ZARNITZA
ROBOTICS

Производственное объединение «Зарница»

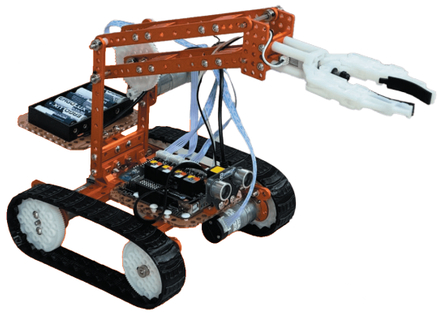
ОБРАЗОВАТЕЛЬНЫЙ НАБОР ДЛЯ ИЗУЧЕНИЯ МНОГОКОМПОНЕНТНЫХ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ И МАНИПУЛЯЦИОННЫХ РОБОТОВ Z.ROBO-4

Методическое пособие

РОБОТОТЕХНИКА

ZARNITZA
ROBOTICS

Разрабатываем и производим
высокотехнологичное учебное
оборудование для любых специальностей



УП9148

Конструктор для обучения и проведения
соревнований роботов Z.Robo-5
Приказ 838 Минпросвещения РФ

УП9145

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

УП6738

Стол для робототехники
Приказ 838 Минпросвещения РФ

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

Производственное объединение «Зарница»

ОБРАЗОВАТЕЛЬНЫЙ НАБОР ДЛЯ ИЗУЧЕНИЯ МНОГОКОМПОНЕНТНЫХ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ И МАНИПУЛЯЦИОННЫХ РОБОТОВ Z.ROBO-4

Методическое пособие

2025

ОГЛАВЛЕНИЕ

Введение **3**

Робот-манипулятор **4**

Основные понятия кинематики робота-манипулятора 4

Угловой 5-звенный манипулятор (5 DOF) 10

Плоскопараллельный манипулятор (4 DOF) 11

Параллельный манипулятор (гексапод) типа «Стюарт» 11

Дельта-робот *(Delta Robot)* 12

Использование программируемой кнопки набора 13

Среда программирования микроконтроллеров *Arduino IDE.* Использование библиотек 15

Функции в среде программирования *Arduino IDE* 19

Понятие и типы переменных в *Arduino IDE* 20

Логические структуры *Arduino IDE* 21

Оператор множественного ветвления 24

Циклы. Логические операторы 24

Сервоприводы. Характеристики сервоприводов 26

Программирование сервоприводов *Feetech* 31

Лабораторная работа № 1. Управление сервоприводом 35

Лабораторная работа № 2. Использование обратной связи для управления сервоприводами 38

Робот 1 ..... **42**

Работа с угловым манипулятором (5 степеней свободы) 42

Библиотека для решения задачи обратной кинематики для *Arduino InverseK.h* 50

Лабораторная работа № 3. Решение обратной задачи кинематики с использованием библиотеки *InverseK.h* 55

Режим робота-манипулятора *drag-to-teach* 60

Робот 2 ..... **65**

Работа с плоскопараллельным манипулятором (4 степени свободы) 65

Лабораторная работа № 4. Управление манипулятором с помощью *Bluetooth*-модуля 69

Техническое зрение роботов. Модуль технического зрения (МТЗ) робота-манипулятора 80

Подключение и настройка *OpenMV IDE* 80

Робот 3 ..... **83**

Работа с *Delta*-роботом (3 степени свободы) с использованием модуля технического зрения *OpenMV H7 Plus* 83

Робот 4 ..... **104**

Работа с платформой Стюарта с использованием модуля технического зрения *OpenMV H7 Plus* 104

Получение информации об окружающем пространстве. Обратная связь 112

Роботы и датчики 113

Лабораторная работа № 5. Управление *LED*-светодиодом 114

Лабораторная работа № 6. Управление *RGB*-светодиодом 117

Лабораторная работа № 7. Мониторинг показаний датчиков. Считывание данных инерциального датчика. 121

Лабораторная работа № 8. Считывание данных датчика касания 134

Лабораторная работа № 9. Считывание данных тактовой кнопки 138

Лабораторная работа № 10. Управление датчиком цвета 142

ВВЕДЕНИЕ

Образовательный набор для изучения многокомпонентных робототехнических систем и манипуляционных роботов *Z.Robo-4* предназначен для формирования у учащихся навыков работы с современными робототехническими системами. Набор включает в себя модульные компоненты для сборки четырёх типов промышленных манипуляторов:

- Угловой манипулятор (5 степеней свободы) – аналог промышленных роботов для сварки, покраски или точной сборки.
- Плоскопараллельный манипулятор (4 степени свободы) – идеален для задач упаковки, перемещения объектов в одной плоскости.
- Delta-робот (3 степени свободы) – высокоскоростная система для сортировки, фасовки, работы на конвейерах.
- Платформа Стюарта (гексапод) – применяется в авиасимуляторах, медицинской робототехнике и прецизионной обработке.

Собранные манипуляторы могут быть оснащены сменными захватами:

- Электромеханическим – для точного управления усилием сжатия (например, работа с мелкими деталями).
- Пневматическим – для работы с хрупкими или нестандартными объектами (стекло, пластик).

В данном методическом пособии будут рассмотрены различные конструкции роботов-манипуляторов, их кинематические схемы, основные принципы их программирования.

РОБОТ–МАНИПУЛЯТОР

Основные понятия кинематики робота-манипулятора

Робот-манипулятор – это тип промышленных роботов. Такие роботы выполняют функции человеческой руки. Роботы-манипуляторы используются на производственных линиях, в медицинских учреждениях и даже в домашних условиях – от сборки и упаковки до хирургических операций. При этом задачи определения положения и ориентации захвата манипулятора относительно его базовой платформы являются критически важными для обеспечения точности и эффективности выполнения задач.

Вопросами решения таких задач используется такой раздел физики, как механика, ибо **механика** – это раздел физики, изучающий законы механического движения. Механика делится на три раздела: **кинематику, динамику и статику.** В рамках рассмотрения вопросов управления роботами-манипуляторами, в данном случае, необходимо подробно рассмотреть кинематику.

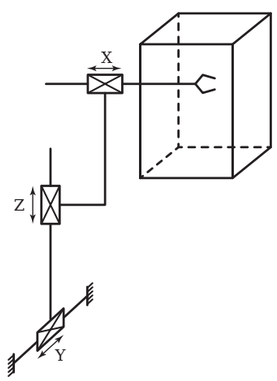
Кинематика, как один из основных разделов механики, изучает движение тел без учета сил, вызывающих это движение. В контексте роботостроения и, в частности, при проектировании роботов-манипуляторов кинематика играет ключевую роль в понимании и управлении движением этих сложных систем.

Кинематикой роботов называется раздел робототехники о движении робототехнических систем в трехмерном пространстве относительно заданной абсолютной системы координат в зависимости от времени, но без учета сил и моментов, порождающих такое движение.

Кинематика может давать очень точные вычисления во многих задачах, таких как позиционирование захвата в определенном месте в пространстве, проектирование механизма, который может перемещать инструмент из точки A в точку B, или прогнозирование столкновений робота с препятствиями. Кинематика занимается только мгновенными значениями координат робота и игнорирует их движение под действием сил и моментов (которые рассматриваются в разделе динамики).

Кинематика также актуальна для разработки алгоритмов управления манипуляторами. С помощью расчетов, основанных на кинематических моделях, робот может быть оснащён системами обратной связи, которые корректируют его действия в реальном времени. Это позволяет обеспечить более высокую точность движений и адаптировать действия робота к изменениям в окружающей обстановке и условиям работы.

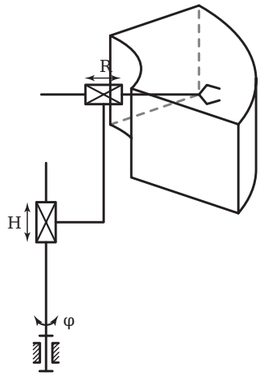
Одной из главных причин изучения кинематики в разработке роботов-манипуляторов является необходимость преобразования между различными системами координат.



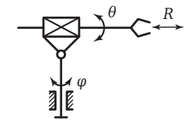
Система координат робота – комплекс определений, реализующий метод координат, то есть способ определять положение и перемещение точки или тела с помощью чисел или других символов. Совокупность чисел, определяющих положение конкретной точки, называется координатами этой точки.

Характер переносных степеней подвижности (поступательных П и вращательных В) определяет **базовую систему координат манипулятора.**

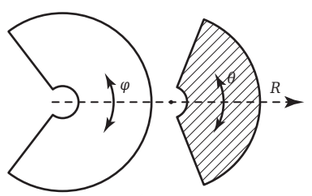
Если поступательных переносных степеней подвижности три (П = 3), а вращательных вообще нет (В = 0), то базовая система координат является **прямоугольной (декартовой),** а рабочая зона имеет форму параллелепипеда.



В том случае, когда П = 2, В = 1, рабочая зона ПР приобретает **цилиндрическую форму,** вернее форму неполного цилиндра. Соответствующая этому случаю базовая система координат *R, H, φ* удобна и получила большое распространение. Она обеспечивает обслуживание большого объема рабочей зоны, но имеет недостаток, связанный с трудностью организации манипулирования предметами на малой высоте.

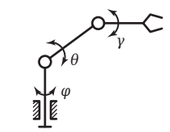


а)

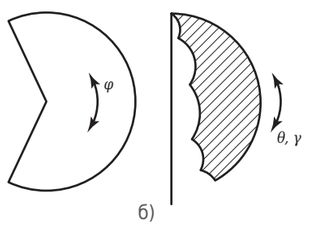


б)

Если П = 1, В = 2, то рабочая зона представляет собой неполный шар, а **базовая система координат *R, θ, φ* является сферической.** Структурная схема манипулятора, работающего в такой системе координат, приведена на рис. а, а проекции его рабочей зоны показаны на рис. б. Это наиболее универсальная базовая система координат. Она обеспечивает обслуживание большего объема рабочей зоны, чем при прямоугольной и цилиндрической системах координат. Однако конструкция манипулятора в этом случае получается более сложной, а ПР нуждается в более сложной системе управления.



а)



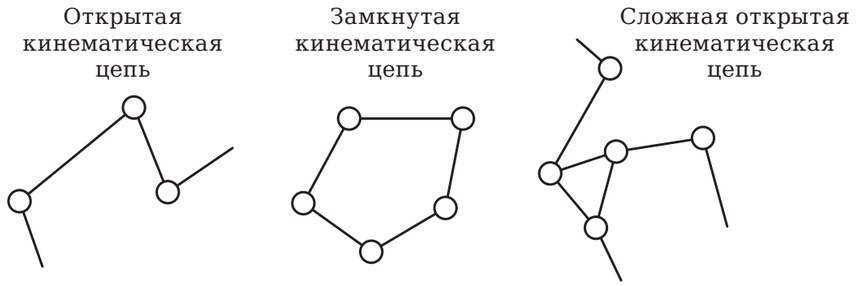
б)

При П = 0, В = 3 получают **ангулярную (угловую)** базовую систему координат γ, θ, φ. Структурная схема манипулятора, работающего в такой системе координат, приведена на рис а. На рис. б показаны проекции его рабочей зоны, которая имеет довольно сложную форму, поскольку ограничена частями сфер. Такая система координат наиболее универсальна, обеспечивает обслуживание наибольшего объема рабочей зоны и позволяет строить ПР, обладающие максимальной антропоморфностью.

Манипулятор промышленного робота является многозвенным механизмом с последовательным соединением звеньев и разомкнутой (открытой) кинематической цепью.

Тела, образующие манипулятор, называются его звеньями. Звенья, образующие попарные соединения и допускающие относительные перемещения, называются **кинематическими парами.**

Кинематическая цепь – это система звеньев, связанных между собой кинематическими парами. Любой механизм представляет собой кинематическую цепь звеньев, соединенных в кинематические пары. Кинематические цепи могут быть простыми и сложными, открытыми и замкнутыми, плоскими и пространственными.



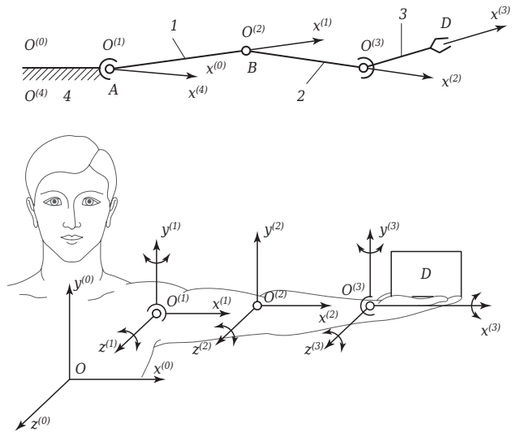
Виды кинематических цепей

В простой кинематической цепи каждое из ее звеньев входит в состав одной или двух кинематических пар, а в сложной кинематической цепи имеются звенья, входящие в состав трех и более кинематических пар. В открытой кинематической цепи имеются звенья, входящие в состав одной кинематической пары, а в замкнутой цепи каждое звено входит в состав двух и более кинематических пар. Если точки всех звеньев двигаются в одной или параллельных плоскостях, то кинематическая цепь называется плоской, в противном случае кинематическая цепь – пространственная (точки звеньев описывают плоские кривые в непараллельных плоскостях, или пространственные кривые).

Кинематическая схема манипулятора представляет собой соединение звеньев, определяющих основные движения схвата робота в рабочей зоне, и описывается в системе координат, оси которой целесообразно совместить с направлениями основных перемещений схвата, так как это упрощает математическое описание движений манипулятора.

Наиболее распространены пространственные манипуляторы, работающие в сферической, цилиндрической, декартовой или ангулярной системах координат. Гораздо реже используются плоские манипуляторы.

Ангулярная система координат характеризуется тем, что перемещение объекта манипулирования обеспечивается согласованным взаимным поворотом звеньев робота, имеющих постоянную длину. Эта система координат оказалась весьма удобной для производственных роботов. Роботы, использующие **ангулярную систему координат,** называются **антропоморфными** (антропоморфный – по форме, устройству схожий с человеком, его телом) промышленными роботами. Они более компактны по сравнению с традиционными конструкциями манипуляторов, которые являются комбинацией вращательных и поступательных кинематических пар. Звенья манипулятора соединяются



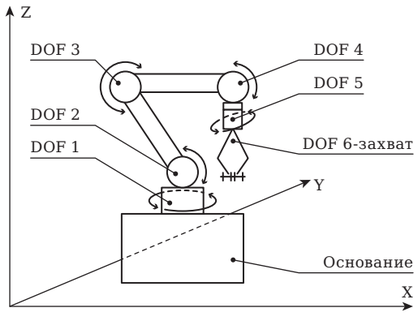
Звенья манипулятора антропоморфного робота

кинематическими парами пятого класса, т. е. каждое звено может иметь только одну степень подвижности относительно предыдущего звена, так что возможно либо вращательное, либо поступательное движение последующего звена относительно предыдущего.

Поступательное движение является наиболее простым видом движения. При поступательном движении все точки тела движутся одинаково. Примером поступательного движения может служить движение машин, автобусов, поездов, самолетов и так далее.

Вращательным называется такое движение твердого тела, при котором все его точки описывают окружности, центры которых лежат на одной прямой, называемой осью вращения. Традиционным примером вращательного движения тела является движение волчка вокруг неподвижной оси. Также примером вращательного движения твердого тела вокруг неподвижной оси является вращение винта вертолета или пропеллера самолета.

Понятие кинематической цепи и степени свободы



Основу манипуляторов составляют пространственные механизмы со многими степенями свободы. Степень подвижности, или *DOF (Degree Of Freedom)* – это количество независимых координат (обобщенных координат), однозначно определяющих положение всех звеньев механической системы относительно стойки (основания).

Для робота-манипулятора *DOF* определяет количество степеней свободы его рабочего органа. Чтобы произвольно позиционировать и ориентировать объект в пространстве, необходимо 6 степеней свободы: 3 для определения положения (координаты *X, Y, Z*) и 3 для определения ориентации (например, углы Эйлера: крен, тангаж, рыскание).

Манипулятор может иметь больше 6 *DOF.* В этом случае он называется избыточным *(redundant)*. Его кинематика становится еще сложнее, так как для решения ОЗК существует бесконечное множество решений, что позволяет роботу избегать препятствий или оптимизировать свои движения.

Понятие прямой и обратной задачи кинематики

Существуют две фундаментальные задачи кинематики:

- Прямая задача кинематики (ПЗК): По известным углам поворота (или линейным смещениям) в каждом суставе манипулятора определить положение и ориентацию его рабочего органа.
  - Пример: Мы знаем, на какой угол повернулся каждый сервомотор робота. Где сейчас находится его схват или инструмент?
  - Решение этой задачи относительно простое и заключается в последовательном перемножении матриц преобразования, описывающих переход от одного звена к другому.
- Обратная задача кинематики (ОЗК): По заданному положению и ориентации рабочего органа определить требуемые значения углов (или линейных смещений) во всех суставах манипулятора.
  - Пример: Нам нужно, чтобы инструмент робота оказался в точке *(X, Y, Z)* под определенным углом. В какие положения должны прийти все сервомоторы?

– Решение этой задачи является более сложным, часто не единственным (существует несколько кинематических конфигураций, приводящих к одному положению эндеффектора), и критически важно для управления роботом по траектории.

Таким образом:

Прямая задача кинематики – это вычисление положения *(X, Y, Z)* рабочего органа манипулятора (захвата) по его кинематической схеме и заданной ориентации (углам) $(A_1, A_2 \dots A_n)$ его звеньев (*n* – число степеней свободы манипулятора, *A* – углы поворота).

Обратная задача кинематики – это вычисление углов $(A_1, A_2 \dots A_n)$ по заданному положению *(X, Y, Z)* рабочего органа и известной схеме его кинематики.

Наиболее распространенной и важной является именно обратная задача кинематики. Эта задача не всегда может быть решена однозначно, потому что, хотя для углов всегда существует единственное положение рабочего органа, но не факт, что для положения существует такая же единственная комбинация углов. Достичь заданного положения возможно и при другой комбинации углов.

Чтобы избежать многозначности выбора решений, можно вводить дополнительные условия или ограничения, например, ограничения длин звеньев, радиусов углов поворота, или особые условия необходимости огибания препятствия, или непосредственно четко заданные траектории движения. Данные условия не всегда являются самодостаточными для конкретного решения обратной задачи кинематики, поэтому даже при их выполнении бывает так, что решение обратной задачи кинематики может отсутствовать, например, если заданные координаты находятся вне досягаемости манипулятора.

Таким образом, система управления роботом должна включать в себя решение обратной задачи кинематики и использовать более совершенные средства очувствления, т. е. больше сенсоров. Сейчас все более широко используется система технического зрения для точного определения положения звеньев манипулятора. Уменьшение погрешности в определении обобщенных координат и точности позиционирования дают возможность эффективно использовать робот-манипулятор для работы с различными деталями, расположенными в пространстве.

Классификация кинематических пар

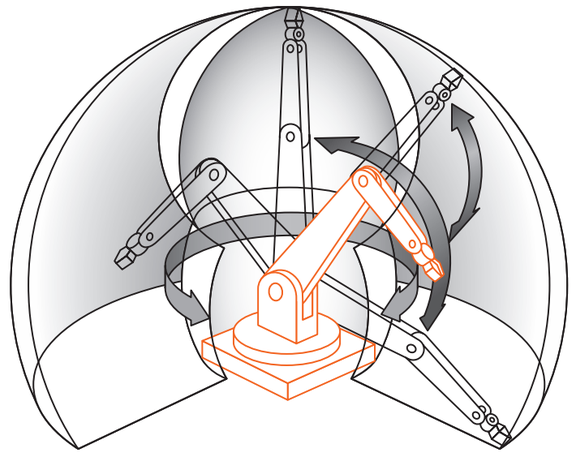
Кинематические пары классифицируются по числу степеней свободы, которые они обеспечивают соединяемым звеньям:

- Вращательная пара *(R - Revolute)*: обеспечивает одну степень свободы – вращение вокруг общей оси.
- Поступательная пара *(P - Prismatic):* обеспечивает одну степень свободы – линейное перемещение вдоль общей оси.
- Цилиндрическая пара *(C - Cylindrical)*: 2 *DOF* – вращение и линейное перемещение вдоль одной оси.
- Сферическая пара *(S - Spherical)*: 3 *DOF* – вращение вокруг трех осей (шарнир).

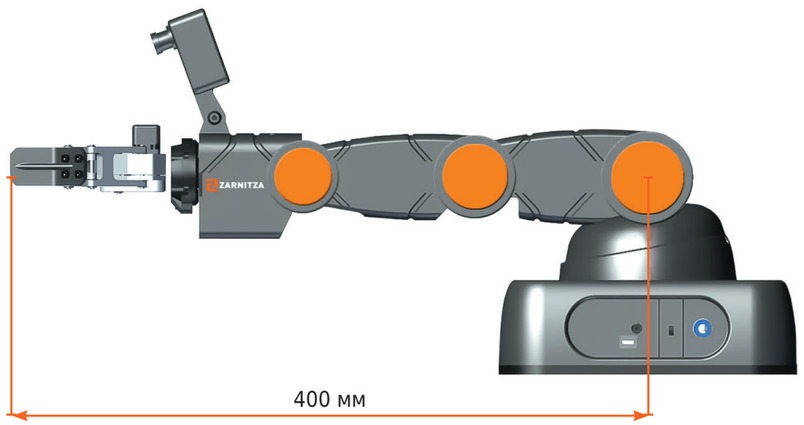
Кинематическая схема робота часто описывается последовательностью таких пар, например, схема 6-*DOF* промышленного робота часто имеет вид *RRRRRR* (шесть вращательных пар), а Стюарт-платформа – *6xUPS* (шесть пар типа *Universal-Prismatic-Spherical*).

Рабочая зона робота-манипулятора

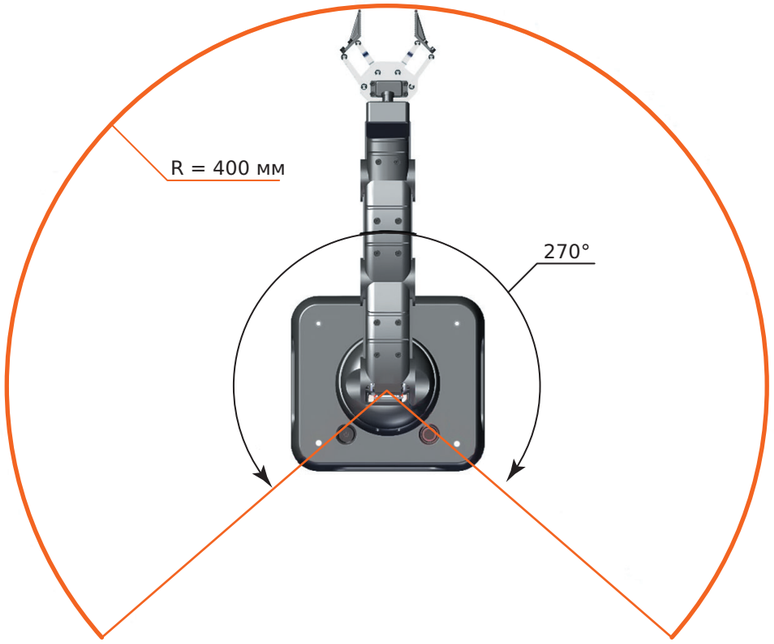
Рабочая зона манипулятора (иногда называемая также областью досягаемости) – это область, в которой манипулятор (например, робот или промышленный манипулятор) способен выполнять свои функции без перемещения. Определение рабочей зоны является ключевым этапом в проектировании и использовании манипуляторов, поскольку оно определяет, какие области пространства доступны для выполнения задач.



Для определения рабочей зоны манипулятора необходимо учитывать геометрические параметры манипулятора, такие как его длина, ширина и высота вместе с диапазоном движения каждого сустава. Эти параметры позволят определить границы рабочей зоны.



Представленный образовательный набор для изучения многокомпонентных робототехнических систем и манипуляционных роботов *Z.Robo-4* позволяет собрать и изучить несколько наиболее распространеннх вариантов кинематических схем.



Угловой 5-звенный манипулятор (5 DOF)

Это последовательный (серийный) манипулятор, звенья которого соединены последовательно, образуя кинематическую цепь. 5 *DOF* означает, что его рабочий орган обладает пятью степенями свободы. Чаще всего это 3 степени для положения и 2 для ориентации (например, он не может повернуть инструмент вокруг своей оси, если это не предусмотрено конструктивно на последнем звене). Типичная конфигурация суставов: *RRRRR*.

Кинематическая схема: Основание -> *R* (пояс) -> *R* (плечо) -> *R* (локоть) -> *R* (запястье: наклон) -> *R* (запястье: поворот).

Примеры и применение: Учебные роботы (например, *OWI-535*), манипуляторы для несложных операций захвата и перемещения, где не требуется полная ориентация в пространстве. Низкая стоимость и простота управления.

Особенности кинематики:

- ПЗК: решается стандартным методом Денавита-Хартенберга (последовательное перемножение матриц однородных преобразований).
- ОЗК: для 5 *DOF* решение сложнее, чем для 6 *DOF*, так как существует ограничение на ориентацию. Часто решается численными методами или путем декомпозиции задачи.

Плоскопараллельный манипулятор (4 DOF)

Это манипулятор смешанной или параллельной архитектуры, рабочее звено которого связано со стойкой несколькими кинематическими цепями (ногами). Ключевая особенность – его рабочее звено движется в параллельной плоскости относительно основания. 4 *DOF* обычно означают 2 перемещения в плоскости *(X, Y)* и 2 ориентации (крен и тангаж) или 3 перемещения и 1 ориентацию.

Кинематическая схема: Классический пример – манипулятор *SCARA (Selective Compliance Assembly Robot Arm).* Его схема часто описывается как *RRP* (два вращательных и одно поступательное сочленение), но он имеет 4 *DOF:* вращение в основании, вращение в плече, линейное перемещение по вертикали и вращение в схвате.

Примеры и применение: Роботы *SCARA* широко используются в высокоскоростной сборке, монтаже электронных компонентов на платы, паллетировании. Их кинематика оптимизирована для движений в горизонтальной плоскости.

Особенности кинематики:

- ПЗК: достаточно проста благодаря движению в параллельных плоскостях.
- ОЗК: имеет аналитическое решение, что позволяет производить очень быстрое и точное позиционирование.

Параллельный манипулятор (гексапод) типа «Стюарт»

Это полностью параллельный манипулятор. Он состоит из неподвижного основания и подвижной платформы, соединенных шестью раздвижными телескопическими стойками (актуаторами). Стойки соединены с основанием и платформой через сферические или карданные шарниры. Такая конструкция обеспечивает все 6 *DOF.*

Кинематическая схема: Схема описывается как *6-UPS (Universal-Prismatic-Spherical)* или *6-SPS (Spherical-Prismatic-Spherical)*. Каждая нога имеет одну поступательную пару (актуатор) и два шарнира, обеспечивающих по две или три степени свободы на концах.

Примеры и применение: Авиационные тренажеры (движущиеся платформы), высокоточные станки (гексаподы), телескопы, симуляторы тряски, хирургические роботы.

- Особенности кинематики:
  - ПЗК: относительно сложна. Требует решения системы нелинейных уравнений для нахождения положения платформы по известным длинам штанг.

– ОЗК: напротив, очень проста. Для заданного положения и ориентации платформы длины штанг вычисляются по простым геометрическим формулам (через расстояния между точками крепления). Это главное преимущество параллельных структур.

Преимущества: Высокая жесткость, точность и грузоподъемность. Меньшая инерция, так как приводы расположены на основании.

Недостатки: Малая рабочая зона относительно габаритов, сложные кинематические *singularities* (особенности).

Дельта-робот *(Delta Robot)*

Это пространственный параллельный манипулятор. Состоит из неподвижного основания и небольшой легкой подвижной платформы, к которой крепится рабочий инструмент. К основанию крепятся три рычага (часто называемые плечами), каждый из которых приводится в движение отдельным мотором. С каждым рычагом через параллелограммный механизм соединена одна из «ног», ведущих к платформе. Параллелограммы обеспечивают постоянную ориентацию платформы.

Кинематическая схема: Архитектура *RSS* или *RRR* (в зависимости от реализации параллелограмма). Основная схема: 3 вращательных привода на основании -> 3 рычага -> 3 параллелограмма -> подвижная платформа.

Примеры и применение: Невероятно высокоскоростные операции: сортировка, упаковка, перемещение мелких объектов (конфеты, таблетки, электронные компоненты) на конвейерных линиях.

Особенности кинематики:

- *DOF*: как правило, *3 DOF* (перемещение в пространстве *X, Y, Z*). Ориентация платформы остается постоянной (горизонтальной). Существуют модификации с 4 *DOF* (добавлен поворот схвата).
- ПЗК: сложнее, чем ОЗК, но решается аналитически через поиск пересечения сфер.
- ОЗК: очень проста и аналогична задаче для платформы Стюарта. Задание координат *(X, Y, Z)* платформы однозначно определяет углы наклона трех ведущих рычагов. Это позволяет достичь феноменальных скоростей управления.

Преимущества: Экстремально высокое быстродействие и точность, малая инерция подвижных частей.

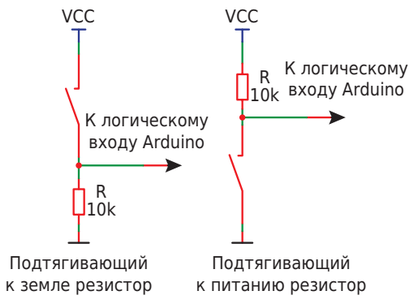
Недостатки: Ограниченная рабочая зона, неспособность ориентировать объект в пространстве (в базовой версии).

Разнообразие кинематических схем роботов-манипуляторов позволяет оптимально решать различные производственные и научные задачи. Последовательные манипуляторы *(5 DOF, SCARA)* обеспечивают большую рабочую зону и гибкость, в то время как параллельные (Стюарт, Дельта) – высочайшую скорость, точность и жесткость. Выбор архитектуры определяется требованиями технологического

процесса: необходимым количеством степеней свободы, требуемым рабочим пространством, скоростью, точностью и нагрузкой. Понимание основ кинематики является фундаментом для проектирования, программирования и эффективного применения роботизированных систем.

Использование программируемой кнопки набора

На этажерке контроллера установлена кнопка без фиксации положения. Ее можно использовать в качестве программируемой кнопки для аварийной остановки движения робота, в качестве переключателя режимов движения, для запуска программ, для сброса контроллера и т. д.



Для программирования кнопок на Arduino используйте встроенную функцию *INPUT_PULLUP* в *pinMode()* для подключения без внешних резисторов или схему с внешним подтягивающим резистором (например, 10 кОм). Для создания «программируемых» кнопок, которые выполняют разные действия в зависимости от времени нажатия (короткое/длительное) или комбинации кнопок, используйте функции задержки или библиотеки для определения «дребезга».

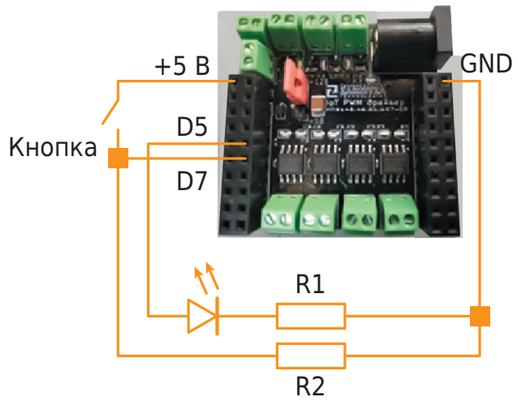
В схемах с внешним подтягивающим резистором используется два варианта подключения резисторов.

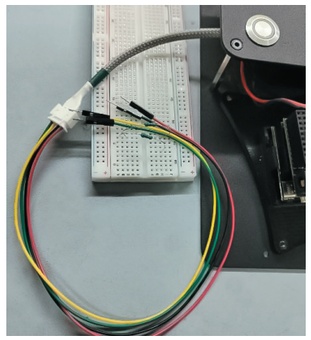
Когда тактовая кнопка не нажата, логический вход подключен только к земле через подтягивающий резистор и на этом входе будет считываться *LOW.* А когда кнопка нажата, появляется контакт между входом и питанием *VCC*, и считываться будет *HIGH.*

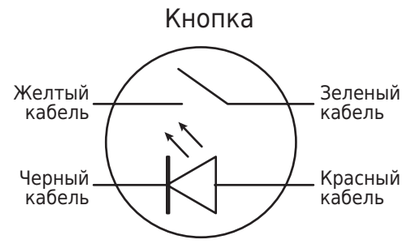
Можно также подключить кнопку наоборот – через подтягивающий резистор к питанию и через кнопку к земле. Тогда с входа будет считваться *HIGH,* а при нажатии кнопки – *LOW.*

Пример скетча программы для кнопки с подсветкой для *Arduino:*

```
int led_pin=5;// пин подключения светодиода подсветки кнопки
int button_pin = 7; // пин кнопки

void setup() {
    pinMode(led_pin, OUTPUT); // Инициализируем цифровой вход/выход в режиме выхода.
    pinMode(button_pin, INPUT); // Инициализируем цифровой вход/выход в режиме входа.
}

void loop() {
   if (digitalRead(button_pin) == HIGH) { // Если кнопка нажата
      digitalWrite(led_pin, HIGH);// зажигаем светодиод
 }
  else { //Иначе
      digitalWrite(led_pin, LOW);// выключаем светодиод
  }
}
```







Для работы с кнопкой воспользуйтесь макетной платой, резисторами и проводами, входящими в комплект.

Резисторы номиналом *R1* = 330 Ом, *R2* = 10 кОм. Провода: красный и черный – к светодиоду положительной и отрицательной полярности, желтый и зеленый – к замыкаемым контактам кнопки.

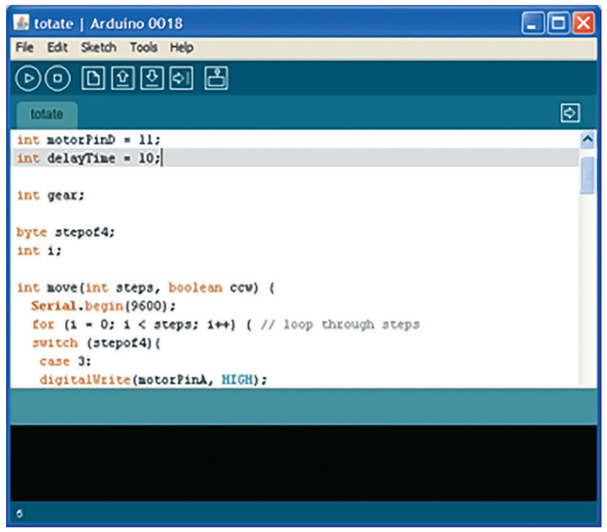
Пример скетча для программного сброса контроллера.

```
void(*resetFunc) (void) = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("Arduino запущен");
}

void loop() {
  // Полезная работа здесь...

  // Сброс только при определенном условии
  if (/* какое-то условие */) {
    Serial.println("Выполняется перезагрузка...");
    delay(1000);
    resetFunc(); // Сброс только когда это действительно нужно
  }
}
```


1. *void(*resetFunc) (void) = 0;*
   - *void(*resetFunc) (void)* – объявление указателя на функцию;
   - Первый *void* – функция не возвращает значение;
   - *(void)* – функция не принимает параметры;
   - *= 0* – присваивание нулевого адреса (адреса вектора сброса).
2. Как это работает:
   - При вызове *resetFunc()* процессор переходит по адресу 0;
   - В архитектуре *AVR (Arduino)* по адресу 0 находится вектор сброса;
   - Это вызывает программную перезагрузку микроконтроллера.
3. Особенности этого подхода:
   - Эквивалентно нажатию кнопки *Reset* на плате;
   - Сбрасываются все регистры и переменные;
   - Программа начинается заново с функции *setup().*

Среда программирования микроконтроллеров Arduino IDE. Использование библиотек

Разработка собственных приложений на базе плат, совместимых с архитектурой *Arduino*, осуществляется в официальной бесплатной среде программирования *Arduino IDE*. Среда предназначена для написания, компиляции и загрузки собственных программ в память микроконтроллера, установленного на плате *Arduino* – совместимого устройства. Основой среды разработки является язык *Processing/Wiring* – это фактически обычный *C++*, дополненный простыми и понятными функциями для управления вводом/выводом на контактах.



Программа, написанная в среде *Arduino*, носит название скетч. Скетч пишется в текстовом редакторе, который имеет цветовую подсветку создаваемого программного кода. Во время сохранения и экспорта проекта в области сообщений появляются пояснения и информация об ошибках. Окно вывода текста показывает сообщения *Arduino,* включающие полные отчеты об ошибках и другую информацию.

Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины.

Разрабатываемым скетчам дополнительная функциональность может быть добавлена с помощью библиотек, представляющих собой специальным образом оформленный программный код, реализующий некоторый функционал, который можно подключить к создаваемому проекту. Специализированных библиотек существует множество. Обычно библиотеки пишутся так, чтобы упростить решение той или иной задачи и скрыть от разработчика детали программно-аппаратной реализации.

Таким образом, библиотека – это набор функций, предназначенных для того, чтобы максимально упростить работу с различными датчиками, ЖК-экранами, модулями и пр., например, встроенная библиотека *LiquidCrystal* позволяет легко взаимодействовать с символьными *LCD*-экранами. Существуют сотни дополнительных библиотек, которые можно скачать в интернете. Стандартные библиотеки *Arduino* и ряд наиболее часто используемых дополнительных библиотек уже предустановлены и перечислены в справке. Но перед тем, как использовать дополнительные библиотеки, необходимо сперва установить их.

Чаще всего библиотеки выкладываются в виде *ZIP*-архива или просто папки. Название этой папки является названием библиотеки. Внутри папки будет файл с расширением *.cpp*, файл с расширением *.h,* а также текстовый файл *keywords.txt,* папка с примерами *examples* и другие файлы, требуемые библиотекой.

Например, для работы с сенсорами образовательного набора для изучения многокомпонентных робототехнических систем и манипуляционных роботов *Z.Robo-4* используется библиотека *Z-Robots.zip* (ее можно найти на *USB*-флеш накопителе, входящим в комплект представленного набора), разработанная программистами компании «Зарница».

Библиотека также содержит набор команд и функций, которые намного упрощают доступ к возможностям подключаемого устройства без написания дополнительного кода. Для каждой библиотеки существуют подробные описания команд для работы с ними.

Также для работы с библиотеками придется познакомиться с понятием **класс.**

Arduino IDE, как и *C++,* является объектно-ориентированным языком программирования. Это значит, что можно компактно группировать разнообразные объекты в классы, регламентируя поведение объекта универсально с помощью свойств и методов.

Классы в *Arduino* – это набор функций и переменных, которые могут использоваться для выполнения определенных задач программирования. В программировании на *Arduino* класс – это по сути то же самое, что и библиотека, модель для создания объектов определённого типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами.

Например, датчик температуры и влажности *DHT11* – принадлежит к целой группе схожих с ним датчиков: *DHT11, DHT21, DHT22*. У них общий принцип действия и управлять ими можно одним и тем же набором команд. Для этого и используется понятие класс. Дать точное определение этому понятию достаточно трудно, так как это абстракция. Класс не обязательно может быть привязан к какому-то устройству.

Логично предположить, что датчиков *DHT11* может быть несколько. Они будут принадлежать к одному классу, но быть при этом разными объектами.

У каждого класса есть свойства: это объявленные в пределах класса переменные. А также методы: это функции, с помощью которых классы осуществляют свою работу. Подключаемая библиотека фактически являет собой класс с необходимым набором заранее определенных методов и свойств.

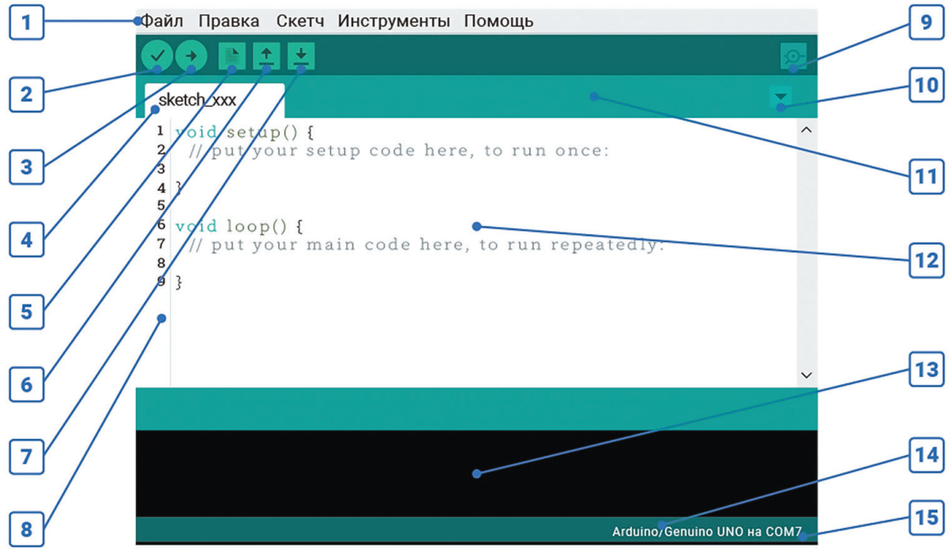
Среда *Arduino IDE* поставляется с набором стандартных библиотек: *Serial, EEPROM, SPI, Wire* и др.

Они находятся в подкаталоге *libraries* каталога установки *Arduino.* Необходимые библиотеки могут быть также загружены с различных ресурсов. Папка библиотеки копируется в каталог стандартных библиотек (подкаталог *libraries* каталога установки *Arduino)*. Если библиотека установлена правильно, то она появляется в меню Скетч/Подключить библиотеку. Выбор библиотеки в меню приведет к добавлению в исходный код строчки:

```
#include <имя библиотеки.h>
```

Эта директива подключает заголовочный файл с описанием объектов, функций и констант библиотеки, которые теперь могут быть использованы в проекте. Среда *Arduino* будет компилировать создаваемый проект вместе с указанной библиотекой.

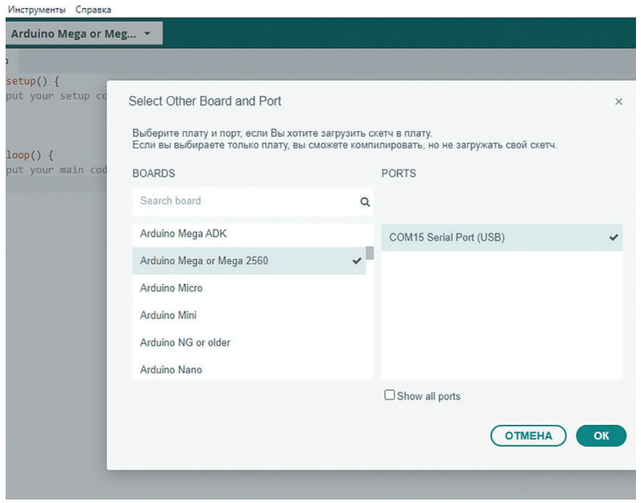
Интерфейс среды программирования *Arduino IDE:*



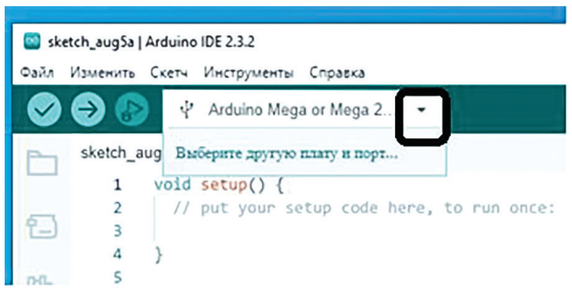
1. **Пункты меню.** Через пункты меню можно получить доступ ко всем функциям *Arduino IDE.*
2. **Проверить скетч (компилировать).** Проверка кода программы на ошибки.
3. **Загрузить скетч в *Arduino.*** Компиляция кода и его загрузка в плату.
4. **Вкладка с названием текущего скетча.** Имя скетча.
5. **Создать новый скетч.** Новое окно с чистой рабочей областью.
6. **Открыть скетч.** Список программ из папки со скетчами.
7. **Сохранить скетч.** Сохранение скетча в указанную папку.
8. **Номера строк.** Количество и номера строк, которые занимает код.
9. **Монитор последовательного порта.** Кнопка для откртия окна монитора последовательного порта. Это встроенная утилита, которая позволяет отправлять и получать данные между компьютером и платой *Arduino* через *USB*-кабель, используя последовательный интерфейс для обмена сообщениями с платой через *COM*-порт. Используется для отладки программ, отображения значений переменных и данных с датчиков,
10. **Управление вкладками.** Переключение между вкладками с разными скетчами.

11. **Область вкладок.** Область вкладок для быстрого переключения между скетчами.
12. **Рабочая область скетча.** Текстовый редактор для написания кода программы.
13. **Область уведомлений.** Область, информирующая пользователя обо всех событиях при записи или проверке кода. В том числе и сообщения об ошибках.
14. **Название платы.** Модель выбранной платы.
15. **Номер порта.** *COM*-порт, к которому подключена плата.

Перед загрузкой скетча требуется задать необходимые параметры – модель платы *Arduino*, с которой происходит работа, и номер *Serial Port*, к которому подключена плата *Arduino*. Для этого необходимо нажать на панель выбора платы и *COM*-порта:



Далее в выпадающем списке необходимо выбрать текущий *COM*-порт и модель платы *Arduino*:



Современные платформы *Arduino* перезагружаются автоматически перед загрузкой. На старых платформах необходимо нажать кнопку перезагрузки. На большинстве плат во время процесса загрузки будут мигать светодиоды *RX* и *TX*.

При загрузке скетча используется загрузчик *(bootloader) Arduino* – небольшая программа, загружаемая в микроконтроллер на плате. Она позволяет загружать программный код без использования

дополнительных аппаратных средств. Работа загрузчика распознается по миганию светодиода на цифровом выводе *D13.*

Монитор последовательного порта *(Serial Monitor)* отображает данные, посылаемые в платформу *Arduino* (плату *USB* или плату последовательной шины). Для отправки данных необходимо ввести в соответствующее поле текст и нажать кнопку *«Передать» (Send)* или клавишу *Enter*. Затем следует из выпадающего списка выбрать скорость передачи, соответствующую значению *Serial.begin* в скетче. На *ОС Mac* или *Linux* при подключении мониторинга последовательной шины платформа *Arduino* будет перезагружена (скетч начнется сначала).

Функции в среде программирования *Arduino IDE*

Функция является основной структурной единицей *C++* и *Arduino IDE*. Это часть программы, блок кода, имеющий своё название. Большая программа может строиться из нескольких функций, каждая из которых выполняет свою задачу, поэтому можно назвать функцию подпрограммой. Использование функций очень сильно упрощает написание и чтение кода, и в большинстве случаев делает его оптимальным по объёму занимаемой памяти.

- Функция должна быть описана, и после этого может вызываться.
- Функция должна быть описана снаружи других функций.

В общем виде функция имеет следующую структуру:

```
тип_данных имя_функции (аргументы) {
тело_функции
}
```

Где тип данных – это тип данных, который возвращает функция, имя функции – имя, по которому функция вызывается, набор аргументов (параметров) – необязательный набор переменных, и тело функции – код, который будет выполняться.

Вызов функции осуществляется именем функции и передачей набора аргументов, если таковые имеются:

```
имя_функции (аргумент1, аргумент2, аргументN);
```

Если аргументы не передаются, в любом случае нужно указать пустые скобки.

```
имя_функции ();
```

Функция в *C++* всегда возвращает результат. Этот термин означает, что после выполнения функции она выдаёт некое значение, которое можно присвоить другой переменной.

```
результат = имя_функции ();
```

Функция может принимать аргументы, может не принимать, может возвращать какое-то значение, может не возвращать.

Существует несколько правил объявления функции.

Перед *setup()* и *loop()* мы пишем слово *void,* что означает «пустой». Это вовсе не означает, что внутри функции ничего нет: просто для ее работы не используются внешние данные из других функций, и эта функция в процессе работы также не возвращает никаких данных для работы других функций. Функций в пределах одного скетча может быть несколько.

Правило первое. Объявляя функцию, необходимо указать ее тип.

Второе правило. Если функций в скетче много, значит, они должны обладать уникальным именем для собственной идентификации. Имя должно быть написано слитно и английскими буквами. Также имена функций и переменных не должны повторять служебные слова и команды среды *Arduino IDE.*

Правило третье. У функции есть свое тело, которое ограничивается фигурными скобочками {...}.

Правило четвертое. Каждая команда отделяется от другой точкой с запятой *«;».*

Правило пятое. Необходимо комментировать код: писать в некоторых частях программы «перевод» того, что выполняется тем или иным участком кода. Это делается с помощью символов «//».

Понятие и типы переменных в *Arduino IDE*

Для работы с той или иной информацией компьютер выделяет для ее хранения некую область памяти. В программировании это называется переменной.

Для того, чтобы использовать ограниченные ресурсы устройства, компьютер должен знать, что именно будет хранить: было бы нелепо выделить для хранения одного листа бумаги целый шкаф.

Для того, чтобы компьютер, управляющий роботом, знал, сколько ему необходимо выделить памяти для хранения той или иной информации, данные были разделены на несколько типов, фундаментальных и общих для всех языков программирования:

1. `int` – целочисленный тип. Хранит только целые числа.
2. `float` – неточный тип, т. е. позволяет хранить дробные числа.
3. `double` – данный тип ничем не отличается от *float* в *Arduinno IDE*.
4. `char` – символьный тип. Позволяет хранить, например, буквы.
5. `bool` – позволяет хранить данные логического типа: *true* – истина, или *false* – ложь.

Таким образом, переменная – это ячейка в оперативной памяти микроконтроллера, которая имеет своё уникальное название (а также адрес в памяти) и хранит значение соответственно своему размеру. К переменной мы можем обратиться по её имени или адресу и получить это значение либо изменить его. В переменной могут храниться промежуточные результаты вычислений, полученные данные (с датчиков, интернета, интерфейсов связи) и так далее.

Минимальным блоком памяти, к которому можно обратиться из программы по имени или адресу, является байт *(byte)*, который состоит из 8 бит, таким образом, любой тип данных будет кратен 1 байту. Любая память состоит из элементарных ячеек, которые имеют всего два состояния: 0 и 1. Эта единица информации называется бит *(bit).* Максимальное количество значений, которое можно записать в один байт, составляет $2^8 = 256$. В программировании счёт всегда начинается с нуля, поэтому один байт может хранить число от 0 до 255.

- 1 байт = 8 бит = 256
- 2 байта = 16 бит = 65 536
- 4 байта = 32 бита = 4 294 967 296

Тип данных переменной определяется при объявлении переменной в самом начале скетча после подключения необходимых библиотек. Чтобы объявить тип данных переменной, необходимо написать тип данных перед именем переменной:

```
int variable;
```

Используемый тип данных будет зависеть от размера и типа значения, которое будет храниться в переменной. А также определяет, какой объем памяти будет зарезервирован для переменной. Наиболее распространенным типом данных является целое число. Переменные, объявленные как *int,* могут содержать целые числа, такие как 7, 12 или 1003. Чтобы объявить переменную с типом *int,* необходимо объявить ее следующим образом:

```
int variable = 15;
```

В этом примере объявляется переменная целочисленного типа *int* с именем *variable* и устанавливается равной числу 15.

Целочисленные переменные используют два байта памяти, поэтому они могут содержать 216 различных чисел (до 65 536). Но тип данных *int* может содержать положительные и отрицательные значения, поэтому диапазон фактически разделен между -32 768 и 32 767.

В таблице ниже приведены типы всех переменных и их размерность в байтах

Название	Альтернативное название	Вес	Диапазон	Особенность
`boolean`	`bool`	1 байт*	0 или 1, *true* или *false*	Логический тип
`char`		1 байт	-128...127 (AVR), 0...255 (esp)	Символ (код символа) из таблицы ASCII
–	`int8_t`	1 байт	-128...127	Целые числа
`byte`	`uint8_t`	1 байт	0...255	Целые числа
`int**`	`int16_t, short`	2 байта	-32 760...32 767	Целые числа. На ESP8266/ESP32 – 4 байта! См. ниже
`unsigned int**`	`uint16_t, word`	2 байта	0...65 535	Целые числа. На ESP8266/ESP32 – 4 байта! См. ниже
`long`	`int32_t`	4 байта	-2 147 483 648...2 147 483 647	Целые числа
`unsigned long`	`uint32_t`	4 байта	0...4 294 967 295	Целые числа
`float`	–	4 байта	-3.4E+38...3.4E+38	Числа с плавающей точкой, точность 6–7 знаков

Название	Альтернативное название	Вес	Диапазон	Особенность
`double`	–	4/8 байта	1.7E+308...1.7E+308	Для AVR то же самое, что float. На ESP и прочих 32 бит МК8 байт, точность 15-16 знаков
–	`int64_t`	8 байт***	$-(2^{64})/2...(2^{64})/2 - 1$	Целые числа
–	`uint64_t`	8 байт***	$2^{64} - 1$	Целые числа

Переменные бывают глобальными и локальными. Глобальные переменные действуют в любом месте программы, внутри любой функции. Поэтому они должны быть объявлены вне тела какой-либо функции. Локальные переменные, объявленные внутри функции, работают только внутри функции, а после ее выполнения удаляются из памяти. Это экономит память, а также позволяет не запутаться в огромном количестве разнообразных данных, необходимых для работы большой программы.

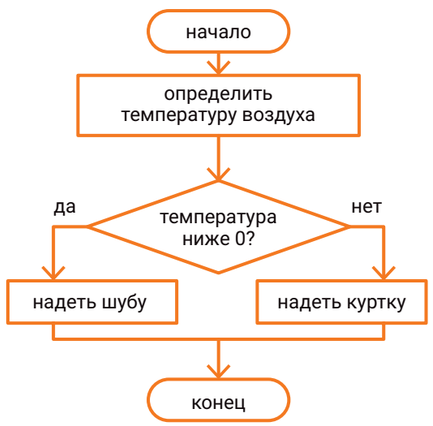
Логические структуры *Arduino IDE*

Для осуществления ветвления в любом языке программирования есть специальный оператор ветвления *if.* Оператор должен содержать однозначное условие, а также содержать два варианта действий в зависимости от значения логического выражения.

```
if (логическое условие) {
<если условие истинно>
} else {
<если условие ложно>
}
```

Обратите внимание, что условие заключается в круглые скобки.

Что такое <условие ложно> или <условие истинно>? В языке *C++* есть такое понятие, как логическая величина, которая принимает два значения: правда и ложь, *true* и *false*, 1 и 0. Для хранения логических величин существует тип данных *boolean (bool),* который может принимать значения *0 (false)* или *1 (true)*. Логические переменные часто называют флагами: если переменная равна *true* – флаг поднят, если *false* – опущен.



Графическое изображение ветвления на блок-схеме алгоритма

Два числа можно сравнить при помощи операторов сравнения:

- `==` равенство (a == b)
- `!=` неравенство (a != b)
- `>=` больше или равно (a >= b)
- `<=` меньше или равно (a <= b)
- `>` больше (a > b)
- `<` меньше (a < b)

В рассмотренных выше абстрактных примерах с *a* и *b* получается логическое значение, которое является результатом сравнения чисел. Пусть *a = 10* и *b = 20*, тогда скобка *(a > b)* вернёт значение *false,* потому что *a* меньше *b*. А вот *(a != b)* вернёт true, т. к. *a* действительно не равно *b*.

Для связи нескольких логических величин используются логические операторы:

- `!` логическое НЕ, отрицание. Есть аналог - оператор not
- `&&` логическое И. Есть аналог - оператор and
- `||` логическое ИЛИ. Есть аналог - оператор or

В языке Си не предусмотрено «двойных неравенств», то есть запись вида *0 < a < 10* приведёт к ошибке компиляции. Для записи таких условий нужно разделить неравенство на одиночные и соединить их оператором И:

```
0 <a && a <10.
```

В паре с оператором *if* работает оператор *else* (англ. «иначе»). Он позволяет предусмотреть действие на случай невыполнения *if:*

```
if (лог. величина) {
  // выполняется, если лог. величина - true
} else {
  // выполняется, если лог. величина - false
}
```

Также есть третья конструкция, позволяющая ещё больше разветвить код, которая называется *else if:*

```
if (лог. величина 1) {
  // выполняется, если лог. величина 1 - true
} else if (лог. величина 2) {
  // выполняется, если лог. величина 2 - true
} else {
  // выполняется иначе
}
```

Оператор множественного ветвления

Оператор *if* предусматривает только два варианта развития событий: условие может либо выполняться, либо быть ложным. Зачастую же возникает необходимость множественного выбора.

Подобно конструкции *if,* ***switch...case*** управляет процессом выполнения программы, позволяя программисту задавать альтернативный код, который будет выполняться при разных условиях. В частности, оператор *switch* сравнивает значение переменной со значением, определенном в операторах *case*. Когда найден оператор case, значение которого равно значению переменной, выполняется программный код в этом операторе.

Ключевое слово ***break*** является командой выхода из оператора *case* и обычно используется в конце каждого *case.* Без оператора *break* оператор *switch* будет продолжать вычислять следующие выражения, пока не достигнет *break* или конец оператора *switch:*

```
switch (var) {      // var - переменная, которая вычисляется для сравнения с
                                вариантами в case
case label:         //  label, label2 - значения, с которыми сравнивается значение
                                переменной
//код для выполнения
break;
case label2:
//код для выполнения
break;
default:
//код для выполнения, если ни одно из значений label не соответствует полученному
значению var
break;
}
```

Циклы. Логические операторы

В программировании существует еще одно фундаментальное понятие: циклы. Многие процессы в жизни происходят циклично: начиная от смены времен года до функции *loop()* в *Arduino IDE.* Однако если *loop()* работает бесконечное число раз, то операторы цикла позволяют контролировать количество повторений (это называется итерацией цикла) неким заданным числом, либо до тех пор, пока выполняется некое условие.

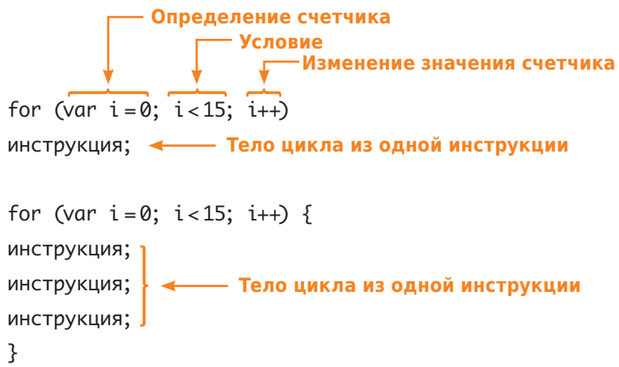
Первый тип цикла называется *for.* Фактически это счетчик повторений. В общем виде он выглядит следующим образом:

```
for (int i=0; i<100; i++>{
<тело цикла>
}
```

i - это переменная-счетчик, которая инициализируется прямо в условии цикла. Затем ставится

условие работы цикла: до тех пор, пока *i* меньше 180. Как только *i* достигнет этого значение, выполнение цикла прекратится. И операция инкремента: математический смысл символа ++:

```
i = i + 1
```



Каждое выражение условия должно отделяться от другого точкой с запятой «;», а само условие должно заключаться в скобки. Тело цикла ограничивается фигурными скобками.

Конструкция *for* используется для повторения блока операторов, заключенных в фигурные скобки. Счетчик приращений обычно используется для приращения и завершения цикла. Оператор *for* подходит для любых повторяющихся действий и часто используется в сочетании с массивами коллекций данных/выводов.

Конструкция *WHILE* используется в *C++* и *Arduino* для организации повтора одних и тех команд произвольное количества раз. По сравнению с *FOR* цикл *WHILE* выглядит проще, он обычно используется там, где нам не нужен подсчет числа итераций в переменной, а просто требуется повторять код, пока что-то не изменится, не наступит какие-то событие.

While будет вычислять в цикле непрерывно и бесконечно до тех пор, пока выражение в круглых скобках *()* не станет равно логическому ЛОЖНО. Что-то должно изменять значение проверяемой переменной, иначе выход из цикла *while* никогда не будет достигнут. Это изменение может происходить как в программном коде, например, при увеличении переменной, так и во внешних условиях, например, при тестировании датчика.

Синтаксис

```
while(выражение)  {
 //  оператор(ы)
}
```

Конструкция ***do ... while***

Цикл *do* работает так же, как и цикл *while,* за исключением того, что условие проверяется в конце цикла, таким образом, цикл *do* будет всегда выполняться хотя бы раз.

```
do
{
    // последовательность операторов
} while (проверка условия);
```

Пример:

```
do
{
  delay(50);          // подождать, пока датчики стабилизируются
  x = readSensors(); // проверить датчики
} while (x <100);
```

Бывают ситуации, когда нужно экстренно прервать выполнение цикла внутри блока цикла, не дожидаясь до перехода к блоку проверки условий. Для этого можно использовать оператор *break:*

```
while (true) {
if (checkSomething()) {
break;
}
}
```

Если необходимо просто остановить ход выполнения текущей итерации, но не выходить из цикла, а перейти к блоку проверки условий, то необходимо использовать оператор continue:

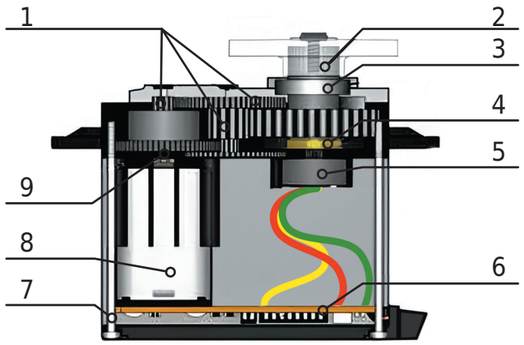
```
while (true) {

if (checkSomething()) {

continue;

}
```

Сервоприводы. Характеристики сервоприводов

Сервопривод (лат. *servus* – слуга, помощник; следящий привод) – привод с управлением через отрицательную обратную связь, позволяющую точно управлять параметрами движения.

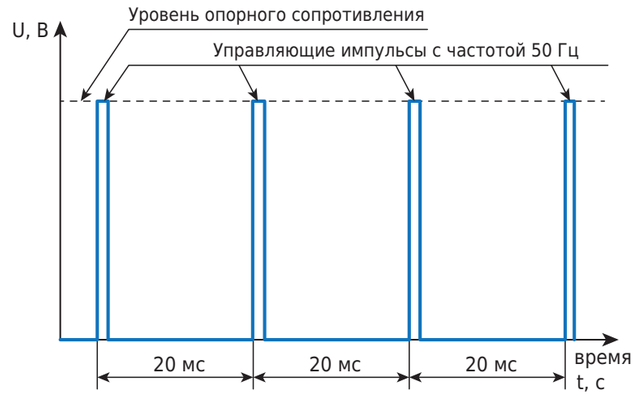
Сервопривод чаще всего встречается в робототехнике и чаще всего служит для решения задач точного перемещения грузов или предметов. Такая задача возникает при выполнении какой-либо механической работы (покраска, сварка, шлифовка, перемещение изделий на конвейере и т. д.). Выполняют такую работу манипуляторы, которые выглядят как механические руки. И эти манипуляторы не обходится без сервоприводов, которые приводят в действие его звенья.

Устройство большинства сервоприводов показано на рисунке. В его состав входит электродвигатель, редуктор с набором шестеренок, потенциометр (выполняет функцию датчика положения для обратной связи), электронная плата управления электродвигателем и корпус, в который заключено все содержимое. Провод сервопривода состоит из 3 жил: питание «плюс», питание «минус» и провод, на который подается управляющий сигнал. Как правило, провод питания «плюс» окрашен в красный цвет, а провод питания «минус» – в черный. Сигнальный провод (для передачи управляющего сигнала) может быть белым, оранжевым или желтым.



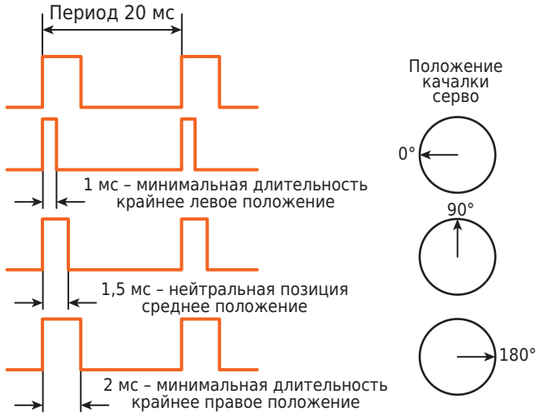
1. Редуктор;
2. Вал;
3. Подшипник вала;
4. Втулка;
5. Потенциометр (датчик обратной связи);
6. Плата управления;
7. Винты;
8. Электродвигатель;
9. Передача электродвигателя (шестерня).

Для управления такими двигателями принят стандарт управляющего сигнала, формируемого контроллером. Он представляет собой постоянно повторяющиеся импульсы. Частота этих импульсов все время остается постоянной и составляет 50 Гц. Получается, что временной период импульсов (время между передними фронтами соседних импульсов) составляет 1с/50 = 0,02 секунды, т. е. 20 миллисекунд.



Угловое положение выходного вала сервопривода задается продолжительностью подаваемого импульса. Для пояснения на рисунке показано приблизительное соотношение ширины импульса во временных координатах и угла поворота вала сервопривода. Управление поворотом вала сервопривода выполняется с помощью импульсов продолжительностью от 1 до 2 мс (миллисекунд).

Как видно из графика, для управления сервоприводом используется не что иное, как сигнал с широтно-импульсной модуляцией – ШИМ.



Широтно-импульсная модуляция (ШИМ, англ. *pulse-width modulation (PWM)*) – процесс управления мощностью методом пульсирующего включения и выключения потребителя энергии.

Характеристики сервопривода

Крутящий момент

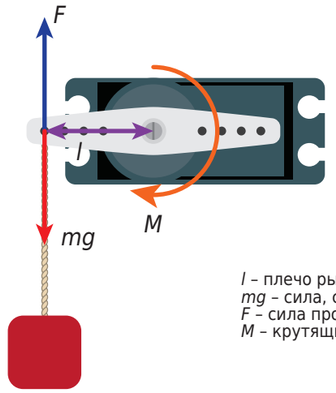
Крутящий момент представляет собой произведение силы на длину рычага. Другими словами, он показывает, насколько тяжёлый груз сервопривод способен удержать в покое на рычаге заданной длины.

Например, если крутящий момент равен 1 кг · см, это означает, что сервопривод удержит в горизонтальном положении рычаг длиной 1 см с подвешенным грузом 1 кг на свободном конце.

Если подвесить груз массой *m*, равной, например, 2 кг, то он будет воздействовать на рычаг сервопривода с силой, равной произведению массы на ускорение свободного падения *m·G*, равной 19,6 Н (Ньютон):

$$2\ кг \cdot 9{,}8\ м/с^2 = 19{,}6\ Н$$

Или при 1 Н = 0,101972 кг · см, получим 1,998644 кг · см.



l – плечо рычага
mg – сила, создаваемая нагрузкой
F – сила противодействия сервопривода
M – крутящий момент

Скорость поворота

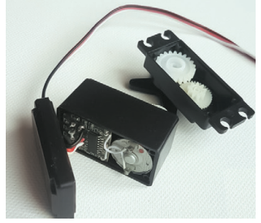
Скорость сервопривода выражается через время, за которое выходной вал успеет повернуться на 60°. Характеристика 0,1 с/60° означает, что сервопривод поворачивается на 60° за 0,1 с. Из неё можно вычислить скорость в оборотах в минуту, но так сложилось, что для сервоприводов чаще всего используют именно интервал времени поворота на 60°.

Форм-фактор

Сервоприводы различаются по размерам. И хотя официальной классификации не существует, производители давно придерживаются нескольких размеров с общепринятым расположением крепёжных элементов.

Форм-фактор	Вес	Размеры
Микро	9–25 г	22×15×25 мм
Стандартный	40–80 г	40×20×37 мм
Большой	50–90 г	49×25×40 мм

Внутренний интерфейс

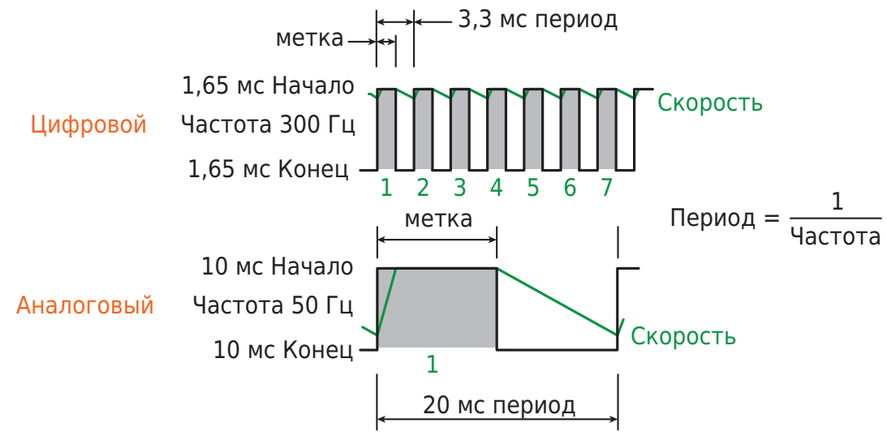


Сервоприводы бывают аналоговые и цифровые.

В аналоговых сервоприводах, как правило, установлена специальная микросхема, конфигурируемая аналоговыми элементами, как резисторы и конденсаторы, тогда как в цифровых серводвигателях – микроконтроллер с кварцевым генератором и зашитым ПО, вследствие чего цифровые сервоприводы могут воспринимать сигнал с большей частотой, чем аналоговые.

Управляющий сигнал для аналоговых и цифровых сервоприводов:

Форма ШИМ-сигналов (50 % от полного цикла)



Материалы шестерней редуктора

Шестерни редуктора могут быть пластиковые или металлические. Пластиковые шестерни редуктора изготавливаются из силикона или нейлона, они слабо подвержены износу, мало весят и недорого стоят. Это делает их довольно популярными в любительских проектах, где не предполагаются большие нагрузки на механизм.

Металлические шестерни редуктора тяжелее и дороже, но зато способны выручить там, где предполагаются нагрузки, непосильные для пластика. Поэтому более мощные двигатели обычно оснащаются именно металлическим редуктором. Шестерни из титана – фавориты среди металлических шестерней, причём как по техническим характеристикам, так и, к сожалению, по цене.

Однако металлические шестерни быстро изнашиваются, так что придётся менять их практически каждый сезон.

Коллекторные и бесколлекторные моторы

Существует три типа моторов для сервоприводов:

- Коллекторный мотор с сердечником *(Brush motor).*
- Коллекторный мотор без сердечника *(Coreless motor).*
- Бесколлекторный мотор *(Brushless motor).*

Коллекторный мотор с сердечником обладает плотным железным ротором с проволочной обмоткой и магнитами вокруг него. Ротор имеет несколько секций, поэтому, когда мотор вращается, ротор вызывает небольшие колебания мотора при прохождении секций мимо магнитов. В результате получается, что сервопривод вибрирует и не отличается точностью, зато это самый доступный по цене тип двигателей.

Коллекторный мотор с полым ротором обладает единым магнитным сердечником с обмоткой в форме цилиндра или колокола вокруг магнита. Конструкция без сердечника легче по весу и не разделена на секции, что приводит к более быстрому отклику и ровной работе без вибраций. Такие моторы дороже, но они обеспечивают более высокий уровень контроля, крутящего момента и скорости по сравнения с мотором с сердечником.

Безколлекторный мотор обладает всеми положительными качествами моторов без сердечников, но к тому же способен развивать в тех же условиях более высокую скорость и крутящий момент. Такой тип двигателей самый дорогой.

Тестер сервопривода

Для проверки работоспособности простых аналоговых сервоприводов и для первоначальной установки положения вала сервопривода существуют специальные тестеры. С их помощью можно проверить сервопривод перед установкой в систему. Для этого необходимо подключить сервопривод и питание к тестеру через соответствующие разъемы. Напряжение питания должно находиться в пределах от 4,8 В до 6 В.

Для изменения параметров тестера на передней панели установлены переменный резистор и кнопка выбора режима тестера *Select.* Переменный резистор используется для управления вращением вала сервопривода в ручном режиме.

Кнопка *Select* предназначена для выбора одного из трех режимов:

- ручной;
- нейтральный;
- автоматический.

В ручном режиме вращение ручки переменного резистора *Servo Consistency Test* приводит к изменению длительности управляющих импульсов ШИМ с поворотом вала сервопривода на соответствующий угол.

В режиме нейтраль на сервопривод подается сигнал с длительностью импульса 1,5 мс, что должно привести к установке вала сервопривода в центральное положение.

И, наконец, в автоматическом режиме ось сервопривода вращается автоматически на весь диапазон вращения, а с помощью переменного резистора, вращая ручку его управления, можно регулировать скорость вращения.

Программирование сервоприводов *Feetech*

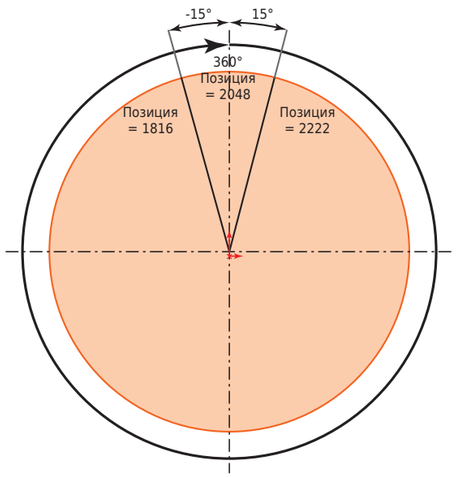
В качестве приводов, управляющих параметрами движения манипуляторов, используются цифровые сервоприводы серии *STS* компании *Feetech* - *STS3235* для суставов роботов и *STS3032* для механизма захвата.

Это интеллектуальные сервоприводы с последовательной шиной управления, напряжением питания 12 В (6 В для *STS3032*). Выполнены в корпусе из алюминиевого сплава и имеют металлический редуктор. Управляются посредством платы управления *TTL,* разработанной компанией *Feetech.* Оборудованы 12-битным высокоточным магнитным энкодером. Крутящий момент составляет 30 кг/см для *STS3235* и 4,5 кг/см для *STS3032*. Угол поворота 0–360 градусов, поддерживают режим работы двигателя непрерывного вращения («режим колеса»).

Обеспечивает обратную связь по параметрам:

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

Сервопривод имеет энергонезависимую память *EEPROM* (флеш-память), где хранятся параметры управления. Параметры, хранящиеся в *EEPROM*, есть двух типов – *Read only* и *Read&Write*, т.е. только для чтения, и параметры, которые можно изменять в процессе работы и которые сохраняются в энергонезависимой памяти. Настраивать сервопривод под конкретные условия эксплуатации, изменяя параметры типа *Read&Write* можно с помощью прилагаемого приложения *FT SCServo Debug V1.9.8.3.*



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

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

При использовании сервопривода в реальных устройствах использовать угол поворота 360 градусов требуется не всегда. Например, в манипуляторах, при движении суставов робота на угол, выходящий за рамки рабочего пространства, существует опасность упора сустава в следующий сустав. Это приведет к перегрузке сервопривода. Для предотвращения возможности выхода сустава робота-манипулятора за рамки разрешенных углов поворота необходимо использовать настраиваемые «мертвые зоны» сервопривода. Это зоны – значения *Position* от 0 до, например, 200 в начале хода и от 3000 до 4095 в конце. Использование сервоприводом значений *Position* в этих диапазонах являются

запрещенными – *dead zone*. Эти значения задаются программно путем записи в соответствующие регистры флеш-пямяти сервопривода требуемых значений *(Minimum Angle Limitation и Maximum Angle Limitation).*

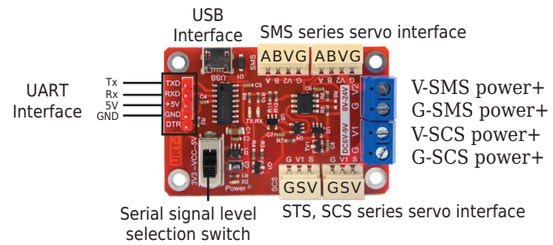
Сервоприводы *Feetech* серии *Serial Bus* управляются по последовательному протоколу связи, применимому для сервоприводов серий *FEETECH STS, SCS и SMS.* Сервопривод серии *STS* использует уровень *TTL* и одну общую шину соединения (сигнальная линия с разделением передачи/приема сигналов по времени), или временное мультиплексирование *(TDM)* – это метод передачи и приема независимых сигналов по общему сигнальному пути с помощью синхронизированных коммутаторов на каждом конце линии передачи, так что каждый сигнал появляется на линии только часть времени в соответствии с согласованными правилами, например, когда каждый передатчик работает по очереди. Физическое соединение организовано по трем проводам шлейфа, включая сигнальную линию, положительный и отрицательный провод питания *(S, V G).*

Протокол управления позволяет подключить до 254 сервоприводов на одну шину, поэтому каждый сервопривод должен иметь свой уникальный идентификационный адрес в сети. Для того, чтобы команда управления, выдаваемая контроллером для конкретного сервопривода, достигла адресата, он должен иметь соответствующий идентификатор *(ID1 – ID254).* Только сервопривод, соответствующий идентификационному номеру, может получить команду и вернуть ответную информацию контроллеру.

Режим связи протокола – последовательный асинхронный. Кадр данных делится на 1 бит стартового бита, 8 бит данных и 1 стопового бита. Битов четности нет, всего кадр состоит из 10 бит.

Схема подключения сервоприводов к контролеру *Arduino* представлена на рисунке.

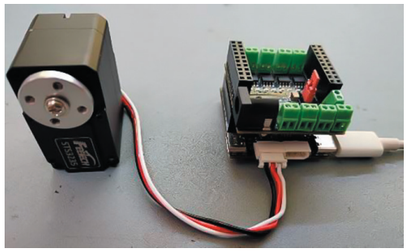
Для связи с управляющим контроллером сервоприводы *Feetech* используют интерфейс *UART* – универсальный асинхронный приёмопередатчик *(Univsersal Asynchronos Reciever-Transmitter)* – это физическое устройство приёма и передачи данных по двум проводам. Оно позволяет двум устройствам обмениваться данными на различных скоростях.



У каждого устройства, поддерживающего *UART,* обычно обозначены два вывода: *RX* и *TX. TX* – означает transmit (передаю), *RX – receive* (принимаю). Также необходимо объединить референсные уровни двух устройств *(GND-GND),* если не подразумевается гальваническая развязка. Для преобразования протокола *UART* в протокол обмена данными сервоприводов *Feetech* используется специальное устройство – *URT-1.*

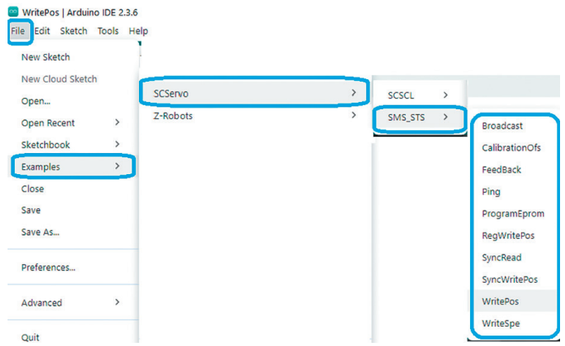
Наш *IoT* контроллер имеет встроеннй модуль управления сервоприводами. Достаточно подключить сервопривод в соответствующий разъем с помощью трехжильного кабеля.

Для работы с сервоприводами, например, *STS3235*, и написания программного кода, необходимо использовать библиотеку от компании *Feetech SCServo.h,* которую можно скачать с официального сайта компании:



(https://gitee.com/ftservo/SCServoSDK/blob/12bc1f74eeb2bfd37383f513d5ff53d89874f238/SCServo_Arduino_220524.7z)

Библиотека сервопривода включает в себя несколько примеров скетчей для управления сервоприводами, задание текущего положения как среднее *(CalibrationOfs),* получения данных обратной связи (в том числе о текущей позиции вала сервопривода).



Существует три типа команд для управления сервоприводом *STS3235.*

- Команда *WritePos* - команда подается для управления одним сервоприводом
- Команда *RegWritePos*
  | - синхронная одновременная работа с несколькими
  | сервоприводами с предварительной подготовкой
  | данных для одновременного исполнения
- Команда *SyncWritePos*

Лабораторная работа

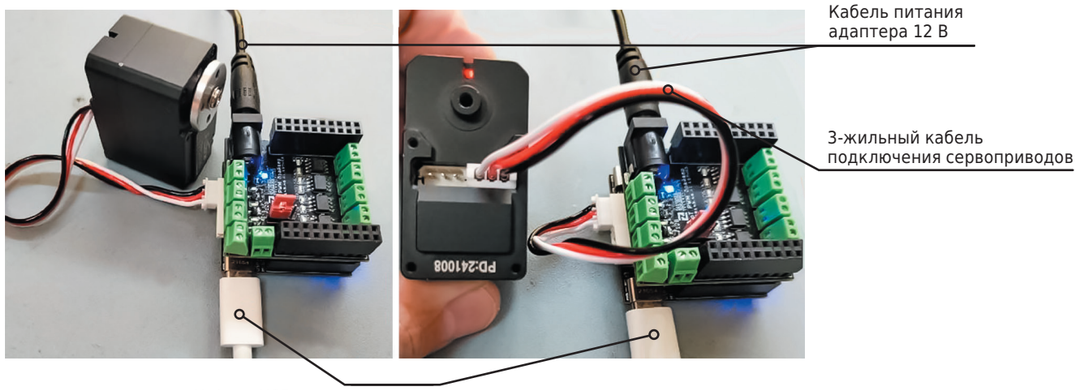
№ 1 УПРАВЛЕНИЕ СЕРВОПРИВОДОМ

Цель работы: *Научиться писать программы для управления сервоприводами Feetech.*

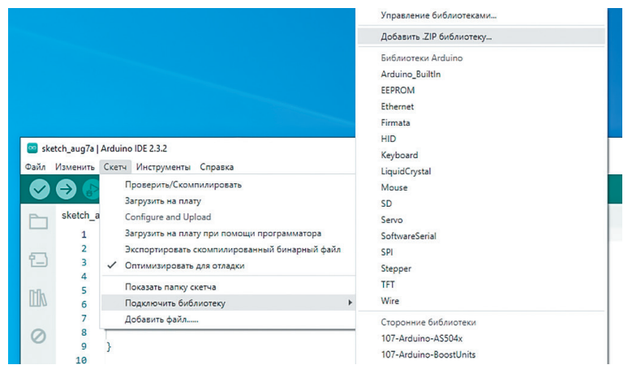
Задачи: *1. Установить в Arduino IDE библиотеку для сервоприводов Feetech.*
2. Выбрать из примеров библиотеки и запустить скетч для управления сервоприводом WritePos.

Порядок выполнения работы:

Для начала необходимо подключить один из входящих в комплект сервоприводов набора к роботехническому контроллеру.

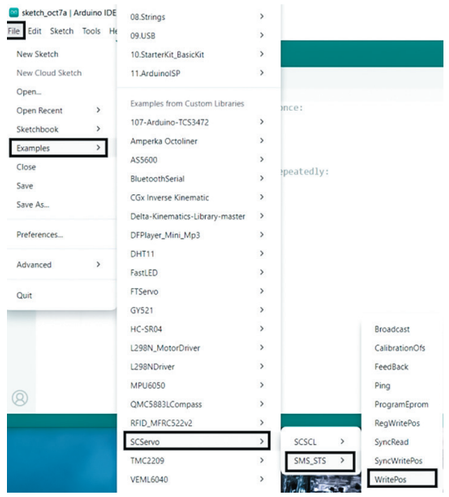


1. Установить прилагаемую на *USB* накопителе (флешке) библиотеку сервоприводов *Feetech* в *Arduino IDE*, или скачать с сайта
https://gitee.com/ftservo/SCServoSDK/blob/12bc1f74eeb2bfd37383f513d5ff53d89874f238/SCServo_Arduino_220524.7z
2. Открыть *Arduino IDE*. Выбрать пункт «Добавить *ZIP* библиотеку...».



Выбрать библиотеку *SCServo_Arduino_220524.7z*. После установки библиотеки в перечне примеров появится строчка *SCServo.*

3. Теперь можно загрузить скетч для тестирования режима *WritePos* управления сервоприводом путем выбора пунктов меню «Файл» – «Примеры» – *SCServo - WritePos.*



В *Arduino IDE* загрузится скетч:

```
#include <SCServo.h> // Подключаем библиотеку
SMS_STS sms_sts; // Создаем объект sms_sts

void setup()
{
 Serial1.begin(1000000); //Устанавливаем скорость обмена данными
 sms_sts.pSerial = &Serial1;// будем использовать Serial1 для связи

  delay(1000);
}

void loop()
{
  sms_sts.WritePosEx(1, 4095, 240, 50); // ID=1, Pos=4095, Speed=2400,
Acceleration=50
  delay(2240); //Расчет задержки [(P1-P0)/V]*1000+[V/(A*100)]*1000, P-позиция,V -
скорость

  sms_sts.WritePosEx(1, 0, 240, 50); //ID=1, Pos=0, Speed=2400, Acceleration=50
  delay(2240); //Расчет задержки [(P1-P0)/V]*1000+[V/(A*100)]*1000
}
```


4. Загрузите и запустите скетч.

5. Таким же образом загрузить пример программы для *RegWritePos* и *SyncWritePos* (для двух и более сервоприводов).

При выполнении работ с двумя и более сервоприводами необходимо помнить, что ID каждого из них должен быть уникальным.

Лабораторная работа

№ 2 ИСПОЛЬЗОВАНИЕ ОБРАТНОЙ СВЯЗИ ПО НАГРУЗКЕ ДЛЯ УПРАВЛЕНИЯ СЕРВОПРИВОДАМИ

Цель работы: *Изучить возможность использования обратной связи сервоприводов Feetech по нагрузке.*

Краткие теоретические сведения

При работе роботов-манипуляторов необходимо контролировать степень нагрузки на вал сервопривода – следить за значением крутящего момента. Это важно при поднятии и перемещении грузов для защиты от перегрузки сервоприводов, при попытке взять предмет захватом манипулятора. Сервопривод *Feetech STS3235* позволяет постоянно контролировать положение оси сервопривода, нагрузку (крутящий момент), напряжение, ток и температуру сервопривода.

Физически максимальное значение крутящего момента для сервопривода *STS3235* равно 30 кг·см при питании напряжением 12 В, что является нагрузкой в 100 %. В ячейке 16 флеш-памяти сервопривода можно прописать значение, которое устанавливает максимальное значение – *Maximum torque.* По умолчанию это значение 1000 (т. е. 100 % от исходного). Можно установить это значение равным, например, 800. Это будет означать, что сервопривод будет работать с максимальным значением крутящего момента, равным 80 % от максимального (для настройки сервопривода и работ с ячейками флеш-памяти необходимо дополнительно докупить модуль *URT-1* – многофункциональный *USB*-конвертер для настройки и управления сервоприводами компании *Feetech* – в комплекте не поставляется).

В ячейке флеш-памяти сервопривода с адресом 36 хранится параметр *Overload torque* – значение (в процентах от максимального) предела крутящего момента, при достижении которого включится защита от перегрузки. По умолчанию это значение равно 80 – это означает, что при нагрузке на ось серводвигателя крутящим моментом более чем 80 % от максимального сервопривод блокирует дальнейшее движение в заданном направлении на определенное время – *Protection time* (по умолчанию *Protection time* = 200, т. е. 200 x 10 ms = 2 с) и автоматически устанавливается на это время значение крутящего момента, прописанного в ячейке 34 – *Protective torque.* По умолчанию это значение *Protective torque* = 20 (или 20 % от максимального).

Таким образом, защита работает так: при достижении нагрузки на вал сервопривода, равной 80 % от максимальной (примерно 24 кг·см), вал сервопривода не переводится в полностью свободное состояние, а просто на 2 секунды крутящий момент снижается до 20 % от максимального. При этом вал не вращается до тех пор, пока не будет отправлена команда вращения вала в противоположном направлении.

После этого сервопривод возвращается в нормальное состояние.

Преимущества:

1. Температура не будет постоянно повышаться;
2. Ток не будет продолжать увеличиваться, защищая источник питания;
3. Предотвращается выход из строя сервопривода.

Особенность контроля по положению и нагрузке состоит в том, что эти данные нужны в реальном режиме времени, вне зависимости от того, чем в данный момент занят робот. То есть практически это задача, которая должна выполняться постоянно и в фоновом режиме.

Однако робот-манипулятор может выполнять различные программы, в которых могут быть использованы временные задержки *delay()*, которые приостанавливают работу программы на указанное в скобках число миллисекунд. (В одной секунде 1000 миллисекунд.) Максимальное значение может быть *4294967295 мс*, что примерно ровняется 50 суткам. При этом нужно понимать, что на время паузы с помощью *delay* работа программы приостанавливается, приложение не будет получать никаких данных с датчиков. Это является самым большим недостатком использования функции *delay* в *Arduino.*

Можно обойти эту проблему путем использования команд *millis()* и *micros().* С помощью команды *millis* выполнение всего скетча не останавливается, просто указывается, сколько времени *Arduino* должна «обходить» именно тот блок кода, который необходимо приостановить. В отличие от *delay* функция *millis* ничего не останавливает. Но использование данной команды требует определенных навыков. Для упрощения задачи на начальном этапе изучения программирования можно использовать функцию *yield().*

Функция *yield()* предоставляет возможность выполнять код во время задержек *delay().* Это позволяет создавать параллельно выполняющиеся задачи без использования сложных таймеров. Таким образом, используем эту функцию для считывания данных о нагрузке сервоприводов.

Внимание!!! Функция *yield()* работает только для *AVR Arduino* и не поддерживается на платформах esp8266 и esp32.

Задача: *Получить данные о нагрузке сервопривода в мониторе Arduino IDE.*

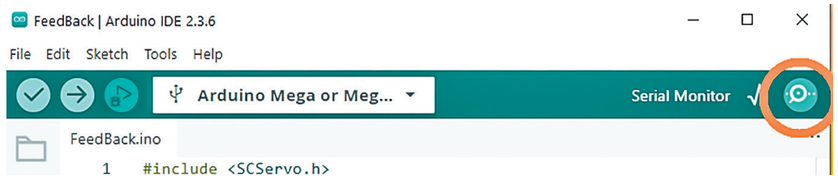
Порядок выполнения работы:

1. Подключить сервопривод к контроллеру.
2. Подключить питание к контроллеру.
3. Подключить контроллер к компьютеру с помощью кабеля *USB.*
4. Загрузить в *Arduino IDE* приведенный в качестве примера ниже скетч и выполнить его (скетч для сервопривода с *ID=1*).

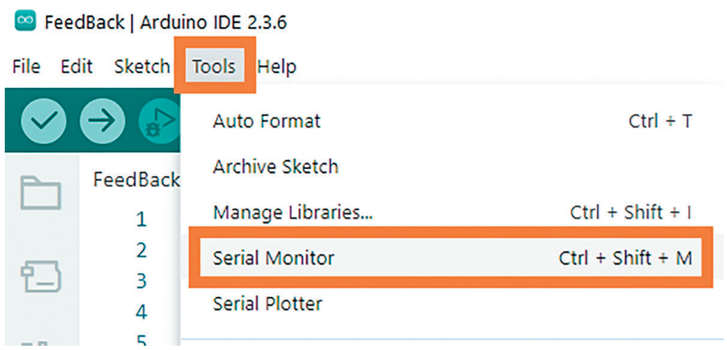
```
#include <SCServo.h>
SMS_STS sms_sts;
int Load;

void setup()
{
  Serial.begin(9600);
  Serial1.begin(1000000);
  sms_sts.pSerial = &Serial1;
  delay(1000);
}

void loop()
{
```

```
  sms_sts.WritePosEx(1, 4095, 240, 50);
  delay(9000);

  sms_sts.WritePosEx(1, 0, 240, 50);
  delay(9000);
}

  void yield() {
    Load = sms_sts.ReadLoad(1);

    Serial.print("Servo Load:");
    Serial.println(Load, DEC);

     }
```

5. Запустите монитор в *IDE* и убедитесь в том, что данные о нагрузке сервоприводов поступают без задержек. Монитор можно запустить, нажав значок в правом верхнем углу экрана – «Монитор».

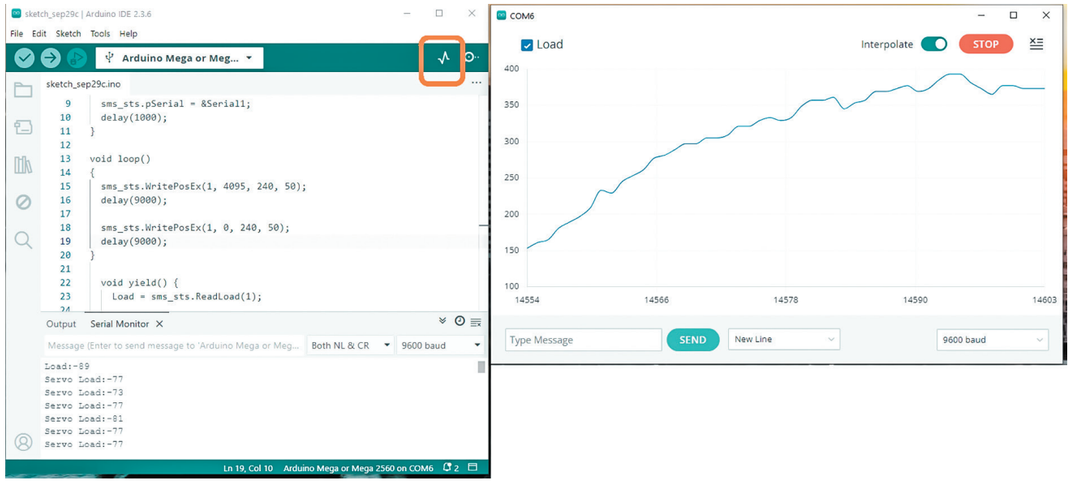


Или выбрать пункт меню «Инструменты» – «Монитор порта».



Но лучше наблюдать изменение (рост значения загрузки вала сервопривода при попытке удержать его рукой) параметра, запустив *Serial Plotter.*

6. Напишите свой скетч, в котором сервопривод *ID1* будет постоянно вращать вал от значения *Pos = 200* до *Pos = 3000*. При этом необходимо постоянно считывать значения нагрузки. В процессе вращения попытайтесь пальцами остановить вращение сервопривода, тем самым увеличив нагрузку

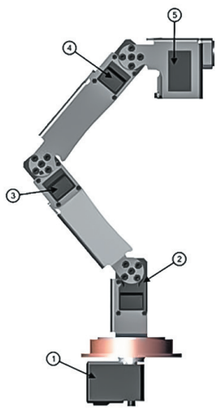


на вал. Обеспечьте вывод сообщения в монитор «Мне тяжело» с помощью команды *SerialPrint()* каждый раз при значении нагрузки = 70 процентов от максимальной.

РОБОТ 1

Работа с угловым манипулятором (5 степеней свободы)

Это последовательный (серийный) манипулятор, звенья которого соединены последовательно, образуя кинематическую цепь. *5 DOF* означает, что его рабочий орган обладает пятью степенями свободы. Это 4 степени для положения и 1 для ориентации (например, он не может повернуть инструмент вокруг своей оси, если это не предусмотрено конструктивно на последнем звене).



Кинематическая схема: Основание -> *R* (пояс) -> *R* (плечо) -> *R* (локоть) -> *R* (запястье: наклон) -> *R* (запястье: поворот).

Для начала необходимо собрать этажерку робототехнического контроллера и угловой робот-манипулятор в ссответствии с Инструкцией по сборке *Z.Robo-4*. Если вы уже прописали *ID* сервоприводов в соответствии с инструкцией, приведенной в Руководстве по эксплуатации, то при сборке используйте нумерацию *ID* сервоприводов как на рисунке. Если прописали, но собрали в другой последовательности, или пока не прописывали *ID*, то выполните процесс смены *ID* сервоприводов на этом этапе (процесс смены *ID* сервоприводов описан в Инструкции по эксплуатации). Сервоприводу захвата необходимо присвоить *ID = 6*.

Подключите все сервоприводы последовательно с помощью входящих в комплект трёхжильных кабелей. Два длинных кабеля длиной 25 см используйте для подключения нижнего сервопривода. Остальные подключайте с помощью 15-сантиметровх кабелй. Для подключения захвата необходимо в разрыв цепи установить *DC-DC*-преобразователь напряжения.

Для удобства программирования и задания позиций для сервоприводов робота-манипулятора примем среднее положение вала сервопривода исходя из положения по умолчанию (или исходного положения робота-манипулятора).

В этом случае при программировании позиций сервоприводов необходимо учитывать, что выставленное значение *Poition = 2048* является средним, или «нулевым», и вращение вала будет относительно этого положения – до *Position = 4096* в одну сторону и до *Position = 0* в другую.

Установите робот-манипулятор в исходное положение, подключите его к робототехническому контроллеру, подайте питание. Подключите контроллер к компьютеру с помощью *USB*-кабеля и загрузите скетч:

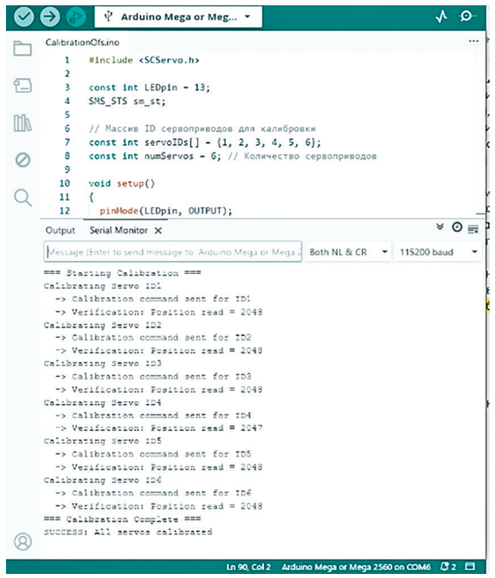
```
#include <SCServo.h>

const int LEDpin = 13;
SMS_STS sm_st;

// Массив ID сервоприводов для калибровки
const int servoIDs[] = {1, 2, 3, 4, 5, 6};
const int numServos = 6; // Количество сервоприводов

void setup()
{
  pinMode(LEDpin, OUTPUT);
  digitalWrite(LEDpin, LOW); // Индикация начала

  Serial1.begin(1000000);
  sm_st.pSerial = &Serial1;

  Serial.begin(115200); // Для отладки
  Serial.println("=== Starting Calibration ===");

  delay(1000);
}

void loop()
{
  bool calibrationSuccess = true;

  // Последовательная калибровка всех сервоприводов
  for(int i = 0; i < numServos; i++)
  {
    int currentID = servoIDs[i];

    Serial.print("Calibrating Servo ID");
```

```
Serial.println(currentID);

digitalWrite(LEDpin, HIGH);
delay(200);
digitalWrite(LEDpin, LOW);

// Выполнение калибровки для текущего сервопривода
sm_st.CalibrationOfs(currentID);

Serial.print("  -> Calibration command sent for ID");
Serial.println(currentID);

delay(1000); // Пауза между калибровками

// Проверка связи с сервоприводом после калибровки
int pos = sm_st.ReadPos(currentID);
if(pos != -1)
{
  Serial.print("  -> Verification: Position read = ");
  Serial.println(pos);
}
else
{
  Serial.println("  -> Verification: ERROR - No response");
  calibrationSuccess = false;
}

delay(500);
}

// Финальная индикация результата
Serial.println("=== Calibration Complete ===");
if(calibrationSuccess)
{
  Serial.println("SUCCESS: All servos calibrated");
  // Мигание для индикации успеха
  for(int i = 0; i < 5; i++)
  {
    digitalWrite(LEDpin, HIGH);
    delay(200);
    digitalWrite(LEDpin, LOW);
    delay(200);
  }
```



```
  }
  else
  {
    Serial.println("WARNING: Some servos may not have calibrated properly");
    // Постоянное свечение для индикации предупреждения
    digitalWrite(LEDpin, HIGH);
  }

  // Остановка программы после завершения калибровки
  while(1)
  {
    delay(1000);
  }
}
```

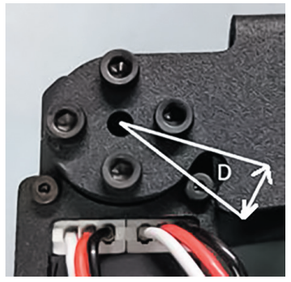
Сервоприводы примут текущее положение вала за среднее (0).

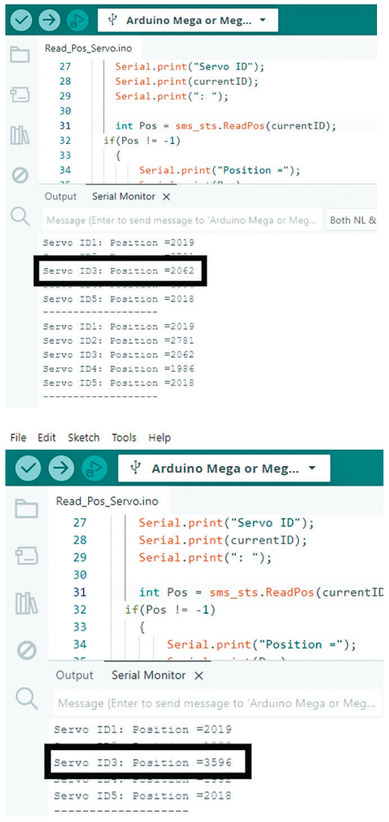


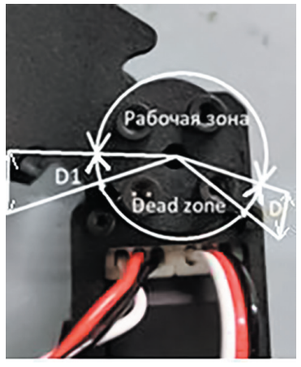
Теперь осталось только учесть одно обстоятельство – при движении суставов робота на угол, выходящий за рамки рабочего пространства, существует опасность упора сустава в следующий сустав. Это приведет к перегрузке сервопривода. Для предотвращения возможности выхода сустава робота-манипулятора за рамки разрешенных углов поворота необходимо использовать настраиваемые «мертвые зоны» сервопривода. Для определения рабочего пространства сервоприводов установите робот-манипулятор в положение, которое является для него критическим, подключите его к контроллеру и загрузите приведенный ниже скетч для считывания положения вала сервопривода. Подайте питание на контроллер робота-манипулятора. Запишите полученные значения *Position* в таблицу,пример которой приведен ниже. Установите максимально и минимально допустимые значения позиций.

Например, для сервопривода с *ID = 3* необходимо выставить вручную положение, немного не доходящее до упора – значение *D*. В этом положении необходимо считать текущую позицию сервопривода с *ID=3: minPos = 2062.*

Затем развернуть сустав робота в обратную сторону и выставить положение, исключающее упор сустава в предыдущий сустав – *D1*. Также необходимо считать позицию вала сервопривода *ID = 3: maxPos = 3596.*







Таким образом, безопасная зона работы сервопривода *ID = 3* от *minPos = 2062* до *maxPos = 3596.*

Для удобства работы с роботом-манипулятором в дальнейшем при программировании траектории движения суставов заполняется таблица для всех сервоприводов.

ID серво	Min значение Pos	Среднее значение (угол = 0)	Max значение Pos
1		2047	
2		2047	
3		2047	
4		2047	
5		2047	
6		2047	

ВНИМАНИЕ!!! Неправильно выставленные минимальные и максимальные значения minPos и maxPos могут привести к поломке робота.

```
#include <SCServo.h>

SMS_STS sms_sts;
const int LEDpin = 13;
const int servoIDs[] = {1, 2, 3, 4, 5}; // Массив идентификаторов сервоприводов
const int numServos = 5; // Количество сервоприводов

void setup()
{
  pinMode(LEDpin, OUTPUT);
  digitalWrite(LEDpin, HIGH); // Индикация начала работы
  Serial1.begin(1000000);     // Скорость связи с сервоприводами
  Serial.begin(115200);       // Скорость отладочного порта
  sms_sts.pSerial = &Serial1;
  delay(1000);
  Serial.println("=== System Started ===");
}

void loop()
{
  bool allSuccess = true; // Флаг успешного опроса всех сервоприводов

  // Цикл опроса всех сервоприводов
  for(int i = 0; i < numServos; i++)
  {
    int currentID = servoIDs[i];
    Serial.print("Servo ID");
    Serial.print(currentID);
    Serial.print(": ");

  // Дополнительная проверка через ReadPos
  int Pos = sms_sts.ReadPos(currentID);
  if(Pos != -1)
    {
        Serial.print("Position =");
        Serial.print(Pos);
      }
      else
      {
        Serial.print("Position Error");
```

```
        allSuccess = false;
      }

    Serial.println();
  }

  // Управление индикаторным светодиодом
  digitalWrite(LEDpin, allSuccess ? LOW : HIGH);

  Serial.println("-------------------");
```

Для исключения ошибок при задании положения вала сервопривода значения углов, определяющих «мертвую зону» *(dead zone),* значения *maxAngle* и *minAngle* прописываются во флеш-память сервоприводов с помощью приведенного ниже скетча. Просто подключите контроллер к компьютеру и загружайте для каждого сервопривода по очереди значения *maxAngle* и *minAngle.*

```
/*******************************************************************************
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*******************************************************************************/

#include <Dynamixel2Arduino.h>

const uint8_t DXL_ID = 1;

using namespace ControlTableItem;

Dynamixel2Arduino dxl(Serial1);

void setup() {
  // put your setup code here, to run once:
```



```
  // Use Serial to debug.
  Serial.begin(115200);

  dxl.begin();
  dxl.setPortProtocolVersion(1.0);
}

void loop() {
  // put your main code here, to run repeatedly:
  ChangeDeviceId();
  delay(5000);
}

DYNAMIXEL::InfoFromPing_t ping_info;
void ChangeDeviceId(void) {
  if (dxl.ping(DXL_BROADCAST_ID/*OLD_DXL_ID*/, &ping_info, 1)) {
    ping_info.model_number     = dxl.getModelNumber(ping_info.id);
    ping_info.firmware_version = dxl.getFirmwareVersion(ping_info.id);

    Serial.print("Detected the device: \n");
    Serial.print(" ID: ");
    Serial.print(ping_info.id, DEC);
    Serial.print(", Model: ");
```

Результат увидим в Мониторе *Arduino IDE:*

```
59    Serial.println( Device is reetecn servo );
60
61    dxl.unlockEprom(ping_info.id);
62    Serial.println("Device EPROM is unlocked");
63
64    uint16_t min_angle      = 200;
65    uint16_t max_angle      = 4000;
66    //uint8_t  cw_dead_angle  = 8;
67    //uint8_t  ccw_dead_angle = 16;
68
69    Serial.print("Set the min angle limit (steps, 0-4094): ");
70    Serial.println(min_angle);
71    dxl.setMinAngle(ping_info.id, min_angle);
72
73    Serial.print("Set the max angle limit (steps, 1-4095): ");
74    Serial.println(max_angle);
75    dxl.setMaxAngle(ping_info.id, max_angle);
76
77    //Serial.print("Set the clockwise dead angle (steps, 0-32): ");
78    //Serial.println(cw_dead_angle);
79    //dxl.setCWDeadAngle(ping_info.id, cw_dead_angle);
80
81    //Serial.print("Set the counterclockwise dead angle (steps, 0-32): ");
82    //Serial.println(ccw_dead_angle);
83    //dxl.setCCWDeadAngle(ping_info.id, ccw_dead_angle);
84
```

Output | Serial Monitor

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM3') | New Line | 115200 baud

```
Set the min angle limit (steps, 0-4094): 200
Set the max angle limit (steps, 1-4095): 4000
Device EPROM is locked
Check that values were set correctly.
Get the min angle limit: 200
Get the max angle limit: 4000
```

Теперь робот готов к работе и можно начинать работу с угловым манипулятором.

При работе по изучению возможностей позиционирования манипулятора с угловой кинематикой вы можете использовать как различные методы для самостоятельного расчета – алгебраический метод, метод геометрической инверсии, метод матриц Денавита-Хартенберга *(D-H)*, численные методы, так и готовые библиотеки, например *InvereK.h.*

Для решения задачи обратной кинематики рассмотрим, как пример, готовую библиотеку *InvereK.h.*

Библиотека для решения задачи обратной кинематики для *Arduino InverseK.h*

Понятие «библиотека» уже рассматривалось ранее в разделе «Среда программирования микроконтроллеров *Arduino IDE*. Использование библиотек». Напомним, что библиотеки в *Arduino* – это коллекции готового кода, которые позволяют разработчикам легко использовать и взаимодействовать с определенными аппаратными компонентами или выполнять определённые задачи без необходимости писать код с нуля. Библиотеки обрабатывают сложные функции и операции, предоставляя простой интерфейс для программирования.

Использование библиотек дает следующие преимущества:

1. Упрощение разработки – библиотеки упрощают процесс написания кода, позволяя избежать повторяющегося кода и ошибок.
2. Код повторного использования – можно использовать одну и ту же библиотеку для разных проектов, что экономит время и усилия.
3. Многие библиотеки создаются и поддерживаются сообществом *Arduino*, что позволяет быстро находить решения для распространённых задач.
4. Упрощение управления оборудованием – библиотеки обеспечивают простые команды для работы с конкретными модулями, компонентами и протоколами.

Для управления позиционированием захвата робота-манипулятора путем задания координат *X, Y* и *Z* в прямоугольной декартовой системе координат и решения задачи обратной кинематики используется библиотека *InverseK.h.*

Библиотека *InverseK.h* (скачать ее можно здесь *https://github.com/cgxeiji/CGx-InverseK*) предназначена для решения задачи обратной кинематики – нахождения углов сочленений или значений управляемых параметров на основе целевой позиции (конца манипулятора).

Основные элементы библиотеки:

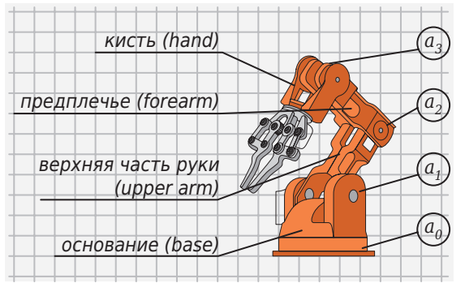
1. Определения и константы:
- определение длин звеньев манипулятора;
- определение координат системы координат для манипулятора.
2. Методы:
- Инициализация: Метод для установки начальных параметров манипулятора.
- Функция обратной кинематики: Метод, который принимает координаты цели *(X, Y, Z)* и вычисляет необходимые углы для сервоприводов манипулятора.

- Вспомогательные функции: могут быть включены методы для проверки валидности входных данных или для вычисления промежуточных значений.

Библиотека оперирует следующими понятиями:

Link (звено, сустав) – это прямая линия от одного сустава к другому. Длина звеньев необходима для расчета обратной кинематики. Используются 4 звена:

- основание *(base)*, которое соединяет a_0 с a_1 длиной 0 мм;
- верхняя часть руки *(upper arm)*, которая соединяет a_1 и a_2 длиной 200 мм;
- предплечье *(forearm)*, которое соединяет a_2 с a_3 длиной 200 мм;
- кисть *(hand),* которая соединяет a_3 с конечным эффектором длиной 270 мм.



Здесь a_0, a_1, a_2 и a_3 соответствуют каждому серводвигателю руки. Для упрощения расчетов предполагается, что исходная точка начинается в a_1, а a_0 разделяет ту же точку с a_1 (длина звена 0 мм).

Создание объекта *Links* для работы с методами библиотеки:

Устанавливается ссылка, объявив переменную типа *Link*:

Link myLink;

Затем инициализируется *Link* с указанием переменных:

myLink.init(_length_, _min_angle_, _max_angle_);

Углы задаются в радианах, длины – в миллиметрах. Далее каждый *Link* подключается к функции («решателю») задачи обратной кинематики:

InverseK.attach(base, upperarm, forearm, hand)

Для того, чтобы получить результат решения задачи обратной кинематики, необходимо объявить и задать 4 переменные типа *float*:

float a_0, a_1, a_2, a_3;

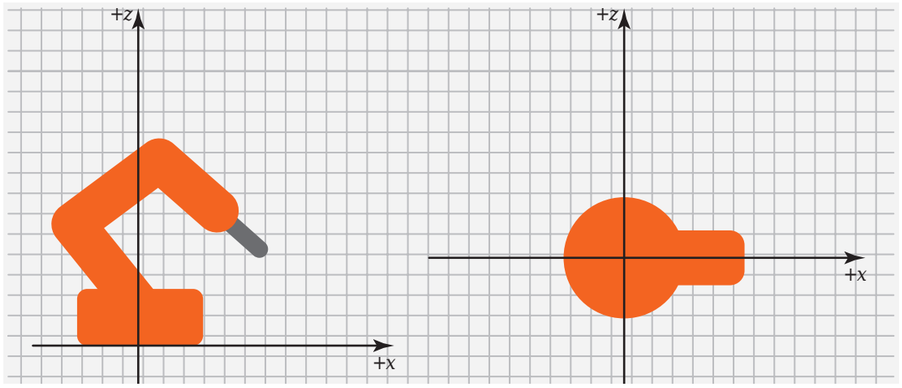
Эти переменные передаются в функцию и их значения там модифицируются, когда вызывается функция:

InverseK.solve(_x_, _y_, _z_, a_0, a_1, a_2, a_3) // this returns TRUE or FALSE

Если необходимо рассчитать целевую точку подхода захвата *(hand)* под определенным углом *phi*, необходимо указать его при вызове функции:

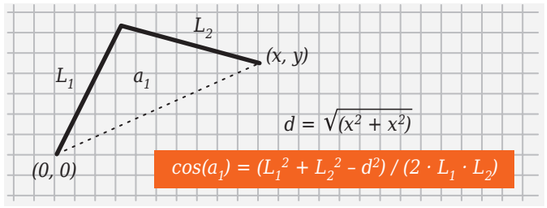
InverseK.solve(_x_, _y_, _z_, a_0, a_1, a_2, a_3, phi) // this returns TRUE or FALSE

Значения *X*, *Y* и *Z* возвращаются в миллиметрах и соответствуют схеме:



Функция возвращает значение *TRUE*, если решение для заданных координат найдено и рассчитано, и *FALSE* если данная задача с определенными координатами для данного манипулятора не имеет решения. В случае возврата функцией *FALSE* правильность значений для a_0, a_1, a_2 и a_3 не гарантируется.

При расчетах «решатель» данной библиотеки использует правило косинусов для нахождения углов, которые приводят к определенной координате в пространстве.



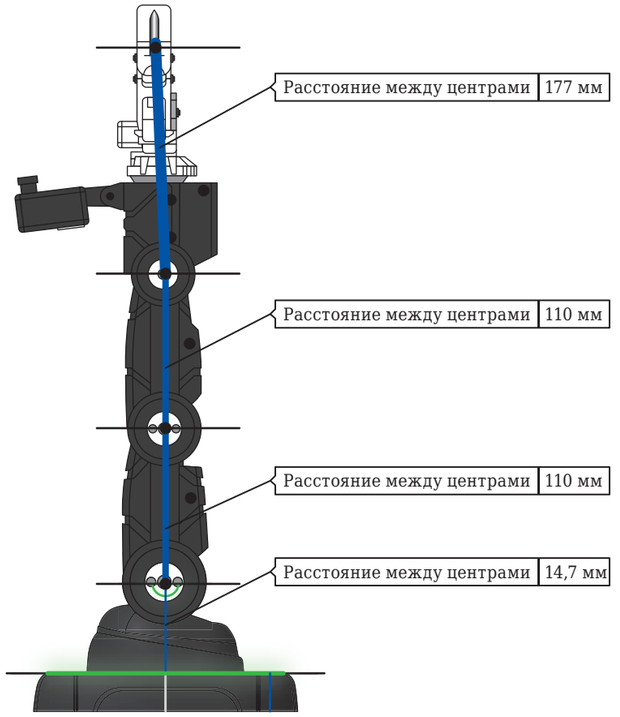
Для вычисления угла поворота основания декартовы координаты преобразуются в полярные координаты.

Таким образом, библиотека *InverseK.h* предоставляет разработчикам мощный инструмент для решения задачи обратной кинематики, позволяя управлять манипуляторами и роботами более эффективно. Использование библиотек в общем увеличивает возможности проекта и значительно упрощает процесс разработки.

Значения *base, upper arm, forearm* и *hand* у робота-манипулятора составляют 74,7 мм; 110 мм; 110 мм; 177 мм соответственно.

Пример использования библиотеки для расчета:

```
// Подключаем библиотеку InverseK.h
#include <InverseK.h>

void setup() {
 Serial.begin(9600);
 // Устанавливаем длину и ограничения вращения для каждого сустава Link
```


```
Link base, upperarm, forearm, hand;

base.init(0, b2a(0.0), b2a(180.0));
upperarm.init(200, b2a(15.0), b2a(165.0));
forearm.init(200, b2a(0.0), b2a(180.0));
hand.init(270, b2a(0.0), b2a(180.0));

// Attach the links to the inverse kinematic model
InverseK.attach(base, upperarm, forearm, hand);

float a0, a1, a2, a3;

// InverseK.solve() возвращает true если было найдено решение и false если нет

// Расчитываются углы без учета определенного угла наклона захвата
// InverseK.solve(x, y, z, a0, a1, a2, a3)
if(InverseK.solve(550, 0, 50, a0, a1, a2, a3)) {
  Serial.print(a2b(a0)); Serial.print(',');
  Serial.print(a2b(a1)); Serial.print(',');
  Serial.print(a2b(a2)); Serial.print(',');
  Serial.println(a2b(a3));
} else {
```

```
    Serial.println("No solution found!");
  }

 // Расчитываются углы с учетом определенного угла наклона захвата, например, 90°
  // InverseK.solve(x, y, z, a0, a1, a2, a3, phi)
  if(InverseK.solve(550, 0, 50, a0, a1, a2, a3, b2a(90.0))) {
    Serial.print(a2b(a0)); Serial.print(',');
    Serial.print(a2b(a1)); Serial.print(',');
    Serial.print(a2b(a2)); Serial.print(',');
    Serial.println(a2b(a3));
  } else {
    Serial.println("No solution found!");
  }
}

void loop() {

}

// Quick conversion from the Braccio angle system to radians
float b2a(float b){
  return b / 180.0 * PI - HALF_PI;
}

// Quick conversion from radians to the Braccio angle system
float a2b(float a) {
  return (a + HALF_PI) * 180 / PI;
}
```

В результате решения задачи обратной кинематики и вычисления необходимых углов, на которые необходимо повернуть вал сервопривода каждого звена манипулятора, получаем значения углов a_0, a_1, a_2, a_3. Однако напрямую этих данных недостаточно для управления сервоприводами *STS3235.* Управляющий сигнал для сервоприводов *STS3235* должен содержать количество микрошагов, на которое необходимо повернуть вал двигателя. Ранее был подробно рассмотрен принцип действия и устройство этих сервоприводов. Основное, что в данный момент нужно учесть – на какое количество микрошагов необходимо провернуть вал серводвигателя для поворота на полученный в результате вычислений угол.

Согласно таблице характеристик сервопривода, разрешение [град/импульс] = 0,087° (360°/4096). То есть один импульс управляющего сигнала проворачивает вал двигателя на 0,087°. Таким образом получается 11,5 импульса (или значение *Pos = 11,5*) на 1 градус. Значит, например, при текущем значении положения вала сервопривода 2048 для поворота на 15 градусов необходимо задать приращение на 11,5 x 15 = 172 импульса, или 2048 + 172 = 2220 (или 2048 – 172 = 1876). При этом вал сервопривода совершит поворот в 15 градусов, от значения 2048 до 2220 или 1876, если нужно вращать в другую сторону.

Лабораторная работа

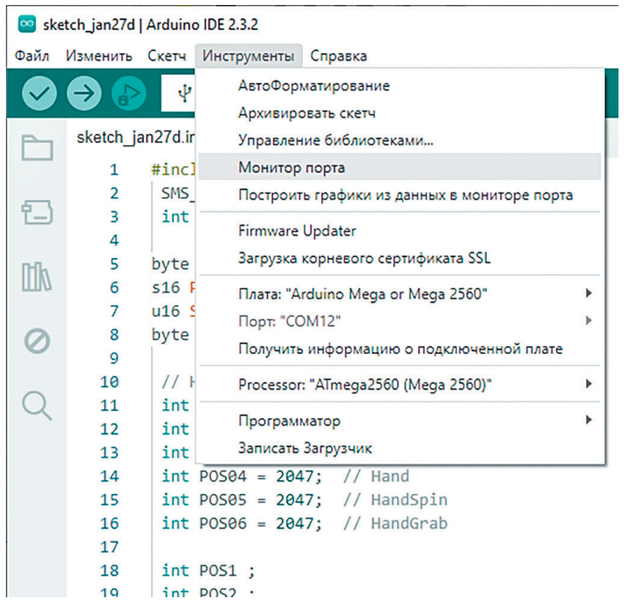
№ 3 РЕШЕНИЕ ОБРАТНОЙ ЗАДАЧИ КИНЕМАТИКИ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ INVERSEK.H

Цель работы: *Научиться использовать возможность библиотеки InverseK.h для решения задачи обратной кинематики при вычислении координат положения захвата манипулятора X, Y и Z относительно центра основания манипулятора.*

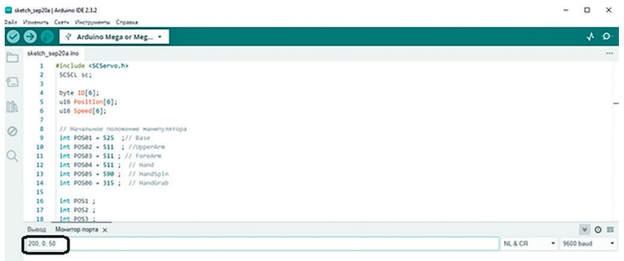
Задача: *С помощью программы, загруженной в контроллер манипулятора, указывая в качестве входных параметров конечные координаты целевой точки доступа захвата относительно центра основания манипулятора X, Y и Z, рассчитать углы сгиба звеньев манипулятора a_0, a_1, a_2, a_3.*

Порядок проведения работы

1. Подключите контроллер робота-манипулятора к компьютеру при помощи *USB*-кабеля.
2. Запустите *Arduino IDE.* Убедитесь, что устройство обнаружено.
3. Выберите в *Arduino IDE* модель платы как *Arduino Mega 2560.*
4. Напишите программу по образцу. Нажмите на кнопку загрузки на плату.
5. Запустите монитор, выбрав пункты меню «Инструменты», «Монитор порта».



6. Введите координаты *X, Y* и *Z* через запятую в строку монитора *Arduino IDE* и нажмите клавишу «Ввод».



Захват робота-манипулятора должен переместиться в точку с указанными координатами.

```
 #include <SCServo.h>
 SMS_STS sms_sts;

byte ID[6];
s16 Position[6];
u16 Speed[6];
byte ACC[6];

 // Начальное положение манипулятора
 int POS01 = 2047;// Base
 int POS02 = 2047; //UpperArm
 int POS03 = 2047; // ForeArm
 int POS04 = 2047;  // Hand
 int POS05 = 2047;  // HandSpin
 int POS06 = 2047;  // HandGrab

 int POS1 ;
 int POS2 ;
 int POS3 ;
 int POS4 ;
 int POS5 ;
 int POS6 ;

float x;
float y;
float z;

float b;
float u;
float f;
float h;

// Include the library InverseK.h
```



```
#include <InverseK.h>

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

// Setup the lengths and rotation limits for each link
  Link base, upperarm, forearm, hand;

  base.init(75, b2a(0.0), b2a(180.0));
  upperarm.init(110, b2a(-90.0), b2a(180.0));
  forearm.init(110, b2a(-17.0), b2a(180.0));
  hand.init(177, b2a(0.0), b2a(180.0));

  // Attach the links to the inverse kinematic model
  InverseK.attach(base, upperarm, forearm, hand);
   Serial1.begin(1000000);
  sms_sts.pSerial = &Serial1;
  delay(1000);
  ID[0] = 1;
  ID[1] = 2;
  ID[2] = 3;
  ID[3] = 4;
  ID[4] = 5;
  ID[5] = 6;

  Speed[0] = 240;
  Speed[1] = 240;
  Speed[2] = 240;
  Speed[3] = 240;
  Speed[4] = 240;
  Speed[5] = 240;

  ACC[0] = 50;
  ACC[1] = 50;
  ACC[2] = 50;
  ACC[3] = 50;
  ACC[4] = 50;
  ACC[5] = 50;

  Position[0] = 2047;
  Position[1] = 2047;
  Position[2] = 2047;
  Position[3] = 2047;
```

```
  Position[4] = 2047;
  Position[5] = 2490;
  sms_sts.SyncWritePosEx(ID, 6, Position, Speed, ACC);

  delay(2240);//((P1-P0)/V)*1000+(V/(A*100))*1000
}

void loop() {

  if (Serial.available() > 0) {
   String input = Serial.readStringUntil('\n');
    input.trim();
    int firstSpaceIndex = input.indexOf(' ');
    int secondSpaceIndex = input.indexOf(' ', firstSpaceIndex + 1);

    if (firstSpaceIndex > 0 && secondSpaceIndex > firstSpaceIndex) {
      x = input.substring(0, firstSpaceIndex).toFloat();
      y = input.substring(firstSpaceIndex + 1, secondSpaceIndex).toFloat();
      z = input.substring(secondSpaceIndex + 1).toFloat();

float a0, a1, a2, a3;
  //if(InverseK.solve(x, y, z, a0, a1, a2, a3, b2a(45.0))){ // Подход под углом 45
град.

  if(InverseK.solve(x, y, z, a0, a1, a2, a3)) {

    float b = (a2b(a0));
    float u = (a2b(a1));
    float f = (a2b(a2));
    float h = (a2b(a3));

          Serial.print ("b =  ");
          Serial.print (b);
          Serial.print ("\t");
          Serial.print ("u =  ");
          Serial.print (u);
          Serial.print ("\t");
          Serial.print ("f =  ");
          Serial.print (f);
          Serial.print ("\t");
          Serial.print ("h =  ");
          Serial.print (h);
          Serial.println ("\t");
```



```
    delay (100);

          float u1 = 90 - u;
          float b1 = b - 90;
          float h1 = 90 - h;

          POS1 = POS01 - (b1 / 0.087);
          POS2 = POS02 - (u1 / 0.087);
          POS3 = POS03 + (f / 0.087);
          POS4 = POS04 - (h1 / 0.087);

              delay(100);

             Position[0] = POS1;// Значения для четрех сервоприводов – в данном случае не
работаем с захватом и положением захвата
             Position[2] = POS3;
             Position[1] = POS2;
             Position[3] = POS4;

             sms_sts.SyncWritePosEx(ID, 4, Position, Speed, ACC);

          delay(2000);

            }

          Serial.print ("x =  ");
          Serial.print (x);
          Serial.print ("\t");
          Serial.print ("y =  ");
          Serial.print (y);
          Serial.print ("\t");
          Serial.print ("z =  ");
          Serial.print (z);
          Serial.println ("\t");

          delay(1000);
      }

   else {
   Serial.println("No solution found!");
   delay(1000);
  }
```

```
  }
}

  // Quick conversion from the Braccio angle system to radians
float b2a(float b){
  return b / 180.0 * PI - HALF_PI;
}

// Quick conversion from radians to the Braccio angle system
float a2b(float a) {
  return (a + HALF_PI) * 180 / PI;
}
```

Захват манипулятора переместится в точку, координаты которой были введены в строке ввода монитора *Arduino IDE*. Убедитесь в правильности расчетов координат.

Режим робота-манипулятора *drag-to-teach*

Для упрощения процесса работы с роботом-манипулятором в начале его изучения может быть полезен режим *drag-to-teach*. Манипулятор вручную устанавливается в требуемое положение. Запуская скетч считывания текущей позиции всех сервоприводов робота, фиксируются и записываются текущие положения *Position* всех сервоприводов. Затем манипулятор устанавливается в следующую позицию и все действия повторяются. Таким образом заполняется таблица всех необходимых конечных и промежуточных положений робота-манипулятора. Далее пишется скетч для контроллера, используя записанную последовательность положений, учитывая требуемое время выполнения каждой операции.

Попробуем написать простой скетч для перемещения захвата манипулятора, используя данный режим.

Для программирования робота в режиме *drag-to-teach* необходимо получить значения позиций фиксированных положений всех сервоприводов для каждого этапа перемещения манипулятора. Обратная связь по положению сервопривода *Feetech STS3235* позволяет это сделать с помощью программного кода скетча.

Допустим, необходимо выполнить следующий алгоритм:

1. Установить манипулятор в исходное положение при включении.
2. Подвести захват манипулятора к определенной точке пространства.
3. Открыть захват.
4. Свести захват на определенный угол.
5. Переместить захват в следующую точку пространства.
6. Открыть захват.
7. Установить манипулятор в исходное положение.

Порядок вполнения будет такой:

1. Установите манипулятор в требуемое исходное положение.
2. Загрузите скетч. Запишите данные *Position* по положению сервоприводов.

```
#include <SCServo.h>

SMS_STS sms_sts;
const int LEDpin = 13;
const int servoIDs[] = {1, 2, 3, 4, 5}; // Массив идентификаторов сервоприводов
const int numServos = 5; // Количество сервоприводов

void setup()
{
  pinMode(LEDpin, OUTPUT);
  digitalWrite(LEDpin, HIGH); // Индикация начала работы
  Serial1.begin(1000000);     // Скорость связи с сервоприводами
  Serial.begin(115200);       // Скорость отладочного порта
  sms_sts.pSerial = &Serial1;
  delay(1000);
  Serial.println("=== System Started ===");
}

void loop()
{
  bool allSuccess = true; // Флаг успешного опроса всех сервоприводов

  // Цикл опроса всех сервоприводов
  for(int i = 0; i < numServos; i++)
  {
    int currentID = servoIDs[i];
    Serial.print("Servo ID");
    Serial.print(currentID);
    Serial.print(": ");

  int Pos = sms_sts.ReadPos(currentID);
  if(Pos != -1)
    {
        Serial.print("Position =");
        Serial.print(Pos);
      }
      else
      {
```

```
            Serial.print("Position Error");
            allSuccess = false;
          }

        Serial.println();
      }

      // Управление индикаторным светодиодом
      digitalWrite(LEDpin, allSuccess ? LOW : HIGH);

      Serial.println("--------------------");
      delay(5000); // Интервал между опросами
    }
```

3. Установите манипулятор в требуемое положение – подведите захват манипулятора к требуемой точке пространства, откройте захват. Запишите новые значения *Position* для всех сервоприводов

4. Далее аналогично перемещаем манипулятор в требуемую точку, записываем значения положения всех сервоприводов,

5. Раскройте захват – запишите полученное значение.

6. И по аналогии верните манипулятор в начальное положение. Запишите значения положения всех сервоприводов.

Теперь остается подставить полученные значения в скетч, при этом обязательно учесть задержку на выполнение движения вплоть до останова перемещения. Ее необходимо рассчитать по уже рассмотренной формуле:

$$[(P_1 - P_0) / V] \cdot 1000 + [V / (A \cdot 100)] \cdot 1000,$$

где P_0 – начальное положение, P_1 – целевое положение, A – требуемое ускорение, V – требуемая скорость перемещения.

Значение ускорения расчитывается как $A = X \cdot 100$ шагов/секунду^2.

Скорость лучше для начала использовать минимальную, скажем, 100 шагов/секунду. Скорость перемещения манипулятора всегда можно рассчитать, учитывая, что минимальный угол разрешения сервопривода (градус/шаг) = 0,087. При скорости 100 шагов/сек можно рассчитать время прибытия элементов манипулятора в требуемую точку.

Используйте при этом пример скетча, приведенный ниже. В примере заданные движения выполнятся однократно. Если есть необходимость повторения – пропишите эти движения в *loop():*

```
#include <SCServo.h>

SMS_STS sms_sts;
```



```
byte ID[5];
u16 Position[5];
u16 Speed[5];
byte ACC[5];

void setup() {

  Serial1.begin(1000000);
  sms_sts.pSerial = &Serial1;
  delay(100);

  ID[0] = 1;
  ID[1] = 2;
  ID[2] = 3;
  ID[3] = 4;
  ID[4] = 5;
  ID[5] = 6;

 Speed[0] = 240;
 Speed[1] = 240;
 Speed[2] = 240;
 Speed[3] = 240;
 Speed[4] = 240;
 Speed[5] = 240;

 ACC[0] = 50;
 ACC[1] = 50;
 ACC[2] = 50;
 ACC[3] = 50;
 ACC[4] = 50;
 ACC[5] = 50;

 Position[0] = 2047;
 Position[1] = 2047;
 Position[2] = 2047;
 Position[3] = 2047;
 Position[4] = 2047;
 Position[5] = 2047;

 sms_sts.SyncWritePosEx(ID, 5, Position, Speed, ACC);

  delay(2000);
////////   1 положение  ////////////////////////////////////
```

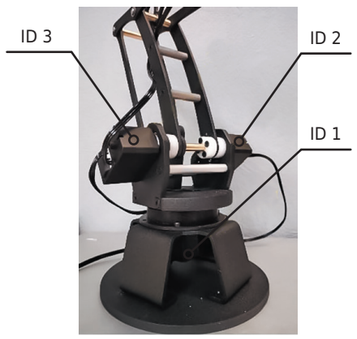
```
 Position[0] = XXXX; //Вместо XXXX вставьте полученное ранее значение
 Position[1] = XXXX;
 Position[2] = XXXX;
 Position[3] = XXXX;
 Position[4] = XXXX;
 Position[5] = XXXX;

sms_sts.SyncWritePosEx(ID, 5, Position, Speed, ACC);

delay(2500); // Значение задержки необходимо корректировать в зависимости от времени
выполнения движения

////////   2 положение    ////////////////////////////////////

 Position[0] = XXXX;
 Position[1] = XXXX;
 Position[2] = XXXX;
 Position[3] = XXXX;
 Position[4] = XXXX;
 Position[5] = XXXX;

 sms_sts.SyncWritePosEx(ID, 5, Position, Speed, ACC);

delay(2500);
……..

////////   X положение ////////////////////////////////////

 Position[0] = XXXX;
 Position[1] = XXXX;
 Position[2] = XXXX;
 Position[3] = XXXX;
 Position[4] = XXXX;
 Position[5] = XXXX;

 sms_sts.SyncWritePosEx(ID, 5, Position, Speed, ACC);

delay(2500);

}
 void loop()
 {
 }
```

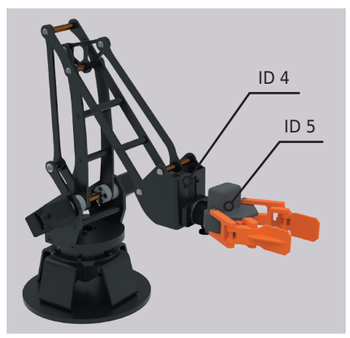

РОБОТ 2

Работа с плоскопараллельным манипулятором (4 степени свободы)

Это манипулятор смешанной или параллельной архитектуры, рабочее звено которого связано со стойкой несколькими кинематическими цепями (ногами). Ключевая особенность – его рабочее звено движется в параллельной плоскости относительно основания. 4 *DOF* означают 3 перемещения и 1 ориентацию.

Примеры и применение: Широко используются в высокоскоростной сборке, монтаже электронных компонентов на платы, паллетировании. Их кинематика оптимизирована для движений в горизонтальной плоскости.





Соберите робот-манипулятор в соответствии с Инструкцией по сборке. Подключите нижний сервопривод робота к робототехническому контроллеру. Ниже приведен скетч для позиционирования захвата робота-манипулятора путем решения задачи обратной кинематики. То есть вам необходимо задать координаты желаемого положения захвата робота в пространстве *x, y* и *z* путем ввода значений в строку монитора *Arduino IDE.*

При работе с предоставленным скетчем необходимо учесть:

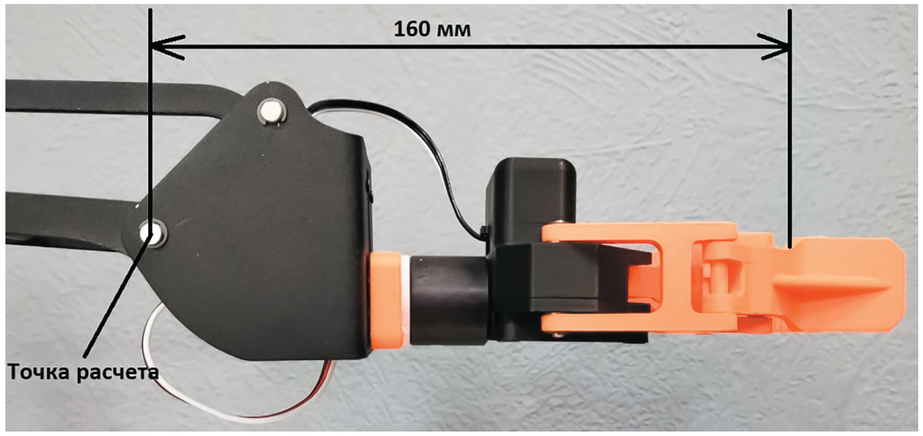
- **Обратите внимание, в этом скетче предоставлен пример ввода значений БЕЗ ЗАПЯТОЙ между значениями. Значения координат *X Y Z* вводятся через ПРОБЕЛ.**
- **Прежде чем начать работать с приведенным скетчем, необходимо, как и для предыдущего робота, выставить значения *dead zone* для сервоприводов.** Значения:

```
ID1 = minAngle = 1500, maxAngle = 2700
ID2 = minAngle = 1850, maxAngle = 2700
ID3 = minAngle =  950, maxAngle = 2250
```

Эти значения прошиты в код программы, и без их установки программа будет выполняться некорректно.

```
Position[0] = constrain(pos0, 1500, 2700);
Position[1] = constrain(pos1, 1850, 2700);
Position[2] = constrain(pos2, 950, 2250);
```

- Точкой для расчета координат позиционирования является середина фланца сустава робота-манипулятора, БЕЗ УЧЕТА ДЛИНЫ ЗАХВАТА. Т. е. при задании положения захвата необходимо к значению по оси *X* прибавлять 160 мм.




```
#include <SMS_STS.h>

const float pi = M_PI;
#define FOREARM_LEN 160 // l1
#define ARM_LEN 185     // l2

SMS_STS servo;
byte ID[5] = {1, 2, 3, 4, 5};                    // Массив ID сервоприводов
s16 Position[5] = {2048, 2048, 2048, 2048, 2048};  // Массив положений сервоприводов
u16 Speed[5] = {400, 400, 400, 400, 400};          // Массив скоростей сервоприводов
byte ACC[5] = {10, 10, 10, 10, 10};                // Массив ускорений сервоприводов

void setup() {
  Serial.begin(9600);
  Serial1.begin(1000000);
  servo.pSerial = &Serial1;

  servo.SyncWritePosEx(ID, 5, Position, Speed, ACC);
  delay(1000);
  Serial.println("\nВведите координаты в формате: x y z");
}

void loop() {
  if (Serial.available() > 0) {
   String input = Serial.readStringUntil('\n');
    input.trim();
    int firstSpaceIndex = input.indexOf(' ');
    int secondSpaceIndex = input.indexOf(' ', firstSpaceIndex + 1);
    if (firstSpaceIndex > 0 && secondSpaceIndex > firstSpaceIndex) {
      float x = input.substring(0, firstSpaceIndex).toFloat();
      float y = input.substring(firstSpaceIndex + 1, secondSpaceIndex).toFloat();
      float z = input.substring(secondSpaceIndex + 1).toFloat();
      MoveRobot(x, y, z, 300, 10);

      Serial.println("\n Введите координаты в формате: x y z");
    }
  }
}

bool MoveRobot(float x, float y, float z, float speed, float acc) {
  Serial.println("Введено: " + String(x) + ", " + String(y) + ", " +  String(z));

  // Заполнение массива скоростей и ускорений для сервоприводов
  for (int i = 0; i < 5; i++) {
    Speed[i] = speed;
    ACC[i] = acc;
  }

  x = (x - 165) / 100.0;
```

```
  y = y / 100.0;
  z = z / 100.0;
  float l1 = FOREARM_LEN / 100.0;
  float l2 = ARM_LEN / 100.0;

  // Длина вектора от основания манипулятора до его
  float len_vec = sqrtf( x*x + z*z );

  if (len_vec > l1 + l2 || len_vec < fabsf(l1 - l2)) {
    Serial.println("Точка недостижима");
    return false;
  }

  float alpha = (acosf( (l2*l2 - l1*l1 - len_vec*len_vec) / (-2 * l1 * len_vec) )) *
(180 / pi);
  float beta = (acosf( (len_vec*len_vec - l2*l2 - l1*l1) / (2 * l1 * l2) )) * (180 /
pi);
  float theta = (atan2f(z, x)) * (180 / pi);

  float angleBase = atan2f(y, x) * (180 / pi);
  float angle1 = alpha + theta;
  float angle2 = beta - angle1;

  s16 pos0 = 2047 + angleBase / 0.088;
  s16 pos1 = 2047 + (90 - angle1) / 0.088;
  s16 pos2 = 2047 - angle2 / 0.088;

  Position[0] = constrain(pos0, 1500, 2700);
  Position[1] = constrain(pos1, 1850, 2700);
  Position[2] = constrain(pos2, 950, 2250);

  if (pos0 != Position[0] || pos1 != Position[1] || pos2 != Position[2]) {
    Serial.println("Точка недостижима из-за физических ограничений");
    return false;
  }

  servo.SyncWritePosEx(ID, 3, Position, Speed, ACC);
}

float radToDeg(float rad) {
  return rad * 180 / PI;
}
```


Лабораторная работа

№ 4 УПРАВЛЕНИЕ МАНИПУЛЯТОРОМ С ПОМОЩЬЮ BLUETOOTH-МОДУЛЯ

Цель работы: *Изучить возможность управления манипулятором по интерфейсу Bluetooth.*

Краткие теоретические сведения

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

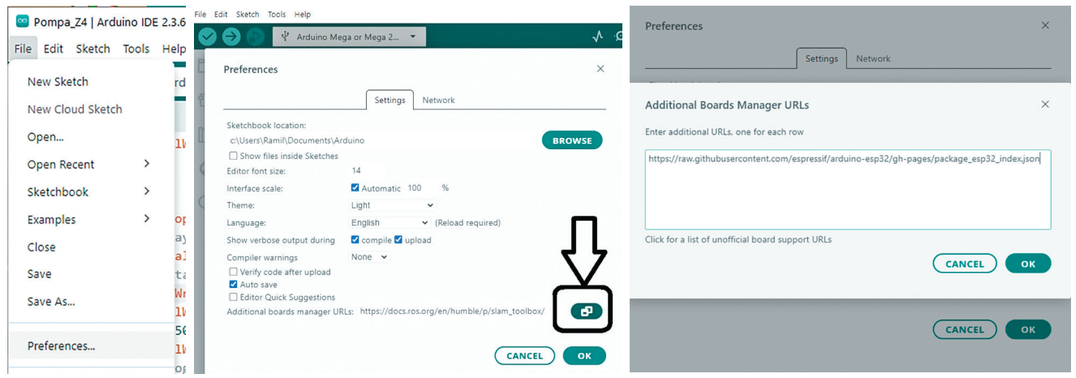
Беспроводной интерфейс с небольшим радиусом действия, получивший название *Bluetooth,* был разработан в 1994 году инженерами шведской компании *Ericsson.* Протокол получил своё название в честь Гарольда Синезубого – короля Дании, который примирил знатные семьи из Дании и Норвегии. Начиная с 1998-го развитием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

В зависимости от расстояния, *Bluetooth*-датчики делятся на три класса:

- Первый класс, способный поддерживать устойчивую связь на расстоянии 100–200 метров. В бытовых устройствах встречается редко и используется на промышленном оборудовании.
- Второй класс удерживает стабильную связь на расстоянии 10–20 метров. Такие датчики чаще всего установлены в смартфонах или планшетах.
- Третий класс наименее мощный и подходит для объединения устройств на расстоянии до пяти метров. Устанавливается на небольших гаджетах – фитнес-браслетах, умных часах и так далее.

Задача: *На примере приведенного скетча научиться писать программы с использованием встроенного беспроводного модуля Bluetooth на базе EP32.*

Порядок выполнения работы

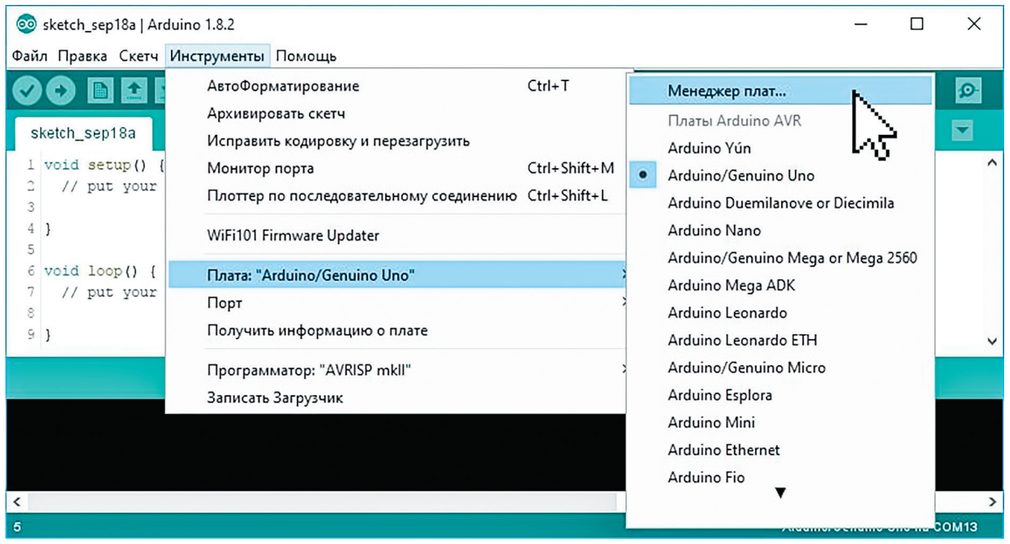


Прежде всего необходимо подготовить компьютер для работы с встроенным в контроллер модулем *ESP32*. Для этого необходимо открыть *Arduino IDE*. Далее выбрать пункт *File – Preferences.*

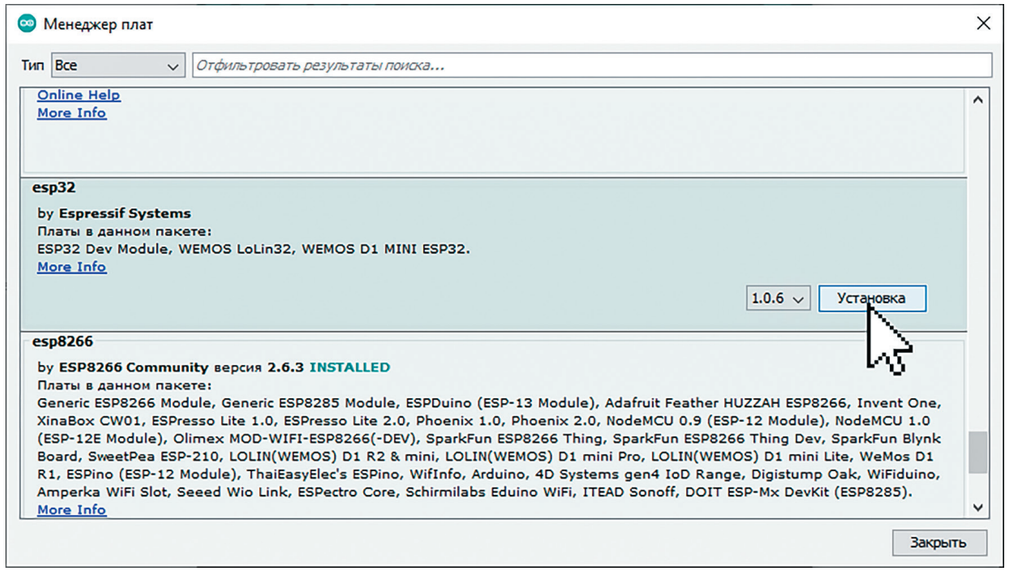
В открывшемся окне выбрать значок внизу экрана и в следующем открывшемся окне ввести ссылку на необходимые драйверы для поддержки *ESP32:*

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

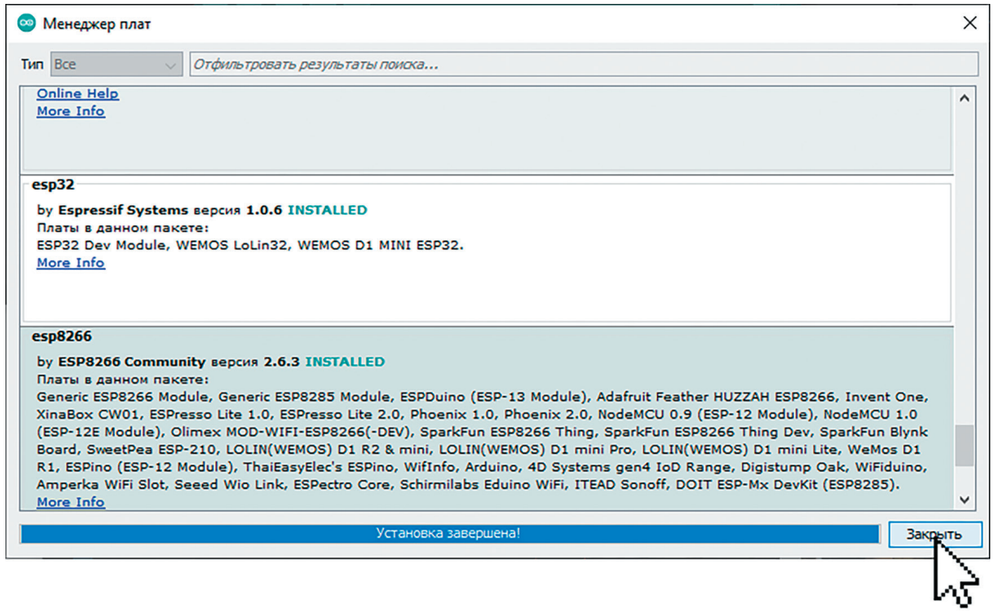
Нажмите кнопку «ОК». Запустите «Менеджер плат», выбрав пункт меню «Инструменты» > «Плата» > «Менеджер плат».



В открывшемся окне «Менеджер плат» выберите пункт *esp32 by Espressif Systems* из списка сборок и нажмите на кнопку «Установка» (при желании можно выбрать версию устанавливаемой сборки).



Дождитесь окончания установки сборки.



После успешной установки сборки в списке напротив её названия *esp32 by ESP32 Systems* появится фраза *INSTALLED.* Закройте «Менеджер плат», нажав на кнопку «Закрыть».

На этом настройка *Arduino IDE* завершена. Теперь в списке плат есть раздел *«ESP32 Arduino.*

В качестве управляющего устройства для робота-манипулятора воспользуемся смартфоном с установленным на нем *Bluetooth Terminal,* например, таким

Теперь рассмотрим структуру программируемой функциональности робота-манипулятора.

Смартфон	Bluetooth module	IoT контроллер	Робот-манипулятор
(приложение Bluetooth Terminal)	(ESP32 module)	Arduino Mega 2560	Сервоприводы

1. Установите из *Google Play* приложение на смартфон *Bluetooth Terminal.*
2. Напишем скетч для приема команд по интерфейсу *Bluetooth* для модуля *ESP32.*

Основная задача *ESP32:*

ESP32 выполняет роль «моста» для передачи команд от мобильного приложения (через *Bluetooth*) к *Arduino Mega* для управления роботом. В данном случае используется классический *Bluetooth (SPP – Serial Port Profile)* для связи.

Подключение к *Bluetooth:*

- Мы используем библиотеку *BluetoothSerial,* которая позволяет *ESP32* работать как *Bluetooth*-устройство.
- В функции *setup()* мы инициализируем *Bluetooth* с именем *ESP32-Robot*, и, если инициализация прошла успешно, *ESP32* начинает работать как *Bluetooth*-сервер, ожидающий подключение от мобильного устройства.

```
    if (!SerialBT.begin("ESP32-Robot")) {  // Имя Bluetooth-устройства
     Serial.println("Ошибка при инициализации Bluetooth!");
     while (1);
}
```

Обработка данных от мобильного устройства:

Когда мобильное приложение отправляет данные по *Bluetooth,* мы получаем их с помощью *SerialBT.readString().* После этого данные передаются на *Arduino Mega* через *Serial2.write(),* используя *UART (Serial2)* с определенными пинами *RX* и *TX.*

```
if (SerialBT.available()) {
     String command = SerialBT.readString();
     if (command.length() > 0) {
          Serial.print("Получена команда от Bluetooth: ");
          Serial.println(command);
          Serial2.write((const uint8_t*)command.c_str(), command.length());  // Отправляем команду на Arduino Mega
       }
}
```

Обработка данных от *Arduino Mega:*

В случае, если *Arduino Mega* отправляет ответ (например, подтверждение команды или данные о статусе робота), *ESP32* получает эти данные через *Serial2.readString()* и передает их обратно на мобильное устройство через *Bluetooth.*

```
if (Serial2.available()) {
     String response = Serial2.readString();
     if (response.length() > 0) {
          Serial.print("Ответ от Arduino Mega: ");
          Serial.println(response);
           SerialBT.write((const uint8_t*)response.c_str(), response.length());  // Отправляем ответ обратно по Bluetooth
       }
}
```


Используемые пины:

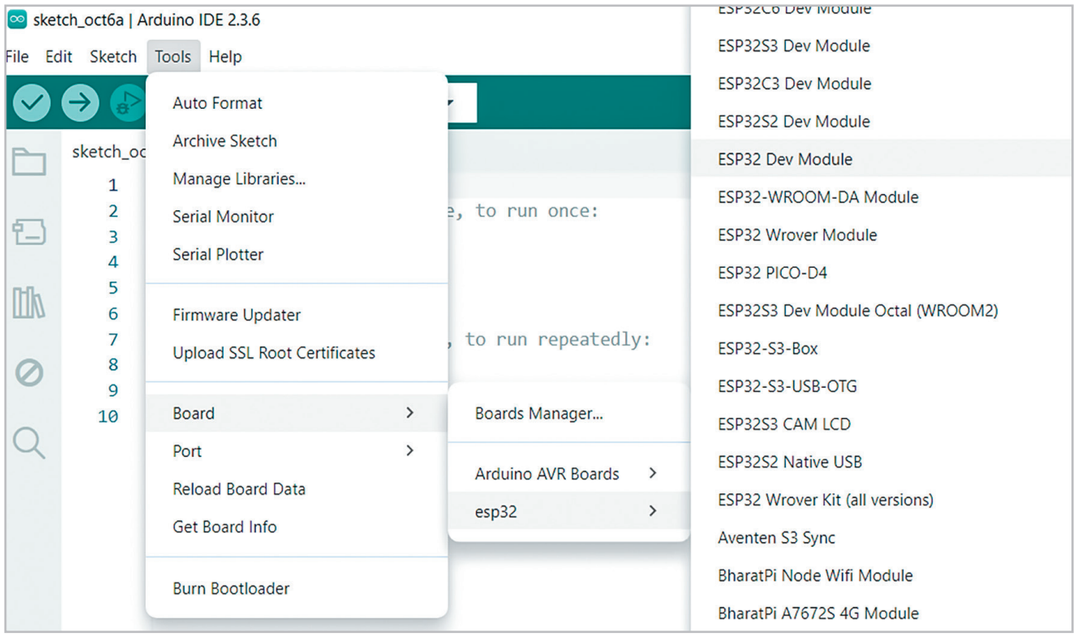
- *RX_PIN = 17:* Пин для получения данных от *Arduino Mega.*
- *TX_PIN = 16:* Пин для отправки данных на *Arduino Mega.*

Таким образом *ESP32* взаимодействует с мобильным приложением через классический *Bluetooth,* получает команды, передает их на *Arduino Mega,* а затем отправляет ответы обратно в приложение. Вот пример скетча для *ESP32:*

```
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;

// Пины для подключения с Arduino Mega (RX/TX)
#define RX_PIN 17
#define TX_PIN 16

void setup() {
  // Инициализация Serial для отладки
  Serial.begin(9600);

  // Инициализация Bluetooth
  if (!SerialBT.begin("ESP32-Robot")) {  // Имя Bluetooth-устройства
    Serial.println("Ошибка при инициализации Bluetooth!");
    while (1);
  }
  Serial.println("Bluetooth запущен, ожидаю подключения...");

  // Инициализация Serial2 для работы с Arduino Mega
  Serial2.begin(115200, SERIAL_8N1, RX_PIN, TX_PIN);  // RX=17, TX=16
}

void loop() {
  // Проверка на поступление данных от Bluetooth
  if (SerialBT.available()) {
    String command = SerialBT.readString();
    if (command.length() > 0) {
      Serial.print("Получена команда от Bluetooth: ");
      Serial.println(command);

      // Отправляем команду на Arduino Mega через Serial2
      Serial2.write((const uint8_t*)command.c_str(), command.length());
    }
  }
```

```
    // Проверка на поступление данных от Arduino Mega
    if (Serial2.available()) {
      String response = Serial2.readString();
      if (response.length() > 0) {
        Serial.print("Ответ от Arduino Mega: ");
        Serial.println(response);

        // Отправляем ответ обратно по Bluetooth
        SerialBT.write((const uint8_t*)response.c_str(), response.length());
      }
    }
}
```

Подключите модуль *ESP32* к компьютеру – разъем *Type C* находится слева на контроллере. Запустите *Arduino IDE,* выберите модуль *ESP32* и нужный *COM*-порт.



Загрузите скетч и нажмите кнопку «Загрузить».

3. Теперь рассмотрим программный код, который необходимо загрузить в *IoT* контроллер для *Arduino Mega 2560* для обработки принимаемых команд и отправки их в исполнительное устройство – сервоприводы робота-манипулятора.

Основная задача *Arduino Mega:*

Модуль *Arduino Mega* управляет движением суставов робота-манипулятора. Он получает команды от *ESP32* через *Serial2* и отправляет их сервоприводам для выполнения соответствующих действий.

Инициализация и подключение устройств:

- *Serial2* – для получения команд от *ESP32* через *Bluetooth.*
- *Serial1* – для управления сервомоторами манипулятора.

Сервомоторы манипулятора управляются через библиотеку *SCServo*, которая отправляет команды для перемещения сервомоторов.

```
Serial2.begin(115200); // Инициализация Serial2 для получения команд от ESP32
Serial1.begin(1000000); // Инициализация Serial1 для управления сервоприводами
```

Обработка команд от *ESP32:*

В основном цикле (в функции *loop*) *Arduino Mega* принимает команды от *ESP32* через *Serial2.* Команды – это простые строки, например, *MF, MB, ML, MR* для управления манипулятором.

- Движение захвата вверх и вниз.
- Движение захвата влево и вправо.
- Движение захвата вперед и назад.
- Установка в начальную позицию.

```
void manipLeft() {
Position[0] = max(BASE_MIN, Position[0] - 100);
sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]); }
void manipRight() {
Position[0] = min(BASE_MAX, Position[0] + 100);
sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]); }

void manipForward() {
Position[1] = min(LEFT_MAX, Position[1] + 150);
Position[2] = min(RIGHT_MAX, Position[2] + 30);
sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]); }

void manipBackward() {
Position[1] = max(LEFT_MIN, Position[1] - 150);
Position[2] = max(RIGHT_MIN, Position[2] - 30);
sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]); }

void manipUp() {
Position[1] = max(LEFT_MIN, Position[1] - 100);
Position[2] = min(RIGHT_MAX, Position[2] + 150);
sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]); }
```

```
   void manipDown() {
   Position[1] = min(LEFT_MAX, Position[1] + 100);
   Position[2] = max(RIGHT_MIN, Position[2] - 150);
   sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
   sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]); }

   void manipInitialPosition() {
   Position[0] = 2047;
   Position[1] = 2111;
   Position[2] = 2047;
   sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]); sms_sts.WritePosEx(ID[1], Position[1],
Speed[1], ACC[1]); sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]); }
```

Ниже приведен пример скетча для управления манипулятором. Обратите внимание, что в программе введены программные ограничения для движения валов сервоприводов:

```
   const int BASE_MIN = 1000, BASE_MAX = 3000;
   const int LEFT_MIN = 2100, LEFT_MAX = 3100;
   const int RIGHT_MIN = 1300, RIGHT_MAX = 2700;
   const int GRAB_MIN = 1300, GRAB_MAX = 1900;
```

Вот пример скетча:

```
#include <SCServo.h>
SMS_STS sms_sts;

const int BASE_MIN = 1000, BASE_MAX = 3000;
const int LEFT_MIN = 2100, LEFT_MAX = 3100;
const int RIGHT_MIN = 1300, RIGHT_MAX = 2700;
const int GRAB_MIN = 1300, GRAB_MAX = 1900;

byte ID[5] = {1, 2, 3, 4, 5};
s16 Position[3] = {2047, 2111, 2047};
u16 Speed[3] = {700, 700, 700};
byte ACC[3] = {50, 50, 50};

void setup() {
    Serial2.begin(115200);
    Serial1.begin(1000000);
    sms_sts.pSerial = &Serial1;
    Serial.begin(9600);

   delay(2000);
```

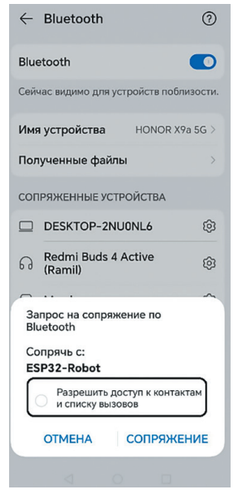


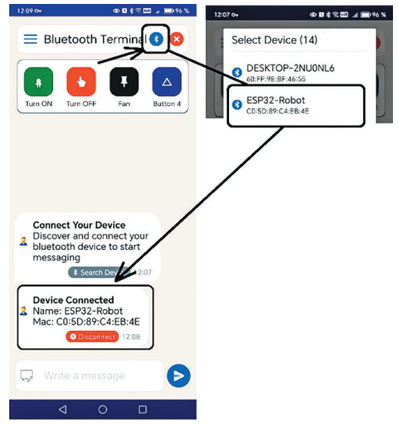
```
    for (int i = 0; i < 3; i++) {
        sms_sts.WritePosEx(ID[i], Position[i], Speed[i], ACC[i]);
        delay(100);
    }
}

void loop() {

    if (Serial2.available()) {
        String command = Serial2.readStringUntil('\n'); // Читаем строку до перевода
строки
        command.trim(); // Убираем лишние пробелы и переводы строки

        Serial.print("Получена команда: ");
        Serial.println(command);

          if (command == "MF") { manipForward(); }
        else if (command == "MB") { manipBackward(); }
        else if (command == "ML") { manipLeft(); }
        else if (command == "MR") { manipRight(); }
        else if (command == "MU") { manipUp(); }
        else if (command == "MD") { manipDown(); }
        else if (command == "IP") { manipInitialPosition(); }
        else if (command == "MW") { manipWider(); }
        else if (command == "MT") { manipTinner(); }
        else { Serial.println("⚠ Неизвестная команда!"); }
    }
}

void manipLeft() {
    Position[0] = max(BASE_MIN, Position[0] - 50);
    sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]);
}

void manipRight() {
    Position[0] = min(BASE_MAX, Position[0] + 50);
    sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]);
}

void manipForward() {
    Position[1] = min(LEFT_MAX, Position[1] + 150);
    Position[2] = min(RIGHT_MAX, Position[2] + 30);
    sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
```

```
    sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]);
}

void manipBackward() {
    Position[1] = max(LEFT_MIN, Position[1] - 150);
    Position[2] = max(RIGHT_MIN, Position[2] - 30);
    sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
    sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]);
}

void manipUp() {
    Position[1] = max(LEFT_MIN, Position[1] - 100);
    Position[2] = min(RIGHT_MAX, Position[2] + 150);
    sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
    sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]);
}

void manipDown() {
    Position[1] = min(LEFT_MAX, Position[1] + 50);
    Position[2] = max(RIGHT_MIN, Position[2] - 150);
    sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
    sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]);
}

void manipInitialPosition() {
    Position[0] = 2047;
    Position[1] = 2111;
    Position[2] = 2047;
    sms_sts.WritePosEx(ID[0], Position[0], Speed[0], ACC[0]);
    sms_sts.WritePosEx(ID[1], Position[1], Speed[1], ACC[1]);
    sms_sts.WritePosEx(ID[2], Position[2], Speed[2], ACC[2]);
}

void manipWider() {
    sms_sts.WritePosEx(ID[4], GRAB_MAX, Speed[2], ACC[2]);
}
```

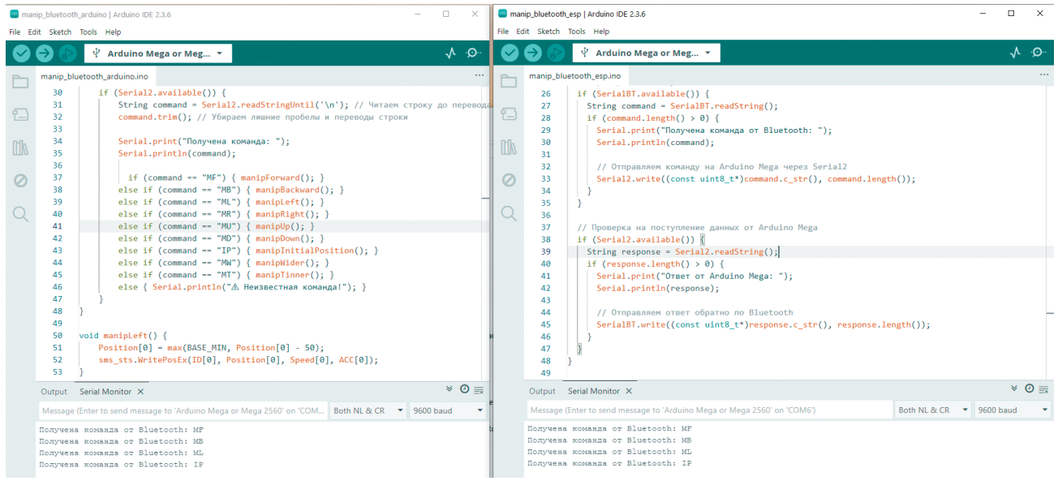
Для управления роботом-манипулятором включите режим *Bluetooth* на смартфоне, подключитесь к роботу. Небходимо дать разрешение на доступ к контактам.

Загрузите приложение *Bluetooth Terminal*. Подключитесь к роботу в приложении. Теперь можно вводить команды в строке ввода. Убедитесь в работе программы.





Слева приведен экран монитора *Arduino IDE* для *Arduino Mega 2560*, справа – для *ESP* модуля.



Техническое зрение роботов. Модуль технического зрения (МТЗ) робота-манипулятора

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

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

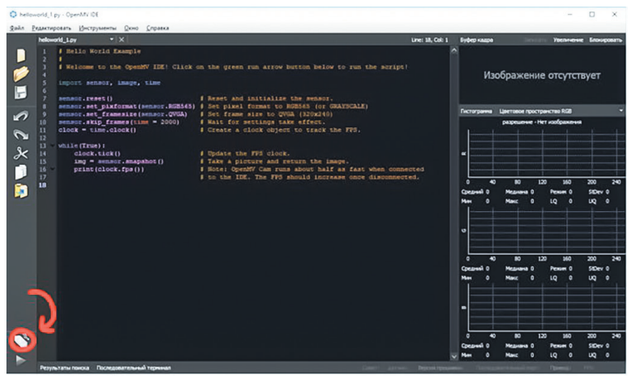
Начинка справляется с алгоритмами компьютерного зрения разной сложности, среди которых:

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

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

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

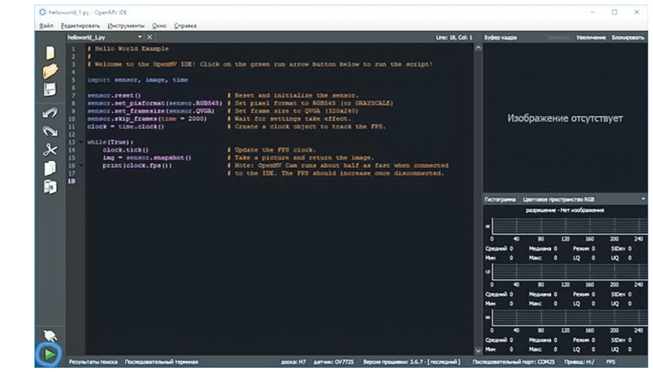
Подключение и настройка *OpenMV IDE*



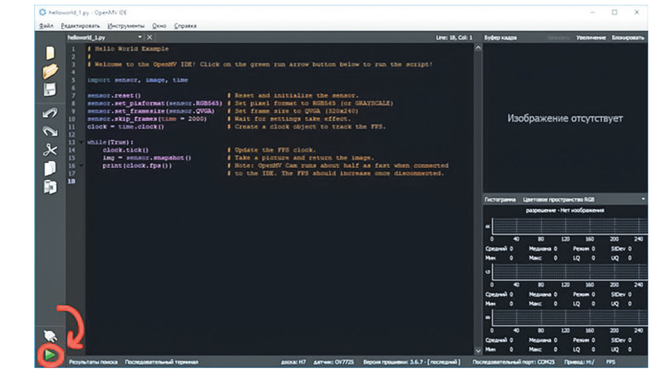
Подключите камеру модуля технического зрения к компьютеру через разъём *micro-USB*.

- Скачайте и установите программу *OpenMV IDE (https://openmv.io/pages/download/)*
- Откройте среду *OpenMV IDE.*

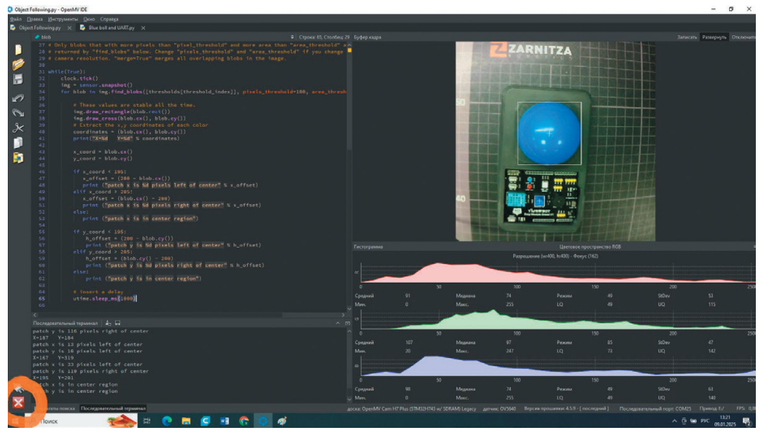
- В среде разработки нажмите на иконку подключения к плате.
- В случае успешного подключения загорится зелёный треугольник.



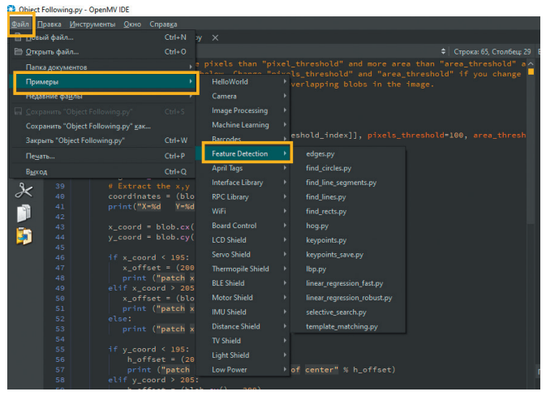
- По умолчанию при открытии *IDE* уже открыт тестовый пример *helloworld.py.*
- Нажмите на зелёный треугольник для запуска скрипта.



- При успешном запуске скрипта треугольник сменится на крестик, а в окне изображения вы увидите картинку с камеры.



Для остановки программы нажмите на крестик. Выбрав пункт меню «Файл» – «Примеры» можно найти примеры программ, реализующих основные функции камеры модуля технического зрения.



Более подробно ознакомиться с комплексом *OpenMV IDE* и программированию на *MicroPython* можно на сайте:

https://wiki.amperka.ru/articles:openmv-ide-install
https://docs.openmv.io/openmvcam/tutorial/openmvide_overview.html
https://docs.openmv.io/

РОБОТ 3

Работа с DELTA-роботом (3 степени свободы)

Это пространственный параллельный манипулятор. Состоит из неподвижного основания и небольшой легкой подвижной платформы. К основанию крепятся три рычага (часто называемые плечами), каждый из которых приводится в движение отдельным мотором. С каждым рычагом через параллелограммный механизм соединена одна из «ног», ведущих к платформе. Параллелограммы обеспечивают постоянную ориентацию платформы.

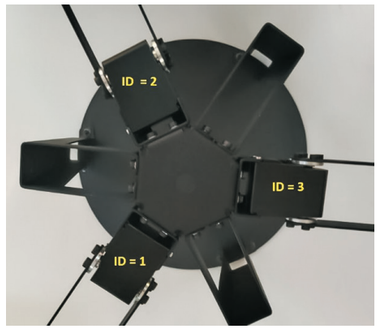
Кинематическая схема: Архитектура *RSS* или *RRR* (в зависимости от реализации параллелограмма). *Основная схема:* 3 вращательных привода на основании -> 3 рычага -> 3 параллелограмма -> подвижная платформа.

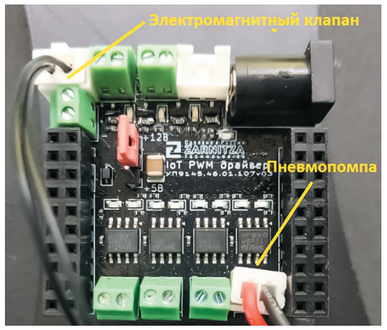
Для изучения возможностей дельта-робота используем комбинацию совместно с модулем технического зрения. Соберите дельта-робот в соответствии с Инструкцией по сборке, закрепите штатив с камерой, подключите контроллер к компьютеру с помощью кабеля *USB*.

В первую очередь необходимо подготовить сервоприводы к работе. Установите робот в начальное положение, как показано на рисунке.

Теперь пропишите ID сервоприводов и значение среднего положения вала каждого сервопривода. Процесс подготовки сервоприводов был рассмотрен в предыдущих работах.

Для подключения пневмопомы и электромагнитного клапана подготовлены разъемы на плате расширения робототехнического контроллера с пинами 8, 9 и 10 соответственно.





Для подключения модуля технического зрения на контроллер необходимо установить дополнительную плату расширения – интерфейс *UART, I²C*. Соединение с модулем технического зрения производится с помощью входящего в комплект 4-пинового кабеля.

Для ознакомления с возможностями манипулирования объектами при помощи *Delta*-робота с использованием библиотеки *<DeltaKinematics.h>* ниже представлен пример для перемещения кубика по трем точкам в плоскости нижней пластины (основания) дельта-робота.

```
#include <DeltaKinematics.h>
#include <SCServo.h>

#define POMPA_1_PIN 8  // Пины помпы (ШИМ)
#define POMPA_2_PIN 9
#define KLAP_PIN 10    // Пин клапана
#define MAX_POINTS 10  // Максимальное кол-во точек
#define CUBE_HEIGHT 40 // Высота кубика (мм)
#define ERROR_STEP 10  // Допустимая погрешность положения сервопривода

SMS_STS servo; // Объект sms_sts для сервоприводов
DeltaKinematics DK(100, 236, 74.9, 22.4); // Объект DeltaKinematics для расчета
кинематики робота

byte ID[3] = {1, 2, 3};                 // Массив ID сервоприводов
s16 Position[3] = {2047, 2047, 2047};   // Массив положений сервоприводов
u16 Speed[3] = {1800, 1800, 1800};      // Массив скоростей сервоприводов
byte ACC[3] = {10, 10, 10};             // Массив ускорений сервоприводов

// Координаты начальной точки
float xBase = 0; float yBase = 0; float zBase = -250;
float baseCoord[3] = { xBase, yBase, zBase };

unsigned long startMillis;    // Количество мс, отсчитывающих "начало" процесса
unsigned long currentMillis;  // Количество мс на текущий момент времени
```



```
const unsigned long servoPeriod = 1500;  // Период (в мс) / ограничение по времени для
процесса

void setup() {
  Serial.begin(115200);     // Для ввода координат с "Монитора порта"
  Serial1.begin(1000000);  // Для сообщения с сервоприводами
  servo.pSerial = &Serial1;

  pinMode(POMPA_1_PIN, OUTPUT); // Помпа (изначально выключена)
  pinMode(POMPA_2_PIN, OUTPUT);
  analogWrite(POMPA_1_PIN, 0);
  analogWrite(POMPA_2_PIN, 0);

  pinMode(KLAP_PIN, OUTPUT);   // Клапан (изначально закрыт)
  digitalWrite(KLAP_PIN, LOW);
}

void loop() {
  // Перенос кубика между трех точек с использованием насоса
  int num_points = 3;
  float x_Points[MAX_POINTS] = {50, -45, -45};
  float y_Points[MAX_POINTS] = {0, -45, 45};
  MoveOneCube(x_Points, y_Points, num_points);
}
// Перемещение куба по нескольким точкам с использованием насоса
void MoveOneCube(float X[], float Y[], int num_points) {
   // Перемещение в начальную точку
  MoveRobot(baseCoord[0], baseCoord[1], baseCoord[2] + 15 + (2 * CUBE_HEIGHT));

  for (int i = 0; i < num_points; i++) {
    // Взятие кубика
    MoveRobot(X[i], Y[i], baseCoord[2] + 15 + (2 * CUBE_HEIGHT));
    MoveRobot(X[i], Y[i], baseCoord[2] + 25 + CUBE_HEIGHT);
    analogWrite(POMPA_1_PIN, 70); // Включение помпы
    analogWrite(POMPA_2_PIN, 0);
    delay(70);
    digitalWrite(KLAP_PIN, HIGH);  // Открытие клапана
    delay(70);
    MoveRobot(X[i], Y[i], baseCoord[2] + 5 + CUBE_HEIGHT);
    MoveRobot(X[i], Y[i], baseCoord[2] + 15 + (2 * CUBE_HEIGHT));
    delay(70);

    // Перемещение кубика в точку
```

```
    // (Третья точка переносит куб в первую, создавая цикл)
    if (i != 2) {
      MoveRobot(X[i + 1], Y[i + 1], baseCoord[2] + 20 + (2 * CUBE_HEIGHT));
      MoveRobot(X[i + 1], Y[i + 1], baseCoord[2] + 17 + CUBE_HEIGHT);
    } else {
      MoveRobot(X[0], Y[0], baseCoord[2] + 20 + (2 * CUBE_HEIGHT));
      MoveRobot(X[0], Y[0], baseCoord[2] + 17 + CUBE_HEIGHT);
    }
    analogWrite(POMPA_1_PIN, 0);   // Выключение помпы
    analogWrite(POMPA_2_PIN, 0);
    delay(100);
    digitalWrite(KLAP_PIN, LOW);  // Закрытие клапана
    delay(100);
  }
}

// Вычисление кинематики Дельта-манипулятора и перемещение в точку
void MoveRobot(float x, float y, float z) {
  int r = 94;              // Радиус окружности, ограничивающей передвижение рабочего
органа
  double eps = 0.01;       // Погрешность для работы с числами типа float
int current_pos[3];      // Массив для хранения текущего положения сервоприводов
bool finished = false;   // Флажок для выхода из цикла "while"

startMillis = millis(); // Получение количества мс, задающих "начало" движения
Serial.println("Введено: " + String(x) + ", " + String(y) + ", " + String(z));

// Проверка соответствия ограничениям окружности
if ((pow(x, 2) + pow(y, 2)) > (pow(r, 2) + eps)) {
  Serial.println("Вне границ окружности");
  Serial.println();

  return;
}
// Проверка корректности вводимой высоты
if (z < zBase || z > -135) {
  Serial.println("Высота не та");
  Serial.println();
  return;
}

DK.inverse(x, y, z);  // Вычисление обратной кинематики дельта-манипулятора
Serial.println("Углы " + String(DK.a) + ", " + String(DK.b) + ", " + String(DK.c));
```



```
  // Проверка на возможность вычисления обратной кинематики для заданной точки
  if ((DK.a == 0) && (DK.b == 0) && (DK.c == 0) || (DK.z > 0)) {
    Serial.println("Точка недостижима");
    return;
  }

  // Вычисление положения сервоприводов (2048 - центральное положение, а DK.a, DK.b,
DK.c - углы поворота)
  int pos1 = 2048 + DK.a * 11.38;
  int pos2 = 2048 + DK.b * 11.38;
  int pos3 = 2048 + DK.c * 11.38;

  Position[0] = constrain(pos1, 1200, 2500);
  Position[1] = constrain(pos2, 1200, 2500);
  Position[2] = constrain(pos3, 1200, 2500);

  Serial.println("Position: " + String(pos1) + ", " + String(pos2) + ", " +
String(pos3));

  if (pos1 != Position[0] || pos2 != Position[1] || pos3 != Position[2]) {
    Serial.println("Точка не достижима при текущих ограничениях сервоприводов");
    return;
  }

  // Перемещение сервопривода
  servo.SyncWritePosEx(ID, 3, Position, Speed, ACC);

  // Пока не истечет период "finished" (или пока сервопривод не придет в заданную
точку)
  while (!finished) {
    currentMillis = millis();  // Получение количества мс на текущий момент времени

    // Проверка истечения периода "servoPeriod"
    if (currentMillis - startMillis < servoPeriod) {
      for (int i = 0; i < 3; i ++) {
        if (servo.FeedBack(ID[i]) != -1) {
          // Получение текущих положений сервоприводов
          current_pos[i] = servo.ReadPos(ID[i]);
        }
      }
      // Если все сервоприводы находятся в нужных положениях, то ожидание истечения
периода "servoPeriod" прерывается
      if (abs(current_pos[0] - Position[0]) < ERROR_STEP && abs(current_pos[1] -
```

```
Position[1]) < ERROR_STEP && abs(current_pos[2] - Position[2]) < ERROR_STEP) {
        finished = true;
      }
    } else {
      // Истекло время ожидания перемещения сервоприводов
      finished = true;
    }
  }
  Serial.println();
}
```

Ниже приведен пример написания программы для совместной работы робота с техническим зрением. Задача сводится к поиску кубика на поверхности нижней пластины (основания) дельта-робота и перенос найденного кубика в угол рабочего пространства манипулятора.

1. Настройка камеры

OpenMV H7 Plus – это модуль технического зрения, наделенный широким спектром функций, как, например, поиск объекта по его цвету. Он позволяет не только использовать уже готовые примеры кодов, но также предлагает пользователям возможность писать свои собственные алгоритмы на языке *MicroPython* и сохранять его в памяти камеры при помощи специализированного *IDE (OpenMV IDE).*

Задача поиска кубика предполагает, что находящаяся возле дельта-робота камера будет постоянно искать кубик, выделяя его среди остальных объектов на изображении с помощью информации о его цвете. И настройка камеры предполагает именно обучение *OpenMV H7 Plus* на нужный оттенок, а также подбор определенных коэффициентов для работы алгоритма.

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

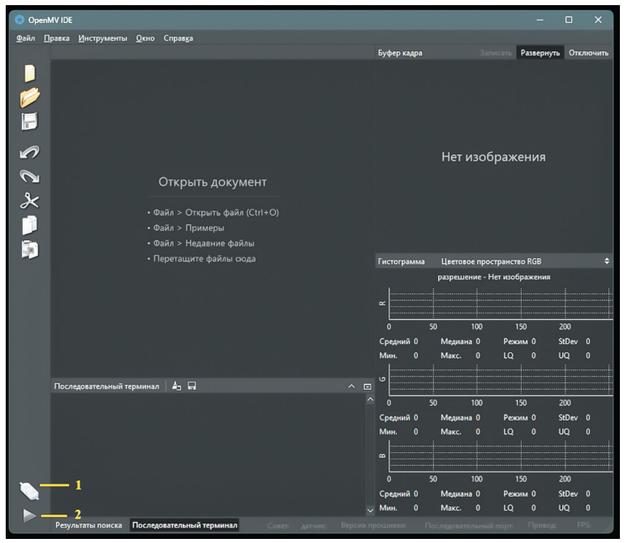


Рисунок 1. Общий интерфейс приложения *OpenMV IDE*

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

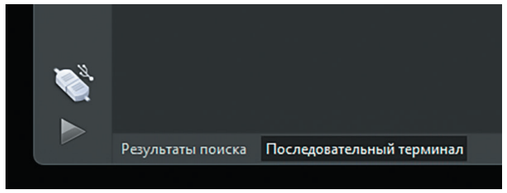


Рисунок 2. Значок подключения к камере

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

Примечание: Необходимо использовать камеру с подключенной *SD*-картой для загрузки необходимых для корректной работы алгоритмов папок и скетчей. Для того, чтобы вставить *SD*-карту, необходимо снять модуль технического зрения. Извлеките плату модуля технического зрения, открутив два фиксирующих самореза.

Вставьте *SD*-карту в соответствующий слот и установите плату назад, зафиксировав ее саморезами.

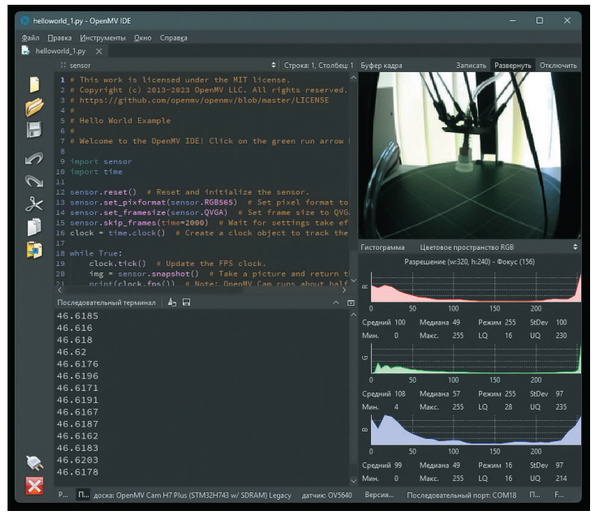


Рисунок 3. Интерфейс приложения *OpenMV IDE* при корректно подключенной камере

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

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

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

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

1.1. Калибровка параметров обработки изображения

Откройте файл *calibrate_white_balance.py* с предоставленного электронного носителя. Данный код позволяет включать и выключать автоматическую коррекцию параметров обработки изображений с камеры: баланса белого, экспозиции, усиления. В рассмотренном примере *HelloWorld* эти параметры были включены по умолчанию, и в зависимости от освещения и яркости окружения картинка на экране автоматически менялась. Данная функция полезна, но не в контексте отслеживания объекта по цвету, так как при смене данных параметров меняются оттенки цветов на изображении, и поэтому искомый кубик может не опознаваться корректно.

Предполагается, что дельта-робот работает при детерминированных условиях освещенности, и поэтому автонастройка параметров камеры в приведенных в данной работе программных алгоритмах отключена. Представленные в коде значения параметров баланса белого, экспозиции и усиления можно изменять под конкретные условия вашего окружения при помощи файла *calibrate_white_balance.py.*

Откройте файл *calibrate_white_balance.py* и запустите его. При запуске данного кода в течение пяти секунд (строка № 6) автоматическая калибровка параметров обработки изображения будет включена, и вы сможете, вращая камеру, найти оптимальные условия для вашей камеры. Как только вы найдете состояние, в котором вас будет устраивать отображаемая в *IDE* картинка, задержите камеру, пока не увидите в последовательном терминале (рис. 4) сообщение «Отключение автоматической корректировки параметров камеры». После этого параметры усиления, баланса белого и экспозиции будут выводиться в терминал раз в секунду. Выпишите себе строку с данными параметрами, чтобы использовать в последующем коде.

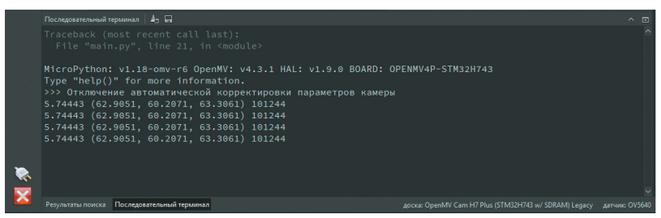


Рисунок 4. Результат работы кода *calibrate_white_balance.py*

1.2. Обучение на цвет

Калибровка камеры на заданный цвет кубика происходит при помощи встроенных возможностей модуля технического зрения *OpenMV H7 Plus.* Выберите кубик яркого цвета, контрастирующий с поверхностью основания робота и окружением (стены, шторы и т. д.).

Откройте файл *get_object_thresholds.py* с предоставленного электронного носителя. В строчках № 6-8 подставьте свои значения из строки, полученной при помощи файла *calibrate_white_balance.py* и запустите код. Как пример, рассмотрим строку с рисунка 4 – «5.74443 (62.9051, 60.2071, 63.3061) 101244». Подставим полученные параметры в код следующим образом:

```
sensor.set_auto_gain(False, gain_db=5.74443)
sensor.set_auto_whitebal(False, rgb_gain_db=(62.9051, 60.2071, 63.3061))
sensor.set_auto_exposure(False, exposure_us=101244)
```

Примечание: Если получившееся изображение некорректно, повторите шаг 1.1. и/или проверьте, правильно ли вы подставили полученные значения.

Для того, чтобы натренировать камеру на отслеживание конкретного цвета, наведитесь камерой на нужный вам куб и нажмите кнопку «Стоп» (появляется на месте зелёной кнопки при запуске камеры, рис. 2. № 2). Последнее изображение, полученной *OpenMV H7 Plus,* «застынет» у вас на экране, что будет удобно для калибровки запоминаемого цвета. Найдите на верхней панели «Инструменты» и перейдите «Машинное зрение» → *Threshold Editor.* Во всплывающем окне «Расположение исходного изображения?» выберите «Кадровый буфер», то есть последнее захваченное изображение камерой, которое вы видите в окне «Буфер кадра» в правой части *IDE.*

В открывшемся окне (рис. 5) найдите справа кнопку «Сбросить ползунки» и нажмите ее.

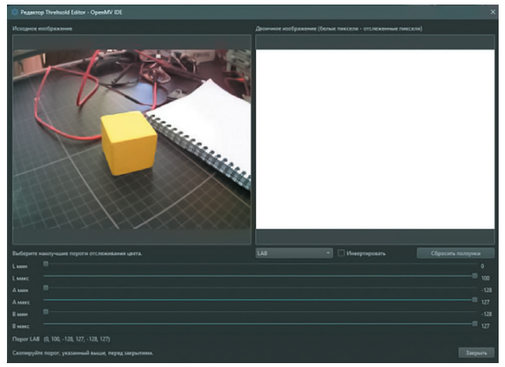


Рисунок 5. Интерфейс *Threshold Editor*

Представленные в нижней части окна ползунки позволяют регулировать пороги отслеживания цвета, чтобы выделять именно тот объект, который нужно отслеживать. Отрегулируйте ползунки таким образом, чтобы только кубик был выделен белым на изображении справа (двоичном) (рис. 6).

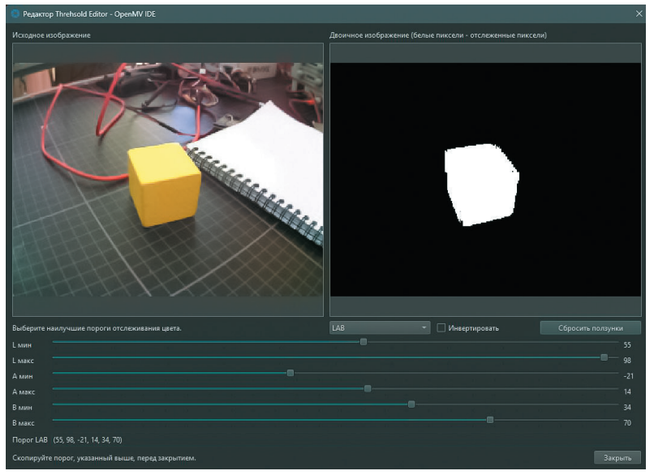


Рисунок 6. Отображение кубика при корректно настроенных порогах остлеживания цвета

Примечание: Наилучший вариант настройки порогов предполагает, что кубик на правом изображении виден полностью и лишние предметы не были выделены вместе с ним. Чем точнее будут настроены параметры на данном этапе, тем лучше будет определяться цвет при дальнейшей работе.

Как только пороговые значения для отслеживания цвета кубика были настроены, выпишите из графы «Порог *LAB*» в нижней части значения в круглых скобках и/или скопируйте строчку полностью. Подставьте полученные числа в переменную *thresholds* в коде файла *get_object_thresholds.py*. Для значений с рисунка 6 переменная будет следующей:

thresholds = [(55, 98, -21, 14, 34, 70)]

Запустите программу и проверьте, что ваш кубик определяется камерой и выделяется в рамку на экране в *IDE* (рис. 7).

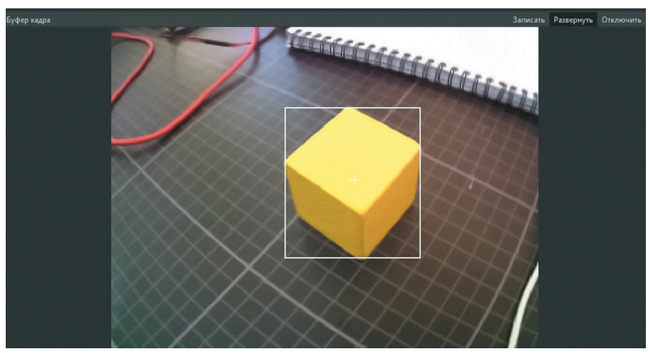
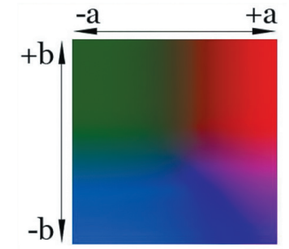


Рисунок 7. Корректное определение кубика по цвету

Примечание: Проверьте, что при одном и том же освещении кубик корректно определяется с разных сторон и под разными углами. Если поиск цвета захватывает другие объекты или находит куб неполностью, повторите настройку порогов и/или замените кубик на более контрастный.

Примечание: Ниже раскроем, что такое *LAB* в значении, которое мы использовали – «Порог *LAB*». *LAB* – аббревиатура названия двух разных (хотя и похожих) цветовых пространств. Более известным и распространенным является *CIELAB* (точнее, *CIE 1976 L*a*b**), другим – *Hunter Lab* (точнее, *Hunter L, a, b*). Таким образом, *Lab* – это неформальная аббревиатура, не определяющая цветовое пространство однозначно. Чаще всего, говоря о пространстве *Lab*, подразумевают *CIELAB.*

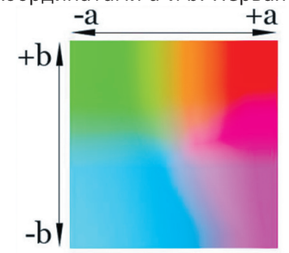


Светлота 25 %

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

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

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



Светлота 75 %

1.3. Определение координат куба

Для определения координат кубика на плоскости нижней платформы дельта-робота используется камера. Она смотрит на основание робота, находит нужный объект и возвращает координаты его центра. Однако возвращаемые ею координаты – это отражение положения объекта в системе координат камеры, а не мировой. То есть необработанные получаемые данные не подходят для использования дельта-роботом сразу, так как камера видит кубик в своей перспективе – под наклоном, и возвращаемые ею показания имеют перспективные искажения. Для коррекции данных искажений используется

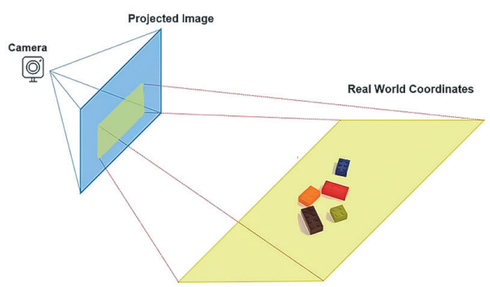


Рисунок 8. Разница перспектив камеры и стола

специальное преобразование из системы координат камеры в мировую. В данном примере рассматривается преобразование гомография, позволяющее «переложить» точки с одной перспективы на другую, зная соответствие (специальное «правило переноса») между ними, выраженное матрицей гомографии.

Инструкция ниже представляет собой краткую выжимку описания, используемого нами проекта для вычисления матрицы гомографии между двумя перспективами. Для более глубокого изучения вопроса вы можете ознакомиться с проектом самостоятельно.

https://github.com/nickoala/openrv/blob/master/homography/README.md

Перенесите на *SD*-карту камеры папки *vec, rv* и файл *mtx.py* с предоставленного электронного носителя. Откройте файл *get_world_coords.py* с предоставленного электронного носителя. Подставьте параметры изображения из шага 1.1 и параметры пороговых значений цвета из шага 1.2 в код, запустите его. На экране *IDE* вы увидите красный крест, который будет играть роль точки отсчета для преобразования координат куба с камеры в мировую систему координат.

Примечание: Заметьте, что изображение с камеры было выровнено и слегка повернуто, корректируя искажение по бокам картинки, создаваемое линзой. Вы можете изменить данные коэффициенты в строчках:

```
img = sensor.snapshot() \
        .lens_corr(1.9) \
        .rotation_corr(x_rotation = -32)
```

Примечание: При работе с гомографией расположение камеры и ее поворот относительно плоскости основания имеют большую роль, поэтому старайтесь сохранять одно и то же положение камеры и штатива для корректной работы алгоритма.

Расположите камеру таким образом, чтобы большая часть основания манипулятора была видна на экране и соотнесите центр красного перекрестия с линиями на нижней платформе дельта-робота так, чтобы они совпадали (рис. 9).

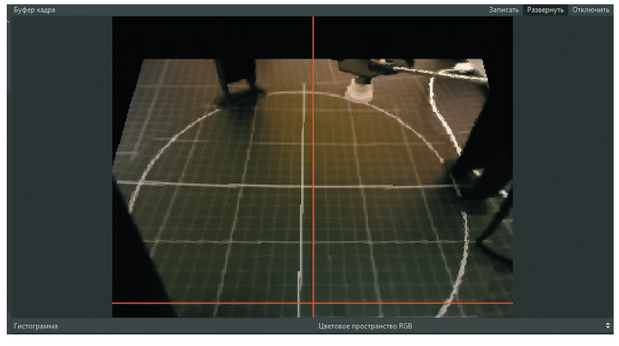


Рисунок 9. Пример расположение перекрестия относительно основания робота

Необходимое условие: направление осей координат *X* и *Y* с камеры должны соответствовать направлению осей координат *X* и *Y Delta*-робота.

Для создания матрицы гомографии создайте файл формата *.example*, отвечающий за «связывание» мировых координат с координатами на изображении следующим образом:

а) Найдите точку на пересечении линий нижней платформы, например, точку с координатами (2, 1) относительно центра красного перекрестия, имеющего координаты (0, 0) (рис. 10). Нажмите на данную точку и запишите появившиеся координаты данной точки в пикселях, взяв их значения в строке «разрешение – точка» ниже буфера кадра изображения, в созданный ранее файл следующим образом: 171 203 2 1.

Примечание: Сначала идут координаты в пикселях: *x = 171, y = 203*, а далее – координаты в единицах измерения (в данном случае это клетки поля на основании робота по 10 мм каждый), где сначала идет *y = 2, x = 1,* т. е. два раза вверх и один раз вправо.

б) Заполните текстовый файл как минимум шестью точками, но при этом учитывайте, что чем больше точек, тем выше точность алгоритма. Нежелательно брать точки, лежащие на одной прямой, как, например, (1, 2) и (2, 2).

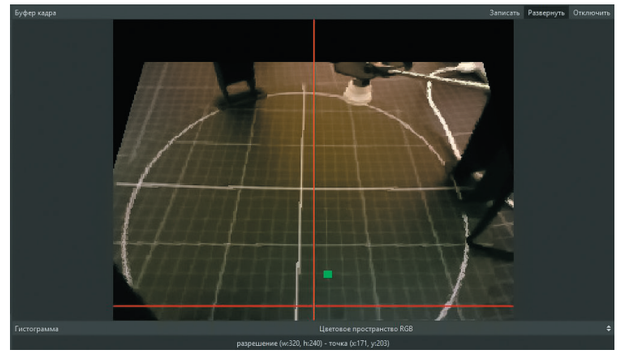


Рисунок 10. Пример выбранной точки на изображении картинки

в) Перенесите созданный вами файл формата *.example* в одну папку с файлом *fit_homography.py.* Откройте терминал (для *Windows* рекомендуется использовать *PowerShell,* для *Linux* и *MacOS – Terminal*) и перейдите в папку с этими файлами. Пропишите команду

python3 fit_homography.py ***ваше_название_файла****.example -unit 10,*

чтобы запустить алгоритм нахождения матрицы гомографии по выписанным вами точкам.

Примечание: *-unit 10* означает, что одна клетка на поверхности основания платформы из множества, которые мы использовали в файле формата .example для координат вида (1, 2) – это квадрат 10x10 мм.

г) В результате выполнения алгоритма в терминал будет выведена матрица гомографии 3x3 (рис. 11). Данная матрица будет использоваться для преобразования координат кубика из системы координат камеры в мировую, поэтому желательно проверить точность созданной матрицы при помощи приведенного после текста *Verify* обратного преобразования, где по координатам пикселей вычисляются координаты в клетках на плоскости робота. Проверьте, чтобы выведенные значения отражали те, что у вас находятся в файлике. Замените матрицу гомографии в переменной *H* в файле *get_world_coords.py* на вашу.

```
[[-2.61318468e-03 -1.79753062e+00  4.10167159e+02]
 [ 1.95262704e+00 -2.07676364e-02 -3.10633873e+02]
 [-1.13144656e-03  7.52301868e-03  1.00000000e+00]]
--- Verify ---
[183.0, 180.0] -> [np.float64(4.011638528020936), np.float64(2.00078840567155546)]
[142.0, 135.0] -> [np.float64(9.0099552195054411), np.float64(-1.94962764927755526)]
[149.0, 214.0] -> [np.float64(1.0283794061831741), np.float64(-0.988666622984882)]
[197.0, 203.0] -> [np.float64(1.9421983778811142), np.float64(3.02992212510174)]
[55.0, 124.0] -> [np.float64(10.003590340206792), np.float64(-11.002451019066612)]
[160.0, 170.0] -> [np.float64(4.9654298099095975), np.float64(-0.083133612923082213)]
[228.0, 191.0] -> [np.float64(3.0401666799492999), np.float64(5.9937064484992666)]
```

Рисунок 11. Результат работы файла *fit_homography.py*

д) Удостоверьтесь, что камера не сдвинулась относительно робота, и красное перекрестие совпадает с ранее выбранным положением. Закомментируйте строки, отвечающие за отрисовку перекрестия:

```
y = (img.height() * 19) // 20
img.draw_line(0, y, img.width(), y, color=(255, 0, 0))
x = img.width() // 2
img.draw_line(x, 0, x, img.height(), color=(255, 0, 0))
```

е) Запустите алгоритм и проверьте, находится ли кубик на изображении камеры и правильно ли выводится его центр в последовательном терминале. Приведенные в терминале координаты уже преобразованы в миллиметры (рис. 12).

Примечание: Учитывайте, что если ваша камера смещена относительно центра основания робота, то у вас появится сдвиг в координатах, который можно исправить, изменив коэффициенты в массиве *offset.*

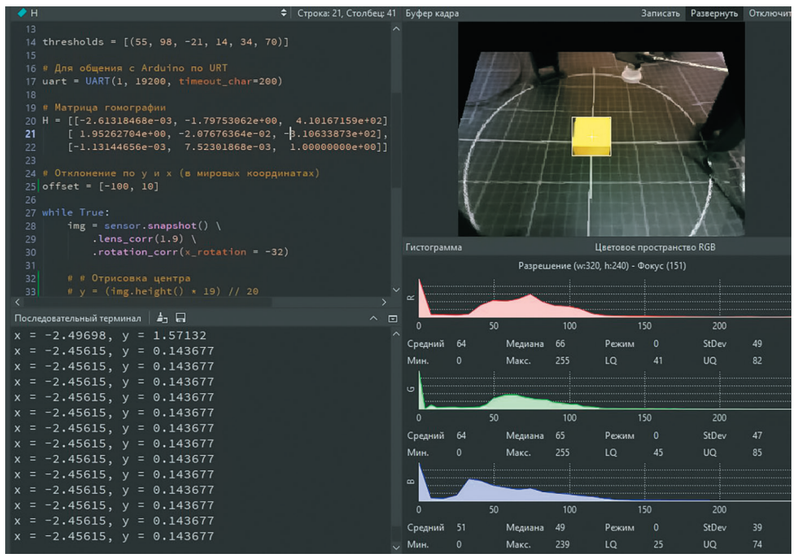


Рисунок 12. Результат работы файла *get_world_coords.py*

ж) Для того, чтобы данный код работал и без подключения к ноутбуку, загрузите его на *SD*-карту камеры, перейдя на верхней панели во владку «Инструменты» и нажав на поле «Сохранить открытый скрипт в *OpenMV Cam* (как *main.py*)». Во всплывающем окне «Сохранить сценарий» нажмите «Да» на вопрос о сбросе комментариев.

2. Программирование робота

«Мозгами» дельта-робота, которые будут получать данные с камеры о шарике (его положении, цвете и т. п.) для выполнения различных вычислений и управления самим роботом, является *IoT* контроллер.

Для загрузки на него управляющих команд и получения обратных данных установите приложение *Arduino IDE* (рис. 13) – с официального сайта *Arduino* (если еще не устанавливали). Что такое *Arduino IDE*? Это среда разработки, собравшая в себе множество удобных инструментов для программирования и работы с контроллерами, что позволяет быстро и качественно писать управляющие команды для, как пример, роботов.

https://www.arduino.cc/en/software/

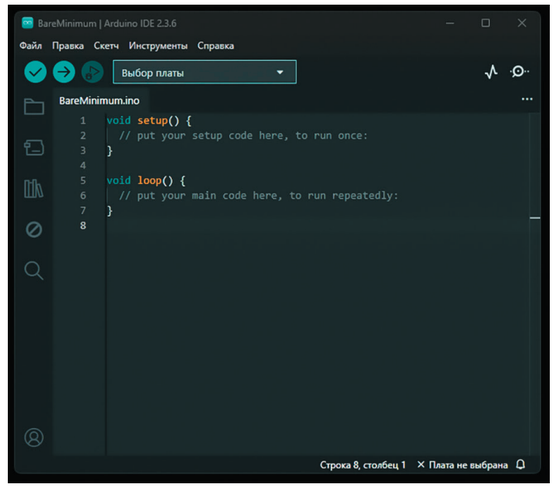


Рисунок 13. Интерфейс *Arduino IDE*

После установки в *Arduino IDE* необходимо подключить в него нужные библиотеки. Они нужны для того, чтобы упростить разработку программного кода и заново не изобретать код, уже написанный кем-то ранее. По своей сути, библиотеки – это код, выполняющий какую-то определенную задачу и при этом «спрятанный» от глаз пользователя. Для реализации движения робот должен знать, как ему прийти в точку «А», а для этого ему необходимо провести сложные математические вычисления для решения задачи обратной кинематики (как мне согнуть свои суставы, чтобы мой рабочий инструмент – в контексте дельта-робота это присоска – пришел в нужное место?). Для того, чтобы сделать это, у контроллера есть огромный готовый код, который он раз за разом выполняет для каждой точки, в которую он должен прийти. Программно – это огромное количество строчек кода, которые мало того, что сделают программу менее читаемой, так еще и не всем интересны, ведь не каждый желает

знать, как именно робот решает определенную задачу. Подобный код удобно складывать «под капот», ассоциируя его с какой-то командой, упомянув которую, он будет выполнен. За одним словом спрятана сложная математика и работа с большими объемами данными, которые, во-первых, уже кем-то реализованы и которые можно использовать, не вникая в логику их кода.

Загрузить библиотеку для работы с дельта-роботом в приложение *Arduino IDE* можно через верхнюю панель, как на рисунке (рис. 14), по пути «Скетч» → «Подключить библиотеку» → «Добавить *.ZIP* библиотеку». Выберите файл *Delta-Kinematics-Library-master.zip*, который шел на электронном носителе вместе с роботом. При успешной загрузке в выпадающем списке в разделе «Подключить библиотеку» в «Сторонних библиотеках» у вас появится *Delta-Kinematics-Library-master.*

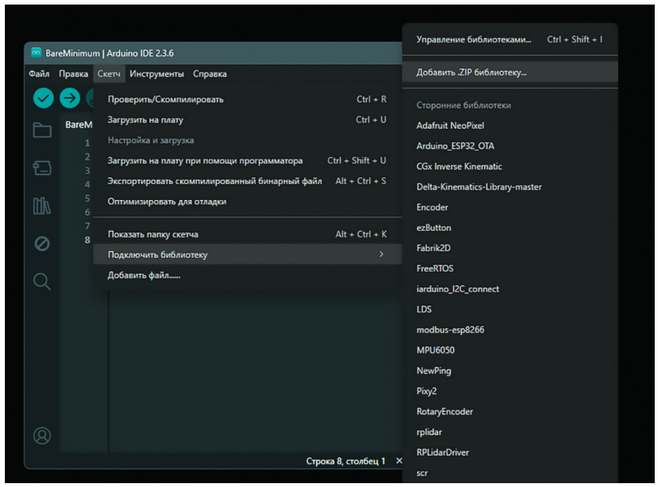


Рисунок 14. Добавление библиотеки в *Arduino IDE*

Примечание: Альтернативн, вы можете установить библиотеку, распаковав файл *Delta-Kinematics-Library-master.zip* в папку со всеми библиотеками вашего *Arduino IDE*, которая чаще всего находится по пути *C:\Пользователи\{ваше имя пользователя}\Документы\Arduino\libraries.*

Программный код для поиска шарика роботом при помощи камеры *OpenMV H7 Plus* уже готов, вы можете его взять с предоставленного электронного носителя. Для этого найдите папку *FindCubeOpenMV* и откройте файл *FindCubeOpenMV.ino.*

Перед вами откроется код программы для контроллера. В нем уже имеются поясняющие комментарии, описывающие основную суть написанного алгоритма, с помощью которых можно разобраться в том, как именно получает и передает данные, производит вычисления и в целом работает дельта-робот. В самом верху скетча (то, как называется управляющая программа, разработанная в *Arduino IDE*) можно заметить, что кроме библиотеки робота используются еще одна – *SCServo.* Что это за библиотека?

Библиотека *SCServo* – это библиотека для управления сервоприводами *Feetech*, что используются для приведения в движения платформы Стюарта. Данную библиотеку можно скачать с официального сайта *Feetech.*

https://gitee.com/ftservo/SCServoSDK/blob/12bc1f74eeb2bfd37383f513d5ff53d89874f238/SCServo_Arduino_220524.7z

Для загрузки скетча на контроллер необходимо нажать на «Выбор платы» (рис. 15) в верхней части приложения и во вкладке «Платы» выбрать *Arduino Mega or Mega 2560* и выбрать порт, по которому будет реализовано общение с контроллером. Учитывайте, что порт появится в списке доступных только после подключения контроллера к компьютеру по кабелю *USB.* Нажмите «Ок».

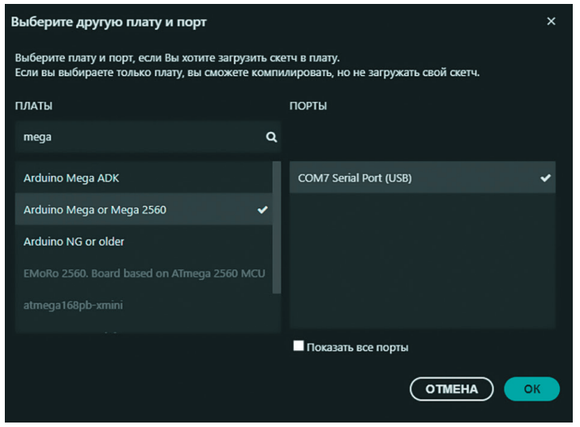


Рисунок 15. Выбор платы в *Arduino IDE*

После этого откроется возможность проверить код (галочка в верхней части приложения *Arduino IDE*) и загрузить скетч на контроллер (рис. 16).

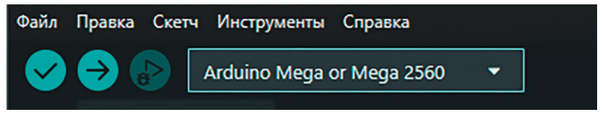


Рисунок 16. Кнопки «Проверить» и «Загрузить скетч»

После загрузки скетча на контроллер подайте питание на робота, необходимое для его корректной работы, подключите камеру к вашему компьютеру по *USB* и при помощи *OpenMV IDE* запустите программу для камеры, затем поставьте на основание робота кубик, и дельта-робот начнет работу по его цикличному поиску. Учитывайте, что в зависимости от расположения камеры относительно дельта-робота может понадобиться поворот системы координат *OXY*, в которой находятся передаваемые данные с камеры, относительно системы координат робота.

```
#include <DeltaKinematics.h>
#include <SCServo.h>

// Объект sms_sts для сервоприводов
SMS_STS servo;

// Объект DeltaKinematics для расчета кинематики робота
DeltaKinematics DK(100, 232, 74.9, 22.4);
```

```
int r = 94;              // Радиус окружности, ограничивающей передвижение
рабочего органа
double eps = 0.01;       // Погрешность для работы с числами типа float
float cube_height = 40;  // Высота кубика
int zBase = -240;

unsigned long startMillis;               // Количество мс, отсчитывающих
"начало" процесса
unsigned long currentMillis;             // Количество мс на текущий момент
времени
const unsigned long servoPeriod = 1500;  // Период (в мс) / ограничение по
времени для процесса

byte ID[3];       // Массив ID сервоприводов
s16 Position[3];  // Массив положений сервоприводов
u16 Speed[3];     // Массив скоростей сервоприводов
byte ACC[3];      // Массив ускорений сервоприводов

void setup() {
  Serial.begin(115200);    // Для ввода координат с "Монитора порта"
  Serial1.begin(1000000);  // Для сообщения с сервоприводами
  servo.pSerial = &Serial1;

  ID[0] = 1; ID[1] = 2; ID[2] = 3;  // Задание ID сервоприводов
  Speed[0] = 1400; Speed[1] = 1400; Speed[2] = 1400;  // Задание скорости
сервоприводов (шаг/с)
  ACC[0] = 10; ACC[1] = 10; ACC[2] = 10;  // Задание ускорений сервоприводов
(шаг/с^2)
}

void loop() {
  GetCoordsCamera();
  delay(1000);
}

bool GetCoordsCamera() {
  Serial3.begin(19200);  // Для получения данных с камеры
  delay(210);
  // Прием координат с камеры по URT
  if (Serial3.available()) {
    Serial3.readStringUntil('\n');
    String dataCamera = Serial3.readStringUntil('\n');  // Чтение строки до
символа новой строки
```



```
    dataCamera.trim();
    Serial3.end();

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

    // Получение чисел из строки
    if (dataCamera.length() > 0) {
      byte index = 0;
      while (dataCamera.length() > 0 && index < 3) {
        int spaceIndex = dataCamera.indexOf(' ');  // Нахождение первого с
начала строки пробела
        // Если пробел не был найден
        if (spaceIndex == -1) {
          camParameters[index++] = dataCamera;  // Добавляем оставшуюся
строку в массив
          break;
        } else {
          camParameters[index++] = dataCamera.substring(0, spaceIndex);  //
Добавляем в массив все символы от начала строки до пробела (число)
          dataCamera = dataCamera.substring(spaceIndex + 1);             //
Убираем обработанную часть строки
          dataCamera.trim();
        }
      }
      if (index == 2) {
        float x = camParameters[0].toFloat();
        float y = camParameters[1].toFloat();
        Serial.println("x = " + String(x));
        Serial.println("y = " + String(y));
        MoveRobot(x, y, -180);
        MoveRobot(x, y, -200);

        MoveRobot(x, y, -180);
        MoveRobot(-70, 30, -170);
        return true;
      } else return false;  // if (index == 2)
    } else return false;    // if (dataCamera.length() > 0)
  } else {
    Serial3.end();
    return false;
  }  // if (Serial3.available())
}
```

```
// Вычисление кинематики Дельта-манипулятора и перемещение в точку
void MoveRobot(float x, float y, float z) {
  int error_step = 10;    // Допустимая погрешность положения сервопривода
  int current_pos[3];     // Массив для хранения текущего положения сервоприводов
  bool finished = false;  // Флажок для выхода из цикла "while"

  startMillis = millis();  // Получение количества мс, задающих "начало" движения
  Serial.println("Введено: " + String(x) + ", " + String(y) + ", " + String(z));

  // Проверка соответствия ограничениям окружности
  if ((pow(x, 2) + pow(y, 2)) > (pow(r, 2) + eps)) {
    Serial.println("Вне границ окружности");
    Serial.println();
    return;
  }
  // Проверка корректности вводимой высоты
  if (z < zBase || z > -135) {
    Serial.println("Высота не та");
    Serial.println();
    return;
  }

  DK.inverse(x, y, z);  // Вычисление обратной кинематики дельта-манипулятора
  Serial.println("Углы " + String(DK.a) + ", " + String(DK.b) + ", " + String(DK.c));

  // Проверка на возможность вычисления обратной кинематики для заданной точки
  if ((DK.a == 0) && (DK.b == 0) && (DK.c == 0) || (DK.z > 0)) {
    Serial.println("Точка недостижима");
    return;
  }

  // Вычисление положения сервоприводов (2048 - центральное положение, а DK.a, DK.b, DK.c - углы поворота)

  Position[0] = constrain(2048 + (DK.a * 11.38), 1400, 2500);
  Position[1] = constrain(2048 + (DK.b * 11.38), 1400, 2500);
  Position[2] = constrain(2048 + (DK.c * 11.38), 1400, 2500);
```



```
  // Перемещение сервопривода
  Serial.println("Position: " + String(Position[0]) + ", " + String(Position[1]) + ", " + String(Position[2]));
  servo.SyncWritePosEx(ID, 3, Position, Speed, ACC);

  // Пока не истечет период "finished" (или пока сервопривод не придет в заданную точку)
  while (!finished) {
    currentMillis = millis();  // Получение количества мс на текущий момент времени

    // Проверка истечения периода "servoPeriod"
    if (currentMillis - startMillis < servoPeriod) {
      for (int i = 0; i < 3; i++) {
        if (servo.FeedBack(ID[i]) != -1) {
          // Получение текущих положений сервоприводов
          current_pos[i] = servo.ReadPos(ID[i]);
        }
      }
      // Если все сервоприводы находятся в нужных положениях, то ожидание истечения периода "servoPeriod" прерывается
      if (abs(current_pos[0] - Position[0]) < error_step && abs(current_pos[1] - Position[1]) < error_step && abs(current_pos[2] - Position[2]) < error_step) {
        finished = true;
      }
    } else {
      // Истекло время ожидания перемещения сервоприводов
      finished = true;
    }
  }
  Serial.println();
}
```

РОБОТ 4

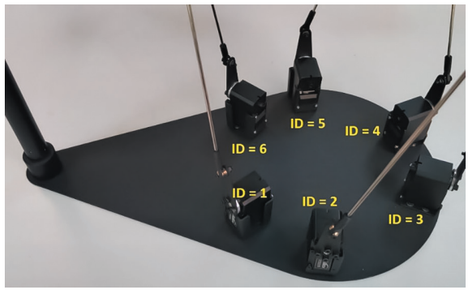
Работа с платформой Стюарта с использованием модуля технического зрения *OpenMV H7 Plus*

Это полностью параллельный манипулятор. Он состоит из неподвижного основания и подвижной платформы, соединенных шестью раздвижными телескопическими стойками (актуаторами). Стойки соединены с основанием и платформой через сферические или карданные шарниры. Такая конструкция обеспечивает все 6 *DOF.*

Кинематическая схема: Схема описывается как *6-UPS (Universal-Prismatic-Spherical)* или 6-SPS *(Spherical-Prismatic-Spherical)*. Каждая нога имеет одну поступательную пару (актуатор) и два шарнира, обеспечивающих по две или три степени свободы на концах.

Соберите платформу Стюарта в соответствии с Инструкцией по сборке. Закрепите штатив с камерой на ее базу. Для изучения возможностей платформы Стюарта будем использовать ее совместно с модулем технического зрения.

В первую очередь необходимо подготовить сервоприводы к работе. Установите робот в начальное положение, как показано на рисунке. Это среднее положение валов сервоприводов *Pos = 2048*, или 0.



Теперь пропишите ID сервоприводов и значение среднего положения вала каждого сервопривода. Процесс подготовки сервоприводов был рассмотрен в предыдущих работах.

Для подключения модуля технического зрения на контроллер необходимо установить дополнительную плату расширения – интерфейс *UART, I²C*. Соединение с модулем технического зрения производится с помощью входящего в комплект 4-пинового кабеля.

Подключите робот к контроллеру.

В данном примере работа с техническим зрением для платформы Стюарта сводится к задаче балансирования шарика на поверхности ее верхней пластины (платформы).

1. Настройка камеры

OpenMV H7 Plus – это модуль технического зрения, наделенный широким спектром функций, как, например, поиск объекта по его цвету. Он позволяет не только использовать уже готовые примеры кода, но также предлагает пользователям возможность писать свои собственные алгоритмы на языке *MicroPython* и сохранять его в памяти камеры при помощи специализированного *IDE (OpenMV IDE).*

Задача балансирования шарика предполагает, что находящаяся над платформой Стюарта камера будет постоянно искать шарик, выделяя его среди остальных объектов на изображении с помощью информации о его цвете. И настройка камеры предполагает именно обучение *OpenMV* на нужный оттенок отслеживаемого объекта при помощи ее встроенных возможностей.

Калибровка камеры на выбранный цвет происходит с использованием возможностей официального *IDE* для камер фирмы *OpenMV – OpenMV IDE* (рис. 1) – которое вы можете скачать напрямую с официального сайта.

https://openmv.io/pages/download

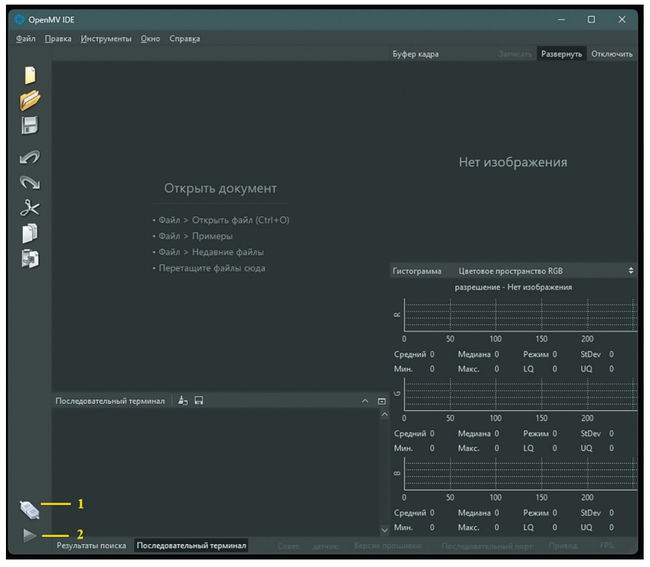


Рисунок 1. Общий интерфейс приложения *OpenMV IDE*

После успешной установки приложения подключите камеру к компьютеру при помощи *USB*-кабеля для передачи изображения с *OpenMV H7 Plus* на ваш экран. Данное подключение позволяет отслеживать корректность обучения камеры и вносить правки в ее работу в режиме реального времени. При успешном подключении значок соединения в левом нижнем углу (рис. 1) сменится на другой (рис. 2), и при нажатии на него вы подключитесь к модулю.

Примечание: Рекомендуется использовать камеру с подключенной *SD*-картой для ее корректной работы. Процесс установки *SD*-карты был рассмотрен в разделе описания работы с дельта-роботом.

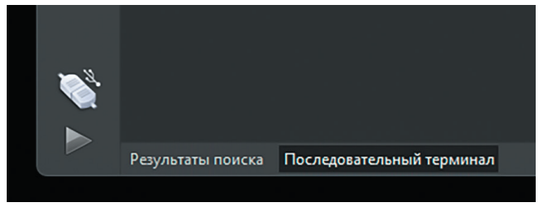


Рисунок 2. Значок подключения к камере

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

При запуске *OpenMV IDE* автоматически открывается заготовленный скрипт-пример *helloworld.py.* Запустите его, нажав зеленую стрелочку в левом нижнем углу (рис. 1, 2). Если нужный скрипт по каким-либо причинам не открылся, то найдите на верхней панели «Файл» → «Примеры» → *HelloWorld* → *helloworld.py.*

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

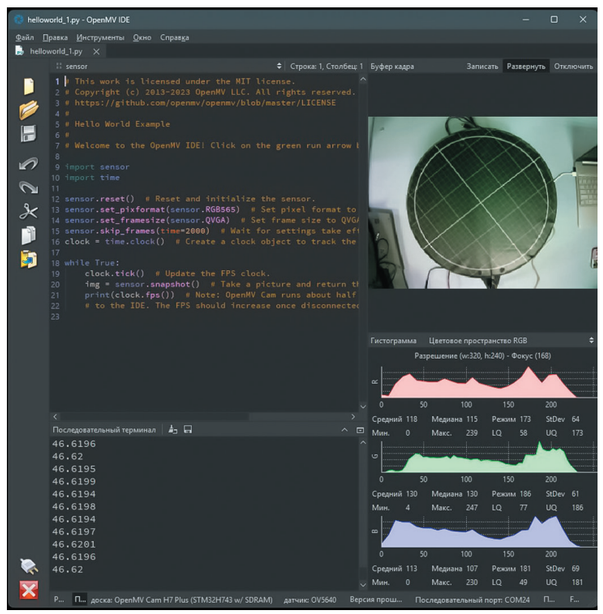


Рисунок 3. Интерфейс приложения *OpenMV IDE* при корректно подключенной камере

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

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

1.4. Калибровка параметров обработки изображения

Откройте файл *calibrate_white_balance.py* с предоставленного электронного носителя. Данный код позволяет включать и выключать автоматическую коррекцию параметров обработки изображений с камеры: баланса белого, экспозиции, усиления. В рассмотренном примере *HelloWorld* эти параметры были включены по умолчанию, и в зависимости от освещения и яркости окружения картинка на экране автоматически менялась. Данная функция полезна, но не в контексте отслеживания объекта по цвету, так как при смене данных параметров меняются оттенки цветов на изображении, и поэтому искомый шарик может не опознаваться корректно.

Предполагается, что платформа Стюарта работает при детерминированных условиях освещенности, и поэтому автонастройка параметров камеры в приведенных в данной работе программных алгоритмах отключена, т. к. она может ухудшить распознавание отслеживаемого роботом шарика. Представленные в коде значения параметров баланса белого, экспозиции и усиления можно изменять под конкретные условия вашего окружения при помощи файла *calibrate_white_balance.py.*

Откройте файл *calibrate_white_balance.py* и запустите его. При запуске данного кода в течение пяти секунд (строка 6) автоматическая калибровка параметров обработки изображения будет включена, и вы сможете, вращая камеру, найти оптимальные условия для вашей камеры. Как только вы найдете состояние, в котором вас будет устраивать отображаемая в *IDE* картинка, задержите камеру, пока не увидите в последовательном терминале (рис. 4) сообщение «Отключение автоматической корректировки параметров камеры». После этого параметры усиления, баланса белого и экспозиции будут выводиться в терминал раз в секунду. Выпишите себе строку с данными параметрами, чтобы использовать в последующем коде.

1.5. Обучение на цвет

Калибровка камеры на заданный цвет шарика происходит при помощи встроенных возможностей модуля технического зрения *OpenMV H7 Plus*. Выберите шарик яркого цвета, контрастирующий с поверхностью основания робота и окружением (стены, шторы и т. д.).

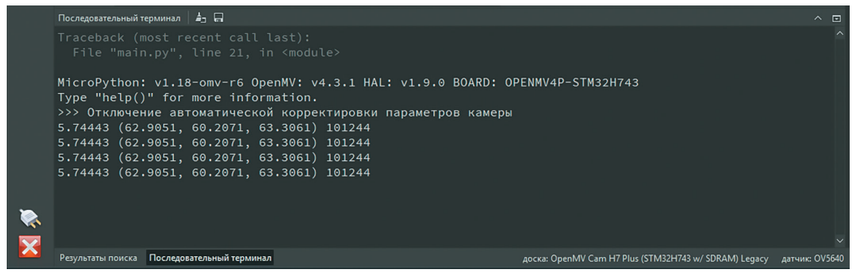


Рисунок 4. Результат работы кода *calibrate_white_balance.py*

Не закрывая файл *calibrate_white_balance.py* после настройки параметров изображения, наведитесь камерой на нужный вам шар и нажмите кнопку «Стоп» (появляется на месте зеленой кнопки при запуске камеры, рис. 1. 2).

Последнее изображение, полученной *OpenMV H7 Plus,* «застынет» у вас на экране, что будет удобно для калибровки запоминаемого цвета. Найдите на верхней панели «Инструменты» и перейдите «Машинное зрение» → *Threshold Editor.* Во всплывающем окне «Расположение исходного изображения?» выберите «Кадровый буфер», то есть последнее захваченное изображение камерой, которое вы видите в окне «Буфер кадра» в правой части *IDE.*

В открывшемся окне (рис. 5) найдите справа кнопку «Сбросить ползунки» и нажмите ее.

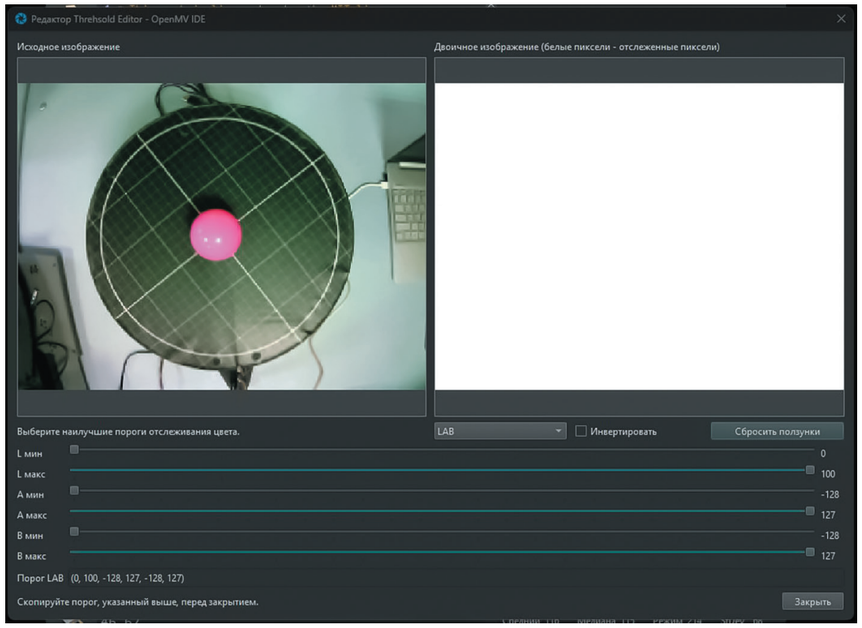


Рисунок 5. Интерфейс *Threshold Editor*

Представленные в нижней части окна ползунки позволяют регулировать пороги отслеживания цвета, чтобы выделять именно тот объект, который нужно отслеживать. Отрегулируйте ползунки таким образом, чтобы только шарик был выделен белым на изображении справа (двоичном) (рис. 6).

Примечание: Наилучший вариант настройки порогов предполагает, что шарик на правом изображении виден полностью и лишние предметы не были выделены вместе с ним. Чем точнее будут настроены параметры на данном этапе, тем лучше будет определяться цвет при дальнейшей работе. Как только пороговые значения для отслеживания цвета шарика были настроены, выпишите из графы «Порог *LAB*» в нижней части значения в круглых скобках и/или скопируйте строчку полностью.

1.6. Использование коэффициентов в коде программы

Полученные значения из предыдущих шагов необходимо подставить в файл *fine_ball.py.* Значения порогов для определения цвета шарика из пункта 1.2 необходимо вставить в переменную *thresholds.*

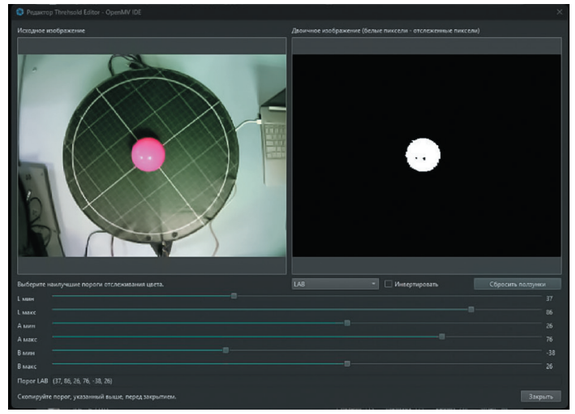


Рисунок 6. Отображение шарика при корректно настроенных порогах отслеживания цвета

Для значений с рисунка 6 переменная будет следующей:

thresholds = [(37, 86, 26, 76, -38, 26)]

А значения параметров изображения из пункта 1.1 необходимо вставить следующим образом (для данных с рис. 4 – *«5.74443 (62.9051, 60.2071, 63.3061) 101244»):*

sensor.set_auto_gain(False, gain_db=5.74443)
sensor.set_auto_whitebal(False, rgb_gain_db=(62.9051, 60.2071, 63.3061))
sensor.set_auto_exposure(False, exposure_us=101244)

Примечание: Если получившееся изображение выглядит слишком темным или светлым, повторите пункт 1.1 и/или проверьте, правильно ли вы подставили полученные значения.

Примечание: Проверьте, что при одном и том же освещении шарик корректно определяется с разных сторон и под разными углами. Если поиск цвета захватывает другие объекты или находит шар не полностью, повторите настройку порогов из пункта 1.2 и/или замените шарик на более контрастный.

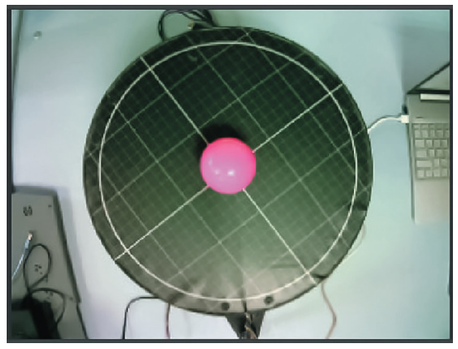


Рисунок 7. Корректное положение камеры относительно платформы

Убедитесь, что камера находится относительно верхней платформы Стюарта на достаточном расстоянии, чтобы вся верхняя пластина была видна на изображении и была отцентрирована и выровнена как на рисунке 7.

Запустите алгоритм *fine_ball.py* и проверьте, правильно ли выводится центр шарика на изображении в последовательном терминале. Приведенные в терминале координаты уже преобразованы в мм (рис. 8). Также удостоверьтесь, что окружность верхней платформы (белая линия или края диска верхней платформы) также опознается камерой, и стабильно, четко отображается на вашем экране.

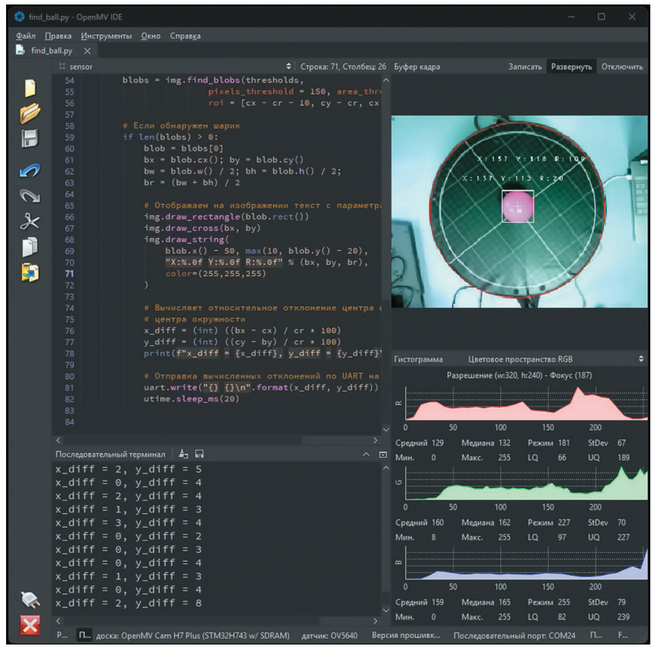


Рисунок 8. Результат работы файла *find_ball.py*

Для того, чтобы данный код работал и без подключения к ноутбуку, загрузите его на *SD*-карту камеры, перейдя на верхней панели во вкладку «Инструменты» и нажав на поле «Сохранить открытый скрипт в *OpenMV Cam* (как main.py)». Во всплывающем окне «Сохранить сценарий» нажмите «Да» на вопрос о сбросе комментариев.

3. Программирование робота

«Мозгами» робота, которые будут получать данные с камеры о шарике (его положении, цвете и т. п.) для выполнения различных вычислений и управления самим роботом, является *IoT* контроллер на базе миикроконтроллера *Arduino Mega 2560.*

Для работы с ним откройте *Arduino IDE.* Для начала необходимо установить нужные библиотеки.

Загрузить библиотеку для работы с платформой Стюарта в приложение *Arduino IDE* можно через верхнюю панель, как на рисунке 9, по пути «Скетч» → «Подключить библиотеку» → «Добавить .*ZIP* библиотеку». Выберите файл *StewartRobotZar.zip,* который шел на электронном носителе вместе с роботом. При успешной загрузке в выпадающем списке в разделе «Подключить библиотеку» в «Сторонних библиотеках» у вас появится *StewartRobotZar.*

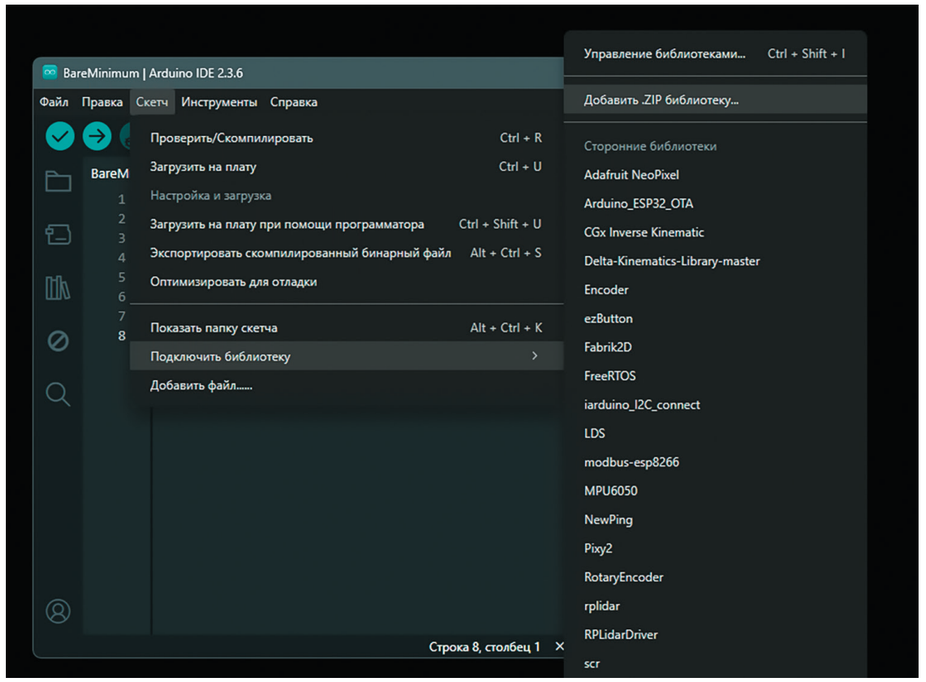


Рисунок 9. Добавление библиотеки в *Arduino IDE*

Примечание: Альтернативно, вы можете установить библиотеку, распаковав файл *StewartRobotZar.zip* в папку со всеми библиотеками вашего *Arduino IDE,* которая чаще всего находится по пути *C:\Пользователи**{ваше имя пользователя}***\Документы\Arduino\libraries (для Windows).*

Программный код для поиска шарика роботом при помощи камеры *OpenMV H7 Plus* уже готов, вы можете его взять с предоставленного электронного носителя. Для этого найдите папку *BallBalance* и откройте файл *BallBalance.ino.*

Перед вами откроется код программы для контроллера. В нем уже имеются поясняющие комментарии, описывающие основную суть написанного алгоритма, с помощью которых можно разобраться в том, как именно получает и передает данные, производит вычисления и в целом работает робот. В самом верху скетча (то, как называется управляющая программа, разработанная в *Arduino IDE*) можно заметить, что кроме библиотеки робота используются еще одна – *SCServo*. Что это за библиотека?

Библиотека *SCServo* – это библиотека для управления сервоприводами *Feetech*, что используются для приведения в движения платформы Стюарта. Данную библиотеку можно скачать с официального сайта *Feetech.*

https://gitee.com/ftservo/SCServoSDK/blob/12bc1f74eeb2bfd37383f513d5ff53d89874f238/SCServo_Arduino_220524.7z

Для загрузки скетча на контроллер необходимо нажать на «Выбор платы» (рис. 11) в верхней части приложения и во вкладке «Платы» выбрать *Arduino Mega or Mega 2560* и выбрать порт, по которому будет реализовано общение с контроллером. Учитывайте, что порт появится в списке доступных только после подключения контроллера к компьютеру по кабелю *USB.* Нажмите «Ок».

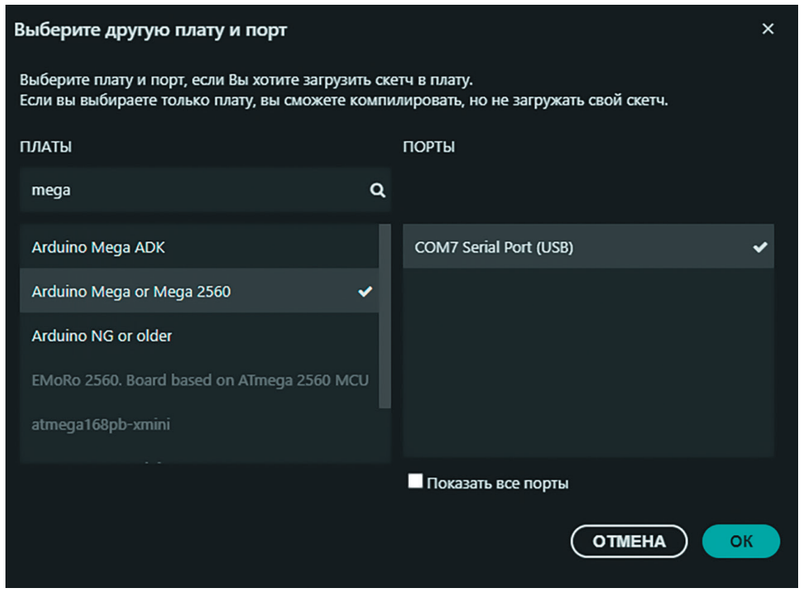


Рисунок 10. Выбор платы в *Arduino IDE*

После этого откроется возможность проверить код (галочка в верхней части приложения *Arduino IDE*) и загрузить скетч на контроллер (рис. 12).

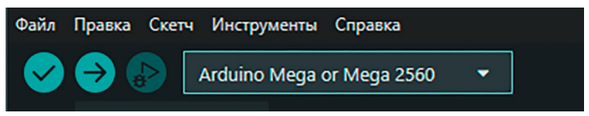


Рисунок 12. Кнопки «Проверить» и «Загрузить скетч»

После загрузки скетча на контроллер подайте питание на робота, необходимое для его корректной работы, поставьте на основание робота шарик, и робот начнет работу по его балансированию.

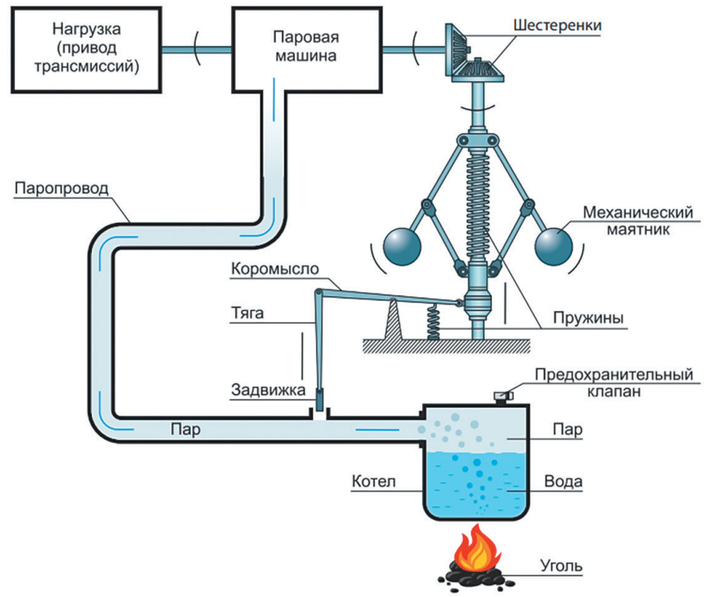
Получение информации об окружающем пространстве. Обратная связь

Очевидно, что для эффективного использования роботов в повседневной жизни совершенно недостаточно, чтобы роботы воспринимали окружающую действительность лишь как пассивные наблюдатели. Не менее важно, чтобы «органы чувств» обеспечивали «обратную связь»; иными словами, с помощью этих органов роботы должны обрести возможность «осознавать», каким образом их действия

отражаются на состоянии внешнего мира. В данном случае под «миром», естественно, понимается лишь та ничтожно малая его часть, с которой взаимодействует робот.

Поскольку концепция обратной связи играет ключевую роль для любой автономной или самоуправляемой системы, например, робота, необходимо четко понимать ее суть. Основная идея здесь заключается в том, что функция управляющего органа (мозга животного или компьютера робота) должна состоять не только в формировании и передаче инструкций (в виде нервных импульсов, поступающих на мышцы, или электрических сигналов, идущих к электродвигателю), но и в восприятии и интерпретации тех данных, по которым можно судить о выполнении этих инструкций и о характере их воздействия на объект управления.

Таким образом, обратная связь в робототехнике – это отрицательная обратная связь, соединяющая датчики робота с виртуальными (алгоритм софта) или реальными датчиками источника сигнала управления роботом через каналы связи, приводы, пропорциональные датчики. Устройства обратной связи представляют собой класс устройств, необходимых для работы в замкнутом контуре. Они передают сигнал обратно приводу или контроллеру движения для мониторинга операции или процесса и проверки правильности работы. На рисунке приведен пример использования одного из первых механизмов обратной связи для управления подачей пара и скорости вращения. Этот механизм называется регулятор Уатта.



Роботы и датчики

Датчиков и индикаторов, используемых в робототехнике, существует огромное количество. Мы рассмотрим некоторые. Это датчики для обнаружения препятствий – датчик касания, для определения ориентации робота в пространстве – акселерометр и гироскоп, датчик определения цвета, *LED*-индикатор и *RGB*-светодиод.

Лабораторная работа

№ 5 УПРАВЛЕНИЕ LED-СВЕТОДИОДОМ

Цель работы: *Познакомиться с принципами управления устройствами вывода на примере светодиода.*

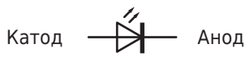
Задачи: *Собрать макет из прилагаемых в наборе деталей по приведенной схеме.*
Выбрать из примеров библиотеки и запустить скетч для управления светодиодом.

Краткие теоретические сведения

Светодиод, или светоизлучающий диод (СД, СИД; англ. *light-emitting diode, LED*) – полупроводниковый прибор с электронно-дырочным переходом, создающий оптическое излучение при пропускании через него электрического тока в прямом направлении.

Другими словами, это маленькая лампочка, которая светится, когда через нее проходит электрический ток. В отличие от обычных лампочек, светодиоды не нагреваются, потребляют мало энергии и служат очень долго.

На электрических схемах они обозначаются как:

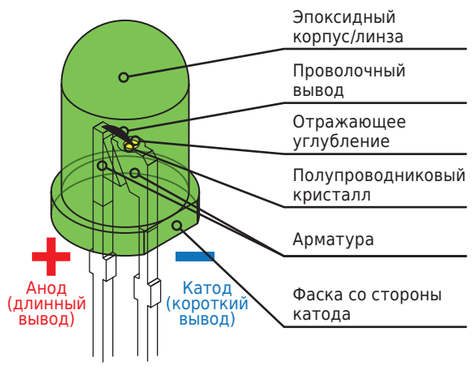


У них есть две ножки:

- «+» (анод) – подключается к источнику питания.
- «–» (катод) – подключается к «минусу» (например, к «земле» на плате).

Важно! Если подключить светодиод неправильно, он не загорится.

Из-за высокой яркости одного светодиода его можно сделать очень маленьких размеров.

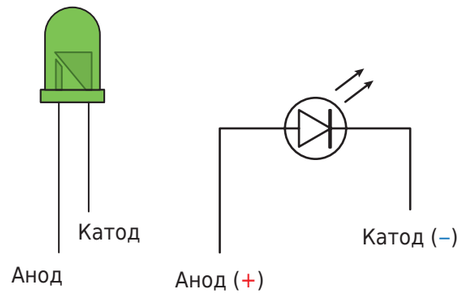


Так, например, на экранах современных телевизоров и мониторов могут одновременно работать несколько десятков миллионов светодиодов. С точки зрения обывателя всё же есть некоторая разница в использовании традиционных ламп накаливания и *LED*. Светодиод должен быть подключен одним концом к катоду (минус), а другим к аноду (плюс), и никак не наоборот. Чтобы зажечь *LED,* необходимо всего лишь подать ток.

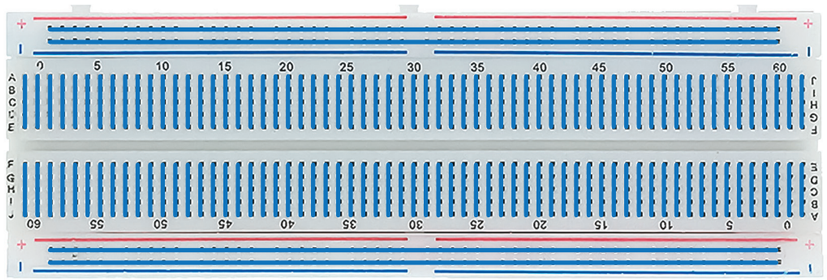
На платах *Arduino* есть встроенный светодиод, и чаще всего он подключен к контакту (13). Второй контакт при этом должен быть соединён с «землей» *(GND).*

Порядок выполнения работы

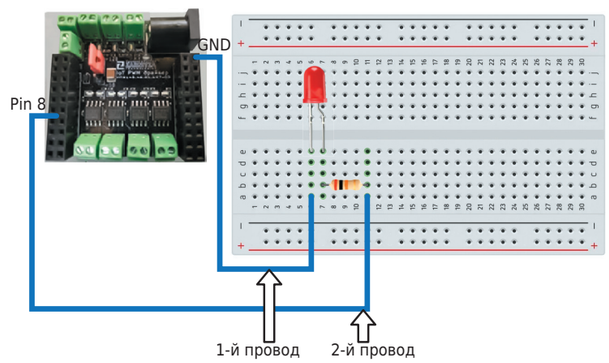
1. Возьмите макетную плату, один из светодиодов, резистор и 2 провода *DuPont* из прилагаемого набора.



2. Соберите схему, как показано на рисунке. Контакты макетной платы внутри соединяются так:



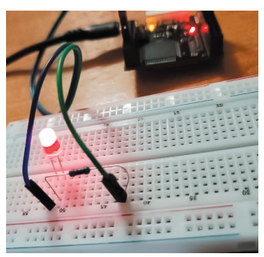
Где и какие пины на плате шилда контроллера, можно посмотреть в Инструкции по эксплуатации.



3. Подключите контроллер к компьютеру при помощи *USB*-кабеля.
4. Запустите *Arduino IDE*. Убедитесь, что устройство обнаружено.
5. Выставьте модель платы как *Arduino Mega 2560.*
6. Напишите программу по образцу. Нажмите на кнопку загрузки на плату.

```
int led = 8;  //объявление переменной целого типа, содержащей номер
порта к которому мы подключили второй провод

void setup()  //обязательная процедура setup, запускаемая в начале
программы; объявление процедур начинается словом void

{
pinMode(led, OUTPUT); //объявление используемого порта, led - номер
порта, второй аргумент - тип использования порта - на вход (INPUT)
или на выход (OUTPUT)

}

void loop() //обязательная процедура loop, запускаемая циклично после
процедуры setup
{

digitalWrite(led, HIGH); //эта команда используется для включения или
выключения напряжения на цифровом порте; led - номер порта, второй
аргумент - включение (HIGH) или выключение (LOW)

delay(1000); //эта команда используется для ожидания между
действиями, аргумент - время ожидания в миллисекундах

digitalWrite(led, LOW);

delay(1000);
}
```

7. Убедитесь, что светодиод на плате мигает длительностью 1 секунда.



8. Напишите программу для мигания диода с частотой раз в 2 секунды (секунду горит, секунду нет).
9. Измените программу, чтобы светодиод горел пятую часть всего времени при той же частоте.

Лабораторная работа

№ 6 УПРАВЛЕНИЕ RGB-СВЕТОДИОДОМ

Цель работы: *Познакомиться со световой схемой RGB и принципом работы RGB-светодиода. Использовать функции при написании скетча.*

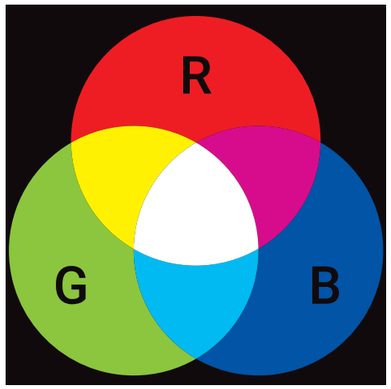
Задачи: *1. Установить в Arduino IDE библиотеку для сервоприводов Feetech STS3235.*
2. Выбрать из примеров библиотеки и запустить скетч для управления сервопривода Write.

Краткие теоретические сведения

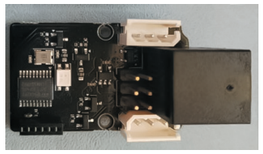
Человеческий глаз способен воспринимать свет в определённом диапазоне от красного до фиолетового. Каждому цвету радуги при этом соответствует определённая длина волны. Привычные нам цвета, не представленные в спектре радуги, являются комбинацией нескольких цветов радуги, зачастую взятых в разных пропорциях. Художники издревле используют палитру для смешения имеющихся красок и получения уникального нужного цвета.

Цифровая природа вычислительных устройств не позволяет в полной мере передать всё разнообразие цветов реального мира. Существующие системы представления цветов позволяют получить разнообразные оттенки, и для большинства людей такой точности достаточно. Самой распространённой цифровой схемой представления цвета на сегодняшний день является *RGB* (*red green blue* – красный, зелёный, синий). Полученный цвет является суммой красного, зелёного и синего цветов с разной интенсивностью. Эта световая схема обязана своей популярностью устройству большинства экранов. Каждая точка (пиксель) современного экрана содержит три маленькие лампочки (например, светодиода) красного, зелёного и синего цветов. Регулируя интенсивность каждого цвета, можно получить всю палитру. Яркость каждого цвета изменяется от 0 до максимального значения в 255.

Управление *RGB*-светодиода требует четырёх контактов. Один из них должен быть подключен к *(GND)*, а остальные три будут управлять соответствующими каналами цвета.

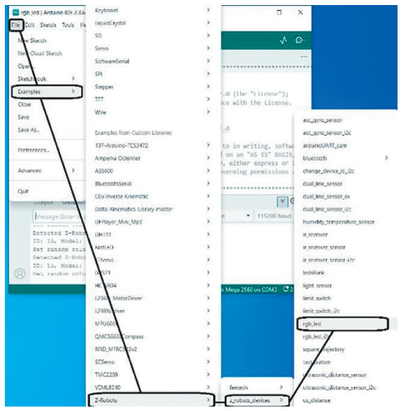


RGB-светодиод и представление цвета



RGB-светодиод – это светодиод, который может светить сразу тремя цветами: красным, зеленым и синим, отчего и называется *RGB – red, green, blue.* В сущности, это три маленьких светодиода, скомпонованные в один. Поэтому и ножек у него 4. Для удобства они отличаются по длине.

Программировать прилагаемый в наборе модуль можно с помощью библиотеки *Z-Robots.* Найти ее можно на прилагаемом *USB*-накопителе. Ниже приведен пример скетча из прилагаемой библиотеки.



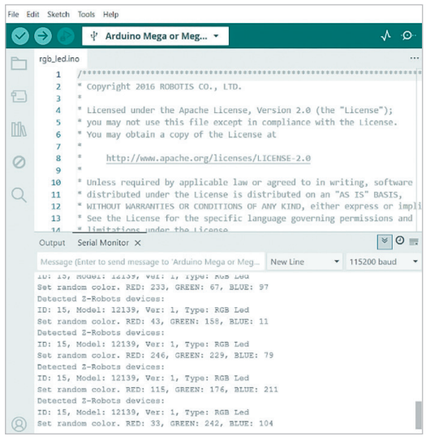
```
/*******************************************************************************
***
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*******************************************************************************
**/

#include <Dynamixel2Arduino.h>

const uint8_t DXL_ID = Z_ROBOTS_RGB_LED_DEFAULT_ID;
```



```
Dynamixel2Arduino dxl(Serial1);

//This namespace is required to use Control table item names
using namespace ControlTableItem;

void setup() {
  // put your setup code here, to run once:

  // Use Serial to debug.
  Serial.begin(115200);

  dxl.begin();
  dxl.setPortProtocolVersion(1.0);

  Serial.println("Demo for the RGB Led I2C");
}

bool line_sensor_left = false;
bool line_sensor_right = false;

void loop() {
  // put your main code here, to run repeatedly:
  FindDevices();
  delay(500);
}

DYNAMIXEL::InfoFromPing_t ping_info[32];
void FindDevices(void) {

  if (uint8_t count_pinged = dxl.ping(DXL_ID, ping_info, sizeof(ping_info) /
sizeof(ping_info[0]))) {
    Serial.print("Detected Z-Robots devices: \n");
    for (int i = 0; i < count_pinged; i++)
    {
      Serial.print("ID: ");
      Serial.print(ping_info[i].id, DEC);
      Serial.print(", Model: ");
      Serial.print(ping_info[i].model_number);
      Serial.print(", Ver: " );
      Serial.print(ping_info[i].firmware_version, DEC);

      if (ping_info[i].model_number == Z_ROBOTS_RGB_LED) {
        Serial.println(", Type: RGB Led");
```

```
        uint8_t red   = random(255);
        uint8_t green = random(255);
        uint8_t blue  = random(255);

        Serial.print("Set random color. RED: ");
        Serial.print(red);
        Serial.print(", GREEN: ");
        Serial.print(green);
        Serial.print(", BLUE: ");
        Serial.println(blue);

        dxl.setColor(ping_info[i].id, red, green, blue);
      } else {
        Serial.println(", Type: Not suitable for this demo");
      }
    }
   } else {
    Serial.print("Ping returned no items\n");
  }
}
```

В данном случае включаются все три светодиода последовательно. Для получения других цветов следует изменить комбинацию включенных диодов.



Лабораторная работа

№ 7 МОНИТОРИНГ ПОКАЗАНИЙ ДАТЧИКОВ СЧИТЫВАНИЕ ДАННЫХ ИНЕРЦИАЛЬНОГО ДАТЧИКА

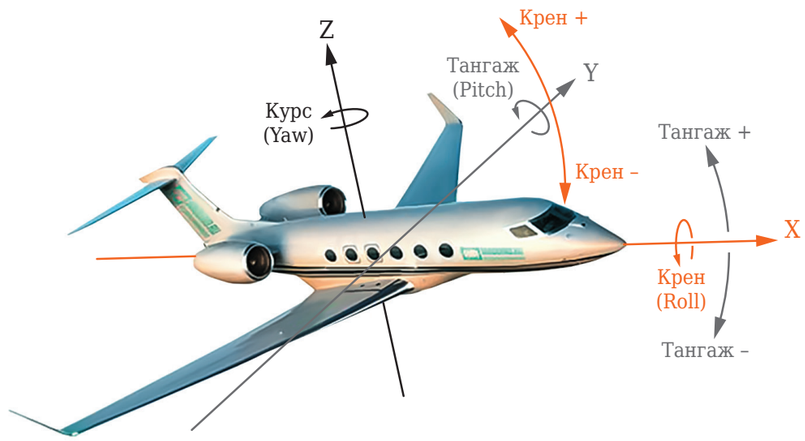
Акселерометр – датчик, измеряющий кажущееся угловое ускорение, которое является геометрической разницей между истинным угловым ускорением и ускорением силы гравитации. Показания датчика можно получать в м/с^2, или в *g* (количестве ускорений свободного падения). Предположим, что датчик неподвижен или движется равномерно, ось *Z* направлена вверх (детали на плате модуля смотрят вверх). В таком случае проекция вектора силы гравитации не оказывает влияния на оси *XY,* их показания равны 0 м/с^2, но оказывает влияние на ось *Z* и направлена в противоположную сторону (к земле), значит

$$Z = 0 - g = 9{,}81\ м/с^2,$$

где 0: проекция истинного ускорения модуля на ось *Z, g:* ускорение свободного падения, взятое со знаком минус, так как его вектор противоположен направлению оси Z. Если модуль наклонить, то влияние *g* на ось *Z* ослабнет, но увеличится на те оси, в направлении которых был наклонён модуль. Таким образом, зная проекцию вектора ускорения свободного падения на оси *X, Y, Z,* можно вычислить положение датчика относительно поверхности земли (углы: «крен» и «тангаж»), но только если датчик неподвижен или движется равномерно.

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

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



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

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

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

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

- «Тангаж» *(Pitch)* – определяет наклон тела вокруг поперечной оси *Y,* например, тангаж самолёта показывает, насколько градусов поднят (или опущен) его нос относительно земной поверхности. Если самолёт находится параллельно земле, то тангаж = 0°. Если самолёт поднял нос вверх (кабрирует, взлетает), то тангаж положительный (от 0° до 90°). Если самолёт опускает нос (пикирует, приземляется), то тангаж отрицательный (от 0°, до -90°). Если самолёт выполняет «мертвую петлю», то тангаж доходит до ±180° (полёт назад вверх ногами).
- «Курс» *(Yaw)* – это самый простой для понимания угол Эйлера (еще его нарывают «рыскание»), он определяет направление вдоль земной поверхности. Например, для самолёта курс определяет куда самолёт летит. Если самолёт летит на север, то курс = 0°. Если самолёт отклоняется от севера влево (на запад, или против часовой стрелки, если смотреть сверху) то курс отрицательный (от 0°, через -90° восток, до -180° юг). Если самолёт отклоняется от севера вправо (на восток, или по часовой стрелке, если смотреть сверху) то курс положительный (от 0°, через 90° запад, до 180° юг).

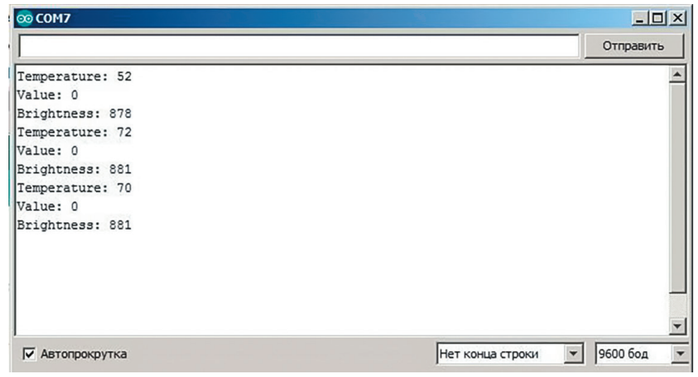
Цель работы: *Научиться считывать показания данных инерциального датчика и отображать их на мониторе.*

Задачи: *1. Установить в Arduino IDE библиотеку для GY-521 MPU6050.ZIP.*
2. Выбрать из примеров библиотеки и запустить скетч для отображения в Мониторе Arduino IDE значений x, y и z.

Краткие теоретические сведения

Зачастую при работе требуется иметь возможность напрямую отслеживать значения разнообразных параметров. Среда *Arduino IDE* позволяет осуществлять такой мониторинг с помощью последовательного соединения с микроконтроллером. Для этого предусмотрен набор функций *Serial.*

Ниже приведен, как пример, результат вывода данных программы, которая выводит показания датчиков температуры, освещения и данные переменного резистора (потенциометра) в диалоговое окно мониторинга порта.



Окно «Монитор порта»

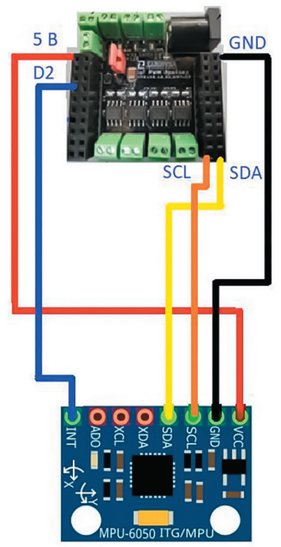
Аргументом функции *Serial.print* является скорость передачи данных.

Обратите внимание, что функция *println* содержит в себе команду на перевод строки. Так же как и *delay()* в данном скетче, это призвано улучшить удобство восприятия информации. Если задержку убрать, а *println* заменить на *print,* получим бесконечно длинную строку однообразного текста.

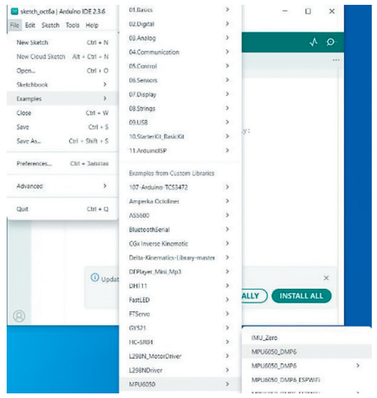
Загрузив скетч на микроконтроллер, откройте меню «Инструменты» и выберите пункт «Монитор порта». Появится диалоговое окно, содержащее информацию, выводимую функцией *print().*

Порядок выполнения работы

1. Подключите модуль *GY-521* к контроллеру *IoT.*



2. Подключите контроллер к компьютеру при помощи *USB*-кабеля.
3. Запустите *Arduino IDE*. Убедитесь, что устройство обнаружено.
4. Выставьте модель платы как *Arduino Mega 2560.*
5. Установите библиотеку *MPU6050*, представленную на *USB*-накопителе.
6. Выберите пример из библиотеки.



7. Напишите программу по образцу.

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation
board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
```



```
 * ========================================================================
*/


/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ========================================================================
*/


// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION
// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
```

```
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT


#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars

Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r',
'\n' };


// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================
```



```
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt
pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}


// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if
having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue
immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or
Arduino
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must
use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
```

```
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
: F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo:
"));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external
interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay
to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;
```



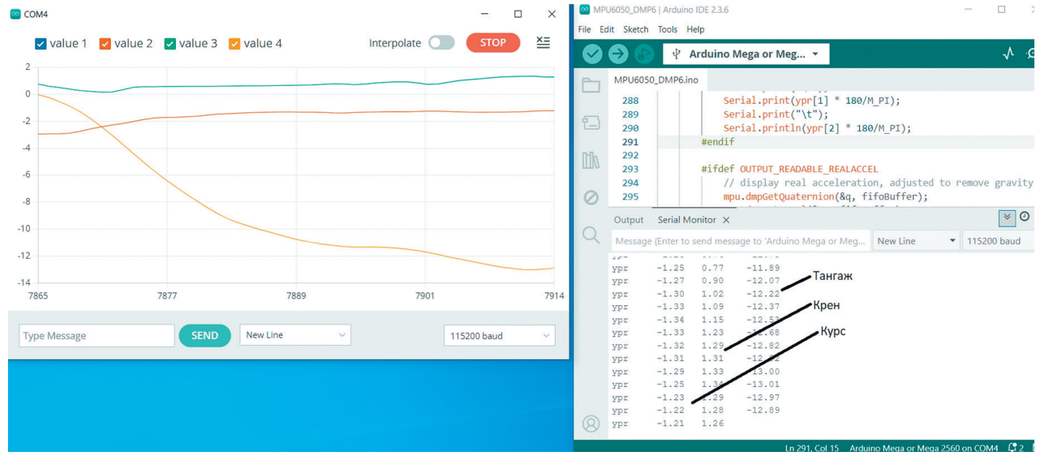
```
        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}


// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        if (mpuInterrupt && fifoCount < packetSize) {
          // try to get out of the infinite loop
          fifoCount = mpu.getFIFOCount();
        }
        // other program behavior stuff here
        // .
        // .
        // .
        // if you are really paranoid you can frequently test in between
other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
        // .
        // .
        // .
    }
```

```
    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();
    if(fifoCount < packetSize){
            //Lets go back and wait for another interrupt. We shouldn't be
here, we got an interrupt from another event
            // This is blocking so don't do it   while (fifoCount <
packetSize) fifoCount = mpu.getFIFOCount();
    }
    // check for overflow (this should never happen unless our code is too
inefficient)
    else if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) ||
fifoCount >= 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
      //  fifoCount = mpu.getFIFOCount();  // will be zero after reset no
need to ask
        Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen
frequently)
    } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {

        // read a packet from FIFO
    while(fifoCount >= packetSize){ // Lets catch up to NOW, someone is using
the dreaded delay()!
        mpu.getFIFOBytes(fifoBuffer, packetSize);
        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an
interrupt)
        fifoCount -= packetSize;
    }
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
```



```
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif
```

```
        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove
gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

8. Загрузите программу в микроконтроллер *Arduino.*
9. Убедитесь в правильности показаний датчика.

Лабораторная работа

№ 8 СЧИТЫВАНИЕ ДАННЫХ ДАТЧИКА КАСАНИЯ

Цель работы: *Познакомиться с принципом работы датчика касания.*

Задачи: *1. Установить в Arduino IDE библиотеку для датчиков Z-Robots.*
2. Выбрать из примеров библиотеки и запустить скетч для управления датчиком касания.

Краткие теоретические сведения

Модуль представляет собой плату с установленным на ней механическим микропереключателем (кнопкой) с длинным рычагом-«усом». Этот рычаг увеличивает площадь контакта и позволяет срабатывать датчику даже при легком касании. На плате установлен светодиод, который загорается при срабатывании.

Это цифровой датчик. Его состояние может быть только одним из двух:

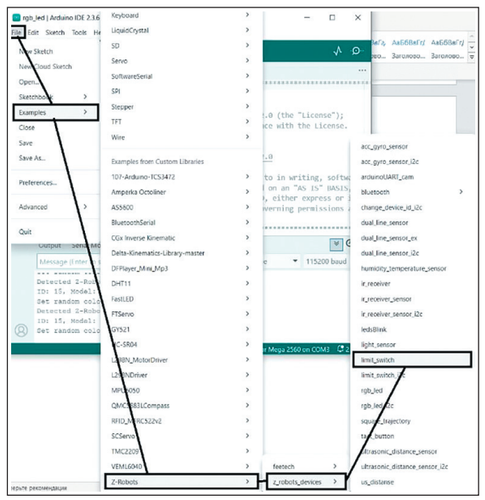
- *0 (LOW)* – кнопка отпущена (цепь разомкнута).
- *1 (HIGH)* – кнопка нажата (цепь замкнута).

Датчик предназначен для обнаружения физического контакта с объектами. Это один из самых простых и надежных сенсоров в робототехнике.

Порядок выполнения работы

Загрузите *Arduino IDE.*

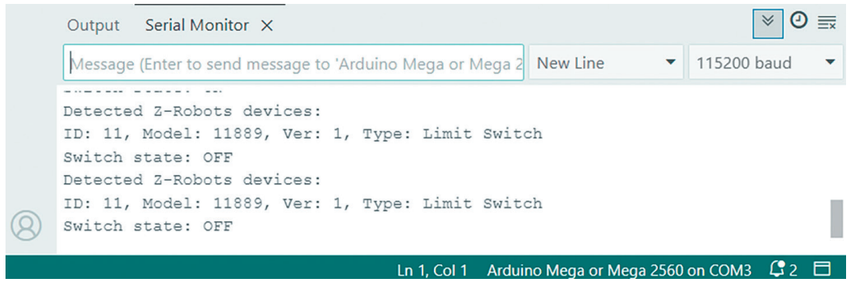
Выберите пункт меню *File → Examples → Z-Robots → z_robots_devices →*




```
/*******************************************************************************
***
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*******************************************************************************
**/

#include <Dynamixel2Arduino.h>

const uint8_t DXL_ID = 11;

Dynamixel2Arduino dxl(Serial1);

//This namespace is required to use Control table item names
using namespace ControlTableItem;

void setup() {
  // put your setup code here, to run once:

  // Use Serial to debug.
  Serial.begin(115200);

  dxl.begin();
  dxl.setPortProtocolVersion(1.0);

  Serial.println("Demo for the Limit Switch module");
}

bool switch_state = 0;

void loop() {
  // put your main code here, to run repeatedly:
```

```
  FindDevices();
  delay(2000);
}

DYNAMIXEL::InfoFromPing_t ping_info[32];
void FindDevices(void) {

  if (uint8_t count_pinged = dxl.ping(DXL_BROADCAST_ID/*DXL_ID*/, ping_info,
sizeof(ping_info) / sizeof(ping_info[0]))) {
    Serial.print("Detected Z-Robots devices: \n");
    for (int i = 0; i < count_pinged; i++)
    {
      Serial.print("ID: ");
      Serial.print(ping_info[i].id, DEC);
      Serial.print(", Model: ");
      Serial.print(ping_info[i].model_number);
      Serial.print(", Ver: " );
      Serial.print(ping_info[i].firmware_version, DEC);

      if (ping_info[i].model_number == Z_ROBOTS_LIMIT_SWITCH) {
        Serial.println(", Type: Limit Switch");
        switch_state = dxl.getSwitchState(ping_info[i].id);
        Serial.print("Switch state: ");
        if (switch_state) {
          Serial.println("ON");
        } else {
          Serial.println("OFF");
        }
      } else {
        Serial.println(", Type: Not suitable for this demo");
      }
    }
   } else {
    Serial.print("Ping returned no items");
  }
}
```


Состояние кнопки *Switch state* меняется при нажатии.



Лабораторная работа

№ 9 СЧИТЫВАНИЕ ДАННЫХ ТАКТОВОЙ КНОПКИ

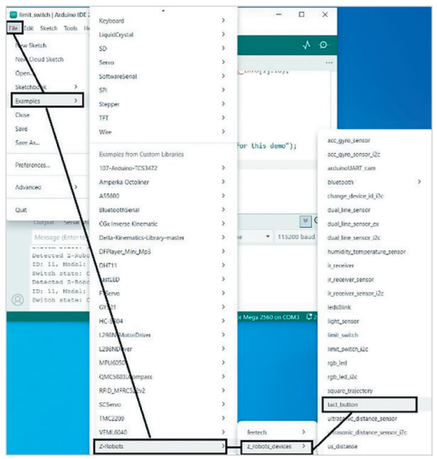
Это также цифровой сенсор, работающий по тому же принципу «0/1». Однако его физическая конструкция и предназначение другие. Кнопка нажимается пальцем и имеет короткий ход.

Цель работы: *Познакомиться с принципом работы тактовой кнопки.*

Задачи: *1. Установить в Arduino IDE библиотеку для датчиков Z-Robots.*
2. Выбрать из примеров библиотеки и запустить скетч для управления сервопривода Write.

Порядок выполнения работы

Загрузите *Arduino IDE*.
Выберите пункт меню *File → Examples → Z-Robots → z_robots_devices →*



```
/*********************************************************************
***
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
```



```
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*********************************************************************
**/

#include <Dynamixel2Arduino.h>

#define DXL_SERIAL   Serial1
#define DEBUG_SERIAL Serial
const int DXL_DIR_PIN = 2; // DYNAMIXEL Shield DIR PIN


const uint8_t DXL_ID = 10;
const float DXL_PROTOCOL_VERSION = 1.0;

Dynamixel2Arduino dxl(DXL_SERIAL, DXL_DIR_PIN);

//This namespace is required to use Control table item names
using namespace ControlTableItem;

void setup() {
  // put your setup code here, to run once:

  // Use Serial to debug.
  DEBUG_SERIAL.begin(115200);

  // Set Port baudrate to 1000000. This has to match with Z-Robots baudrate.
  dxl.begin(1000000);
  // Set Port Protocol Version. This has to match with DYNAMIXEL protocol
version.
  dxl.setPortProtocolVersion(DXL_PROTOCOL_VERSION);
}

bool button_state = false;
bool prev_button_state = false;

void loop() {
  // put your main code here, to run repeatedly:

  FindDevices();
```

```
  delay(500);
}

DYNAMIXEL::InfoFromPing_t ping_info[32];
void FindDevices(void) {
  Serial.flush(); // flush it as ping may take awhile...

  if (uint8_t count_pinged = dxl.ping(DXL_BROADCAST_ID/*DXL_ID*/, ping_info,
sizeof(ping_info) / sizeof(ping_info[0]))) {
    Serial.print("Detected Z-Robots devices: \n");
    for (int i = 0; i < count_pinged; i++)
    {
      Serial.print(" ID: ");
      Serial.print(ping_info[i].id, DEC);

      ping_info[i].model_number     = dxl.getModelNumber(ping_info[i].id);
      ping_info[i].firmware_version =
dxl.getFirmwareVersion(ping_info[i].id);

      Serial.print(", Model: ");
      Serial.print(ping_info[i].model_number);
      Serial.print(", Ver: " );
      Serial.println(ping_info[i].firmware_version, DEC);

      if (ping_info[i].model_number == Z_ROBOTS_TACT_BUTTON) {
        prev_button_state = button_state;
        button_state = dxl.getButtonState(ping_info[i].id);
        Serial.println("Connected device is the Tact Button Module");
        Serial.print("Button state: ");
        Serial.println(button_state);
      } else if ((ping_info[i].model_number == FEETECH_SCS215) ||
(ping_info[i].model_number == FEETECH_STS3215)) {
        if (prev_button_state != button_state) {
          Serial.println("Connected device is the Feetech SCS/STS servo");
          if (button_state) {
            dxl.setGoalPosition(ping_info[i].id, 4095);
            delay((4095-0)*1000/(60*50) + (60*50)*10/(50) + 50); //[(P1-
P0)/(V*50)]*1000+[(V*50)/(A*100)]*1000 + 50(误差)
          } else {
            dxl.setGoalPosition(ping_info[i].id, 0);
            delay((4095-0)*1000/(60*50) + (60*50)*10/(50) + 50); //[(P1-
P0)/(V*50)]*1000+[(V*50)/(A*100)]*1000 + 50(误差)
          }
```



```
        }
      } else {
        Serial.println("Connected device is not the Tact Button Module or the
Feetech servo!");
      }
    }
  } else {
    Serial.print("Ping returned no items : ");
    Serial.println(dxl.getLastLibErrCode());
  }
}
```

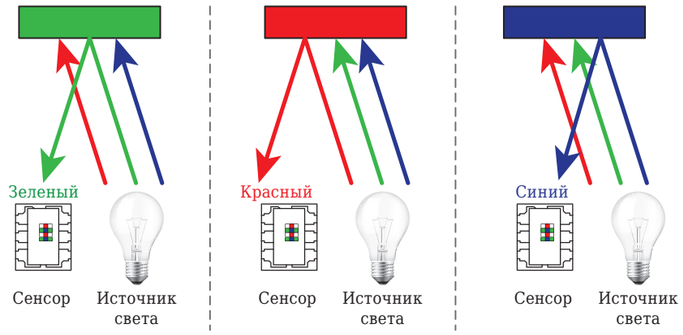
Сотояние кнопки *Button state* меняется при нажатии.

Лабораторная работа

№ 10 УПРАВЛЕНИЕ ДАТЧИКОМ ЦВЕТА

Датчик цвета набора выполнен на базе сенсора *TCS34725*. Датчик цвета *RGB TCS34725* позволяет различать цвета окружающих предметов. Доступны значения таких параметров, как: цветовая температура (по Кельвину), освещенность (в люксах), а также непосредственные значения красной, зелёной и синей составляющей. Все это благодаря наличию ИК-фильтра, позволяющего добиться более точной цветопередачи, блокируя невидимую человеческому глазу инфракрасную часть спектра.

Принцип работы заключается в анализе отражённых цветовых составляющих, излучаемых интегрированными в датчик светодиодами трёх цветов. ИК-фильтр отсекает инфракрасную составляющую, позволяя получить наиболее точные данные. Кроме того, в модуль встроен сверхяркий светодиод для подсветки точки измерения.

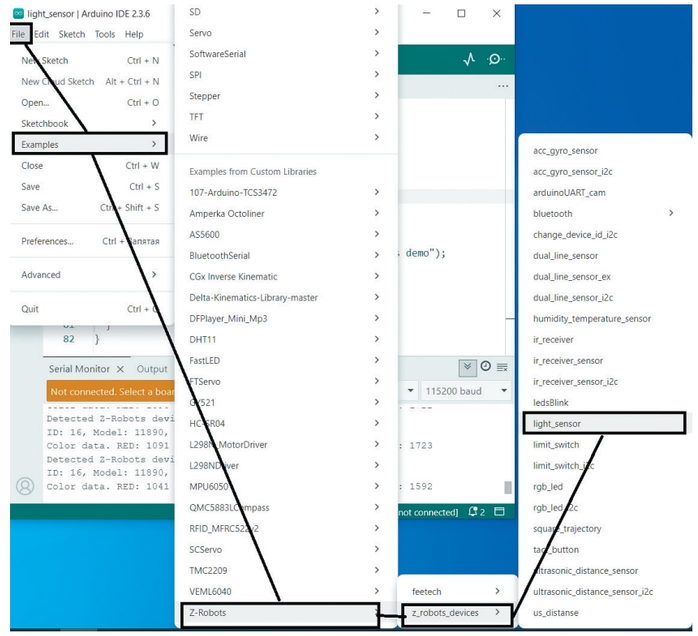


Цель работы: *Познакомиться с принципами работы датчика цвета.*

Задачи: *1. Установить в Arduino IDE библиотеку для датчиков Z-Robots*
2. Выбрать из примеров библиотеки и запустить скетч для чтения данных с датчика цвета.

Порядок выполнения работы

Загрузите *Arduino IDE.*
Выберите пункт меню *File → Examples → Z-Robots → z_robots_devices →*




```
/*********************************************************************
***
* Copyright 2016 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*********************************************************************
**/

#include <Dynamixel2Arduino.h>

const uint8_t DXL_ID = 16;

Dynamixel2Arduino dxl(Serial1);
```

```
//This namespace is required to use Control table item names
using namespace ControlTableItem;

void setup() {
  // put your setup code here, to run once:

  // Use Serial to debug.
  Serial.begin(115200);

  dxl.begin();
  dxl.setPortProtocolVersion(1.0);

  Serial.println("Demo for the IR Receiver sensor I2C");
}

void loop() {
  // put your main code here, to run repeatedly:
  FindDevices();
  delay(2000);
}
DYNAMIXEL::InfoFromPing_t ping_info[32];
void FindDevices(void) {

  if (uint8_t count_pinged = dxl.ping(DXL_BROADCAST_ID/*DXL_ID*/, ping_info,
sizeof(ping_info) / sizeof(ping_info[0]))) {
    Serial.print("Detected Z-Robots devices: \n");
    for (int i = 0; i < count_pinged; i++)
    {
      Serial.print("ID: ");
      Serial.print(ping_info[i].id, DEC);
      Serial.print(", Model: ");
      Serial.print(ping_info[i].model_number);
      Serial.print(", Ver: " );
      Serial.print(ping_info[i].firmware_version, DEC);

      if (ping_info[i].model_number == Z_ROBOTS_LIGHT_SENSOR) {
        uint16_t r = 0;
        uint16_t g = 0;
        uint16_t b = 0;
        uint16_t c = 0;
        dxl.getRawColorData(ping_info[i].id, r, g, b, c);

        Serial.println(", Type: Light sensor");
```

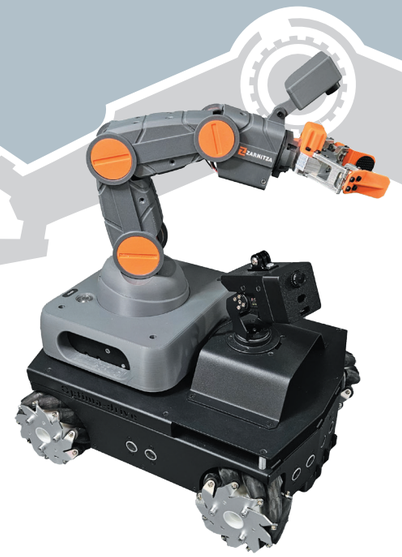


```
        Serial.print("Color data. RED: ");
        Serial.print(r);
        Serial.print(" , GREEN: ");
        Serial.print(g);
        Serial.print(" , BLUE: ");
        Serial.print(b);
        Serial.print(" , Clear light: ");
        Serial.println(c);
      } else {
        Serial.println(", Type: Not suitable for this demo");
      }
    }
   } else {
    Serial.println("Ping returned no items");
  }
}
```

Меняя предметы разного цвета, можно видеть различные значения для составляющих цвет – *R*, *G* и *B*.

ZARNITZA ROBOTICS

Разрабатываем и производим
высокотехнологичное учебное
оборудование для любых специальностей



УП6153

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

Приказ 838 Минпросвещения РФ

УП6161

Комплект для изучения операционных систем реального времени и систем управления автономных мобильных роботов Optima Drive

Приказ 838 Минпросвещения РФ

УП6340

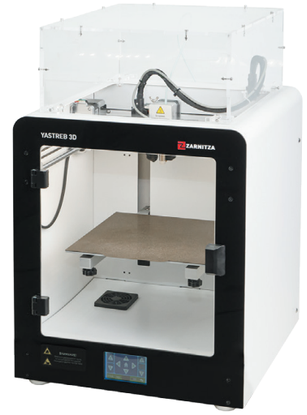
3D-сканер ручной профессиональный Yastreb 3D

Приказ 838 Минпросвещения РФ

УП6338

3D-принтер профессионального качества Zarnitsa Yastreb 3D

Приказ 838 Минпросвещения РФ



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

РОБОТОТЕХНИКА

ZARNITZA

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

RU СДЕЛАНО В РОССИИ