoreGive(mutex) освобождает мьютекс, давая возможность другим задачам войти в критическую секцию.

10. Работа планировщика:

FreeRTOS управляет задачами, каждая из которых имеет одинаковый приоритет (1).

Задачи Task1, Task2, Task3 и TaskBlink выполняются в многозадачном режиме.

Мьютекс позволяет выполнять критические секции задач последовательно, предотвращая одновременный доступ и предотвращая ошибки в работе с последовательным портом и светодиодами.

ЧАСТЬ 2. УПРАВЛЕНИЕ МОБИЛЬНОЙ ПЛАТФОРМОЙ. СВЯЗКА RASPBERRY PI 4 + ARDUINO MEGA 2560

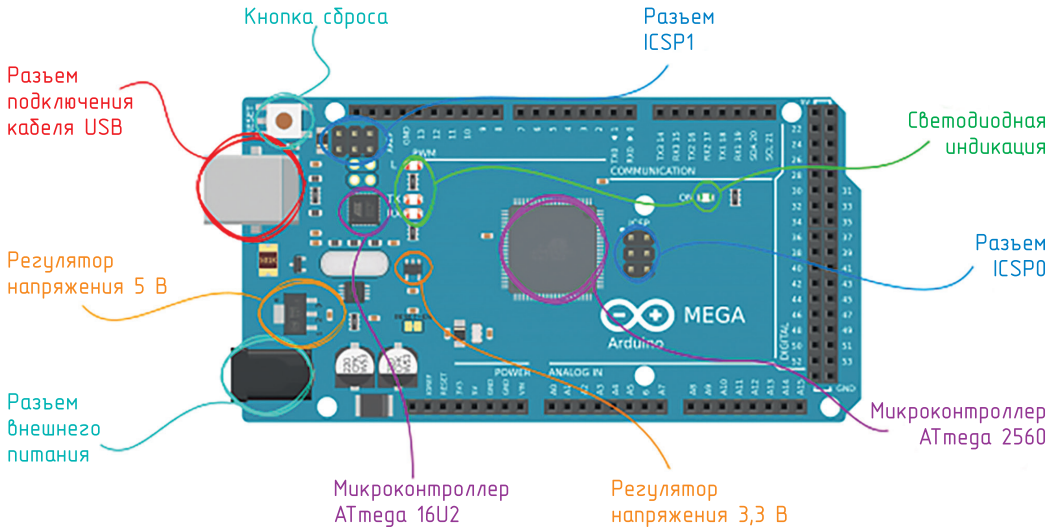
Управление мобильной платформой Optima-Drive осуществляется с помощью связки контроллер + микрокомпьютер – это Arduino-совместимый контроллер + микрокомпьютер на базе Raspberry Pi 4 (или аналог).

Мобильная платформа оснащена набором датчиков:

- 8 ультразвуковых датчиков;
- Гироскоп + акселерометр + магнитометр;
- Датчик линии Octoliner v2 8-канальный;
- Лидар;
- Курсовая видеокамера;
- Pan-tilt камера (модуль технического зрения).

Наличие на борту мобильной платформы как контроллера, так и микрокомпьютера позволяет использовать ее как для начинающих – программирование Arduino (движение платформы по линии, объезд препятствий, работа с гироскопом и т. д.), так и для продвинутых пользователей, подключив Arduino к Raspberry Pi для максимального использования вычислительных мощностей микрокомпьютера (использование удаленного подключения и управления по беспроводному интерфейсу Bluetooth, Wi-Fi, использование лидара и системы технического зрения – для ориентации в пространстве, построения карт помещений, возможности различать статичные и динамичные объекты и притормаживать перед препятствиями и т. д.).

Итак, в качестве контроллера в платформе используется Arduino Mega 2560. Этот контроллер имеет на борту достаточное количество необходимых ресурсов для подключения всех необходимых внешних устройств.



Сердцем платформы Arduino Mega является 8-битный микроконтроллер семейства AVR – ATmega2560 с тактовой частотой 16 МГц. Контроллер предоставляет 256 Кб Flash-памяти для хранения прошивки, 8 Кб оперативной памяти SRAM и 4 Кб энергонезависимой памяти EEPROM для хранения данных.

Порты ввода/вывода:

• Цифровые входы/выходы: пины 0–53.

Логический уровень единицы – 5 В, нуля – 0 В. Максимальный ток выхода – 40 мА. К контактам подключены подтягивающие резисторы, которые по умолчанию выключены, но могут быть включены программно;

• ШИМ: пины 2–13 и 44–46.

Позволяет выводить аналоговые значения в виде ШИМ-сигнала. Разрядность ШИМ не меняется и установлена в 8 бит;

• АЦП: пины A0–A16.

Позволяет представить аналоговое напряжение в цифровом виде. Разрядность АЦП не меняется и установлена в 10 бит. Диапазон входного напряжения от 0 до 5 В. При подаче большего напряжения – вы «убьёте» микроконтроллер;

• TWI/I²C: пины 20(SDA) и 21(SCL).

Для общения с периферией по интерфейсу I²C. Для работы используйте библиотеку Wire;

• SPI: пины 50(MISO), 51(MOSI), 52(SCK) и 53(SS).

Для общения с периферией по интерфейсу SPI. Для работы – используйте библиотеку SPI;

• UART: пины 0(RX) и 1(TX), 19(RX1) и 18(TX1), 17(RX2) и 16(TX2), 15(RX3) и 14(TX3).

Используется для коммуникации платы Arduino с компьютером или другими устройствами по последовательному интерфейсу. Выводы 0(RX) и 1(TX) соединены с соответствующими выводами микроконтроллера ATmega16U2, выполняющего роль USB-UART преобразователя. Для работы с последовательным интерфейсом – используйте методы библиотеки Serial.

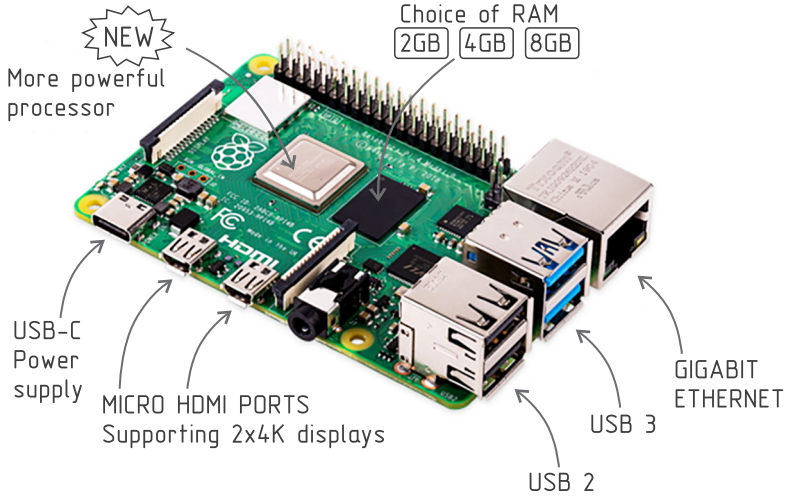
Характеристики:

- Микроконтроллер: ATmega2560;
- Ядро: 8-битный AVR;
- Тактовая частота: 16 МГц;
- Объём Flash-памяти: 256 Кб (8 Кб занимает загрузчик);
- Объём SRAM-памяти: 8 Кб;
- Объём EEPROM-памяти: 4 Кб;
- Портов ввода-вывода всего: 54;
- Портов с АЦП: 16;
- Разрядность АЦП: 10 бит;
- Портов с ШИМ: 15;
- Разрядность ШИМ: 8 бит;
- Аппаратных интерфейсов SPI: 1;
- Аппаратных интерфейсов I²C / TWI: 1;
- Аппаратных интерфейсов UART / Serial: 4;
- Номинальное рабочее напряжение: 5 В;
- Максимальный выходной ток пина 5V: 800 мА;
- Максимальный выходной ток пина 3V3: 150 мА;
- Максимальный ток с пина или на пин: 40 мА;

- Допустимое входное напряжение от внешнего источника: 7–12 В;
- Габариты: 101×53 мм.

Контроллер Arduino Mega 2560 используется для подключения внешних сенсоров и управления двигателями мобильной платформы. Это «спинной мозг» робота, который в первую очередь отвечает за движение и безопасность.

Однако для таких функциональных возможностей, как построение карт помещений при расчете маршрута к заданной точке, возможности различать статичные и динамичные объекты, для ориентации в пространстве, ну и для координации работы всех бортовых систем необходимо намного больше вычислительных мощностей. Такие задачи для Optima-Drive решает микрокомпьютер на базе Raspberry Pi 4. Кроме того, Raspberry Pi имеет встроенные беспроводные интерфейсы Wi-Fi и Bluetooth, которые используются для удаленного взаимодействия с мобильной платформой – получения информации от платформы (например, видео с курсовой камеры) и управления движением, например, управлением установленным на мобильной платформе роботом-манипулятором Optima.



Итак, используя Raspberry Pi 4, мы получаем мощный и универсальный микрокомпьютер, который идеально подходит для управления мобильными роботами. В последние годы Raspberry Pi завоевал популярность среди разработчиков и энтузиастов благодаря своей доступности, гибкости и широкому сообществу. Давайте рассмотрим основные преимущества использования Raspberry Pi 4 в качестве контроллера для мобильного робота.

1. Высокая производительность

Raspberry Pi 4 оснащен четырехъядерным процессором ARM Cortex-A72 с тактовой частотой до 1,5 ГГц и до 8 Гб оперативной памяти. Это обеспечивает достаточную вычислительную мощность для выполнения сложных алгоритмов обработки данных, таких как компьютерное зрение и машинное обучение, что особенно важно для автономных мобильных роботов.

2. Подключение и расширяемость

Raspberry Pi 4 предлагает множество интерфейсов для подключения различных датчиков и модулей: GPIO, I2C, SPI, UART и USB. Это позволяет легко интегрировать камеры, ультразвуковые датчики, гироскопы и другие устройства, необходимые для навигации и взаимодействия с окружающей средой.

3. Поддержка операционных систем

Raspberry Pi 4 поддерживает различные операционные системы, включая Raspbian (основанный на Debian), Ubuntu и другие дистрибутивы Linux. Это дает возможность использовать широкий спектр программного обеспечения и библиотек для разработки приложений, таких как OpenCV для обработки изображений или ROS (Robot Operating System) для разработки робототехнических систем.

4. Беспроводная связь

С встроенными модулями Wi-Fi и Bluetooth Raspberry Pi 4 позволяет легко организовать беспроводное управление роботом и обмен данными с другими устройствами. Это открывает возможности для удаленного мониторинга, управления через интернет или интеграции с облачными сервисами.

6. Компактные размеры

С размерами всего 85,6 x 56,5 мм Raspberry Pi 4 легко помещается в ограниченные пространства мобильного робота, что позволяет создавать компактные конструкции без ущерба для производительности.

Характеристики:

- Однокристальная система: SoC Broadcom BCM2711;
- Центральный процессор: 4-ядерный 64-битный CPU на ARM Cortex A72 с тактовой частотой 1,5 ГГц;
- Графический процессор: VideoCore VI GPU с тактовой частотой 500 МГц;
- Оперативная память: 4 Гб LPDDR4-3200 SDRAM;
- Стандарт Wi-Fi: 802.11 b/g/n/ac;
- Стандарт Bluetooth: v5.0 с BLE;
- Частотный диапазон: 2,4 / 5 ГГц;
- Цифровой аудио/видеовыход: 2× micro-HDMI версии 2.0;
- Максимальное выходное разрешение: 2160p (60 Гц);
- Максимальное разрешение в режиме двух мониторов: 2160p (30 Гц);
- Аналоговый аудио/видеовыход: 4-контактный мини-джек 3,5 мм;
- Порты для периферии: 2× USB 2.0, 2× USB 3.0;
- Порт для камеры: MIPI CSI (15 пинов, шаг 1 мм);
- Порт для дисплея: MIPI DSI (15 пинов, шаг 1 мм);
- Карта памяти: microSD;
- Порты ввода-вывода GPIO: 40;
- Напряжение питания: 5 В;
- Максимальный ток потребления: 3 А;
- Габариты: 85×56×17 мм.

По умолчанию Raspberry Pi поддерживает SD карты до 32 Гб памяти, однако для работы с этим устройствами можно отформатировать и большую емкость. Учитывайте, что для установки официальной ОС Raspbian вам понадобится карта microSD емкостью не менее 8 Гб, тогда как для Raspbian Lite требуется минимум 4 Гб.

Управляющий модуль мобильной платформы – микрокомпьютер Raspberry Pi. И для работы с ним, как и с любым другим компьютером, необходимо установить операционную систему. В нашем случае на Raspberry Pi установлена Raspberry Pi OS (ранее Raspbian) – основанная на Debian операционная система для Raspberry Pi. Существует несколько версий Raspbian, в том числе Raspbian Stretch и Raspbian Jessie. С 2015 года Raspbian официально представлена Raspberry Pi Foundation в качестве основной операционной системы для одноплатных компьютеров Raspberry Pi. Raspbian была создана Майком Томпсоном и Питером Грином в качестве независимого проекта. Первоначальная сборка была выполнена в июне 2012 года. Операционная система находится в стадии активной разработки. Raspbian оптимизирована для низкопроизводительных процессоров ARM, используемых в линейке компьютеров Raspberry Pi.

В нашем случае установлена ОС Raspbian Buster.

```
PRETTY_NAME=»Raspbian GNU/Linux 10 (buster)»
NAME=»Raspbian GNU/Linux»
VERSION_ID=»10»
VERSION=»10 (buster)»
VERSION_CODENAME=buster
ID=raspbian
```

Версия установленной ROS – noetic, версия установленного пакета OpenCV:

```
#python3 -c «import cv2; print(cv2.__version__)»
#3.2.0
```

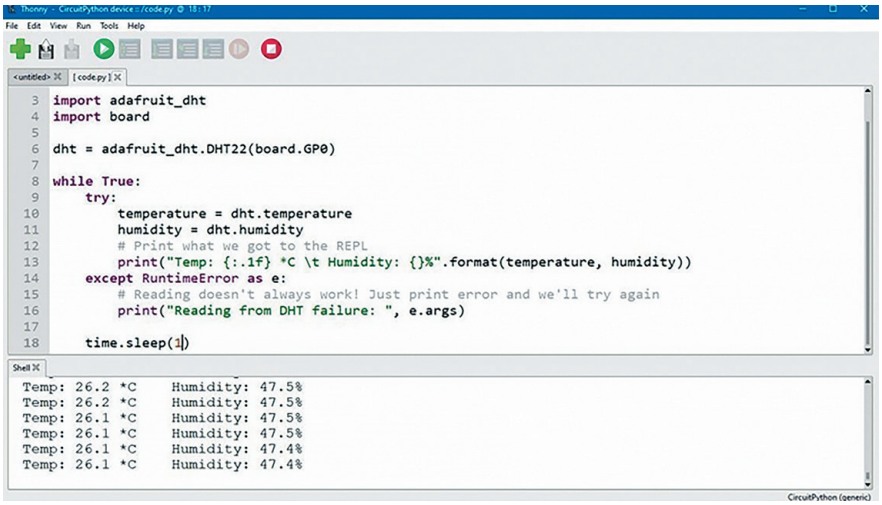
Если вы начинаете изучать программирование роботов с нуля, то Python, вероятно, самый быстрый и эффективный способ начать работу.

Во-первых, Python – один из самых простых языков для освоения.

Другая причина заключается в том, что существует множество библиотек, написанных на Python для различных датчиков и компонентов. В результате, большое количество скриптов для учебников и проектов будет написано на Python. Если бы вы использовали другой язык, например, JavaScript (через NodeJS), вы могли бы оказаться в затруднительном положении без библиотеки для обычного датчика.

Наконец, изучение языка Python поможет вам взаимодействовать со всеми платами линейки Raspberry Pi. Вы сможете создавать проекты как для Raspberry Pi 4, так и для Raspberry Pi Pico. Создатели Raspberry Pi также упрощают создание проектов на Python.

Для написания кода программ на языке Python будем использовать встроенную среду разработки Thonny Python IDE – это продвинутая Python-IDE, которая хорошо подходит для новичков. Хотя пользоваться ей могут вполне и профессионалы, некоторые черты этой IDE говорят о том, что она особенно хороша для начинающих питонистов. Она даёт в распоряжение программиста возможности по пошаговому выполнению выражений, средства визуализации стека вызовов и множество других полезных мелочей.



Если новичок возьмёт всё это на вооружение – он не только улучшит свои навыки Python-программирования, но и будет лучше понимать то, что происходит во время выполнения кода.

Для начала работы с Raspberry Pi достаточно подключить монитор, клавиатуру и мышь. Но такой режим можно использовать только в стационарном режиме при изучении операционной системы и написании программного кода. В случае, когда Raspberry Pi установлена на мобильную платформу, и платформа при этом движется, такой вариант подключения невозможен.

ВНИМАНИЕ! При работе в стационарном режиме необходимо установить платформу на подставку, обеспечив свободное вращение колес.

Поэтому рассмотрим возможность удаленного беспроводного подключения к рабочему столу Raspberry Pi посредством сети Wi-Fi. Условием работы является подключение Raspberry Pi и вашего компьютера к одной Wi-Fi сети. Для того воспользуемся приложением VNC viewer.

VNC – это широко распространенный метод удаленного доступа к рабочему столу компьютера по сети. Данные о нажатии клавиш и движении мыши, выполняемых пользователем на собственном компьютере, передаются по сети на удаленный компьютер и воспринимаются им как действия с его собственными клавиатурой и мышью. Информация с экрана удаленного компьютера выводится на экране компьютера пользователя.

Работа по VNC через интернет с удаленным компьютером, находящимся в противоположной точке мира, для пользователя выглядит так, как будто этот компьютер находится непосредственно перед ним. Особенно VNC удобен при работе с графическим интерфейсом – с рабочим столом и программами для рабочего стола операционных систем Windows, Linux и других.

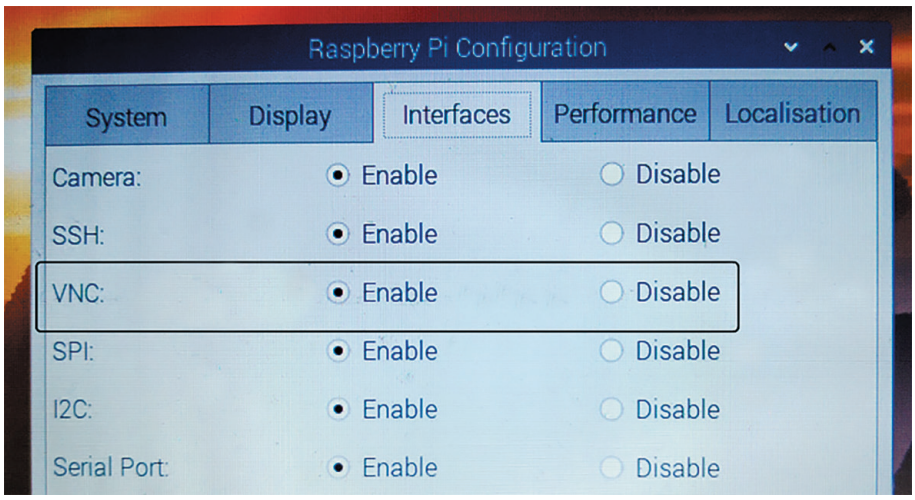
Virtual Network Computing (VNC) – система удалённого доступа к рабочему столу компьютера, использующая протокол RFB (англ. Remote FrameBuffer – удалённый кадровый буфер). Управление осуществляется путём передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть.

VNC платформонезависимая и состоит из двух частей: серверная и клиентская. VNC-клиент, называемый VNC viewer, запущенный на одной операционной системе, может подключаться к VNC-серверу, работающему на любой другой ОС. Реализации клиентской и серверной части на сегодняшний момент существуют практически для всех операционных систем. К одному VNC-серверу одновременно могут подключаться множество клиентов.

При подключении VNC-клиента достаточно указать DNS-имя или IP-адрес удаленного компьютера, и пароль, если доступ к VNC-серверу защищен паролем.

Основной объем трафика по VNC – это передача графической информации, выводимой на экран. Характеристика пропускной возможности канала для работы от 32 Кбит/сек до 2 Мбит/сек. Комфортная работа в полноцветном режиме при разрешении экрана 1024x768 будет при скорости 1-2 Мбит/сек. Канал передачи максимально нагружен только при обновлении больших участков экрана, при печати текста трафик заметно меньше. При больших задержках передачи пакетов – ухудшение времени реакции на нажатие клавиш и движение мыши.

Для запуска серверной части на Raspberry Pi достаточно в разделе Raspberry Pi Configuration включить необходимый параметр.



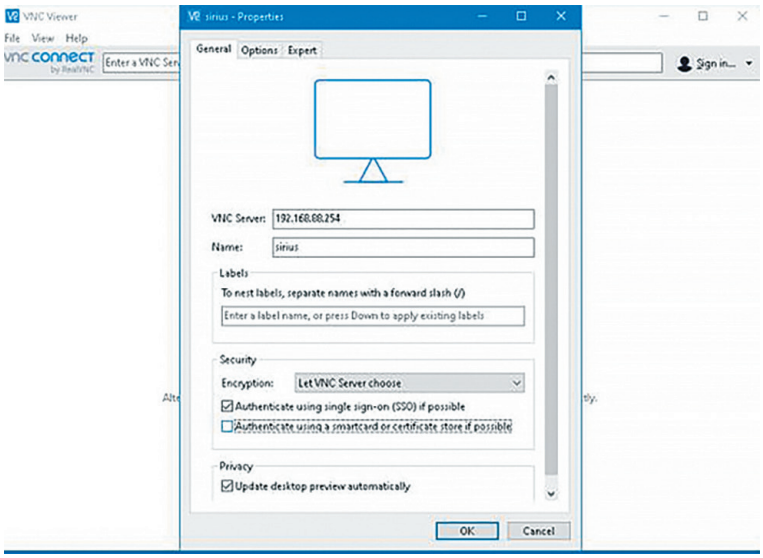
В результате после перезагрузки в верхней части экрана увидим значок.

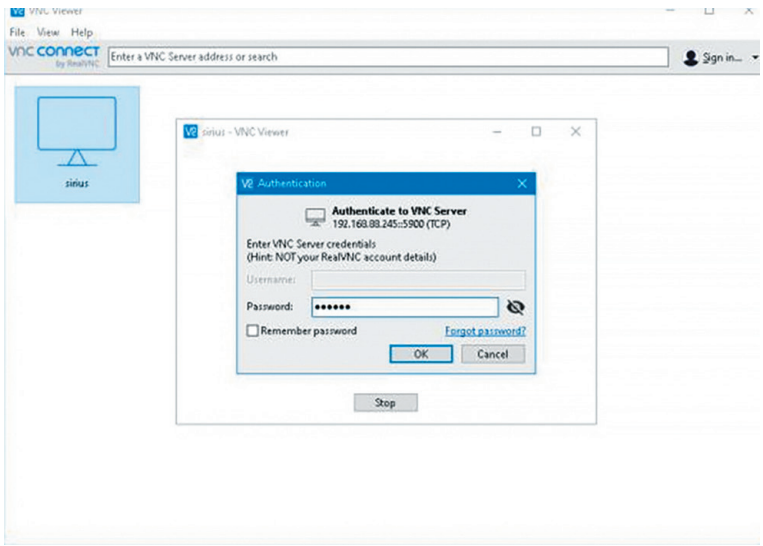
Для установки клиента RealVNC client скачайте дистрибутив из интернета, например, отсюда:

https://apps.microsoft.com/detail/xp99dvcpgktxnj?hl=ru-ru&gl=US

Установите на свой компьютер. Запустите программу. Настройте соединение с сервером – необходимо указать его IP-адрес.

После установления соединения система предложит ввести имя пользователя и пароль.



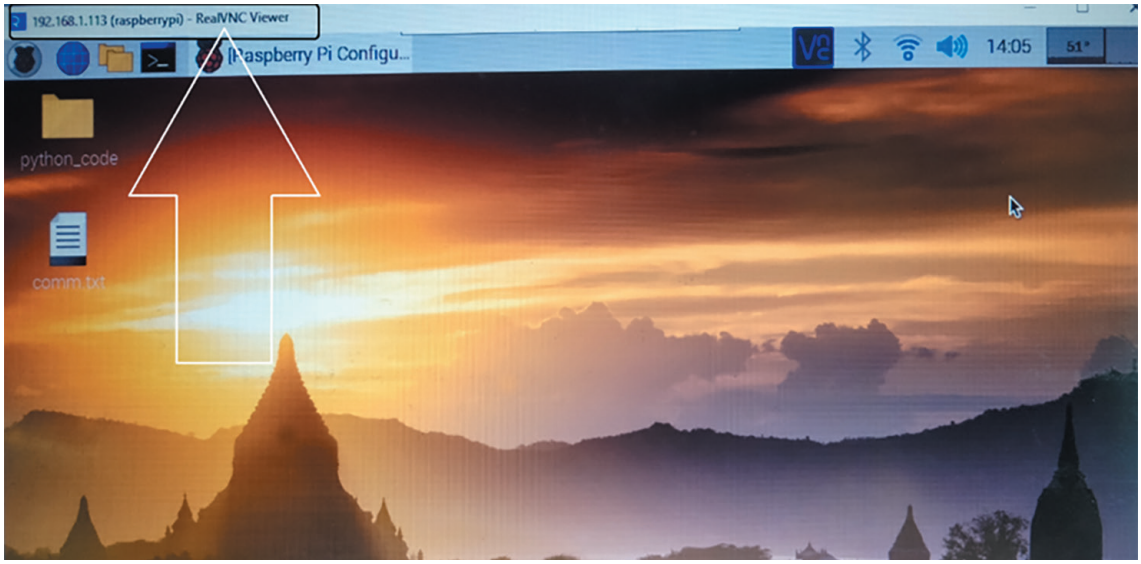


Для работы с Raspberry Pi, используя VNC-viewer, необходимо предварительно обеспечить подключение микрокомпьютера Raspberry Pi к вашей Wi-Fi сети. После подключения к вашей сети запишите полученный IP-адрес Raspberry Pi. Узнать полученный IP-адрес можно так:

откройте терминал и введите команду #ip address

Теперь у вас есть установленный и настроенный сервер VNC на вашем Raspberry Pi и VNC-клиент на компьютере. Вы можете управлять своими файлами, программным обеспечением и настройками с помощью простого и знакомого графического интерфейса пользователя.

На поставляемой с мобильной платформой операционной системе VNC уже установлен. Остается только подключить Optima-Drive к вашей сети Wi-Fi.



Также в образе операционной системы уже установлен Arduino IDE.

Процедура установки была такой:

mkdir ArduinoPgm

mkdir Arduino

cd ArduinoPgm

wget https://downloads.arduino.cc/arduino-1.8.19-linuxarm.tar.xz

tar -xvf arduino-1.8.19-linuxarm.tar.xz

cd arduino-1.8.19

sudo ./install.sh

- /home/(user name)/ArduinoPgm is where the IDE programs are stored. This cannot be «Arduino» or there will be problems starting the application.
- /home/(user name)/Arduino is where the sketches are stored by the IDE.

Рассмотрим более подробно архитектуру системы типа Raspberry Pi + Arduino

Raspberry Pi – это полноценный микрокомпьютер, который может выполнять сложные вычисления, обрабатывать данные и управлять сетевыми соединениями. Arduino Mega 2560, в свою очередь, является микроконтроллером, который отлично подходит для работы с реальными устройствами благодаря своим GPIO-портам и поддержке различных протоколов связи.

В данной системе Raspberry Pi будет выполнять роль «мозга», обрабатывая данные и принимая решения, тогда как Arduino будет отвечать за управление физическими компонентами, такими как моторы, датчики и другие исполнительные устройства.

Для подключения Raspberry Pi к Arduino Mega 2560 используется порт USB. Arduino будет распознаваться Raspberry Pi как последовательное устройство (ttyUSB1), что позволяет осуществлять обмен данными через последовательный порт.

Взаимодействие между Raspberry Pi и Arduino Mega 2560 по USB предоставляет мощные возможности для создания сложных систем управления. Используя простые методы передачи данных через последовательный порт, можно реализовать множество интересных задач.

Вы можете отправлять команды от Raspberry Pi к Arduino для управления моторами или другими устройствами, Arduino может собирать данные с датчиков (например, расстояние) и отправлять их обратно на Raspberry Pi для дальнейшей обработки. Оба устройства могут обмениваться сигналами для синхронизации своих действий.

Это сочетание позволяет использовать сильные стороны каждого устройства – вычислительную мощность Raspberry Pi и возможности управления реальными устройствами от Arduino – для достижения поставленных целей.

Для тестирования возможностей взаимодействия Raspberry Pi и Arduino по интерфейсу USB выполним лабораторную работу – Raspberry Pi отправляет сообщение Arduino, Arduino принимает сообщение и отправляет его назад.

ЛАБОРАТОРНАЯ РАБОТА

№ 4 ВЗАИМОДЕЙСТВИЕ ARDUINO С RASPBERRY PI

Цель работы: Научиться организовывать связь между контроллером Arduino и микрокомпьютером Raspberry Pi.

Задачи:

Со стороны Raspberry Pi:

- открыть последовательный порт
- отправлять сообщение Hello from Raspberry Pi! на Arduino каждую секунду
- выводить ответ от Arduino на экран

Со стороны Arduino:

- открыть и слушать последовательный порт
- принимать сообщения от Raspberry Pi
- отправлять обратно принятые сообщения.

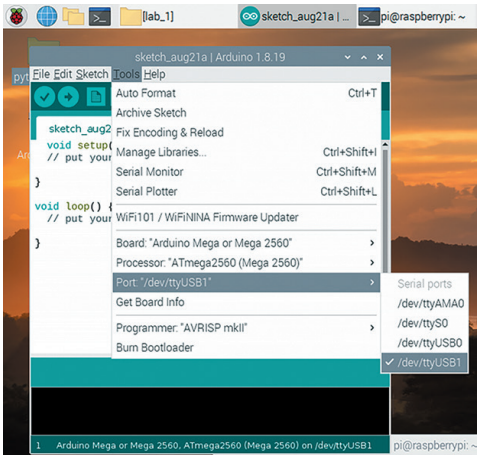
Порядок выполнения работы

1. Для начала работы необходимо подключить Arduino к Raspberry Pi с помощью заглушки.

2. Подключиться к Raspberry Pi – локально, подключив монитор, клавиатуру и мышь (или посредством VNC после загрузки системы).

3. Включить питание Optima-Drive и дождаться загрузки операционной системы.

4. Проверьте подключение: на Raspberry Pi выполните команду ls /dev/tty*, чтобы найти имя устройства (обычно это будет что-то вроде /dev/ttyUSB0 или /dev/ttyACM0).



5. Запустить Arduino IDE на Raspberry Pi.

6. Загрузить программу (скетч) в Arduino, которая будет обрабатывать входящие данные от Raspberry Pi и отправлять ответы обратно из папки python_code\Скетчи\номер лабораторной работы на рабочем столе Raspberry Pi.

```
#include <Arduino.h>

void setup() {
    Serial.begin(9600); // Инициализация последовательного порта
}

void loop() {
    if (Serial.available() > 0) { // Проверка наличия данных
        String input = Serial.readStringUntil('\n'); // Чтение строки до символа
новой строки
        Serial.print("Received: "); // Отправка ответа обратно
        Serial.println(input);
    }
}
```

В этом коде мы настраиваем последовательный порт на скорость 9600 бод и ожидаем входящие данные от Raspberry Pi. Когда данные поступают, они считываются и отправляются обратно с префиксом Received.

7. На стороне Raspberry Pi мы можем использовать Python для взаимодействия с Arduino через последовательный порт.

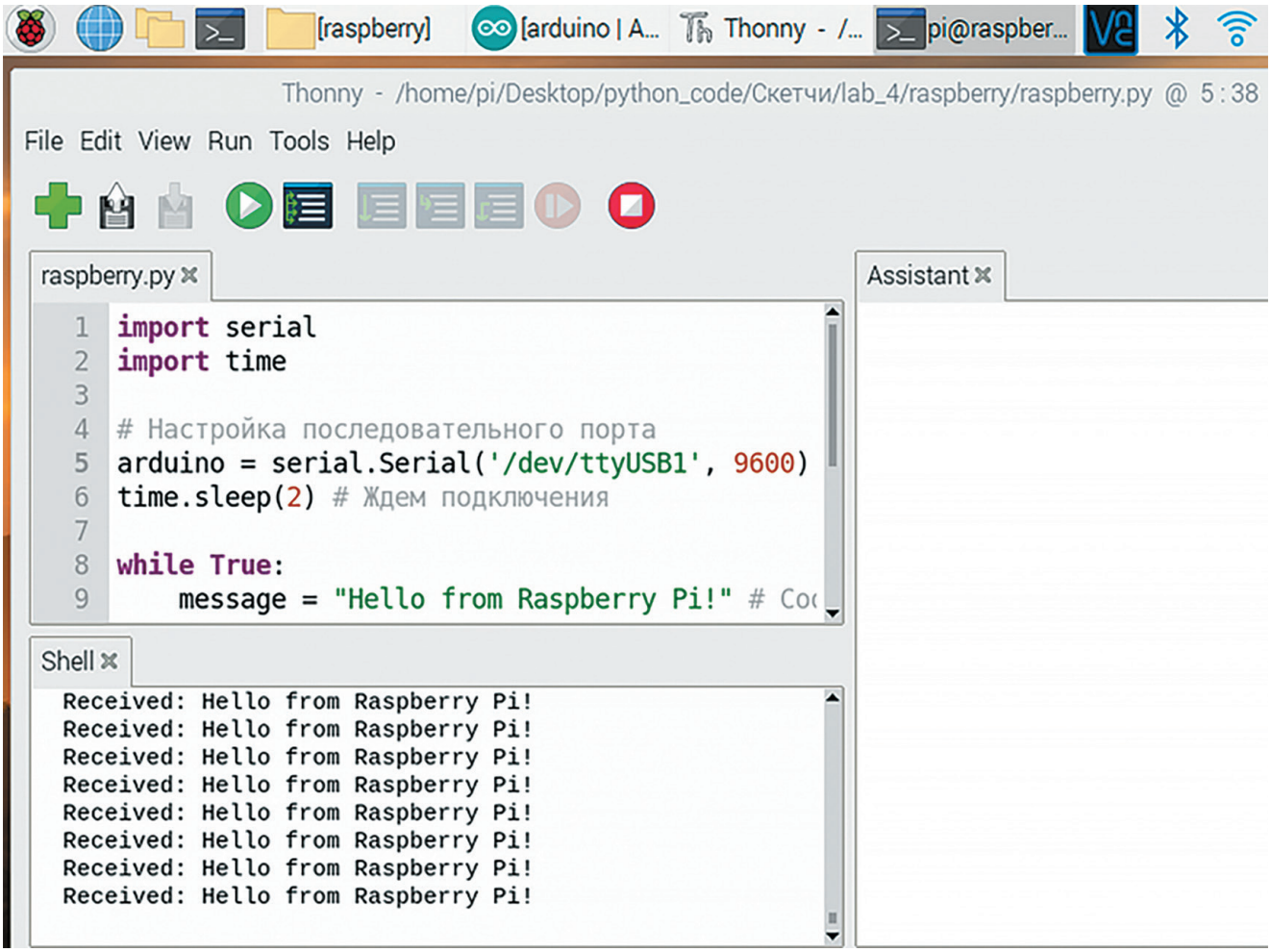
- Запустите Thonny Python IDE. Для этого слева вверху нажимаем иконку «Пуск» (символ малинки)
- Выбираем пункт Programming
- Выбираем Thonny Python IDE
- Нажмите сивол «+» (NEW)
- Скопируйте туда код, представленный ниже (текст программы находится в папке python_code\Скетчи\номер лабораторной работы)
- Исполнение программы запускается с помощью кнопки Run
- На экране монитора (внизу) можно видеть отображение выполнения

```
import serial
import time

# Настройка последовательного порта
arduino = serial.Serial('/dev/ttyUSB1', 9600) # Убедитесь, что используете правильное имя порта
time.sleep(2) # Ждем подключения

while True:
    message = «Hello from Raspberry Pi!» # Сообщение для отправки
    arduino.write((message + '\n').encode()) # Отправка сообщения на Arduino

    time.sleep(1) # Задержка перед отправкой следующего сообщения
```

```
if arduino.in_waiting > 0: # Проверяем наличие данных от Arduino
    response = arduino.readline().decode().strip() # Чтение ответа от Arduino
    print(response) # Вывод ответа на экран.
```

8. Запустите программу.

9. Убедитесь в правильности выполнения программы.



ХОДОВАЯ ЧАСТЬ МОБИЛЬНОЙ ПЛАТФОРМЫ. УПРАВЛЕНИЕ ДВИЖЕНИЕМ

Современные мобильные платформы находят широкое применение в различных областях, включая робототехнику, автоматизацию и логистику. Одним из интересных решений является использование всенаправленных колёс, известных как «колёса Илона», которые обеспечивают высокую маневренность и гибкость движений. Именно такие колёса использует мобильная платформа Optima-Drive для перемещения.

Мобильная платформа на базе колёс Илона представляет собой систему, состоящую из четырех или более всенаправленных колёс, каждое из которых оснащено собственным двигателем. Это позволяет платформе перемещаться в любом направлении без необходимости разворота.

Основные компоненты ходовой части платформы включают:

Колёса Илона (Mecanum) – роликонесущее колесо, позволяющее транспорту двигаться в любом направлении. Своё название оно получило от шведского изобретателя Бенгта Илона, разработавшего его. Идея создать такое колесо пришла к нему в 1973 году, когда он работал инженером в шведской компании Mecanum AB. Эти колёса имеют уникальную конструкцию с роликами, расположенными под углом к оси вращения. Путём изменения направления и скорости вращения отдельных колёс можно заставить машину на илоновых колёсах двигаться в любом направлении – перпендикулярно движению вперёд-назад, по диагонали под любым углом в зависимости от скорости каждого отдельного мотора. При этом трения скольжения между роликами и опорной поверхностью практически не будет. Конструкция колёс Илона позволяет вращаться на месте при минимальной силе трения и низком вращательном моменте.

Использование колёс Илона и индивидуальных двигателей на каждом колесе предоставляет множество преимуществ:

• Высокая маневренность.

Одним из главных преимуществ данной конструкции является высокая маневренность. Платформа может перемещаться в любом направлении без необходимости разворота или изменения ориентации. Это особенно полезно в ограниченных пространствах или при выполнении сложных маневров.

• Простота управления.

Система управления на базе драйверов L298N позволяет легко интегрировать платформу с различными контроллерами (например, Arduino или Raspberry Pi). Управление движением может быть реализовано через простые команды, что упрощает разработку программного обеспечения для управления.

• Гибкость конфигурации.

Платформа может быть сконфигурирована для выполнения различных задач в зависимости от требований проекта. Например, можно изменить количество колёс или их расположение для достижения оптимальной производительности в конкретной среде.

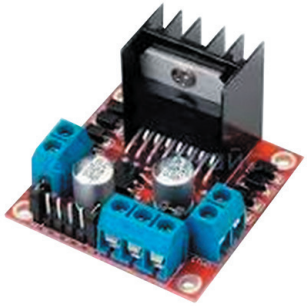
• Возможность реализации сложных алгоритмов движения.

Благодаря независимому управлению каждым колесом можно реализовать сложные алгоритмы движения, такие как следование за объектом, избегание препятствий и другие интеллектуальные функции.

• Эффективное использование пространства.

Подобные колёса позволяют платформе эффективно использовать доступное пространство при перемещении.

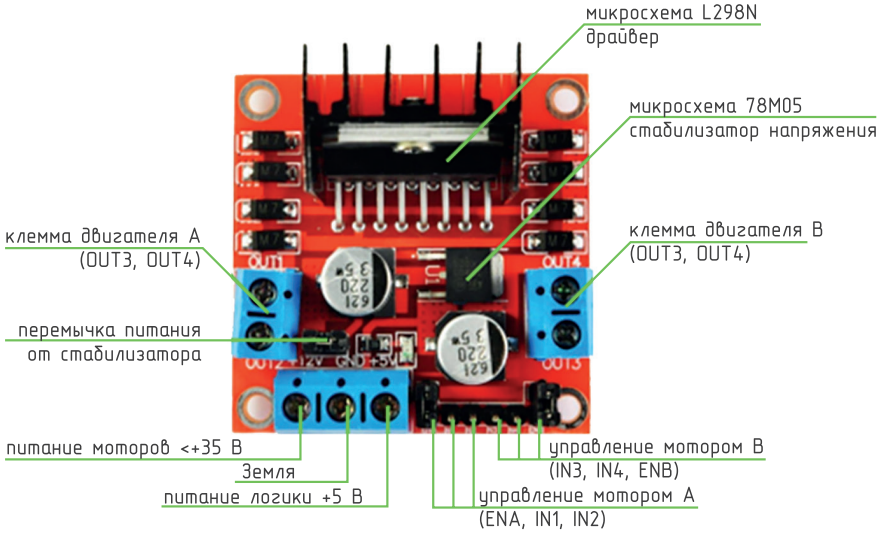
Двигатели: На каждом колесе установлен отдельный электродвигатель постоянного тока с редуктором JGB37-50 (200 RPM 12V), что обеспечивает независимое управление каждым колесом. Это позволяет реализовать сложные траектории движения и маневры.



Драйверы L298N: Для управления двигателями используются драйверы L298N, которые позволяют управлять направлением и скоростью вращения каждого двигателя. Motor Shield разработан на базе микросхемы L298N. Позволяет управлять двумя моторами постоянного тока либо одним шаговым двигателем. Максимальный потребляемый ток не должен превышать 2 Ампера. Как и в случае L293N, драйвер представляет собой полный H-мост, главная функция которого – менять полярность на нагрузке. А если в качестве нагрузки будет мотор постоянного тока, то смена полярности приведет к смене направления его вращения.

Спецификация модуля L298N:

• Напряжение питания двигателей: до 35 В;
• Рабочий ток (на каждый канал): 2 А;
• Периодический ток (80% – вкл, 20% – выкл): 2,5 А;
• Кратковременный ток: 3 А;
• Вес: 33 г.



Логика микросхемы L298N питается напряжением 5 Вольт. Для этого на модуле предусмотрен стабилизатор напряжения 78M05. На вход этого стабилизатора можно подавать напряжение до 35 В, а на выходе всегда получается 5 В. Рабочий ток у 78M05 небольшой – до 500 мА.

Тройная клемма снизу отвечает за питание модуля. Самый левый контакт – питание моторов. Сюда можно подавать до 35 В. Средний контакт – земля, которая должна быть общей для модуля и контроллера. Правый контакт имеет двоякую функцию. Если на модуле стоит перемычка питания стабилизатора, то на этом контакте будет +5 В и к нему можно ничего не подключать, либо питать от него контроллер. Но если перемычку убрать, то к этому контакту нужно будет непременно подключить +5 В от контроллера, чтобы питать драйвер.

Две другие винтовые клеммы (OUT1/2 и OUT 3/4) служат для подключения моторов. Надо отметить, что моторы постоянного тока неполярные, но от того, на какой контакт мотора подается плюс, а на какой минус, зависит направление их вращения.

Наконец осталось разобраться с контактами управления. Их по три штуки на каждый мотор. Контакты ENA и ENB позволяют управлять моторами с помощью ШИМ-сигнала. Если ENA и ENB подключить строго к +5 В, то моторы будут всегда вращаться с максимальной возможной скоростью. Именно для этого режима на модуле предусмотрены две перемычки рядом с ENA и ENB.

С помощью контактов IN1, IN2, IN3, IN4 задаётся режим работы моторов. Таблица режимов для двигателя А имеет вид:

Режим	**IN1**	**IN2**
Вращение в одну сторону	1	0
Вращение в обратную сторону	0	1
Блокировка мотора	1	1
Отключение мотора	0	0

Тут следует пояснить последние два режима. Если нам необходимо резко остановить мотор, то выбираем режим блокировки. Для плавной остановки – выбираем «отключение мотора».

Обратите внимание – из-за наличия инертности тока в моторе (мотор по сути катушка индуктивности) и при резком изменении полярности подключенного тока мы заставляем ток, отдаваемый мотором/катушкой работать против ЭДС/питания. Из-за этого очень сильно просаживается питание и может привести к нестабильности/перезагрузке контроллера.

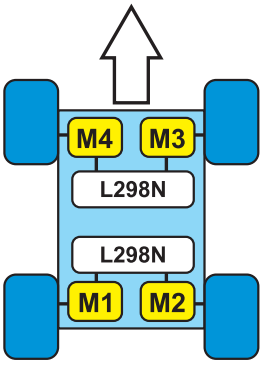
Чтобы этого избежать, можно прижимать пины к одному напряжению на несколько миллисекунд и только потом выполнять реверс.

В мобильной платформе Optima-Drive каждый из четырех моторов управляется с помощью трех пинов контроллера. Два цифровых пина (IN1 и IN2 для одного мотора и IN3 и IN4 для второго) отвечают за направление движения, а с помощью подачи ШИМ-сигнала (что такое ШИМ, мы поговорим далее) на третий пин (EN1 для первого мотора и EN2 для второго) задаем скорость вращения двигателей. Для второй пары моторов аналогично.

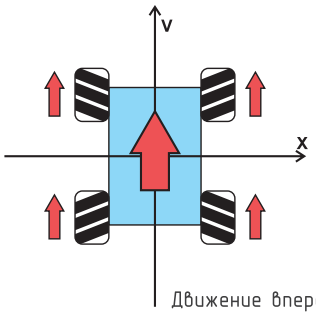
Для управления двигателями мобильного робота Optima-Drive используются следующие цифровые пины контроллера Arduino Mega 2560:

Motor 1: IN1 = 3 IN2 = 4 ENA = 2
Motor 2: IN3 = 5 IN4 = 6 ENB = 7
Motor 3: IN1_1=9 IN1_2 = 10 ENA_1 = 8
Motor 4: IN1_3 = 11 IN1_4 = 12 ENB_1 = 13

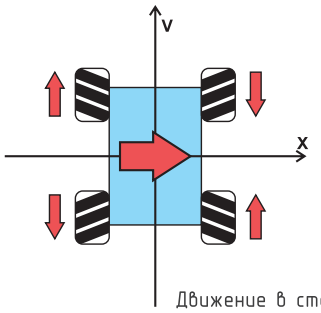
На рисунке ниже показано расположение и нумерация моторов мобильной платформы Optima-Drive.



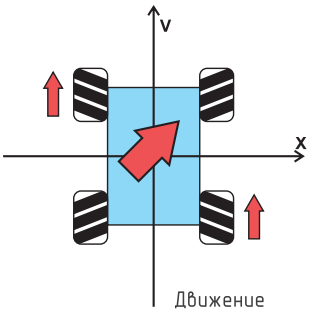
Как уже было сказано ранее, в зависимости от направления вращения каждого из четырех колёс можно получить различные варианты изменения направления движения платформы:



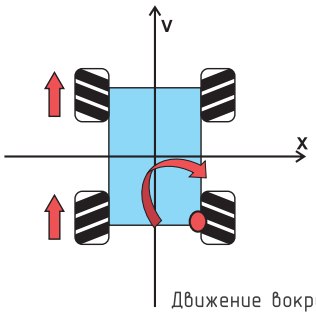
Движение вперед



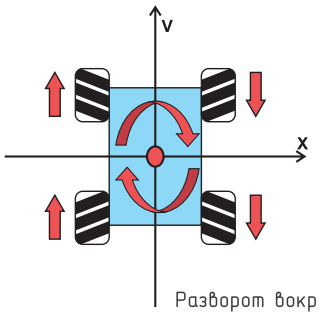
Движение в сторону



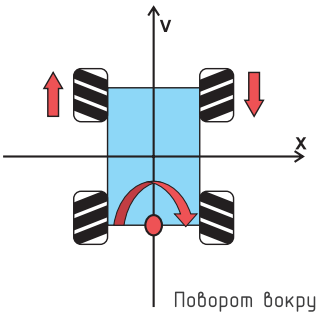
Движение по диагонали



Движение вокруг одного колеса



Разворот вокруг своей оси



Поворот вокруг задней оси

ЛАБОРАТОРНАЯ РАБОТА

№ 5 УПРАВЛЕНИЕ НАПРАВЛЕНИЕМ ДВИЖЕНИЯ ПЛАТФОРМЫ

Цель работы: Научиться с помощью программного кода управлять параметрами движения мобильной платформы.

Задачи: Написать код программы (скетч) для Arduino, обеспечив движение платформы вперед 2 секунды остановка на 2 секунды поворот направо и вновь движение – 2 секунды.

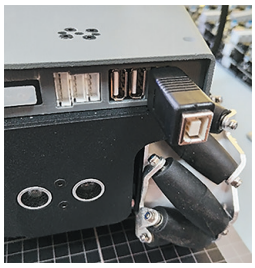
Оформить код направления движения в виде функций forward (), stop (), right ().

Порядок выполнения работы

Перед началом работы с ходовой частью платформы рекомендуется установить платформу на постамент, обеспечив свободное вращение колёс, для отладки программ.

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino, подключить переходник USB Type A – Type B и подключиться к контроллеру с помощью кабеля USB type A-type B.



Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Загрузить код программы-примера или написанной самостоятельно в Arduino.

3. Включить питание платформы.

4. Убедиться в правильности выполнения кода программы.

```
// Определение пинов для каждого мотора
int IN1 = 3; int IN2 = 4; int ENA = 2;          // Мотор 1
int IN3 = 5; int IN4 = 6; int ENB = 7;          // Мотор 2
int IN1_1 = 9; int IN1_2 = 10; int ENA_1 = 8;   // Мотор 3
int IN1_3 = 11; int IN1_4 = 12; int ENB_1 = 13; // Мотор 4

void setup() {
  // Последовательный порт для вывода в Монитор
  Serial.begin(9600);

  // Устанавливаем все пины как выходы
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1_1, OUTPUT); pinMode(IN1_2, OUTPUT); pinMode(ENA_1, OUTPUT);
  pinMode(IN1_3, OUTPUT); pinMode(IN1_4, OUTPUT); pinMode(ENB_1, OUTPUT);
}
```

```
void loop() {
  delay(2000);
  // Движение вперед
  forward();
  delay(2000);

  // Остановка
  stop();
  delay(2000);

  // Поворот направо
  right();
  delay(2000);

  // Остановка
  stop();
delay(2000);

}

void forward () {
  Serial.println("Forward"); // Вывод в монитор

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA, 200);

  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB, 200);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1, 200);

  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1, 200);
}

void right() {
  Serial.println("Right");  // Вывод в монитор
```

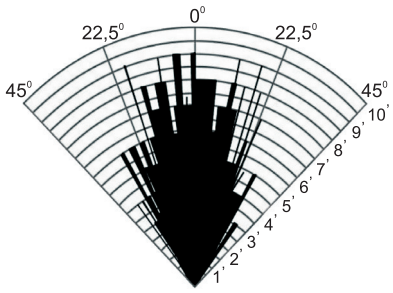
```
  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA, 255);

  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB, 255);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, LOW);
  digitalWrite (IN1_2, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1, 255);

  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1, 255);
}

void stop() {
  Serial.println("Stop");  // Вывод в монитор

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, LOW);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA, 0);

  digitalWrite (IN3, LOW);
  digitalWrite (IN4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB, 0);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, LOW);
  digitalWrite (IN1_2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1, 0);

  digitalWrite (IN1_3, LOW);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1, 0);
}
```

СИСТЕМА БЕЗОПАСНОСТИ ДВИЖЕНИЯ МОБИЛЬНОЙ ПЛАТФОРМЫ НА БАЗЕ УЛЬТРАЗВУКОВЫХ СЕНСОРОВ HC-SR04

Системы безопасности являются неотъемлемой частью современных мобильных платформ, особенно в условиях, где требуется высокая степень маневренности и автономности. Система безопасности движения мобильной платформы Optima-Drive основана на использовании восьми ультразвуковых сенсоров HC-SR04. Эти сенсоры обеспечивают надежное обнаружение препятствий и предотвращение столкновений, что особенно важно при движении назад.

Характеристики ультразвукового датчика расстояния HC-SR04:

- Питание: 5V;
- Рабочий ток: 15 мА;
- Звуковая частота: 40 кГц;
- Угол измерения: 15 градусов;
- Диапазон измерения: 2 см - 4 м;
- Точность: ~1 мм при грамотной фильтрации.



Practical test of performance, Best in 30 degree angle

Ультразвуковые сенсоры HC-SR04 работают по принципу эхолокации – они излучают ультразвуковые волны и измеряют время, необходимое для их отражения от препятствия и возвращения обратно к сенсору.

Принцип метода измерения расстояния заключается в следующем: один из пьезоэлементов (передатчик) излучает ультразвуковой сигнал, а другой пьезоэлемент (приёмник) принимает этот же отражённый сигнал от препятствия. Затем замеряется время, которое прошло с момента отправки до момента приёма отраженного сигнала. Следующий импульс может быть передан, когда эхо исчезло. Этот период времени называется период цикла. Рекомендованный период цикла должен быть не менее 50 мс. Импульс длительностью 10 мкс дает команду на отправку 8 импульсов 40 КГц ультразвукового сигнала и детектирование эхо-сигнала обратно. Измеренное расстояние пропорционально длительности импульса на выходе Echo и может быть рассчитано по формуле.

Расстояние до объекта рассчитывается по следующей формуле:

$$\text{Distance} = \frac{\text{Time} \times \text{Speed of Sound}}{2}$$

Distance – расстояние до объекта (в сантиметрах или метрах).

Time – время, за которое ультразвуковой сигнал прошел до объекта и обратно (в микросекундах).

Speed of Sound – скорость звука в воздухе (примерно 343 метра в секунду или 0,0343 сантиметра в микросекунду при температуре 20 °C).

Поскольку сигнал проходит туда и обратно, мы делим полученное значение времени на 2, чтобы получить расстояние только до объекта.

Таким образом, если вы хотите получить расстояние в сантиметрах, формула будет выглядеть так:

$$\text{Distance (cm)} = \frac{\text{Time (µs)} \times 0.0343}{2}$$

Эти формулы позволяют датчику HC-SR04 точно определять расстояние до ближайшего объекта.

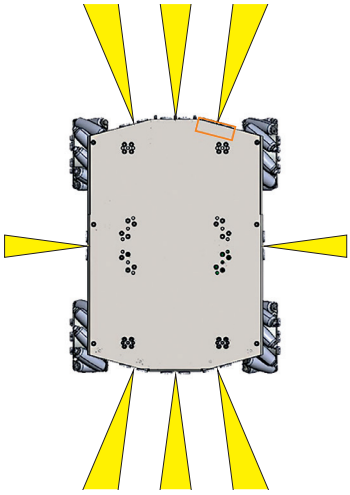
Если никаких препятствий не обнаруживается, то выходной контакт даст сигнал на 38ms высокого уровня.

Дополнительно можно ознакомиться с датчиками по ссылкам:

https://alexgyver.ru/lessons/filters/

https://kit.alexgyver.ru/tutorials/hc-sr04/

Для обеспечения максимальной безопасности на мобильной платформе Optima-Drive устанавливаются восемь ультразвуковых сенсоров HC-SR04:



• Три сенсора спереди: эти сенсоры обеспечивают обнаружение препятствий перед платформой, что позволяет предотвратить столкновения с объектами на пути движения.

• По одному сенсору по бокам: боковые сенсоры помогают контролировать пространство слева и справа от платформы, что особенно важно при маневрировании в узких проходах или при поворотах.

• Три сенсора сзади: эти сенсоры играют ключевую роль при движении назад, обеспечивая обнаружение объектов позади платформы и предотвращая возможные столкновения.

Такое распределение сенсоров позволяет создать полное «облако» защиты вокруг мобильной платформы, что значительно повышает уровень безопасности.

Для эффективного использования данных от ультразвуковых сенсоров необходимо разработать алгоритмы обработки информации. Например, при направлении движения вперед – опрашиваются передние 3 сенсора, при движении назад – задние 3. Боковые подключаются при движении вбок и при прохождении узких мест.

Система безопасности на базе ультразвуковых сенсоров HC-SR04 обладает рядом значительных преимуществ:

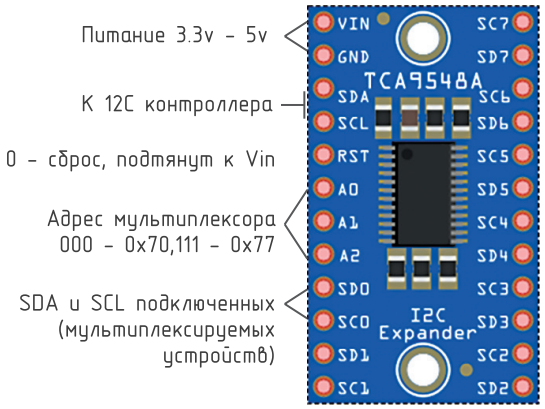
• Высокая точность обнаружения: Ультразвуковые сенсоры обеспечивают высокую точность измерений расстояния до объектов, что позволяет эффективно предотвращать столкновения.

• Широкий угол обзора: Расположение трех спереди, одного по бокам и трех сзади обеспечивает полный контроль над пространством вокруг мобильной платформы.

• Низкая стоимость: Сенсоры HC-SR04 являются доступными по цене, что делает систему экономически выгодной для реализации.

Однако при использовании восьми сенсоров имеются и недостатки – каждый сенсор подключается к двум 2 пинам контроллера. Восемь сенсоров таким образом возьмут на себя 16 пинов, что является не очень удобным. Для решения этой проблемы в Optima-Drive используются сенсоры HC-SR04, подключаемые по интерфейсу I²C. Но в данном случае просто подключить их к одной шине данных не получится – у всех этих сенсоров один и тот же адрес I²C - 0x57. Для устранения этой проблемы в мобильной платформе Optima-Drive используется мультиплексор I²C TCA9248A.

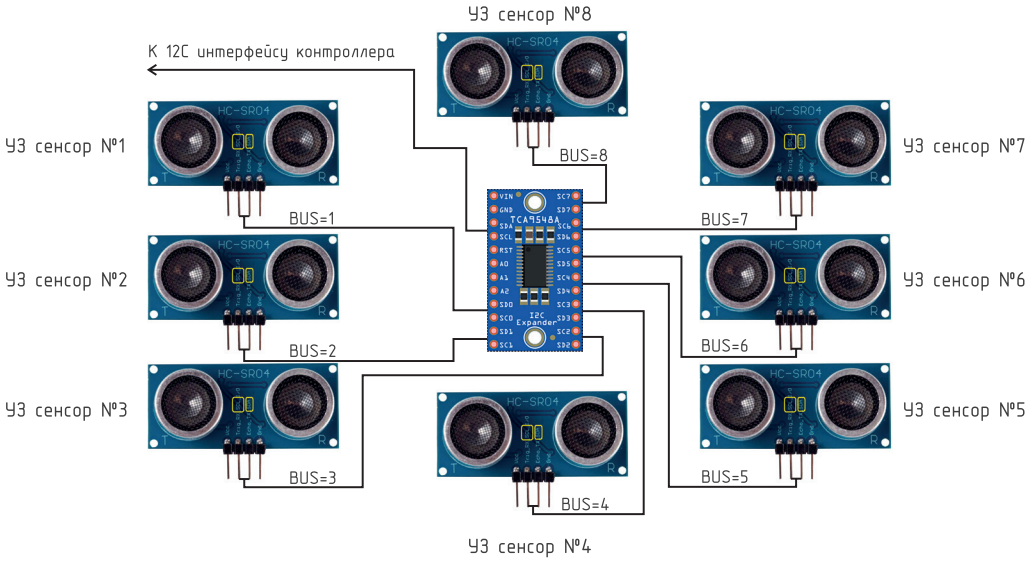
TCA9548A это мультиплексор – переключатель I²C интерфейсов. На выходных линиях могут располагаться устройства с одинаковыми адресами, а их переключение производится управляющими командами по адресу самой микросхемы TCA9548A. Адрес TCA9548A для I²C интерфейса выбирается подачей на адресные контакты A0-A2 определенных уровней: 000 – это шестнадцатеричный адрес 0x70; 111 - 0x77. Если мы используем распаянную на плате микросхему, как показано на рисунке ниже, то адресные контакты уже подтянуты к земле через резисторы и их можно не подключать, адрес устройства при этом будет 0x70. Теперь технически можно подключить до 8 I²C устройств с одинаковыми адресами.



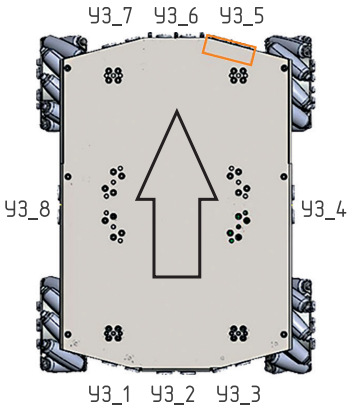
Эти 8 устройств с одинаковыми адресами и будут нашими сенсорами. Обращаясь к мультиплексору по интерфейсу I²C по адресу 0x70, просто нужно указать один из 8 каналов, к которому подключен нужный сенсор.

Например, используя рисунок внизу, можно сказать, что для приема данных с ультразвукового сенсора № 2 нужно обратиться к мультиплексору по адресу 0x70, указав номер шины 2. В лабораторной работе рассмотрим этот вопрос более подробно.

Таким образом, схема подключения сенсоров HC-SR04 к мультиплексору выглядит следующим образом:



Располагаются ультразвуковые датчики на мобильной платформе в следующем порядке:



ЛАБОРАТОРНАЯ РАБОТА

№ 6 РАБОТА СЕНСОРА HC-SR04

Цель работы: Познакомиться с работой УЗ сенсора.

Задачи: Изучить принцип работы и варианты настройки ультразвукового сенсора, подключенного к каналу 2 (УЗ_3) мультиплексора.

Порядок выполнения работы

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

4. Установите в Arduino IDE библиотеку SGBotic_I2CPing.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

```
#include <Wire.h>
#include <SGBotic_I2CPing.h>
SGBotic_I2CPing sonar;

// Инициализация шины
uint8_t bus;

void setup() {
  Serial.begin(9600);
}

void loop() {
  TCA9548A(2);// Подключаем второй канал
  unsigned long d2 = sonar.ping_cm();
  Serial.println(d2);
  delay(500);
}

// Обращение к датчику по его номеру на шине
void TCA9548A(uint8_t bus) {
  Wire.beginTransmission(0x70);
  Wire.write(1 << bus);
  Wire.endTransmission();
}
```

6. Включить питание платформы.

7. Убедиться в правильности выполнения кода программы.

Для улучшения качества полученного сигнала можно использовать различные фильтры. Попробуйте отфильтровать данные при помощи простейшего экспоненциального фильтра.

Более подробно об этом можно посмотреть по ссылке:

https://alexgyver.ru/lessons/filters/

```
#include <SGBotic_I2CPing.h>
#include <Wire.h>
SGBotic_I2CPing sonar;

// Инициализация шины
uint8_t bus;

// Расстояние до объекта после фильтра
float distFilt = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  TCA9548A(2);
  unsigned long dist = sonar.ping_cm();
  distFilt += (dist - distFilt) * 0.2; // Фильтр
  Serial.println(distFilt);
  delay(500);
}

// Обращение к датчику по его номеру на шине
void TCA9548A(uint8_t bus) {
  Wire.beginTransmission(0x70);
  Wire.write(1 << bus);
  Wire.endTransmission();
}
```

В Лабораторной работе № 15 «Пример комплексного использования систем мобильной платформы» представлен код программы, которая реализует более сложный алгоритм работы системы безопасности движения платформы – там задействованы 3 передних сенсора.

ГИРОСКОП

Современные мобильные платформы, такие как робототехнические системы, дроны и автономные транспортные средства, требуют высокоточных сенсоров для навигации и ориентации в пространстве. Одним из ключевых компонентов таких систем является инерциальный измерительный блок (IMU).

Например, такой модуль, как iArduino IMU 9 DOF, сочетает в себе акселерометр, гироскоп и магнитометр, что позволяет ему эффективно отслеживать движение и ориентацию устройства. Мобильная платформа Optima-Drive использует именно такой модуль – IMU-9 DOF компании iArduino – для позиционирования себя в пространстве. Модуль IMU-9 DOF подключен к контроллеру Arduino Mega 2560 по интерфейсу I²C.

Trema-модуль IMU 9 DOF (Inertial Measurement Unit 9 Degrees Of Freedom) – инерционное измерительное устройство на 9 степеней свободы, построен на базе чипа BMX055 (Bosch Module X) – модуль фирмы Bosch, где X означает, что в чип интегрировано несколько датчиков: (accelerometer) акселерометр, (gyroscope) гироскоп, (magnetometer) магнитометр. Все 3 датчика чипа BMX055 выдают показания по 3 осям (X, Y, Z), следовательно, с чипа считываются показания для 9 осей (9 DOF).

Специально для Trema-модуля IMU 9 DOF разработана **библиотека iarduino_Position_BMX055**, которая значительно упрощает процесс получения данных с модуля.

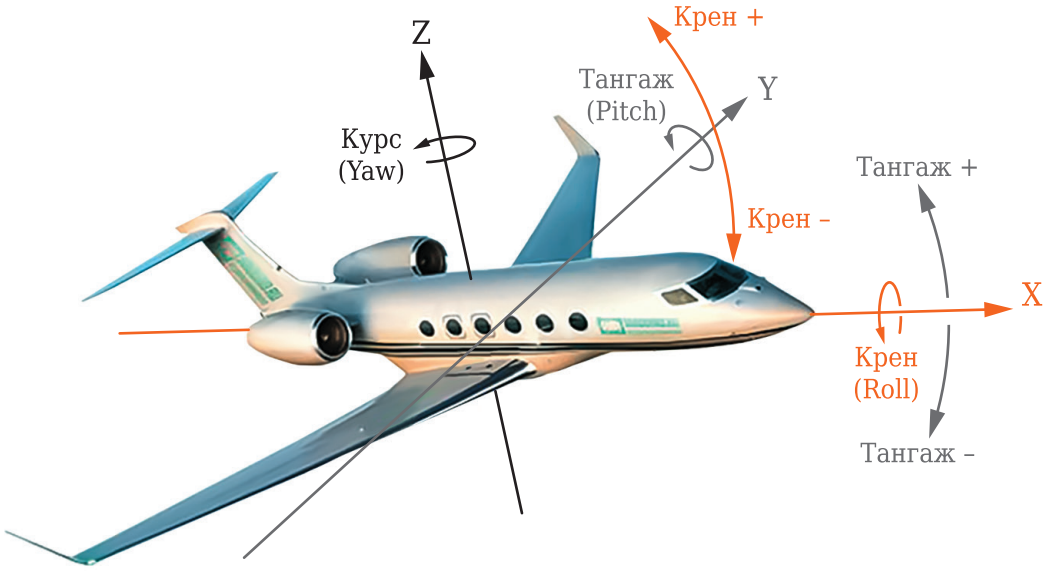
Библиотека способна работать как со всеми датчиками сразу, так и по отдельности. В библиотеке имеются функции аппаратного самотестирования и калибровки датчиков, есть возможность выбора диапазонов измерений, частоты обновлений и единиц измерений выводимых данных. В библиотеку интегрированы фильтры Маджвика (по умолчанию) и Махони (можно выбрать).

Дополнительная информация по работе с модулем

Акселерометр – датчик, измеряющий кажущееся угловое ускорение, которое является геометрической разницей между истинным угловым ускорением и ускорением силы гравитации. Показания датчика можно получать в м/с², или в g (количестве ускорений свободного падения). Предположим, что датчик неподвижен, или движется равномерно, ось Z направлена вверх (детали на плате модуля смотрят вверх). В таком случае проекция вектора силы гравитации не оказывает влияния на оси XY, их показания равны 0 м/с², но оказывает влияние на ось Z и направлена в противоположную сторону (к земле), значит Z = 0 - -g = 9,81 м/с², где 0 – проекция истинного ускорения модуля на ось Z, g – ускорение свободного падения, взятое со знаком минус, так как его вектор противоположен направлению оси Z. Если модуль наклонить, то влияние g на ось Z ослабнет, но увеличится на те оси, в направлении которых был наклонён модуль. Таким образом, зная проекцию вектора ускорения свободного падения на оси X, Y, Z, можно вычислить положение датчика относительно поверхности Земли (углы «крен» и «тангаж»), но только если датчик неподвижен или движется равномерно!

Гироскоп – датчик, измеряющий угловую скорость вокруг собственных осей. Показания датчика можно получать в °/с, или рад/с. Данный датчик способен определять воздействие момента внешней силы вокруг своих осей. Используя эти данные, можно компенсировать воздействие истинного ускорения на акселерометр, следовательно, используя акселерометр и гироскоп, получать положение этих датчиков относительно поверхности Земли (углы: «крен» и «тангаж») во время их неравномерного движения.

Магнитометр – датчик, измеряющий индукцию магнитного поля. Показания датчика можно получать в мГс или в мкТл. Данный датчик способен определять своё положение в пространстве относительно магнитных полюсов Земли. Добавление этого датчика к двум предыдущим даёт возможность получить последний угол Эйлера - «курс», а также повлиять на точность определения предыдущих двух углов «крен» и «тангаж».



Показания всех датчиков можно использовать как входные данные для фильтра Маджвика, Махони, Калмана или др., для получения кватернионов абсолютной ориентации устройства, из которых рассчитываются углы Эйлера («крен», «курс» и «тангаж»). Некоторые фильтры позволяют получать кватернионы, используя данные только первых двух датчиков (без магнитометра), из которых также можно рассчитать углы Эйлера, но угол «курс» будет не истинным, а рассчитанным, он будет указывать не на север, а на изначальное направление датчика.

Углы Эйлера позволяют определить положение объекта в трёхмерном (евклидовом) пространстве. Для определения положения используются 3 угла «крен», «тангаж» и «курс».

• **«Крен» (Roll)** – определяет наклон тела вокруг продольной оси X, например, крен самолёта показывает, на сколько градусов в бок наклонились его крылья относительно земной поверхности. Если самолёт находится параллельно земле, то крен = 0°. Если самолёт накренился (наклонился) вправо (левое крыло выше правого), то крен положительный (от 0° до 90°). Если самолёт накренился (наклонился) влево (левое крыло ниже правого), то крен отрицательный (от 0° до -90°). Если самолёт выполняет «бочку» (перевернулся), то крен ±180°.

В библиотеке iarduino_Position_BMX055 крен привязан к оси Y, а не X, так как под креном легче понимать отклонение оси Y от горизонта Земли (на картинке указано стрелками «Крен+», «Крен-» от оси Y).

• **«Тангаж» (Pitch)** – определяет наклон тела вокруг поперечной оси Y, например, тангаж самолёта показывает, на сколько градусов поднят (или опущен) его нос относительно земной поверхности. Если самолёт находится параллельно Земле, то тангаж = 0°. Если самолёт поднял нос вверх (кабрирует, взлетает), то тангаж положительный (от 0° до 90°). Если самолёт опускает нос (пикирует, приземляется), то тангаж отрицательный (от 0° до -90°). Если самолёт выполняет «мертвую петлю», то тангаж доходит до ±180° (полёт назад вверх ногами).

В библиотеке iarduino_Position_BMX055 тангаж привязан к оси X а не Y, так как под углом тангаж легче понимать отклонение оси X от горизонта Земли (на картинке указано стрелками «Тангаж+», «Тангаж-» от оси X).

• **«Курс» (Yaw)** – это самый простой для понимания угол Эйлера (еще его нарывают «рыскание»), он определяет направление вдоль земной поверхности. Например, для самолёта курс определяет, куда самолёт летит. Если самолёт летит на север, то курс = 0°. Если самолёт отклоняется от севера влево (на запад, или против часовой стрелки, если смотреть сверху, то курс отрицательный (от 0°, через -90° восток, до -180° юг). Если самолёт отклоняется от севера вправо (на восток, или по часовой стрелке, если смотреть сверху) то курс положительный (от 0°, через 90° запад, до 180° юг).

Спецификация:

• Чип BMX055
• Питание модуля: 3,3 В или 5 В (оба напряжения входят в диапазон допустимых значений).
• Диапазоны измерений:
 - акселерометра: ±2g, ±4g, ±8g, ±16g, где g - ускорение свободного падения = 9,81 м/с^2.
 - гироскопа: ±125 °/с, ±250 °/с, ±500 °/с, ±1000 °/с, ±2000 °/с.
 - магнитометра: ±1300 мкТл для осей XY, ±2500 мкТл для оси Z.
• Максимальная чувствительность:
 - акселерометра: 9,5 * 10-3 м/с^2.
 - гироскопа: 3,8 * 10-3 °/с.
 - магнитометра: 625 мкГс.
• Частота обновления фильтрованных данных (в Гц):
 - акселерометра: 8, 16, 31, 63, 125, 250, 500, 1000.
 - гироскопа: 12, 23, 32, 64, 47, 116, 230.
 - магнитометра: 2, 6, 8, 10, 15, 20, 25, 30.
• Входной уровень «0» на шине I2C: -0,3 ... 0,3 * Vcc В.
• Входной уровень «1» на шине I2C: 0,7 * Vcc ... Vcc + 0,3 В.
• Рабочая температура: -40...85 °С.
• Габариты модуля: 30х30 мм.

Более подробную информацию (в том числе и о калибровке модулей датчика) можно найти по ссылке https://wiki.iarduino.ru/page/Trema_IMU9/#h3_5

ЛАБОРАТОРНАЯ РАБОТА

№ 7 РАБОТА СЕНСОРА IMU 9DOF

Цель работы: Познакомиться с принципами работы гироскопа.

Задачи: Загрузить скетч для считывания данных с гироскопа, посмотреть в мониторе порта Arduino IDE считываемые данные.

Порядок выполнения работы

1.Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

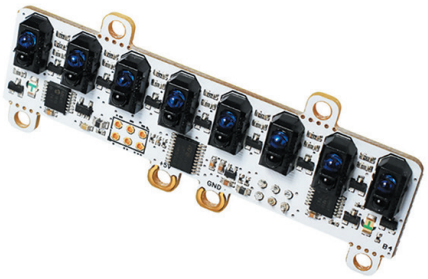
4. Установите в Arduino IDE библиотеку SGBotic_I2CPing.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

```
#include <Wire.h> //    Подключаем библиотеку для работы с аппаратной шиной I2C, до подключения библиотеки iarduino_Position_BMX055.
#include <iarduino_Position_BMX055.h> // Подключаем библиотеку iarduino_Position_BMX055 для работы с Trema-модулем IMU 9 DOF.
iarduino_Position_BMX055 sensor(BMX); // Создаём объект sensor, указывая, что требуется работать со всеми датчиками модуля.
//   Если указать параметр BMA - то объект будет работать только с акселерометром.
//   Если указать параметр BMG - то объект будет работать только с гироскопом.
//   Если указать параметр BMM - то объект будет работать только с магнитометром.
//   Если указать параметр BMX - то объект будет работать со всеми датчиками сразу.
void setup(){
     Serial.begin(9600); // Инициируем передачу данных в монитор последовательного порта на скорости 9600 бит/сек.
     while(!Serial){} // * Ждём готовность Serial к передаче данных в монитор последовательного порта.
     sensor.begin(&Wire); // Инициируем работу со всеми датчиками модуля, так как именно для работы с ними создан объект sensor. Указав ссылку на объект для работы с шиной I2C, на которой находится модуль (по умолчанию &Wire).
}

void loop(){
     sensor.read(); //   Функция read() читает данные того датчика, для которого был создан объект sensor.
     Serial.print((String) "КУРС="  +sensor.axisZ+", "); //   КУРС   - угол поворота вдоль оси Z самолёт меняет направление полёта (0: север, +90: запад, -90: восток, ±180: юг).
     Serial.print((String) "ТАНГАЖ="+sensor.axisX+", "); //   ТАНГАЖ - угол наклона оси X (нос самолёта) относительно горизонта (0: горизонтально, +90: нос вверх, -90: нос вниз).
```

```
      Serial.print((String) "КРЕН="  +sensor.axisY+"\r\n");      //    КРЕН   - угол
наклона оси Y (крылья самолёта) относительно горизонта (0:горизонтально, +90:левое вверх, -90:левое
вниз, ±180: бочка).
//    ПРИМЕЧАНИЕ:                                                    //
//    Углы Эйлера удобно представлять на примере самолёта.       //
//    Ось X проходит вдоль фюзеляжа от хвоста к носу.             //
//    Ось Y проходит от правого крыла к левому.                    //
//    Ось Z проходит через центр самолёта снизу вверх.            //
//    КРЕН   поворот самолёта вдоль оси X (наклон крыльев).      //    Для удобства в библиотеке под
углом КРЕН   подразумевается наклон оси Y относительно горизонта (наклон крыльев). Крен берётся из
переменной axisY.
//    ТАНГАЖ поворот самолёта вдоль оси Y (нос вверх/вниз).      //    Для удобства в библиотеке под
углом ТАНГАЖ подразумевается наклон оси X относительно горизонта (нос вверх/вниз). Тангаж берётся
из переменной axisX.
//    КУРС   поворот самолёта вдоль оси Z (направление).        //   Курс берётся из переменной
axisZ.
}
```

ДАТЧИК ЛИНИИ OCTOLINER



Датчик линии на мобильной платформе выполняет несколько ключевых функций, которые делают его незаменимым в различных приложениях. Вот основные причины, зачем он нужен:

• Следование по линиям: Основная функция датчика линии – это возможность следовать за заданным маршрутом, который обозначен линией (например, черной лентой на светлом фоне). Это позволяет мобильным платформам, таким как роботы или автоматизированные транспортные средства, перемещаться по определенному пути без необходимости постоянного контроля со стороны оператора.

• Навигация: Датчики линии помогают в навигации по заранее заданным маршрутам. Это особенно полезно в производственных помещениях, складах или на выставках, где необходимо перемещение по определенным путям.

• Упрощение управления: Использование датчиков линии значительно упрощает управление мобильной платформой. Вместо сложных алгоритмов навигации можно использовать простые правила следования за линией, что делает систему более интуитивной и легкой в реализации.

• Снижение затрат на оборудование: Датчики линии являются более доступными по сравнению с другими системами навигации (например, GPS или лазерными дальномерами). Это делает их идеальными для образовательных проектов и прототипов.

• Гибкость применения: Датчики линии могут быть использованы не только в робототехнике, но и в других областях – например, в автоматизированных системах сборки или транспортировки товаров.

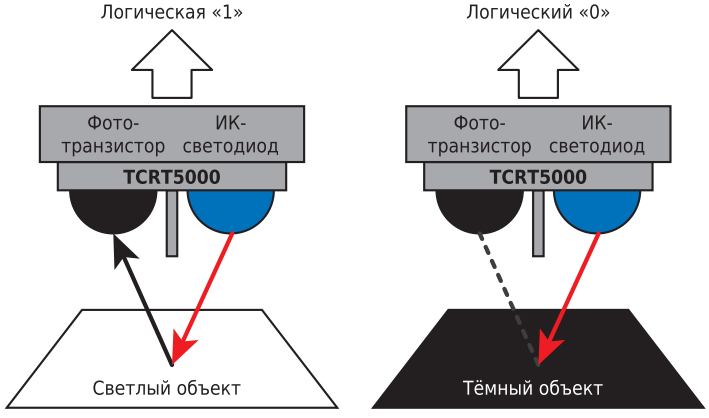
• Надежность и точность: Датчики линии обеспечивают высокую точность определения положения линии и могут работать в различных условиях освещения, что делает их надежными для использования в реальных сценариях.

• Легкость интеграции: Датчики легко интегрируются с другими компонентами системы управления мобильной платформой (например, моторами или дополнительными сенсорами), что позволяет создавать более сложные и функциональные системы.

В целом датчик линии является важным инструментом для обеспечения автономности и эффективности мобильных платформ, позволяя им выполнять задачи с высокой степенью точности и надежности.

В мобильной платформе Optima-Drive в качестве датчика линии используется Octoliner V2.

На борту сенсорной сборки расположены восемь датчиков линии на оптопаре TCRT5000. Каждый сенсор способен отличить все оттенки серого – от белого до чёрного.



Датчик состоит из двух элементов – светодиода (излучателя) и фототранзистора (приёмника). Когда светодиод излучает инфракрасный свет, световой поток отражается от поверхности и попадает на фототранзистор, где преобразуется в электрический сигнал.

Тёмный цвет отражает меньше света, светлый – больше. Используя несколько таких датчиков, робот определяет тёмную линию трассы и следует по ней.

Датчики линии подключены к собственному 32-разрядному микроконтроллеру STM32F030F4P6 с вычислительным ядром ARM Cortex-M0. Контроллер собирает данные с восьми датчиков линии и передаёт управляющей платформе по интерфейсу I²C. Интенсивность излучения и чувствительность фотоприёмников можно регулировать программно.

Модуль программируется в среде Arduino IDE через готовую библиотеку Octoliner, а для Raspberry Pi есть OctolinerPi на Python. С помощью библиотек можно быстро настроить чувствительность сенсора и приступить к считыванию данных.

Характеристики модуля:

- Модель: Amperka Octoliner v2 (AMP-B133);
- Количество каналов: 8;
- Чувствительные элементы: оптопары TCRT5000;
- Микроконтроллер: STM32F030F4P6;
- Интерфейс: I²C;
- Частота опроса: 190 Гц (8-канальный режим);
- Напряжение питания: 3,3–5 В;
- Потребляемый ток: до 87 мА;
- Размеры: 80×30×16 мм.

ЛАБОРАТОРНАЯ РАБОТА

№ 8 ДАТЧИК ЛИНИИ OCTOLINER V2

Цель работы: Изучить принцип работы датчика линии на примере датчика Octoliner.

Задачи: Изучить приведенный пример программного кода использования датчика линии и написать свою управляющую программу, обеспечивающую движение платформы по черной линии.

Порядок выполнения работы

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

4. Установите в Arduino IDE библиотеку Octoliner.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

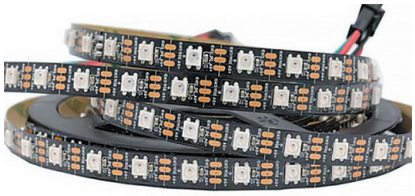
```
#include <Wire.h>
#include <Octoliner.h>

// Создание объекта Octoliner с адресом 42
Octoliner octoliner(42);

// Пины ENA двигателей
#define ENA 2
#define ENB 7
#define ENA_2 8
#define ENB_2 13

// Пины двигателей
// Передний левый
#define IN1 11
#define IN2 12

// Передний правый
#define IN3 9
#define IN4 10

// Задний левый
#define IN5 3
#define IN6 4
```

```
// Задний правый
#define IN7 5
#define IN8 6

// Переменные для PID-регулятора
float lastError = 0, integral = 0;
float Kp = 5.3;
float Ki = 0.01;
float Kd = 3.5;

// Базовая скорость моторов
int baseSpeed = 120;

void setup() {
  // Инициализация пинов моторов
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(IN5, OUTPUT);
  pinMode(IN6, OUTPUT);
  pinMode(IN7, OUTPUT);
  pinMode(IN8, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(ENA_2, OUTPUT);
  pinMode(ENB_2, OUTPUT);
  digitalWrite(ENA, 1);
  digitalWrite(ENB, 1);
  digitalWrite(ENA_2, 1);
  digitalWrite(ENB_2, 1);

  // Инициализация Octoliner
  Wire.begin();
  octoliner.begin();
  octoliner.setSensitivity(255);

  Serial.begin(9600);
}

void loop() {
  // Считывание положения линии
  float position = octoliner.trackLine();

  // Вывод положения в Serial Monitor
  Serial.println("position: " + String(position));
```

```
  // Масштабируем значение для более широкого диапазона
  int scaledError = position * 10;

  // PID-регулятор
  integral += scaledError;
  float derivative = scaledError - lastError;
  float correction = Kp * scaledError + Ki * integral + Kd * derivative;
  lastError = scaledError;

  Serial.println("correction: " + String(correction));

  // Расчёт скоростей для каждого колеса
  int speedFL = baseSpeed + correction; // Передний левый
  int speedFR = baseSpeed - correction; // Передний правый
  int speedBL = baseSpeed + correction; // Задний левый
  int speedBR = baseSpeed - correction; // Задний правый

  // Управление моторами
  setMotorDirection(IN1, IN2, speedFL); // Передний левый
  setMotorDirection(IN3, IN4, speedFR); // Передний правый
  setMotorDirection(IN5, IN6, speedBL); // Задний левый
  setMotorDirection(IN7, IN8, speedBR); // Задний правый

  // Задержка для повторного измерения (200 мс)
  delay(150);
}

// Функция управления моторами
void setMotorDirection(int dirPin1, int dirPin2, int speed) {
  if (speed > 0) {
    digitalWrite(dirPin1, HIGH);
    digitalWrite(dirPin2, LOW);
  } else {
    digitalWrite(dirPin1, LOW);
    digitalWrite(dirPin2, HIGH);
    speed = -speed;
  }
  analogWrite(dirPin1, speed);
}
```

Платформа должна двигаться по чёрной линии. Для увеличения скорости реакции на изменения при поворотах отрегулируйте коэффеценты PID-контроллера.

RGB-ЛЕНТА

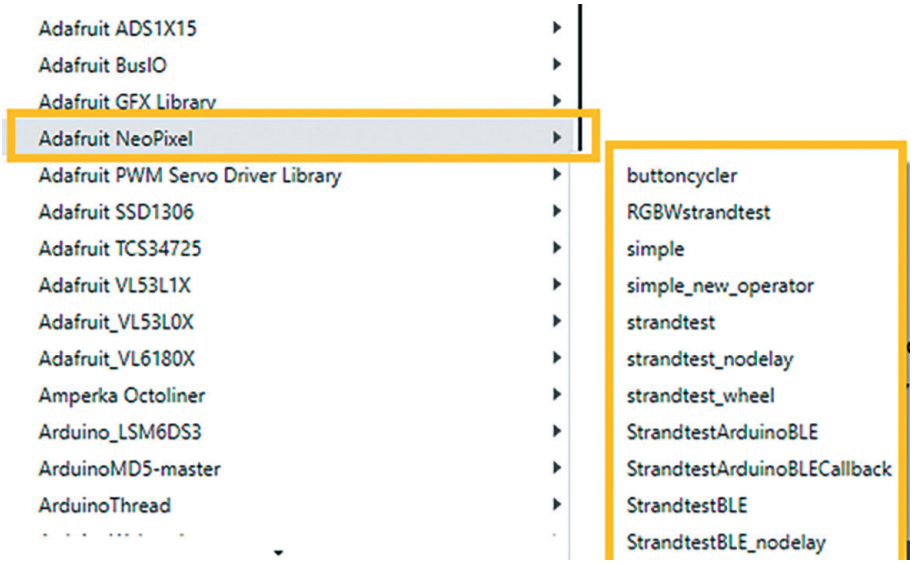


На мобильном роботе Optima-Drive установлены по бокам две светодиодные адресные ленты на базе чипа WS2812. Адресная светодиодная лента представляет собой ленту из адресных диодов, один такой светодиод состоит из RGB-светодиода и контроллера. Благодаря такой начинке есть возможность управлять цветом (яркостью r g b) любого светодиода в ленте. Адресная лента может иметь 3-4 контакта для подключения, два из них всегда питание (5V и GND), и остальные (один или два) – логические, для управления.

Светодиодные ленты, установленные на мобильной платформе Optima-Drive, имеют 18 светодиодов в каждой ленте и один контакт для управления, который подключен к пину 44 для одной ленты и к пину 46 другой ленты контроллера Arduino Mega 2560.

Сигналы светодиодной ленты на мобильной платформе можно использовать, например, для индикации окончания загрузки операционной системы Raspberry Pi – в процессе загрузки ленты моргают красным цветом, после загрузки – загораются зеленым цветом.

При установке библиотеки Adafruit_NeoPixel.h можно увидеть в примерах несколько других программ. Попробуйте запустить другие программы из примеров.



ЛАБОРАТОРНАЯ РАБОТА

№ 9 УПРАВЛЕНИЕ СВЕТОДИОДНОЙ ЛЕНТОЙ

Цель работы: Познакомиться с принципом управления адресными светодиодными лентами.

Задачи: Последовательно включать светодиоды от первого до восемнадцатого, использовать красный цвет.

Порядок выполнения работы

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

4. Установите в Arduino IDE библиотеку FastLED.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

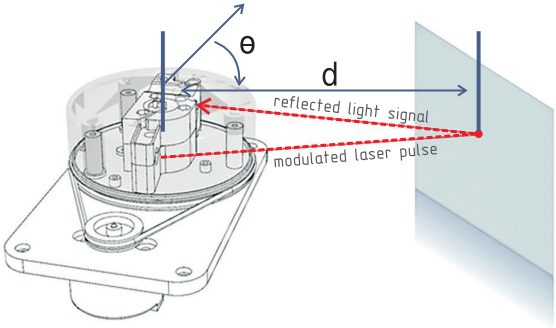
```
#include "FastLED.h"

// Указываем, какое количество пикселей у нашей ленты.
#define LED_COUNT 18

// Указываем, к какому порту подключен вход ленты DIN.
#define LED_PIN 44 // Правый борт, 46 пин – левый борт

// Создаем переменную strip для управления нашей лентой.
CRGB strip[LED_COUNT];

void setup() {
  // Добавляем ленту
  FastLED.addLeds<WS2812B, LED_PIN, GRB>(strip, LED_COUNT);
}

void loop() {
  // Включаем все светодиоды
  for (int i = 0; i < LED_COUNT; i++) {
    strip[i] = CRGB::Red; // Красный цвет.
  }
  // Передаем цвета ленте.
  FastLED.show();

  // Ждем 500 мс.
  delay(500);
```

```
  // Выключаем все светодиоды.
  for (int i = 0; i < LED_COUNT; i++) {
    strip[i] = CRGB::Black; // Черный цвет, т.е. выключено.
  }
  // Передаем цвета ленте.
  FastLED.show();

  // Ждем 500 мс.
  delay(500);
}
```

Задание для самостоятельной работы

Напишите программу для RGB-ленты, реализующую алгоритм – при движении платформы вперед «бегущий огонь» ленты зеленого цвета движется вперед, при движении назад – красного цвета назад.

ПОЛУЧЕНИЕ ДАННЫХ ТЕХНОЛОГИЧЕСКОЙ ИНФОРМАЦИИ С ЛАЗЕРНОГО ДАЛЬНОМЕРА (LIDAR)



Лидар (Light Detection and Ranging – обнаружение, идентификация и определение дальности с помощью света) – это технология, использующая лазерные импульсы для измерения расстояний до объектов.

Базовая идея лидара проста: датчик испускает лазерные лучи в разных направлениях и ждёт, пока их отражения вернутся. Скорость света известна, и время в пути туда и обратно даёт точную оценку расстояния.

Лидар представляет собой компактное и доступное решение для создания карт окружающей среды и определения расстояний до объектов. Его использование на мобильных платформах обусловлено несколькими факторами:

• Точность измерений: Лидар обеспечивает высокую точность и разрешение при измерении расстояний, что позволяет создавать детализированные карты окружающей среды.

• Скорость сканирования: Лидар способен выполнять сканирование на скорости до 8000 точек в секунду, что делает его идеальным для динамичных сред, где требуется быстрая реакция.

• Широкий угол обзора: Угол сканирования составляет 360 градусов, что позволяет получать полное представление о пространстве вокруг мобильной платформы без необходимости поворота самого устройства.

• Простота интеграции: Лидар легко интегрируется с различными микроконтроллерами и одноплатными компьютерами, такими как Arduino или Raspberry Pi.

• Надежность: Лидары обладают высокой устойчивостью к внешним условиям, таким как освещение или погодные условия, что делает их надежными в различных сценариях использования.

Использование лидара на роботизированной мобильной платформе позволяет решать множество задач:

• Картографирование окружающей среды: С помощью лидара можно создавать карты местности в реальном времени. Это особенно полезно для автономных роботов, которые должны ориентироваться в незнакомых пространствах.

• Обнаружение препятствий: Лидар позволяет эффективно обнаруживать препятствия на пути движения робота. Это критически важно для обеспечения безопасности и предотвращения столкновений.

• Навигация и локализация: Используя данные от лидара, можно реализовать алгоритмы навигации и локализации робота в пространстве. Это позволяет ему самостоятельно перемещаться по заданному маршруту или избегать препятствий.

Для обеспечения более точного моделирования и визуализации данных с лидара RPLidar A1 в среде Gazebo и RViz необходимо выполнить переход от использования реального устройства к виртуальной модели. Это включает настройку соответствующих плагинов Gazebo для симуляции данных с лидара, а также конфигурацию RViz для отображения получаемых данных. Такой подход позволяет тестировать алгоритмы навигации и обработки данных без необходимости постоянного подключения физического устройства, что значительно упрощает разработку и отладку робототехнических систем.

Gazebo – это высококачественный симулятор, который позволяет моделировать физическое поведение роботов в сложных трехмерных средах. Он поддерживает реалистичную физику, включая динамику, столкновения и взаимодействие с окружающей средой. Gazebo предоставляет возможность создавать сложные сценарии тестирования, что делает его идеальным инструментом для разработки и отладки робототехнических систем.

Основные функции Gazebo

• Физическая симуляция: Gazebo использует мощные физические движки (такие как ODE, Bullet или DART) для моделирования взаимодействия объектов.

• Поддержка различных сенсоров: Симулятор может имитировать работу различных сенсоров, таких как камеры, лидары и ультразвуковые датчики.

• Моделирование окружающей среды: Gazebo позволяет создавать сложные 3D-сцены с различными объектами и препятствиями.

Gazebo имеет встроенную поддержку ROS2, что позволяет легко обмениваться данными между симуляцией и реальными роботами.

RViz – это инструмент визуализации данных в реальном времени для ROS. Он позволяет разработчикам визуализировать информацию о состоянии робота, его окружении и сенсорах. RViz предоставляет графический интерфейс для отображения различных типов данных, таких как карты, траектории движения, данные с сенсоров и многое другое.

Основные функции RViz

• Визуализация данных: RViz может отображать различные типы данных, включая облака точек, карты местности и модели роботов.

• Интерактивность: Пользователи могут взаимодействовать с визуализированными данными, изменяя параметры отображения или управляя движением робота.

• Поддержка различных форматов данных: RViz может обрабатывать данные из различных источников ROS2, включая сообщения о состоянии робота и данные с сенсоров.

RViz работает в связке с ROS, что позволяет ему получать данные в реальном времени от запущенных узлов.

Взаимодействие Gazebo и RViz с ROS

Gazebo и RViz могут работать совместно в рамках одного проекта на базе ROS, обеспечивая мощный инструмент для разработки и тестирования робототехнических приложений:

1. Создаем модель робота в Gazebo и запускаем симуляцию. Робот начинает выполнять заданные команды или алгоритмы навигации.

2. Во время работы симуляции Gazebo отправляет данные о состоянии робота (например, его позицию или данные с сенсоров) через сообщения ROS.

3. RViz получает эти сообщения и визуализирует их в реальном времени. Это позволяет нам наблюдать за поведением робота в симулированной среде. После успешного тестирования алгоритмов в симуляции можно перенести их на реальный робот без значительных изменений кода.

Основные различия между Gazebo и RViz

Цель использования:

Основная цель Gazebo – это симуляция физического поведения роботов в трехмерной среде. Он предоставляет возможность моделировать взаимодействие робота с окружающей средой, включая

физику, динамику и столкновения. Gazebo позволяет разработчикам тестировать алгоритмы управления и навигации в условиях, приближенных к реальным.

RViz предназначен для визуализации данных, получаемых от робота или симулятора. Он позволяет разработчикам видеть информацию о состоянии системы в реальном времени, включая данные с сенсоров, карты местности и траектории движения.

Тип визуализации:

В Gazebo вы видите полную 3D-сцену с моделями роботов и окружающей средой. Это позволяет вам наблюдать за физическим поведением робота в контексте его окружения.

RViz предоставляет более абстрактный подход к визуализации данных. Вы можете отображать различные типы информации (например, облака точек от лидара или карты) без необходимости отображать всю 3D-сцену. Это может быть полезно для анализа данных с сенсоров или состояния системы.

Интерактивность:

В Gazebo вы можете управлять симуляцией, изменять параметры физики или взаимодействовать с объектами в сцене.

RViz больше ориентирован на анализ данных. Вы можете настраивать отображение различных типов информации и получать представление о состоянии системы без необходимости взаимодействовать с самой симуляцией.

Почему оба инструмента важны?

Комплексный анализ: Использование обоих инструментов позволяет получить более полное представление о работе робототехнической системы. Gazebo помогает понять физическое поведение робота в среде, а RViz позволяет анализировать данные с сенсоров и состояние системы.

Отладка алгоритмов: В Gazebo вы можете тестировать алгоритмы управления в реалистичных условиях, а затем использовать RViz для анализа результатов работы этих алгоритмов, на основе данных с сенсоров.

Разделение задач между Gazebo и RViz позволяет разработчикам сосредоточиться на разных аспектах разработки – симуляции физики и визуализации данных – что делает процесс разработки более эффективным.

Лабораторная работа далее поможет на практике получить данные с лидара и отобразить полученные данные в RViz.

Большинство ведущих лидаров используют один из четырёх методов направления лазерных лучей в разные стороны:

- Вращающийся лидар.
- Механический сканирующий лидар использует зеркало, перенаправляя единственный лазерный луч в разных направлениях.
- Некоторые лидары используют подход под названием «микроэлектромеханическая система» (МЭМС) для управления зеркалом.
- Активная фазированная антенная решетка использует ряд излучателей, способных изменять направление лазерного луча, подстраивая относительную фазу сигнала между соседними передатчиками.

- Лидар на основе вспышек подсвечивает всю область сразу. Существующие технологии используют один широкоугольный лазер. Технология испытывает трудности с большими расстояниями, поскольку до любой точки доходит лишь малая часть лазерного света.

Лидар запускает быстрые короткие импульсы лазерного излучения на объект (поверхность) с частотой до 150 000 импульсов в секунду. Датчик на приборе измеряет промежуток времени, необходимый для возврата импульса. Свет движется с постоянной и известной скоростью, поэтому лидар может вычислить расстояние между ним и целью с высокой точностью.

Существуют две основные категории импульсных лидаров: микроимпульсные и высокоэнергетические системы.

Микроимпульсные лидары работают на более мощной компьютерной технике с большими вычислительными возможностями.

Эти лазеры меньшей мощности и классифицируются как «безопасные для глаз», что позволяет использовать их практически без особых мер предосторожности.

В конструкции мобильной платформы Optima-Drive используется сканирующий лидар RPLIDAR A1 на вращающейся платформе, который способен создать карту окружающего пространства в плоскости.

RPLIDAR A1 – это решение для 360-градусного 2D-лазерного сканера (LIDAR), разработанное компанией SAMTEC. Система может выполнять сканирование на 360 градусов в пределах 12-метрового диапазона. Полученные 2D-данные облака точек могут быть использованы при картографировании, локализации и моделировании объектов / окружающей среды.

Частота сканирования RPLIDAR A1 достигла 5,5 Гц при выборке 1450 точек в каждом раунде. И он может быть настроен максимум на 10 Гц. Он может отлично работать во всех видах внутренней среды и на открытом воздухе без воздействия прямых солнечных лучей.

Основные характеристики:

- Диапазон расстояния (м) – 0,15–12 (белые объекты)
- Угловой диапазон (°) – 0–360
- Поле сканирования (°) – -1,5–1,5
- Разрешение расстояния (мм) – < 1 % от дистанции
- Угловое разрешение (°) – ≤ 1
- Продолжительность выборки (мс) – 0,125
- Частота дискретизации (Гц) – ≥ 8000
- Скорость сканирования (Гц) – 1–10 (5,5)
- Вес (г) – 170
- Диапазон температур (°C) – 0–40 (20)

Характеристики лазера:

- Длина волны лазера (нм) – 775–795
- Мощность лазера (мВт) – 3 (норм)–5 (макс)
- Длительность импульса (мкс) – 110 (норм)–300 (макс)

Коммуникационный интерфейс:

- Интерфейс связи – TTL
- Полоса пропускания (б/с) – 115 200
- Рабочий режим – 8N1
- Выходное напряжение (high, В) – 2,9–3,5 (HIGH)
- Выходное напряжения (low, В) – 0,4 (LOW)
- Входное напряжение (high, В) – 1,6–3,5 (HIGH)
- Входное напряжение (low, В) – -0,3–0,4 (LOW)

Электропитание и потребление:

- Напряжение системы сканера (В) – 4,9–5,5
- Пульсация напряжения системы сканера (мВ) – 20 (рекомендовано)–50
- Ток системы сканера (мА) – 80-100 (спящий режим, 5 В),300–350 (рабочий режим, 5 В)
- Напряжение мотора (В) – 5-9
- Ток мотора (мА) – 100 (при 5 В)

ЛАБОРАТОРНАЯ РАБОТА

№ 10 СКАНИРОВАНИЕ ОКРУЖАЮЩЕГО ПРОСТРАНСТВА С ПОМОЩЬЮ ЛИДАРА

Цель работы: Научиться работать с лидаром, подключенным к Raspberry Pi.

Задачи: Для работы с лидаром запустим приложение RViz. Инструмент RViz позволяет в реальном времени визуализировать на 3D-сцене все компоненты робототехнической системы – системы координат, движущиеся части, показания датчиков, изображения с камер.

Порядок выполнения работы

В первую очередь необходимо подключиться к Raspberry Pi (локально или используя VNC).

Открываем терминал и вводим команды

#source ~/lidar_ws/devel/setup.bash

#ls /dev|grep ttyUSB0

Если лидар подключен и активен, команда вернёт строку:

#ttyUSB0

Открыть новый терминал Ctrl+Shift+T и в нем ввести команды

#source /opt/ros/noetic/setup.bash

Запускаем ROS

#roscore

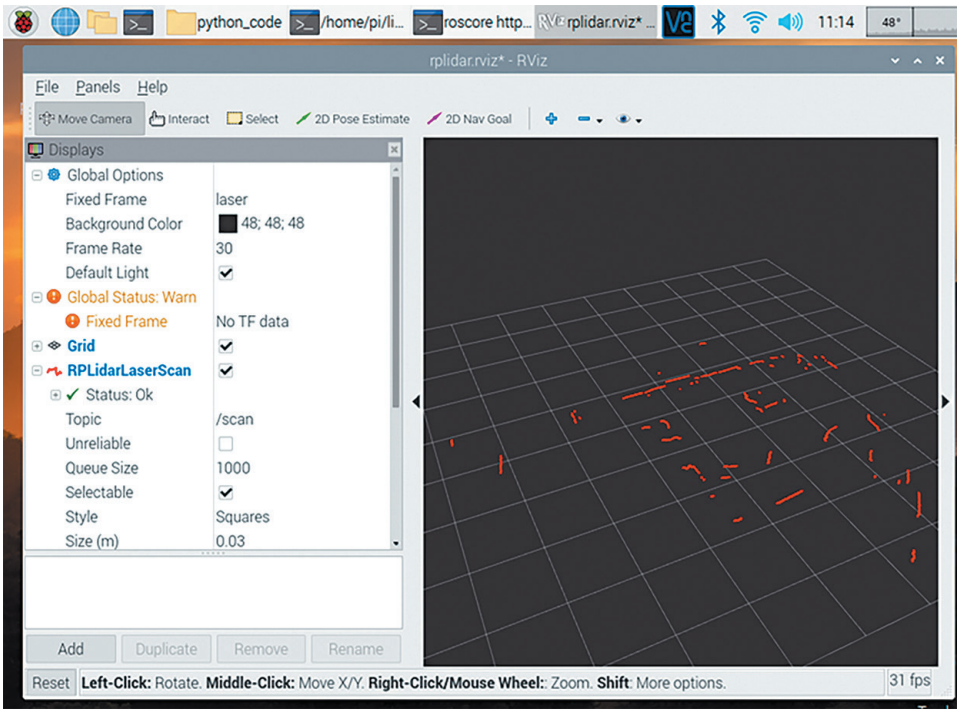
Возвращаемся в первый терминал и перейдем в папку

#cd ~/lidar_ws/src/rplidar_ros/launch/

И запускаем программу

#roslaunch rplidar_ros view_rplidar_a1.launch

Теперь на экране, в основном окне RViz, можно увидеть контуры окружения робота, зафиксированные лидаром.



БЕСПРОВОДНОЕ УПРАВЛЕНИЕ РОБОТОМ

Bluetooth – это стандартный отраслевой протокол, который обеспечивает беспроводное подключение для множества устройств, включая компьютеры, принтеры, мобильные телефоны.

Беспроводной интерфейс с небольшим радиусом действия, получивший название Bluetooth, был разработан в 1994 году инженерами шведской компании Ericsson. Протокол получил своё название в честь Гарольда Синезубого – короля Дании, который примирил знатные семьи из Дании и Норвегии. Начиная с 1998-го развитием и продвижением данной технологии занимается организация Bluetooth Special Interest Group (Bluetooth SIG), основанная компаниями Ericsson, IBM, Intel, Nokia и Toshiba. К настоящему времени список членов Bluetooth SIG включает более 13 тысяч компаний.

В зависимости от мощности Bluetooth-датчики делятся на три класса:

• Первый класс, способный поддерживать устойчивую связь на расстоянии 100–200 метров. В бытовых устройствах встречается редко и используется на промышленном оборудовании.

• Второй класс удерживает стабильную связь на расстоянии 10–20 метров. Такие датчики чаще всего установлены в смартфонах или планшетах.

• Третий класс наименее мощный и подходит для объединения устройств на расстоянии до пяти метров. Устанавливается на небольших гаджетах – фитнес-браслетах, умных часах и так далее.

Raspberry Pi 4 имеет встроенный Bluetooth модуль, который поддерживает как классический Bluetooth, так и Bluetooth Low Energy (BLE). Он работает по стандарту Bluetooth 5.0 и позволяет подключать различные устройства. За беспроводную передачу данных отвечает микросхема Cypress CYW43438 с поддержкой Wi-Fi 802.11b/g/n/ac (2,4 и 5 ГГц) и Bluetooth 5.0 с BLE.

Управлять с помощью Bluetooth можно любыми подсистемами мобильной платформы, используя канал доступа к ним Raspberry Pi – Arduino по Serial интерфейсу (через USB). Рассмотрим возможность управления платформой Optima-Drive по Bluetooth, используя смартфон и приложение для управления, установленное на нем.

Для работы с Bluetooth были установлены необходимые модули **(уже присутствуют в образе системы, устанавливать не надо):**

```
sudo apt update
sudo apt dist-upgrade

sudo apt-get install bluetooth libbluetooth-dev
sudo python3 -m pip install pybluez
```

ЛАБОРАТОРНАЯ РАБОТА

№ 11 ПЕРЕДАЧА УПРАВЛЯЮЩИХ СИГНАЛОВ ПО BLUETOOTH

Цель работы: Научиться использовать возможности беспроводного интерфейса Bluetooth микрокомпьютера Raspberry Pi в связке с микроконтроллером Arduino.

Задачи: Написать программу анализа принимаемых по Bluetooth команд для Raspberry Pi и передачу их в Arduino для управления двигателями мобильной платформы Optima-Drive. В качестве передатчика использовать смартфон с установленным на него управляющим приложением. Установите управляющее приложение на ваш смартфон. В данной работе использовалось приложение.

Порядок выполнения работы

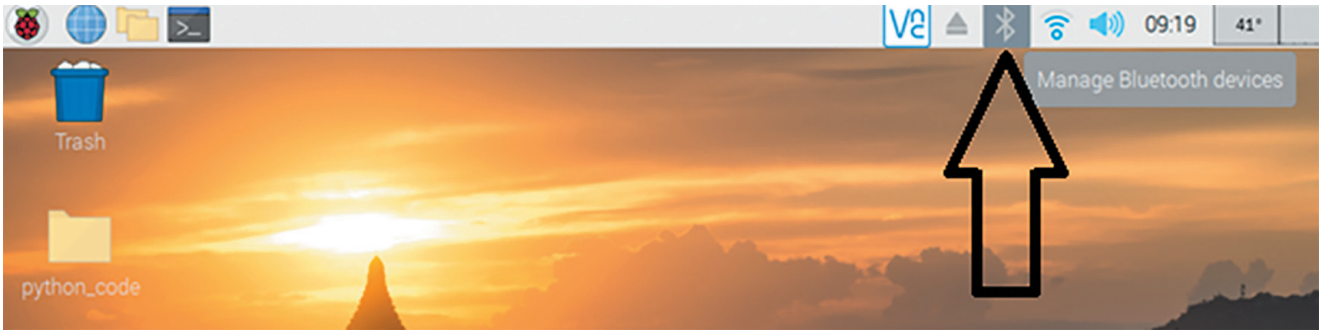
1. Для начала работы необходимо подключить Arduino и Raspberry Pi с помощью заглушки.

2. Подключиться к Raspberry Pi – локально, подключив монитор, клавиатуру и мышь (или посредством VNC после загрузки системы).

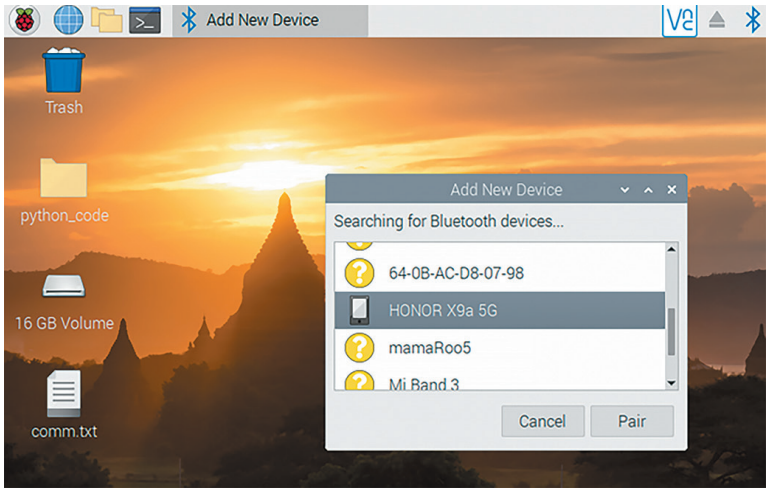
3. Включить питание Optima-Drive и дождаться загрузки операционной системы.

4. Проверьте подключение: на Raspberry Pi выполните команду ls /dev/tty*, чтобы найти имя устройства (обычно это будет что-то вроде /dev/ttyUSB1 или /dev/ttyACM0).

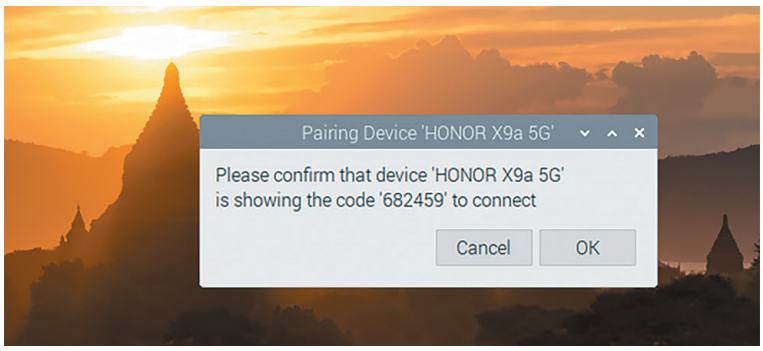
Установите соединение между смартфоном и Raspberyy Pi. Для этого включите сервис Bluetooth на Raspberry Pi, нажав иконку в верхней части экрана.



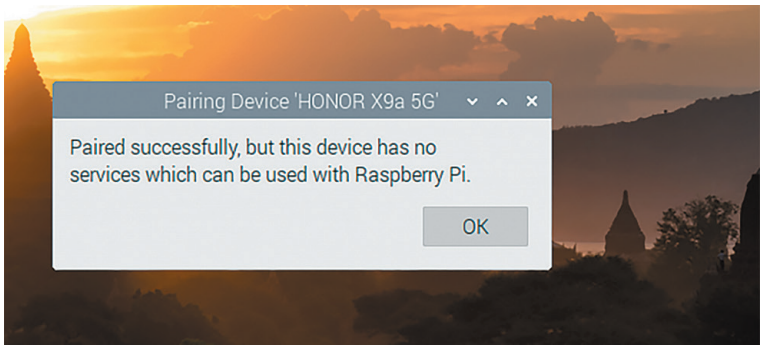
Нажмите на иконку Bluetooth – Add New Device. В списке доступных устройств выберите свой смартфон.



Подтвердите действие на смартфоне и на Raspberry Pi.



Соединение установлено.

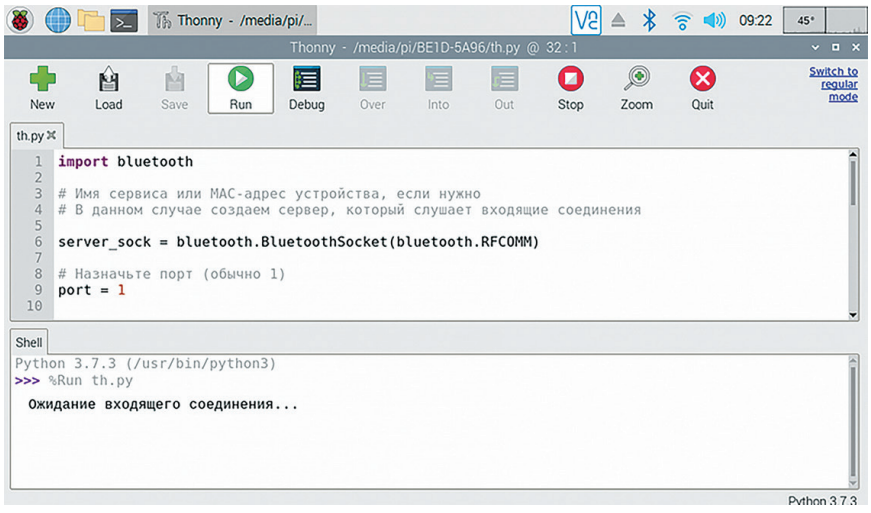


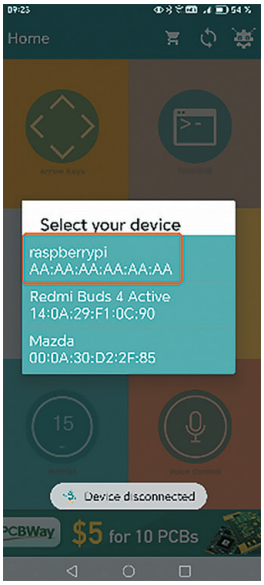
На стороне Raspberry Pi мы можем использовать Python для взаимодействия с Arduino через последовательный порт.

- Запустите Thonny Python IDE. Для этого слева вверху нажимаем иконку «Пуск» (символ малинки)
- Выбираем пункт Programming
- Выбираем Thonny Python IDE
- Нажмите сивол «+» (NEW)
- Скопируйте туда код, представленный ниже (код программ находится на рабочем столе в папке python_code\Скетчи\lab_11)
- Исполнение программы запускается с помощью кнопки Run

```
import bluetooth

# Имя сервиса или MAC-адрес устройства, если нужно
# В данном случае создаем сервер, который слушает входящие
соединения

server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)

# Назначьте порт (обычно 1)
port = 1

# Связываем сокет с портом
server_sock.bind(("", port))
server_sock.listen(1)

print("Ожидание входящего соединения...")

try:
    client_sock, address = server_sock.accept()
    print(f"Соединение установлено с {address}")

    while True:
        data = client_sock.recv(1024)
        if not data:
            break
        print (f"Получено сообщение: {data.decode('utf-8')}")
except OSError:
    pass
finally:
    print ("Закрытие соединения")
    client_sock.close()
    server_sock.close()
```

Приложение запустится и Raspberry Pi войдет в режим ожидания.

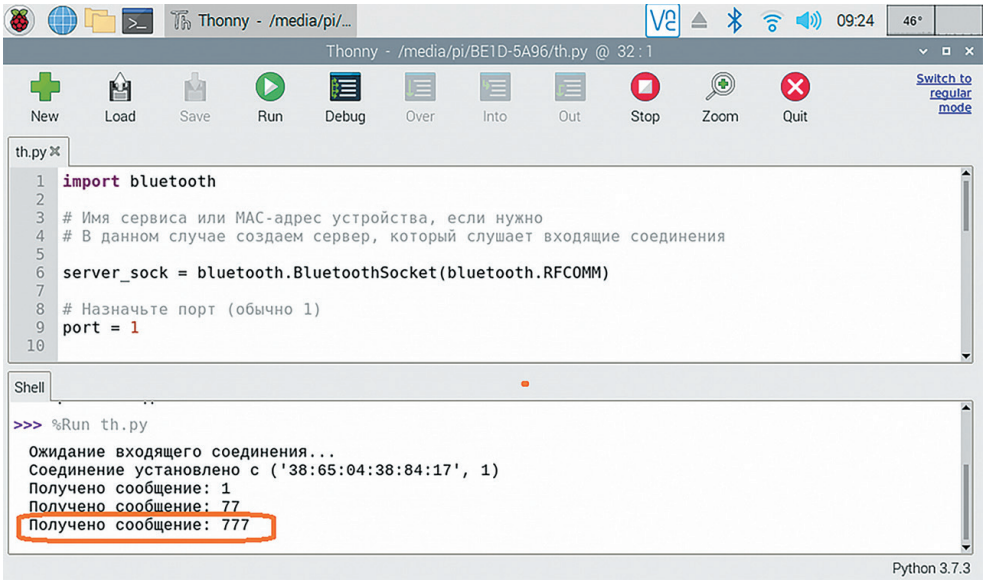




Запустите приложение на смартфоне. Выберете Raspberry Pi в качестве устройства для подключения.

Откроется окно терминала. Введите необходимое сообщение/код и отправьте его на Raspberry Pi.

В окне терминала Raspberry Pi мы увидим принятый код.



Таким образом мы можем принимать управляющие команды (символы) для управления платформой.

Теперь необходимо на примере лабораторных работ:

• Лабораторная работа № 4. Взаимодействие Arduino с Raspberry Pi.

• Лабораторная работа № 5. Управление направлением движения платформы.

1. Дописать рассмотренный выше код таким образом, чтобы Raspberry Pi принимала управляющий код (символ) и отправляла его на Arduino Mega 2560, где принятый код распознавался микроконтроллером и с помощью, например, структуры switch...case, управлял двигателями платформы Optima-Drive.

2. Запустить Arduino IDE на Raspberry Pi.

3. Загрузить написанную программу (скетч) в Arduino, которая будет обрабатывать входящие данные от Raspberry Pi.

Прежде чем тестировать программы, необходимо установить платформу на подставки, обеспечив свободное вращение колёс.

ЧАСТЬ 3. ТЕХНИЧЕСКОЕ ЗРЕНИЕ РОБОТОВ. МОДУЛЬ ТЕХНИЧЕСКОГО ЗРЕНИЯ (МТЗ) МОБИЛЬНОЙ ПЛАТФОРМЫ

Модуль технического зрения мобильной платформы Optima-Drive выполнен на базе системы компьютерного зрения в виде компактного модуля камеры OpenMV H7. Она отличается от обычных камер дополнительной начинкой с микроконтроллером для обработки изображения на лету и управления внешними устройствами.

Захватом изображения занимается светочувствительная КМОП-матрица OmniVision OV7725 размером 1/3″ с разрешением 640×480. Камера снимает видео в 8-битном режиме оттенков серого или цветном 16-битном формате RGB565 с частотой до 75 кадров в секунду. Поддерживаются форматы сжатия MJPEG, GIF и несжатое видео RAW. На камере предусмотрена подсветка RGB-светодиодом и два ИК-светодиода для съёмки в темноте.

Объектив с фокусным расстоянием 2,8 мм и диафрагмой F2.0 крепится через байонет со стандартной резьбой M12 с шагом 0,5 мм, поэтому к OpenMV H7 подходят сменные объективы от GoPro и других портативных камер.

За обработку изображения отвечает 32-битный микроконтроллер STM32H743VI от компании STMicroelectronics с вычислительным ядром ARM Cortex-M7. Процессор работает на тактовой частоте до 480 МГц, у него на борту 1 МБ оперативной памяти SRAM и 2 МБ Flash-памяти.

Начинка справляется с алгоритмами компьютерного зрения разной сложности, среди которых:

- анализ изображений через TensorFlow Lite;
- детекция движения в кадре;
- распознавание лиц;
- отслеживание цветных объектов и маркеров;
- отслеживание движения зрачков;
- определение и считывание QR-кодов, штрих-кодов и AprilTags;
- скоростное отслеживание линии;
- распознавание геометрических объектов;
- сравнение изображения с заданным шаблоном.

Для записи видео и хранения рабочих данных используется карта памяти microSD. Скорость чтения и записи до 100 Мбит/с позволяет оперативно подгружать объекты для машинного зрения.

Умная камера программируется на MicroPython в среде разработки OpenMV IDE с поддержкой русского языка. Она объединяет в себе редактор программного кода, просмотр видеобуфера камеры и построение RGB-гистограмм в реальном времени, чтобы упростить процесс отладки.

Благодаря поддержке MicroPython доступна масса готовых библиотек для управления периферией и оптимизированными для микроконтроллеров алгоритмами обработки изображений. Это позволяет быстрее запрограммировать свою систему на основе существующих «кирпичиков», а не писать всё с нуля.

На OpenMV H7 предусмотрено 10 контактов ввода-вывода общего назначения (GPIO) для подключения внешних устройств.

- 10 пинов поддерживают прерывания.
- 9 пинов умеют выдавать ШИМ сигнал разрядностью 16 бит.

• Пин P6 оснащён 12-разрядными АЦП и ЦАП для подключения аналоговой периферии.
• Три пина предназначены для управления сервоприводами.
• Аппаратные интерфейсы включают в себя 2× UART, 2× I²C, 1× SPI и 1× CAN.

На пинах выдаётся логическое напряжение 3,3 В и ток до 25 мА, но они толерантны к входному напряжению 5 В (кроме пина P6 в режиме АЦП/ЦАП).

Камера питается через порт micro-USB, разъём питания JST PH-2 или напрямую через контакт Vin напряжением от 3,6 до 5 В.

Характеристики:
• Основные чипы: STM32H743VI, OV7725
• Входное напряжение через USB: 5 В
• Входное напряжение через пин Vin и разъём JST: 3,6–5 В
• Ток потребления в фоновом режиме: 110 мА
• Ток потребления в активном режиме: до 170 мА
• Максимальный ток с пина или на пин: 25 мА
• Максимальный выходной ток пина 3V3: 250 мА
• Напряжение логических уровней: 3,3 В
• Карта памяти: microSD
• Размеры: 45×36×30 мм

Микроконтроллер STMicroelectronics STM32H743VI

• Вычислительное ядро: ARM Cortex-M7
• Разрядность: 32 бита
• Тактовая частота: 480 МГц
• SRAM-память: 1 МБ
• Flash-память: 2 МБ
• Пины ввода-вывода: 10 (с поддержкой прерываний)
• Контакты с АЦП / ЦАП: 1 с разрядностью 12 бит
• Контакты с ШИМ: 9 с разрядностью 16 бит
• Аппаратные интерфейсы:
- 2× UART
- 2× I²C
- 1× SPI
- 1× CAN

Камера OmniVision OV7725

• Сенсор изображения: КМОП-матрица
• Размер матрицы: 1/3″
• Разрешение: 640×480
• Частота кадров: 75 к/с (640×480), 150 к/с (320×240)
• Цветной режим: 16 бит (RGB565)
• Чёрно-белый режим: 8 бит
• Сжатие видео: MJPEG, GIF, несжатый RAW

- Байонет объектива: M12/0,5 мм
- Фокусное расстояние объектива: 2,8 мм
- Диафрагма: F2.0
- ИК-фильтр: 650 нм (убираемый)
- Встроенная подсветка:
  - 1× RGB-светодиод
  - 2× ИК-светодиод (850 нм)

Модуль технического зрения может подключаться к микрокомпьютеру (или микроконтроллеру) мобильной платформы по интерфейсам UART или I^2C.

ЛАБОРАТОРНАЯ РАБОТА

№ 12 РАСПОЗНАВАНИЕ ОБЪЕКТА И ДВИЖЕНИЕ ПЛАТФОРМЫ ЗА НИМ (НА ПРИМЕРЕ ЦВЕТНОГО ШАРИКА)

Цель работы: Получить начальные навыки работы с модулем технического зрения.

Задачи: Разработать код программы (для камеры и для Arduino) на основе приведенного примера. Необходимо распознать с помощью камеры объект (например, желтый шарик) и заставить двигаться мобильную платформу к этому объекту, т. е. задача сводится к поиску шарика на плоскости и следованию за ним.

Порядок выполнения работы

Для начала необходимо настроить камеру с помощью OpenMV IDE.

Задача поиска шарика предполагает, что находящаяся на мобильной платформе камера будет постоянно искать шарик, выделяя его среди остальных объектов на изображении с помощью информации о его цвете. И настройка камеры предполагает именно обучение OpenMV H7 Plus на нужный оттенок, а также подбор определенных коэффициентов для работы алгоритма.

Калибровка камеры на выбранный цвет происходит с использованием возможностей официального IDE для камер фирмы OpenMV – OpenMV IDE (рис. 1), которое вы можете скачать напрямую с официального сайта https://openmv.io/pages/download.

После успешной установки приложения подключите камеру к компьютеру при помощи кабеля USB-A – MicroUSB для передачи изображения с OpenMV H7 Plus на ваш экран. Данное подключение позволяет вручную отслеживать корректность обучения камеры и вносить правки в ее работу в режиме реального времени. При успешном подключении значок соединения в левом нижнем углу (рис. 1, № 1) сменится на другой (рис. 2), и при нажатии на него вы подключитесь к модулю.

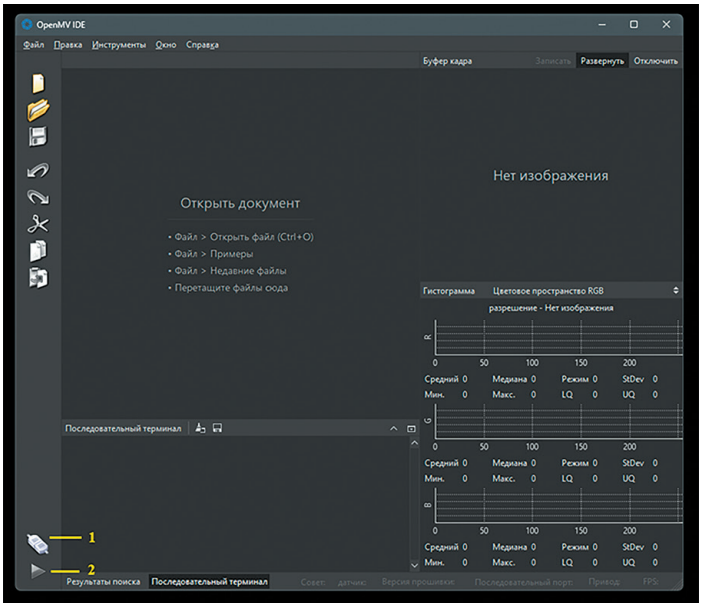


Рисунок 1. Общий интерфейс приложения OpenMV IDE

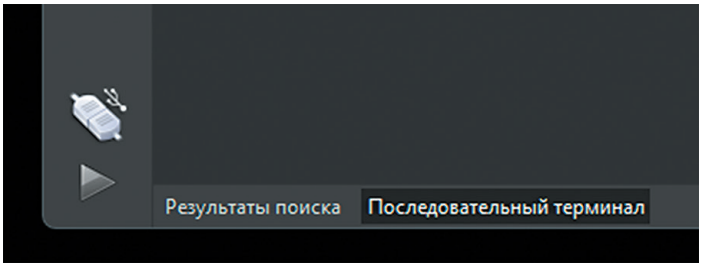


Рисунок 2. Значок подключения к камере

Примечание – Если при подключении кабеля камера не распознается, то попробуйте заменить его на другой и/или проверьте, поддерживает ли он передачу данных.

При запуске OpenMV IDE автоматически открывается заготовленный скрипт-пример helloworld.py. Запустите его, нажав зеленую стрелочку в левом нижнем углу (рис. 2, № 2). Если нужный скрипт по каким-либо причинам не открылся, то найдите на верхней панели «Файл» → «Примеры» → HelloWorld → helloworld.py.

Примечание – Если приложение распознает камеру, но в приложении изображения нет, а только черный фон, то проверьте, сняли ли вы защитную крышку с объектива камеры.

При успешном подключении на экране вы увидите изображение с камеры. Попробуйте перемещать шарик и проверьте, меняется ли картинка в OpenMV IDE (рис. 3).

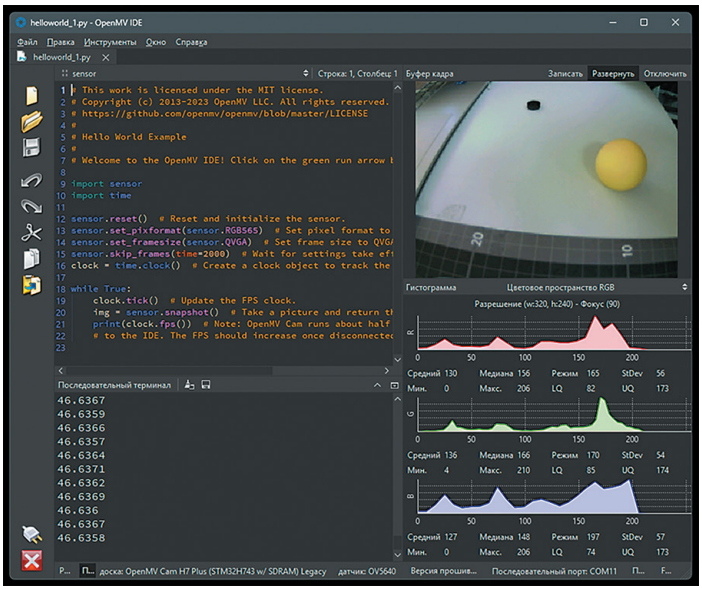


Рисунок 3. Интерфейс приложения OpenMV IDE при корректно подключенной камере

Примечание – При плохом качестве изображения попробуйте отрегулировать объектив камеры, подкрутив его.

Произведите подстройку чувствительности камеры к конкретным условиям освещения. Направьте камеру на объект, с которым вы собираетесь работать. Остановите программу, нажав на крестик внизу слева. В буфере осталось изображение, которое транслировалось с помощью программы helloworld.py. Теперь в приложении OpenMV IDE откройте пункт меню «Инструменты» – «Машинное зрение» – Threshold Editor (Рис. 4).

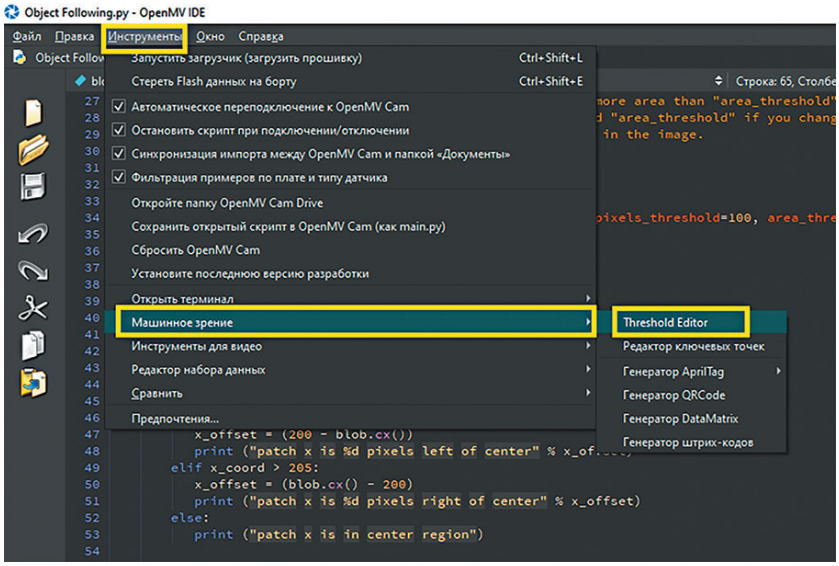


Рисунок 4.

На вопрос системы необходимо ответить «Кадровый буфер» (рис. 5).
(Здесь приведен пример настройки на синий шарик.)

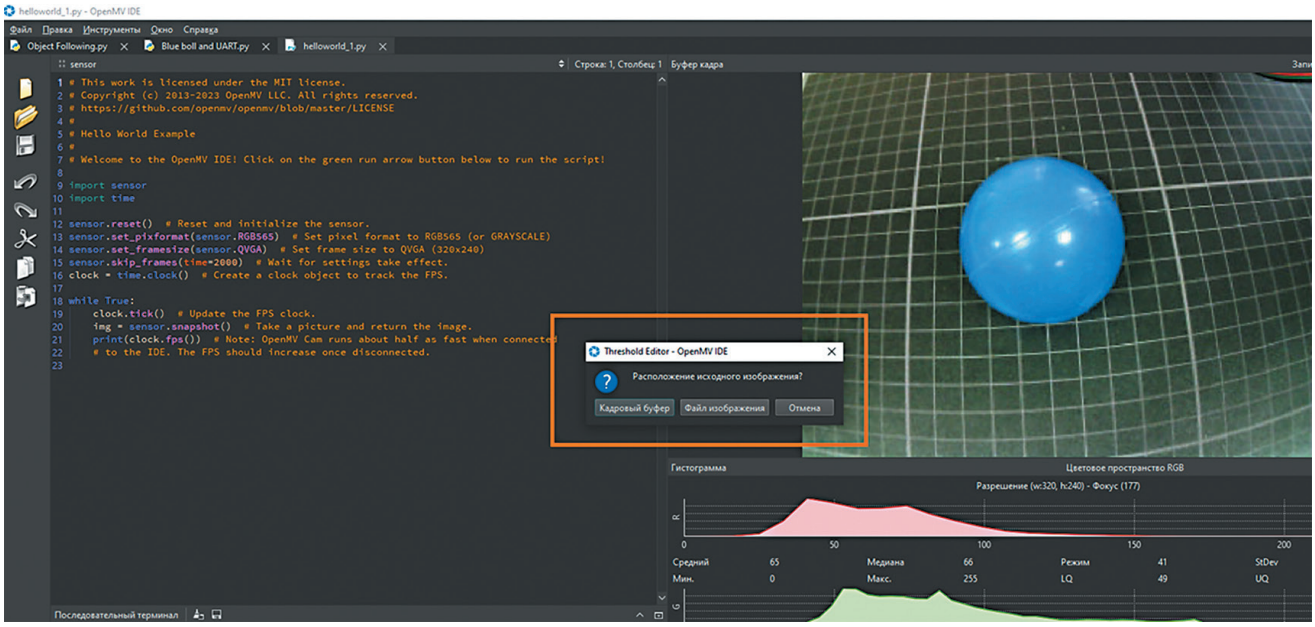


Рисунок 5.

С помощью ползунков, расположенных под изображением, настройте фильтр таким образом, чтобы объект был максимально контрастным. При этом необходимо отфильтровать все помехи (рис. 6).

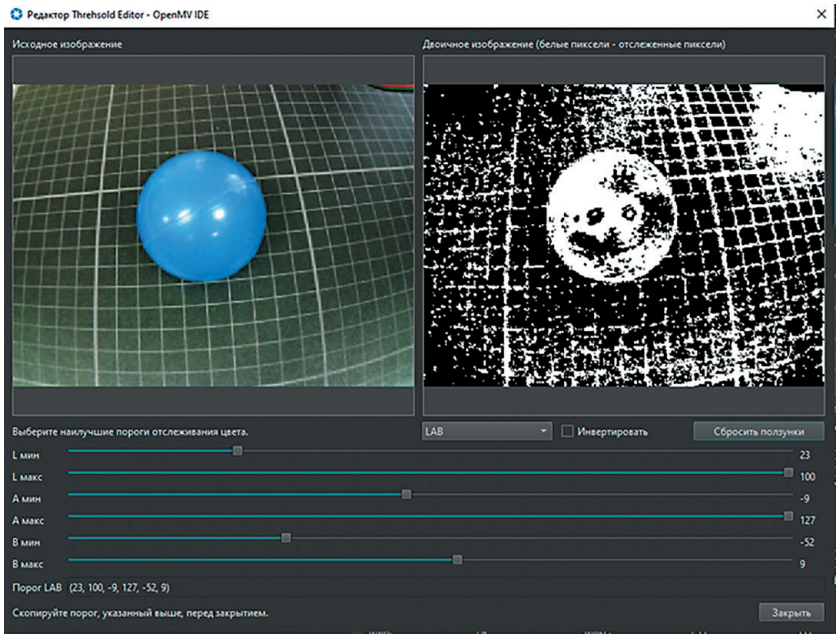


Рисунок 6.

Должно получиться чистое изображение объекта (рис. 7).

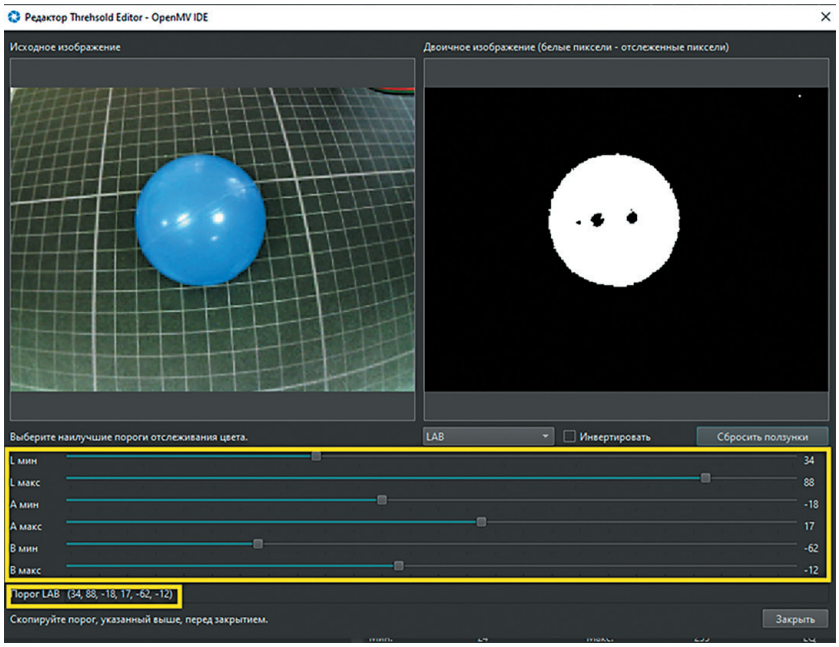


Рисунок 7.

В нижней части экрана показано пороговое значение «Порог LAB (34, 88, -18, 17, -62, -12)». Необходимо сохранить эти значения.

thresholds = (34, 88, -18, 17, -62, -12)

Они понадобятся в дальнейшем при написании кода программ.
Или для нашего желтого шарика картинка должна выглядеть так:

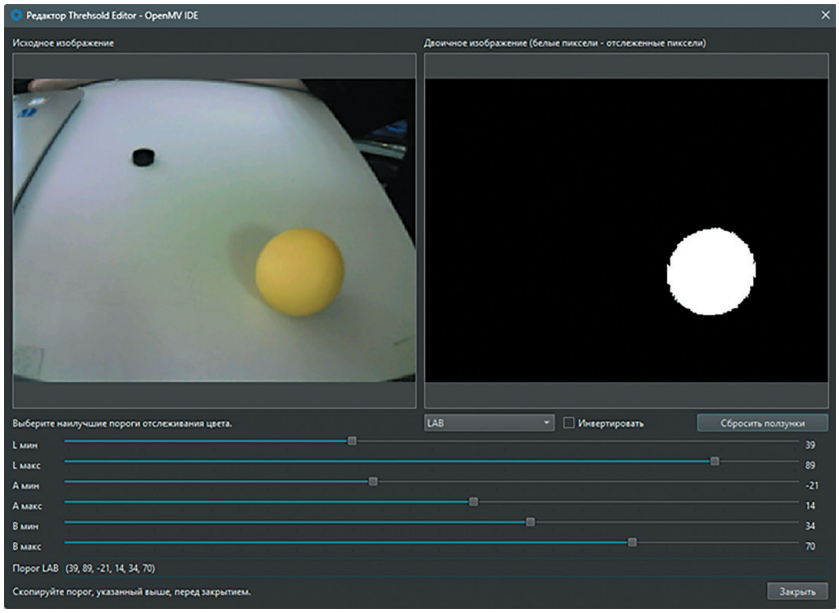
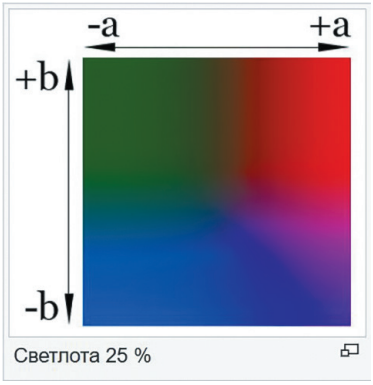


Рисунок 8.

LAB – аббревиатура названия двух разных (хотя и похожих) цветовых пространств. Более известным и распространенным является **CIELAB** (точнее, CIE 1976 L*a*b*), другим – **Hunter Lab** (точнее, Hunter L, a, b). Таким образом, Lab – это неформальная аббревиатура, не определяющая цветовое пространство однозначно. Чаще всего, говоря о пространстве Lab, подразумевают CIELAB.

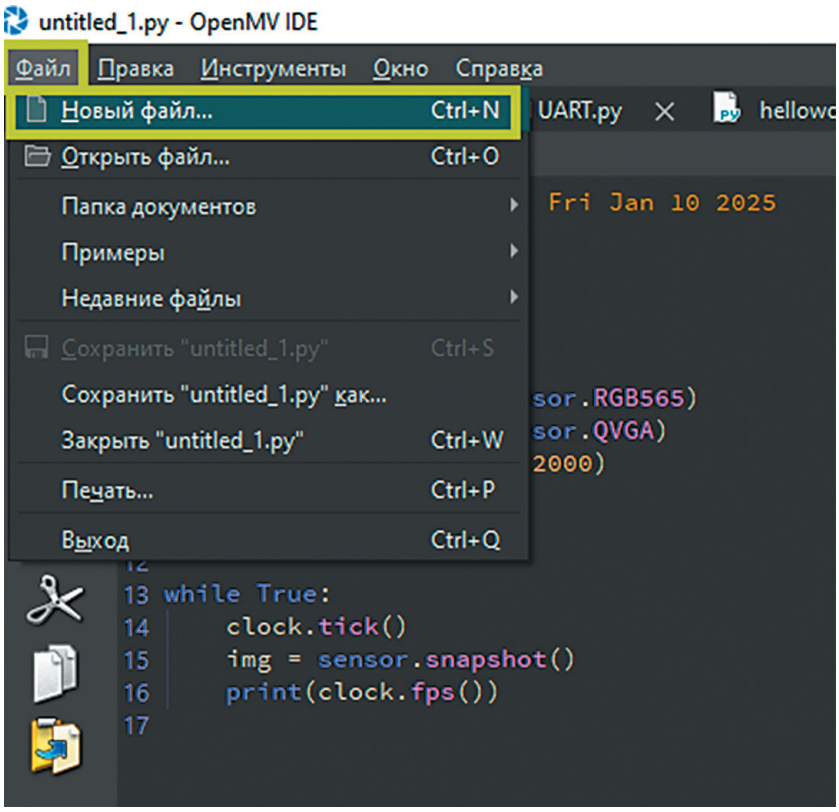


При разработке Lab преследовалась цель создания цветового пространства, изменение цвета, в котором будет более линейным с точки зрения человеческого восприятия (по сравнению с XYZ), то есть с тем, чтобы одинаковое изменение значений координат цвета в разных областях цветового пространства производило одинаковое ощущение изменения цвета.

В цветовом пространстве Lab значение светлоты отделено от значения хроматической составляющей цвета (тон, насыщенность). Светлота задана координатой L (изменяется от 0 до 100, то есть от самого темного до самого светлого), хроматическая составляющая – двумя декартовыми координатами a и b. Первая обозначает положение цвета в диапазоне от зелено-голубого до красно-малинового, вторая – от голубого до желтого.

В отличие от цветовых пространств RGB или CMYK, которые являются по сути набором аппаратных данных для воспроизведения цвета на бумаге или на экране монитора (цвет может зависеть от типа печатной машины, марки красок, влажности воздуха в цеху или производителя монитора и его настроек), Lab однозначно определяет цвет.

Теперь создайте код программы. Нажмите пункт меню «Файл» – «Новый файл...»

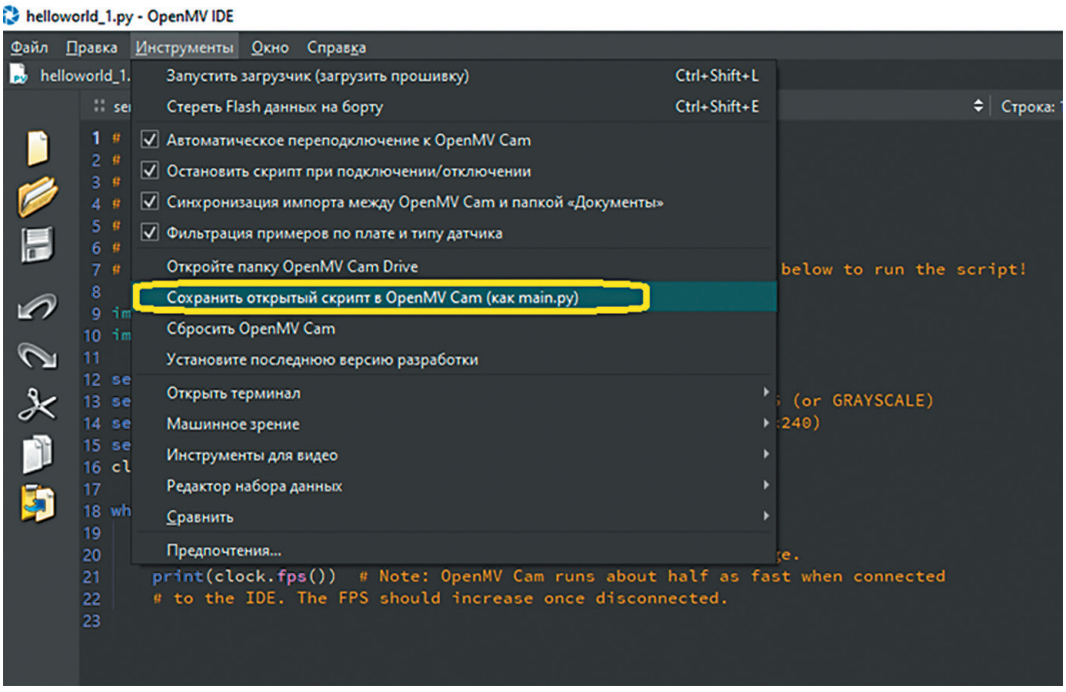


Вставьте код программы, пример которой приведен ниже. В качестве значений thresholds используйте свои полученные значения.

```
import sensor, utime
from pyb import UART

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time=1000)

# Порт общения с Arduino
uart = UART(1, 19200, timeout_char=200)

# Пороговое значение цвета шарика (вставьте свои значения, полученные при настройке)
thresholds = [(55, 98, -21, 14, 34, 70)]

while True:
    img = sensor.snapshot()
```

```
# Поиск шарика на изображении
blobs = img.find_blobs(thresholds,
                       pixels_threshold=150, area_threshold=150)
                       #, roi = [0, 0, 320, 200])

# Если шарик найден
if len(blobs) > 0:
    blob = blobs[0]
    img.draw_rectangle(blob.rect())
    img.draw_cross(blob.cx(), blob.cy())

    # Находим его смещение относительно центра по x
    # и его смещение относительно низа экрана по y
    coordinates = [blob.cx() - (img.width() / 2), img.height() - blob.cy()]

    print(f»x = {coordinates[0]}, y = {coordinates[1]}»)

    # Отправка координат по UART на Arduino
    uart.write(«{} {}\n».format(coordinates[0], coordinates[1]))
    utime.sleep_ms(100)
```

Для того чтобы обеспечить запуск программы в автономном режиме, необходимо сохранить ее на SD-карту модуля технического зрения. Для этого в OpenMV IDE необходимо выбрать «Инструменты» – «Сохранить открытый скрипт в OpenMV Cam (как main.py)».



В этом случае программа будет запускаться сразу после подачи напряжения на модуль технического зрения.

Теперь необходимо подключиться к Arduino одним из способов:

• Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

• Или через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

Напишите свой код или используйте приведенный ниже.

Загрузите его в микроконтроллер Arduino.

Приведенный пример кода устанавливает камеру с помощью сервоприводов в начальное положение. Затем ожидает координаты шарика от камеры и организует движение к нему с помощью управления моторами.

```
#include <SCServo.h> // Сервоприводы

#define CAMERA_SERIAL Serial2  // Порт для общения с камерой
#define CAMERA_BAUDRATE 19200  // Баудрейт для общения с камерой

SMS_STS sms_sts; // Объект сервопривода
byte ID[2] = {1, 2};
s16 Position[2] = {420, 20};
u16 Speed[2] = {200, 200};
byte ACC[2] = {10, 10};

int IN1 = 3; // Мотор 1
int IN2 = 4;
int ENA = 2;

int IN3 = 5; // Мотор 2
int IN4 = 6;
int ENB = 7;

int IN1_1 = 9; // Мотор 3
int IN1_2 = 10;
int ENA_1 = 8;

int IN1_3 = 11; // Мотор 4
int IN1_4 = 12;
int ENB_1 = 13;

const uint16_t GO_TIME   = 600;    // длительность движения
const uint16_t STOP_TIME = 200;    // длительность паузы
```

```
void setup() {
  // Ставим пины моторов на выход
  pinMode (IN1, OUTPUT);
  pinMode (IN2, OUTPUT);
  pinMode (ENA, OUTPUT);

  pinMode (IN3, OUTPUT);
  pinMode (IN4, OUTPUT);
  pinMode (ENB, OUTPUT);

  pinMode (IN1_1, OUTPUT);
  pinMode (IN1_2, OUTPUT);
  pinMode (ENA_1, OUTPUT);

  pinMode (IN1_3, OUTPUT);
  pinMode (IN1_4, OUTPUT);
  pinMode (ENB_1, OUTPUT);

  Serial.begin(9600); // Вывод в монитор порта
  Serial1.begin(1000000); // Общение с сервоприводами
  sms_sts.pSerial = &Serial1;
  sms_sts.SyncWritePosEx(ID, 2, Position, Speed, ACC); // Начальное положение
сервоприводов
  delay(200);

  Serial.println("Init");
}

void loop() {
  float xBall = NAN; float yBall = NAN; // Переменные для координат шарика

  // Считывание данных с камеры
  if (getDataCamera(&xBall, &yBall)) {
    followBall(&xBall, &yBall); // Следование за шариком
    moveStop(); // Остановка движения
    Serial.println();
  }
}

// Считывание данных с камеры
// В данную функцию мы подаем указатель на переменные xBall, yBall, т.е. мы
говорим, в какой
// ячейке памяти компьютера лежит нужные значения. Т.к. функция знает адрес
переменной в памяти,
// она может ее перезаписывать.
// *xBall - получение значения переменной по адресу
// &xBall - получение адреса переменной xBall
```

```
bool getDataCamera(float* x, float* y) {
  CAMERA_SERIAL.begin(CAMERA_BAUDRATE);
  delay(120);


  // Если камера что-то отправила
  if (CAMERA_SERIAL.available() > 0) {
    String dataCamera = CAMERA_SERIAL.readStringUntil('\n'); // Читаем строку до ее
конца
    dataCamera.trim();
    CAMERA_SERIAL.end();

    String camParameters[2];  // Массив для координат (x, y)
    byte index = 0;           // Индекс текущего элемента массива camParameters

    // Получение чисел из строки
    if (dataCamera.length() > 0) {
      while (dataCamera.length() > 0 && index < 2) {
        int spaceIndex = dataCamera.indexOf(' ');  // Нахождение первого с начала
строки пробела
        // Если пробел не был найден
        if (spaceIndex == -1) {
          camParameters[index++] = dataCamera;  // Добавляем оставшуюся строку в
массив
          break;
        } else {
          camParameters[index++] = dataCamera.substring(0, spaceIndex);  //
Добавляем в массив все символы от начала строки до пробела (число)
          dataCamera = dataCamera.substring(spaceIndex + 1);              // Убираем
обработанную часть строки
          dataCamera.trim();
        }
      }

      // Преобразуем x, y в числа
      if (index == 2) {
        *x = camParameters[0].toFloat();
        *y = camParameters[1].toFloat();
        Serial.println("x = " + String(*x));
        Serial.println("y = " + String(*y));

        return true;
      } else { CAMERA_SERIAL.end(); return false; }  // if (index == 2)
    } else { CAMERA_SERIAL.end(); return false; }  // if (dataCamera.length() > 0)
  } else { CAMERA_SERIAL.end(); return false; } // if (Serial2.available() > 0)
}
```

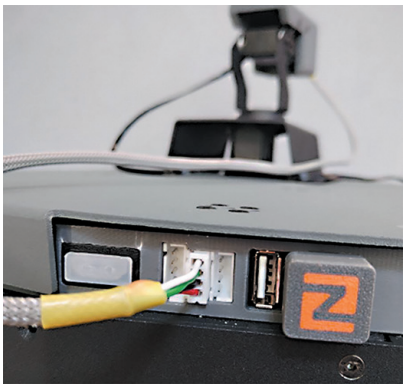
```
// Следование за шариком
void followBall(float* x, float* y) {
  // Если значения xBall, yBall некорректны
  if (isnan(*x) || isnan(*y)) { Serial.println("x или y == NAN"); return; }

  // Задание определенного движения в зависимости от положения шарика
  if (*y > 40) {
    if (*x > 40) {
        moveDiagonalRight();
    } else if (*x < -40) {
        moveDiagonalLeft();
    } else {
        moveForward();
    }
  } else {
    if (*x > 40) {
      moveSideRight();
    }
    else if (*x < -40) {
      moveSideLeft();
    }
    // Если шарик близко по x и y - не двигаемся
  }
}

// Движение по диагонали вправо
void moveDiagonalRight() {
  Serial.println("moveDiagonalRight");

  // 2 - вперед
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite(ENB, 200);

  // 4 - вперед
  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  analogWrite(ENB_1, 200);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis(); }
}

// Движение по диагонали влево
void moveDiagonalLeft() {
  Serial.println("moveDiagonalLeft");
```

```
  // 1 - вперед
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  analogWrite(ENA, 200);

  // 3 - вперед
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  analogWrite(ENA_1, 200);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis(); }
}

// Движение вперед
void moveForward() {
  Serial.println("moveForward");

  // 1 - вперед
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  analogWrite(ENA, 200);

  // 2 - вперед
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite(ENB, 200);

  // 3 - вперед
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  analogWrite(ENA_1, 200);

  // 4 - вперед
  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  analogWrite(ENB_1, 200);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis(); }
}

// Движение вправо
void moveSideRight() {
  Serial.println("moveSideRight");
```

```
  // 1 - назад
  digitalWrite (IN1, LOW);
  digitalWrite (IN2, HIGH);
  analogWrite(ENA, 100);

  // 2 - вперед
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite(ENB, 100);

  // 3 - назад
  digitalWrite (IN1_1, LOW);
  digitalWrite (IN1_2, HIGH);
  analogWrite(ENA_1, 100);

  // 4 - вперед
  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  analogWrite(ENB_1, 100);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis(); }
}

// Движение влево
void moveSideLeft() {
  Serial.println("moveSideLeft");

  // 1 - вперед
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  analogWrite(ENA, 100);

  // 2 - назад
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  analogWrite(ENB, 100);

  // 3 - вперед
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  analogWrite(ENA_1, 100);

  // 4 - назад
  digitalWrite (IN1_3, LOW);
  digitalWrite (IN1_4, HIGH);
  analogWrite(ENB_1, 100);
```

```
  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis(); }
}

// Остановка движения
void moveStop() {
  Serial.println("moveStop");

  // 1 - скорость 0
  analogWrite(ENA, 0);

  // 2 - скорость 0
  analogWrite(ENB, 0);

  // 3 - скорость 0
  analogWrite(ENA_1, 0);

  // 4 - скорость 0
  analogWrite(ENB_1, 0);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < STOP_TIME) { currentMillis = millis(); }
}
```

После загрузки кода программы необходимо подключить входящим в комплект поставки четырехжильным кабелем модуль технического зрения к используемому программой Serial 2 контроллера Arduino Mega 2560. И не забудьте подключить Arduino к Raspberry Pi с помощью переходника.



КУРСОВАЯ КАМЕРА

В качестве курсовой камеры в Optima-Drive установлена камера Raspberry Pi Camera v2.

Камера оснащена восьмимегапиксельным сенсором Sony IMX219 Exmor. Он позволяет захватывать, записывать и транслировать видео в форматах 1080p, 720p и VGA. Максимальное разрешение для фотографий достигает 3280×2464 пикселей.

В отличие от USB-камер, MIPI-камеры (Mobile Industry Processor Interface) подключаются через разъём CSI 2 (Camera Serial Interface-2) напрямую к видеочипу VideoCore и экономят системные ресурсы Raspberry Pi. При этом USB-порты остаются свободными для другой периферии.

Характеристики камеры:

- Тип сенсора: Sony IMX 219 PQ CMOS, ¼ дюйма
- Максимальное разрешение: 8 Мп (3280×2464)
- Поддерживаемые видеоформаты: 1080p (30fps), 720p (60fps), 640×480p (90fps)
- Эквивалентное фокусное расстояние: 33 мм
- Светосила объектива: f/2
- Вес: 3 г

Raspberry Pi OS поддерживает два глобальных API для работы с камерами:

- libcamera – современный стек библиотек, который активно развивается.
- raspicam – устаревший стек библиотек, который потихоньку угасает. В последнее время получил название Legacy Camera.

Raspberry Pi не может одновременно задействовать оба стека libcamera и raspicam. По умолчанию включён libcamera. Если вы перейдёте на raspicam, то libcamera отключится (и наоборот).

Стек libcamera – API через набор утилит libcamera-apps, среди которых:

- libcamera-hello – для отображения видоискателя;
- libcamera-still – для захвата фото;
- libcamera-vid – для записи видео;
- и другие модули.

На основе libcamera также написан программный модуль Picamera2 для Python.

Более подробно про стек libcamera для работы с камерой читайте в документации (https://www.raspberrypi.com/documentation/computers/camera_software.html#libcamera-and-libcamera-apps).

Примеры работы через Python

Перейдём к экспериментам с камерой через интегрированную среду разработки Python. Для работы со стеком libcamera в Python предусмотрена библиотека Picamera. Примеры можно запускать через встроенный терминал или графическую оболочку Thonny Python IDE.

Захват фото
Попробуем сделать снимок с камеры и сохранить полученную фотографию.

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

# инициализируем камеру и ссылку на захват необработанной камеры
camera = PiCamera()
rawCapture = PiRGBArray(camera)

# выделяем время для нагрева камеры
time.sleep(0.1)

# захватываем изображение через камеру
camera.capture(rawCapture, format=»bgr»)
image = rawCapture.array

# выводим полученное изображение на экран
cv2.imshow(«Image»,  image)
cv2.waitKey(0)
```

Запись видео
Используйте модуль в режиме видеокамеры – снимите ролик и сохраните его на рабочий стол.

```
import cv2

cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*’XVID’)
out = cv2.VideoWriter(‘output.avi’, fourcc, 20.0, (800,680))
print(cap.isOpened())
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        out.write(frame)

        cv2.imshow(‘frame’, frame)

        if cv2.waitKey(1) & 0xFF == ord(‘q’):
          break
```

```
        else:
            break

    cap.release()
    out.release()
    cv2.destroyAllWindows()
```

После минутного ожидания на рабочем столе появится видеофайл с камеры. Для просмотра ролика используйте встроенный плеер.

ЛАБОРАТОРНАЯ РАБОТА

№ 13 РАБОТА КАМЕРЫ В РЕЖИМЕ WEB-ТРАНСЛЯЦИИ

Цель работы: Научиться работать с камерой в режиме web-трансляции.

Задачи: Обеспечить просмотр on-line видео с курсовой камеры используя web-browser на удаленном компьютере.

Порядок выполнения работы

Для начала работы включите питание платформы Optima-Drive.

Дождитесь загрузки операционной системы, затем подключитесь к платформе, используя VNC-viewer.

Ниже приведен пример кода программы для просмотра видео с курсовой камеры используя web-browser. Изучите его, создайте свой код или используйте приведенный для демонстрации режима видеотрансляции с курсовой камеры.

```
import io
import picamera
import logging
import socketserver
from threading import Condition
from http import server

PAGE="""\
<html>
<head>
<title>picamera MJPEG streaming demo</title>
</head>
<body>
<h1>PiCamera MJPEG Streaming Demo</h1>
<img src="stream.mjpg" width="640" height="480" />
</body>
</html>
"""

class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and notify all
            # clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
```

```
            self.buffer.seek(0)
        return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-replace;
boundary=FRAME')
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type', 'image/jpeg')
                    self.send_header('Content-Length', len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as camera:
    output = StreamingOutput()
    camera.start_recording(output, format='mjpeg')
```

```
try:
    address = ('', 8000)
    server = StreamingServer(address, StreamingHandler)
    server.serve_forever()
finally:
    camera.stop_recording()
```

Откройте на вашем компьютере, подключенном к той же WI-Fi сети, что и Optima-Drive, любой web-browser, введите в строке IP-адрес, который получила Raspberry Pi в вашей сети, например

http://192.168.51.87:8000/

На странице web-browsera вы увидите видео, транслируемое с курсовой камеры.

СЕРВОПРИВОДЫ УПРАВЛЕНИЯ PAN-TILT КАМЕРОЙ

В качестве приводов, управляющих параметрами движения Pan-Tit камеры, используются цифровые сервоприводы серии STS компании Feetech STS3235.

Это интеллектуальные сервоприводы с последовательной шиной управления, напряжением питания 12 В. Выполнены в корпусе из алюминиевого сплава и имеют металлический редуктор. Управляются посредством платы управления TTL, разработанной компанией Feetech. Оборудованы 12-битным высокоточным магнитным энкодером. Крутящий момент составляет 30 кг/см для STS3235. Угол поворота 0–360 градусов, поддерживают режим работы двигателя непрерывного вращения («режим колеса»).

Обеспечивает обратную связь по параметрам:

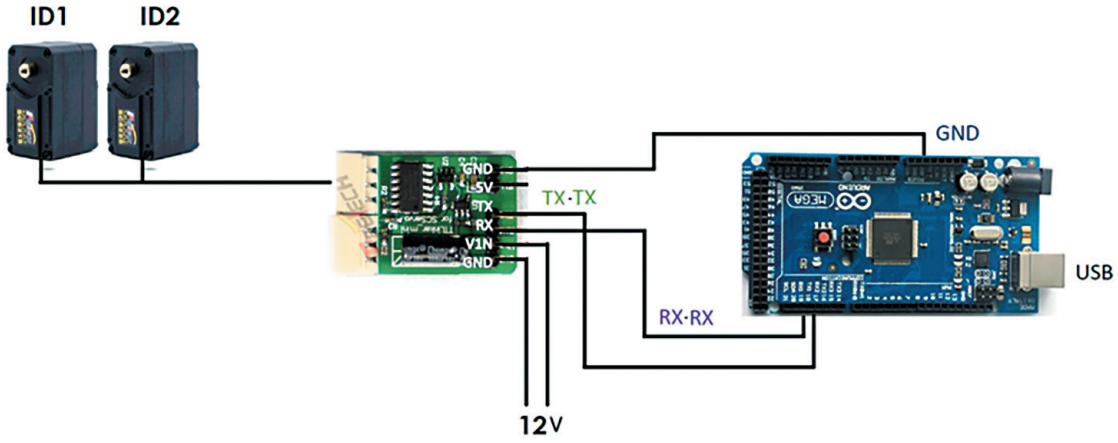
- положению,
- скорости,
- напряжению,
- температуре,
- нагрузке для обеспечения защиты от перегрузки.

Технические характеристики сервоприводов Feetech STS3235

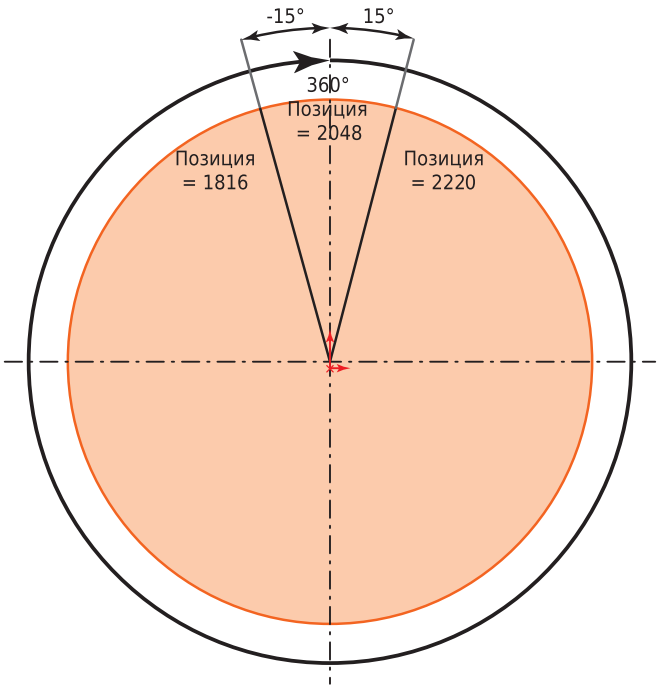
Модель	STS3235
Название продукта	Сервопривод с последовательной шиной управления, 12 В, 30 кг·см
Диапазон рабочих температур	-20~60 ℃
Размер	А: 45,22 мм Б: 24,72 мм С: 35 мм
Вес	70,5±1 г
Тип шестерни	Сталь
Предельный угол	Безлимитный
Материал корпуса	Алюминий
Соединительный провод	15 СМ
Диапазон рабочего напряжения	6-12 В
Скорость без нагрузки	0,222 с/60° при 12 В
Рабочий ток (без нагрузки)	190 мА при 12 В
Пиковый крутящий момент	30 кг.см при 12 В
Номинальный крутящий момент	10 кг·см при 12 В
Ток срыва	2,7 А при 12 В
Тип протокола	Полудуплексная асинхронная последовательная связь
Диапазон идентификаторов ID (количество сервоприводов на одной шине)	0-253
Скорость связи	38 400 бит/с~1 Мбит/с
Алгоритм управления	ПИД-регулятор
Нейтральная позиция	180°(2048)

Угол поворота	360° (0~4096)
Обратная связь	Нагрузка, положение, скорость, входное напряжение, температура
Разрешение [град/импульс]	0,087° (360°/4096)

В связи с тем, что для управления сервоприводами используется единая шина, каждый сервопривод должен иметь свой уникальный адрес – ID. В механизме Pan-Tilt нижний сервопривод, вращающий камеру по горизонтали, имеет ID=1, сервопривод, вращающий по вертикали – ID=2. Сервоприводы через преобразователь интерфейсов TTLinker подключены к Serial1 порту Arduino Mega 2560.

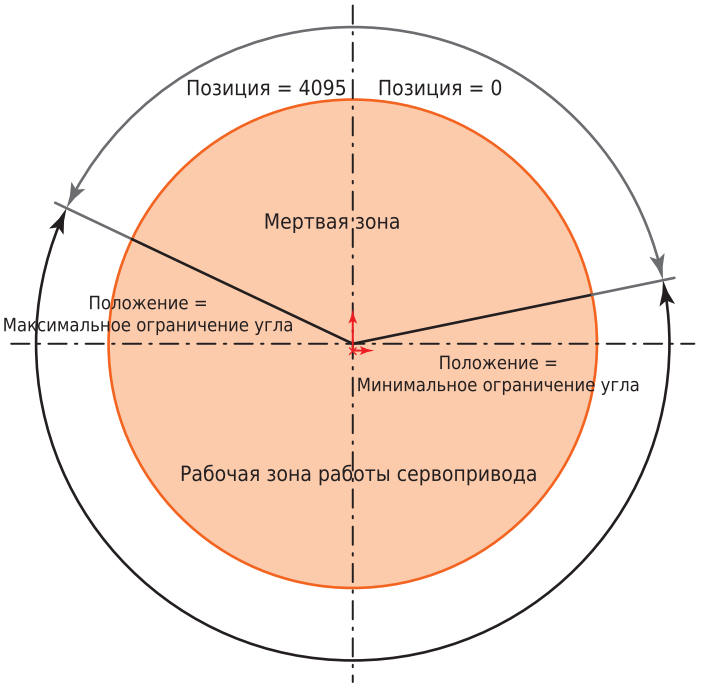


Угол поворота вала сервопривода задается количеством шагов (импульсов управления) Position. Текущее положение вала сервопривода однозначно определяется с помощью встроенного в сервопривод энкодера - это значения от 0 до 4096. Таким образом, полный поворот вала сервопривод совершит при получении от контроллера начального значения положения вала Position1 = 0 и конечного значения Position2 = 4096.



В среднее положение вал сервопривода устанавливается при значении Position = 2048. Из таблицы характеристик сервопривода известно разрешение [град/импульс] = 0,087° (360°/4096). Таким образом, получается примерно 11,5 импульса (или значение Position = 11,5) на 1 градус. Значит, при текущем значении положения вала сервопривода 2048 для поворота вала сервопривода, например, на 15 градусов, необходимо задать приращение на 11,5 x 15 = 172 целых шага, или 2048 + 172 = 2220 (или 2048 – 172 = 1876). При этом вал сервопривода совершит поворот в 15 градусов, от значения 2048 до 2220 или 1876, если нужно вращать вал в другую сторону.

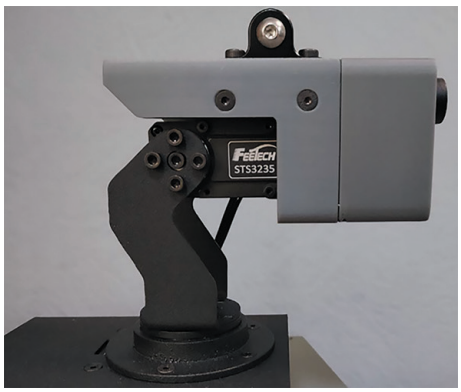
При использовании сервопривода в реальных устройствах использовать угол поворота 360 градусов требуется не всегда. При движении суставов робота на угол, выходящий за рамки рабочего пространства, существует опасность упора сустава в следующий сустав. Это приведет к перегрузке сервопривода. Для предотвращения возможности выхода сустава робота-манипулятора за рамки разрешенных углов поворота необходимо использовать настраиваемые «мертвые зоны» сервопривода. Это зоны - значения Position от 0 до, например, 200 в начале хода, и от 3000 до 4095 в конце. Использование сервоприводом значений Position в этих диапазонах являются запрещенными – dead zone. Эти значения задаются программно путем записи в соответствующие регистры флеш-пямяти сервопривода требуемых значений (Minimum Angle Limitation и Maximum Angle Limitation).



Для обеспечения необходимых максимальных и минимальных углов поворота сервоприводов с помощью приложения FT SCServo Debug V1.9.8.3 производитель прописывает минимальное значение Pos – minPos, и максимальное значение – maxPos.

ID1	Min Position = 20	Среднее положение = 2048	Max Position = 4090
ID2	MinPosition = 1600	Среднее положение = 2048	Max Position = 2600

Для управления сервоприводами используются три типа команд – команда для прямой записи значения требуемого угла поворота одного сервопривода WritePos, команда управления одновременно группой сервоприводов SyncWrite, и RegWrite – предварительно записываются требуемые значения угла поворота и скорости, затем выполняется команда, инициализирующая выполнение – Action.



ЛАБОРАТОРНАЯ РАБОТА

№ 14 УПРАВЛЕНИЕ СЕРВОПРИВОДОМ

Цель работы: Научиться управлять движениями Pan-Tilt механизма с помощью сервоприводов.

Задачи: Необходимо обеспечить поворот камеры с помощью сервопривода ID1, который поворачивает камеру по горизонтали.

Порядок выполнения работы

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

4. Установите в Arduino IDE библиотеку SCServo.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

```
#include <SCServo.h>

SMS_STS servo;

void setup()
{
  Serial1.begin(1000000);
  servo.pSerial = &Serial1;
  delay(1000);
}

void loop()
{
  servo.WritePosEx(1, 1500, 0, 250);
  delay(3500);
  servo.WritePosEx(1, 2500, 0, 250);
  delay(3500);
}
```

ЛАБОРАТОРНАЯ РАБОТА

№ 15 ПРИМЕР КОМПЛЕКСНОГО ИСПОЛЬЗОВАНИЯ СИСТЕМ МОБИЛЬНОЙ ПЛАТФОРМЫ

Цель работы: Продемонстрировать комплексное использование различных систем мобильной платформы Optima-Drive.

Задачи: Платформа должна двигаться прямо, при обнаружении впереди себя с помощью ультразвуковых сенсоров 5, 6 и 7 препятствия, осуществить поворот вправо. В процессе движения должны быть включены светодиодные ленты, должны двигаться оба сервопривода Pan-Tilt механизма, совершая повороты от минимальных разрешенных углов до максимальных.

Порядок выполнения работы

1. Подключиться к Arduino одним из способов:

Локально – снять заглушку-переходник с USB разъемов Raspberry Pi и Arduino и подключиться к контроллеру с помощью кабеля USB.

Через соединение с Raspberry Pi – аналогично методике подключения, описанной в лабораторной работе № 4.

2. Изучите код программы.

3. Напишите свой код или используйте приведенный ниже.

4. Установите в Arduino IDE библиотеку SGBotic_I2CPing.h (прилагается на электронном носителе).

5. Загрузите его в микроконтроллер Arduino.

```
#include <SGBotic_I2CPing.h>
#include <SCServo.h>
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif

SGBotic_I2CPing sonar; // УЗ датчики
uint8_t bus;           // Шина мультиплексера

// Параметры сервоприводов
SMS_STS sms_sts;
byte ID[2] = {1, 2};
u16 Position[2];
u16 Speed[2] = {1000, 1000};
u8 ACC[2] = {10, 10};

#define PIN1        44 // Указываем номер управляющего пина ленты №1
#define PIN2        46 // Указываем номер управляющего пина ленты №2
#define NUMPIXELS 18    // Указываем количество светодиодов в ленте
Adafruit_NeoPixel pixels(NUMPIXELS, PIN1, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel pixels1(NUMPIXELS, PIN2, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 90 // Указываем паузу
```

```
// Определение пинов  для каждого мотора
int IN1 = 3; int IN2 = 4; int ENA = 2;          // Мотор 1
int IN3 = 5; int IN4 = 6; int ENB = 7;          // Мотор 2
int IN1_1 = 9; int IN1_2 = 10; int ENA_1 = 8;   // Мотор 3
int IN1_3 = 11; int IN1_4 = 12; int ENB_1 = 13; // Мотор 4

// Периоды времени
const uint16_t RGB_PERIOD   = 70;    // смена цвета одного сегмента
const uint16_t SERVO_PERIOD = 2500;  // один поворот камеры
const uint16_t SONAR_PERIOD = 50;    // опрос расстояния
const uint16_t GO_TIME      = 1500;  // длительность движения вперед
const uint16_t TURN_TIME    = 250;   // длительность поворота
const uint16_t ROTATE_TIME  = 400;   // длительность разворота на 180

// Переменные для хранения количества миллисекунд
unsigned long tRGB   = 0;
unsigned long tServo = 0;
unsigned long tSonar = 0;
unsigned long tTurn  = 0;

void setup() {
  // Устанавливаем все пины как выходы
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1_1, OUTPUT); pinMode(IN1_2, OUTPUT); pinMode(ENA_1, OUTPUT);
  pinMode(IN1_3, OUTPUT); pinMode(IN1_4, OUTPUT); pinMode(ENB_1, OUTPUT);

  #if defined(__AVR_ATtiny85__) && (F_CPU == 16000000) // Лента
    clock_prescale_set(clock_div_1);
  #endif

  pixels.begin();  // Инициализируем объект pixels
  pixels1.begin(); // Инициализируем объект pixels

  Serial.begin(9600); // Вывод в монитор порта
  Serial1.begin(1000000); // Последоватеьный порт для сервоприводов
  sms_sts.pSerial = &Serial1;

  delay(1000);
}

void loop() {
  rgb();
  servo();
  checkSonar();
  forward();
}
```

```
// Обращение к УЗ датчику по номеру на шинемультиплексера
void TCA9548A(uint8_t bus) {
  Wire.beginTransmission(0x70);
  Wire.write(1 << bus);
  Wire.endTransmission();
}

void forward() {
  Serial.println("Forward");

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA, 200);

  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB, 200);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1, 200);

digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1, 200);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < GO_TIME) { currentMillis = millis();
checkSonar(); }
  turn_right();
}

void turn_right() {
  Serial.println("Turn Right");

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA,255);
```

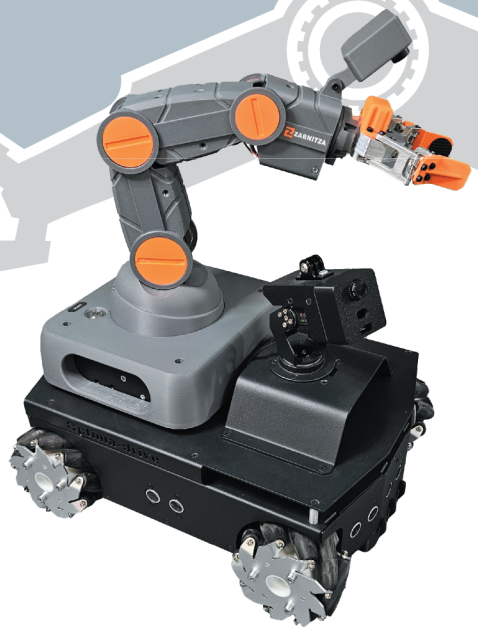
```
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB,255);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, LOW);
  digitalWrite (IN1_2, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1,255);

  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1,255);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < TURN_TIME) { currentMillis = millis(); }
}

void turn_left() {
  Serial.println("Turn Left");

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, LOW);
  digitalWrite (IN2, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA,255);


digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB,255);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, HIGH);
  digitalWrite (IN1_2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1,255);

  digitalWrite (IN1_3, LOW);
  digitalWrite (IN1_4, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1,255);
```

```
  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < TURN_TIME) { currentMillis = millis(); }
}

void rotate() {
  Serial.println("ROTATE");

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA,255);

  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB,255);

  // На пару выводов "IN" поданы разноименные сигналы, мотор готов к вращению
  digitalWrite (IN1_1, LOW);
  digitalWrite (IN1_2, HIGH);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENA_1,255);

  digitalWrite (IN1_3, HIGH);
  digitalWrite (IN1_4, LOW);
  // подаем на вывод ENB управляющий ШИМ сигнал
  analogWrite(ENB_1,255);

  unsigned long startMillis = millis();
  unsigned long currentMillis = millis();
  while (currentMillis - startMillis < ROTATE_TIME) { currentMillis = millis(); }
}

// Моргание светодиодной лентой
void rgb() {
  if (millis() - tRGB < RGB_PERIOD) return;
  tRGB = millis();

  static uint8_t phase = 0;                // 0 зел, 1 красн, 2 син
  uint32_t color;

  switch (phase) {
    case 0:  color = pixels.Color(0,250,0);  break;
    case 1:  color = pixels.Color(250,0,0);  break;
    default: color = pixels.Color(0,0,250);  break;
  }
```

```
  for (uint8_t i = 0; i < NUMPIXELS; ++i) {
    pixels .setPixelColor(i, color);
    pixels1.setPixelColor(i, color);
  }
  pixels.show();
  pixels1.show();

  phase = (phase + 1) % 3;
}

// Управление сервоприводами
void servo() {
  if (millis() - tServo < SERVO_PERIOD) return;
  tServo = millis();

  static bool lastMove = false;
  if (lastMove) {
    Position[0] = 50;
    Position[1] = 1600;
  } else {
    Position[0] = 4000;
    Position[1] = 2600;
  }
  sms_sts.SyncWritePosEx(ID, 2, Position, Speed, ACC);

  lastMove = !lastMove;
}

// Опрос УЗ датчика
void checkSonar() {
  if (millis() - tSonar < SONAR_PERIOD) return;
  tSonar = millis();

  for (int i = 5; i < 8; i++) {
    Serial.print("Check start for id = ");
    Serial.println(i);
    TCA9548A(i);

    unsigned long dist = sonar.ping_cm();

    Serial.println(dist);
    if (dist > 0 && dist < 20) {
      turn_right();
      return;
    }
  }
}
```

ZARNITZA
ROBOTICS

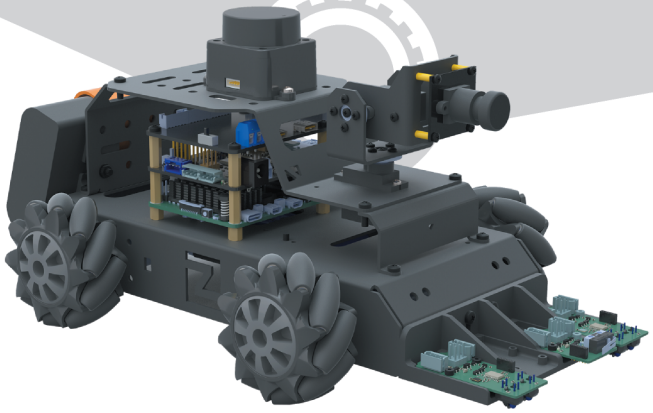
Разрабатываем и производим
высокотехнологичное учебное
оборудование для любых специальностей



УП6153

Робот-манипулятор с колёсами всенаправленного движения Optima Pro + Optima Drive

Приказ 838 Минпросвещения РФ



УП9149

Расширенный робототехнический набор для конструирования, изучения электроники и микропроцессоров и электронных систем и устройств (электронные устройства (датчики, моторы, сервоприводы) Z.Robo-6

Приказ 838 Минпросвещения РФ

УП6340

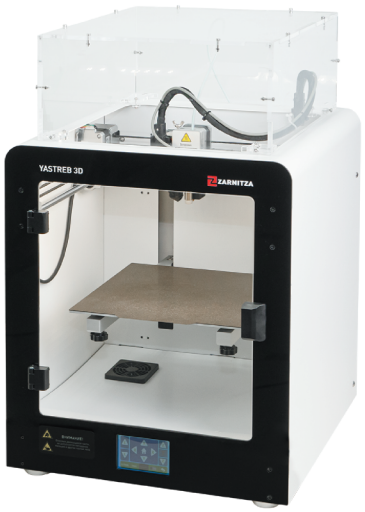
3D-сканер ручной профессиональный Yastreb 3D

Приказ 838 Минпросвещения РФ

УП6338

3D-принтер профессионального качества Zarnitsa Yastreb 3D

Приказ 838 Минпросвещения РФ



Телефон 8-800-775-37-97
www.zarnitza.ru, zakaz@zrnc.ru

РОБОТОТЕХНИКА

ZARNITZA

8 (800) 775-37-97
zakaz@zrnc.ru

RU СДЕЛАНО В РОССИИ