ТОМСКИЙ ПОЛИТЕХ

Михаил Момот

Конструктор робототехнический на базе микроконтроллера ESP32

Г. Томск 2024

УДК 007.52

ВВК 32.816

М.76

Момот М.В.

М.76 Конструктор робототехнический на базе микроконтроллера ESP32. Изд. ТПУ: Томск, 2024 – 73с.: илл.

Учебное пособие для начинающих робототехников. В книге последовательно представлен теоретический и практический материал, для сборки и программирования самоходной роботизированной платформы, оснащенной электронными датчиками. Пособие предназначено для старших школьников и содержит разделы от процесса сборки конструктора до от программирования микроконтроллера ESP32 и организации соревнований.

Содержание

Введение

Здравствуй, дорогой читатель!

Представляю учебное пособие для начинающих робототехников. Книга про мобильных роботов. Эту книгу я составлял как руководство для начинающих конструкторов, т. е. людей, которым нравится *конструировать*. А за основу взял конструирование несложных роботов на весьма популярной в настоящее время платформе Arduino. Arduino же выбрал потому, что проекты, выполненные на ее основе, весьма простые и функциональные. Платформа Arduino открытая, а это значит, что изготавливать дополнительные модули для нее может любой человек или организация, то же относится и к программам.

Изучив книгу, вы получите знания по конструированию и программированию мобильных роботов на платформе Arduino. Поймете принцип действия датчиков, при помощи которых роботы следят за внешним миром. Будете собирать своих оригинальных роботов и удаленно управлять их работой.

Книга разбита на уроки.

- *Первый урок* посвящается изучению *микроконтроллера ESP32* его возможностям и
- *Второй урок* посвящен посвящен установке и настройке среды Arduino IDE для использования микроконтроллера ESP32.
- *Третий урок* вводит процесс программирования в среде Arduino IDE и подключению различных электронных датчиков.
- *В четвертом уроке* подробно описаны процессы, происходящие в самобалансирующих роботах.

Книга поможет вам разобраться в работе основных датчиков: ультразвукового датчика расстояния, детектора препятствия, датчика цвета (черный или белый), электронного гироскопа, энкодеров — чтобы вы смогли обеспечить своим роботам требуемый функционал. В книге также приведены советы по самостоятельному изготовлению некоторых датчиков.

Последовательность сборки конструктора подробно описана в демонстрационных видеоматериалах, которые можно открыть через QR код.

В электронном архиве, сопровождающем книгу, содержатся все программные модули, которые приведены в книге, и даже некоторые не приведенные или приведенные не полностью. Кроме них в архиве содержатся файлы для печати деталей роботов из книги самостоятельно на 3d-принтере.

Электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «Томского политехнического университета» или со страницы книги на сайте **www.tpu.ru**. В электронный архив включена

библиотека **mobrob3.zip** её требуется стандартным путем установить в среду Arduino IDE, после чего файлы библиотеки будут доступны для работы. Я уверен, что изучение книги и конструирование роботов будет вам не только полезно с точки зрения получения новых знаний и умений, но и станет увлекательным и интересным занятием.

Желаю успехов, Михаил Момот

Урок 1 Функциональная схема ESP32 и распиновка

Рассмотрим функциональную схему микроконтроллера ESP32 (рис. 1.7). Незаметно в левом верхнем углу притаился блок Радио, ESP32 можно использовать в качестве радиоприемника. Блок управления питанием содержит энергоэффективный сопроцессор, управляющий «спящим режимом». Блок периферии содержит IR интерфейс, температурный сенсор, сенсоры касания. Конвертор digital-to-analog применяется для воспроизведения звука.

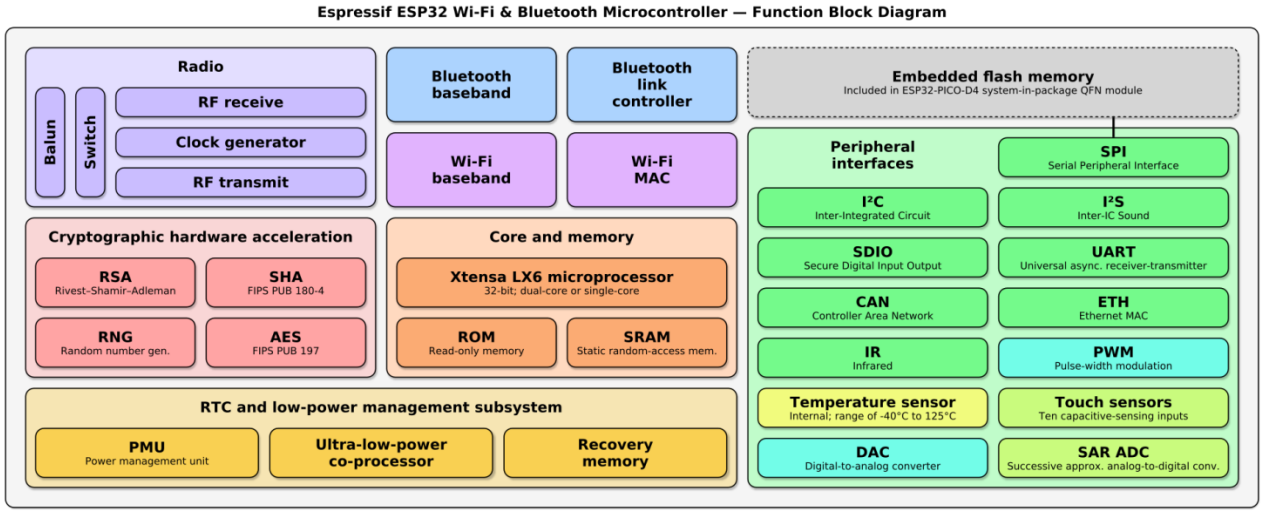


Рисунок 1.1. Модуль ESP32 на плате для макетировния (Wemos Lolin)

На рис. 1.8 приведена типовая распиновка модуля ESP32 на плате. Контакты GPIO, на которых возможна генерация ШИМ/PWM, обозначены знаком волны ~ , ADC – это GPIO с возможностью аналогового ввода, VSPI и HSPI – интерфейсы для подключения устройств, поддерживающих данные протоколы обмена, SDL и SDA – I2C интерфейс, TX0 и RX0 – интерфейс UART0. Еще раз отметим, что некоторые GPIO для стандартных интерфейсов, закрепленные по умолчанию за одними номерами GPIO, можно изменять в программе.

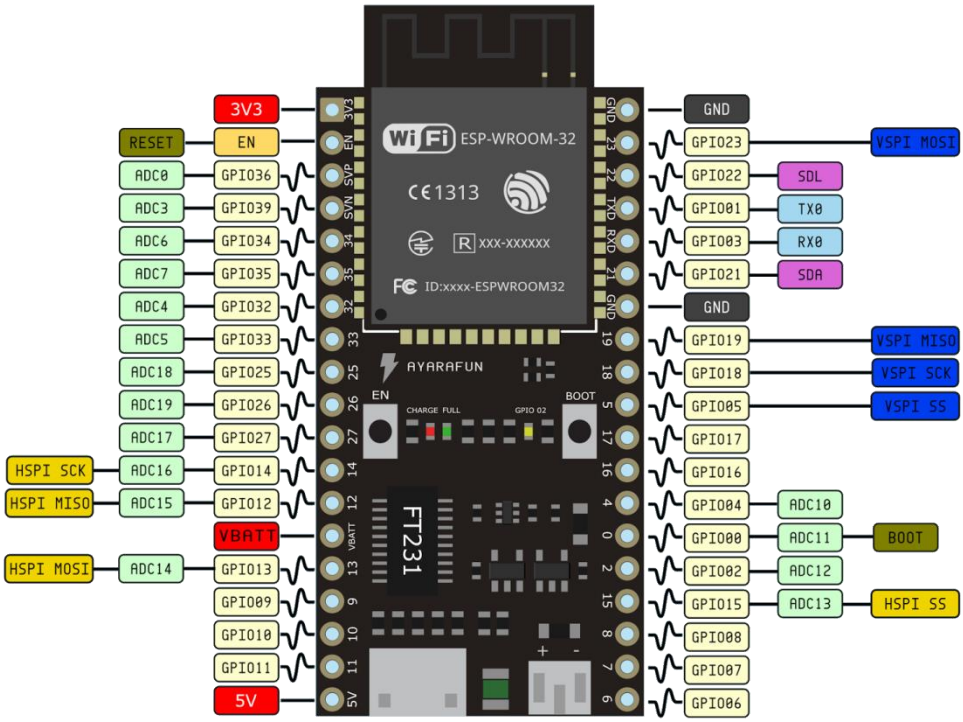


Рисунок 1.2. Распиновка ESP32 (ADC – аналоговый ввод, GPIO-стандартные контакты ввода/вывода)

Среды и средства программированияESP32

Arduino IDE

Программный код управления проектами, представленными в данной книге, основан на языке программирования Wiring (*упрощенная и оптимизированная версия C++ для работы с портами ввода/вывода*), более того само программирование осуществляется в среде Arduino IDE, знакомой большинству мейкеров (творческих *людей, умеющие воплощать свои идеи*) и описанной во многих изданиях.

Однако серия контроллеров ESP32 достаточно универсальна и допускает использование иных сред и средств программирования. Кратко рассмотрим основные из них.

Урок 2. Настройка среды программирования Arduino IDE для работы с ESP32

Arduino IDE – открытая среда программирования, поэтому по мере появления нового железа быстро появляется и новый софт для него от независимых разработчиков. Вот и для ESP32 инженеры-программисты из Espressif Systems реализовали поддержку самостоятельно и разработали инструкции по добавлению её в среду Arduino. Все довольно просто, но внимание никогда лишним не будет, рассмотрим все этапы настройки.

Установка поддержки в Arduino IDE контроллеров ESP32

Установка среды Arduino IDE

Подготовим программное обеспечение. Идем на сайт https://www.arduino.cc/en/Main/Software, выбираем последнюю версию Arduino IDE, соответствующую вашей операционной системе (Windows, Linux или MAC OS), скачиваем и устанавливаем стандартными средствами ОС.

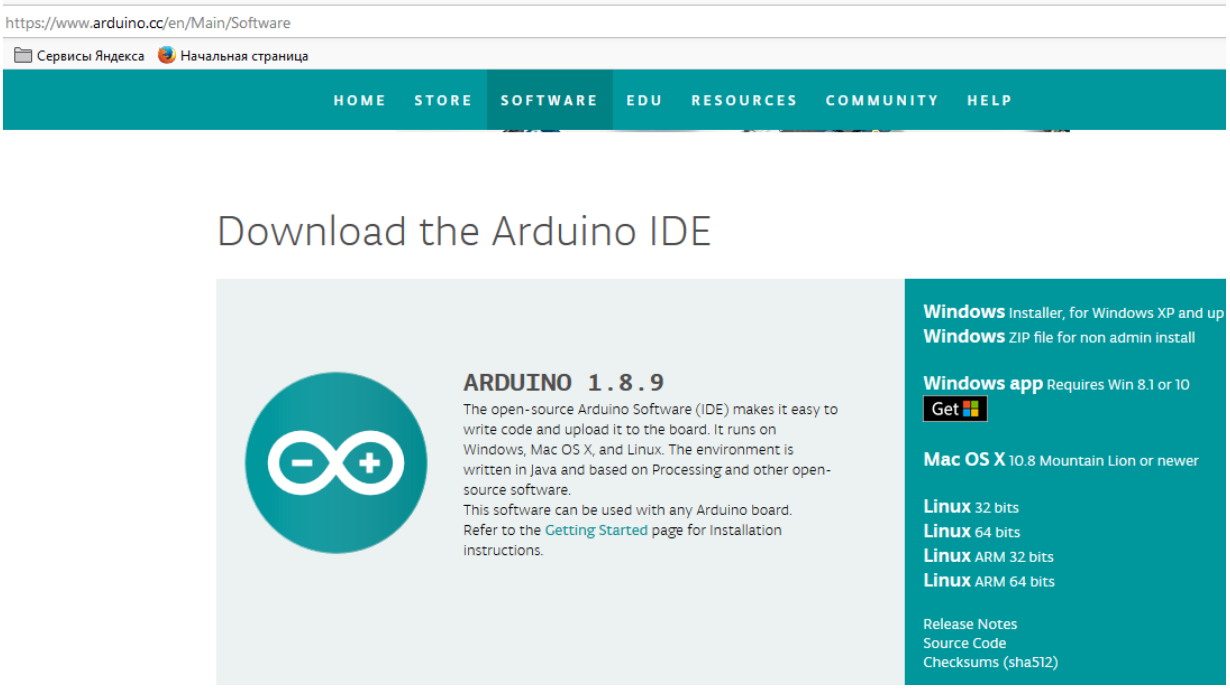


Рис.2.1. Страница загрузки Arduino IDE

Для операционной системы Ubuntu Arduino IDE можно установить, выбрав данный пакет из репозитория в самой ОС, для этого на данной странице ничего скачивать не нужно.

Дальнейшие действия производятся внутри оболочки Arduino IDE и не зависят от типа ОС.

Подключение репозитория

Запускаем среду Arduino IDE (рис.2.2) и выбираем параметр «Файл» -> «Настройки».

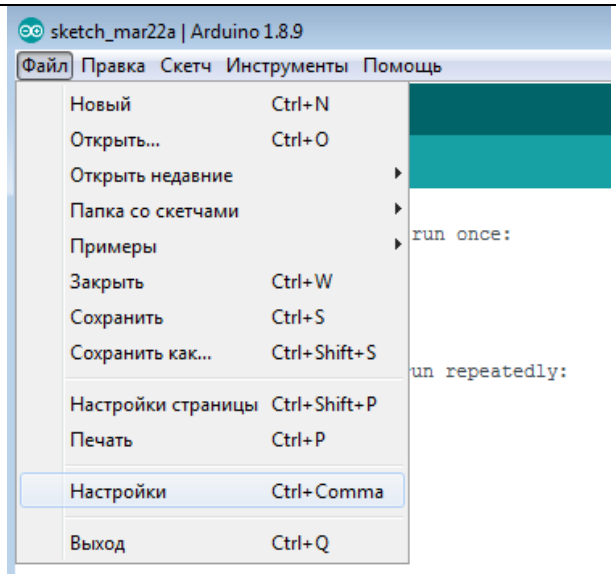


Рис.2.2. Выбор меню «Настройки»

В настройках (рис.2.3) нас интересует параметр «Дополнительные ссылки для Менеджера плат». Нажатием кнопки справа попадаем в окно ввода ссылок (рис.2.4). Добавляем ссылку на ESP32 https://dl.espressif.com/dl/package_esp32_index.json , эта ссылка подключает дополнительный репозиторий, созданный программистами Espressif, и содержащий основные компоненты для работы с ESP32 в среде Arduino IDE.

Закрываем окно кнопкой «ОК» (рис.2.4) , проверяем наличие введенной строки в «Дополнительных ссылках…» (рис.2.5) и подтверждаем внесенные изменения, закрыв окно «Настройки» по «ОК».

Установка платы ESP32 из Менеджера плат

Установим библиотеки и компилятор для ESP32, для чего переходим в меню «Инструменты», подпункт «Плата:». В открывшемся меню выбора плат выбираем «Менеджер плат…» (рис.2.6). В строке поиска Менеджера плат вводим “ESP32”. Если предварительные настройки были сделаны правильно, получаем в окне ссылку на скачивание модулей поддержки ESP32 (рис.2.7). Устанавливаем поддержку, ждем завершения процесса скачивания недостающих файлов с сайта компании Espressif.

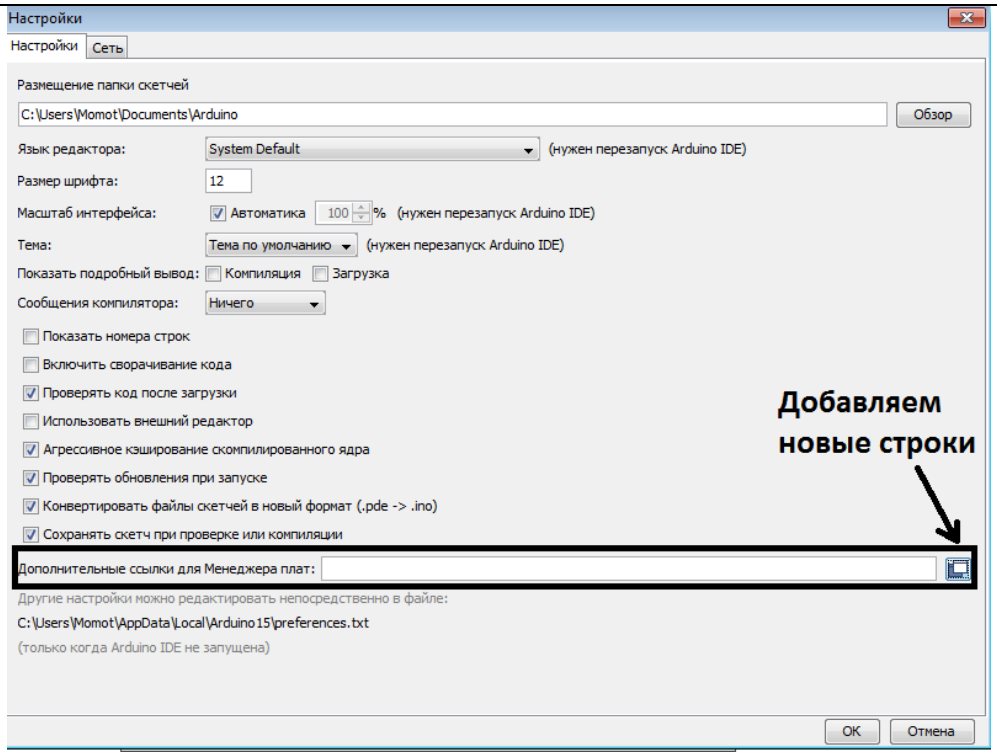


Рис.2.3. Окно настроек Arduino IDE

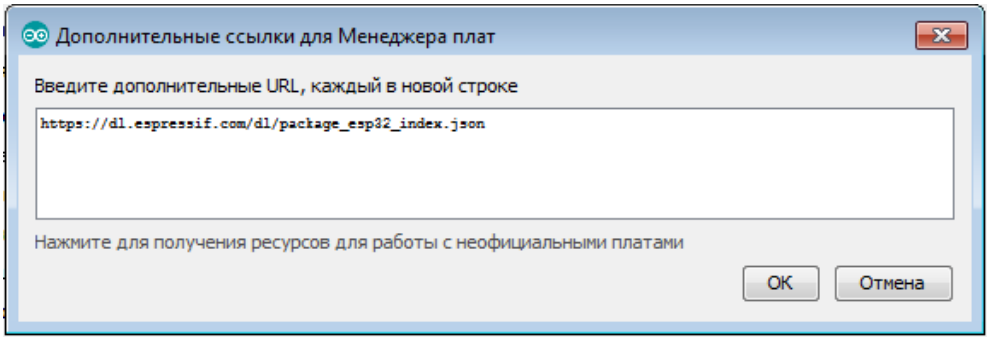


Рис.2.4. Окно ввода дополнительных ссылок для Менеджера плат

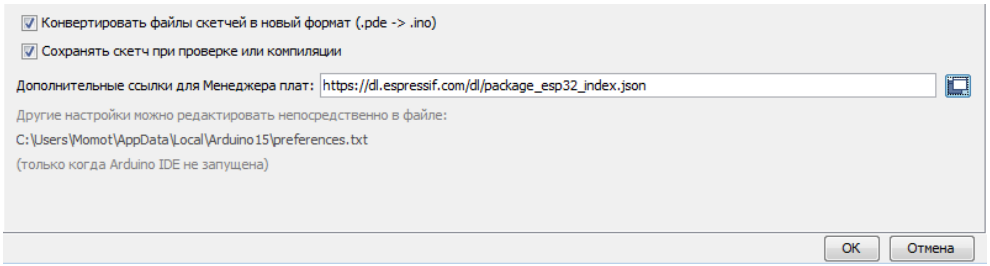


Рис.2.5. Окно настроек Arduino IDE (ссылка добавлена)

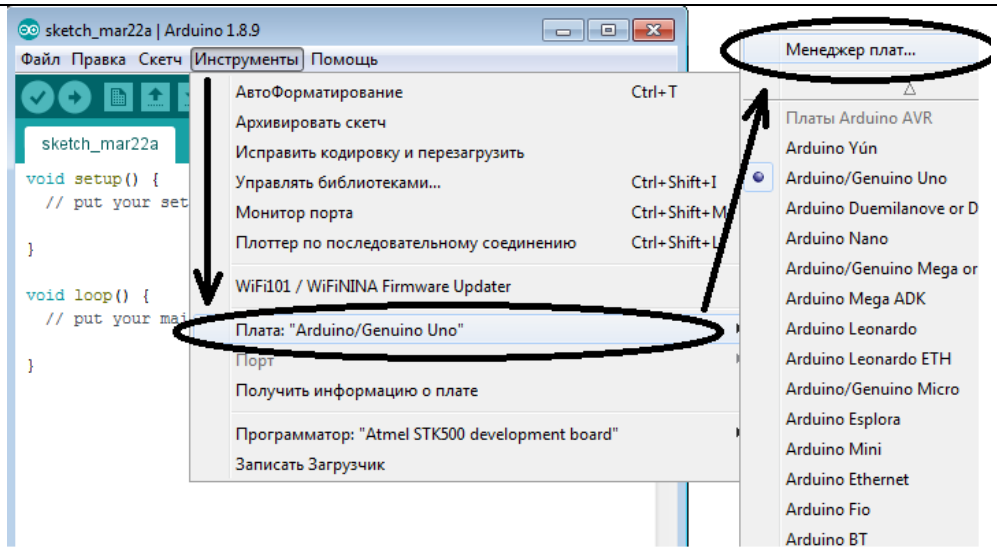


Рис.2.6. Открываем Менеджер плат

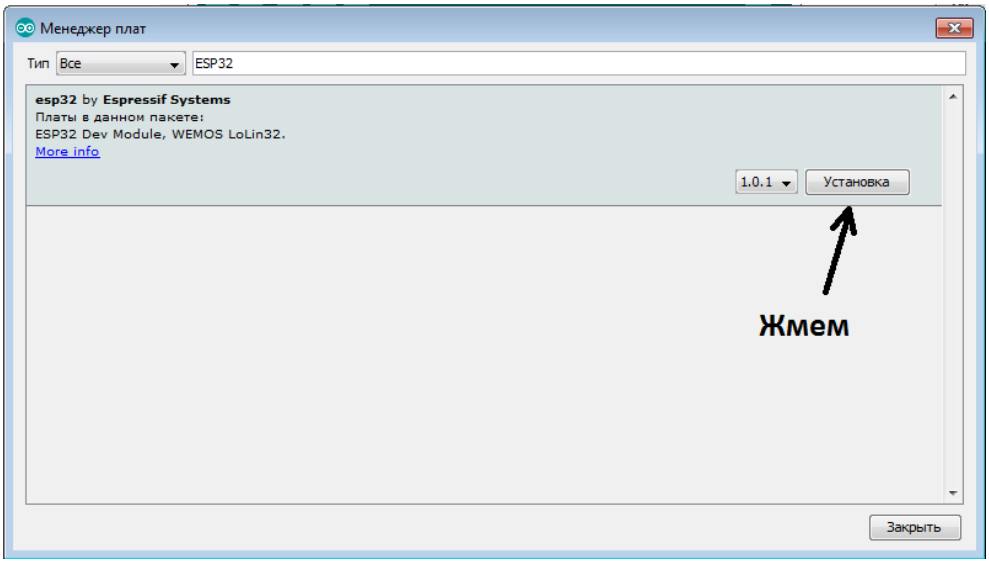


Рис.2.7. Найден пакет esp32, можно приступать к установке

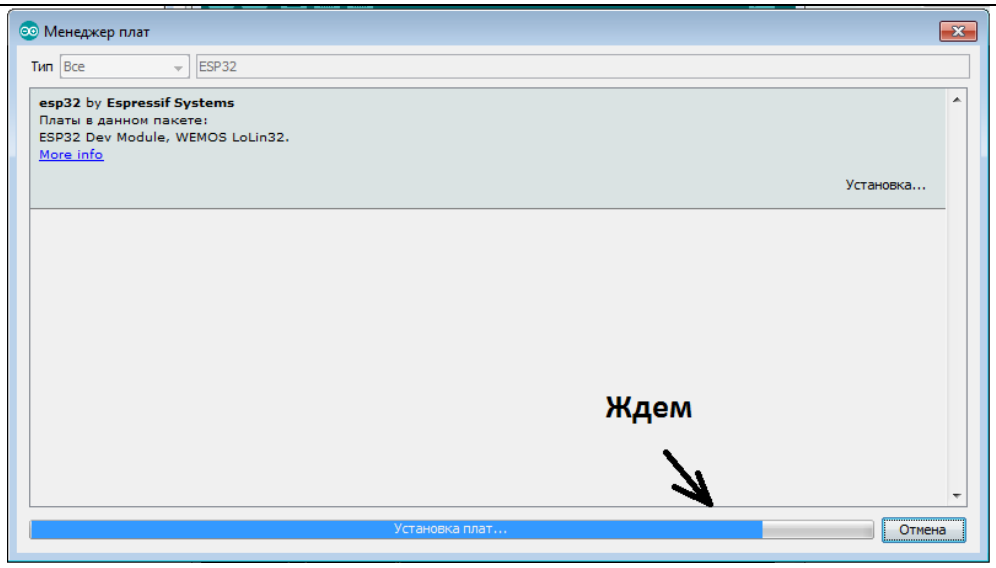


Рис.2.8. Этап установки (скачивание недостающих файлов)

Выбор платы и настройка параметров

Следует выбрать плату из серии ESP32, наиболее соответствующую плате вашего контроллера. К универсальным платам можно отнести WEMOS LOLIN32 (рис. 2.9), большинство недорогих плат, для которых отсутствует наименование в списке плат, будут при этом нормально прошиваться и работать.

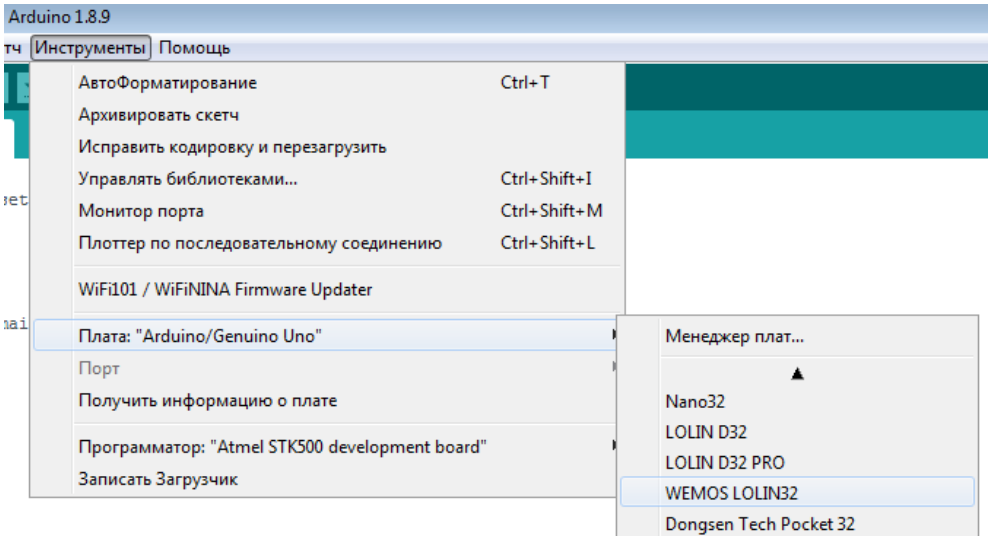


Рис.2.9. Выбор платы ESP32 WEMOS LOLIN32

Устанавливаем скорость загрузки на максимум (рис.2.10).

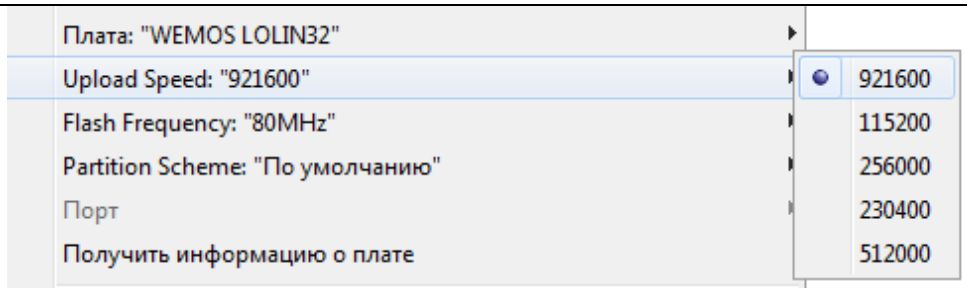


Рис.2.10. Установка скорости загрузки прошивки

Устанавливаем частоту Flash на максимум (рис.2.11).

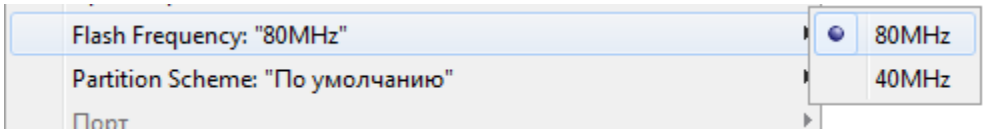


Рис.2.11. Выбор частоты Flash

Выбираем таблицу размещения данных в flash по умолчанию (рис.2.12).

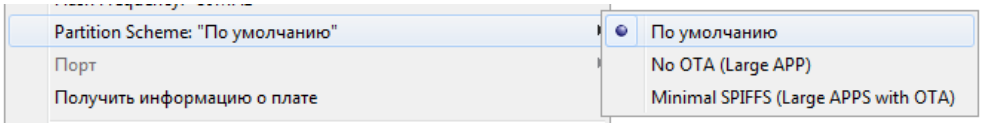


Рис.2.12. Выбор таблицы размещения информации

Тестирование работы ESP32

Давайте сразу «возьмем быка за рога» и создадим веб-сервер с доступом по Wi-Fi, конечно для этого нужно, чтобы у вас была Wi-Fi точка доступа. Этот пример не потребует дополнительных модификаций модуля ESP32 и достаточно объективно продемонстрирует его возможности.

Подключение к компьютеру и выбор COM-порта

Подключаем плату ESP32 к компьютеру, в среде Arduino IDE появляется COM -порт, на рис.2.13 это COM9, его и выбираем.

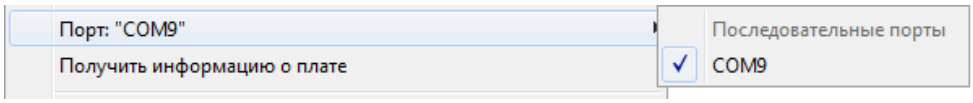


Рис.2.13. Плата подключена, выбор порта

Установка AdvancedWebServer

В главном меню программы (рис.2.14) выбираем Файл-Примеры-WebServer–AdvancedWebServer. Сохраняем пример под любым именем. В коде примера находим строки:

```
const char *ssid = "YourSSIDHere";
const char *password = "YourPSKHere";
```

Заменяем "YourSSIDHere" на имя вашей домашней WI-FI сети, а "YourPSKHere" на пароль к данной сети.

Если ваша сеть поддерживает автоматическую раздачу IP-адресов, то на этом всё. Нажимаем кнопку «Загрузка» в Arduino IDE. Если на вашей плате ESP32 есть кнопка «Boot», после появления информации о начале загрузки нажимаем кнопку на пару секунд.

Переходим в Монитор порта, выставляем скорость 115200, нажимаем на ESP32 кнопку «Reset» (RST). Модуль перезагрузится, при этом возможно появятся нечитаемые символы – модуль еще не установил нужную скорость порта. Если подключение к сети Wi-Fi выполнено успешно, получим IP-адрес нашего модуля (рис.2.15).

Переходим в браузер и заносим полученный IP-адрес в адресную строку, произойдет подключение к Web-серверу, которым является наш ESP32. На странице будет приветствие, время работы модуля и рандомно изменяющийся график (рис.2.16). График не несет какой-либо информации и лишь демонстрирует возможности.

Видеоматериал: Обновление прошивки через WIFI соединение

(Наведите камеру смартфона на QR код)

Урок 3 Основы программирования Arduino

Для того чтобы робот начал решать какую-либо задачу, собрать его недостаточно, потребуется создать и поместить в него специальную программу (план действий). Подобные программы называются *компьютерными*.

Компьютерная программа

Компьютерная программа — это четко формализованный план, состоящий из команд для контроллера (системы принятия решений). Контроллер поочередно читает команды и исполняет их. Стоит указать, что команды внутри любого цифрового устройства, коим и является контроллер Arduino, закодированы нулями и единицами, которые называются *двоичным представлением чисел*, т. е. вся информация перед поступлением в контроллер перекодируется из привычной для нас десятичной системы счисления в двоичную. Таким образом, при выполнении логических или арифметических операций контроллер сравнивает, делит, вычитает, выполняет прочие действия именно над двоичными числами. Эти числа хранятся в последовательных ячейках памяти, имеющих определенные адреса. При поступлении на контроллер Arduino электрического питания автоматически начинается выполнение той программы, которая была в него загружена, если же программа отсутствует или написана некорректно, то происходит сбой, который либо останавливает выполнение команд, либо приводит к *зависанию* программы (переходу ее в бесконечный цикл). Номер выполняемой команды хранится в специальной ячейке памяти, которая называется *счетчиком команд*. Этот номер изменяется на следующий при выполнении арифметических операций, но может измениться и на любой адрес, если выполнялась логическая команда, результатом которой стал переход на некоторый пункт плана, отличный от следующего по порядку.

Алгоритм

Для визуального представления компьютерных программ используют их графическую запись, называемую *блок-схемой алгоритма*. Алгоритм, не имеющий логических блоков и выполняемый строго по пунктам от начала и до конца, называется *линейным* (рис. 4.1). Если алгоритм после достижения последнего пункта подразумевает бесконечное повторение, его называют *циклическим* (рис. 4.2). При наличии в алгоритме анализа некоторого значения и принятия решения в результате такого анализа, этот алгоритм называется *разветвляющимся* (рис. 4.3). Алгоритмы легко анализировать и создавать — для этого нужен только лист бумаги и карандаш. При помощи алгоритмов можно описывать не только компьютерные программы, но и любые планы — например, планировать бизнес-процесс.

Рис. 3.1.
Линейный алгоритм

Рис. 3.2. Циклический алгоритм

Рис. 3.3. Разветвляющийся алгоритм

Алгоритмы можно записывать и в простой текстовой форме:

Ехать вперед 10 секунд.

Повернуть направо на 10 градусов.

А для разветвляющегося и циклического алгоритмов:

1. Получение информации о наличии препятствия с датчика.

Если препятствие есть, то перейти на пункт 5.

Ехать вперед 10 секунд.

Перейти на пункт 6.

Повернуть направо на 10 градусов.

Перейти на пункт 1.

Текстовая форма алгоритма более похожа на компьютерную программу, но визуально воспринимается хуже, в ней труднее найти логические ошибки, особенно если переходов много.

Для того чтобы создать компьютерную программу, требуется ясно понять, что она должна делать, и главное, как она это будет делать. Если некоторые вопросы вами еще не проработаны, а вы уже сели за написание серьезной программы, то, скорее всего, она не будет работать так, как вы рассчитываете. Поэтому нужно идти от простого к сложному, разбивать программы на маленькие простые блоки, добиваться их работоспособности, а только затем собирать из них полную программу для робота.

Урок 4 Балансирующие роботы

Каждый начинающий конструктор должен уметь сделать балансирующего робота.

Такой робот получает информацию о своем наклоне от гиродатчиков и подкручивает колеса в сторону наклона, тем самым уменьшает наклон и восстанавливает равновесие (балансирует).

Наличие информации об отклонении от вертикали и механизма балансировки дает роботу существенные плюсы к двигательным возможностям, робот может ориентироваться, катится он с горы или в гору, на сколько наклонен вбок. Роботы, имеющие только пару колес, существенно манёвреннее своих четырехколесных собратьев, тратят меньше энергии на повороты. Широкое распространение получили сигвеи и гироскутеры на двух колесах

Квадрокоптеры также летают устойчиво потому, что получают информацию по своему наклону от гироприборов и имеют возможность на нее правильно реагировать уменьшая/увеличивая мощность подаваемую на определенный винт (см.рис.8.1) в пропорционально отклонению от вертикали.

Рисунок 4.1. Квадрокоптер в режиме рассогласования мощности моторов

Применяемые инерционные приборы называют акселерометрами и гироскопами (гироприборами). Они также широко используются в современных самолетах, ракетах, кораблях и подводных лодках.

Благодаря встроенным гироприборам, большинство смартфонов умеет реагировать на наклон: переворачивать картинку или управлять игрой без помощи экранных кнопок.

Процесс балансировки робота состоит из двух этапов: 1) получение данных от гироприборов и ходовых моторов; 2) обработка данных с выработкой команд на исполнительные механизмы (моторы колес).

Начнем знакомство с гироприбора, в качестве которого используем популярный совмещенный гироскоп, акселерометр, термометр MPU-6050.

Существуют более дорогие и современные гироприборы с гораздо большим потенциалом, например, BNO055, но существенных преимуществ конкретно нашему роботу они не дают.

MPU-6050

На плате MPU-6050 (рис. 8.2) имеется 8 контактов, но для работы достаточно подключить 4 из них: VCC — питание 3,3 В (возьмем от платы ESP32), GND — «земля», SDA и SCL — контакты шины I2C. SCL — вывод тактового сигнала и SDA — передача/прием данных в обоих направлениях.

Особенность GPIO для I2C ESP32

По умолчанию 21 и 22 GPIO используются в качестве SDA и SCL шины I2C, но это не является обязательным. Номера GPIO можно настроить в программе, задав их в качестве параметров при инициализации I2C. Это актуально, так как не на всех макетных платах 21-й GPIO выведен разъемом на макетную плату и доступен для использования.

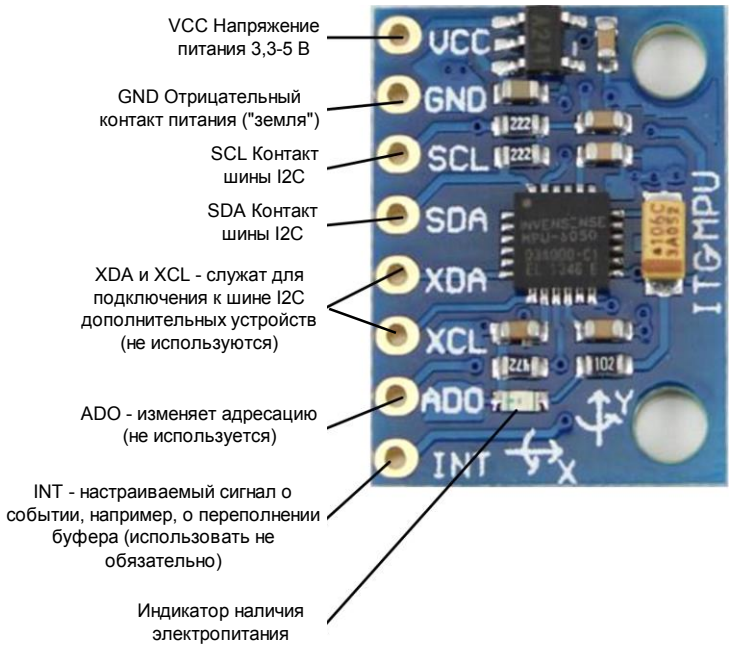


Рис. 4.2. Микросхема гироскопа-акселерометра MPU-6050, смонтированная на плате

I2C – последовательный интерфейс, к которому могут быть подключены одновременно несколько устройств, одно из которых является ведущим, а остальные ведомыми. В нашем случае ведущим будет контроллер ESP32, а

MPU-6050 ведомым. Каждое устройство на шине I2C должно иметь уникальный адрес. Когда ведущий начинает передачу, он передает по шине адрес устройства, к которому выполняется обращение. Устройства, подключенные к шине, проверяют этот адрес, и в случае совпадения начинают обмен информацией.

Подключение нескольких устройств с одинаковым адресом к шине I2C

Если у вас имеется потребность в нескольких одинаковых устройствах, то нужно позаботиться о том, чтобы они имели разные адреса, нередко адресацию можно изменить перемычками на плате устройства, а если подобной функции нет, потребуется специальное устройство I2C мультиплексор (TCA9548).

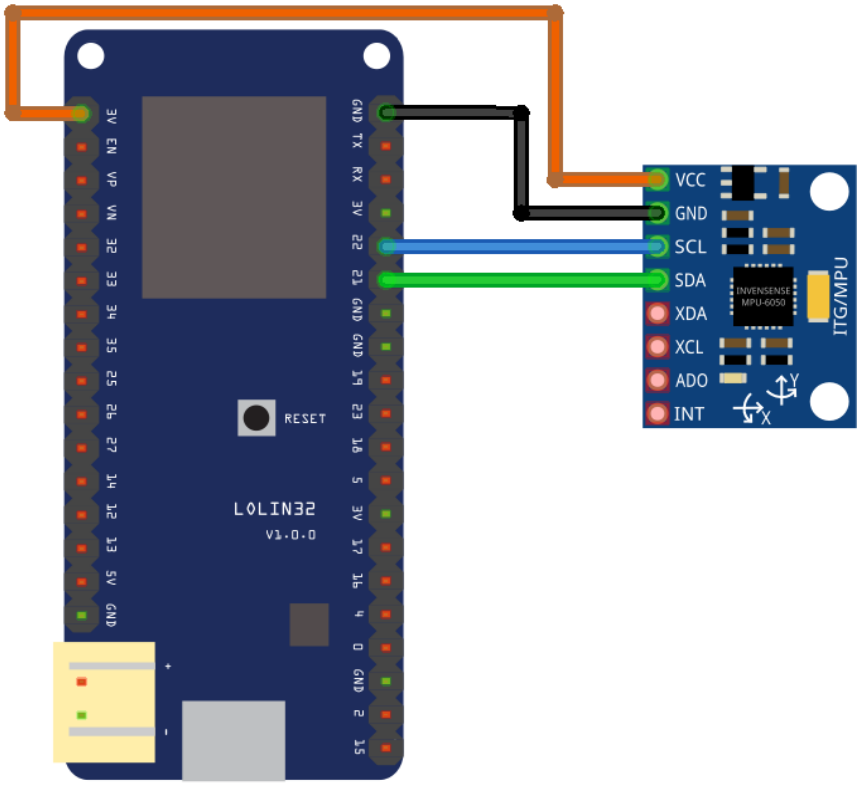


Рис. 4.3. MPU-6050 подключен к ESP32 (питание от внутреннего стабилизатора)

Электронный гироскоп

Гироскоп — это прибор, способный реагировать на изменение ориентации объекта. Простейшим примером гироскопа является волчок. Раскрученный волчок не падает, сохраняя свое вертикальное положение, даже если наклонять поверхность, на которой он установлен. Свойство вращающихся тел удерживать положение оси вращения в пространстве лежит в основе работы механических гироскопов.

Но электронный гироскоп серьезно отличается от механического — он не показывает угол отклонения от заданного направления. Электронный гироскоп регистрирует и возвращает текущую угловую скорость (скорость вращения прибора по осям). Зная мгновенную угловую скорость, можно вычислить угол, на который повернется объект за некоторый промежуток времени:

$$Ang_2 = Ang_1 + \Delta t \times V_{ang}, \tag{4.1}$$

где Ang_1 — начальный угол, рассчитанный ранее, Δt — малый промежуток времени, на котором можно считать, что угловая скорость не менялась, V_{ang} — полученное от электронного гироскопа в течение Δt значение угловой скорости, Ang_2 — рассчитанный угол, на который повернулся объект вокруг исследуемой оси с начала поворота. На рис. 8.4 вращение происходит вокруг оси *Z*. Показания гироскопа именно по этой оси следует использовать в расчетах.

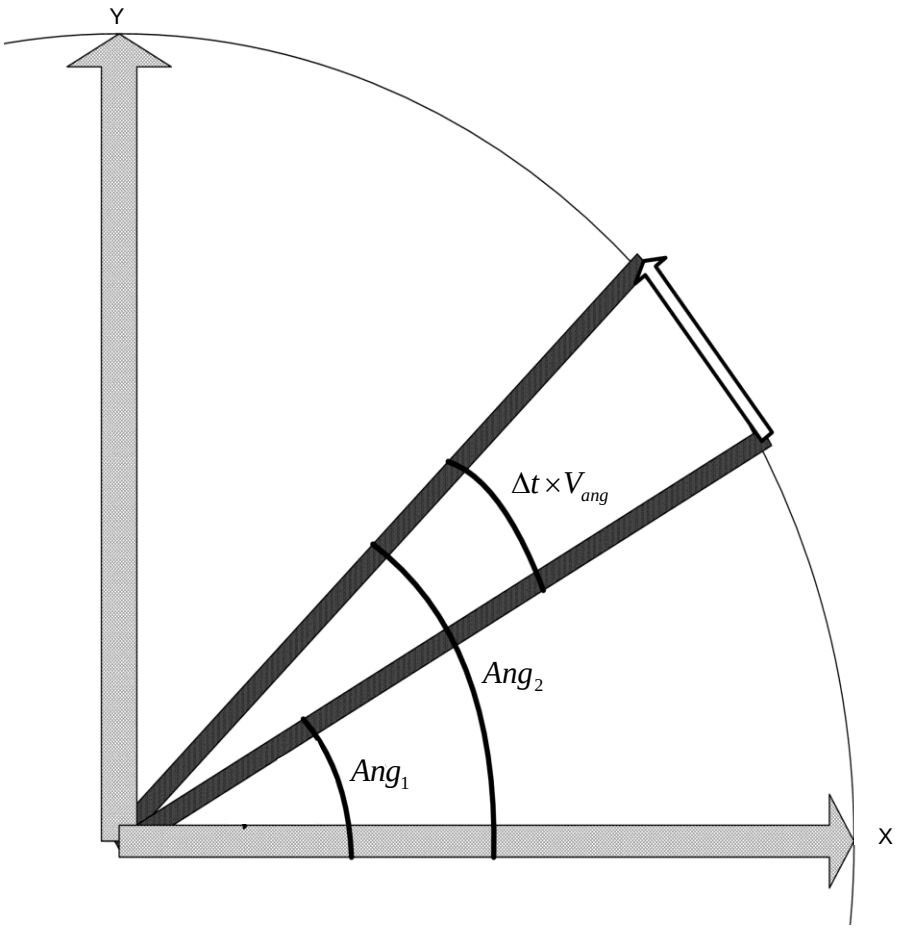


Рис. 4.4. Расчет угла поворота по показаниям электронного гироскопа

Погрешности электронного гироскопа

Вследствие того, что анализ производится дискретно через заданные промежутки времени, использование только электронного гироскопа может приводить к накоплению погрешности. Также

электронные гироскопы могут иметь ограниченный диапазон чувствительности и, если угловая скорость выйдет за пределы измерительной шкалы, измерения станут ошибочными.

Электронный акселерометр

Акселерометр служит для измерения ускорения движущегося тела. На основе его показаний и используя простые формулы, можно отслеживать передвижение тела в пространстве. Следует учитывать, что простой и дешевый MPU6050, в отличие от дорогих сложных устройств, не умеет отделять ускорение свободного падения от ускорения, полученного объектом в результате разгона/торможения. Кроме того, данные от MPU6050 имеют большой уровень шума, что не позволяет их использовать напрямую без дополнительной обработки. Однако в качестве прибора для нахождения углов отклонения объекта от горизонтали MPU6050 **использовать можно.** Для точного измерения объект должен быть неподвижен или двигаться прямолинейно без ускорения. В этом случае на него действует только сила притяжения, направленная к центру Земли.

На рис. 8.5 изображен неподвижный объект (робот) на наклонной плоскости. Координатная ось *Y* акселерометра робота направлена вперед, ось *Z* — вниз под прямым углом к наклонной плоскости, ось *X* — к наблюдателю. Искомый угол отклонения объекта от вертикали Ang_{YZ} находится по формуле:

$$Ang_{YZ} = arctg\left(\frac{a_Y}{a_Z}\right), \tag{4.2}$$

где a_Y — проекция ускорения свободного падения на ось *Y* акселерометра, a_Z — проекция на ось *Z*. Можно также использовать формулы:

$$Ang_{YZ} = \arcsin\left(\frac{a_Y}{g}\right)$$

или

$$Ang_{YZ} = \arccos\left(\frac{a_Z}{g}\right),$$

где g — ускорение свободного падения, но для электронного прибора, который будет рассмотрен далее, рекомендуется использовать формулу (4.2), оперирующую только относительными величинами, в этом случае нам не придется дополнительно калибровать прибор.

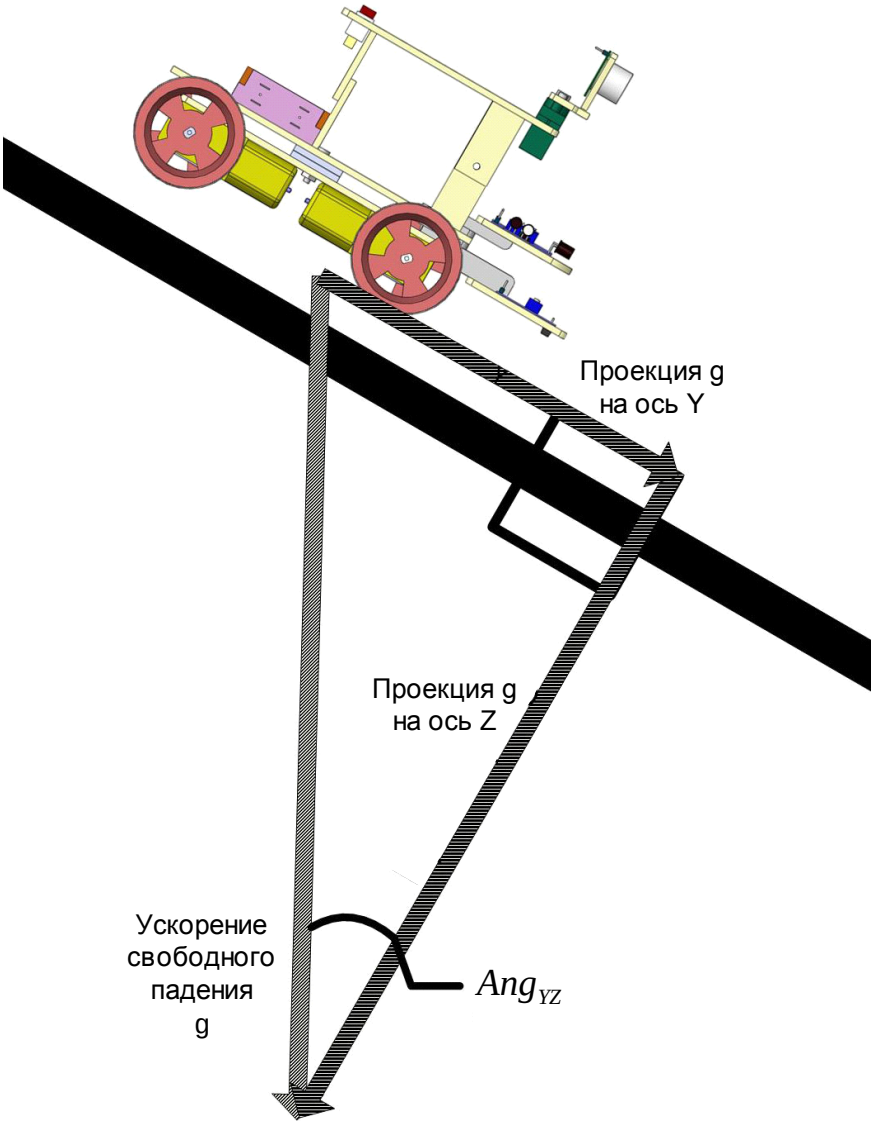


Рис. 4.5. Нахождение угла отклонения от вертикали (робот неподвижен)

Получение и обработка данных от MPU-6050

Контроллер MPU-6050 посылает запрос на получение данных, гироприбор возвращает запрошенные значения. На листинге 4.1.1 приведен пример небольшой библиотеки, в которой реализованы функции инициализации прибора, получение данных от прибора, расчет компенсации смещения нуля гироскопа.

Шкала значений MPU-6050

Если показания акселерометра в относительных единицах нас вполне устраивают (см. формулу 8.2), то с гироскопом немного сложнее, требуется конвертировать величины, возвращенные прибором, в угловую скорость градусы/секунда или радианы/секунда. Чтобы перевести показания прибора из относительных единиц в привычные абсолютные, следует обратиться к документации. На сайте производителя InvenSense (**www.invensense.com**) представлена таблица шкалы измерений MPU-6050 (таб. 4.1). В частности,

чтобы перевести показания гироскопа в градус/сек, их требуется поделить на 131 (при максимальной чувствительности прибора). Для перевода ускорения в привычную для нас систему мер требуется поделить показания прибора на 16384, а затем умножить на 9,8 м/с².

Таблица 4.1 Характеристики шкалы измерений MPU-6050

Gyro Full Scale Range	Gyro Sensitivity	Gyro Rate Noise	Accel Full Scale Range	Magnetometer Full Scale Range	Accel Sensitivity	Digital Output	Logic Supply Voltage	Operating Voltage Supply
(°/sec)	(LSB/°/sec)	(dps/√Hz)	(g)		(LSB/g)		(V)	(V +/-5%)
±250	131	0.005	±2	N/A	16384	I²C	1.8V±5% or VDD	2.375V–3.46V
±500	65.5	0.005	±4		8192			
±1000	32.8	0.005	±8		4096			
±2000	16.4	0.005	±16		2048			

Установим MPU6050 на робота как показано на рисунке 4.6. и проведем тесты по получению информации от гироприбора, рассчитаем углы наклона робота. Будем использовать формулы 4.1, 4.2, а также формулу комплементарного фильтра, который производит пропорциональное смешивание данных от гироскопа и акселерометра, что позволит избавиться от шума (колебания) данных акселерометра и устранить погрешность, которая постоянно накаливается при расчете угла только от гироскопа:

$$Ak_n = (Ak_{n-1} + S \times dt) \times Kf + Aa \times (1 - Kf), \quad (4.3)$$

где Ak_n – рассчитанный угол, Ak_{n-1} – угол, рассчитанный в прошлом опросе; *S*- угловая скорость, полученная от гироскопа; *dt* – время, прошедшее с момента прошлого опроса гироприбора и расчета Ak_{n-1}; *Kf* – коэффициент влияния на результат расчета, основанного только на угловой скорости; *Aa* – текущий угол, рассчитанный по показаниям акселерометра; $(1-Kf)$ – коэффициент влияния на результат угла по показаниям акселерометра.

Kf для MPU6050 изменяется от 0.96 до 0.99, при уменьшении становятся заметны колебания от акселерометра, при увеличении может накапливаться погрешность, связанная с гироскопом. По факту берутся 0.99 частей от расчета по гироскопу и смешиваются с 0.01 частью расчета по акселерометру.

Для MPU6050, установленного как на рисунке 4.6, ось вращения колес соответствует оси Y прибора.. Поэтому в расчете угла наклона по показаниям акселерометра будут использоваться показания акселерометра по осям X и Z, данные угловой скорости по оси Y.

Программа listing_4_1 состоит из двух частей: программы получения данных от MPU-6050 (файл gyro_acsel.h) и основной программы опроса MPU (файл listing_4_1.ino). Файл gyro_acsel.h содержит описание переменных, используемых для хранения полученных от MPU6050 значений: **AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ. К**оэффициент *Kf* из формулы 4.3, это константа ONE_ALFA, а *(1 – Kf)* – константа ALFA. Константа MPU_addr содержит адрес MPU6050 на шине I2C (0x68).

Функция **giroscop_setup()** запускается один раз в начале программы, инициализирует протокол I2C (Wire), включает гироприбор.

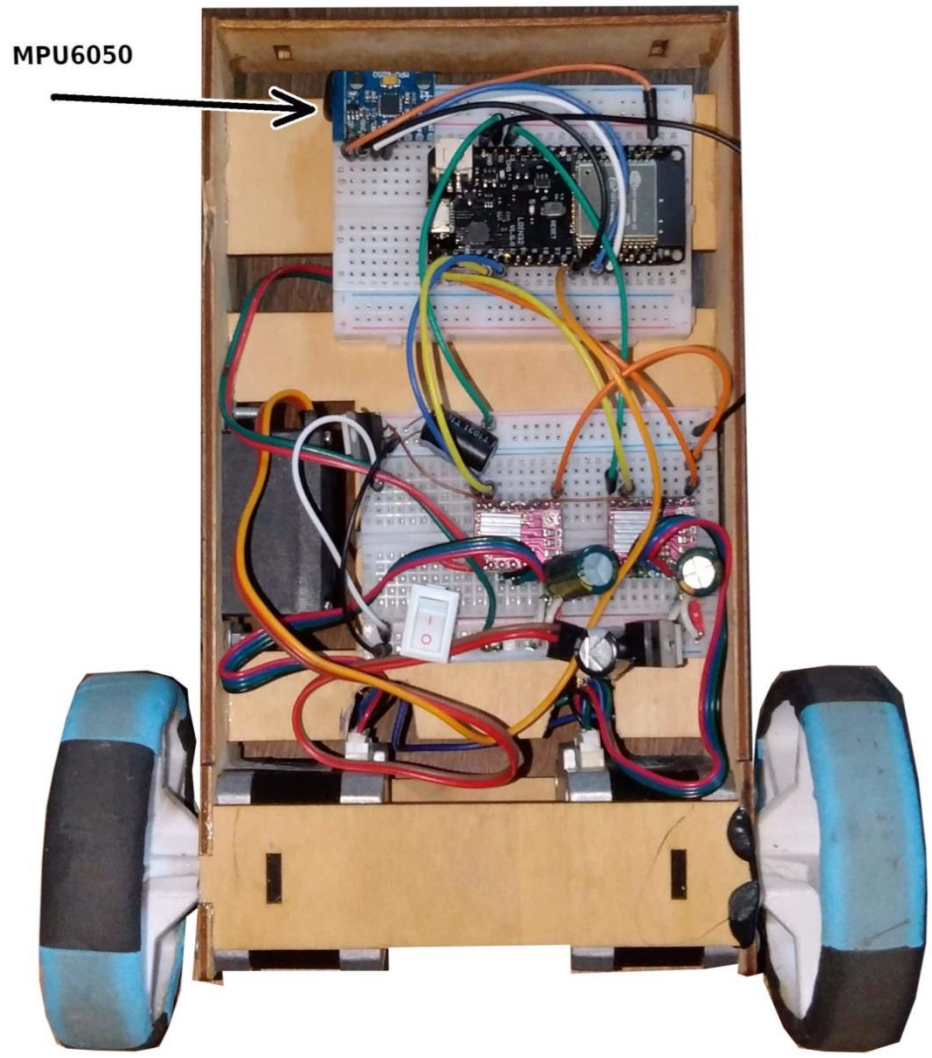


Рис. 4.6. Робот с установленным MPU6050

Функция **Data_mpu6050()** производит запрос данных от MPU6050 и помещает полученные данные в 16-разрядные переменные AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ. AcX, AcY, AcZ – показания акселерометра по трем осям, GyX, GyY, GyZ – показания угловой скорости вращения вокруг соответствующих осей, **Tmp** – температура (может использоваться в расчете температурного смещения показаний прибора).

Функция **Calc_CompensatorZ(float mill_sec)** останавливает работу программы на заданное количество миллисекунд, в течение которых робот должен быть неподвижен, что позволяет рассчитать смещение нуля гироскопа.

Робот неподвижен, с MPU6050 снимаются показания угловой скорости, показания не нулевые и колеблются вокруг некоторого значения – по показаниям прибора присутствует угловая скорость, но реального вращения нет. Получаемые показания суммируются, а затем делятся на их количество. Получаем среднее значение смещения нулевой точки для каждой оси. Смещения заносятся в переменные **CompensatorZ, CompensatorX, CompensatorY** и в дальнейшем для повышения точности должны вычитаться от показаний угловой скорости, полученной от гироскопа.

Листинг 4.1.1. Получение данных от MPU-6050 (файл gyro_acsel.h)

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>

#define ALFA 0.01
#define ONE_ALFA 0.99//

// упрощеный I2C адрес нашего гироскопа/акселерометра MPU-6050.
const int MPU_addr = 0x68;
// переменные для хранения данных возвращаемых прибором.
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
double CompensatorZ, CompensatorX, CompensatorY;
unsigned long timer = 0;
//////////////////////////////////////////////////////////////////
/// Запуск гироскопа
//////////////////////////////////////////////////////////////////
void giroscop_setup()
{
  /* Enable I2C */
  Wire.begin();//22,23); //SDA //SCL
#ifdef ESP8266
  Wire.setClockStretchLimit(1000); // Allow for 1000us of clock stretching
#endif
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // Производим запись в регистр энергосбережения MPU-6050
  Wire.write(0);     // устанавливаем его в ноль
  Wire.endTransmission(true);
  CompensatorZ = 0;
  CompensatorX = 0;
  CompensatorY = 0;
}
//////////////////////////////////////////////////////////////////
//// Считывание данных с mpu6050
//////////////////////////////////////////////////////////////////
void Data_mpu6050()
{
  Wire.beginTransmission(MPU_addr);
```

```
  Wire.write(0x3B);  //Готовим для чтения регистры  с адреса 0x3B.
  Wire.endTransmission(false);
  // Запрос 14 регистров.
  Wire.requestFrom(MPU_addr, 14, true);
  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcX = Wire.read() << 8 | Wire.read();
  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcY = Wire.read() << 8 | Wire.read();
  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  AcZ = Wire.read() << 8 | Wire.read();
  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  Tmp = Wire.read() << 8 | Wire.read();
  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyX = Wire.read() << 8 | Wire.read();
  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyY = Wire.read() << 8 | Wire.read();
  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  GyZ = Wire.read() << 8 | Wire.read();
}
////////////////////////////////////////////////////////////////////////
// Компенсация нуля гироскопа
void Calc_CompensatorZ(float mill_sec)
{
  long ms = mill_sec;
  float i = 0;
  CompensatorZ = 0;
  CompensatorX = 0;
  CompensatorY = 0;
  timer = millis();
  unsigned long endtime = millis() + ms;
  while (endtime > timer) {
    timer = millis();
    Data_mpu6050();
    CompensatorZ += (float)(GyZ);
    CompensatorX += (float)(GyX);
    CompensatorY += (float)(GyY);
    delay(2);
    i++;
  }
  CompensatorZ /= i;
  CompensatorX /= i;
  CompensatorY /= i;
}
```

Вторая часть программы (файл listing_4_1.ino) служит для расчета угла наклона робота вокруг оси Y гироприбора. Переменные **dt, t0, t2, t_period** используются в расчете длительности периода между получением данных от MPU. Переменные **AcYsum, GyXsum0, OldGyro, OldAcYsum, AcYsumOld, Acsel, Gyro** используются в расчете угла наклона робота.

Переменная **_1_d_131** – коэффициент перевода угловой скорости, полученной от гироприбора, в величину градусы/секунды.

Повторение расчетов происходит в функции loop через время, не менее чем **t_period** – 5000 микросекунд.

Dt = double(dt) * 0.000001, это точное время предыдущего периода в секундах, **dt -** это микросекунды.

Показания акселерометра рассчитываются в формуле:

```
//с учетом поворота прибора
  double Atan = atan2(AcX, AcZ);
  if(Atan>-PI/2.0)
  Acsel = (Atan-PI/2.0) * RAD_TO_DEG;
  else Acsel = (PI*1.5+Atan) * RAD_TO_DEG;
```

где Atan – угол, рассчитанный по показаниям акселерометра.

Нулевые показания прибора должны быть при горизонтальном положении платы MPU-6050 робота. Для работы в вертикальном положении (балансирующий робот) показания нужно сместить на 90 градусов. Для этого служит процедура, позволяющая получать правильные (смещенные) значения при всех углах **Atan. RAD_TO_DEG** – коэффициент перевода радиан в градусы.

Угловая скорость рассчитывается по формуле:

```
Gyro = - (float(GyY) - CompensatorY)  * _1_d_131;
```

Фактически угловая скорость уменьшается на дрейф нуля (**CompensatorY**) и конвертируется в градусы/секунды.

Показания акселерометра и гироскопа прибора MPU-6050 имеют особенность, которую необходимо учесть при расчетах. При росте угла по акселерометру, гироскоп показывает отрицательную угловую скорость. Но, как известно, при росте угла угловая скорость должна быть положительной, при уменьшении угла – отрицательной.

Поэтому для получения правильного результата следует согласовать показания гироскопа и акселерометра, инвертировав показания либо угловой скорости, либо угла наклона.

Расчет значения угла наклона по правилам комплементарного фильтра приведен ниже:

```
AcYsum = ONE_ALFA * (OldAcYsum + Gyro * Dt) + ALFA * Acsel.
```

По правилами расчета комплементарного фильтра: **ONE_ALFA + ALFA = 1**, **ONE_ALFA = 0.99, ALFA = 0.01**.

Расчет смещения угла при использовании только гироскопа:

```
GyXsum0 =GyXsum0+Gyro * Dt;
```

Гироскоп ничего не знает о начальном значении угла наклона робота и считает только смещение.

Чтобы убедиться, в правильности выбора для расчетов именно комплементарного фильтра визуально проследим за “особенностью” работы

акселерометра, для этого воспользуемся плоттером по последовательному соединению.

Плоттер по последовательному соединению

Плоттер можно запустить из среды Arduino IDE (Меню – Инструменты – Плоттер по последовательному соединению). Плоттер рисует на экране графики по числовым данным, полученным по Serial каналу от контроллера. Чтобы вывести несколько графиков, данные должны передаваться через пробел. Отрисовка следующей позиции графика происходит после получения кода переноса строки – команда контроллера Serial.println().

Перенаправим вывод рассчитанных данных: результирующий угол (комплементарный фильтр), отклонение угла (гироскоп), результирующий угол (акселерометр). Для этого в конце программы производится вывод в Serial порт рассчитанных значений. Вывод организован каждый повтор, это сделано для прорисовки графика в «Плоттере по последовательному соединению». Если требуется вывести результаты в порт для просмотра, требуется убрать комментарий со строки // if (i > 5). Также можно заменить значение 5 на большее.

Листинг 4.1.2. Основная программа опроса MPU (файл listing_8_1.ino)

```
#include "gyro_acsel.h"
//Переменные управления интервалами опроса гироприбора и интервалами управления
unsigned long dt;
unsigned long t0;
unsigned long t2;
double AcYsum, GyXsum0;
double OldGyro = 0;
double OldAcYsum = 0;
double AcYsumOld = 0;
int32_t t_period;
//double   _1_d_131 = 1.0 * DEG_TO_RAD / 131.0;
double   _1_d_131 = 1.0  / 131.0;
double Acsel = 0;
double Gyro = 0;
void setup() {
  Serial.begin(115200);
  Serial.println();
  giroscop_setup();
  delay(100);
  Calc_CompensatorZ(2000);
  t0 = micros() - 5000;
  t2 = micros();
}
```

```
void loop()
{
  t_period = 5000;
  static int i = 0; // для торможения вывода
  uint32_t micros_;
  micros_ = micros();
  //Если период < t2 (прошло 5000 если да, производим очередной расчет)
  if (micros_ < t2) return;
   dt = micros() - t0; // Длительность предыдущего периода регулирования.
  t0 += dt; //Точка начала нового периода регулирования.
  //переводим в секунды время от предыдущего опроса.
  double Dt = double(dt) * 0.000001;
  //Опрос гироприбора:
  Data_mpu6050();
  //Расчет угла по показаниям акселерометра
  //с учетом поворота прибора
  double Atan = atan2(AcX, AcZ);
  if(Atan>-PI/2.0)
  Acsel = (Atan-PI/2.0) * RAD_TO_DEG;
  else Acsel = (PI*3.0/2.0+Atan) * RAD_TO_DEG;
  // скорость угловая - падения
  Gyro = - (float(GyY) - CompensatorY)  * _1_d_131;
  //Комплементарный фильтр
  AcYsum = ONE_ALFA * (OldAcYsum + Gyro * Dt) + ALFA * Acsel;
  GyXsum0 =GyXsum0+Gyro * Dt;
  OldAcYsum = AcYsum;
  t2 = t0 + t_period;
 if (i > 100)
  {
    i = 0;
   Serial.print(GyXsum0);
    Serial.print(" ");
    Serial.print(Acsel);
    Serial.print(" ");
    Serial.println(AcYsum);
  }
  i++;
}
```

Результат работы программы приведен на рисунке 4.7. Загрузите программу в контроллер робота и после начала ее работы откройте Плоттер. Плавно наклоняйте робота вдоль оси колес, в плоттере должны отрисоваться три кривые – рисунок 4.7. Верхняя синусоида, это результат расчета с использованием только гироскопа, смещение кривой вдоль оси Y графика произвольное (смещен вверх на 90 градусов). Две оставшиеся кривые совпадают по частоте и уровню, но одна из них плавная – расчет угла по

комплементарному фильтру, а вторая имеет значительные резкие всплески, это расчет угла по показаниям акселерометра.

Колебание показаний акселерометра приводит к тому, что в динамических системах использование его без дополнительной математической обработки затруднительно, но смешивание данных от гироскопа и акселерометра дает четкую картину по отклонениям от вертикали.

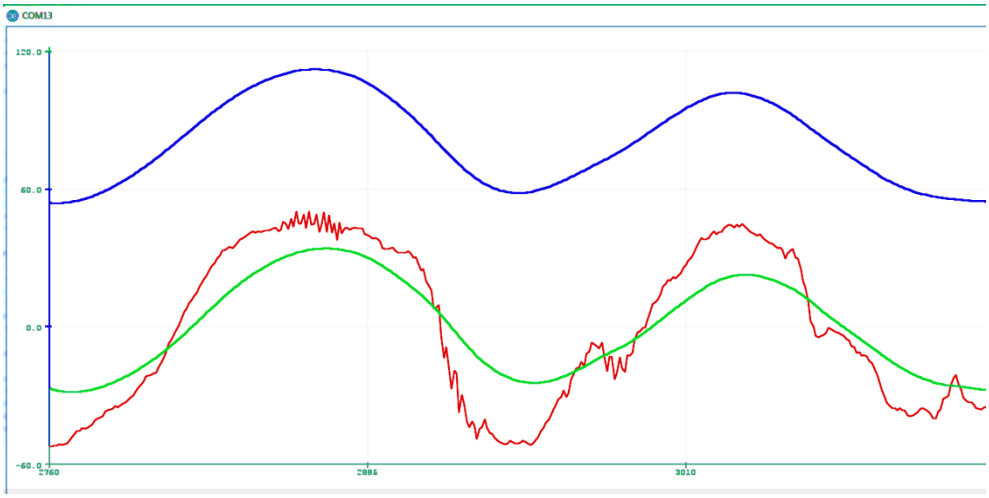


Рис. 4.7. Расчет изменения угла по гироскопу (верхняя синусоида), акселерометру (нижняя синусоида со всплесками), при помощи комплементарного фильтра (нижняя плавная синусоида)

Сопоставим положение робота и результат на экране. Если при установке и удержании робота в вертикальном положении кривая комплементарного фильтра близка к нулевой точке по оси Y графика, значит, оси выбраны правильно. В противном случае это может означать, что прибор на вашей плате расположен по-другому, и следует изменить-проверить ось, используемую в качестве базовой для расчета.

Следует заметить, что возможен также температурный дрейф показаний гироприбора — тогда погрешность будет зависеть и от температуры прибора. Учесть температурную погрешность можно, но эта задача выходит за рамки книги.

Основы регулирования (ПИД регулятор)

Прежде чем приступить к разработке программы балансировки рассмотрим основы регулирования и разберем на более простых примерах.

Для регулирования анализируется состояние системы и вырабатывается управляющее воздействие. Применяются три типа обработки состояния системы при расчете величины управляющего воздействия: пропорциональный, интегральный, дифференциальный. Не редко эти типы регулирования применяют совместно, отсюда возникла аббревиатура **ПИД-регулятор**.

Пропорциональное регулирование применяется тогда, когда управляющее воздействие пропорционально ошибке регулирования.

Дифференциальное регулирование применяется тогда, когда управляющее воздействие пропорционально скорости изменения ошибки регулирования.

Интегральное регулирование применяется тогда, когда управляющее воздействие пропорционально интегралу ошибки регулирования по времени.

Рассмотрим работу ПИД-регулятора на примере электрического мотора. В нашем распоряжении электрический мотор постоянного тока, подключённый к драйверу. Драйвер управляется при помощи ШИМ от контроллера. С вала мотора датчик снимает скорость, которая поступает в контроллер. Требуется научить контроллер управлять скоростью мотора.

Что такое ШИМ

ШИМ, это **широтно-импульсная модуляция**, в англоязычной интерпретации PWM (pulse-width modulation). Управляющий контроллер генерирует импульсы с определенной частотой, которые подаются на вход управления мощностью управляемого устройства. Управляющий контроллер может изменять ширину импульсов от 0 ширины до 100% заполнения, при этом выходная мощность устройства изменяется от 0 до 100% линейно.

В качестве информации о текущем состоянии регулируемой системы выступает скорость на валу мотора, а регулировать состояние системы мы будем, изменяя мощность, подаваемую на мотор при помощи изменения ширины управляющих импульсов (ШИМ) на драйвер мотора.

Рассмотрим действия звеньев ПИД-регулятора:

Пропорциональное регулирование предполагает, что управляющее воздействие (у нас величина ШИМ) прямо пропорционально разности между требуемым значением выходной величины (требуемая скорость) и реально измеренным значением (измеренная скорость).

Имеем заданную скорость $Speed_{com}$, с которой требуется вращать вал мотора, контроллер получает от датчика текущую измеренную скорость $Speed_{sens}$, она может отличаться от заданной скорости. Найдем разницу между заданной и реальной скоростью $Error_{speed}$:

$$Error_{speed} = Speed_{com} - Speed_{sens},$$

Мощность E_{mot}, подаваемая на мотор, рассчитанная с использованием только пропорционального регулятора:

$E_{mot} = Error_{speed} \times Kp$, $\boldsymbol{Kp}$ – коэффициент пропорциональности.

В результате применения пропорционального звена регулирования скорости получим колебательную систему (рис. 8.8). Реальная скорость будет

колебаться вокруг заданного значения. Это связано с тем, что направление мощности прямо пропорционально отклонению, что в реальных системах, которые обладают инерционностью, приведет к перерегулированию и колебаниям.

Недостатком данного звена является отсутствие реакции на изменение нагрузки, например, если мы настраивали коэффициент пропорциональности без нагрузки, а затем начнем тормозить вал мотора, пропорциональное звено никак не сможет это компенсировать. Если же наоборот, коэффициент пропорциональности будет настроен при нагрузке на валу мотора, а затем эту нагрузку убрать, скорость вала двигателя начнет значительно колебаться, регулирования не получится (рис. 8.9).

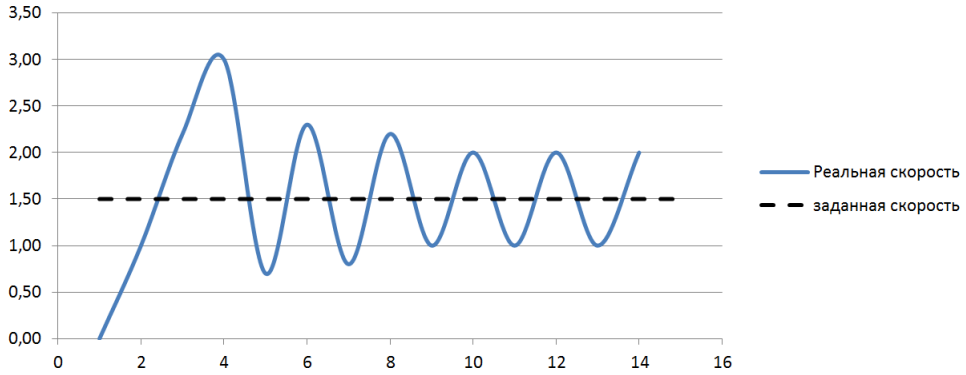


Рис. 4.8. Регулирование П-регулятор (незатухающие колебания)

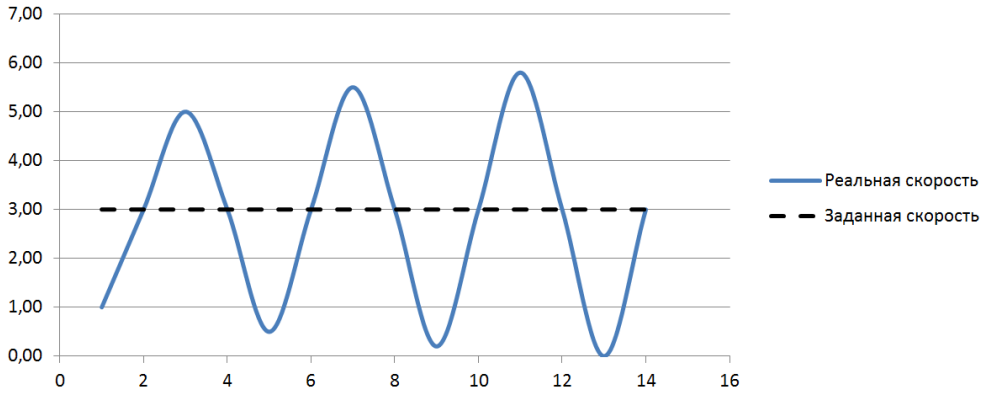


Рис. 4.9. Регулирование П-регулятор (самовозбуждение системы из-за высокого коэффициента П-регулятора)

Уменьшить колебания можно добавив в систему регулирования анализ скорости изменения ошибки – дифференциальное звено.

Дифференциальное регулирование предполагает, что управляющее воздействие прямо пропорционально разности текущей ошибки регулирования и ошибки регулирования прошлого измерения поделенное на время между проведением данных измерений. **Фактически, это скорость изменения ошибки регулирования.**

Если ошибка регулирования с прошлого измерения уменьшилась, то скорость будет отрицательной, в противном случае – положительной. Отрицательная скорость (при положительном знаке ошибки) говорит о том, что ошибка регулирования уменьшается – система приходит в норму, наличие подобного звена уменьшит влияние пропорционального звена и снизит перерегулирование. Положительная скорость (при положительном знаке ошибки) говорит о том, что ошибка растет, в этом случае дифференциальное звено увеличит управляющее воздействие, “поможет” пропорциональному звену регулирования.

Для использования дифференциального регулирования требуется, чтобы измерения производились периодически через небольшие промежутки времени. Данный параметр регулирования подразумевает использование в качестве аргументов скорость изменения ошибки регулирования и время между измерениями-расчетами скорости.

$$d_{error} = \frac{Error_{speed} - Error_{speedOld}}{dt},$$

$$E_{mot} = Error_{speed} \times Kp + d_{error} \times Kd,$$

Пример уменьшения колебаний приведен на рисунке 4.10.

В некоторых случаях применяется скорость изменения скорости или ускорение, расчет его подобен. Смысл применения аналогичный – стабилизация системы регулирования, уменьшение колебаний.

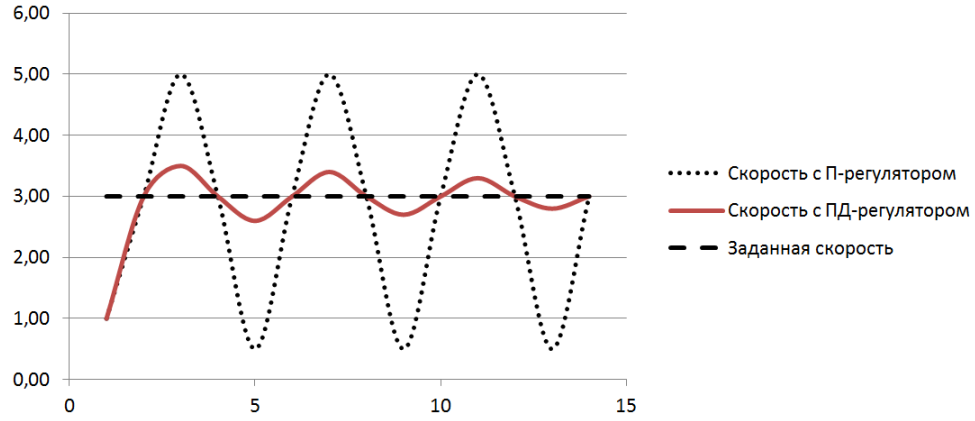


Рис. 4.10. Пример уменьшения колебаний при помощи ПД-регулятора

Интегральное регулирование применяется для компенсации недостаточного влияния пропорционального звена. Также позволяет существенно уменьшить коэффициент пропорциональности при пропорциональном звене, чем снижает колебания (рис. 4.11). Интегральное звено необходимо при регулировании в системах с переменной нагрузкой.

$$I_{error} = I_{errorOld} + Error_{speed} \times dt,$$

$$dE_{mot} = Error_{speed} \times Kp + d_{error} \times Kd + I_{error} \times Ki,$$

$$E_{mot} = E_{motOld} + dE_{mot},$$

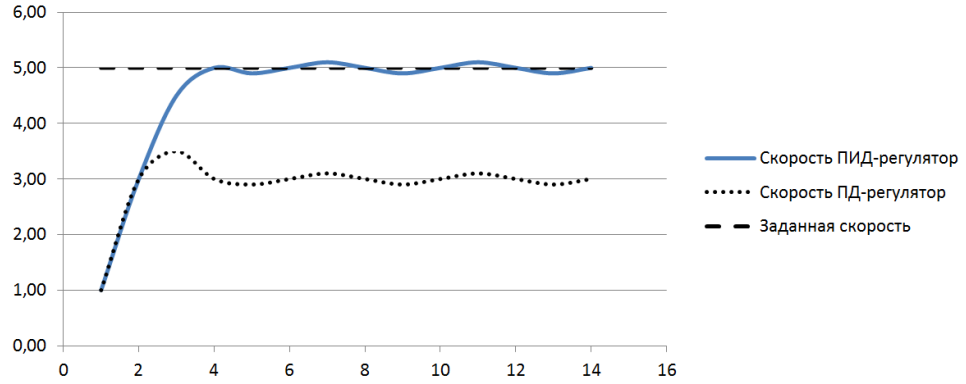


Рис. 4.11. Пример качественного регулирования (ПИД-регулятор)

Настройка ПИД-регулятора балансирующего робота

Для настройки регулятора будем использовать программу **listing_4_2**. Это адаптированный к балансировке без управления программный модуль, который подойдет для отработки навыков настройки параметров балансировки**.** Предварительно проверьте настройки направления вращения моторов (motorstep.h с 8 по 12 строки) и поправьте, если у вашего робота в предыдущих проектах были отличия**.**

В файле **listing_4_2.ino** (листинг 4.2) мы будем изменять строки с 46 по 55, где представлены различные формулы расчет скорости реакции робота. Также изменению могут быть подвергнуты строки, задающие коэффициенты:

```
Ka = 1200.0 / ( PI );
Kg = 300.0 / ( PI ).
```

Раскомментируйте формулу под цифрой //1 (удалите перед ней две косые черты), а остальные оставьте закомментированными:

```
//1
Speed = AcYsum  * Ka;
```

Если робот отклонился от вертикали, для возвращения в вертикальное положение он должен доработать колесами в сторону своего наклона. Это должно привести к тому, что робот будет устойчиво колебаться в районе своего равновесного положения. Но если скорость, с которой робот отрабатывает угол наклона, станет больше определенного значения (скорость прямо пропорциональна углу наклона), робот может перевалиться в сторону обратную от склонения, начать колебаться с увеличивающейся амплитудой и в результате упадет. Если попытаться сильно уменьшить коэффициент пропорциональности **Ka**, то робот, наоборот, не будет доходить до вертикали

и упадет сразу. В любом случае качественной балансировки не получится. Изменяйте **Ka**, отследите изменения.

Раскомментируйте формулу под цифрой //2, а остальные оставьте закомментированными:

```
//Speed = AcYsum  * Ka;
//2
Speed = Gyro  * Kg;
```

Попробуем другой вариант, возьмем угловую скорость (скорость падения робота), подберем коэффициент **Kg** и посмотрим, как робот будет реагировать на прямо пропорциональное угловой скорости изменение своего положения. В положении покоя робот, вне зависимости от наклона, будет бездействовать, и начнет сопротивляться изменению угла склонения. Кажется, вот еще не много и он встанет, но это не происходит, потому что гироскоп ничего не знает о точке равновесия робота. Робот падает, но плавно.

Раскомментируйте формулу под цифрой //3, а остальные оставьте закомментированными:

```
//3
Speed = AcYsum  * Ka + Gyro  * Kg;
```

Мы научились работать с пропорциональным (по углу наклона) и дифференциальным *(по производной по времени от угла наклона – угловой скорости)* регулятором, а теперь подставим их работу вместе. Если робот, используя такой регулятор, резко уходит в нестабильное состояние, уменьшите коэффициенты по обоим звеньям вдвое. Робот не станет балансировать на месте, но некоторое время проедет до своего падения. Если не получилось с первого раза, попробуйте поиграть с коэффициентами сначала одного звена **Ka**, а затем другого звена **Kg**, запомните, как влияет изменение определенного коэффициента и стремитесь к коэффициентам, которые улучшают качества балансировки.

Раскомментируйте формулу под цифрой //4, а остальные оставьте закомментированными:

```
//4
Speed = AcYsum  * Ka + Gyro  * Kg + XSpeed;
```

Теперь робот едет и обязательно падает. Дело в том, что мы забыли про ситуацию, когда робот уже имеет некоторую скорость и все равно наклонен. Мы не учитываем текущую скорость движения робота, но ведь все наклоны робота происходят в его собственной статической относительно колес системе координат. Введем в расчет результирующей скорости скорость, с которой робот уже движется – переданную его колесам в прошлом периоде расчета XSpeed. Возможно, придется значительно уменьшить коэффициенты

Ka и **Kg**. Если затрудняетесь с настройкой, верните те коэффициенты, которые были первоначально:

```
Ka = 1200.0 / ( PI );
Kg = 300.0 / ( PI );
```

Фактически мы изменяем не скорость робота, а его ускорение, так как в ситуации наличия у робота ускорения появляется сила инерции, которая направлена против ускорения, и в общем случае вычисляется по формуле:

$Fi = ma$ **, где** Fi **– сила инерции,** m **– масса объекта,** a **– ускорение объекта.**

Теперь робот станет балансировать качественно, но, к сожалению, будет уезжать, и всегда в одну сторону. Это связано с отклонением нуля акселерометра от равновесного состояния робота.

На рисунке 8.12 продемонстрировано наглядно, что происходит с роботом, если его . центр тяжести смещен (сбоку есть груз). В этом случае робота постоянно тянет наклониться вперед в попытке компенсировать наклон, робот движется в сторону наклона. Конечно, реальный робот не так сильно разбалансирован, но даже небольшого отклонения достаточно, чтобы он начал движение, а скорость зависит от величины смещения равновесного положения робота от нулевого значения отклонения гироприбора.

Второй вариант разбалансировки, это неточная установка гироприбора. В нашем роботе MPU6050 установлен на макетной плате и держится на контактах штырьках, которые могут недостаточно точно фиксировать его положение (рис. 8.13). MPU6050 наклонен, робот стремясь вывести угол гироприбора в ноль, наклоняется, начинает падать, пытается компенсировать падение – катится.

Если перед отъезжающим роботом окажется подъем, он сможет остановиться, стоять и балансировать, но одновременно управлять балансировкой и скоростью движения робота мы еще не научились.

Тем не менее, уже создан качественный ПД (пропорционально-дифференциальный регулятор). Пропорциональное звено – по углу наклона, дифференциальное по скорости наклона.

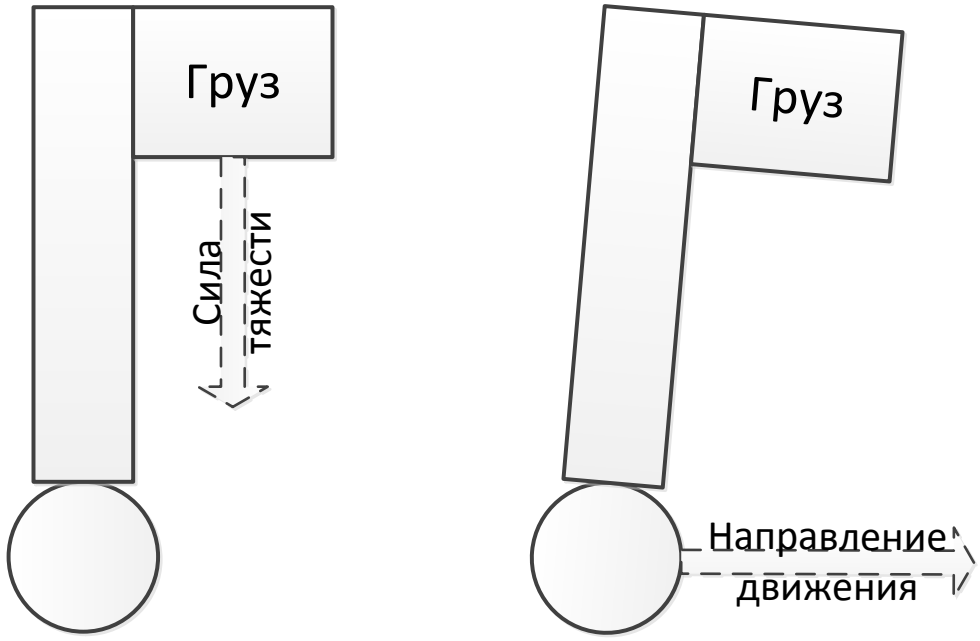


Рисунок 4.12 Робот компенсирует движением смещение своего центра тяжести относительно вертикального положения

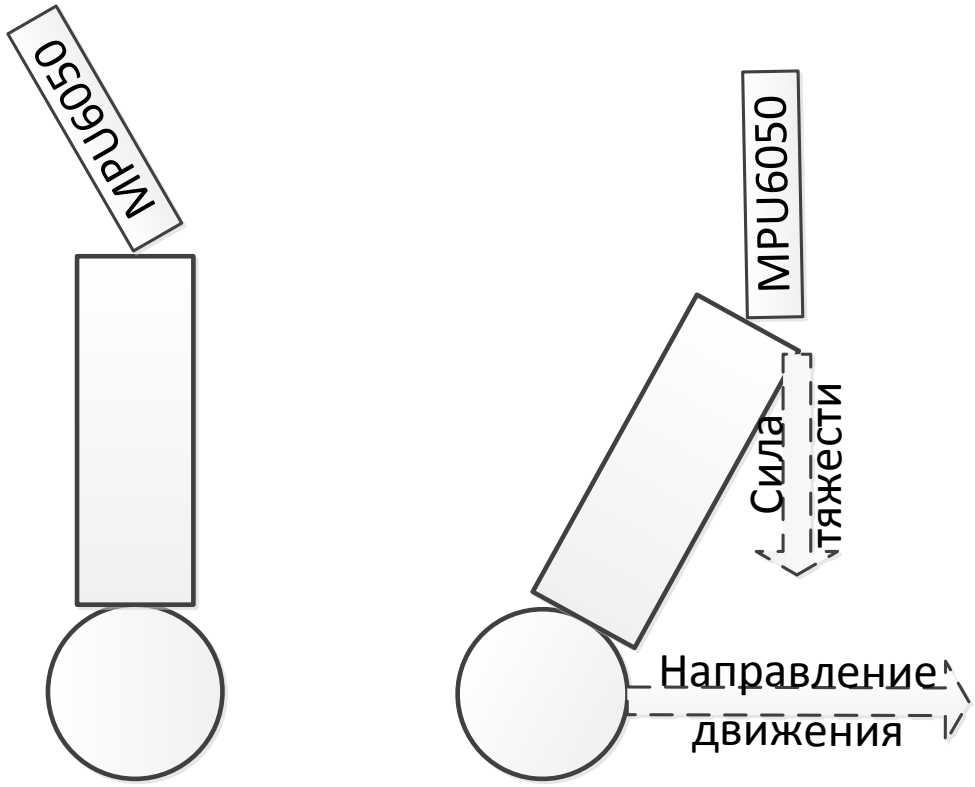


Рисунок 4.13 Робот наклоняется, пытаясь вывести в ноль наклон MPU5060

Листинг 4.2 Подбор коэффициентов балансировочных звеньев (listing_8_2.ino)

```
#include "defin.h"
#include "gyro_acsel.h"
#include "motorstep.h"  //Описание моторов
#include "irq_robot.h"  //Работа с прерываниями

void setup() {
  //Инициализируем моторы
  setup_motor_system();
  //Выключаем моторы.
  motor_off();
  // Инициализируем генерацию шагов моторов
  timer_setup();
  Serial.begin(115200);
  giroscop_setup();
  delay(100);
  Calc_CompensatorZ(4000);
  t0 = micros() - 5000;
  t2 = micros();
  Speed = 0;
}
uint32_t micros_;
void loop()
{
  int32_t speed_L ;
  int32_t speed_R;
  t_period = 5000;
  static int i = 0; // для торможения вывода
  micros_ = micros();
  if (micros_ < t2) return;//прошло 5000мк.сек.
  //Опрос гироприбора:
  Data_mpu6050();
  dt = micros() - t0; // Длительность предыдущего периода регулирования.
  t0 += dt; //Точка начала нового периода регулирования.
  double Dt = double(dt) * 0.000001;
  //Расчет угла по показаниям акселерометра
  Acsel = (atan2(AcX, AcZ)) - PI / 2.0; // * RAD_TO_DEG;
  // скорость угловая В радианах - падения
  Gyro = - (double(GyY) - CompensatorY)  * _1_d_131;

  //Комплементарный фильтр
  AcYsum = ONE_ALFA * (AcYsum + Gyro * Dt) + ALFA * Acsel;
  t2 = t0 + t_period;

  Ka = 1200.0 / ( PI );
  Kg = 300.0 / ( PI);
  //1
  Speed = AcYsum  * Ka;
  //2
  //Speed = Gyro  * Kg;
  //3
  //Speed = AcYsum  * Ka + Gyro  * Kg;
```

```
    //4
    //Speed = AcYsum  * Ka + Gyro  * Kg + XSpeed;

    Speed *= 400.0;

    speed_L = (Speed);
    speed_R = (Speed);
    speed_L = constrain(speed_L, -maxSPEED, maxSPEED);
    speed_R = constrain(speed_R, -maxSPEED, maxSPEED);
    //Speed - это число шагов за 100 секунд
    SetSpeed(speed_L, speed_R);
    //Возвращаем обработанное значение скорости в первоначальную формулу
    XSpeed = (speed_L + speed_R) / 800.0;
}
```

Как остановить робота

На практике, довольно трудно подобрать такое положение гироприбора на роботе, которое точно соответствует центру масс робота. Положение робота, когда ноль гироприбора достигнут, а центр масс смещен вперед или назад, приводит к постоянному стремлению робота наклониться в сторону, которая “тяжелее”. Алгоритм балансировки, рассмотренный в предыдущем разделе, пытаясь привести робота в состояние “нуля” гироприбора, а не в состояние равновесия робота (вектор силы тяжести, действующей на центр масс робота, проходит через ось колес), приводит к тому, что робот постоянно катится в “тяжелую” сторону.

Отсюда задача улучшить алгоритм балансировки и научить робота компенсировать смещение нуля гироприбора для достижения положения, когда “ноль” гироприбора соответствует состоянию равновесия. Это приведет к тому, что движение робота прекратится.

Если робот наклонен, нужно ускорить робота в направлении наклона (рис. 4.14), это приведет к появлению силы инерции, и наклон робота уменьшится.

Обратите внимание, что для компенсации наклона нужно создать именно ускорение, а не скорость! При равномерном движении с любой скоростью сила инерции равна нулю

Усовершенствуем алгоритм балансировки. Для компенсации ошибки согласования нуля гироприбора и равновесия робота создадим два дополнительных звена регулятора, интегральное и пропорциональное.

Интегральное звено

Введем накопительное (интегральное звено) скорости движения робота по времени, иначе пройденной дистанции. Если робот катится в определенную сторону, считаем, что робот имеет ошибку согласования нуля гироприбора и своего равновесного положения.

Робот катится, увеличивается пройденный путь, “растет” значение соответствующего звена регулятора ***Move*Km*** и компенсирует ошибку от звена по углу ***Ang*Kp***:

$$S_n = S_{n-1} + Ang \times Kp + Gy \times Kg + Move \times Km, \qquad (4.4)$$

S_n - расчетная скорость, S_{n-1} - текущая скорость (с которой робот уже движется), ***Ang*** – рассчитанный по гироприбору угол наклона робота, ***Kp*** – коэффициент пропорциональности влияния угла наклона на результат, ***Gy*** – угловая скорость (скорость падения робота по гироскопу), ***Kg*** - коэффициент пропорциональности влияния угловой скорости на результат, ***Move*** – интеграл от скорости передвижения по времени или пройденный роботом путь, ***Km*** – коэффициент пропорциональности влияния пройденного пути на результат.

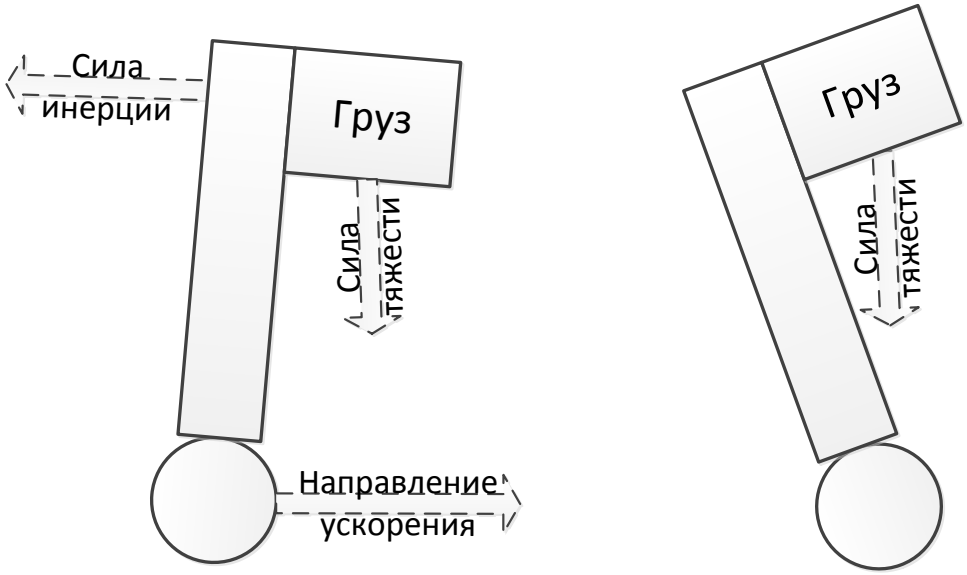


Рисунок 4.14 Робот, компенсирующий смещение своего центра масс

На практике применение формулы 8.14. приведет к колебаниям робота, он проедет определенное расстояние в одну сторону, остановится и поедет в другую. Амплитуда колебаний будет зависеть от ***Km***, и может быть довольно большой (до нескольких метров для нашего робота). Это связано с тем, что пройденный путь будет расти даже тогда, когда робот уже имеет отрицательное линейное ускорение – тормозит. Путь будет расти до момента полной остановки робота, что приведет к перерегулированию, которое роботу придется компенсировать обратным движением.

Пропорциональное звено

Для компенсации колебания от звена ***Move*Km*** введем звено по линейной скорости (производной от пути по времени) ***dMove*Kdm***:

$$S_n = S_{n-1} + Ang \times Kp + Gy \times Kg + Move \times Km + dMove \times Kdm, \qquad (8.5)$$

dMove – линейная скорость робота, рассчитанная по шагам моторов, ***Kdm*** – коэффициент пропорциональности влияния ***dMove*** на результат.

Работает это следующим образом. Робот получает дополнительную корректировку скорости робот от звена по линейной скорости. Причем, чем больше скорость, тем сильнее корректировка скорости (пропорциональное звено). В результат робот останавливается раньше, чем звено ***Move*** переполнится. После остановки звено по линейной скорости соответственно перестает влиять на результат, ведь линейная скорость отсутствует, а вот накопленное значение ***Move*** по дистанции сохранится.

Удобно, если программное обеспечение и моторы робота позволяют задавать не скорость, а ускорение. В этом случае коэффициенты немного изменятся, но в целом суть формулы сохранится:

$$Accel = Ang \times Kp + Gy \times Kg + Move \times Km + dMove \times Kdm, \quad (4.6)$$

Accel – ускорение, с которым должен двигаться робот до следующего опроса гироприбора и перерасчета ускорения (5-10 миллисекунд).

Алгоритм балансировки (4.5) можно еще немного «усилить», если ввести звенья углового ускорения и ускорение по линейной скорости, но делать это нужно взвешенно и осторожно, так как в данные по ускорениям зашумлены. В программе (листинг 4.3) параметры углового ускорения и линейного ускорения интерполируются (используются средние значения за 10 итераций расчета).

Рассмотрим программу (листинг 4.3)**.**

Функцией **setup()** инициализируются шаговые моторы, запускаются прерывания по таймеру для генерации шагов, подключается MPU6050, запускается 4-х секундный цикл пересчета отклонений нуля гироскопа.

В периодически повторяющуюся функцию loop() внесены изменения, которые будут рассмотрены ниже.

Перед расчетом скорости моторов описаны два условных блока, в результате исполнения которых робот может завершить расчет и не пересчитывая скорость уйти на следующую итерация расчета угла, это блоки обработки падения:

if (flag_crash) {…} - если робот упал… при обработке этого блока производится проверка состояния угла наклона, и если он близок к состоянию подъема, включаются моторы и снимается флаг падения (flag_crash).

if (abs(Move) > MoveLimit) {…} - если робот не может сбалансироваться, например, упал, отключаются моторы, обнуляются все расчетные характеристики, робот «отдыхает» 6 секунд, после чего его можно поднять, и он заново начнет балансировать.

XL = counter_stepL; counter_stepL = 0; и XR = counter_stepR; counter_stepR = 0, - получение перемещения по левому и правому мотору робота в шагах от начала предыдущего опроса.

Линейная скорость робота (в шагах) при этом рассчитана по формуле:

dMove = double(XL + XR) * 0.5 / Dt;

Линейное ускорение сглаживается за 10 итераций:

ddMove = ddMove * 0.9 + (dMove - dMoveOld) * 0.1 / Dt;

dMoveOld = dMove;

Перемещение рассчитывается как сумма шагов колес, деленная на 2, шаги за предыдущие периоды складываются (с учетом знака направления вращения):

Move += (XL + XR) / 2.0; //Пройденный путь

Угловое ускорение также рассчитывается за последние 10 итераций:

dGyro = 0.9 * dGyro + 0.1 * ((Gyro - GyroOld) / Dt);

GyroOld = Gyro;

Обновленная скорость рассчитывается по формуле:

Speed = (AcYsum) * Ka + Gyro * Kg + dGyro * Kdg + XSpeed + (dMove) * Kdm + Move * Km + ddMove * Kddm;

Экспериментально подобранные коэффициенты находятся в файле defin.h:

double Km = 0.0325;

double Kdm = 0.03;

double Kddm = 0.00025;

double Ka = 1200.0 / (PI);

double Kg = 300.0 / (PI);

double Kdg = 0.0025;

Чтобы сопоставить расчет **Speed** с используемой при вызове прерывания скорости колеса за 100 секунд, скорость умножается на полученный экспериментально коэффициент:

Speed *= 400.0;

Далее рассчитанные и проверенные на вхождение в установленные лимиты скорости колес передаются на управление моторами:

SetSpeed(speed_L, speed_R);

XSpeed = (speed_L + speed_R) / 800.0 – расчет скорости (S_{n-1} в форм.8.5) робота для следующей итерации расчета.

Листинг 4.3 Робот, научился стоять на месте (listing_8_3.ino)

```
#include "defin.h"
#include "gyro_acsel.h"
#include "motorstep.h"  //Описание моторов
#include "irq_robot.h"  //Работа с прерываниями

void setup() {
  //Инициализируем моторы
  setup_motor_system();
  //Выключаем моторы.
  motor_off();
  // Инициализируем генерацию шагов моторов
  timer_setup();
  Serial.begin(115200);
  giroscop_setup();
  delay(100);
  Calc_CompensatorZ(4000);
  t0 = micros() - 5000;
  t2 = micros();
  Speed = 0;
  flag_crash = true;
  time_stop_move = millis()+3000;
}

uint32_t micros_;
void loop()
{
  int32_t speed_L;
  int32_t speed_R;
  t_period = 5000;
  static int i = 0; // для торможения вывода
  micros_ = micros();
  if (micros_ < t2) return;//Если период < t2 анализ. прошло 5000 если да,
измеряем угловую скорость.
  //Опрос гироприбора:
  Data_mpu6050();
  dt = micros() - t0; // Длительность предыдущего периода регулирования.
  t0 += dt; //Точка начала нового периода регулирования.
  double Dt = double(dt) * 0.000001;
  // Нужно поймать угол, рядом с которым угловая скорость колеблется в районе
нуля.
  //Расчет угла по показаниям акселерометра
  Acsel = (atan2(AcX, AcZ)) - PI / 2.0; // * RAD_TO_DEG;
  // скорость угловая В радианах - падения
  Gyro = - (double(GyY) - CompensatorY)  * _1_d_131;

  //Комплементарный фильтр
  AcYsum = ONE_ALFA * (AcYsum + Gyro * Dt) + ALFA * Acsel;
  t2 = t0 + t_period;

  if ( flag_crash)
  {
```

```
          if(time_stop_move> millis()) return;
          //Если робот поднят после падения
          if (abs(AcYsum * RAD_TO_DEG) < 10)
          {
            motor_on();
            counter_stepR = 0;
            counter_stepL = 0;
            flag_crash = false;
          }
          else return;
        }
        // Если вошли в критичный режим - робот упал
        if (abs(Move) > MoveLimit)
        {
          flag_crash = true;
          XSpeed = 0;
          Speed = 0;
          speed_L = 0;
          speed_R = 0;
          counter_stepR = 0;
          counter_stepL = 0;
          speed_xxx_L = 0;
          newSpeedflag_L  = true;
          speed_xxx_R = 0;
          newSpeedflag_R  = true;
          dGyro = 0;
          Move = 0;
          dMove = 0;
          ddMove = 0;
          dMoveOld = 0;
          motor_off();
          //6 Секунд на стабилизацию после падения
          time_stop_move = millis() + 6000;
          GyroOld = 0;
          return;
        }

        XL = counter_stepL; counter_stepL = 0;
        XR = counter_stepR; counter_stepR = 0;

        dMove = double(XL + XR) * 0.5 / Dt;

        ddMove = ddMove * 0.9 + (dMove - dMoveOld) * 0.1 / Dt;
        dMoveOld = dMove;
        Move += (XL + XR) / 2.0; //Это как раз пройденный путь

        dGyro = 0.9 * dGyro + 0.1 * ((Gyro - GyroOld) / Dt);
        GyroOld = Gyro;

        Speed = (AcYsum)  * Ka + Gyro  * Kg + dGyro * Kdg + XSpeed + (dMove) * Kdm +
Move * Km + ddMove * Kddm;
        Speed *= 400.0;
```

```
    speed_L = (Speed);
    speed_R = (Speed);
    speed_L = constrain(speed_L, -maxSPEED, maxSPEED);
    speed_R = constrain(speed_R, -maxSPEED, maxSPEED);

    //Speed - это число шагов за 100 секунд
    SetSpeed(speed_L, speed_R);
    //Возвращаем обработанное значение скорости в первоначальную формулу
    XSpeed = (speed_L + speed_R) / 800.0;
}
```

Подключаем внешнее управление и подъемный рычаг (окончательная версия программы)

Проведем небольшую модернизацию робота, установив на него сервомотор с рычагом. Цель данного усовершенствования позволить роботу самостоятельно вставать в случае падения. Если у вас под рукой сервомотора не оказалось, не страшно, данная программа будет работать, только робота придется поднимать руками.. Сервомотор (рис. 4.15) используется с фланцем, который имеет отверстие с зубчатой поверхностью для установки на вал сервомотора, и 2мм отверстия по периметру для крепления рычагов винтами или шурупами. Фланец, после установки в сервомотор, фиксируется финтом 3мм. Резиновые вибропрокладки устанавливаются в места крепления сервомотора.

Рисунок 4.15. Сервомотор малой мощности

Фланец сервомотора устанавливаем в подъёмный рычаг и фиксируем шурупами либо винтами по периметру – рисунок 4.16.

Затем подъемный рычаг устанавливаем на робота – рисунок 4.17. Проверяем диапазон движений. Если сервомотор имеет фиксатор предотвращающий прокручивание вала на 360 градусов, добиваемся, чтобы

рычаг опускался в обе стороны на одинаковую величину угла. Если вал сервомотора прокручивается полностью, операцию настройки поворота вала можно будет провести только после электрического подключения и проведения тестов – поворотов в 0 и на 180 градусов.

Рисунок 4.16. Рычаг подъёмный с установленным фланцем

Рисунок 4.17 Установка рычага в сервомотор робота

Робот будет выглядеть, как показано на рисунке 4.18.

Рисунок 8.18. Робот с установленным подъемным рычагом

В балансирующем состоянии рычаг поднят. После падения, когда роботу необходимо подняться, рычаг опускается в сторону, на которой лежит робот, робот приподнимается, после чего включается алгоритм балансировки. Робот заканчивает подъем колесами и начинает балансировать.

Сервомотор, подобный представленному на рисунке 4.15, не должен питаться напряжением более 6 вольт, а в режиме нагрузки может потреблять ток до 1 ампера. Следовательно, его требуется подключать к понижающему стабилизатору. Используем линейный стабилизатор L7805CV. Стабилизатор должен быть отдельным, использование общего стабилизатора для ESP32 и сервомотора приводит к нестабильной работе контроллера ESP32. Логически сервомотор будет управляться от 19GPIO.

Подключаем сервомотор согласно рисунку 4.19.

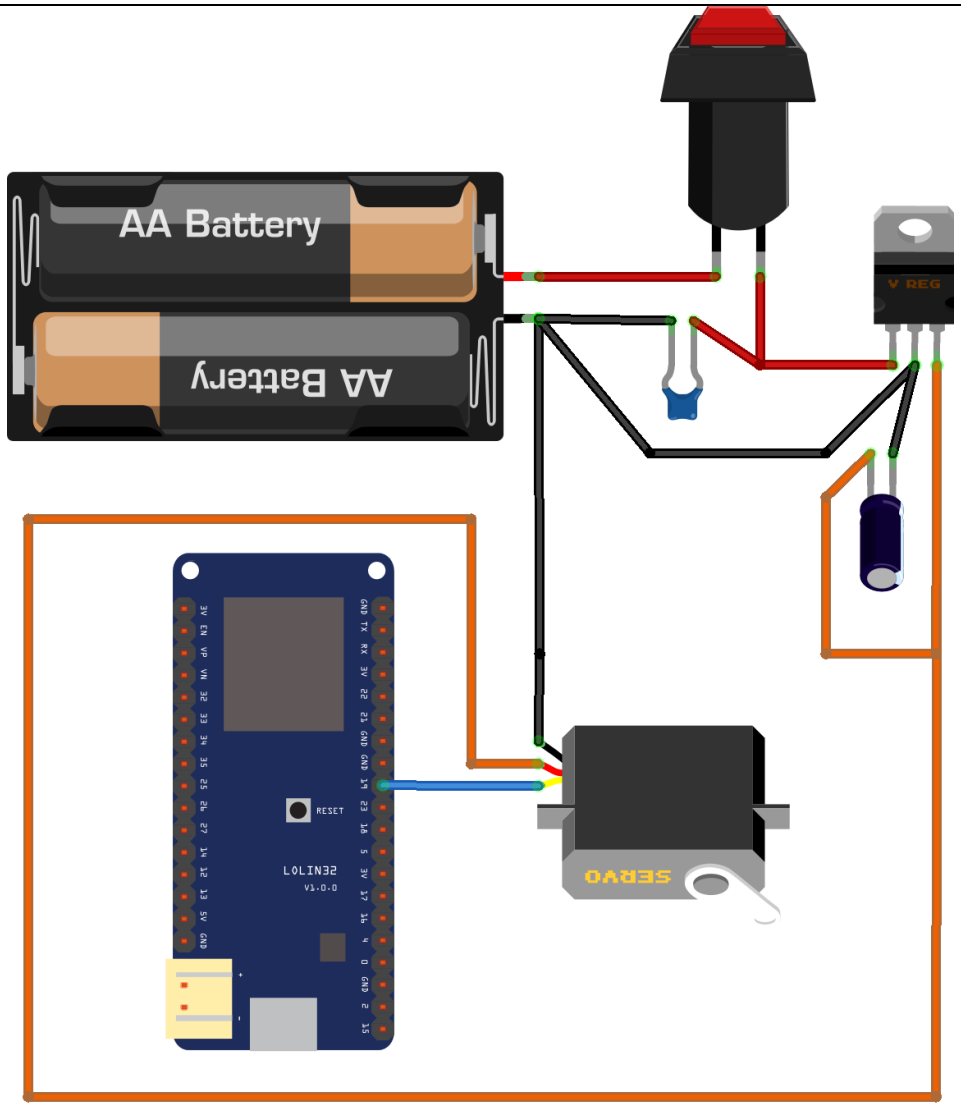


Рисунок 4.19. схема подключения питания и управления подъемного рычага

Расчеты

Для того чтобы робот смог двигаться, нужно создать управляемое отклонение от балансного состояния в нужную сторону, но из формулы 8.5 любое отклонение будет сразу компенсироваться звеньями: интегральным по скорости $Move \times Km$ и и пропорциональным скорости $dMove \times Kdm$.

Если внести отклонения именно при расчете данных звеньев на определенную величину, то компенсация отклонения будет не полной, и робот покатится.

Было:

$$Move+=(S_{n-1})\times Dt\,,\quad dMove=S_{n-1}.$$

Стало

$$Move+=(S_{n-1}-Sz)\times Dt\,,\qquad dMove=S_{n-1}-Sz\,,\qquad (4.7)$$

$$S_n=S_{n-1}+Ang\times Kp+Gy\times Kg+Move\times Km+dMove\times Kdm\,,$$

где ***Sz*** – заданная скорость движения. Изменения расчета данных звеньев на величину ***Sz***, которая задает скорость движения в шагах шаговых моторов, приведут к управляемому (с заданной скоростью) движению робота.

Для поворотов робота нужно скорость правого колеса увеличить на величину скорости поворота, а левого уменьшить на такую же величину или наоборот, важно чтобы при этом не менялось общее изменение скорости робота, заданное формулой балансировки 8.5. Если сложить скорости обоих моторов и поделить на 2 должна получиться скорость, рассчитанная для балансировки.

Реализуем это в программе.

Программная часть робота с удаленным управлением и рычагом

Программа состоит из 7 файлов (рис.4.21):

Listing_8_4.ino – инициализирующая часть и основная функция loop(), в которой производится расчет углов наклона робота, рассчитывается ПИД-регулятор, подается команда на смену скоростей моторов;

defin.h – описание переменных;

gyro_acsel.h – обслуживание MPU6050;

irq_robot.h – обработка прерываний по таймеру, генерация шагов моторов;

motorstep.h – обслуживание шаговых моторов, инициализация, включение, отключение;

move_case.h – обработка команд от смартфона на изменение движения робота;

servo_hand.h – инициализация, функция подъема и опускания «серворуки».

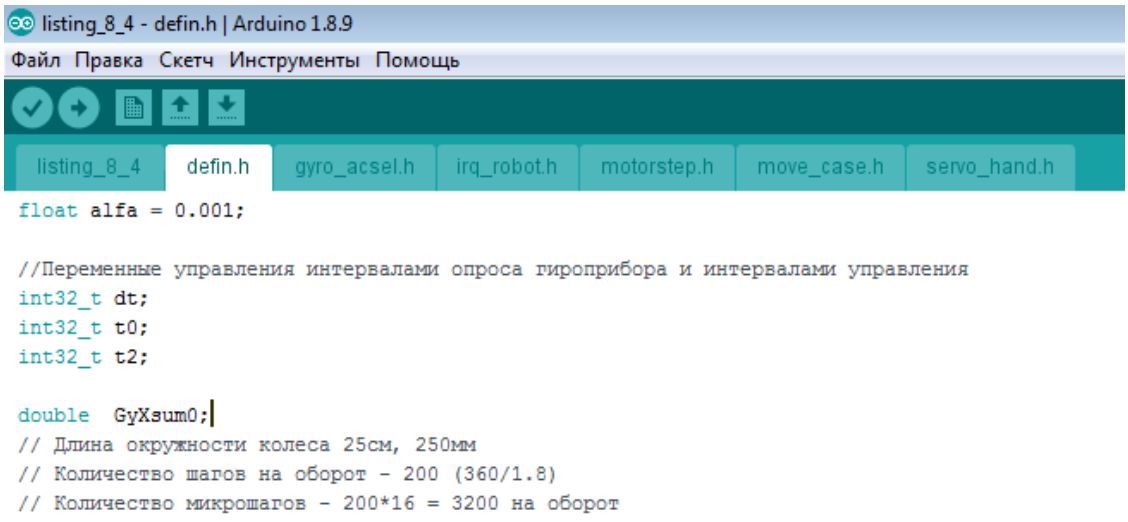


Рисунок 4.21. Программа в Arduino IDE

Рассмотрим только те файлы, которые претерпели серьезные изменения.

Листинг 4.4.1. (файл defin.h) содержит описание глобальных переменных и некоторых констант. Основная цель его создания – выделение переменных общего назначения в отдельный файл для повышения читаемости головного файла программы.

Листинг 4.4.1. Глобальные переменные и константы (файл defin.h)

```
float alfa = 0.001;
//Переменные управления интервалами опроса гироприбора и интервалами управления
int32_t dt;
int32_t t0;
int32_t t2;
double  GyXsum0;
// Длина окружности колеса 25см, 250мм
// Количество шагов на оборот - 200 (360/1.8)
// Количество микрошагов - 200*16 = 3200 на оборот
int32_t t_period;
double CommandSpeed = 0; //Скорость управляемого движения на 100 сек.
double OldCommandSpeed = 0; //Скорость управляемого движения из предыдущего опроса
double Turn = 0; //Поворот.
volatile int32_t counter_stepL; //Счетчик шагов левого мотора
volatile int32_t counter_stepR; //Счетчик шагов правого мотора
bool flag_crash = true;
bool flag_time_standup = true;// Флаг того, что робот встает.
uint32_t time_standup = 0; // Время на подъем робота
double XSpeed = 0; //Реальная скорость по шагам в моторах
double  OldXSpeed = 0; //Реальная скорость по шагам в моторах
//Нужно его использовать
int32_t XL, XR; //Счетчик тактов на колесо в текущей итерации
int32_t OldCommandSpeed_dMove = 0;
bool flag_to_off_robot  = false; //Флаг выключения робота
int32_t time_to_off_robot = 0; //Время на присадку робота
uint32_t time_stop_move=0;

double  Km = 0.0325;
double  Kdm = 0.03;
double  Kddm = 0.00025;
double  Ka = 1200.0 / ( PI );
double  Kg = 300.0 / ( PI);
double  Kdg = 0.0025;
// Предел интегрального звена Move,
// накопление больше которого считается, что робот упал
double MoveLimit = 0.5*Ka/Km;

double Speed = 0;
double OldSpeed = 0;
double OldGyro = 0;

double Move = 0; //Пройденный путь.
double dMove = 0;
double ddMove = 0;
double dMoveOld = 0;
double dGyro = 0;
double GyroOld = 0;
```

Листинг 4.4.2. (файл move_case.h) содержит код обработки команд управления роботом от смартфона.

MOVE_PERIOD и time_start_move – (переменные) отслеживание потери связи со смартфоном для остановки робота.

MOVSPEED0 – MOVSPEED10 (константы) для задания максимальной скорости робота.

ASPEED – (переменная) хранит текущую максимальную скорость управляемого движения.

BT_input() – функция анализирует наличие принятой от смартфона команды, при наличии в операторе **switch(bt_input)** выбирается действие в зависимости от принятой команды. Благодаря функции **constrain(значение, нижний лимит, верхний лимит)** ограничивающей значения определенным диапазоном. Скорости изменяются не на максимальную возможную величину, а на приращение **DeltaSpeed_100sec**, следующее приращение только через 50миллисекунд (кода придет следующая команда).

Листинг 4.4.2 Обработка внешних команд (файл move_case.h)

```
uint32_t MOVE_PERIOD = 300;
uint32_t   time_start_move = 0;
const int32_t  MOVSPEED0 = (maxSPEED/100)*3;
const int32_t  MOVSPEED1 = (maxSPEED/100)*5;
const int32_t  MOVSPEED2 = (maxSPEED/100)*10;
const int32_t  MOVSPEED3 = (maxSPEED/100)*15;
const int32_t  MOVSPEED4 = (maxSPEED/100)*20;
const int32_t  MOVSPEED5 = (maxSPEED/100)*25;
const int32_t  MOVSPEED6 = (maxSPEED/100)*30;
const int32_t  MOVSPEED7 = (maxSPEED/100)*35;
const int32_t  MOVSPEED8 = (maxSPEED/100)*40;
const int32_t  MOVSPEED9 = (maxSPEED/100)*45;
const int32_t  MOVSPEED10 = (maxSPEED/100)*50;
int32_t ASPEED = MOVSPEED2;
int32_t DeltaSpeed_100sec = 30000;

bool BT_input()
{
  static char bt_input;
  if (SerialBT.available())
  {
    bt_input = (char)SerialBT.read();
  }
  else
  {
    if(time_start_move < millis())
    {
     // CommandSpeed = 0;
//Обнуляем скорость управляемого движения.
     if(CommandSpeed>0) CommandSpeed = constrain(CommandSpeed-
DeltaSpeed_100sec,0,CommandSpeed);
```

```
     //Обнуляем скорость управляемого движения.
          if(CommandSpeed<0) CommandSpeed =
constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,0);
     //Обнуляем скорость управляемого движения.
           Turn = 0;
         }
         return false;
       }
       time_start_move = millis() + MOVE_PERIOD;

       //====================

       switch (bt_input)
       {
         case 'S': // вперед
     //      CommandSpeed = 0;
     //Обнуляем скорость управляемого движения.
           if(CommandSpeed>0) CommandSpeed = constrain(CommandSpeed-
DeltaSpeed_100sec,0,CommandSpeed);
     //Обнуляем скорость управляемого движения.
           if(CommandSpeed<0) CommandSpeed =
constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,0);
     //Обнуляем скорость управляемого движения.

           Turn = 0;
           break;
         case 'F': // вперед
           Turn = 0;
           CommandSpeed = constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,
ASPEED);
           break;
         case 'B': //назад
           CommandSpeed = constrain(CommandSpeed-DeltaSpeed_100sec,-
ASPEED,CommandSpeed);
           Turn = 0;
           break;
         case 'L':  //Влево на месте
           Turn = -ASPEED / 2;
           //Обнуляем скорость управляемого движения.
           if(CommandSpeed>0) CommandSpeed = constrain(CommandSpeed-
DeltaSpeed_100sec,0,CommandSpeed);
           if(CommandSpeed<0) CommandSpeed =
constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,0);
           break;
         case 'R':  //Вправо на месте
           Turn = ASPEED / 2;
           //Обнуляем скорость управляемого движения.
           if(CommandSpeed>0) CommandSpeed = constrain(CommandSpeed-
DeltaSpeed_100sec,0,CommandSpeed);
           if(CommandSpeed<0) CommandSpeed =
constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,0);
           break;
         case 'G':  // Влево вперед
           //CommandSpeed = ASPEED; //
```

```
            CommandSpeed = constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,
ASPEED);
            Turn = -ASPEED / 2;
            break;
          case 'I':  //Вправо вперед
            //CommandSpeed = ASPEED; //
            CommandSpeed = constrain(CommandSpeed+DeltaSpeed_100sec,CommandSpeed,
ASPEED);
            Turn = ASPEED / 2;
            break;
          case 'H':  //Вправо
             break;
          case 'J':  //Влево
            break;
          case '0':
            ASPEED = MOVSPEED0;
            break;
          // Скорость 10%
          case '1':
            ASPEED = MOVSPEED1;
            break;
          // Скорость 20%
          case '2':
            ASPEED = MOVSPEED2;
            break;
          // Скорость 30%
          case '3':
            ASPEED = MOVSPEED3;
            break;
          // Скорость 40%
          case '4':
            ASPEED = MOVSPEED4;
            break;
          // Скорость 50%
          case '5':
            ASPEED = MOVSPEED5;
            break;
          // Скорость 60%
          case '6':
            ASPEED = MOVSPEED6;
            break;
          // Скорость 70%
          case '7':
            ASPEED = MOVSPEED7;
            break;
          case '8':
            // Скорость 80%
            ASPEED = MOVSPEED8;
            break;
          // Скорость 90%
          case '9':
            ASPEED = MOVSPEED9;
```

```
        break;
      // Скорость 100%
      case 'q':
        ASPEED = MOVSPEED10;
        break;
      case 'V':
        break;
      case 'v':
        break;
      case 'X':
        break;
      case 'x':
        break;
      default:
        break;
    }
}
```

Листинг 4.4.3. (файл servo_hand.h) подключается библиотека ESP32Servo.h, ее предварительно требуется установить в систему из внутреннего репозитария Arduino IDE – рисунок 8.22.

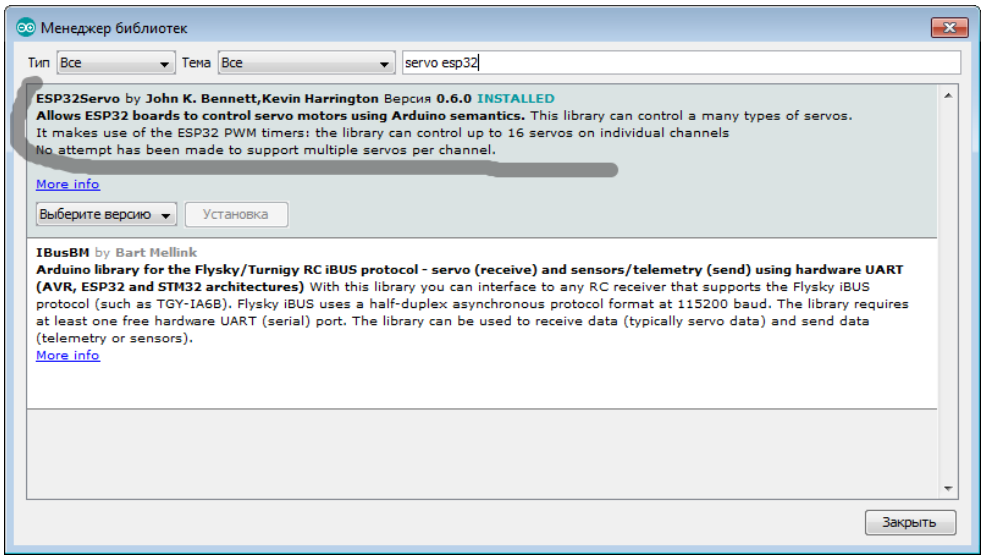


Рисунок 4.22. Установка библиотеки ESP32Servo из репозитария

Создаем объект one_hand типа Servo, переменные minUs и maxUs задают ширину импульсов для управления ШИМ сервопривода, one_handGPIO – GPIO управления.

Функция servo_hand_setup() инициализирует сервопривод.

Функция servo_hand_bottom(double test_angle) проверяет, в какую сторону наклонен робот и поворачивает рычаг в попытке подняться.

Функция servo_hand_up() - если робот поднят, данная функция поднимает серворычаг и через 4 секунды отключает сервопривод (экономия заряда).

Листинг 4.4.3 Управление рычагом подъема (файл servo_hand.h)

```
#include <ESP32Servo.h>
Servo one_hand;
// Published values for SG90 servos; adjust if needed
int minUs = 800;
int maxUs = 2500;
int one_handGPIO = 19;
bool one_handattach = false;

void servo_hand_setup()
{
  one_hand.setPeriodHertz(50);      // Standard 50hz servo
  //one_hand.attach(one_handGPIO, minUs, maxUs);
  //one_hand.write(90);
  one_handattach = false;
}

void servo_hand_bottom(double test_angle)
{
  if (test_angle < - 20.0) //Если вошли в критичный режим
  {
    if (one_handattach == false)
    {
      one_handattach = true;
      one_hand.attach(one_handGPIO, minUs, maxUs);
    }
    one_hand.write(200);
  }
  else if (test_angle >  20.0) //Если вошли в критичный режим
  {
    if (one_handattach == false)
    {
      one_handattach = true;
      one_hand.attach(one_handGPIO, minUs, maxUs);
    }
    one_hand.write(0);
  }
}
void servo_hand_up()
{
  static uint32_t timess = 0;

  if (one_hand.attached())
  {
    if (one_hand.read() != 100)
    {
      one_hand.write(100);
      timess = millis() + 4000;
    }
    else
    {
      if (timess < millis())
```

```
        {
          one_hand.detach();
          one_handattach = false;
        }
      }
    }
  }
```

Листинг 4.4.4. (файл listing_4_4.ino)

В первых строках подключается библиотека BluetoothSerial.h и объявляется объект SerialBT типа BluetoothSerial, тем самым создается возможность принимать и получать данные по Bluethooth, работая как с Serial-интерфейсом.

Далее к головному модулю подключаются остальные файлы проекта по директиве #include.

Функция setup(): инициализирует сервомотор, шаговые моторы; запускает прерывание по таймеру (каждые 10 микросекунд); открывается Serial порт на скорости 115200 бит в секунду; открывается SerialBT порт и роботу в среде Bluethooth присваивается имя BHV_M; запускается расчет (на 4 секунды) отклонений нуля гироскопа (нельзя трогать робота в это время); устанавливаются значения в переменные времени для отслеживания длительности периода перерасчета скорости.

Функция loop() автоматически повторяется в бесконечном цикле. Вначале проверяется, подошло ли время для начала расчета (5000 микросекунд), если подошло, производится расчет угла наклона робота по описанному ранее алгоритму (комплементарный фильтр). Далее робот проверяется на падение (аналогично программе listing_8_3) отличие в том, что в случае падения робот включает серворычаг и пытается встать (функция servo_hand_bottom(AcYsum*RAD_TO_DEG)). Далее идет расчет скорости на следующий период по формуле 8.5 с учетом дополнений 8.7 на движение.

Строки:

```
speed_L = (Speed + Turn);
speed_R = (Speed - Turn);
```

служат для организации поворота робота.

Вызов функции SetSpeed(speed_L, speed_R), приводит к установки флагов наличия новых скоростей, эти флаги проверяются в функции обработки прерывания по таймеру, и при необходимости робот начинает изменять свою скорость.

Листинг 4.4.4 головной модуль (файл listing_8_4.ino)

```
#include <BluetoothSerial.h>
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
```

```
#endif
BluetoothSerial SerialBT;

#include "defin.h"
#include "gyro_acsel.h"
#include "motorstep.h"  //Описание моторов
#include "irq_robot.h"  //Работа с прерываниями
#include "move_case.h"  //Реагирование на команды
#include "servo_hand.h"  //Подъемный рычаг

void setup() {
  //Инициализация рычага
  servo_hand_setup();

  //Инициализируем моторы
  setup_motor_system();
  //Выключаем моторы.
  motor_off();
  // Инициализируем генерацию шагов моторов
  timer_setup();

  Serial.begin(115200);
  Serial.println();

  SerialBT.begin("BHV_M"); //Имя робота в BT
  giroscop_setup();
  delay(100);
  Calc_CompensatorZ(4000);
  t0 = micros() - 5000;
  t2 = micros();
  Speed = 0;
  flag_crash = true;
  time_start_move = millis();
  time_stop_move = millis();
}

uint32_t micros_;
void loop()
{
  int32_t speed_L ;
  int32_t speed_R;
  t_period = 5000;
  static int i = 0; // для торможения вывода
  micros_ = micros();
  if (micros_ < t2) return;//Если период < t2 анализ. прошло 5000 если да,
измеряем угловую скорость.
  BT_input();
  //Опрос гироприбора:
  Data_mpu6050();
  dt = micros() - t0; // Длительность предыдущего периода регулирования.
  t0 += dt; //Точка начала нового периода регулирования.
  double Dt = double(dt) * 0.000001;
```

```
        // Нужно поймать угол, рядом с которым угловая скорость колеблется в районе
нуля.
        //Расчет угла по показаниям акселерометра
        Acsel = (atan2(AcX, AcZ)) - PI / 2.0; // * RAD_TO_DEG;
        // скорость угловая В радианах - падения
        Gyro = - (double(GyY) - CompensatorY)  * _1_d_131;

        //Комплементарный фильтр
        AcYsum = ONE_ALFA * (AcYsum + Gyro * Dt) + ALFA * Acsel;
        t2 = t0 + t_period;

        if ( flag_crash)
        {

          if ((time_stop_move - 3000) < millis())
          {
            //запускаем подъемный рычаг
            servo_hand_bottom(AcYsum * RAD_TO_DEG);
          }
          if ((time_stop_move - 1000) < millis())
          {
            if (abs(AcYsum * RAD_TO_DEG) < 40) //Если поднят
            {
              motor_on();
            }
            if (time_stop_move  < millis())
            {
              if (abs(AcYsum * RAD_TO_DEG) < 40) //Если поднят
              {
                flag_crash = false;
                counter_stepR = 0;
                counter_stepL = 0;
              }
              else {
                //Move = MoveLimit * 2;
              }
            }
          }
          counter_stepR = 0;
           counter_stepL = 0;
          return;
        }
        // Если вошли в критичный режим - , робот упал, но из-за ошибки
        // гироскопа показал достаточное отклонение.
        if (abs(Move) > MoveLimit) //Если вошли в критичный режим)
        {
          flag_crash = true;
          XSpeed = 0;
          Speed = 0;
          OldCommandSpeed = 0;
          CommandSpeed = 0;
          speed_L = 0;
          speed_R = 0;
```

```
          counter_stepR = 0;
          counter_stepL = 0;
          speed_xxx_L = 0;
          newSpeedflag_L  = true;
          speed_xxx_R = 0;
          newSpeedflag_R  = true;
          dGyro = 0;
          Move = 0;
          dMove = 0;
          ddMove = 0;
          dMoveOld = 0;
          motor_off();
          Turn = 0;
          time_stop_move = millis() + 6000; //Секунда на стабилизацию
          GyroOld = 0;
          servo_hand_up();
          return;
        }
        if ((time_stop_move + 2000) < millis()) servo_hand_up();

        XL = counter_stepL; counter_stepL = 0;
        XR = counter_stepR; counter_stepR = 0;

        double dMoveX = double(XL + XR) * 0.5 / Dt;
        OldCommandSpeed_dMove = OldCommandSpeed;

        dMove = dMoveX - OldCommandSpeed_dMove;
        ddMove = ddMove * 0.9 + (dMove - dMoveOld) * 0.1 / Dt;
        dMoveOld = dMove;
        Move += (XL + XR) / 2.0 + Dt * ( - OldCommandSpeed); //Это как раз
пройденный путь

        dGyro = 0.9 * dGyro + 0.1 * ((Gyro - GyroOld) / Dt);
        GyroOld = Gyro;

        Speed = (AcYsum)  * Ka + Gyro  * Kg + dGyro * Kdg + XSpeed + (dMove) * Kdm +
Move * Km + ddMove * Kddm;
        OldCommandSpeed = CommandSpeed / 100.0;
        Speed *= 400.0;

        speed_L = (Speed + Turn);
        speed_R = (Speed - Turn);
        speed_L = constrain(speed_L, -maxSPEED, maxSPEED);
        speed_R = constrain(speed_R, -maxSPEED, maxSPEED);

        //Speed - это число шагов за 100 секунд
        SetSpeed(speed_L, speed_R);
        //Возвращаем обработанное значение скорости в первоначальную формулу
        XSpeed = (speed_L + speed_R) / 800.0;
      }
```

Мы рассмотрели пример поддержания роботом равновесия, программа непростая, но надеемся, что вы справились. Чтобы упростить понимание, к

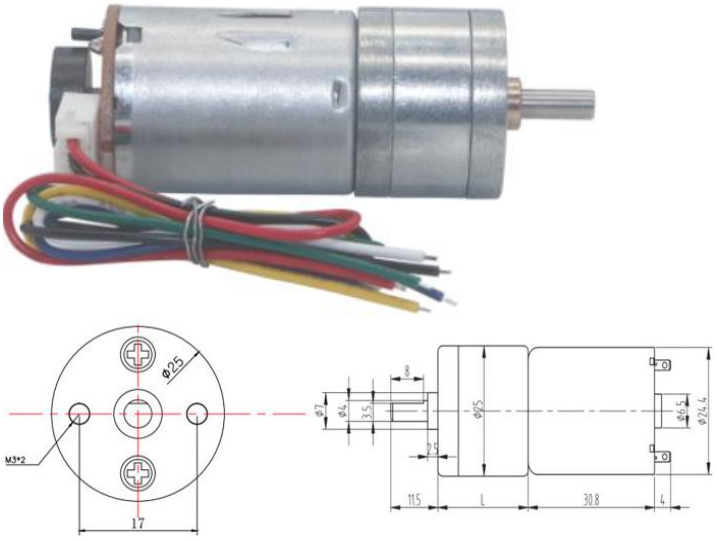
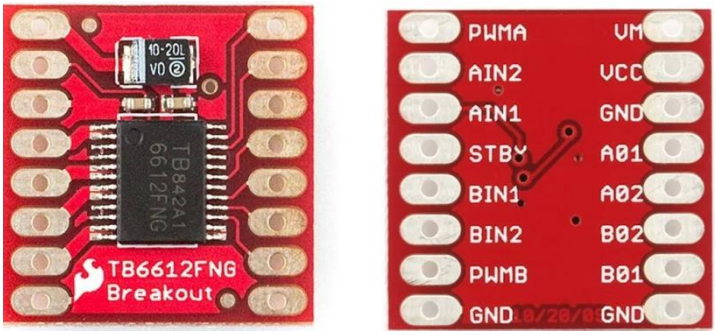
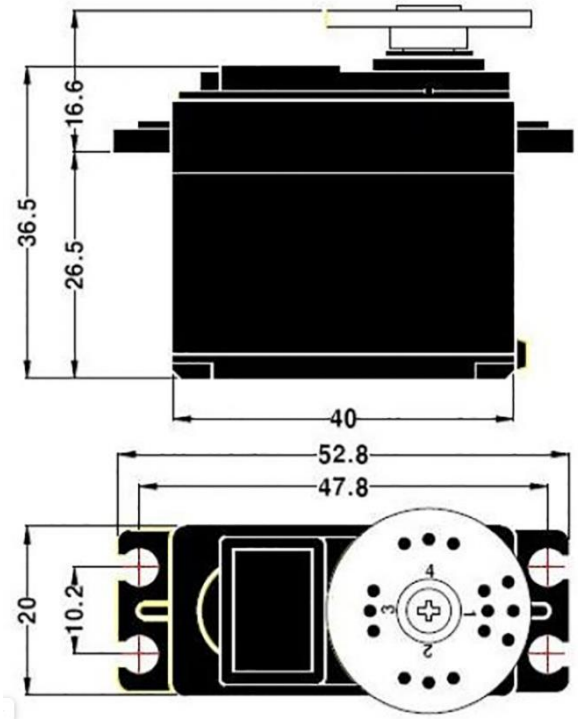
выходу книги будет выпущен учебный ролик на канале Youtube: https://www.youtube.com/user/momotmvu. В ролике дополнительно будут рассмотрены вопросы настройки балансирующего робота по книге.

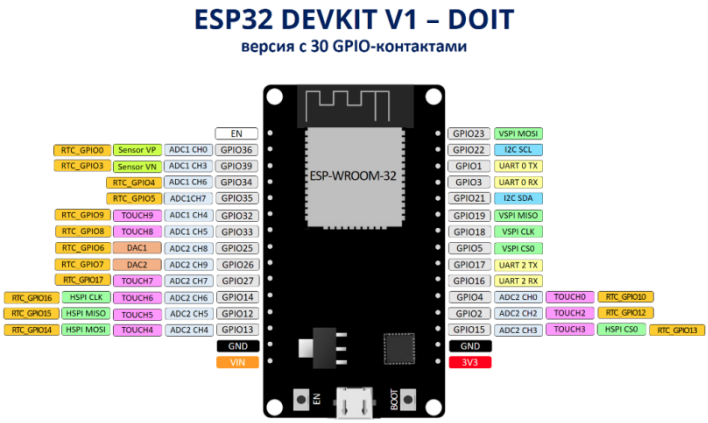
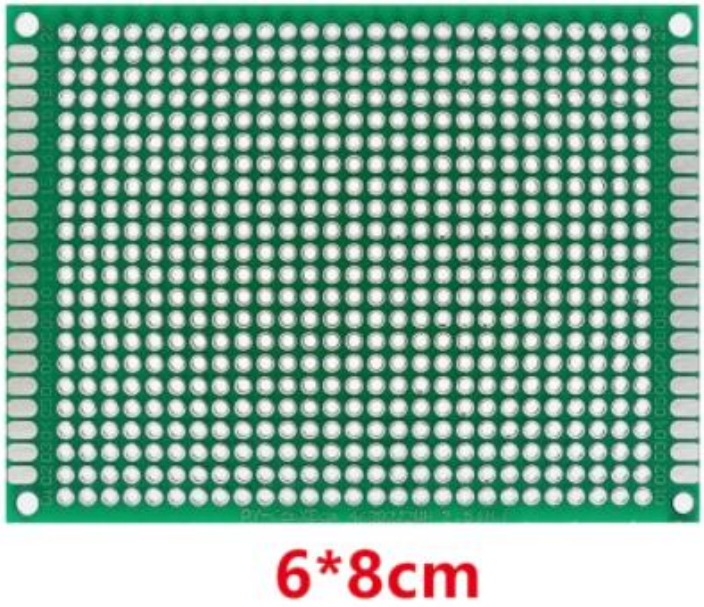
Надеемся, что вы смогли понять основы регулирования, в чем отличия пропорционального, интегрального и дифференциального звена. Подобные регуляторы применяются не только в балансировке, но и для поддержания скорости и направления движения, для поддержания температуры.

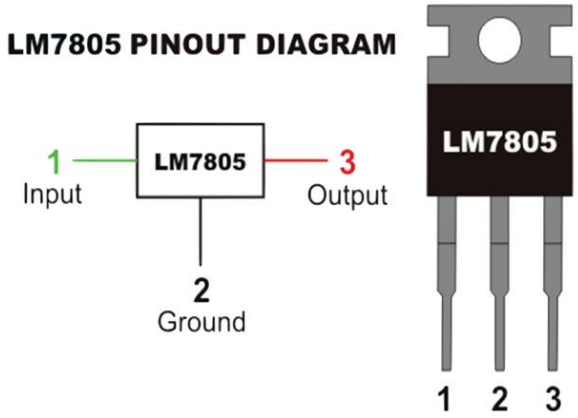
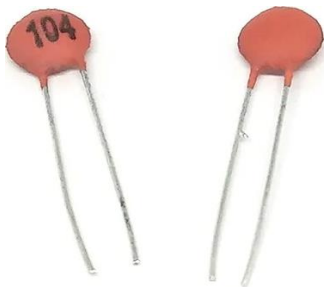
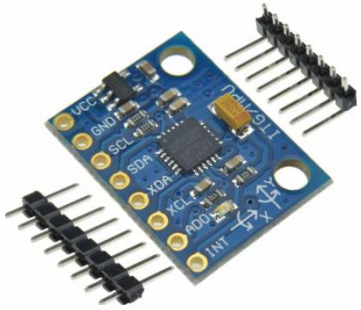
Видеоматериал: Сборка робота на esp 32

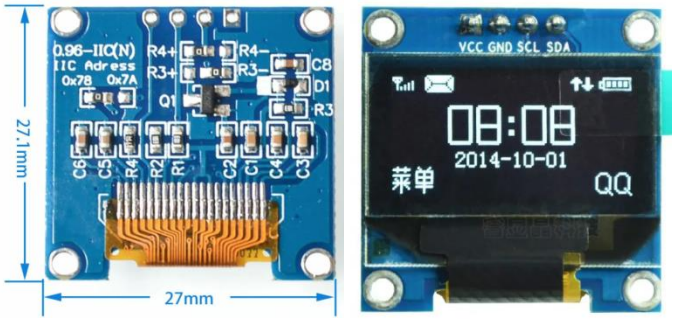
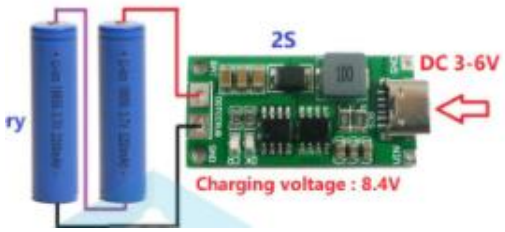
Видеоматериал: настройка ПИД регулятора балансирующего робота

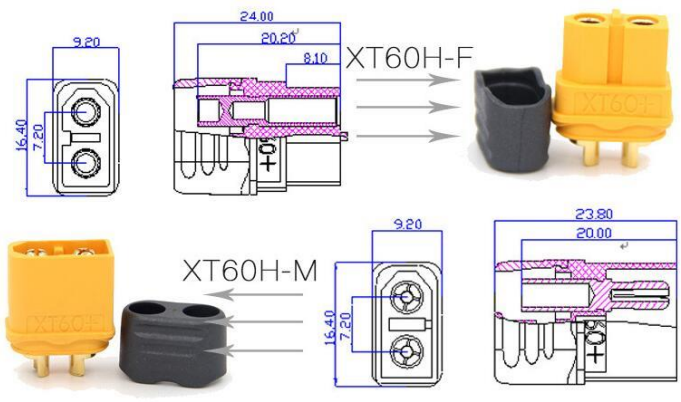
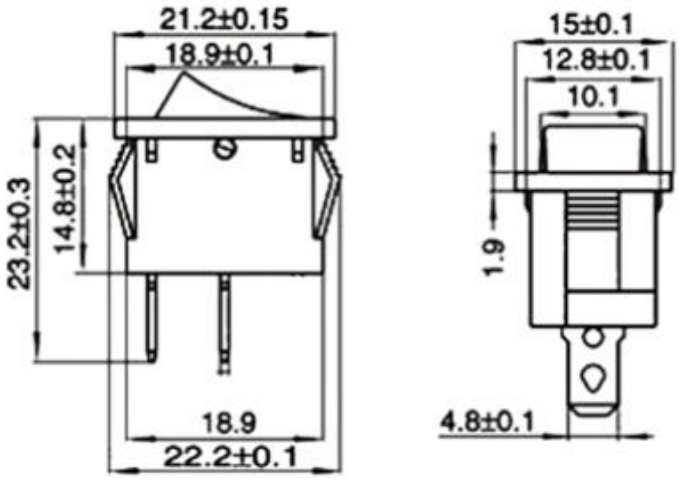
Спецификация конструктора

Компонента	Чертеж и/или фото	Количество
Мотор редукторный с энкодером JGA25-370 DC6V280RPM. Коллекторный мотор 6 вольт со стальным редуктором, редуктор должен поддерживать свободное вращение передаточное отношение 20:1~25:1, не допускается применение червячных редукторов. В задней части мотора на вал установлен энкодер двунаправленный, рекомендованы энкодеры на эффекте Холла или закрытые от пыли оптические энкодеры. В передней части редуктора должны присутствовать крепежные отверстия с резьбой для винта М3		2
Драйвер моторов на основе TB6612FNG. Драйвер должен быть установлен на плате для прототипирования с распаянными контактами под макетную плату.		1
Сервомотор с металлическим редуктором MG995 или MG996 13 кг 15 кг на см, Операционной Скорость: 0.17sec / 60 градусов (4,8 V без нагрузки), Операционной Скорость: 0.13sec / 60 градусов (6,0 V без нагрузки), Крутящий момент, крутящий момент: 13 кг-см (180,5 унций) при температуре не выше 4,8 V, Крутящий момент, крутящий момент: 15 кг-см (208,3 унций) на 6V, Напряжение: 4,8-7,2 Вольт		2

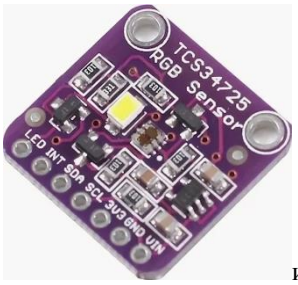
Компонента	Чертеж и/или фото	Количество
Контроллер esp32 esp-wroom32 на плате для макетирования, количество контактов на плате 30, разъем программирования micro-usb или type-c. На плате присутствуют стабилизатор питания AMS1117 -3.3 , конвертер usb-uart.		1
Плата стеклотекстолит для прототипирования двусторонняя с облуженными отверстиями с шагом 2.54 мм по всей поверхности размер платы 6x8см. Толщина 2 мм		1
Линейный стабилизатор с малым падением напряжения AMS1117 - 3.3v		1

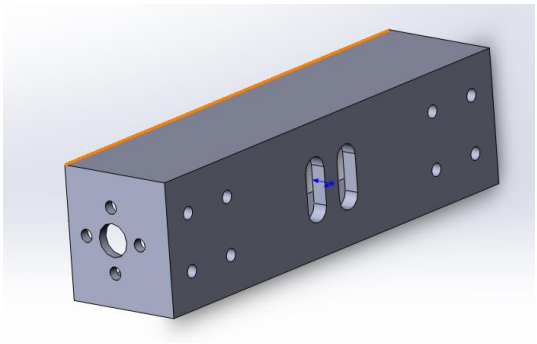
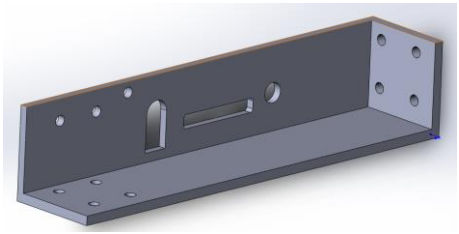
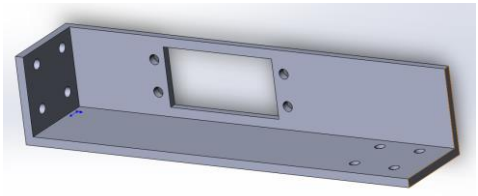
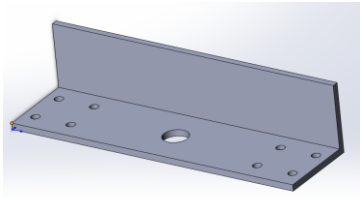
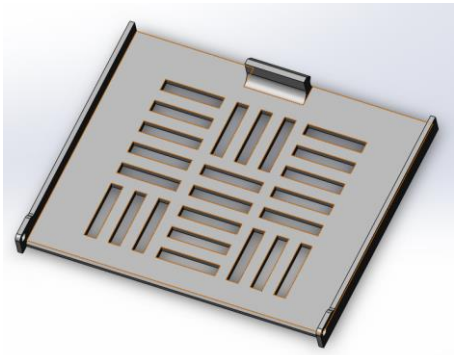
Компонента	Чертеж и/или фото	Количество
Стабилизатор напряжения L7805-5v в корпусе TO-220		1
Конденсатор электролитический 16v 100мкФ		5
Конденсатор керамический 0.1мкФ (104)		2
Диод Шоттки 1A 30V 1N5818 или 1N5819 в корпусе для ручного монтажа		5
Электронный акселерометр и гироскоп MPU6050 на плате для прототипирования GY-521		1

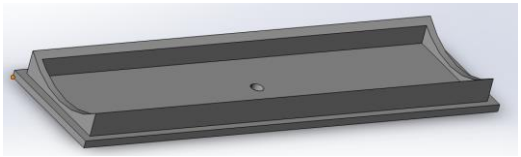
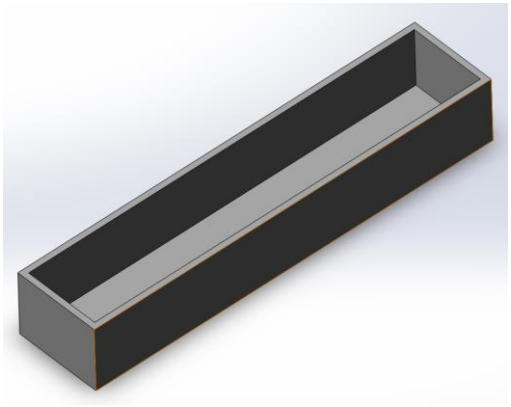
Компонента	Чертеж и/или фото	Количество
Дисплей информационный жидкокристаллический графический OLED 128x64, SSD1306 монохромный, интерфейс I2C, напряжение питания 2.8~5вольт.		33
Аккумулятор Li-Io форм-фактора 18650 3.7вольт емкостью не менее 2000мАчас, внутреннее сопротивление при полном заряде не более 50миллиОм. К аккумулятору приварены контактные площадки.		2
Модуль зарядки литиевой батареи 2S 7,4 В 8,4 в, зарядная плата 5 в 2 А до 8,4 в, зарядка двух батарей. Зарядка от micro-usb или type-c.		1
Провод медный многожильный 24AWG Диаметр проводника: 0,511мм Площадь сечения проводника: 0,205 мм^2 Наружный диаметр: 1.6 мм Сопротивление: 95 Ом/км Допустимый ток: 5 А Вес: 4 г/м Проводник: Медь луженая Изоляция: силикон (кремнейорганическая резина)	Цвет черный	0.5 метров
	Цвет красный	0.5 метров

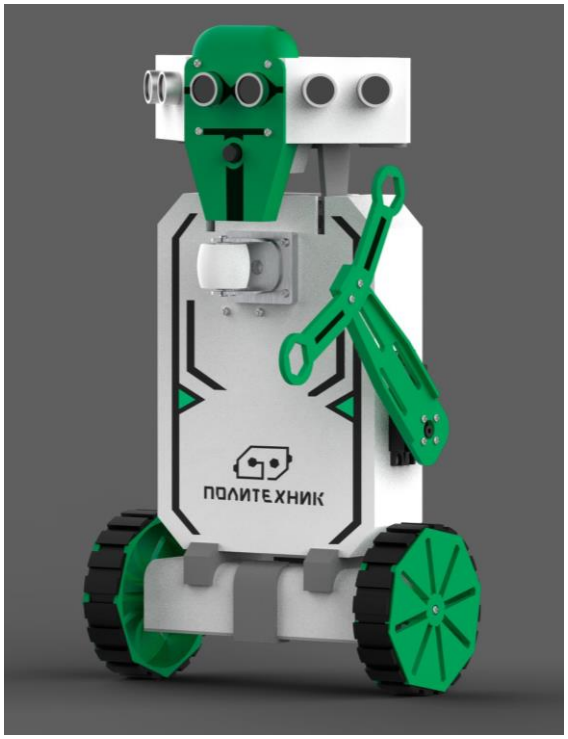
Компонента	Чертеж и/или фото	Количество
В комплекте: разъем электрический XT60 "мама/папа" - 1 пара. Технические характеристики: номинальный ток: 65 А; диаметр контактного элемента: 3,5 мм; корпус: нейлоновый, тугоплавкий; вес: 6,7 г. Наличие нейлонового колпачка с отверстиями под провода		1
JST SM4 светодиодный разъем "мама/папа" -1 пара с припаянными проводами разного цвета Разъем с защелками на четыре ножки Длина провода не менее 150 мм Провод: 22AWG		5
4pin JST SM3 светодиодный разъем "мама/папа" -1 пара с припаянными проводами разного цвета Разъем с защелками на три ножки Длина провода не менее 150 мм Провод: 22AWG		5
Мини-кулисный переключатель от 3А, 10 Вольт. В пластиковом корпусе. Размер 10х15мм с защелкой.		33

Компонента	Чертеж и/или фото	Количество
		
Колесо поворотное диаметр колеса D-25 мм, пластиковое на площадке, толщина колеса 13 мм, высота в сборе с крепежной площадкой не более 40 мм.		33
Винт М3 длина 10 мм шляпка плоская под крестовую отвертку		20
Гайка М3		20
Гайка закладная М3 х 8 х 4.2 мм. Материал: латунь		15

Компонента	Чертеж и/или фото	Количество
Клей/герметик изолирующий Т-7000		10мл
Датчик ультразвуковой дальномер US-025 *Характеристики:* Рабочее напряжение: 3 - 5.5В Рабочий ток: 5.3мА Дальность измерения: 2 - 600см Точность измерения: 0.2см Угол измерения: 15 град. Рабочая температура: -40 - +85℃ Размер: 45 x 20 x 15 мм.		3
Датчик черной линии TCRT5000 *Характеристики:* Расстояние уверенного определения препятствия: 1-25 мм Диапазон рабочего напряжения питания: 3,3 – 5 В Тип используемого ИК датчика: TCRT5000 Длина волны излучения: 950нм Тип используемого компаратора: LM393 Максимальная нагрузка на выход компаратора: 15 мА размеры модуля: 40x11x11 мм.		3
Датчик цвета на основе TCS34725 на плате для прототипирования с светодиодом подсветки, стабилизатором питания *Характеристики:* Интерфейс связи: I2C Рабочее напряжение: 3,3 В / 5 В Разрешение: 4 канала RGBC, 16 бит на канал Рекомендуемое расстояние измерения: 2 мм	 или 	3

Компонента	Чертеж и/или фото	Количество
Блок крепления моторов		
Базовый уголок правый		
Базовый уголок левый		
Поперечина передняя		
Крышка		
Опорный рычаг		

Компонента	Чертеж и/или фото	Количество
Колесо		
Крышка моторного блока		
Аккумуляторный блок		



Общий вид робота в сборе

Соревнования

Движение по черной линии

Робот должен быть полностью автономным после старта. В противном случае робот может быть дисквалифицирован. Роботы должны быть изготовлены из образовательных конструкторов. В конструкциях роботов разрешены пластиковые детали ручного изготовления или напечатанные на 3D-принтере. Любая электроника может быть использована только из образовательного конструктора.

Время заезда замеряется с момента пересечения роботом линии старта до момента пресечения роботом линии финиша. Робот пересекает линию, когда самая передняя его часть касается или пересекает линию. На выполнение одной попытки роботу даётся 2 минуты. Время попыток должно быть зафиксировано судьей по секундомеру. Как только робот пересекает линию старта, он должен оставаться полностью автономным. В противном случае он будет дисквалифицирован. Робот, блуждающий по соревновательному полю, должен быть дисквалифицирован. Считается, что робот покинул соревновательное поле, когда любое колесо, нога или гусеница полностью сошли с поля. Считается, что робот покинул линию (сошёл с линии), если никакая часть робота не находится над линией. Длина робота в этом случае считается по колесной базе.

Если робот потеряет линию более, чем на 10 секунд, он должен быть дисквалифицирован. Если робот срезал участок траектории, то он должен быть дисквалифицирован.

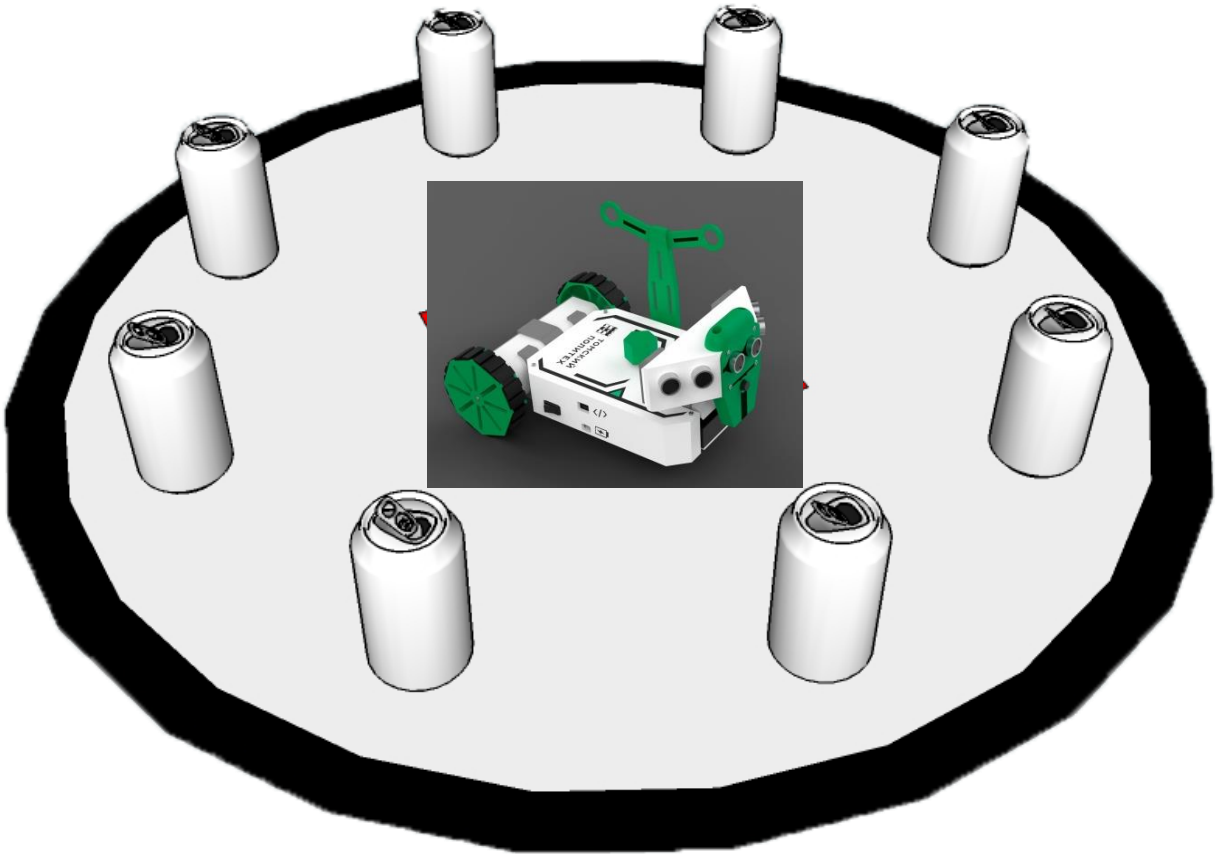
На прохождение заезда каждому игроку дается не менее двух попыток (точное число определяется судейской коллегией в день проведения соревнований). В зачет принимается лучшее время из попыток. Победителем будет объявлен игрок, потративший на преодоление дистанции наименьшее время.

Кегельринг

За наиболее короткое время робот, не выходя более чем на 5 секунд за пределы круга, очерчивающего ринг, должен вытолкнуть расположенные в нем кегли. На очистку ринга от кеглей дается максимум 2 минуты. Если робот полностью выйдет за линию круга более чем на 5 секунд, попытка не засчитывается. Во время проведения состязания участники команд не должны касаться роботов, кеглей или ринга.

Ринг представляет собой круг диаметром 1 м, ограниченный по периметру линией толщиной 50 мм (см. рис. 1). Цвет ринга – светлый (желательно белый). Цвет ограничительной линии - черный.

Кегли представляют собой жесткие цилиндры диаметром 70 мм, высотой 120 мм и весом не более 50 г. Кегли имеют матовую однотонную поверхность. Рекомендация: кегли можно изготовить из пустых стандартных жестяных банок для газированных напитков (330 мл). Для этого пустую банку достаточно обмотать листом обычной бумаги.



Робофутбол

В составе команды 2 участника. На поле одновременно находятся не более двух роботов (по одному от каждой команды). Цель игры – забить как можно больше мячей в ворота противника, не нарушая правил игры за отведенное на матч время. В игре используется мяч, входящий в комплект РОБОФУТБОЛ или аналогичный с инфракрасным сигналом. Роботы в начале игры располагаются во вратарской зоне, не выступая за ее пределы, располагаясь лицом к мячу, находящемуся в центре поля. Мяч в начале игры располагается в белом круге по центру игрового поля.

Движение роботов начинается по сигналу судьи. Разрешается блокировать роботов противника физически, с помощью своих роботов в том случае, если робот противника в этот момент владеет мячом. При выходе мяча с поля мяч устанавливается на центр поля, а роботы в исходное положение аналогично началу игры. Мяч всегда должен быть «на виду» так, чтобы другие игроки имели к нему доступ в любой момент матча, части робота не должны перекрывать мяч более, чем на его радиус.

В случае поломки робота во время игры, может быть произведена замена сломавшегося робота на аналогичного. Но не более, чем один раз за игру. В случае повторной поломки, команде засчитывается техническое поражение со счетом0:3

Выигрывает команда, забившая большее количество мячей в ворота противника. Во время матчей, ведётся видеосъёмка игры – для решения спорных вопросов.