Автономная некоммерческая общеобразовательная организация "Физтех-лицей"
(АНОО «Физтех-лицей» им. П.Л. Капицы)

XX научно-практическая конференция
«Старт в инновации»

Сборка робота-ходока и создание приложения для управления им

Выполнили: Климов Дмитрий, Труфанов Максим, 10 «В» класс

Руководители: Обухан Вячеслав Геннадьевич,
Зыонг Ким Куокович

Московская область, г. Долгопрудный
2021 г.

Содержание.

Введение ……..……..…….. 3

Основная часть:
1. С чего начать, собирая робота……...4
2. Как научить машину двигаться……..…...……6
3. Что надо знать про Qt……15
4. Как создать своё приложение ……..15

Заключение ……..……..18
Список использованной литературы……19

Приложение:
1. Схема сборки робота……..20

Введение.

Уже не одно десятилетие наше представление о будущем, о развитии науки и техники неразрывно связано с роботами, то есть с автоматизированными «умными» машинами, которые способны реализовывать человеческие функции при взаимодействии с миром, окружающим нас. В современном обществе - куда ни глянь - эти механизмы входят в повседневную жизнь с огромной скоростью для того, чтобы сделать её более комфортной, а условия труда – более приемлемыми, увеличить производительность, освободиться от затратных по времени и усилиям рабочих процессов. Сфера применения роботов ширится из года в год: если раньше чаще всего они были задействованы в промышленности, то теперь их можно встретить практически в любой области, будь то космос, медицина, военная отрасль, экология, сфера обслуживания, социальная сфера, индустрия развлечений и пр. В настоящее время роботам позволительно не только исполнять повторяющиеся примитивные задачи по программе, но и замахиваться на покорение новых вершин, что способствует взаимодействию с человеком, ведь общаясь на своем автоматизированном языке можно даже распознавать человеческие жесты и эмоции. Кроме того, это открывает огромные перспективы для всех желающих влиять на индустрию умных машин, создавать свои программы, приложения, совершенствовать робото-функции.

Именно поэтому мы сочли тему создания робота и приложения для управления им весьма актуальной и востребованной. Предпочтение мы отдали шагающему роботу (роботу-ходоку) – и это неспроста. Ведь если провести параллель с колёсными или гусеничными автоматизированными механизмами, то можно невооруженным взглядом увидеть, что шагающие «умные» машины по сравнению с ними имеют целый ряд преимуществ: они более мобильны внутри зданий и сооружений, где есть необходимость перемещаться, например, по лестницам и узким коридорам, или спускаться в подвалы и шахты. Выигрывают они и при движении по пересечённой местности, а также сложно-рельефной поверхности, на разломах и свалках.

Таким образом, целью нашей научно-исследовательской работы является создание актуального для самых разнообразных сфер рыночного продукта под условным названием «робот-ходок», который легко будет управляться с помощью созданного нами же универсального приложения в среде разработки Qt.

В рамках проекта мы изучали основы создания роботов, проектировали и собирали модель робота, «учили» его ходить, изучали технологии программирования микрочипов семейства Arduino, знакомились с уже существующими программами для шагающих роботов с целью анализа требований к программе и грамотной организации интерфейса, пробовали создать программу для чипа робота-ходока, изучали и реализовывали протокол RS232 для взаимодействия микрочипа с компьютером, оптимизировали код, знакомились с технологией Qt, создавали приложение с использованием этой технологии, изучали и реализовывали взаимодействие приложения и программы, то есть решали много непосредственных задач и проводили исследования в области информатики, программирования и робототехники.

Основная часть.

С чего начать, собирая робота? Заявленный нами шагающий робот (робот-ходок) представляет из себя двуногий механизм, который имитирует походку человека. Он создан из деталей, предоставленных кружком гуманоидной робототехники, работающим при нашем лицее под руководством Обухана Вячеслава Геннадьевича. Наш ходок функционирует на плате Arduino Nano.

На сегодняшний день выбор робототехнических платформ достаточно широк – здесь и BBC micro:bit, и Makeblock, и Raspberry Pi, и STM32, и Arduino. Почему же мы выбрали последний? Причин несколько:

- на данной платформе легко «сочинять» различные машины с управлением на расстоянии, элементарными сенсорами и простой логикой;
- имеет довольно несложный язык программирования и программный код;
- годится для разработки электронных устройств как новичками (наш случай), так и профессионалами (задел на будущее);
- широчайшее техническое обеспечение (существует множество совместимых с ней комплектующих – от простых кнопок и датчиков до плат расширения и жидкокристаллических экранов);
- информационная поддержка (большое количество статей в интернете, каналов на YouTube, готовых программных библиотек с кодами и алгоритмами в открытом доступе, наличие форумов с обсуждением любых возникающих вопросов);
- бесплатное приложение разработки Arduino IDE, которое подходит для различных операционных систем (Windows, Linux, Mac OS);
- поскольку язык Arduino представляет собой модифицированный C++, это позволяет любому, владеющему последним, легко освоить Arduino, а не владеющему - с помощью Arduino освоить C++;
- общедоступность.

Ещё в 2006 году Arduino получила признание в категории Digital Communities на фестивале Ars Electronica Prix. [1]

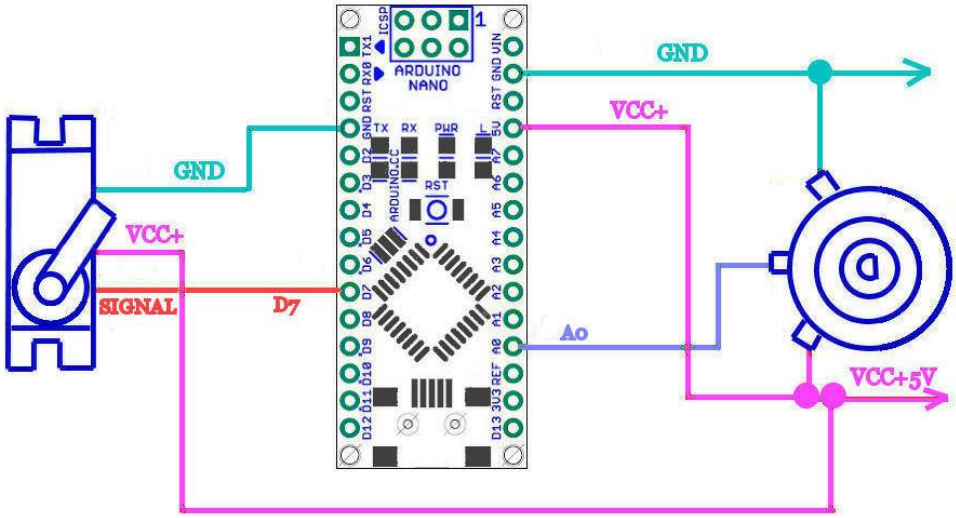
Наш робот-ходок передвигается с помощью сервоприводов. Чтобы задавать их положение, мы используем встроенную в Arduino библиотеку Servo. Следующий немаловажный вопрос – делать это в градусах или микросекундах? В нашем случае мы выбираем микросекунды по следующим причинам:

- ✓ используя градусную меру мы сумеем задавать значения лишь от 0 до 180. В случае же, когда мы зададим значение, большее 180, сервопривод отработает только 180. Аналогичная ситуация ожидает нас и с 0;
- ✓ рассмотрим пример: пусть рычаг системы довольно большой. Тогда даже перемещения всего лишь на один градус могут оказаться слишком резкими. С микросекундами всё проще: их действительное разрешение превосходит разрешение градусов более, чем в десять раз;
- ✓ в процессе обучения робота навыкам ходьбы не будет использоваться никаких тригонометрически-сложных расчётов, а значит необходимость использовать градусы просто-напросто отпадает.

Таким образом в приводимых ниже скетчах сервоприводы будут управляться посредством передачи продолжительности импульса в микросекундах. Определение крайних значений такой длительности требует выполнения нижеприведенного алгоритма действий:

1. сначала пронумеруем сервоприводы (которых у нас 6) от 1 до 6, прикрепив к ним бирки с соответствующими номерами;
2. затем присоединим к плате резистор, обладающий сопротивлением в диапазоне от 10 до 50 кОм, и сервопривод, управляющийся посредством этого самого резистора. Текущее значение длительности импульса выводится на монитор порта и, переводя сервопривод в крайние положения, мы получим искомые цифры.

Далее собираем представленную схему:



Управляющий вывод переменного резистора подключен к выводу A нулевому, а сервопривод подключен к цифровому выводу D7. Следующий шаг – загрузка нижеприведенного скетча:

```
 #include <Servo.h>//Подключение библиотеки Servo
int PtnPin = A0;//Вывод потенциометра
int SrvPin = 7;//Тестируемый серво
int MinTiming = 600;//Нижняя граница длительности
int MaxTiming = 2500;//Верхняя граница длительности
Servo TestServo;//Рабочая переменная типа Servo
void setup(){
  Serial.begin(9600);//Инициализация Serial-соединения
TestServo.attach(SrvPin,MinTiming,MaxTiming);
pinMode(PtnPin,INPUT);
  //Инициализация сервопривода
}
void loop(){
  int i = map(analogRead(PtnPin),0,1023,MinTiming,MaxTiming);
//Преобразование сигнала от потенциометра из 0..1023 в 300..3000
  TestServo.writeMicroseconds(i);
  //Перемещение сервопривода в положение соответствующее сигналу
  Serial.println(i);
  //Отправка текущего значение в монитор порта
  delay(100);//Задержка на выполнение перемещения
```

После этого переходим в “монитор порта” или используем комбинацию клавиш Ctrl+Shift+M. Появится окно, на которое будет выводиться текущее значение продолжительности импульса. Далее, просто перемещая движок, определяем крайние точки для каждого из шести сервоприводов и заносим их в таблицу:

№ сервы	min	max
1	800	2100
2	920	2160
3	600	2500
4	600	2500
5	655	2500
6	600	2250

В результате проделанной работы мы получили минимум и максимум продолжительности импульса и выяснили, что все наши сервоприводы могут быть переведены в крайние положения на участке 600-2500 микросекунд.

В рамках данного проекта мы решили не углубляться в процесс изготовления робота в техническо-конструкторском плане (всё же основную смысловую нагрузку в нашей работе несет на себе программное обеспечение), поэтому непосредственную сборку машины опишем кратко. Чтобы собрать робота, необходимо приобрести набор деталей и следующие инструменты: малую и среднюю крестообразные отвертки, круглогубцы и винты (как правило, идут в комплекте с набором запчастей). Сначала собирается основа робота (как это сделать, обычно указано в инструкции к деталям). Затем необходимо вставить сервомоторы, предварительно прикрутив в нужные места специальные качалки. Осталось соединить две платы (Arduino Nano и плату расширения Arduino Nano) и совместить с собранным роботом (наглядно всё это можно увидеть в Приложении №1).

Как научить машину двигаться. Вообще роботы-ходоки умеют функционировать в двух режимах:

- самостоятельное передвижение (автоматический режим), когда ходок передвигается произвольно и пытается обойти препятствия, встречающиеся ему на пути, путём разворота в одну или другую сторону. Если это становится невозможным, робот начинает идти назад, пока не покинет область контактирования с препятствием;

- заданное передвижение (исполнительный режим), когда ходок выполняет команды, поступающие с консольного приложения для Windows. Переводя сервоприводы в соответствии с командами приложения в указанное положение, робот тем самым сможет реализовывать различные модели поведения, заложенные разработчиками. Наш робот будет функционировать как раз в этом режиме.

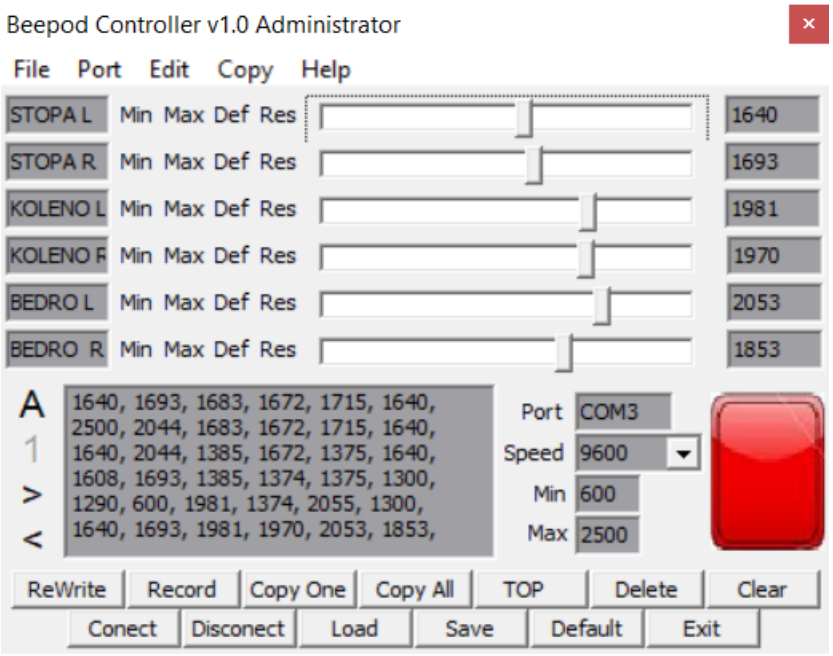
Сейчас мы приступаем к самой интересной части проекта – будем учить машину двигаться и превращать обычного робота в робота-ходока. Осуществлять это станем с помощью бесплатной и общедоступной утилиты Beepod Controller, которая существенно упрощает работу с положениями сервоприводов, позволяет сохранять и редактировать их комбинации в произвольный момент времени. Для успешного функционирования утилиты на робота-ходока должен быть загружен скетч, который будет перемещать все наши сервоприводы в заданное положение.

С помощью программы Arduino IDE загружаем в робота скетч через USB-кабель:

```
//Beepod_Model.ino
//Подключение к проекту библиотеки Servo
#include <Servo.h>
//Номера пинов с сервоприводами
int NumControl[] = {3,4,5,6,7,8};
//Исходное положение
int StartPos[] = {1640, 1693, 1683, 1672, 1715, 1640}
Servo Servos[6];//Массив сервоприводов
int APos[6];//Массив хранит предыдущие координаты сервоприводов для
//расчета дискретных шагов приращения для перехода на заданную позицию
int BPos[6];//Массив для передачи параметров новой позиции
int ServCount = 6;//Константа хранит количество сервоприводов
int MinTiming = 600;//Минимальная длительность импульса
int MaxTiming = 2500;//Максимальная длительность импульса
String St;//Глобальная строчная переменная для приема шифрованной
//пачки хранящей информацию о новой позици
//Функция производит плавный одновременный переход сервоприводов
//из текущей позиции в заданную параметром xPos
void MovePos(int xPos[]){
int Delta=0;//Переменная для хранения величины приращения
int MaxDelta=0;//Переменная для хранения максимальной величины приращения
//В этом цикле определяется максимальная величина приращения
for (int i=0; i<ServCount; i++){
//Абсолютное приращение i-того элемента
    Delta=abs(APos[i]-xPos[i]);
//если i-тое приращение больше текущего значения максимального
//Приравниваем i-тое максимальному
    if (MaxDelta < Delta) MaxDelta = Delta;
}
MaxDelta/=10;//Уменьшаем значение в 10 раз
//Этот цикл опредляет количество дискретных шагов, за которые
//будет выполнен полный переход из APos в xPos
for (int i=0; i<MaxDelta; i++)
//В данном цикле происходит проход по всем 6 сервоприводам в
//рамках действий одного дискретного шага
    for (int j=0; j<ServCount; j++){
//j-тый сервопривод перемещается на i-том шаге на
//величину равную (1/количество шагов)*общая величина приращения
      Servos[j].writeMicroseconds(map(i,0,MaxDelta,APos[j],xPos[j]));
//Пауза предназначенна для обеспечения плавного движения
//при совершении переходов с большой амплитудой
      delayMicroseconds(1000);
```

```
    }
//В этом цикле новая позиция сервоприводов станет предыдущей
//для следующего вызова функции
  for (int i=0; i<ServCount; i++)
    APos[i]=xPos[i];
}
//Начальная инициализация
void setup(){
//Инициализация сервоприводов
for (int i=0; i<ServCount; i++){
    Servos[i].attach(NumControl[i],MinTiming,MaxTiming);
//Вычисление и присвоение "предыдущей позиции"
    APos[i]=Servos[i].readMicroseconds();
}
//Эта скорость должна соответствовать установленной в программе
Serial.begin(9600);
//Переход в стартовую позицию
MovePos(StartPos);
}
void loop(){
  byte a=0;
  byte b=0;
//Если в буфере Serial-порта есть данные
while (Serial.available()){
//Чтение 1 символа из буфера
    char Ch=Serial.read();
//если не символ @ то добавляем символ к строке
//наличие сиввола @ означает конец шифрованного послания
    if (Ch!=64) St+=Ch;
    else{
//В этом цикле шифрованое послание расшифровывается и
//перемещается в BPos
      for(int i=0; i<ServCount; i++){
        if (St[i*2]<=90) a=St[i*2]-65; else a=St[i*2]-71;
        if (St[i*2+1]<=90) b=St[i*2+1]-65; else b=St[i*2+1]-71;
        BPos[i]=(a*52)+b+MinTiming;
      }
//Вызов перемещения в новую позицию
      MovePos(BPos);
//Очистка буфера приема строки
      St = "";
    }
```

В самом начале работы с утилитой требуется первоначальный запуск провести в административном режиме: реализуем это с помощью файла ADMIN.CMD. Окно будет иметь следующий вид:



Для настройки параметров подключения нам нужно ввести в соответствующее поле №порта нашей платы; проконтролировать, чтобы в поле «speed» цифры скорости подключения были равны 9600; ввести значения наших крайних положений сервоприводов в соответствующие поля (у нас это 600 и 2500); подключить ходока к USB-разъему и дважды кликнуть по красному индикатору (как вариант – нажать Connect, или Ctrl+N). Если всё было проделано верно, красный цвет должен смениться на зелёный.

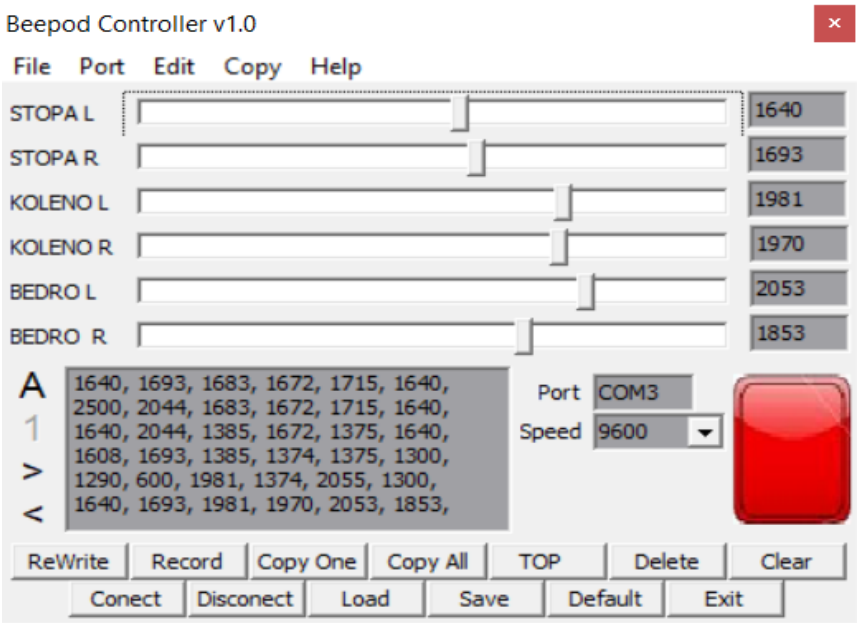
Далее необходимо сопоставить каждый сервопривод со своим окном: перемещаем рычажки в каждой из строк и называем это окно так, чтобы нам было понятно, какое из окон какому сервоприводу соответствует. Мы проименовали их как STOPA L, STOPA R (левая и правая стопа соответственно), KOLENO L, BEDRO L и т.д.

Теперь требуется, передвигая рычажки, задать роботу устойчивую начальную позу. После того, как мы зафиксировали положение всех 6 сервоприводов, в каждой строке необходимо нажать кнопку Def, чтобы установить данное положение сервопривода как начальное. Проверяем, что все значения сохранились: переводим каждый из движков в произвольное положение и в нижней строке нажимаем кнопку Default - робот должен принять исходную позицию.

Последним шагом необходимо настроить рабочие пределы самих сервоприводов. Для этого требуется двигать рычажок в первом окне влево до того момента, пока сервопривод не прекратит движение (фиксируем это кнопкой Min). Затем посредством кнопки Def возвращаем сервопривод в начальное положение и проводим аналогичные действия с правой стороной и кнопкой Max. Останется повторить те же действия с оставшимися пятью сервоприводами.

Осталось занести “реальную” исходную позицию в скетч, который мы загрузили выше: нажимаем кнопку Default, затем Record. В окне появится первая строка, состоящая из шести значений. Эти значения необходимо перенести и заменить на те, что изначально были заданы в нашем скетче. Проверяем: нажимаем кнопку Disconect и перезагружаем новый скетч в робота: после этого он должен занять начальную позицию.

Настройка робота завершена. Теперь, дабы избежать случайный нажатий важных кнопок, переходим из административного в обычный режим утилиты. Выглядит он так:



В поле, показанном на картинке ниже, хранятся значения, которые мы сохранили ранее с помощью кнопки Record. При двойном клике по строке в этом поле робот переводится в положение, соответствующее заданным положениям сервоприводов в этой строке.

Beepod Controller оснащена тринадцатью управляющими кнопками:

может сохранить и перенести в поле хранения значение, которое было установлено на компонентах

эта кнопка аналогична предыдущей в случае, когда в поле хранения ничего нет. В остальных случаях она осуществляет перезапись значений, выбранных в поле хранения

может копировать единственное значение, которое было выбрано из поля хранения

в отличие от предыдущей умеет копировать все значения

с помощью этой кнопки можно менять местами записи в поле хранения (для этого перед нажатием кнопки нужно выделить любое поле, и тогда оно переместится на позицию вверх)

эта кнопка служит для удаления записи в поле хранения (запись предварительно нужно выделить)

с её помощью можно очистить поле хранения

помогает открыть выбранный COM-порт

в противовес предыдущей помогает, наоборот, закрыть соединение с COM-портом

открывает каталог Models, чтобы можно было выбрать файл нужной модели поведения, сохраненной ранее

эта кнопка как раз позволяет сохранить файл с созданной моделью поведения в каталоге Models

помогает роботу занять первоначальную позицию

эта кнопка завершает программу

Теперь, когда мы полностью настроили программу Beepod Controller, можно приступать к написанию алгоритмов походки. Существует два варианта написания моделей поведения робота:

- полный цикл, содержащий стартовую позицию, несколько шагов, и возвращение в первоначальную позицию;
- написание каждого действия (первый шаг из исходной позиции, шаг левой ногой, шаг правой ногой, последний шаг и т.д.) в отдельном текстовом документе.

Мы будем использовать второй вариант, так как он, по нашему мнению, более практичный: робота можно будет остановить (и, например, заставить развернуться) в любой момент времени, что технически невозможно в первом варианте. Задавая роботу правильную походку, мы проводили тесты на облегчённой модели робота, собранной из обычного конструктора (фото 1). Когда мы наглядно увидели, что представляет из себя каждое движение робота, приступаем к работе с Beepod Controller.

Для начала выставляем значения импульсов каждого из сервоприводов так, чтобы робот перевалился на правую ногу (первый шаг делается левой ногой), при этом левая нога должна помогать роботу перевалиться на правую (фото 2). Нажимаем кнопку Record. Первое действие готово. Далее необходимо выровнять стопы и отклонить бедра и колени двух ног на равные углы в разные стороны (фото 3).

Фото 1

Фото 2

Фото 3

Равные углы легко рассчитать математически: например, если значения импульсов у коленных сервомоторов равны 1200, и мы отклоняем левый сервомотор на 1200+200=1400, то правому сервомотору необходимо задать положение 1200-200=1000. Таким образом, разница между полученными импульсами колен в каждом таком положении должна составлять двойной импульс отклонения (в данном примере 200, назовем его так). Аналогичная ситуация и с бёдрами. Нажимаем кнопку Record. Первый шаг из начального положения готов. Сохраняем три строки (начальное положение, перевал, шаг) в текстовом документе. Теперь делаем второе действие - шаг правой ногой. Аналогично первому действию: переваливаем робота на левую ногу и присваиваем импульсам сервомоторов левой ноги значения импульсов сервомоторов правой ноги и наоборот. Сохраняем две строки в текстовом документе. Аналогично строятся действия для шага левой ногой не из начального положения и возвращения в начальное положение с левой и с правой ноги (еще 3 документа). Движения назад строятся обратно движению вперед: все строки каждого действия переставляются задом наперёд, то есть если было 4 строки, то 4-ая строка становится 1-ой, 3-я становится 2-ой и т.д.

Затем строим алгоритмы поворота. Нога, противоположная стороне разворота, ставится на пятку, затем робот возвращается в исходное положение. Таким образом, если зациклить это действие, робот будет каждый раз поворачиваться на определенный градус (зависит от сцепления стопы с поверхностью и высоты подъема стопы). Всё просто! Также сохраняем два файла (разворот вправо и разворот влево), состоящих из двух строк.

Остаётся лишь создать массивы из этих файлов и внести их в код Arduino. Создаём новый файл. Пользуясь ранее сохраненными текстовыми документами с действиями робота, создаём массивы с полученными значениями импульсов:

```
#include <Servo.h>                          // Подключение библиотеки Servo

int ServCount = 6;                          // Количество сервоприводов
int MinTiming = 600;                        // Рабочий предел сервоприводов (минимум)
int MaxTiming = 2500;                       // Рабочий предел сервоприводов (максимум)
int StartTimeOut = 3000;                    // Длительность паузы после включения
int ForwardFirstStepCount = 3;              // Количество шагов в модели ForwardFirst
int ForwardRightStepCount = 2;              // Количество шагов в модели ForwardRight
int ForwardLeftStepCount = 2;               // Количество шагов в модели ForwardLeft
int ForwardLastRighttStepCount = 2;         // Количество шагов в модели ForwardLastRight
int ForwardLastLeftStepCount = 2;           // Количество шагов в модели ForwardLastLeft
//int ReversStepCount = 6;                  // Количество шагов в модели Revers
int TurnRightStepCount = 3;                 // Количество шагов в модели TurnRight
int TurnLeftStepCount = 3;                  // Количество шагов в модели TurnLeft
int Entrop = 10;                            // Максимальный предел энтропийных флюктуаций
// Номера серв      1 2 3 4 5 6
int NumControl[] = {3,4,5,6,7,8};           // Массив номеров выводов Ардуино (D3-D8) с сервоприводами

Servo Servos[6];                            // 6 переменных Servos[i] типа Servo для управления сервоприводами
int APos[6];                                // Массив хранит текущую позицию APos[i]

int StartPos[] = {1434, 1509, 1488, 1482, 1516, 1413, 500}; // Координаты исходной позиции сервоприводов
int StepForwardFirst[] = {                  // Массив модели поведения "Первый шаг левой" ForwardFirst
1434, 1509, 1488, 1482, 1516, 1413, 1000,
2172, 1776, 1488, 1482, 1516, 1413, 1000,
1434, 1509, 1078, 1072, 1140, 1037, 1000};
int StepForwardRight[] = {                  // Массив модели поведения "Шаг вперед правой" ForwardRight
1119, 600, 1078, 1072, 1140, 1037, 1000,
1434, 1509, 1898, 1892, 1892, 1789, 1000};
int StepForwardLeft[] = {                   // Массив модели поведения "Шаг вперед левой" ForwardLeft
2213, 1844, 1898, 1892, 1892, 1789, 1000,
1434, 1509, 1078, 1072, 1140, 1037, 1000};
int StepForwardLastRight[] = {              // Массив модели поведения "Последний шаг правой" ForwardLastRight
1119, 600, 1078, 1072, 1140, 1037, 1000,
1434, 1509, 1488, 1482, 1516, 1413, 1000};
int StepForwardLastLeft[] = {               // Массив модели поведения "Последний шаг левой" ForwardLastLeft
2213, 1844, 1898, 1892, 1892, 1789, 1000,
1434, 1509, 1488, 1482, 1516, 1413, 1000};

//int StepRevers[] = {                      // Модель поведения revers
//};

int TurnRight[] = {                         // Модель поведения "Поворот на право" TurnRight
1750, 1150, 1150, 1800, 2000, 1700, 500,
1750, 1150, 1150, 2500, 1900, 1500, 300,
1750, 1150, 1150, 1800, 1700, 1250, 500};
int TurnLeft[] = {                          // Модель поведения "Поворот на лево" TurnLeft
1750, 1150, 1150, 1800, 1200, 950, 500,
1750, 1150, 500, 1800, 1450, 1050, 300,
1750, 1150, 1150, 1800, 1700, 1250, 500};


void MovePos(int xPos[]){                   // ФУНКЦИЯ производит плавный одновременный переход
                                  //сервоприводов из текущей позиции APos[i] в заданную параметром xPos[i]
int Delta=0;                                // Переменная для хранения величины изменения положения сервы
int MaxDelta=0;                             // Переменная для хранения максимальной величины изменения
unsigned long a=0;
unsigned long b=0;
unsigned long TimeIter=0;                   // Переменная для итерации таймера
unsigned long DeltaTime=0;                  // Переменная для хранения времени выполнения кода
unsigned long TimePoint=0;                  // Переменная для хранения временной точки начала отчета

for (int i=0; i<ServCount; i++){            // Цикл проходит по всем 6 сервам
    Delta=abs(APos[i]-xPos[i]);             // Вычисляется величина изменения между APos[i] и xPos[i]
    if (MaxDelta < Delta) MaxDelta = Delta;    // Вычисляется MaxDelta
}
```

```
MaxDelta/=10;

for (int i=0; i<MaxDelta; i++)            // Цикл с количеством итераций равным MaxDelta

    for (int j=0; j<ServCount; j++){      // Цикл для последовательного изменения на каждой из 6 серв
      TimePoint = micros();               // Сохранение временной точки
      Servos[j].writeMicroseconds(map(i,0,MaxDelta,APos[j],xPos[j]));   // Перемещение j-того сервопривода на i-том шаге
                                  //  на величину равную 1/MaxDelta*величину общего приращения для j-того сервопривода
      a=xPos[ServCount]*1000.0;
      b=MaxDelta*ServCount;
      TimeIter=a/b;             // Вычисление необходимой длины паузы
      // Вычисление времени затраченного на выполнение кода от
      // сохранения TimePoint до этого момента
      DeltaTime=micros()-TimePoint;
      // Если итерация выполнена за время меньше чем необходимая пауза - делаем паузу
      if (DeltaTime<TimeIter) delayMicroseconds(TimeIter-DeltaTime);
    }

      //В этом цикле новое положение перемещают в массив APos для
      //последующего использования в качестве «прошлого» значения
for (int i=0; i<ServCount; i++) APos[i]=xPos[i];
}
void PlaySteps(int arSteps[], int StepCount){   //ФУНКЦИЯ производит «проигрывание» моделей поведения.
                            //Ей сообщается массив хранящий массив модели и число шагов.
  int arTMP[ServCount+1];                 // В этом массиве хранится 1 шаг модели
for(int i=0; i<StepCount; i++){           // Цикл с числом итераций равным числу шагов модели
    for(int j=0; j<ServCount+1; j++)      // Цикл с 6 итерациями
                                          //В массив переносится i-тый шаг модели
      arTMP[j]=arSteps[j+(ServCount+1)*i]+random(-Entrop, Entrop);
    MovePos(arTMP);                       // Перемещение на i-тый шаг модели
  }
}


void setup() {

  for (int i=0; i<ServCount; i++){          // Цикл с 6 итерациями
    Servos[i].attach(NumControl[i],MinTiming,MaxTiming);  // Инициализируются сервоприводы
    APos[i]=Servos[i].readMicroseconds(); //текущее значение сервоприводов сохраняется в качестве «прошлого» значения
  }
  MovePos(StartPos);                      //Переход в исходную позицию
  delay(StartTimeOut);                    //Стартовая пауза
}
```

Остаётся последняя функция void loop, с помощью которой, используя созданные массивы, мы можем написать абсолютно любой алгоритм походки робота. В нашем случае мы приводим простой пример с походкой вперёд:

```
void loop() {

   PlaySteps(StepForwardFirst,ForwardFirstStepCount);       // 1й Шаг
   PlaySteps(StepForwardLastRight,ForwardLastRighttStepCount);
   PlaySteps(StepForwardFirst,ForwardFirstStepCount);       // 1й Шаг
   PlaySteps(StepForwardRight,ForwardRightStepCount);       // Шаг правой
   PlaySteps(StepForwardLeft,ForwardLeftStepCount);         // Шаг левой
   PlaySteps(StepForwardLastRight,ForwardLastRighttStepCount);  // Последний шаг правой
   PlaySteps(StepForwardFirst,ForwardFirstStepCount);       // 1й Шаг
   for(int k=0;k<3;k++){
   PlaySteps(StepForwardRight,ForwardRightStepCount);       // Шаг правой
   PlaySteps(StepForwardLeft,ForwardLeftStepCount);         // Шаг левой
   }
   PlaySteps(StepForwardLastRight,ForwardLastRighttStepCount);  // Последний шаг правой
   delay(5000);
}
```

Таким образом, усовершенствуя старые и создавая новые алгоритмы движений, мы можем заставить робота выполнять определённый набор действий, например, пройти лабиринт. Для обучения машины новым движениям мы создали простую программу, о которой расскажем далее.

Что надо знать про Qt. Qt - это платформа, которая позволяет создавать программное обеспечение для различных операционных систем. Однако есть специальные дополнения, с помощью которых можно писать на других языках программирования. В Qt были созданы такие популярные проекты как Opera, Skype, Google Earth, Adobe Photoshop Album, VirtualBox, Zoom. В списке пользователей Qt такие крупные компании как: Adobe, Amazon, Bosch, BMW, Canon, Disney, Intel, Panasonic, HP, Google, Mercedes, NASA, Nokia, Samsung, Sony, Tesla Yamaha и другие. Qt поддерживается на таких операционных системах как: Microsoft Windows, Mac OS X, Linux, FreeBSD, а также и для мобильных ОС iOS, Android, Windows Phone, Windows RT.

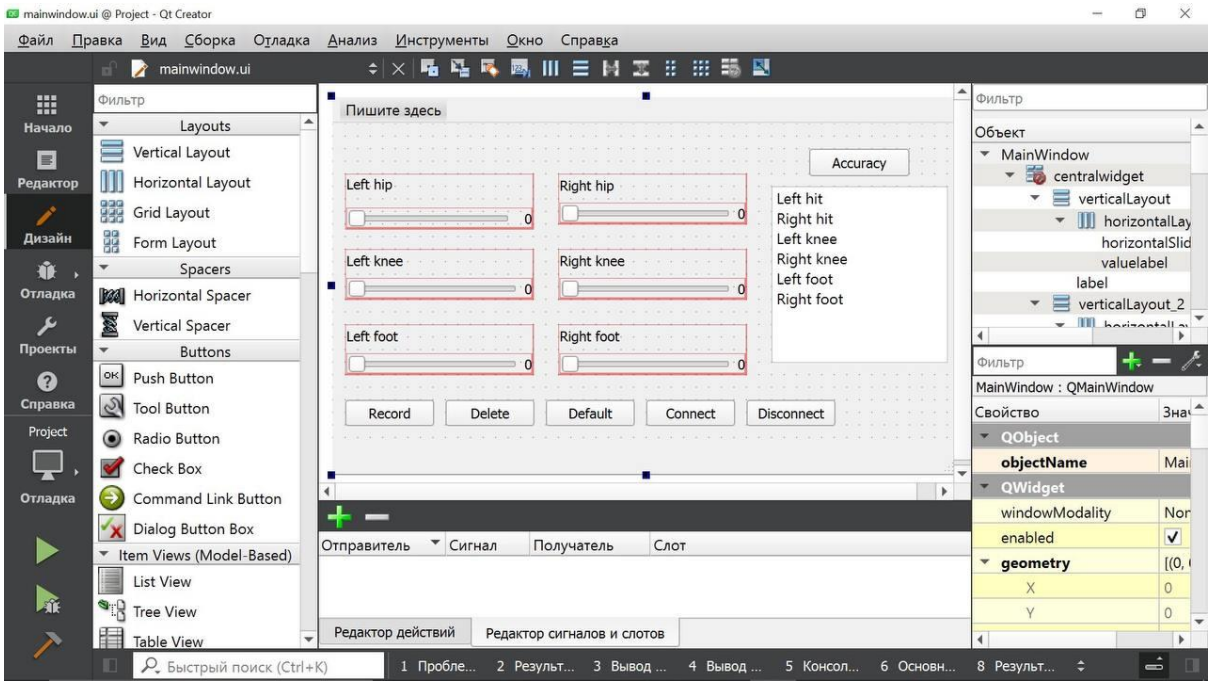
Отличительная её особенность - использование части ядра программной платформы (фреймворка) для поддержки C++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени. Для более быстрого и простого создания пользовательских интерфейсов Qt предоставляет программу Qt Designer. В среде разработки Qt предоставлена возможность непосредственно в ней создавать и редактировать визуальную часть проекта. Большой список фильтров поможет сделать работу приятной для восприятия. Также большой плюс в удобстве ручного редактирования, добавления стилей и анимации. Использование в разработке разных компиляторов C++ еще больше повышает корректность и надежность кода программ.

Qt – это не только средство для создания интерфейса, но и полный комплект для программирования, состоящий из отдельных модулей и предоставляющий поддержку двумерной и трехмерной графики, стандартных протоколов ввода/вывода, классы для работы с сетью, библиотеку контейнеров, поддержку программирования баз данных, включая Oracle, Microsoft SQL, Sybase. В Qt есть особая система расширения, позволяющая создавать модули, которые расширяют функционал. При этом пользователи смогут получить эти расширения не только от создателя, но и от других разработчиков.

Как было сказано ранее, библиотека Qt изначально создавалась для языка программирования C++, но в других языках программирования есть модули, для полноценной работы с данной библиотекой. Еще одна фишка - возможность узнать кое-что новое, используя программу Qt Assistant. При этом, не стоит забывать, что Qt - библиотека с открытым исходным кодом и абсолютно каждый может детальнее ознакомится, как работает та или иная часть библиотеки.

Как создать своё приложение. Имея навыки работы с Qt, можно попробовать создать своё приложение, чтобы максимально облегчить настройку и использование робота. Как это сделать? При запуске приложения, пользователь увидит шесть ползунков, каждый из которых будет отвечать за показания определенного сервомотора. При необходимости в приложение можно с легкостью добавить еще ползунки, если робот состоит из большего числа сервомоторов. Помимо этого для простоты и удобства в приложении есть панель, отвечающая за точность заданных параметров, то есть пользователь может ввести показатели как вручную, так и добиться желаемого результата с помощью переключения ползунка между режимами.

Также стоит отметить простоту кода. Изначально код был для одного ползунка, а впоследствии дописан для остальных пяти - из этого можно сделать вывод, что для более продвинутых роботов достаточно понять, как работает код для одного-единственного ползунка, чтобы потом легко дописать его для других, соответствующих новым сервомоторам. Максимально простой дизайн позволяет пользователю с первого запуска приложения быстро понять, как настроить своего робота.

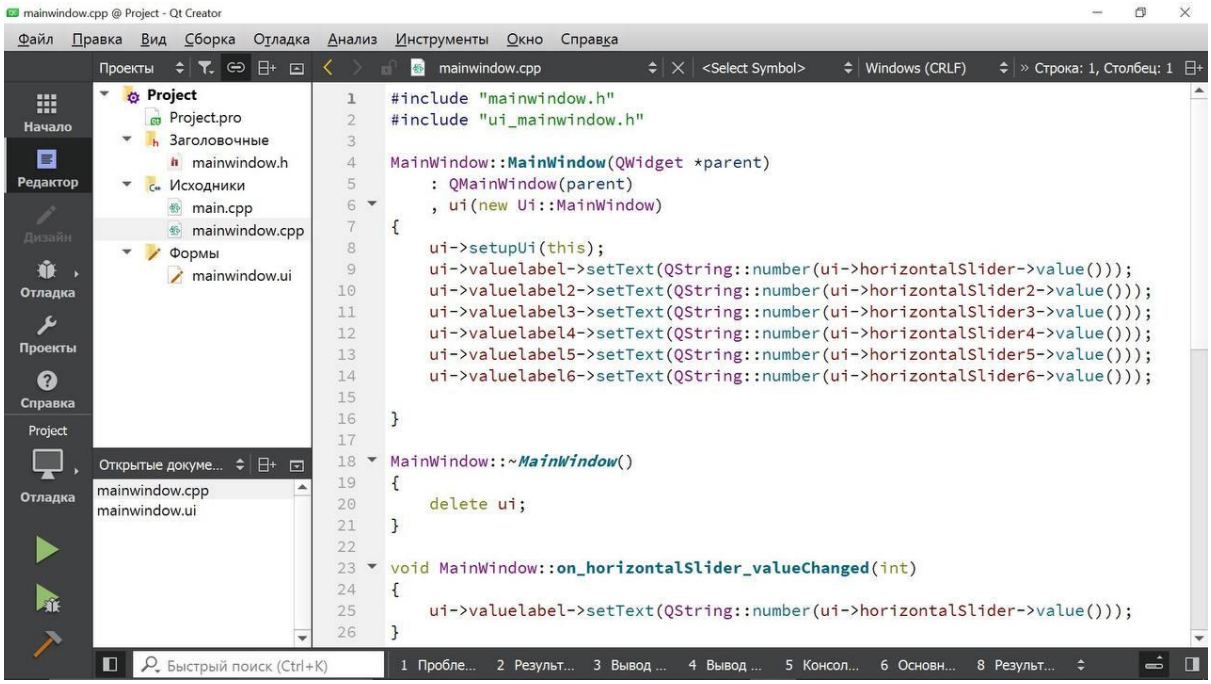


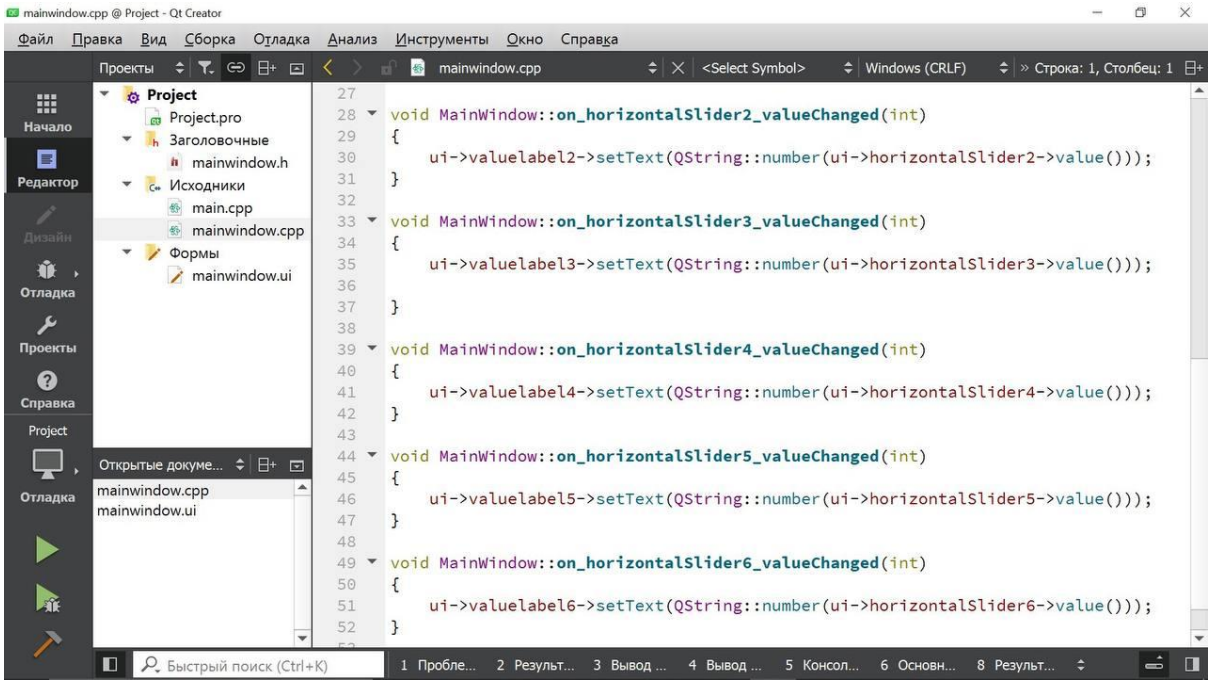
Теперь о написании кода для первого ползунка. Заходим в режим “Дизайн”, на рабочую панель перетаскиваем виджет “Horizontal Slider” и “Label”. Изначально нужно прописать, чтобы в созданной нами строке, были показания ползунка в момент запуска. Для этого, переходим в режим “Редактор” и в файле исходнике прописываем

```
ui->valuelabel->setText(QString::number(ui->horizontalSlider->value()));.
```

Теперь необходимо, чтобы в строку передавались текущие значения нашего ползунка. Переходим к слоту ползунка, выбираем сигнал “Value Changed” и прописываем точно такой же код. Теперь при запуске приложения, изменяя положение ползунка, в нашей строке будут текущие показатели ползунка. Для остальных пяти прописываем то же самое.

Остается прописать кнопки “Delete”, “Record”, “Default”, “Connect”, “Disconnect”, “Accuracy” каждая из которых, соответственно, отвечает за очищение выбранной строки от значений, запись значений в выбранную строку, выставление значений по умолчанию, подключение, отключение, выбор точности. Теперь добавим кнопку, которая, собственно, и будет делать наше приложение, отличным от других. Кнопка “Add”. При ее нажатии пользователь должен написать название. Создастся дополнительный ползунок, и в общей таблице добавится новое название.





В итоге мы создали приложение, которое удобно в использовании и может быть применено для роботов с различным количеством сервомоторов.

Заключение.

Очевидно, что индустрия «умных» машин сегодня и в ближайшее время - это очень перспективное направление развития науки и техники. Кроме того, разработка и создание роботов – занятие весьма увлекательное, многогранное и, что немаловажно, доступное каждому (даже школьнику).

Пройдя этот путь с нуля, мы рекомендовали бы новичкам начать изучение основ с робототехнической платформы Arduino (подробные доводы мы приводили в тексте работы). Ну а для того, кто хочет продвинуться дальше – изучить программы Qt Designer, Qt Creator и попробовать создать своё приложение, помогающее в управлении роботом.

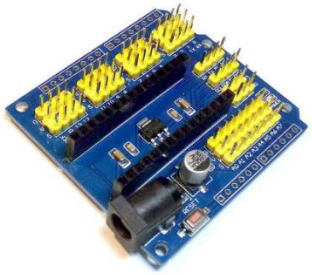
В рамках реализации нашего проекта нам удалось осуществить поставленные задачи и достичь запланированной цели. Мы изучили разнообразные технические ресурсы, научились собирать робота, смогли научить его двигаться, написали своё приложение, чтобы максимально облегчить настройку и использование шагающей машины, и протестировали его на готовом продукте.

Список использованной литературы.

1. Бейктал Дж. Конструируем роботов на Arduino. Первые шаги.- Издательство «Лаборатория знаний», 2019
2. Боровский А. Qt4.7+. Практическое программирование на C++.- Издательство «BHV», 2012
3. Гололобов В.Н. Arduino для любознательных, Или паровозик из Ромашкова.- Издательство «Наука и техника», 2017
4. Научная электронная библиотека «КиберЛенинка» - https://cyberleninka.ru - (открытый доступ)
5. Фокин В. Г., Шаныгин С.В. Обзор и перспективы развития мобильных шагающих робототехнических систем. // Молодой ученый – 2015 - № 18(98)
6. Шлее М. Qt 5.10. Профессиональное программирование на C++.- Издательство «BHV», 2018

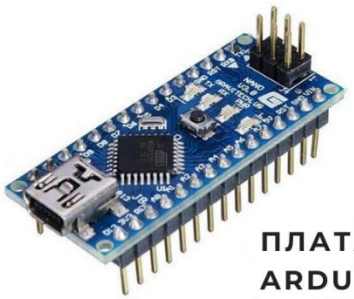
Приложение 1

СБОРКА РОБОТА-ХОДОКА



ПЛАТА РАСШИРЕНИЯ ARDUINO NANO

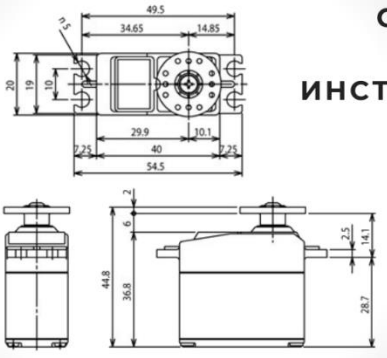
СЕРВОМОТОР FS5106 FEETECH С КОМПЛЕКТУЮЩИМИ



ПЛАТА ARDUINO NANO

АЛЮМИНИЕВЫЕ КРОНШТЕЙНЫ ДЛЯ СЕРВОМОТОРОВ

СХЕМА



ИНСТРУКЦИЯ