**Intel**лектуальная гантель, методическое пособие

# 1. Введение

1.1. Цель

Целью данного пособия является развертывание структуры проекта по сборке и усовершенствованию интеллектуальной гантели, а также формирование у преподавателя данного направления (либо у группы исследователей, ведущих работу) методических компетенций относительно проведения этапов проекта и средств достижения основных задач.

Проект развертывается на базе Intel Genuino. Применительно к используемому аппаратному обеспечению, среди отличительных черт контроллера нас будут интересовать в основном три аспекта:

— Акселерометр;

— Гироскоп;

— Модуль Bluetooth Low Energy (BLE).

Но, прежде чем приступить непосредственно к практике, очертим чуть более внятно предмет рассмотрения.

1.2. Предмет рассмотрения

Прежде всего, необходимо определиться с базовыми понятиями. Под «интеллектуальной гантелью» мы понимаем ее массогабаритный пластиковый макет, оборудованный контроллером, и программно-аппаратным комплексом, решающим ряд специфических задач, например, подсчет числа сделанных упражнений в изолированном сгибании руки:

|  |
| --- |
|  |
| Рисунок 1. Изолированное сгибание руки |

В качестве контура будущего рассмотрения обозначим необходимость выбора упражнения, подсчет числа повторов, а также контроль времени и траектории движения. Траектория может быть отображена в графической форме на стороне «сервера», либо (путем индикации отклонения от идеальной траектории) на «клиенте». Упоминание об идеальной траектории естественным образом порождает задачу калибровки: указания идеальной траектории либо апостериорно, либо параметрически.

Задача рапорта об ошибках (равно как и об успехах) тренирующегося решается с помощью голосовой обратной связи (mp3-модуль + динамик).

Развитием данной задачи можно обозначить возможность воспринимать голосовые команды (модуль распознавания голосовых команд, например, EasyVR Shield).

Предметом рассмотрения данного пособия является комплекс методических рекомендаций по следующим задачам:

1. Задача выбора упражнения #1 (с использованием мобильного приложения);
2. Задача оптимальной траектории;
3. Работа с системой воспроизведения звука;
4. Задача внешнего вида и физического размещения элементов (частично).

Остальные задачи затрагиваются с разной степенью фрагментарности — в качестве знакомства и указания направлений исследования.

Структура основных ветвей развертывания проекта представлена в общих чертах на рисунке 2.

|  |
| --- |
| F:\Documents\My Dropbox\Всякие материалы\Gotocamp\2016\Intelлектуальная гантель, схема проекта.png |
| Рисунок 2. Дерево проекта |

**Задача выбора упражнения #1** описывает ситуацию, когда упражнение выбирается с помощью мобильного приложения, «на сервере». Выбрав упражнение, пользователь тем самым задает гантели конкретные параметры контроля (например, предельно допустимые углы поворотов при сгибании).

**Задача оптимальной траектории** предусматривает контроль траектории движения при совершении упражнений. Контроль может осуществляться как параметрически (например, по максимальным отклонениям относительно осей), так и с помощью заранее измеренной «идеальной» траектории.

**Работа с системой воспроизведения звука** рассматривается как вспомогательная задача оповещения: звуковое оповещение необходимо при старте и завершении упражнения, при отклонении от оптимальной траектории исполнения, а также может быть полезно для индикации совершаемых в ходе упражнения повторов.

**Задачи внешнего вида** и **физического размещения элементов** относятся к прототипированию и дизайну. Задача проектирования внешнего вида гантели сводится к 3D-моделированию основных компонентов гантели и их печати на принтере. По второй задаче отметим ряд очевидных моментов: компактное размещение контроллера, питания и датчиков в корпусе определяет ряд требований к структуре и размерам внутреннего пространства гантели.

Краткое описание всех потенциальных направлений и задач проекта приведено в Приложении 1.

# 2. Методические рекомендации к развертыванию проекта

2.1. Конструкционные компоненты

Ключевые понятия: *конструкция гантели, аппаратное обеспечение.*

Архитектурно, гантель состоит из следующих компонентов: плата Genuino 101, MP3-плеер DFPlayer Mini с подключенным динамиком, корпус.

Плата решает задачи ориентации в гантели в пространстве и связи с «сервером», плеер необходим для решения задач голосового оповещения.

На первых этапах разработки в конструкцию может быть включен внешний Bluetooth-модуль, например, Troyka-модуль Bluetooth HC-05. По поводу этого элемента необходимо дать некоторые комментарии. Встроенный в плату Genuino 101 модуль Bluetooth Low Energy в силу ряда характеристических особенностей и неочевидного алгоритма работы может представлять определенные проблемы в освоении функциональности. Исследователям, не сталкивавшимся ранее с передачей данных по Bluetooth, рекомендуется решать задачу связи устройств итеративно: сначала использовать внешний модуль (ввиду простоты работы), потом перейти к освоению BLE. В рамках изложения данного материала мы будем придерживаться именно этой концепции.

Рассмотрим в общих чертах работу компонент гантели.

2.2. Плата Genuino 101, работа с акселерометром и гироскопом

Ключевые понятия: *библиотека Cure IMU, гироскоп, акселерометр, ориентация в пространстве, углы Эйлера.*

Ключевые функции: CurieIMU.readAccelerometerScaled*.*

Ссылки:

1. <https://www.arduino.cc/en/Guide/Arduino101> — подключение Genuino 101 и функциональных библиотек, базовые понятия.
2. <https://www.arduino.cc/en/Tutorial/Genuino101CurieIMUOrientationVisualiser> — руководство по использованию функциональности встроенного акселерометра и гироскопа.
3. <https://www.arduino.cc/en/Tutorial/Genuino101CurieIMUAccelerometerOrientation> — решение простейшей задачи ориентации с помощью акселерометра.
4. <http://www.classmech.ru/node/24> — углы Эйлера, повороты.

Перед тем как перейти непосредственно к работе со средствами визуализации положения гантели в пространстве, коротко рассмотрим базовые вопросы подключения, подробно изложенные в [1].

Прежде всего, отметим, что подключение платы потребует установки соответствующего программного обеспечения: Меню → Инструменты → Плата → Менеджер плат…

Вбив в строке поиска фразу «Genuino 101» (или просто «101») мы получим доступ к установке драйверов контроллера — см. рисунок 3:

Аналогичным образом (Меню → Скетч → Подключить библиотеку → Управлять библиотеками…) необходимо установить библиотеку CurieIMU.

Если все прошло успешно, можно приступать к работе.

|  |
| --- |
| https://www.arduino.cc/en/uploads/Guide/Board_MGR_OK.png |
| Рисунок 3. Подключение платы Genuino 101. |

**Внимание, важно!** Перед началом работы хотелось бы дать небольшой совет, который может в перспективе сэкономить вам много времени, нервов и денег. При заливке скетча на плату Genuino 101, в случае любых «тормозов» и ошибок, **ни в коем случае не выдергивайте USB-провод.** Подобное действие может привести к фатальному отказу контроллера платы — который придется (в лучшем случае) перепрошивать.

Прежде всего, подробно рассмотрим вопрос ориентации гантели в пространстве. Базовый вариант задачи рассмотрен в [3]: плата определяет свою ориентацию по трем осям и представляет результат в бинарном виде «сторона вверх / сторона вниз».

Нас будет интересовать несколько более сложная задача, связанная с определением оптимальной траектории (см. далее). Под оптимальностью здесь мы понимаем линию движения, правильную с точки зрения тренировки мышечных пучков, специфичных для данного конкретного упражнения. Как мы видим из Рисунка 1, при изолированном сгибании руки, движение производится в одной плоскости (одна степень свободы), где ограничено двумя координатными точками: нижней и верхней. Существенной особенностью данного упражнения является то, что точки разнесены друг относительно друга фактически на 180 градусов (плюс-минус некоторая погрешность). Таким образом, в самых общих чертах, оптимальной траекторией для изолированного сгибания мы будем называть периодическое качение в одной плоскости, ограниченное определенными углами — начальным и конечным.

Исходя из этих соображений, рассмотрим следующую вводную задачу прикладного свойства:

1. Нужно определить ось, относительно которой осуществляется вращение. Эту ось будем называть ключевой (ниже мы полагаем, что эта задача *уже* решена).
2. После того, как пункт 1 пройден, нужно определить положение по ключевой оси в нижней точке (перед началом упражнения).
3. Далее нужно определить положение по ключевой оси в верхней точке и разницу между нижней и верхней точкой.
4. Отдельно отметим, что при разном начальном хвате гантели, положение в нижней точке может изменяться от подхода к подходу (напомним, что мы оперируем угловой координатой), следовательно, исследовательский интерес представляет не столько начальное и конечное положение, сколько разница в показаниях по ключевой оси между верхней и нижней точкой.

Прежде чем мы перейдем непосредственно к работе с кодом, нужно сделать методическое замечание. Исходя из названий приборов «акселерометр» и «гироскоп», пользователь, не вполне знакомый с предметом, изначально ожидает, что данные акселерометра (точнее, соотнесенных с ним функций) будут иметь смысл второй производной некоей угловой координаты — то есть, ускорение. Аналогично, по неопытности ожидается, что гироскоп вернет данные об угловой координате.

Так вот, **НЕТ:**

* Функции вида readAccelerometer\* возвращают данные, которые можно трактовать **только как координату**. По всей видимости, это угловые координаты.
* Функции вида readGyro\* возвращают **ускорение**.

«Это нельзя понять, это надо просто запомнить», как в анекдоте про «тарел*ь*ку» и «вил*ь*ку».

Но вернемся к нашему рассмотрению. Полный код программы калибровки представлен в Листинге 1. Важно обратить внимание на ряд деталей. Прежде всего, это пункт 2 алгоритма — определение начального значения y\_init по ключевой оси. Здесь мы сталкиваемся с неприятной особенностью работы акселерометра: нельзя просто взять и записать в setup() одну строку:

|  |
| --- |
| y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS); |

Точнее, можно, но результат будет некорректный: некоторое время акселерометр тратит на переходной процесс, в ходе которого CurieIMU.readAccelerometerScaled возвращает нули по любой оси. Проведем калибровку до тех пор, пока не получим от акселерометра непустое значение во втором знаке после запятой:

|  |
| --- |
| //Начинаем калибровку прибора, получаем начальные значения по ключевой оси.  //y\_init - некоторое время получают только нули -  //в акселерометре идут какие-то переходные процессы.  //Дожидаемся, пока не придет что-то полезное:  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  } |

Текущее положение гантели по ключевой оси мы получаем с помощью CurieIMU.readAccelerometerScaled(Y\_AXIS); внутри цикла loop. Если модуль разницы между текущим и начальным значением возрос на очередной итерации, мы выводим полученное значение разницы в последовательный порт. Таким образом, максимальное выведенное в порт значение даст нам, выражаясь языком математического анализа, необходимую «точную верхнюю границу»:

|  |
| --- |
| float y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  //Если разница возросла, показываем ее на экране:  if (abs(y - y\_init) > dy) {  dy = abs(y - y\_init);    //Интересный эффект: различные y\_init дают разные "предельные" dy!  Serial.print("y\_init=");  Serial.print(y\_init);    //В итоге мы получим апостериорную величину максимальной разницы  //между нижней и верхней точкой на сгибании:  Serial.print("; dy=");  Serial.print(dy);  Serial.println("");  } |

Важно отметить интересное явление: различные начальные значения y\_init дают в итоге различные значения «дельты» dy. В нашем исследовании мы получили следующие данные зависимости максимума dy от y\_init:

|  |  |
| --- | --- |
| **y\_init** | **Предельное значение dy** |
| 0.00 | 1.94 (sic! Выколотая точка!) |
| 0.07 | 1.10 |
| 0.07 | 1.01 |
| -0.31 | 1.01 |
| -0.41 | 1.28 |
| -0.77 | 1.73 |
| 0.80 | 1.71 |
| 0.80 | 1.75 |
| -0.85 | 1.63 |
| 0.96 | 1.98 |

В задаче подсчета количества сделанных повторов (см. пункт 2.4) и задаче оптимальной траектории (пункт 2.7) мы будем использовать эти данные для калибровки начальной (нижней) и конечной (верхней) точки. (В Листинге 1 используется система голосового оповещения о запуске измерения — подробно этот вопрос рассматривается в следующем пункте пособия.)

|  |
| --- |
| /\*  \* Листинг 1. Определение предельных отклонений акселерометра  \* на упражнении "изолированное сгибание".  \* Определяем максимальное значение отклонения dy  \* по ключевой оси Y при определенном начальном положении y\_init.  \*  \*/  #include <CurieIMU.h>  #include <DFPlayer\_Mini\_Mp3.h>  #include <SoftwareSerial.h>  //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7); // RX (на TX у плеера), TX (на RX у плеера)  //3 порт - на нем читаем состояние плеера (свободен-занят):  #define BUSY\_PIN 3  //Инициирующее значение по ключевой оси в нижней точке:  float y\_init = 0.0;  void music (int);  void setup() {  //Настройки MP3:  pinMode(BUSY\_PIN, INPUT);  mySerial.begin (9600);  mp3\_set\_serial(mySerial); //Отдаем RX-TX  delay(1);  mp3\_set\_volume (25); //Звук в диапазоне 0-30  //Ждем завершения переходных процессов, иначе music может не сработать:  delay(300);  music(1); //Приветствие: "Все склонятся перед твоей мощью!"  delay(2000);    //Обычный последовательный порт:  Serial.begin(9600);  //ВНИМАНИЕ! Пока монитор порта не открыт, код дальше не пойдет:  while(!Serial);  //Инициализируем гироскопы-акселерометры:  CurieIMU.begin();  CurieIMU.setGyroRate(25);  CurieIMU.setAccelerometerRate(25);  //2g, 250 градусов:  CurieIMU.setAccelerometerRange(2);  CurieIMU.setGyroRange(250);  //Начинаем калибровку прибора, получаем начальные значения по ключевой оси.  //y\_init - некоторое время получают только нули -  //в акселерометре идут какие-то переходные процессы.  //Дожидаемся, пока не придет что-то полезное:  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }    Serial.println("Begin!");  music(5); //Начали!  delay(500);  }  //Текущая (абсолютная) разница между данными показаниями  //по оси и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy = 0.0;  void loop() {  float y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  //Если разница возросла, показываем ее на экране:  if (abs(y - y\_init) > dy) {  dy = abs(y - y\_init);    //Интересный эффект: различные y\_init дают разные "предельные" dy!  Serial.print("y\_init=");  Serial.print(y\_init);    //В итоге мы получим апостериорную величину максимальной разницы  //между нижней и верхней точкой на сгибании:  Serial.print("; dy=");  Serial.print(dy);  Serial.println("");  }  delay(250);  }  /\*  \* Воспроизведение заданного трека.  \*  \* Приветствия:  \* 0001 - все склонятся...  \* 0002 - да, хозяин  \* 0003 - да воцарится..  \* 0004 - пришло время бодрости  \*  \* Упражнение:  \* 0005 - начали  \* 0006, 0007 - закончили  \* 0008 - и раз  \* 0009 - и два  \* 0010, 0011 - не халтурь  \*  \* Название упражнений:  \* 0101 - Вертикальная тяга  \* 0102 - Изолированное сгибание  \*/  void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN); //BUSY\_PIN = 3  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  } |
| Листинг 1. Определение предельных отклонений акселерометра на упражнении  «изолированное сгибание» |

Задачи для самостоятельного рассмотрения:

* Отдельно отметим, что ось *Y* в качестве ключевой — следствие конструкционных особенностей имевшегося у нас изделия: для разных гантелей перед запуском данного кода надлежит определить ключевую ось — это упражнение мы оставляем читателю самостоятельному решению.
* Также отдельно отметим, что **ключевых осей на самом деле — две.** Размышления о практической пользе данного факта оставим исследователям.
* Нельзя не отметить ряд концептуальных недостатков кода, изложенного в Листинге 1: каждое следующее измерение нужно проводить, «сбрасывая» гантель в исходное состояние — аппаратным способом. Читателю предлагается модифицировать упражнение, чтобы, во-первых, в рамках одного опыта по определению зависимости максимума dy от y\_init можно было набирать несколько измерений dy (несколько разгибаний руки), а во-вторых, получить возможность менять исходные y\_init без перезагрузки гантели.
* Хорошо бы иметь возможность не просто считать максимум dy, а получать это значение в нужный момент времени (например), чтобы избежать чересчур высокого «задирания» руки в верхней точке.
* Попробуйте модифицировать задачу оптимальной траектории и, соответственно, задачу определения предельных отклонений под какое-либо другое упражнение, например, под сгибание-21, разведение рук с гантелями или базовое сгибание (без изоляции, с поворотом гантели от нижней точки к верхней).

2.3. Работа с системой воспроизведения звука

Ключевые понятия: *mp3-плеер «DFPlayer Mini», библиотека программного последовательного порта SoftwareSerial.* 

Ссылки:

1. <https://www.dfrobot.com/wiki/index.php/DFPlayer_Mini_SKU:DFR0299> — инструкция по работе с mp3-плеером.
2. <https://lesson.iarduino.ru/page/urok-17-podklyuchenie-mini-mp3-pleera-k-arduino/> — аналогичная инструкция на русском языке.
3. <https://github.com/Arduinolibrary/DFRobot_Mini_Player/raw/master/DFPlayer_Mini_mp3.zip> — библиотека для работы с mp3-плеером.
4. <https://www.arduino.cc/en/Reference/SoftwareSerial> — описание работы с SoftwareSerial.

Воспроизведение звука в рамках данной задачи исключительно важно и носит характер оповещения пользователя о различных системных событиях: старт работы, начало калибровки, начало и завершение упражнения в целом и каждого повтора в частности. Поэтому уделим некоторое время вопросам звукового сопровождения.

Отметим, что SD-карта для плеера должна быть отформатирована в FAT16 или в FAT32. На карте должна быть создана папка «mp3», в которую залиты звуковые файлы вида «0001\*.mp3», «0002\*.mp3», где «\*» — любое число любых (допустимых) символов. Подобное ограничение связано с тем, как именно плеер ищет файлы в файловой системе (подробнее см. [6,7]). Номер трека 1, 2, 3 и т.д. — без предваряющих нулей — передается в функцию проигрывания музыки (см. ниже).

Аппаратная часть модуля состоит из mp3-плеера «DFPlayer Mini» и динамика (самого обычного, на 3 Ватта мощности, сопротивлением 4 Ома). Рассмотрим, прежде всего, контактную базу плеера:

|  |
| --- |
| http://iarduino.ru/img/upload/0b89a98606316168855086afc85974f7.jpg |
| Рисунок 4. Пины mp3-плеера «DFPlayer Mini». |

Из всего набора контактов нас опишем те, что имеют к нам отношение в рамках задачи:

|  |  |
| --- | --- |
| **Контакт** | **Описание** |
| VCC | Питание, 5 Вольт. |
| RX | UART serial input, соединяется с портом 7 платы (см. ниже) |
| TX | UART serial output, соединяется с портом 6 платы (также см. ниже) |
| SPK\_1 | Выход на динамик |
| SPK\_2 | Выход на динамик |
| GND | Земля |
| BUSY | Статус текущего состояния. LOW означает, что в данный момент проигрывается трек, HIGH означает, что плеер готов к воспроизведению следующей дорожки. Соединяется с портом 3 платы. |

Дадим некоторые пояснения относительно подключения. Динамик можно подключить к портам 0 и 1 (TX, RX), однако, в нашем случае эти порты были оставлены под другую задачу: подключения по протоколу Bluetooth к мобильному приложению. При переходе на Bluetooth low energy (встроенный модуль платы Genuino) необходимость в этом отпала, но менять работающее решение нет резона. Таким образом, пара контактов RX-TX плеера выходит на 7 и 6 порты платы, которые отданы под программный Serial port:

|  |
| --- |
| //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7); // RX (на TX у плеера), TX (на RX у плеера)  #define BUSY\_PIN 3 |

В завершение темы упомянем BUSY-порт плеера, соединяемый с 3 пином платы. При проигрывании звуковых файлов возможна ситуация, когда команда на воспроизведение следующего трека придет до завершения предыдущего. В этом случае мы ждем, пока BUSY-порт не освободится. Функция music отвечает за воспроизведение конкретного трека:

|  |
| --- |
| void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  } |

Задачи для самостоятельного рассмотрения:

* Отметим, что индикация с помощью голосовых сообщений является не вполне удовлетворительной. Исследователям предлагается разработать систему световой и/или вибрационной индикации. Последняя может быть осуществлена, например, с помощью двигателя с закрепленным эксцентриком. Предлагаем продумать также свои системы индикации, поскольку светом и звуком данная тема, конечно же, не исчерпывается.

2.4. Подсчет количества повторений в упражнении «изолированное сгибание руки»

Ключевые понятия: *калибровка отклонений по ключевой оси, подсчет числа повторов в две фазы.*

Рассмотрим задачу подсчета количества повторений, совершаемых спортсменом в одном подходе. Алгоритм работы в данной задаче, с учетом предыдущих пунктов, выглядит следующим образом:

1. Нужно определить координату нижней точки y\_init и задать величину предельного отклонения от нее dy\_down\_limit.
2. По значению y\_init нужно откалибровать dy\_up\_limit — величину отклонения до верхней точки.
3. Сразу после старта упражнения мы считаем, что движемся снизу вверх. Это состояние сохраняется, пока модуль разницы текущего положения по ключевой оси y и изначального значения y\_init меньше dy\_up\_limit.
4. По достижении верхней точки, мы оповещаем об этом пользователя и начинаем фазу движения вниз. Это состояние сохраняется пока модуль разницы y и y\_init больше погрешности dy\_down\_limit.
5. По достижении нижней точки, мы уведомляем пользователя, увеличиваем число совершенных повторений и переходим к пункту 3 данного алгоритма.

Рассмотрим предложенный алгоритм подробно. Калибровка (пункт 1) вынесена в отдельную функцию void calibrate (bool voice), вызываемую перед началом упражнения. Передаваемый параметр voice служит флагом оповещения: если он равен true, мы выводим на динамик оповещение о процессе калибровки:

|  |
| --- |
| //Инициирующее значение по ключевой оси. Его достижение означает прохождение нижней точки в упражнении:  float y\_init = 0.0;  //Значение МАКСИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если значение стало меньше dy\_down\_limit,  //значит мы пришли в нижнюю точку.  float dy\_down\_limit = 0.2;  //Значение МИНИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если разница между текущим y и y\_init стала больше dy\_up\_limit,  //мы пришли в верхнюю точку.  //Значение dy\_up\_limit калибруется в зависимости от abs(y\_init).  float dy\_up\_limit = 0.0; |

…

|  |
| --- |
| /\*  \* Калибруем начальные значения по ключевой оси.  \* y\_init - некоторое время только нули,  \* в акселерометре идут какие-то переходные процессы.  \* Дожидаемся, пока не придет что-то полезное  \* и выставляем dy\_up\_limit в зависимости от полученного значения.  \*/  void calibrate (bool voice) {  //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }    //Калибруем величину dy\_up\_limit в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit = 1.75;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit = 1.56;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit = 1.2;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit = 1.0;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit = 1.9;  }    //Если мы не в silent mode (перекалибруем посередине упражнения):  if (voice) {  //Калибровка завершена, говорим "Начали!"  music(5);  delay(500);  }  } |

Пункты 2-5 алгоритма вынесены в функцию void ex\_isolated\_flexion (int rpt\_limit), осуществляющую контроль текущего положения и отсчет числа сделанных повторов. Функция принимает в качестве параметра максимальное число повторов, выполняемых в подходе:

|  |
| --- |
| void ex\_isolated\_flexion (int rpt\_limit) {  //Говорим название упражнения: "Изолированное сгибание".  music(102);  delay(1000);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //Выполняем нужное число повторов:  for (int rpt = 1; rpt <= rpt\_limit; rpt++) {  //Пока движемся наверх и dy меньше предела, ничего не делаем:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy < dy\_up\_limit);    //Говорим, "и, раз!":  music(8);  delay(500);  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy > dy\_down\_limit);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate(false);    //Говорим, "и, два!":  music(9);  delay(500);    //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  }    music(6); //Закончили  delay(1000);  } |

Полный код программы приведен в листинге 2:

|  |
| --- |
| /\*  \* Листинг 2.  \* Считаем число повторов для упражнения "изолированное сгибание руки".  \* Фиксируем верхнюю и нижнюю точку, выводим число повторов в порт.  \*/  #include <CurieIMU.h>  #include <DFPlayer\_Mini\_Mp3.h>  #include <SoftwareSerial.h>  //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7); // RX (на TX у плеера), TX (на RX у плеера)  //3 порт - на нем читаем состояние плеера (свободен-занят):  #define BUSY\_PIN 3  //Инициирующее значение по ключевой оси.  //Его достижение означает прохождение нижней точки в упражнении:  float y\_init = 0.0;  //Значение МАКСИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если значение стало меньше dy\_down\_limit,  //значит мы пришли в нижнюю точку.  float dy\_down\_limit = 0.2;  //Значение МИНИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если разница между текущим y и y\_init стала больше dy\_up\_limit,  //мы пришли в верхнюю точку.  //Зачение dy\_up\_limit калибруется в зависимости от abs(y\_init).  float dy\_up\_limit = 0.0;  void music (int);  void calibrate (bool voice);  void ex\_isolated\_flexion (int);  void setup() {  //Обычный последовательный порт, сюда выводим данные:  Serial.begin(9600);  //ВНИМАНИЕ! Пока монитор порта не открыт, код дальше не пойдет:  while(!Serial);    //Инициализируем гироскопы-акселерометры:  CurieIMU.begin();  CurieIMU.setGyroRate(25);  CurieIMU.setAccelerometerRate(25);  //Устанавливаем акселерометр на диапазон 2G:  CurieIMU.setAccelerometerRange(2);  //Устанавливаем гироскоп на диапазон 250 градусов:  CurieIMU.setGyroRange(250);    //Настройки MP3:  pinMode(BUSY\_PIN, INPUT);  mySerial.begin (9600);  mp3\_set\_serial(mySerial); //Отдаем RX-TX  delay(1);  mp3\_set\_volume (25); //Звук в диапазоне 0-30  //Ждем завершения переходных процессов, иначе music может не сработать:  delay(300);    //Приветствие: "Да, хозяин!"  music(2);  Serial.println("Begin!");  delay(2000);  }  void loop() {  //Начинаем калибровку акселерометра,  //получаем начальное значение y\_init  //и рассчитываем dy\_up\_limit:  calibrate(true);  //Делаем изолированное сгибание на пять повторов:  ex\_isolated\_flexion(5);  //Не повторяем:  exit(0);  }  void ex\_isolated\_flexion (int rpt\_limit) {  //Говорим "Начали!"  music(5);  delay(1000);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //На всякий случай выводим, сколько у нас начальное  //и сколько предельное значение:  Serial.print("y\_init = ");  Serial.print(y\_init);  Serial.print("; dy\_up\_limit = ");  Serial.print(dy\_up\_limit);  Serial.println("");    for (int rpt = 1; rpt <= rpt\_limit; rpt++) {    //Пока движемся наверх и dy меньше предела, ничего не делаем:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy < dy\_up\_limit);    //Говорим, "и, раз!":  music(8);  delay(300);  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy > dy\_down\_limit);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate(false);    //Говорим, "и, два!":  music(9);  delay(300);    //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  }  //Говорим "Закончили!"  music(6);  delay(1000);  }  /\*  \* Калибруем начальные значения по ключевой оси.  \* y\_init - некоторое время только нули,  \* в акселерометре идут какие-то переходные процессы.  \* Дожидаемся, пока не придет что-то полезное  \* и выставляем dy\_up\_limit в зависимости от полученного значения.  \*/  void calibrate (bool voice) {  //Если мы не в silent mode (перекалибруемся посередине упражнения):  if (voice) {  //Говорим название упражнения: "Изолированное сгибание".  music(102);  //И ждем секунду, чтобы спортсмен успел принять начальное положение  delay(2000);  }    //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }    //Калибруем величину dy\_up\_limit в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit = 1.75;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit = 1.56;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit = 1.2;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit = 1.0;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit = 1.9;  }  dy\_up\_limit = dy\_up\_limit\*0.8;    }  /\*  \* Воспроизведение заданного трека.  \*  \* Приветствия:  \* 0001 - все склонятся...  \* 0002 - да, хозяин  \* 0003 - да воцарится..  \* 0004 - пришло время бодрости  \*  \* Упражнение:  \* 0005 - начали  \* 0006, 0007 - закончили  \* 0008 - и раз  \* 0009 - и два  \* 0010, 0011 - не халтурь  \*  \* Название упражнений:  \* 0101 - Вертикальная тяга  \* 0102 - Изолированное сгибание  \*/  void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN); //BUSY\_PIN = 3  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  } |
| Листинг 2. Подсчет числа выполненных повторов в упражнении  «изолированное сгибание» |

Задачи для самостоятельного рассмотрения:

* Рекомендуем исследователю обратить внимание на потенциальные усовершенствования в математическом аспекте алгоритма подсчета числа сгибаний. Сейчас этот процесс реализуется достаточно примитивным образом: сравниваются данные по одной оси с их предельными значениями. Можно использовать более продвинутый подход: положение в пространстве представлять в виде вектора *(x, y, z)*, где каждая из координат получена вызовом readAccelerometerScaled по соответствующей оси. В таком представлении достижение предельных значений можно оценивать, например, с помощью векторного произведения трехмерных координат текущего положения и начального.

2.5. Средства беспроводной связи, передача данных от мобильного приложения гантели и обратно: работа с Troyka-модулем Bluetooth HC-05

Ключевые понятия: *Bluetooth HC-05.*

Ссылки:

1. <http://wiki.amperka.ru/продукты:troyka-bluetooth> — установка и настройка модуля HC-05 troyka module.
2. <https://play.google.com/store/apps/details?id=eu.jahnestacado.arduinorc> — Arduino Bluetooth Controller; контроллер для Bluetooth-соединения, мобильное приложение для обмена информацией с модулем.
3. <https://play.google.com/store/apps/details?id=Qwerty.BluetoothTerminal> — терминал Bluetooth, приложение для мобильного телефона, аналогичное Arduino Bluetooth Controller.

Прежде, чем перейти к использованию встроенного модуля BLE, рассмотрим более простой вариант — работу с использованием troika-модуля Bluetooth HC-05. Исчерпывающее руководство по аппаратной части приведено в [9], программная часть фактически неотличима от работы с обычным последовательным портом. Перед тем как перейти к подробному рассмотрению работы, сформулируем задачу: код из Листинга 2 (подсчет числа выполненных повторов) модифицируем таким образом, чтобы:

1. Упражнение начиналось только по команде от мобильного приложения
2. Число повторений задавалось командой мобильного приложения.
3. Текущее число повторов (переменная rpt) отсылалось на мобильное приложение.

Прежде всего, нужно инициализировать порт связи Bluetooth, это делается командой Serial1.begin — абсолютно аналогично инициализации Serial port’а:

|  |
| --- |
| void setup() {  …  …  //Инициализируем Bluetooth:  Serial1.begin(9600);  …  } |

Далее, в цикле мы должны убедиться, что связь с мобильным устройством установлена и от мобильного клиента получено максимальное число сгибаний. Если число сгибаний получено, запускаем упражнение на калибровку и исполнение:

|  |
| --- |
| void loop() {  //Ожидаем, пока по Bluetooth-каналу придет что-либо:  while (Serial1.available() <= 0);  //Приходит один байт, ascii-код символа.  //Число 0 будет представлено как 48, 1 как 49, 9 как 57.  //Для получения числа сгибаний, извлекаем цифру из кода  int rpt\_limit = Serial1.read()-48;  if ((rpt\_limit > 0) && (rpt\_limit <= 9)){  //Начинаем калибровку акселерометра,  //получаем начальное значение y\_init  //и рассчитываем dy\_up\_limit:  calibrate(true);    //Делаем изолированное сгибание на пять повторов:  ex\_isolated\_flexion(rpt\_limit);  }  } |

Обращаем внимание, что команда возвращает один байт, ASCII-код переданного символа. Цифра 0 таким образом «на приеме» будет представлена как 48, цифра 1 как 49, 9 — как 57. Полный код примера приведен в листинге 3:

|  |
| --- |
| /\*  \* Листинг 3.  \* Использование bluetooth hc-05.  \* Считаем число повторов для упражнения "изолированное сгибание руки".  \* Максимальное число повторов (от 1 до 9) получаем по команде  \* от мобильного приложения.  \* Текущее число повторов отсылаем мобильному приложению.  \*/  #include <CurieIMU.h>  #include <DFPlayer\_Mini\_Mp3.h>  #include <SoftwareSerial.h>  //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7); // RX (на TX у плеера), TX (на RX у плеера)  //3 порт - на нем читаем состояние плеера (свободен-занят):  #define BUSY\_PIN 3  //Инициирующее значение по ключевой оси.  //Его достижение означает прохождение нижней точки в упражнении:  float y\_init = 0.0;  //Значение МАКСИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если значение стало меньше dy\_down\_limit,  //значит мы пришли в нижнюю точку.  float dy\_down\_limit = 0.2;  //Значение МИНИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если разница между текущим y и y\_init стала больше dy\_up\_limit,  //мы пришли в верхнюю точку.  //Зачение dy\_up\_limit калибруется в зависимости от abs(y\_init).  float dy\_up\_limit = 0.0;  void music (int);  void calibrate (bool voice);  void ex\_isolated\_flexion (int);  void setup() {  //Обычный последовательный порт, сюда выводим данные:  Serial.begin(9600);  //Инициализируем Bluetooth:  Serial1.begin(9600);    //ВНИМАНИЕ! Пока монитор порта не открыт, код дальше не пойдет:  while(!Serial);    //Инициализируем гироскопы-акселерометры:  CurieIMU.begin();  CurieIMU.setGyroRate(25);  CurieIMU.setAccelerometerRate(25);  //Устанавливаем акселерометр на диапазон 2G:  CurieIMU.setAccelerometerRange(2);  //Устанавливаем гироскоп на диапазон 250 градусов:  CurieIMU.setGyroRange(250);    //Настройки MP3:  pinMode(BUSY\_PIN, INPUT);  mySerial.begin (9600);  mp3\_set\_serial(mySerial); //Отдаем RX-TX  delay(1);  mp3\_set\_volume (25); //Звук в диапазоне 0-30  //Ждем завершения переходных процессов, иначе music может не сработать:  delay(300);    //Приветствие: "Да, хозяин!"  music(2);  Serial.println("Begin!");  delay(2000);  }  void loop() {  //Ожидаем, пока по Bluetooth-каналу придет что-либо:  while (Serial1.available() <= 0);  //Приходит один байт, ascii-код символа.  //Число 0 будет представлено как 48, 1 - как 49, 9 - как 57.  //Для получения числа сгибаний, извлекаем цифру из кода  int rpt\_limit = Serial1.read()-48;  if ((rpt\_limit > 0) && (rpt\_limit <= 9)){  //Начинаем калибровку акселерометра,  //получаем начальное значение y\_init  //и рассчитываем dy\_up\_limit:  calibrate(true);    //Делаем изолированное сгибание на пять повторов:  ex\_isolated\_flexion(rpt\_limit);  }  }  void ex\_isolated\_flexion (int rpt\_limit) {  //Говорим "Начали!"  music(5);  delay(1000);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //На всякий случай выводим, сколько у нас начальное  //и сколько предельное значение:  Serial.print("y\_init = ");  Serial.print(y\_init);  Serial.print("; dy\_up\_limit = ");  Serial.print(dy\_up\_limit);  Serial.println("");    for (int rpt = 1; rpt <= rpt\_limit; rpt++) {    //Пока движемся наверх и dy меньше предела, ничего не делаем:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy < dy\_up\_limit);    //Говорим, "и, раз!":  music(8);  delay(300);  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy > dy\_down\_limit);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate(false);    //Говорим, "и, два!":  music(9);  delay(300);    //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  //Отсылаем текущее число повторов по Bluetooth.  //rpt-48 - по той же причине, что и на приеме -  //отправляем ascii-код. Понятно, что можем отправить  //только число от 0 до 9:  Serial1.println(rpt-48);  }  //Говорим "Закончили!"  music(6);  delay(1000);  }  /\*  \* Калибруем начальные значения по ключевой оси.  \* y\_init - некоторое время только нули,  \* в акселерометре идут какие-то переходные процессы.  \* Дожидаемся, пока не придет что-то полезное  \* и выставляем dy\_up\_limit в зависимости от полученного значения.  \*/  void calibrate (bool voice) {  //Если мы не в silent mode (перекалибруемся посередине упражнения):  if (voice) {  //Говорим название упражнения: "Изолированное сгибание".  music(102);  //И ждем секунду, чтобы спортсмен успел принять начальное положение  delay(2000);  }    //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }    //Калибруем величину dy\_up\_limit в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit = 1.75;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit = 1.56;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit = 1.2;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit = 1.0;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit = 1.9;  }  dy\_up\_limit = dy\_up\_limit\*0.8;    }  /\*  \* Воспроизведение заданного трека.  \*  \* Приветствия:  \* 0001 - все склонятся...  \* 0002 - да, хозяин  \* 0003 - да воцарится..  \* 0004 - пришло время бодрости  \*  \* Упражнение:  \* 0005 - начали  \* 0006, 0007 - закончили  \* 0008 - и раз  \* 0009 - и два  \* 0010, 0011 - не халтурь  \*  \* Название упражнений:  \* 0101 - Вертикальная тяга  \* 0102 - Изолированное сгибание  \*/  void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN); //BUSY\_PIN = 3  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  } |
| Листинг 3. Подсчет числа выполненных повторов в упражнении  «изолированное сгибание» с передачей данных по Bluetooth |

Задачи для самостоятельного рассмотрения:

* В приведенном выше скетче мы ограничиваемся приемом одного байта и числом повторов до девяти. Расширение функциональности до работы с любым числом повторений оставим читателю.

2.6. Средства беспроводной связи, передача данных от мобильного приложения гантели и обратно: работа с BLE

Ключевые понятия: *Bluetooth low energy, сервисы BLE, характеристики BLE.*

Ссылки:

1. <https://www.arduino.cc/en/Reference/CurieBLE> — описание функциональности библиотеки CurieBLE library.
2. <https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner> — мобильное приложение для работы с BLE-устройствами

BLE — новый стандарт Bluetooth-обмена, оптимизированный под экономный расход энергии и передачу данных на малых скоростях обмена. BLE работает по принципу «доски объявлений»: каждое периферийное устройство (наподобие нашей гантели) представляет собой такую доску, информацию с которой читают (и записывают) центральные устройства.

Информация, предоставляемая периферийными устройствами, структурирована в виде **сервисов**. Сервисы в свою очередь подразделяются на **характеристики**. Сервисы могут быть представлены как заметки на упомянутой выше доске объявлений, а характеристики — как отдельные параграфы таких заметок. Или, чуть менее поэтично, сервис можно представить как канал связи с конкретным устройством, а характеристики — как разнородные потоки данных внутри канала.

Сервис в первую очередь характеризуется своим UUID — уникальным идентификатором:

|  |
| --- |
| //Создаем сервис и присваиваем ему UUID:  BLEService dumbbellService("19B10010-E8F2-537E-4F6C-D104768A1214"); |

Что касается характеристик, то они подразделяются, прежде всего, по типу передаваемых данных. Вот основные типы характеристик:

* BLEBoolCharacteristic;
* BLECharCharacteristic;
* BLEUnsignedCharCharacteristic;
* BLEShortCharacteristic;
* BLEUnsignedShortCharacteristic;
* BLEIntCharacteristic;
* BLEUnsignedIntCharacteristic;
* BLELongCharacteristic;
* BLEUnsignedLongCharacteristic;
* BLEFloatCharacteristic;
* BLEDoubleCharacteristic.

Как мы видим, именование класса характеристики содержит исчерпывающую информацию о том, данными какого типа мы будем обмениваться с центральным устройством. Каждая характеристика состоит из:

* Имени;
* UUID;
* Хранимого значения;
* Свойства read/write/notify.

На последнем пункте остановимся подробнее. Свойство BLERead определяет возможность периферийного устройства получить данные от центрального, в нашем случае — гантели получить данные от телефона. BLEWrite, напротив, позволяет отправить данные от гантели к телефону. Наконец, BLENotify позволяет центральному устройству вести непрерывный опрос периферийного устройства, чтобы оперативно реагировать на изменение данных.

|  |
| --- |
| //Создаем характеристику типа char, в которую будет записываться  //Желаемое количество повторов (rpt\_limit: смартфон -> гантеля).  //BLERead - запрос значение характеристики из мобильного приложения;  //BLENotify - непрерывный опрос характеристики телефоном;  //BLEWrite - отправка сообщения от телефона плате.  BLECharCharacteristic rptLimitCharacteristic  ("19B10011-E8F2-537E-4F6C-D104768A1215", BLERead | BLENotify | BLEWrite);  //Создание характеристики типа int, в которую будет записываться  //текущее число повторов (rpt: гантеля -> смартфон)  BLEIntCharacteristic rptCharacteristic  ("19B10012-E8F2-537E-4F6C-D104768A1216", BLERead | BLENotify); |

Инициализация периферийного устройства проводится в функции initBLE:

|  |
| --- |
| void initBLE (void)  {  blePeripheral.setLocalName("Dumbbell");  blePeripheral.setAdvertisedServiceUuid(dumbbellService.uuid());  blePeripheral.addAttribute(dumbbellService);  blePeripheral.addAttribute(rptLimitCharacteristic);  blePeripheral.addAttribute(rptCharacteristic);  //Инициализируем периферийное устройство, публикуем сервис:  blePeripheral.begin();  } |

В основном цикле программы мы опрашиваем список событий на предмет обновления с помощью функции blePeripheral.poll(). Далее мы проверяем, поступили ли данные в характеристику rptLimitCharacteristic с помощью вызова rptLimitCharacteristic.written(). Если данные есть, мы считываем значение с помощью rptLimitCharacteristic.value() и выполняем упражнение:

|  |
| --- |
| void loop() {  //Опрашиваем устройство на предмет событий:  blePeripheral.poll();  //Есть ли данные от мобильного приложения?  if(rptLimitCharacteristic.written())  {  //Если данные поступили, получаем значение rpt\_limit:  int rpt\_limit = int(rptLimitCharacteristic.value()) - 48;  Serial.print("rpt\_limit = ");  Serial.println(rpt\_limit);    //Начинаем калибровку акселерометра,  //получаем начальное значение y\_init  //и рассчитываем dy\_up\_limit:  calibrate(true);    //Делаем изолированное сгибание на rpt\_limit повторов:  ex\_isolated\_flexion(rpt\_limit);    //Не повторяем:  exit(0);  }  } |

Данные о совершенном числе повторов мы отправляем на мобильное приложение с помощью rptCharacteristic.setValue():

|  |
| --- |
| //И отправляем в мобильник:  rptCharacteristic.setValue(rpt); |

Полный код примера приводится в листинге 4:

|  |
| --- |
| /\*  \* Листинг 4.  \* Использование Bluetooth Low Energy.  \* Считаем число повторов для упражнения "изолированное сгибание руки".  \* Максимальное число повторов (от 1 до 9) получаем по команде  \* от мобильного приложения.  \* Текущее число повторов отсылаем мобильному приложению через BLE.  \*/  #include <CurieBLE.h>  #include <CurieIMU.h>  #include <DFPlayer\_Mini\_Mp3.h>  #include <SoftwareSerial.h>  /\*\*\* BLE data \*\*\*/  //Создаем перефирийное устройство:  BLEPeripheral blePeripheral;  //Создаем сервис и присваиваем ему UUID:  BLEService dumbbellService("19B10010-E8F2-537E-4F6C-D104768A1214");  //Создаем характеристику типа char, в которую будет записываться  //Желаемое количество повторов (rpt\_limit: смартфон -> гантеля).  //BLERead - запрос значение характеристики из мобильного приложения;  //BLENotify - непрерывный опрос характеристики телефоном;  //BLEWrite - отправка сообщения от телефона плате.  BLECharCharacteristic rptLimitCharacteristic  ("19B10011-E8F2-537E-4F6C-D104768A1215", BLERead | BLENotify | BLEWrite);  //Создание характеристики типа int, в которую будет записываться  //текущее число повторов (rpt: гантеля -> смартфон)  BLEIntCharacteristic rptCharacteristic  ("19B10012-E8F2-537E-4F6C-D104768A1216", BLERead | BLENotify);  void initBLE (void);  /\*\*\* BLE data end \*\*\*/  //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7);  //3 порт - на нем читаем состояние плеера (свободен-занят):  #define BUSY\_PIN 3  //Инициирующее значение по ключевой оси.  //Его достижение означает прохождение нижней точки в упражнении:  float y\_init = 0.0;  //Значение МАКСИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если значение стало меньше dy\_down\_limit,  //значит мы пришли в нижнюю точку.  float dy\_down\_limit = 0.2;  //Значение МИНИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если разница между текущим y и y\_init стала больше dy\_up\_limit,  //мы пришли в верхнюю точку.  //Зачение dy\_up\_limit калибруется в зависимости от abs(y\_init).  float dy\_up\_limit = 0.0;  void music (int);  void calibrate (bool voice);  void ex\_isolated\_flexion (int);  void initBLE (void);  void setup() {  //Обычный последовательный порт, сюда выводим данные:  Serial.begin(9600);    //Инициализируем гироскопы-акселерометры:  CurieIMU.begin();  CurieIMU.setGyroRate(25);  CurieIMU.setAccelerometerRate(25);  //Устанавливаем акселерометр на диапазон 2G:  CurieIMU.setAccelerometerRange(2);  //Устанавливаем гироскоп на диапазон 250 градусов:  CurieIMU.setGyroRange(250);    //Настройки MP3:  pinMode(BUSY\_PIN, INPUT);  mySerial.begin (9600);  mp3\_set\_serial(mySerial); //Отдаем RX-TX  delay(1);  mp3\_set\_volume (25); //Звук в диапазоне 0-30  //Ждем завершения переходных процессов, иначе music может не сработать:  delay(300);  //Запускаем BLE:  initBLE();    //Приветствие: "Все склонятся перед твоей мощью!"  music(1);  delay(2000);  }  void loop() {  //Опрашиваем устройство на предмет событий:  blePeripheral.poll();  //Есть ли данные от мобильного приложения?  if(rptLimitCharacteristic.written())  {  //Если данные поступили, получаем значение rpt\_limit:  int rpt\_limit = int(rptLimitCharacteristic.value()) - 48;  Serial.print("rpt\_limit = ");  Serial.println(rpt\_limit);    //Начинаем калибровку акселерометра,  //получаем начальное значение y\_init  //и рассчитываем dy\_up\_limit:  calibrate(true);    //Делаем изолированное сгибание на rpt\_limit повторов:  ex\_isolated\_flexion(rpt\_limit);    //Не повторяем:  exit(0);  }  }  void ex\_isolated\_flexion (int rpt\_limit) {  //Калибровка пошла, говорим "Начали!"  music(5);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //На всякий случай выводим, сколько у нас начальное  //и сколько предельное значение:  Serial.print("y\_init = ");  Serial.print(y\_init);  Serial.print("; dy\_up\_limit = ");  Serial.print(dy\_up\_limit);  Serial.println("");    for (int rpt = 1; rpt <= rpt\_limit; rpt++) {    //Пока движемся наверх и dy меньше предела, ничего не делаем:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy < dy\_up\_limit);    //Говорим, "и, раз!":  music(8);  delay(500);  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  } while (dy > dy\_down\_limit);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate(false);    //Говорим, "и, два!":  music(9);  delay(500);    //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  //И отправляем в мобильник:  rptCharacteristic.setValue(rpt);  }    music(6); //Закончили  delay(1000);  }  /\*  \* Калибруем начальные значения по ключевой оси.  \* y\_init - некоторое время только нули,  \* в акселерометре идут какие-то переходные процессы.  \* Дожидаемся, пока не придет что-то полезное  \* и выставляем dy\_up\_limit в зависимости от полученного значения.  \*/  void calibrate (bool voice) {  //Если мы не в silent mode (перекалибруемся посередине упражнения):  if (voice) {  //Говорим название упражнения: "Изолированное сгибание".  music(102);  //И ждем секунду, чтобы спортсмен успел принять начальное положение  delay(2000);  }    //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }    //Калибруем величину dy\_up\_limit в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit = 1.75;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit = 1.56;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit = 1.2;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit = 1.0;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit = 1.9;  }  dy\_up\_limit = dy\_up\_limit\*0.8;    }  void initBLE (void)  {  blePeripheral.setLocalName("Dumbbell");  blePeripheral.setAdvertisedServiceUuid(dumbbellService.uuid());  blePeripheral.addAttribute(dumbbellService);  blePeripheral.addAttribute(rptLimitCharacteristic);  blePeripheral.addAttribute(rptCharacteristic);  //Инициализируем периферийное устройство, публикуем сервис:  blePeripheral.begin();  }  /\*  \* Воспроизведение заданного трека.  \*  \* Приветствия:  \* 0001 - все склонятся...  \* 0002 - да, хозяин  \* 0003 - да воцарится..  \* 0004 - пришло время бодрости  \*  \* Упражнение:  \* 0005 - начали  \* 0006, 0007 - закончили  \* 0008 - и раз  \* 0009 - и два  \* 0010, 0011 - не халтурь  \*  \* Название упражнений:  \* 0101 - Вертикальная тяга  \* 0102 - Изолированное сгибание  \*/  void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN); //BUSY\_PIN = 3  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  } |
| Листинг 4. Подсчет числа выполненных повторов в упражнении  «изолированное сгибание» с передачей данных через BLE |

2.7. Задача оптимальной траектории

Ключевые понятия: *Работа с гироскопом, работа с акселерометром, работа с mp3-плеером.*

Ключевые функции: CurieIMU.readAccelerometerScaled, CurieIMU.readGyroScaled.

Ссылки:

1. <https://www.arduino.cc/en/Reference/CurieIMU> — список функций библиотеки CureIMU, предназначенной для работы с данными системы акселерометр-гироскоп.
2. <https://www.arduino.cc/en/Reference/CurieTimerOne> — библиотека для работы с прерываниями таймера.

Рассмотрим простейшую реализацию задачи оптимальной траектории — на примере упражнения «изолированное сгибание руки», см. Рисунок 1.

Как уже было отмечено ранее, в данном случае контроль траектории можно обеспечить исходя из следующего соображения: упражнение должно совершаться с перемещением гантели в одной плоскости. По сути, в этой плоскости происходит качение, ограниченное двумя константными угловыми координатами. Таким образом, алгоритм контроля оптимальной траектории в данном случае следующий:

1. Для контроля траектории достаточно фиксировать данные по углу (угловой координате) в нижней и верхней точке упражнения. В нашем случае это данные y-координаты, получаемые с помощью CurieIMU.readAccelerometerScaled(Y\_AXIS).
2. Нижнюю точку мы получаем в самом начале упражнения, считывая данные по y-координате. Назовем эту координату y\_init.
3. В дальнейшем контроль достижения нижней точки мы ведем по разнице между текущей y-координатой и начальной координатой y\_init. Как только разница стала меньше некоей (апостериорно найденной) величины «дельта», мы считаем, что в очередной раз проходится нижняя точка (при условии, что мы движемся вниз — это тоже фиксируется).
4. Контроль достижения верхней точки мы осуществляем на основании разницы между текущей y-координатой и начальной координатой y\_init. Грубо говоря, эта координата должна составлять величину, близкую к 180 градусам. Реальные значения разницы опять-таки получены апостериори (см. пункт 2.2).
5. Контроль отклонения от оптимальной траектории производится по ходу движения путем подсчета отклонений *x*-координаты от начального значения x\_init. Если это отклонение больше определенных «предельных» значений, мы фиксируем ошибку.

Для разнообразия в код программы добавлено второе упражнение: вертикальная тяга с отведением руки вбок. Представление об этом упражнении дает Рисунок 4.

В скетч также добавлена возможность выбора упражнения — для этого мы в цикле основной программы опрашиваем характеристику rptLimitCharacteristic дважды: сначала для выбора упражнения, потом — для получения требуемого числа повторений:

|  |
| --- |
| //Выбираем упражнение (по умолчанию - изолированное сгибание):  int ex\_number = ISOLATED\_FLECTION;  //Опрашиваем события до момента записи в характеристику:  do {  blePeripheral.poll();  } while (!rptLimitCharacteristic.written());  //Данные приходят как char, получам число из кода:  ex\_number = int(rptLimitCharacteristic.value()) - 48;  //Дождались номера упражнения, если он корректный, ждем число повторов:  if ((ex\_number == ISOLATED\_FLECTION) or (ex\_number == VERTICAL\_TRACTION)) {  //Ждем количество повторов и делаем упражнение:  int rpt\_limit = 5;  CurieTimerOne.start(oneSecInUsec/2, &timed\_blink\_isr);  do {  blePeripheral.poll();  } while (!rptLimitCharacteristic.written());  rpt\_limit = int(rptLimitCharacteristic.value()) - 48; |

|  |
| --- |
|  |
| Рисунок 4. Упражнение «Вертикальная тяга с отведением руки вбок» |

Сделаем ряд методических замечаний по поводу контроля ошибки на примере изолированного сгибания. В нашей реализации контроль ошибки выведен в функцию bool test\_error\_isolated\_flexion (void). Функция рассчитывает ошибку исходя из величины отклонения по оси X от исходного x\_init. Если эта ошибка больше некоторого предела (полученного опытным путем), мы выводим предупреждение.

|  |
| --- |
| /\*  \* Проверяем текущую ошибку по вспомогательным осям (здесь X).  \* Если ошибка вышла за предел err\_limit,  \* выводим голосовое предупреждение.  \*/  bool test\_error\_isolated\_flexion (void) {  //Максимально допустимое отклонение по вспомогательным осям.  //Выход за это отклонение означает ошибку выполнения упражнения:  float err\_limit = 0.45;    static bool err\_flag = false;  static unsigned long int err\_time = millis();    float x = CurieIMU.readAccelerometerScaled(X\_AXIS);  float dx = abs(x - x\_init);  const int SIZE = 20;  static int i = -1;  //Ошибки собираем в массив и считаем среднее, чтобы  //защититься от ситуации пограничных флуктуаций  //когда текущее dx находится вблизи предела err\_limit  static float error\_list[SIZE];  //Инициализация:  if (-1 == i) {  for (int j = 0; j < SIZE; j++) error\_list[j] = 0;  }  //ФИКСИРУЕМ НОВУЮ ОШИБКУ:  i = (i+1)%SIZE;  error\_list[i] = dx;    float error = 0.0;  for (int j = 0; j < SIZE; j++) {  error += error\_list[j];  }  error /= SIZE;  //Если функция ошибки вышла за предел и сделала это ТОЛЬКО ЧТО  //(err\_flag опущен; так защищаемся от "атаки ошибок"),  //то обновляем измерение и оповещаем спортсмена.  if ((error > err\_limit) && !err\_flag && (millis() > err\_time+1000)) {  err\_time = millis();  err\_flag = true;  char buffer[255];  sprintf(  buffer,  "%.2f: err\_flag ON with error = %.2f; %c", err\_time/1000.0,  error, 0  );  Serial.println(buffer);  //Предупреждаем: "Неправильно!"  music(11);  }  //Если же мы в корректной области, то сбрасываем флаг ошибки,  //чтобы иметь возможность реагировать на новые:  if ((error < err\_limit) && err\_flag) {  err\_flag = false;  char buffer[255];  sprintf(buffer, "%.2f: err\_flag OFF %c", millis()/1000.0, 0);  Serial.println(buffer);  }  digitalWrite(LED\_PIN, err\_flag? HIGH : LOW);  return err\_flag;  } |

Функция рассчитывает ошибку error как среднее арифметическое от величины abs(x - x\_init), используя окно на 20 повторений. Если ошибка превысила определенный апостериорный предел err\_limit, мы выставляем флаг состояния ошибки err\_flag и обновляем момент ее фиксации:

|  |
| --- |
| err\_time = millis();  err\_flag = true; |

Это сделано для того, чтобы не выводить сигнал об ошибке несколько раз подряд.

В целом методика не лишена недостатков. Прежде всего, своего вдумчивого исследователя еще ждет более корректная функция расчета ошибки; имеющаяся формула на одни «ложные» движения будет реагировать хуже, чем на другие.

Полная реализация, сопровождаемая подробными комментариями, описана ниже:

|  |
| --- |
| /\*  \* Листинг 5, основной.  \* 1. Выбираем упражнение.  \* 2. Выбираем максимальное число повторов для него.  \* 3. Считаем число повторов для упражнения "изолированное сгибание руки".  \* или для упражнения "вертикальная (боковая) тяга".  \* 4. Фиксируем верхнюю и нижнюю точку, выводим число повторов в порт.  \* и на мобильное приложение через BLE.  \* 5. Фиксируем ошибку в течение 3-4. Ошибку выводим голосом и светодиодом.  \*  \*/  #include <CurieBLE.h>  #include <CurieIMU.h>  #include <DFPlayer\_Mini\_Mp3.h>  #include <SoftwareSerial.h>  #include <CurieTimerOne.h>  /\*\*\* BLE data \*\*\*/  //Создаем перефирийное устройство:  BLEPeripheral blePeripheral;  //Создаем сервис и присваиваем ему UUID:  BLEService dumbbellService("19B10010-E8F2-537E-4F6C-D104768A1214");  //Создаем характеристику типа char, в которую будет записываться  //1) Номер упражнения;  //2) Желаемое количество повторов (rpt\_limit: смартфон -> гантеля)  //BLERead - запрос значение характеристики из мобильного приложения;  //BLENotify - непрерывный опрос характеристики телефоном;  //BLEWrite - отправка сообщения от телефона плате.  BLECharCharacteristic rptLimitCharacteristic  ("19B10011-E8F2-537E-4F6C-D104768A1215", BLERead | BLENotify | BLEWrite);  // Создание характеристики типа int, в которую будет записываться  // текущее число повторов (rpt: гантеля -> смартфон)  // и подтверждающий сигнал о выборе упражнения  BLEIntCharacteristic rptCharacteristic  ("19B10012-E8F2-537E-4F6C-D104768A1216", BLERead | BLENotify);  void initBLE (void);  /\*\*\* BLE data end \*\*\*/  //Используем таймер, чтобы моргать светодиодом.  //Количество микросекунд в секунде:  const int oneSecInUsec = 1000000;  //Порт светодиода:  #define LED\_PIN 13  //Порт под MP3-плеер:  SoftwareSerial mySerial(6, 7); // RX (на TX у плеера), TX (на RX у плеера)  #define BUSY\_PIN 3  //Инициирующее значение по ключевой оси. Его достижение означает  //прохождение нижней точки в упражнении (0 - инициализиция):  //И инициирующее значение по осям контроля. Отклонение от этих значений на величину,  //большую чем err\_limit означает ошибку выполнения упражнения.  //(0.0 - просто инициализиция переменной):  float y\_init = 0.0, x\_init = 0.0, z\_init = 0.0;  //Значение МАКСИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если значение стало меньше dy\_down\_limit,  //значит мы пришли в нижнюю точку.  float dy\_down\_limit = 0.2;  //Значение МИНИМАЛЬНО допустимой разницы между текущим  //положением гантели и начальным y\_init.  //Если разница между текущим y и y\_init стала  //больше dy\_up\_limit\_isolated\_flexion (dy\_up\_limit\_vertical\_traction),  //мы пришли в верхнюю точку.  //Зачение dy\_up\_limit\_isolated\_flexion (dy\_up\_limit\_vertical\_traction)  //калибруется в зависимости от abs(y\_init).  float dy\_up\_limit\_isolated\_flexion = 0;  float dy\_up\_limit\_vertical\_traction = 0;  void music (int);  void calibrate\_isolated\_flexion (bool voice);  void calibrate\_vertical\_traction (bool voice);  void ex\_isolated\_flexion (int);  void ex\_vertical\_traction (int);  bool test\_error\_isolated\_flexion (void);  bool test\_error\_vertical\_traction (void);  void timed\_blink\_isr(void) ;  #define ISOLATED\_FLECTION 1  #define VERTICAL\_TRACTION 2  void setup() {  pinMode(LED\_PIN, OUTPUT);  //Мигаем светодиодом 4 раза в секунду с помощью прерывания таймера:  CurieTimerOne.start(oneSecInUsec / 4, &timed\_blink\_isr);  //Последовательный порт:  Serial.begin(9600);    //Инициализируем гироскопы-акселерометры:  CurieIMU.begin();  CurieIMU.setGyroRate(25);  CurieIMU.setAccelerometerRate(25);  //Устанавливаем акселерометр на диапазон 2G:  CurieIMU.setAccelerometerRange(2);  //Устанавливаем гироскоп на диапазон 250 градусов:  CurieIMU.setGyroRange(250);    //Настройки MP3:  //3 порт - на нем читаем состояние плеера (свободен-занят)  pinMode(BUSY\_PIN, INPUT);  mySerial.begin (9600);  mp3\_set\_serial(mySerial); //Отдаем RX-TX  delay(1);  mp3\_set\_volume (20); //Звук в диапазоне 0-30  //Ждем завершения переходных процессов,  //иначе music может не сработать:  delay(300);    //Приветствие: "Да, хозяин!"  music(2);  delay(2000);  //Запускаем BLE:  initBLE();  }  void loop() {  //Выбираем упражнение (по умолчанию - изолированное сгибание):  int ex\_number = ISOLATED\_FLECTION;  CurieTimerOne.start(oneSecInUsec/3, &timed\_blink\_isr);  //Опрашиваем события до момента записи в характеристику:  do {  blePeripheral.poll();  } while (!rptLimitCharacteristic.written());  //Данные приходят как char, получам число из кода:  ex\_number = int(rptLimitCharacteristic.value()) - 48;    /\*  //Тестовая альтернатива - опрашиваем Serial port:  while(Serial.available() <= 0);  ex\_number = int(Serial.read()) - 48;  \*/  Serial.print("ex\_number = ");  Serial.println(ex\_number);  if (ex\_number == ISOLATED\_FLECTION) {  //Произносим название упражнения:  music(102);  }  if (ex\_number == VERTICAL\_TRACTION) {  //Произносим название упражнения:  music(101);  }  //Дождались номера упражнения, если он корректный, ждем число повторов:  if ((ex\_number == ISOLATED\_FLECTION) or (ex\_number == VERTICAL\_TRACTION)) {  //Ждем количество повторов и делаем упражнение:  int rpt\_limit = 5;  CurieTimerOne.start(oneSecInUsec/2, &timed\_blink\_isr);  do {  blePeripheral.poll();  } while (!rptLimitCharacteristic.written());  rpt\_limit = int(rptLimitCharacteristic.value()) - 48;    /\*  while(Serial.available() <= 0);  rpt\_limit = int(Serial.read()) - 48;  \*/    Serial.print("rpt\_limit = ");  Serial.println(rpt\_limit);  if (ISOLATED\_FLECTION == ex\_number) {  Serial.print("Calibrate... ");  CurieTimerOne.start(oneSecInUsec/1, &timed\_blink\_isr);  delay(500);  calibrate\_isolated\_flexion(true);  Serial.println("Calibrated!");  CurieTimerOne.pause();  digitalWrite(LED\_PIN, LOW);  //  ex\_isolated\_flexion(rpt\_limit);    } else if (VERTICAL\_TRACTION == ex\_number) {  Serial.print("Calibrate... ");  CurieTimerOne.start(oneSecInUsec/2, &timed\_blink\_isr);  delay(500);  calibrate\_vertical\_traction(true);  Serial.println("Calibrated!");  CurieTimerOne.pause();  digitalWrite(LED\_PIN, LOW);  //  ex\_vertical\_traction(rpt\_limit);  }    } else {  //Отсылаем "ошибку при выборе упражнения":  rptCharacteristic.setValue(9);    Serial.print("ex\_number wrong value: ");  Serial.println(ex\_number);  }  }  //УПРАЖНЕНИЯ:  void ex\_isolated\_flexion (int rpt\_limit) {    //Говорим "Начали!".  music(5);  delay(1000);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //На всякий случай выводим, сколько у нас начальное  //и сколько предельное значение:  Serial.print("y\_init = ");  Serial.print(y\_init);  Serial.print("; dy\_up\_limit\_isolated\_flexion = ");  Serial.print(dy\_up\_limit\_isolated\_flexion);  Serial.println("");    for (int rpt = 1; rpt <= rpt\_limit; rpt++) {  bool error = false;  //Пока движемся наверх и dy меньше предела,  //проверяем текущую ошибку по вспомогательным осям:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  //Проверяем на ошибку, уведомляем, если ошиблись:  error = test\_error\_isolated\_flexion();  } while ((dy < dy\_up\_limit\_isolated\_flexion) || error);    //Говорим, "и, раз!":  music(8);  Serial.println("Up!");  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  //Снова проверяем на ошибку, уведомляем, если ошиблись:  error = test\_error\_isolated\_flexion();  } while ((dy > dy\_down\_limit) || error);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate\_isolated\_flexion(false);    //Говорим, "и, два!":  music(9);  Serial.println("Down!");  //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  //И отправляем в мобильник:  rptCharacteristic.setValue(rpt);  }    music(6); //Закончили  Serial.println("Finished!");  delay(3000);  }  void ex\_vertical\_traction (int rpt\_limit) {  //Говорим "Начали!".  music(5);  delay(1000);    //Текущая координата по ключевой оси:  float y;  //Текущая (абсолютная) разница между данными показаниями по оси  //и их инициирующим значением.  //Чем она больше, тем на больший угол мы отклонились по оси:  float dy;    //На всякий случай выводим, сколько у нас начальное  //и сколько предельное значение:  Serial.print("y\_init = ");  Serial.print(y\_init);  Serial.print("; dy\_up\_limit\_vertical\_traction = ");  Serial.print(dy\_up\_limit\_vertical\_traction);  Serial.println("");    for (int rpt = 1; rpt <= rpt\_limit; rpt++) {  bool error = false;  //Пока движемся наверх и dy меньше предела,  //проверяем текущую ошибку по вспомогательным осям:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  //Проверяем на ошибку, уведомляем, если ошиблись:  error = test\_error\_vertical\_traction();  } while ((dy < dy\_up\_limit\_vertical\_traction) || error);    //Говорим, "и, раз!":  music(8);  Serial.println("Up!");  //Движемся вниз:  do {  y = CurieIMU.readAccelerometerScaled(Y\_AXIS);  dy = abs(y - y\_init);  //Снова проверяем на ошибку, уведомляем, если ошиблись:  error = test\_error\_vertical\_traction();  } while ((dy > dy\_down\_limit) || error);    //Тут, в принципе, можно перекалибровать данные y\_init  //в нижней точке вызовом calibrate\_isolated\_flexion(false);    //Говорим, "и, два!":  music(9);  Serial.println("Down!");  //Количество сделанных повторов выводим в порт:  Serial.print("rpt = ");  Serial.println(rpt);  //И отправляем в мобильник:  rptCharacteristic.setValue(rpt);  }    music(6); //Закончили  Serial.println("Finished!");  delay(3000);  }  //ОШИБКИ:  /\*  \* Проверяем текущую ошибку по вспомогательным осям (здесь X).  \* Если ошибка вышла за предел err\_limit,  \* выводим голосовое предупреждение.  \*/  bool test\_error\_isolated\_flexion (void) {  //Максимально допустимое отклонение по вспомогательным осям.  //Выход за это отклонение означает ошибку выполнения упражнения:  float err\_limit = 0.45;    static bool err\_flag = false;  static unsigned long int err\_time = millis();    float x = CurieIMU.readAccelerometerScaled(X\_AXIS);  float dx = abs(x - x\_init);  const int SIZE = 20;  static int i = -1;  //Ошибки собираем в массив и считаем среднее, чтобы  //защититься от ситуации пограничных флуктуаций  //когда текущее dx находится вблизи предела err\_limit  static float error\_list[SIZE];  //Инициализация:  if (-1 == i) {  for (int j = 0; j < SIZE; j++) error\_list[j] = 0;  }  //ФИКСИРУЕМ НОВУЮ ОШИБКУ:  i = (i+1)%SIZE;  error\_list[i] = dx;    float error = 0.0;  for (int j = 0; j < SIZE; j++) {  error += error\_list[j];  }  error /= SIZE;  //Если функция ошибки вышла за предел и сделала это ТОЛЬКО ЧТО  //(err\_flag опущен; так защищаемся от "атаки ошибок"),  //то обновляем измерение и оповещаем спортсмена.  if ((error > err\_limit) && !err\_flag && (millis() > err\_time+1000)) {  err\_time = millis();  err\_flag = true;  char buffer[255];  sprintf(  buffer,  "%.2f: err\_flag ON with error = %.2f; %c", err\_time/1000.0,  error, 0  );  Serial.println(buffer);  //Предупреждаем: "Неправильно!"  music(11);  }  //Если же мы в корректной области, то сбрасываем флаг ошибки,  //чтобы иметь возможность реагировать на новые:  if ((error < err\_limit) && err\_flag) {  err\_flag = false;  char buffer[255];  sprintf(buffer, "%.2f: err\_flag OFF %c", millis()/1000.0, 0);  Serial.println(buffer);  }  digitalWrite(LED\_PIN, err\_flag? HIGH : LOW);  return err\_flag;  }  bool test\_error\_vertical\_traction (void) {  //Максимально допустимое отклонение по вспомогательным осям.  //Выход за это отклонение означает ошибку выполнения упражнения:  float err\_limit = 0.33;    static bool err\_flag = false;    static unsigned long int err\_time = millis();    float x = CurieIMU.readAccelerometerScaled(X\_AXIS);  float dx = abs(x - x\_init);  const int SIZE = 20;  static int i = -1;  //Ошибки собираем в массив и считаем среднее, чтобы  //защититься от ситуации пограничных флуктуаций  //когда текущее dx находится вблизи предела err\_limit  static float error\_list[SIZE];  //Инициализация:  if (-1 == i) {  for (int j = 0; j < SIZE; j++) error\_list[j] = 0;  }  //ФИКСИРУЕМ НОВУЮ ОШИБКУ:  i = (i+1)%SIZE;  error\_list[i] = dx;    float error = 0.0;  for (int j = 0; j < SIZE; j++) {  error += error\_list[j];  }  error /= SIZE;  //Если функция ошибки вышла за предел и сделала это ТОЛЬКО ЧТО  //(err\_flag опущен; так защищаемся от "атаки ошибок"),  //то обновляем измерение и оповещаем спортсмена.  if ((error > err\_limit) && !err\_flag && (millis() > err\_time+1000)) {  err\_time = millis();  err\_flag = true;  char buffer[255];  sprintf(  buffer,  "%.2f: err\_flag ON with error = %.2f; %c",  err\_time/1000.0,  error,  0  );  Serial.println(buffer);  //Предупреждаем: "Неправильно!"  music(11);  }  //Если же мы в корректной области, то сбрасываем флаг ошибки,  //чтобы иметь возможность реагировать на новые:  if ((error < err\_limit) && err\_flag) {  err\_flag = false;  char buffer[255];  sprintf(buffer, "%.2f: err\_flag OFF %c", millis()/1000.0, 0);  Serial.println(buffer);  }  digitalWrite(LED\_PIN, err\_flag? HIGH : LOW);  return err\_flag;  }  //КАЛИБРОВКИ:  /\*  \* Калибруем начальные значения по ключевой оси.  \* y\_init - некоторое время только нули,  \* в акселерометре идут какие-то переходные процессы.  \* Дожидаемся, пока не придет что-то полезное  \* и выставляем dy\_up\_limit\_isolated\_flexion в зависимости от полученного значения.  \*/  void calibrate\_isolated\_flexion (bool voice) {    //Если мы не в silent mode (перекалибруемся посередине упражнения):  if (voice) {  //Калибровка начинается, произносим название упражнения  //"Изолированное сгибание":  music(102);  //Ждем пару секунд, чтобы спортсмен успел встать  //в исходную позицию на калибровку:  delay(3000);  }  y\_init = 0;    //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }  //Наберем сотню значений за секунду и возьмем среднее:  for (int i = 1; i < 100; i++){  delay(10);  y\_init += CurieIMU.readAccelerometerScaled(Y\_AXIS);  }  y\_init /= 100;      //Калибруем величину dy\_up\_limit\_isolated\_flexion в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit\_isolated\_flexion = 1.65;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit\_isolated\_flexion = 1.46;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit\_isolated\_flexion = 1.1;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit\_isolated\_flexion = 0.9;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit\_isolated\_flexion = 1.8;  }  //Теперь калибруем вспомогательные оси.  //Вопрос о калибровке предельной ошибки  //в зависимости от показаний - остается открытым:  x\_init = 0;  int x = x\_init \* 100;  while (x == 0) {  x\_init = CurieIMU.readAccelerometerScaled(X\_AXIS);  x = x\_init \* 100;  }  z\_init = 0;  int z = z\_init \* 100;  while (z == 0) {  z\_init = CurieIMU.readAccelerometerScaled(Z\_AXIS);  z = z\_init \* 100;  }  }  void calibrate\_vertical\_traction (bool voice) {    //Если мы не в silent mode (перекалибруемся посередине упражнения):  if (voice) {  //Калибровка начинается, произносим название упражнения  //"Вертикальная тяга":  music(101);  //Ждем пару секунд, чтобы спортсмен успел встать  //в исходную позицию на калибровку:  delay(3000);  }  y\_init = 0;    //Калибруем, пока не получим значение, отличное от нуля в  //двух знаках после запятой (y\_init\*100):  int y = y\_init \* 100;  while (y == 0) {  y\_init = CurieIMU.readAccelerometerScaled(Y\_AXIS);  y = y\_init \* 100;  }  //Наберем сотню значений за секунду и возьмем среднее:  for (int i = 1; i < 100; i++){  delay(10);  y\_init += CurieIMU.readAccelerometerScaled(Y\_AXIS);  }  y\_init /= 100;      //Калибруем величину dy\_up\_limit\_isolated\_flexion в зависимости от y\_init  //(апостериорные данные).  //Наибольшим y\_init соответствует максимальный лимит:  if (0.8 < abs(y\_init)) {  dy\_up\_limit\_vertical\_traction = 1.65;  }    //Постепенно понижаем лимит при уменьшении y\_init:  if (0.8 >= abs(y\_init)) {  dy\_up\_limit\_vertical\_traction = 1.46;  }    if (0.5 >= abs(y\_init)) {  dy\_up\_limit\_vertical\_traction = 1.1;  }    if (0.31 >= abs(y\_init)) {  dy\_up\_limit\_vertical\_traction = 0.9;  }    //То ли это ошибка нормировки, то ли выделенная точка,  //но при 0.0 - предельное значение вот такое:  if (0.0 == abs(y\_init)) {  dy\_up\_limit\_vertical\_traction = 1.8;  }  dy\_up\_limit\_vertical\_traction \*= 0.85;  //Теперь калибруем вспомогательные оси.  //Вопрос о калибровке предельной ошибки  //в зависимости от показаний - остается открытым:  x\_init = 0;  int x = x\_init \* 100;  while (x == 0) {  x\_init = CurieIMU.readAccelerometerScaled(X\_AXIS);  x = x\_init \* 100;  }    z\_init = 0;  int z = z\_init \* 100;  while (z == 0) {  z\_init = CurieIMU.readAccelerometerScaled(Z\_AXIS);  z = z\_init \* 100;  }  }  /\*  \* Воспроизведение заданного трека.  \*  \* Приветствия:  \* 0001 - все склонятся...  \* 0002 - да, хозяин  \* 0003 - да воцарится..  \* 0004 - пришло время бодрости  \*  \* Упражнение:  \* 0005 - начали  \* 0006, 0007 - закончили  \* 0008 - и раз  \* 0009 - и два  \* 0010, 0011 - не халтурь  \*  \* Название упражнений:  \* 0101 - Вертикальная тяга  \* 0102 - Изолированное сгибание  \*/  void music(int i) {  boolean play\_state;  do {  play\_state = digitalRead(BUSY\_PIN); //BUSY\_PIN = 3  } while (play\_state == LOW);  mp3\_play(i);  do {  play\_state = digitalRead(BUSY\_PIN);  } while (play\_state == LOW);  }  //Моргаем светодиодом с помощью прерываний таймера:  void timed\_blink\_isr(void)  {  static volatile bool toggle = 0;  digitalWrite(LED\_PIN, toggle);  toggle = !toggle;  }  void initBLE (void)  {  blePeripheral.setLocalName("Dumbbell");  blePeripheral.setAdvertisedServiceUuid(dumbbellService.uuid());  blePeripheral.addAttribute(dumbbellService);  blePeripheral.addAttribute(rptLimitCharacteristic);  blePeripheral.addAttribute(rptCharacteristic);  //Ожидаем подключения:  blePeripheral.begin();  } |
| Листинг 5. Контроль траектории в упражнении «изолированное сгибание», передача данных по BLE |

Дадим ряд комментариев. В скетче используется дополнительная световая индикация (по отношению к изначально проектируемой голосовой) — с помощью описанной в [15] библиотеки CurieTimerOne.h мы имеем возможность нужное количество раз в секунду вызывать функцию по прерыванию:

|  |
| --- |
| //Мигаем светодиодом 4 раза в секунду с помощью прерывания таймера:  CurieTimerOne.start(oneSecInUsec / 4, &timed\_blink\_isr); |

В данном случае функция вызывается четыре раза в секунду. Сама функция просто меняет состояние светодиода (13 пин) с включенного на выключенное и наоборот на каждом следующем вызове:

|  |
| --- |
| void timed\_blink\_isr(void)  {  static volatile bool toggle = 0;  digitalWrite(LED\_PIN, toggle);  toggle = !toggle;  } |

Статическая переменная toggle объявлена volatile на всякий случай: чтобы компилятор не применил какую-либо несовместимую с планируемой функциональностью оптимизацию.

Задачи для самостоятельного рассмотрения:

* Для разминки можно попробовать усложнить используемую световую индикацию, чтобы было более удобно различать режимы функционирования гантели: настройку, ожидание ввода упражнения, ожидание числа повторений, процесс работы.
* Попробуйте добавить дополнительные упражнения к имеющимся двум.
* Предыдущее задание можно дополнить задачей разбиения упражнений на отдельные независимые модули, подключаемые как .h-файлы.
* Функцию расчета ошибки можно усовершенствовать, если использовать весовую функцию для скользящего среднего.
* Дальнейшим усовершенствованием является более продвинутый алгоритм расчета ошибки: например, используя представление текущий координаты в виде трехмерного вектора *(x, y, z)* и рассчитывая ошибку как оценку векторного произведения текущей координаты и координаты «идеальной траектории».

2.8. Прототипирование и дизайн; внешний вид гантели

Ссылки:

1. <http://www.autodesk.com/products/fusion-360/overview> — пакет 3D-моделирования.

Задачи прототипирования и дизайна относятся к сфере 3D-моделирования и технологического проектирования. Коснемся отдельных аспектов данной темы.

При проектировании внешнего вида гантели исследователю надлежит начать с формулировки технического задания, охватывающего следующие вопросы:

1. Какой функциональностью будет наделено проектируемое устройство?
2. Исходя из предыдущего пункта, какие электронные компоненты проектируемое устройство должно содержать?
3. Какие управляющие и индикаторные элементы будет содержать конструкция (кнопка включения-выключения, световой индикатор, динамик и т.д.)?
4. Какие ограничения накладываются на физические параметры корпуса (прочностные и температурные характеристики, предельные размеры и т.д.)?
5. Как разместить компоненты в пространстве внутри корпуса?
6. Как обеспечить доступ к компонентам в режиме отладочного тестирования?
7. Какие требования предъявляются к внешнему виду устройства и как эти требования соотносятся с перечисленными выше вопросами?

Данный этап — ключевой при проектировании. Важно не пренебрегать составлением ТЗ и тщательно продумывать ответы на приведенные выше вопросы. Небрежности и недочеты этапа составления ТЗ неизбежно приводят к заметным сложностям на этапе разработки конкретной функциональности.

*Иллюстративный пример: в исходном ТЗ на гантель была обойдена вниманием такая вещь как… выключатель. При проектировании, соответственно, этот момент был упущен. В результате, во-первых, для перезапуска скетча требуется серия непростых манипуляций с корпусом (нужно отворачивать болты, крепящие центральную часть ручки гантели — накладку) и проводами питания. Во-вторых, гантель постоянно потребляет ток от батареи, что приводит к совершенно необоснованному снижению «моторесурса».*

*Для ликвидации данного конструктивного недостатка выключатель пришлось «врезать» в уже готовый корпус.*

Перед тем как перейти к моделированию с использованием трехмерного редактора, необходимо составить эскиз с обозначением основных элементов конструкции и оценкой размеров. Это даст представление о том, как подойти к задачам трехмерного моделирования:

|  |
| --- |
|  |
| Рисунок 5. Эскиз изделия |

Благодаря эскизу (Рисунок 5) конструктор понимает, как изделие видит «заказчик», а также, где располагаются те или иные компоненты.

Работая с эскизом, полезно также получить т.н. «референсные данные»: какие решения такой (или подобной) задачи уже существуют:

|  |
| --- |
|  |
| Рисунок 6. Референсные данные |

При решении задачи 3D-моделирования первостепенную важность приобретает вопрос взаимной компоновки элементов устройства в корпусе. Для его решения полезно создать 3D-модель каждого элемента архитектуры (плата Genuino, динамик, элемент питания, макетная плата с плеером) — чтобы иметь возможность решать вопрос размещения компонентов более предметно.

При проектировании данного конкретного изделия решено было следовать принципу наборной гантели с утяжелителями. Результат представлен на Рисунке 7:

|  |
| --- |
|  |
| Рисунок 7. Внешний вид гантели. |

Выбор именно такой конструкции корпуса исходил из концепции горизонтального расположения платы контроллера, определившего конфигурацию внутренних компонентов гантели. Самым жестким ограничением было горизонтальное расположение платы — обуславливающее увеличение линейных размеров гантели.

Это диктовало свое направление в моделировании корпуса. В ходе дизайнерского исследования с учетом заложенных ограничений, было принято решение сделать съемные утяжелители:

|  |
| --- |
|  |
| Рисунок 8. Утяжелители. |

Внутри площадки крепления утяжелителей разместилась плата, для балансировки гантели аккумулятор был смещен немного от центра в противоположную сторону от платы:

|  |
| --- |
|  |
| Рисунок 9. Площадка крепления утяжелителей, вид изнутри. |

Задачи для самостоятельного рассмотрения:

* При дальнейшей модернизации гантели имеет смысл предусмотреть систему встроенной индикации (дисплей), что позволит выводить дополнительную информацию для человека.
* Разумным представляется предусмотреть систему включения-выключения с помощью тумблера или двухпозиционной кнопки.
* Важным **(на самом деле — критически важным)** аспектом является обеспечение возможности заливки в гантель нового скетча без внесения длительных операций по сборке-разборке. Для данного изделия такая возможность отсутствует: заливка новой программы требует снятия центральной накладки с гантели (см. Рисунок 7), закрепляемой четырьмя винтами.

# Приложение 1. Обзор задач и направлений исследования в проекте

Дадим краткий обзор всех задач, представленных на рисунке 1 в качестве дерева развития проекта.

**Введение** представляет собой занятие-знакомство с основами Arduino-разработки. Это первое из ряда ознакомительных занятий, продвинутые участники вполне смогут его пропустить.

**Область** **программирования** естественным образом разбивается на две: серверную и клиентскую, или, точнее, «мобильную» и «железную». Это крайне обширный комплекс занятий, ориентированный в основном на новичков, продолжительность которого сильно зависит от изначальной программистской подготовки участников. Для не владеющих предметом в самом простом варианте это два-три занятия на «железную часть» (переменные, ввод-вывод, условные конструкции, циклы) и одно-два на клиентскую (платформа мобильного прототипирования «MIT App Inventor»).

**Схемотехника** представляет собой знакомство с макетной платой, правилами соединения датчиков с контроллером и подключения питания. Ориентировочно одно занятие. Продвинутые слушатели этот этап, понятно, пропускают.

Схемотехнике и задачам программирования «железа» наследуют направления работы с индикаторами (в частности, с **графическим дисплеем** и **системой воспроизведения звука**) и **сенсорами** — в число которых входят как раз акселерометр и гироскоп. Отметим, что это направление развития гантели может включать в себя самые разнообразные элементы: кнопки, джойстики и т.п.

**Прототипирование** и **дизайн** включают в себя все, что относится к внешнему виду гантели: 3D-моделирование, печать, задачи дизайна внешнего вида и компоновки элементов.

Вышеперечисленным общим областям наследуют конкретные частные задачи.

**Задача протокола обмена.** Речь здесь не идет о транспортных протоколах, а о системе команд, которыми обмениваются гантель и мобильное приложение (компьютер). Как отдавать приказ гантели на выбор упражнения? Как возвращать количество повторов? Как прерывать выполнение упражнения? Все упомянутое входит в круг задач, обозначенный как «протокол обмена».

**Задача выбора упражнения #1** описывает ситуацию, когда упражнение выбирается на мобильном «сервере». Выбрав упражнение, пользователь задает гантели конкретные параметры контроля (например, предельно допустимые углы поворотов при сгибании).

**Задача сбора статистики** подразумевает «отчет» от гантели о проделанной работе, длительное хранение данных между сессиями-тренировками и построение отчетов-графиков, например, по динамике повторов и числу ошибок на каждое упражнение.

**Задачу будильника** следует понимать в смысле составления программы тренировок, например, планирования числа подходов и тайминга времени отдыха между подходами, с выдачей оповещений о необходимости старта упражнения. Задача очевидным образом расширяется до планирования тренировки целиком (число повторов, число подходов, конкретные упражнения) и выдачи уведомлений о том, когда и к какому упражнению надо приступить.

**Задача нескольких устройств** актуальна для группы исследователей, параллельно реализующих несколько устройств. Задача включает в себя передачу статистических данных на сервер (см. «Задача сбора статистики»), осложненную необходимостью идентификации конкретного устройства и более сложной моделью данных.

**Задача оптимальной траектории** предусматривает контроль траектории движения при совершении упражнений. Контроль может осуществляться как параметрически (например, по максимальным отклонениям относительно осей), так и с помощью заранее измеренной «идеальной» траектории.

**Работа с системой воспроизведения звука** рассматривается как вспомогательная задача оповещения: звуковое оповещение необходимо при старте и завершении упражнения, при отклонении от оптимальной траектории исполнения, а также может быть полезно для индикации совершаемых в ходе упражнения повторов.

**Работа с графическим дисплеем** в общем и целом решает задачи, обозначенные в предыдущем пункте: индикацию базовых моментов, как-то совершаемого упражнения, текущего подхода и повтора, времени, затраченного на упражнение.

**Задача выбора упражнения #2** описывает ситуацию, когда упражнение выбирается на стороне «клиента», то есть, на самой гантели с помощью кнопок, джойстика, etc.

**Задачи внешнего вида** и **физического размещения элементов** относятся к прототипированию и дизайну. Задача проектирования внешнего вида гантели сводится к 3D-моделированию основных компонентов гантели и их печати на принтере. По второй задаче отметим ряд очевидных моментов: компактное размещение контроллера, питания и датчиков в корпусе определяет ряд требований к структуре и размерам внутреннего пространства гантели.