

Конструктор-робот «Печенег Батана»



Конструкторы ЛАРТ предназначены для изучения школьниками в возрасте от 12-14 лет основ электроники, программирования, конструирования.

Конструктор «**Печенег**» подходит как для новичков, так и для опытных любителей конструирования.

Школьникам, которые уже познакомились с контроллером ARDUINO, будет интересно продолжить изучение принципов программирования, взаимодействие датчиков и исполнительных устройств и механизмов.

Набор состоит из блока управления **LART R-5** с контроллером Arduino Nano, датчиков, электромоторов, деталей металлического каркаса и крепежных элементов.

Детали шасси изготовлены из листового металла толщиной 0,8 мм и способны нести достаточно высокую механическую нагрузку. Размер штатных крепежных отверстий составляет 3,5 мм. На пластине размещены дополнительные отверстия диаметром 4,5 мм и шагом 10 мм. Контроллер Arduino прост в освоении программирования. Не требует платных лицензий.

Подросток без помощи профессиональных инженеров и программистов сможет самостоятельно собирать и программировать роботов. От простых моделей роботов, следующих по заданной траектории до сложных кибернетических систем.

Особенности конструктора

Ардуино - популярный контроллер. В русском и англоязычном сегментах интернета присутствует большое количество учебных материалов, примеров программ в свободном обращении. Это позволяет достаточно быстро осваивать работу с контроллером, программирование.

Так же в конструкторе возможно использование недорогих Arduino-совместимых датчиков, модулей связи (Bluetooth, Wi-Fi), механизмов.

Шасси робота.

Применение в качестве несущей шасси робота металлической пластины с отверстиями 4,5 мм и шагом 10 мм позволяет в качестве строительных компонентов использовать линейки, пластины, крепежные элементы и др. от обычных металлических конструкторов.

Состав конструктора

Пластина металлическая несущая	- 1 шт.
Пластина металлическая дополнительная	- 1 шт.
Пластина акриловая	- 1 шт.
Уголок крепления мотора правый	- 2 шт.
Уголок крепления мотора левый	- 2 шт.
Мотор с редуктором T-130 (1:48, 250 rpm)	- 4 шт.
Пластиковое колесо 65 мм	- 4 шт.
Шариковая опора	- 1 шт.
Сервомотор SG90	- 1 шт.
Блок управления R-5	- 1 шт.
Контроллер Arduino Nano	- 1 шт.
USB кабель	- 1 шт.
УЗ датчик HC-SR04	- 1 шт.
Модуль «пищалка»	- 1 шт.
Модуль «светодиод»	- 2 шт.
Модуль «кнопка»	- 1 шт.
Датчик линии	- 2 шт.
Микрофонный датчик	- 1 шт.
Модуль «энкодер»	- 2 шт.
Пластиковое колесо энкодера	- 2 шт.
Светодиод красный 5 мм	- 10 шт.
Светодиод желтый 5 мм	- 2 шт.
Светодиод зеленый 5 мм	- 2 шт.
Светодиод RGB	- 1 шт.
Резистор 0,5 Вт 360 Ом	- 10 шт.
Резистор 0,5 Вт 1 кОМ	- 4 шт.
Резистор 0,5 Вт 10 кОМ	- 4 шт.
Резистор 0,5 Вт 30 кОМ	- 4 шт.
Фоторезистор	- 1 шт.
Батарейный отсек на 6 бат. АА	- 1 шт.
Ножка резиновая	- 4 шт.
Макетная плата	- 1 шт.
Пластиковый держатель УЗ датчика	- 1 шт.
Винт М3 х 6	- 40 шт.
Винт М2 х 4	- 2 шт.
Винт М3 х 24	- 8 шт.
Гайка М3	- 10 шт.
Гайка М2	- 2 шт.
Стойка мет. 10 мм.	- 8 шт.
Стойка мет. 25 мм.	- 4 шт.
Стойка мет. 35 мм.	- 2 шт.
Стойка мет. 40 мм	- 4 шт.
Комплект соединительных проводов	- 1 к-т.
CD диск	- 1 шт.

Arduino. Установка программного обеспечения.

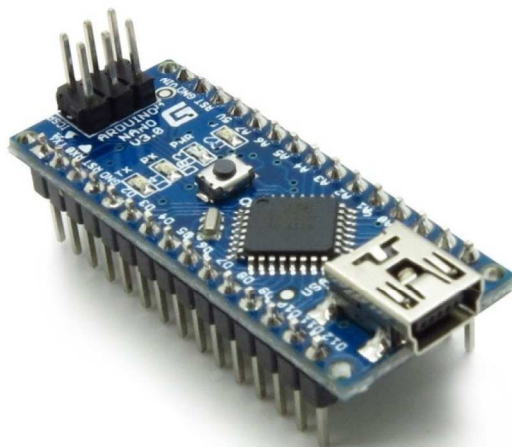
Arduino - это контроллер (управляющий модуль). На плате размещены процессор, микросхема конвертора USB и выводы для подключения к контроллеру внешних устройств. В качестве внешних устройств могут различные исполнительные устройства и элементы: светодиоды, реле, электромоторы.

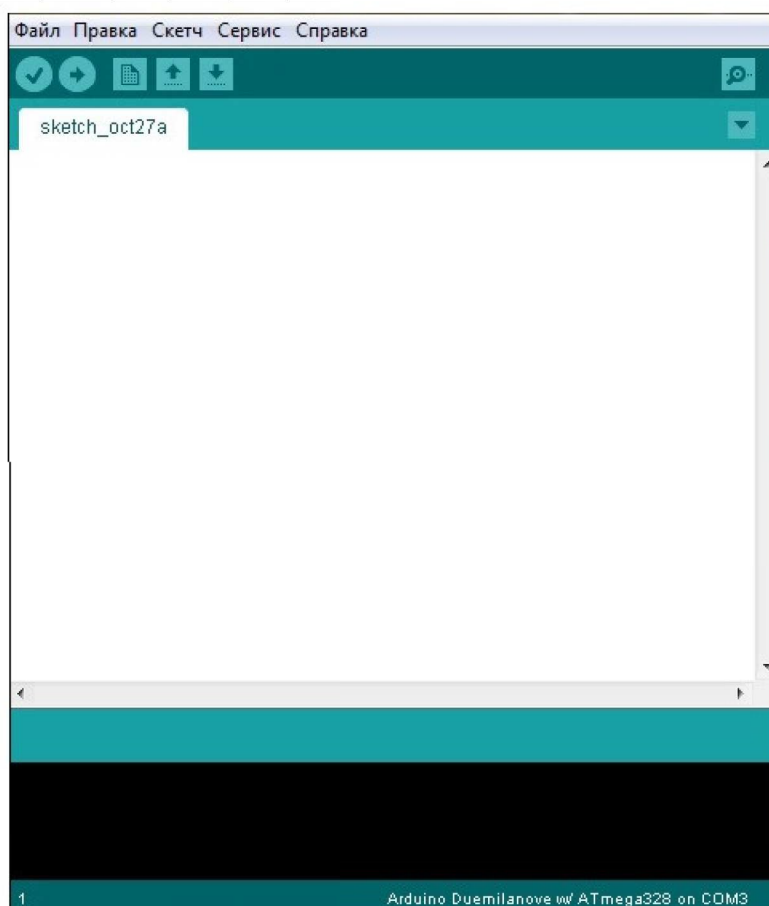
Порядок управления (алгоритм) этими устройствами определяется программой, написанной программистом для контроллера Arduino и загруженной в память процессора.

Для написания, редактирования, отладки программ создана специализированная среда – IDE. Программа пишется на языке C++. Для загрузки написанной программы в процессор не требуется каких-либо дополнительных устройств. Надо просто подключить плату Arduino к компьютеру USB кабелем.

Платформа Arduino.

Arduino состоит из двух основных частей – непосредственно платы Arduino , которая является частью аппаратного обеспечения, над которым вы работаете при создании собственных объектов. И среды разработки IDE - программного обеспечения, которое вы запускаете на своём компьютере. IDE необходимо для создания скетчей (маленьких компьютерных программ), которые затем выгружаются на плату Arduino. Скетч «говорит» плате, что делать.





Аппаратное обеспечение Arduino.

Плата Arduino - небольшая плата микроконтроллера, состоящая из небольшой схемы, содержащей целый компьютер в маленьком чипе (микроконтроллер). Существует много вариантов этой платы. Но наши проекты мы будем релизовывать на плате Arduino Nano. Так как на наш взгляд это самый удобный вариант для наших занятий.

На рисунке представлена плата Arduino Nano, с которой мы будем работать.

Поясним назначение контактов платы.

Контакты 0-13 - контактов цифрового ввода-вывода.

Они могут быть как входами, так и выходами, что определяется вашей программой (скетчем).

Контакты 3, 5, 6, 9, 10 и 11 могут использоваться, как ШИМ – широтно-импульсная модуляция.) Контакты 0–5 - контакты аналогового входа.

Эти контакты для аналогового входа получают аналоговые значения (например, величину напряжение в датчике) и преобразовывают их в цифровые значения от 0 до 1023.

Электрическое питание может быть подано на плату через USB-порт от компьютера, или от AC-адаптера 9 вольт через разъем питания. Плата может сама выбирать источник питания. Т.е., если плата подключена к компьютеру через USB кабель и к разъему питания подключен адаптер, то плата выберет адаптер.

Интегрированная среда разработки (IDE).

IDE (интегрированная среда разработки) - это специальная программа, работающая на вашем компьютере, которая позволяет писать управляющую программу (скетч) для платы Arduino на простом языке. После нажатия кнопки «загрузить»



код, который вы написали, будет транслирован в команды, понятные микроконтроллеру и загружен в память.

Процесс программирования Arduino упрощённо выглядит следующим образом:

- подключаем плату Ардуино к USB-порту компьютера.
- пишем скетч, для работы платы.
- выгружаем этот скетч на плату через USB-соединение и ждём несколько секунд для перезапуска платы.

Плата приступит к выполнению написанного вами скетча.

Установка Arduino на ваш компьютер.

Чтобы начать работать с платой Arduino, сначала необходимо установить среду разработки (IDE).

Скачать и установить среду IDE можно с сайта разработчика:

www.arduino.cc/en/Main/Software.

Удобнее сразу запустить установщик:

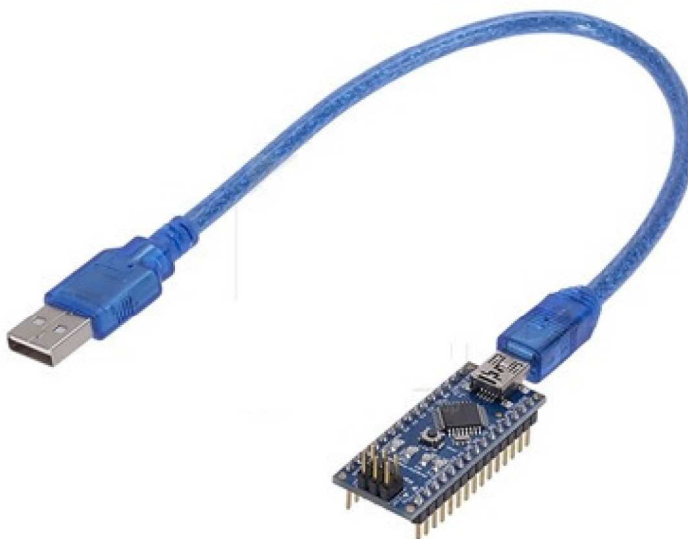


После установки ПО запустите IDE.



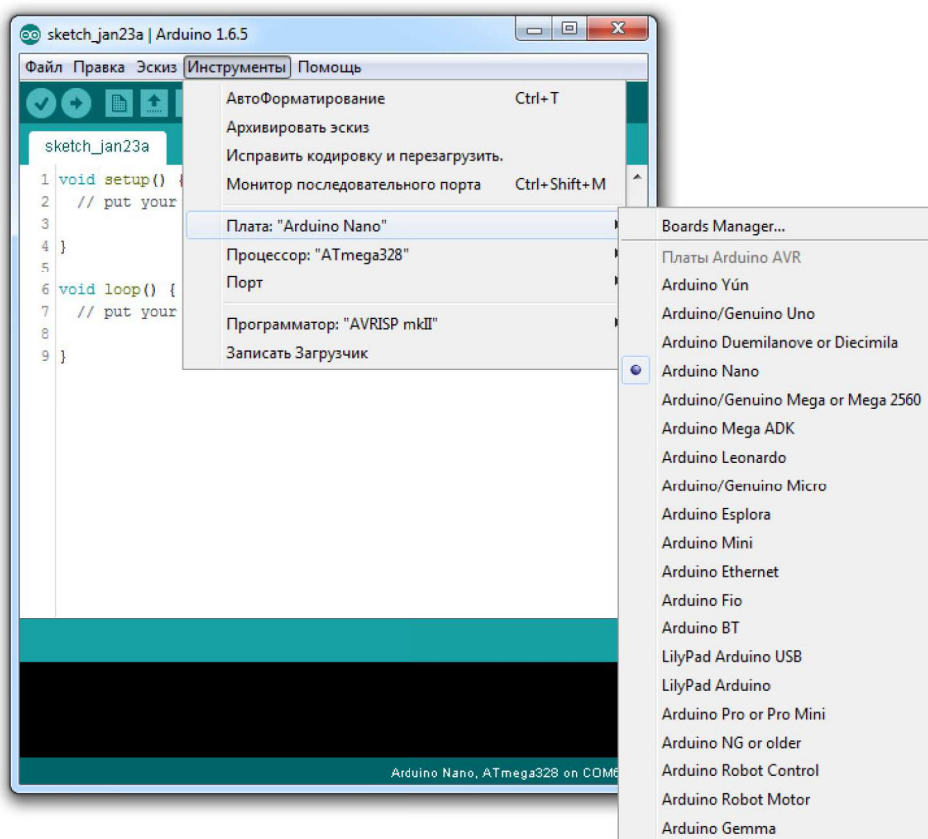
На экране монитора у вас появится окно среды проектирования IDE.

Подключите USB кабелем контроллер Arduino Nano к компьютеру.



Последовательно нажмите кнопки ИСТРУМЕНТЫ, ПЛАТА, ARDUINO Nano. Далее необходимо выбрать процессор ATmega328 порт, который предложит система. Теперь плата готова к работе.

Выбор платы ARDUINO NANO

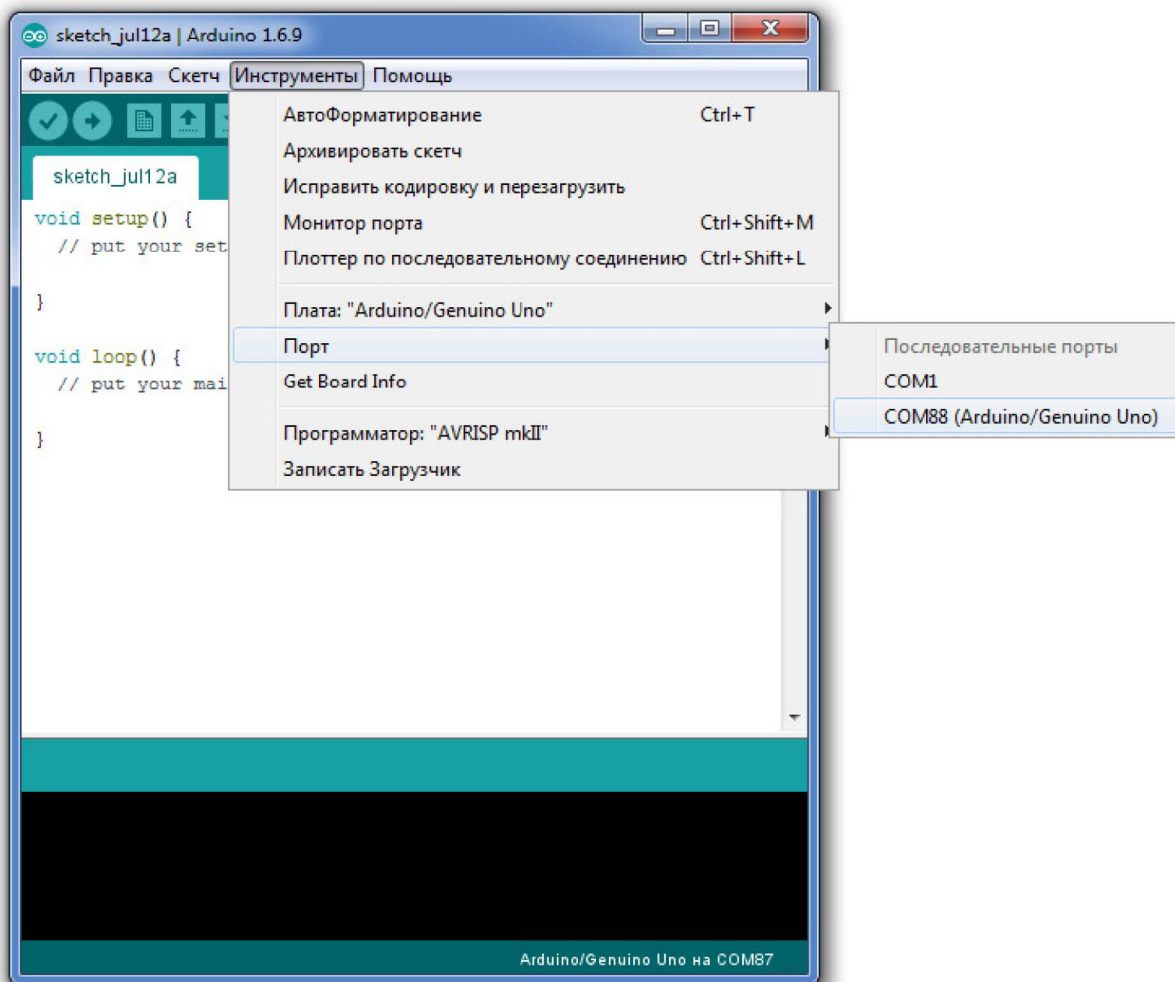


Выбор порта

Далее необходимо выбрать порт, по которому плата Ардуино будет общаться с компьютером.

Для этого так же последовательно нажимаем кнопки Инструменты -> Порт -> COM...

Ком-порт, который предложит компьютер.



Если эти настройки завершили удачно, то плата Ардуино готова к работе.

Система готова к работе.

Теперь проведем небольшой ознакомительный эксперимент.

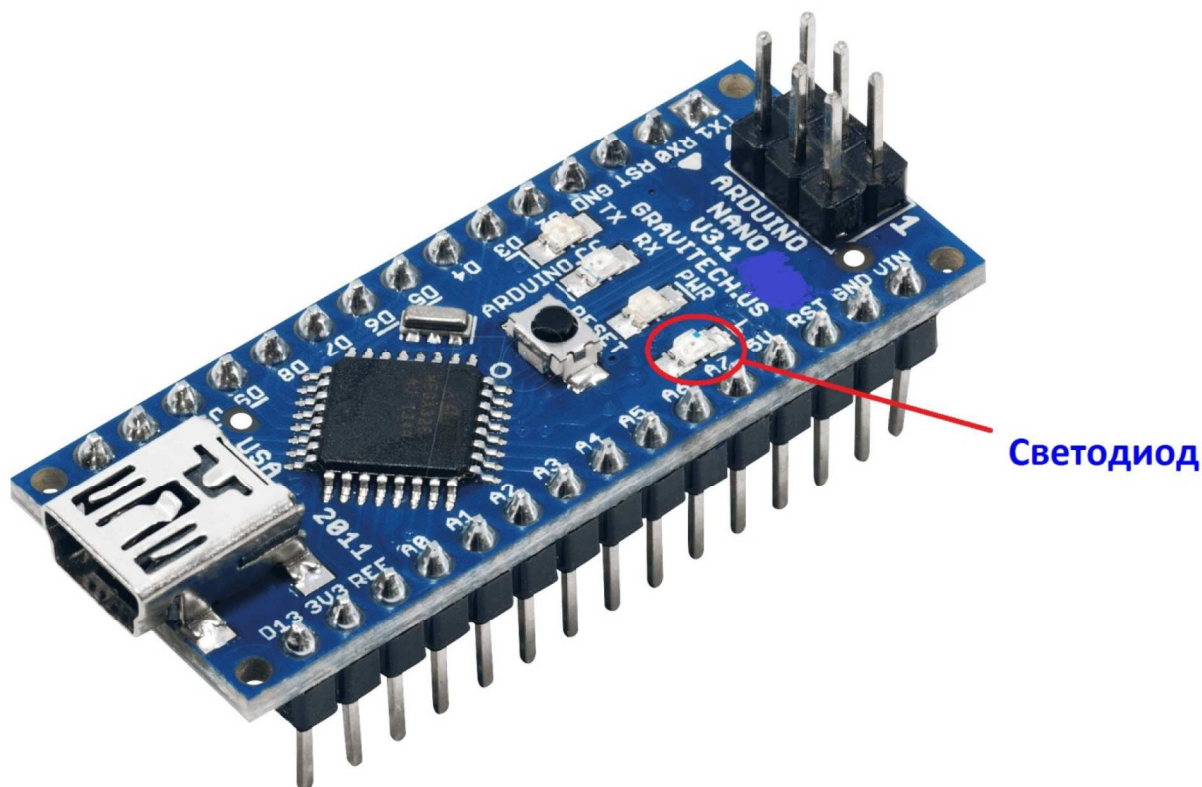
Мигание светодиодом.

Задачи эксперимента. Знакомство со средой разработки, проверкой работоспособности программы и контроллера Arduino Nano.

Скетч, который управляет миганием светодиода - первая программа, которую мы загрузим в плату Arduino.

Светоизлучающий диод (светодиод) - это маленький электрический компонент, который похож на маленькую лампочку. И ему для работы хватит напряжения питания 5 вольт.

Плата Arduino уже имеет установленный светодиод. На рисунке он выделен красным кружком

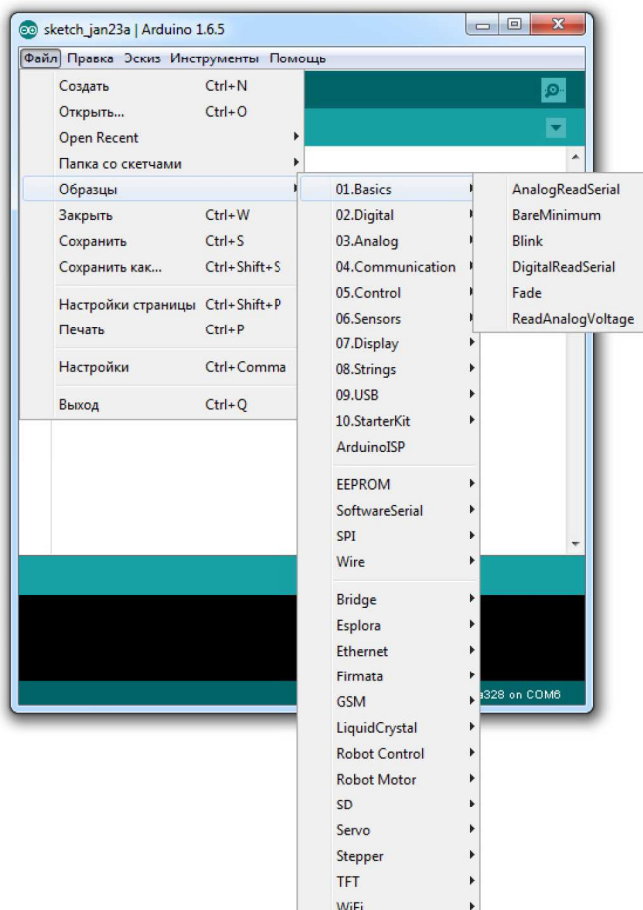


При подключении платы Arduino Nano к компьютеру светодиод мигает с частотой 0,5 Герц. Т.е. 1 секунду светится, затем на 1 секунду гаснет.

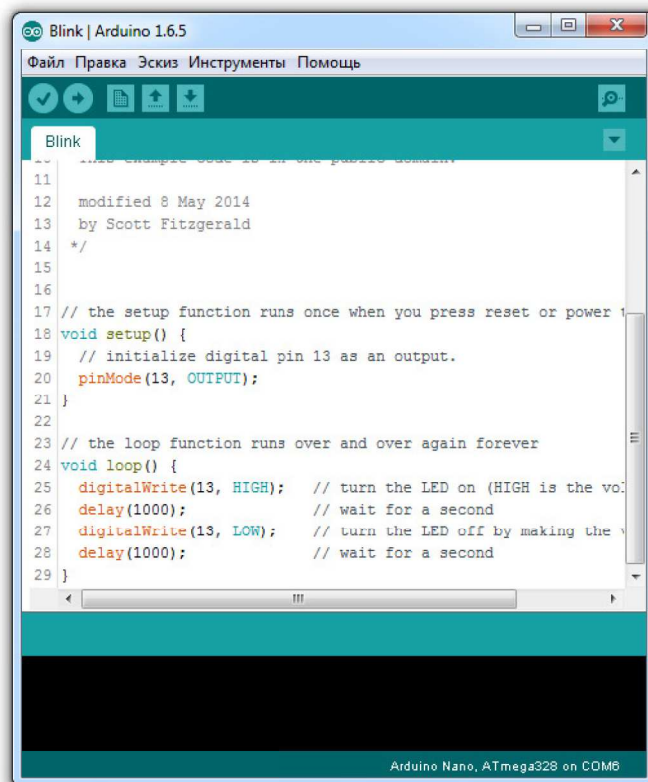
«Научим» светодиод мигать по-другому.

В среде разработки программ Arduino IDE есть уже большое количество примеров.

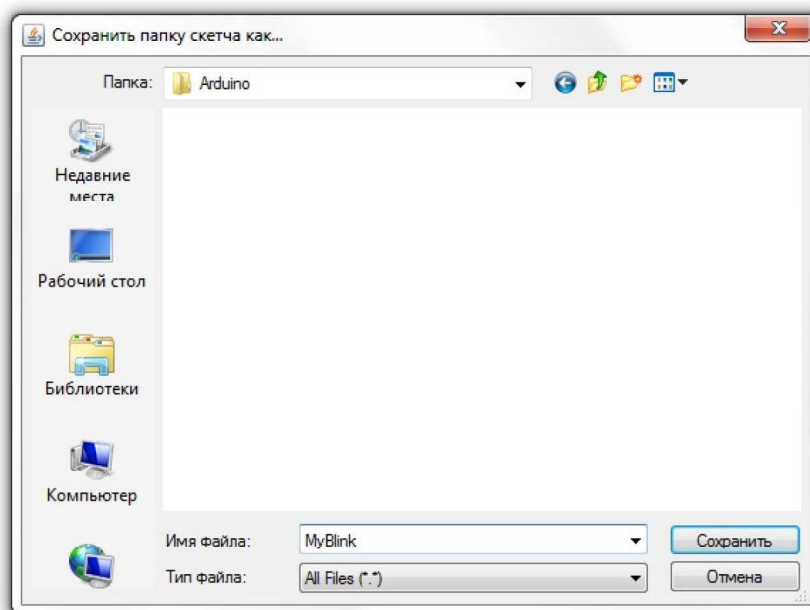
Откроем пример для управления мигания светодиодом. Для этого в Arduino IDE выбираем путь до кнопки «Blink» (Файл – образцы – 01.Basic – Blink)



После загрузки в окне IDE появится текст программы для мигания светодиода.



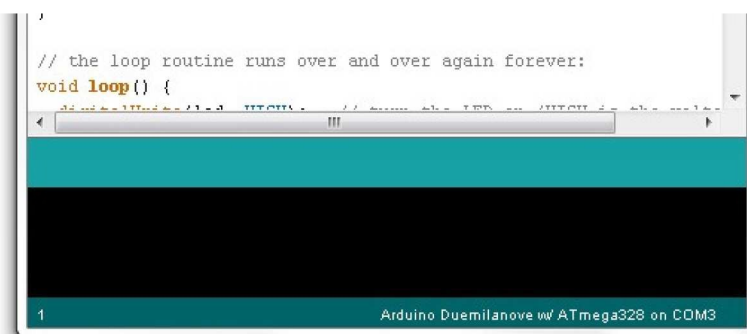
Но прежде, чем начать работы по изменению скорости мигания светодиода необходимо сохранить копию этого скетча. Для этого в меню Файл в IDE выберите опцию «Сохранить как...». А затем сохраните скетч, присвоив ему имя MyBlink.



Вы сохранили копию 'Blink' в своей папке. Это означает, что, если вы хотите найти его снова, вы можете просто открыть его с помощью опции Файл – Папка со скетчами.

Подключите плату Arduino к компьютеру с помощью USB-кабеля и убедитесь, что тип платы и тип последовательного порта установлены правильно. Дождитесь сообщения, что драйвера найдены и установлены.

Arduino IDE покажет вам текущие настройки платы в нижней части окна.



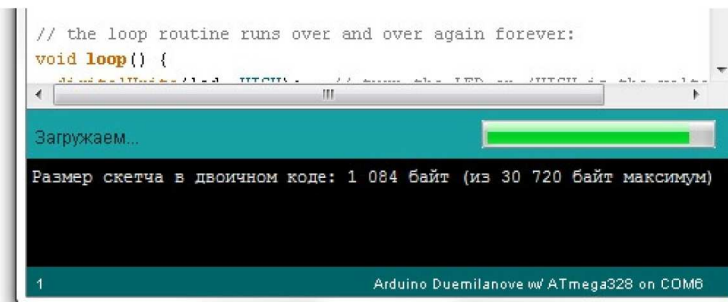
Нажмите на кнопку «Загрузить». Это вторая кнопка слева на панели инструментов.



В нижней части IDE выводит ряд сообщений. Сначала выводится сообщение о компиляции скетча. Т.е. о преобразовании программы в формат, понятный для процессора.

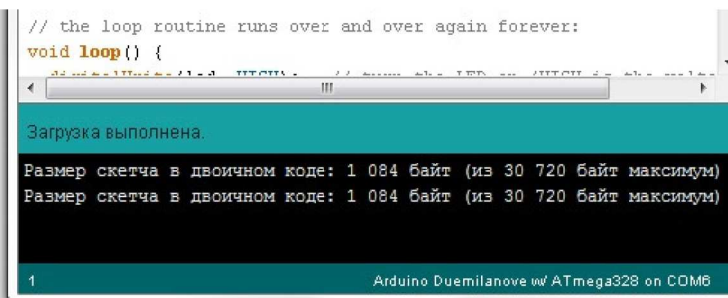


Далее статус процесса изменится на «Загружаем». На данный момент, светодиоды на Arduino должны начать мерцать, указывая, что скетч загружается в плату.

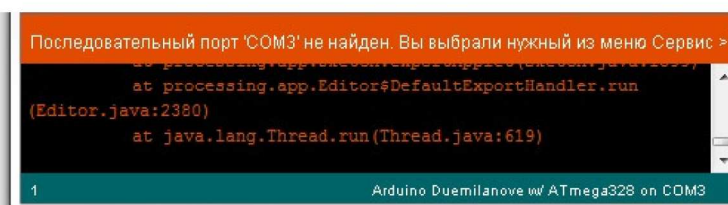


Наконец в строке состояния появится текст «Загрузка выполнена».

Текст в информационном окне говорит нам, что скетч занимает 1084 байта из 30720 байт максимум.



Но, если вы забыли подключить плату, или драйверы установлены неправильно, или порт выбран неправильно, то в окне может появиться следующая надпись:



Проверьте подключение платы к компьютеру и еще раз загрузите скетч.

Код программы.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
```

```
pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

Обратите внимание, что текст программы состоит из непосредственно инструкций и комментариев этих инструкций. Комментарии не влияют на выполнение программы процессором. С помощью комментариев можно более подробно описывать работу программы.

Многострочные комментарии выделяются символами `/*` и `*/`.

Однострочные комментарии выделяются символами `//` только в начале строки.

Так же обратите внимание, что инструкции пишутся на английском языке.

Комментарии в примерах, которые входят в IDE, так же написаны на английском языке.

Но при самостоятельном написании скетча комментарии можно писать на русском языке.

Рассмотрим код программы построчно:

```
void setup() {
pinMode(13, OUTPUT);
}
```

Функция **Setup** выполняется один раз после старта скетча или при включении питания Arduino. И эта функция должна присутствовать в каждом скетче.

В данном примере она определяет следующее:

Мы работаем с контактом 13 платы Arduino Nano. И то, что этот контакт мы определяем, как **ВЫХОД**. Это означает то, что на выводе 13 платы будет появляться сигнал, который создает сам контроллер.

Светодиод на плате Ардуино и на самом деле подсоединен к выводу 13.

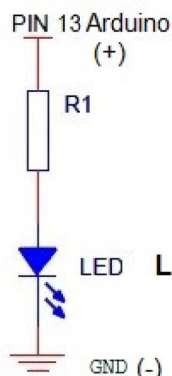
Для любого скетча необходима функция **Loop**. Эта функция вызывается сразу после завершения **Setup**, и повторяется снова и снова. Т.е. **Loop** после того, как закончит работу своей команды, сразу же начнет выполнять её снова.

```
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

Внутри цикла функции, размещена команда, которая переключает состояние пина в одно из двух возможных. **HIGH** или **LOW**. **HIGH** соответствует логической **единице** (она же в свою очередь соответствует напряжению питания)

микроконтроллера +5 Вольт. Состояние вывода **LOW** соответствует логическому нулю, это примерно 0 Вольт.

Светодиод **L** (пин 13) подключен в Arduino следующим образом:



Если пин 13 на плате Ардуино установить в состояние **HIGH** (5 Вольт), то светодиод начнет светиться так, как через него потечет электрический ток. Если пин 13 установить в состояние **LOW** (0 Вольт), то светодиод светиться не будет.

Следовательно, оператор **digitalWrite(13, HIGH)** формирует +5В на выходе 13 и зажигает таким образом светодиод **L**.

delay(1000);

Эта команда заставляет микроконтроллер остановиться и ничего не делать целую 1000 миллисекунд. 1000 мс = 1 с ;). Т.е. на выходе 13 в течении 1000 мс будет высокий уровень и светодиод будет светиться.

digitalWrite(13, LOW);

Эта команда устанавливает пин 13 в состояние **LOW**. Напряжение на выходе становится близким к нулю. И светодиод «гаснет». Затем снова выполняется команда паузы на 1000 миллисекунд.

На этом выполнение цикла заканчивается, но функция **loop** снова автоматически вызывается и запускает команду **digitalWrite(13, HIGH)**. И так далее пока подается питание на контроллер.

Теперь мы попробуем изменить скорость мигания светодиода. Для этого надо изменить параметры команды **delay**.

```

21 }
22
23 // the loop function runs over and over again forever
24 void loop() {
25   digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
26   delay(1000); // wait for a second
27   digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
28   delay(1000); // wait for a second
29 }

```

Сохранение отменено

3 Arduino Nano, ATmega328 on COM8

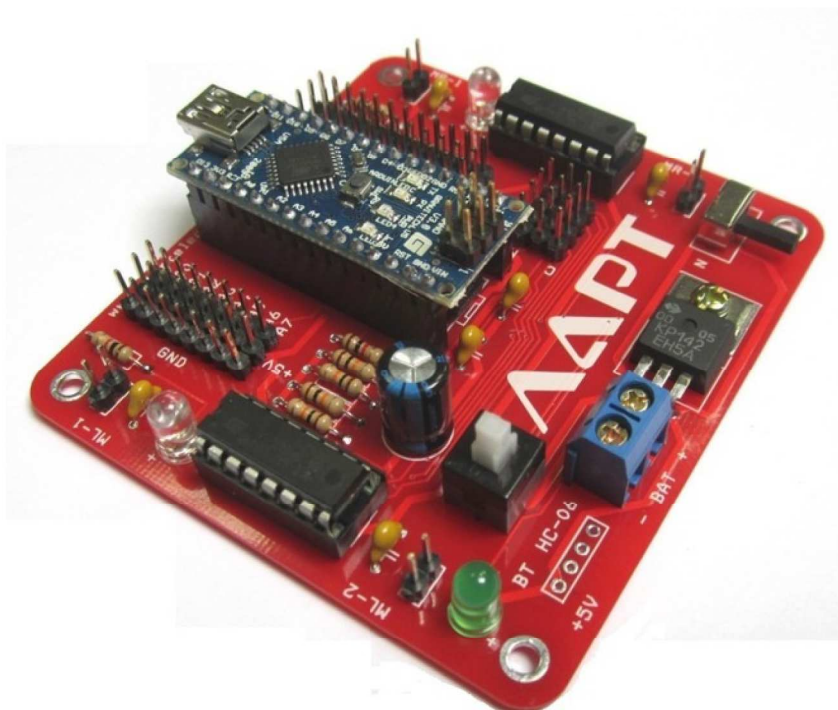
Для того, чтобы заставить мигать светодиод быстрее изменим значение паузы с 1000 миллисекунд на 500. Загрузим скетч и увидим, что светодиод действительно мигает в два раза быстрее.

Затем загрузите измененную вами программу в плату Arduino, нажав кнопку “Загрузить”.

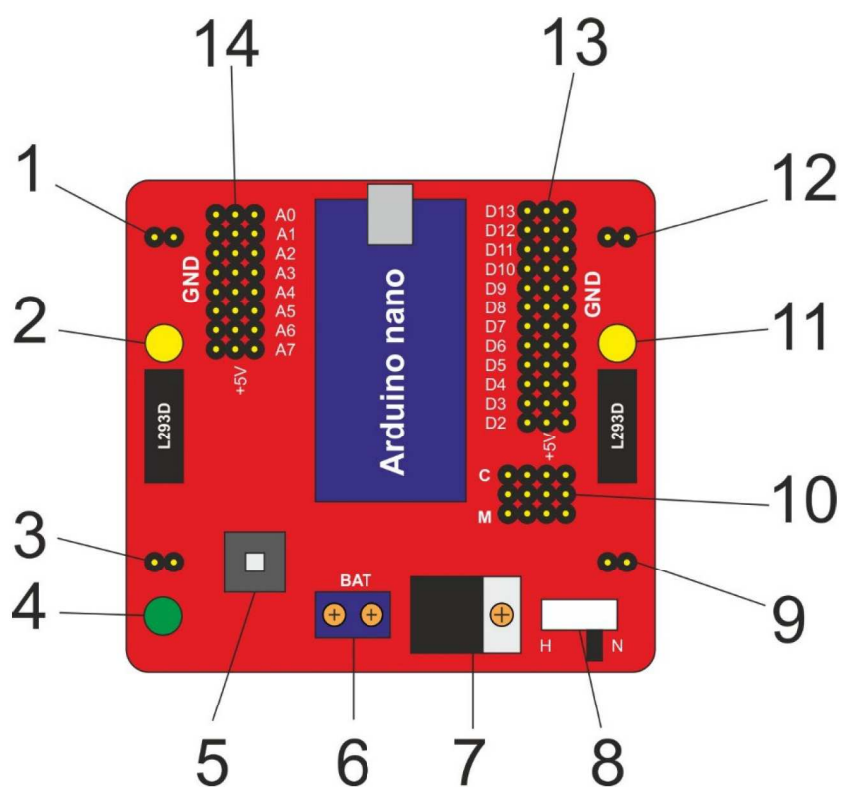


Если изменения в скетч внесли без ошибок, то после загрузки светодиод на плате Arduino Nano должен мигать с длительностью, установленной в скетче

Блок управления R-5



Назначение элементов блока R-5



1,3,9,12 – разъемы для подключения электромоторов.

2,11 – двунаправленные светодиоды, индицирующие направление вращения электромоторов.

4 – светодиод – индикатор электропитания.

5 – кнопка включения питания.

6 – клемма для подключения батареи питания.

7 – стабилизатор 5 Вольт.

8 – переключатель питания электромоторов. В положении N на драйвер электромоторов подается напряжение 5 вольт. В положении H на драйвер электромоторов подается питание непосредственно с батареи.

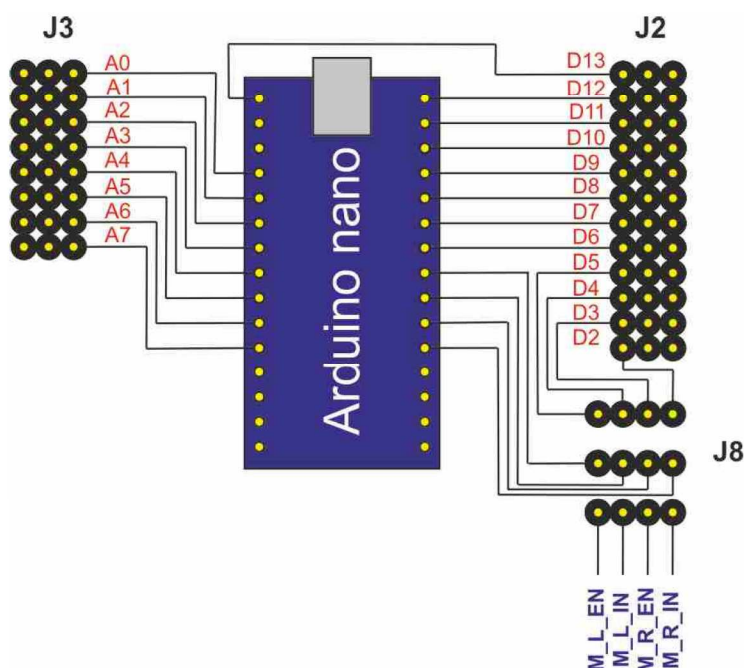
10 – разъем J8 для коммутации цифровых выходов Ардуино D2, D3, D4, D5 на вход драйвера или на разъем J2.

13 – разъем J3. К этому разъему выведены соответствующие выходы Ардуино.

14 – разъем J2. К этому разъему выведены аналоговые выходы Ардуино.

Разъемы J2, J3 имеют три ряда контактов. Внешний ряд подключен к минусу батареи (GND), средний ряд подключен к выходу стабилизатора 5 вольт. Внутренние ряды подключены к соответствующим контактам Ардуино.

Схема подключения выходов Ардуино



Цифровые и аналоговые контакты платы Arduino nano выведены на соответствующие разъемы J2, J3.

Цифровые контакты D2, D3, D4, D5 с контроллера идут на разъем J8. В разьеме J8 джамперами выходы контроллера подключаются или к разъему J2 или к драйверу электромоторов.

Если блок управления R-5 используется для управления роботом, то джамперы необходимо установить в нижнее положение, что бы выходы Ардуино соединились со входами драйвера.

Если блок R-5 используется для других экспериментов, то джамперы необходимо установить в верхнее положение.

Назначение входов драйвера электромоторов.

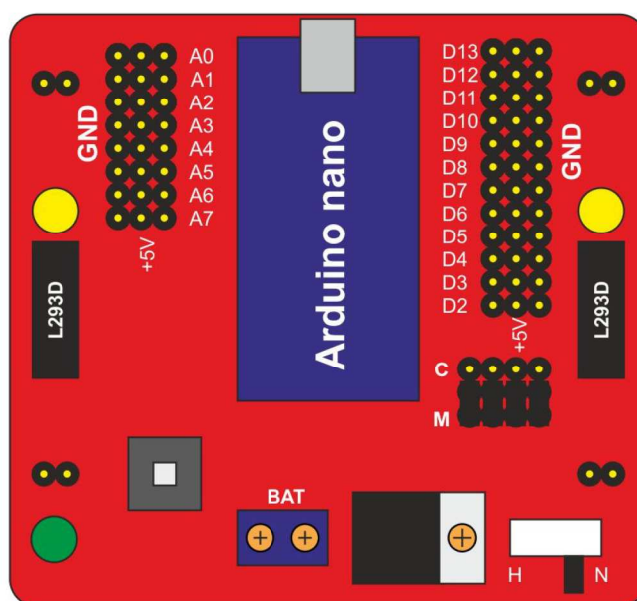
M_R_IN – вход драйвера, управляющий направлением вращения правого электромотора. Высокий уровень (HIGHT) – вращение вперед.

M_R_EN – вход драйвера, разрешающий вращение правого электромотора. Высокий уровень (HIGHT) разрешает вращение. При подаче на вход сигнала с широтно-импульсной модуляцией (ШИМ, PWM) вход управляет скоростью вращения.

M_L_IN - вход драйвера, управляющий направлением вращения левого электромотора. Высокий уровень (HIGHT) – вращение вперед.

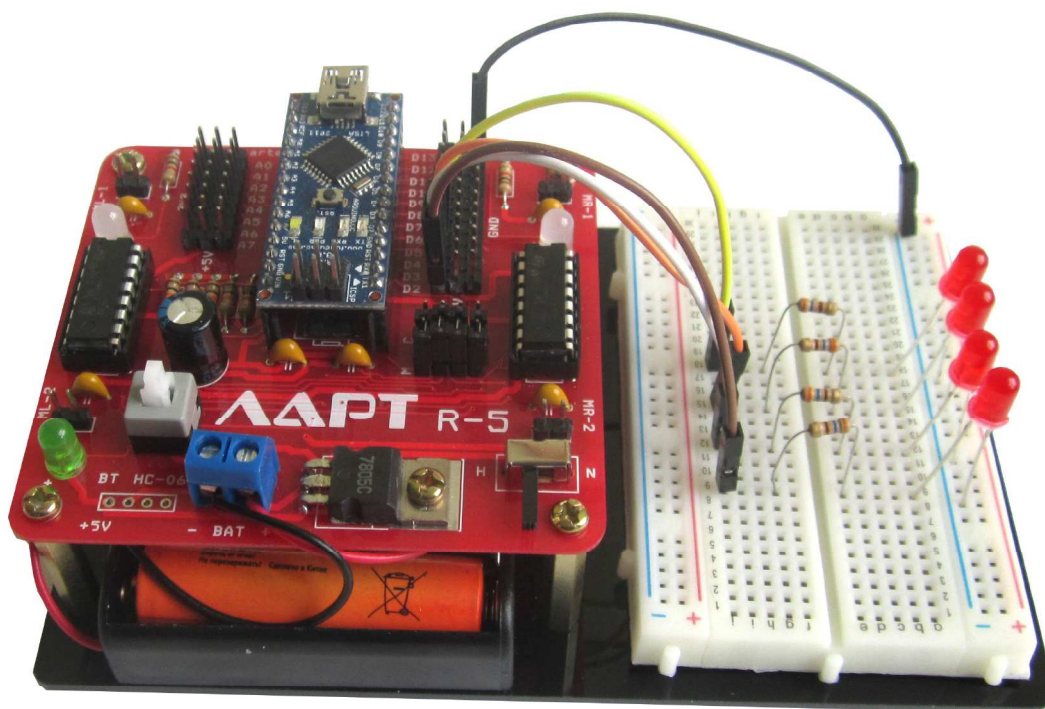
M_L_IN - вход драйвера, разрешающий вращение левого электромотора. Высокий уровень (HIGHT) разрешает вращение. При подаче на вход сигнала с широтно-импульсной модуляцией (ШИМ, PWM) вход управляет скоростью вращения.

Т.е. исходя из вышесказанного для того, чтобы мы могли управлять моторами робота мы должны соединить джамперами средние и нижние контакты разъема J8.



Если мы хотим использовать выходы D2, D3, D4, D5 в работе устройств, то в этом случае джамперами необходимо соединить средние и верхние контакты разъема.

Макетный модуль



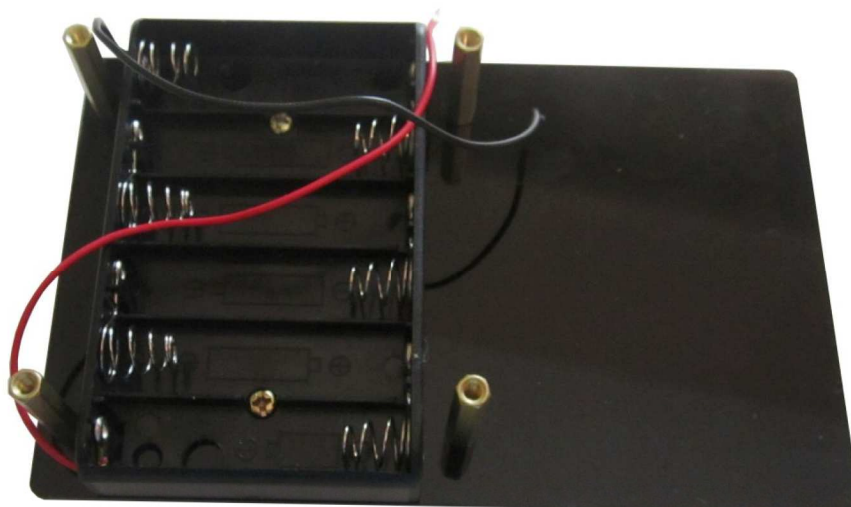
Макетный модуль предназначен для проведения различных экспериментов с программированием, экспериментов с электронными схемами.

Порядок сборки модуля

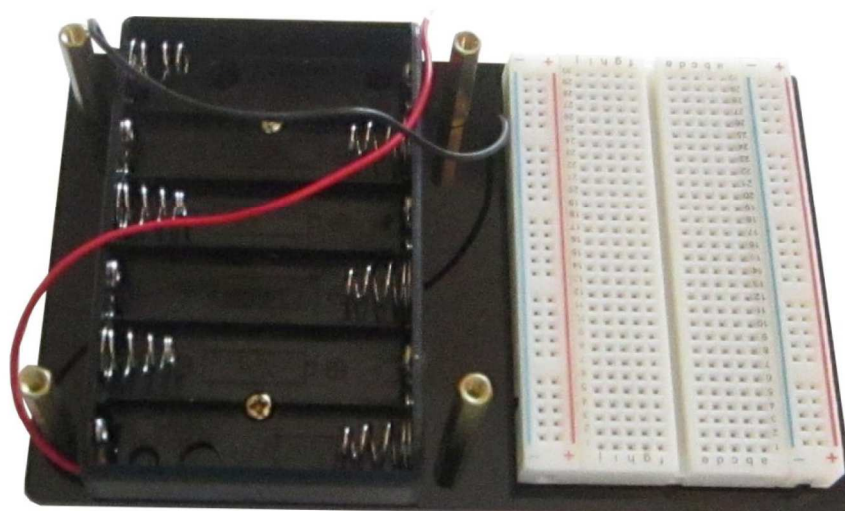
Все компоненты модуля устанавливаются на пластиковую подставку черного цвета размером 145 x 90 мм.



Закрепляем батарейный отсек на пластине. Для этого понадобятся 2 винта М3 x 6мм и 2 гайки М3. Так же устанавливаем 4 металлические стойки длиной 25 мм.

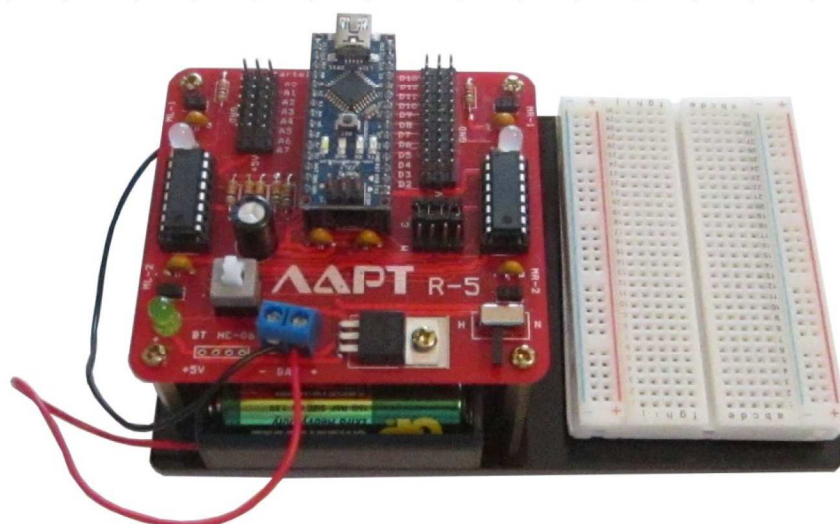


Закрепляем макетку на подставку. Для этого снимаем защитную пленку с нижней стороны макетки и аккуратно прикладываем клеящим слоем к пластиковой подставке. Ориентировать макету удобнее так, что бы она располагалась синей линией к модулю R-5.

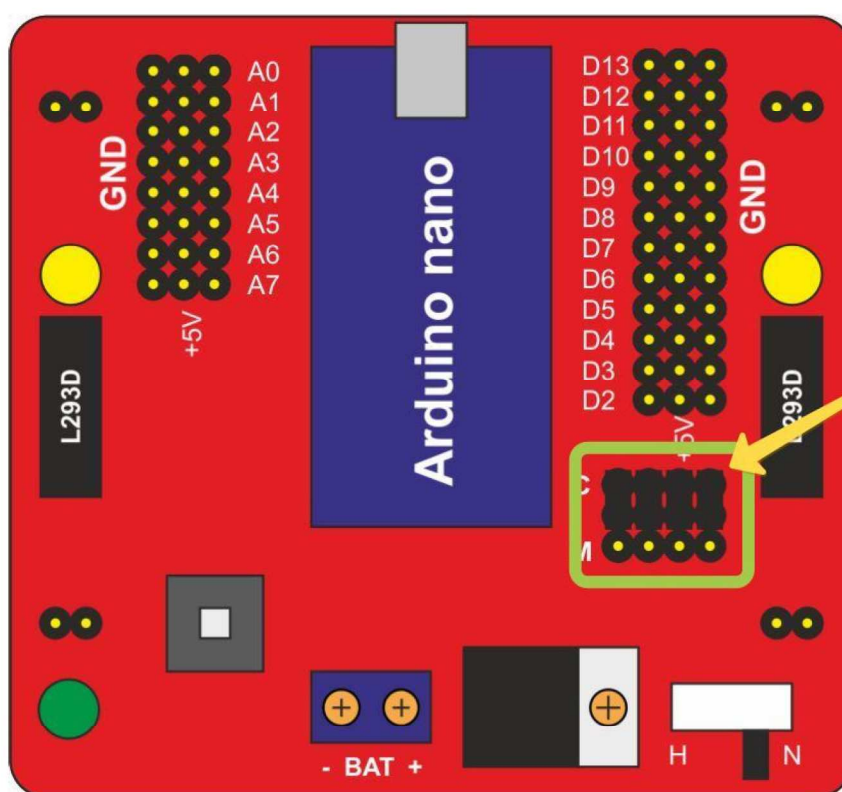


Устанавливаем в батарейный отсек 6 батареек типа AA.

Устанавливаем и закрепляем винтами М3 х 6мм на металлические стойки блок управления R-5. Закрепляем провода батарейного отсека в клеммы питания модуля R-5. **Красный провод – плюс. Черный провод – минус.**



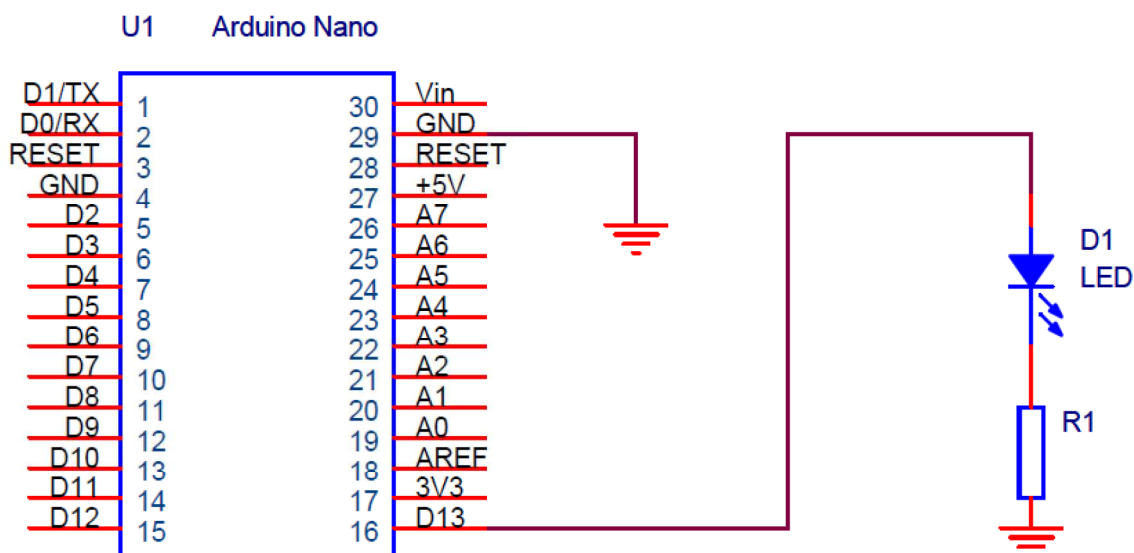
Так как в ближайших экспериментах мы не будем использовать электромоторы, то джамперы желательно переключить вверх



Подключение светодиода

Этот опыт мы уже проводили, когда настраивали среду программирования Arduino IDE и проверяли работоспособность платы Арудино. Сейчас постараемся более подробно остановиться на электронных компонентах, схеме и программе.

Схема устройства.



Электрическая схема мигающего светодиода достаточно простая. Ножка контроллера Arduino D13 соединяется с анодом светодиода D1. Катод светодиода соединяется с резистором R1. И второй вывод резистора подключается к общему минусу  схемы. Общий минус здесь обозначен символом «земля».

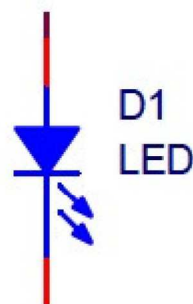
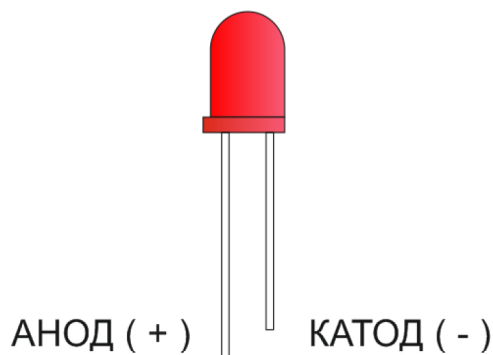
Ножка контроллера Arduino так же подключена к общему минусу схемы.

Светодиод.

Светодиод - это светоизлучающий полупроводниковый электронный компонент.

Внешний вид светодиода

Обозначение на электрической схеме



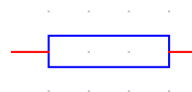
Светодиод имеет полярность. Т.е. один вывод светодиода должен подключаться к плюсу схемы, а второй вывод к минусу схемы. Плюсовой вывод светодиода имеет большую длину, чем минусовой. Срез ободка корпуса светодиода расположен со стороны катода.

Резистор.

Резистор – электронный компонент. В основном он используется для ограничения электрического тока. Резистор не полярный компонент. Поэтому в схеме его можно подключать любой стороной к батарее питания.

Внешний вид резистора

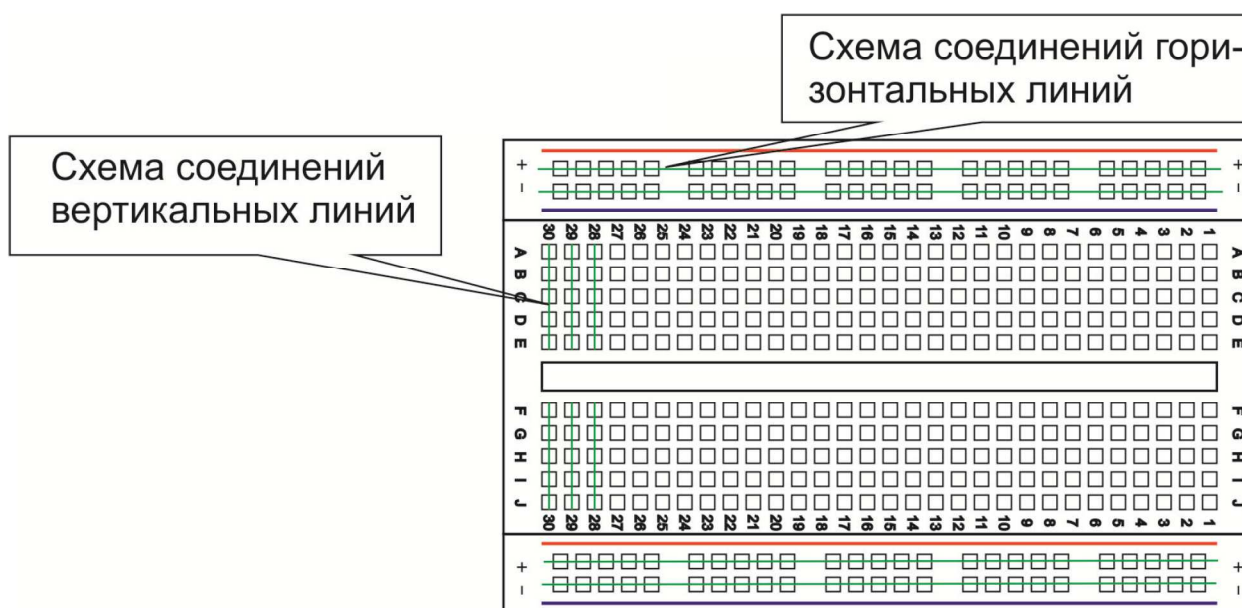
Обозначение на электрической схеме



Резисторы так же имеют различную величину. В нашей схеме мы используем резистор величиной 360 Ом.

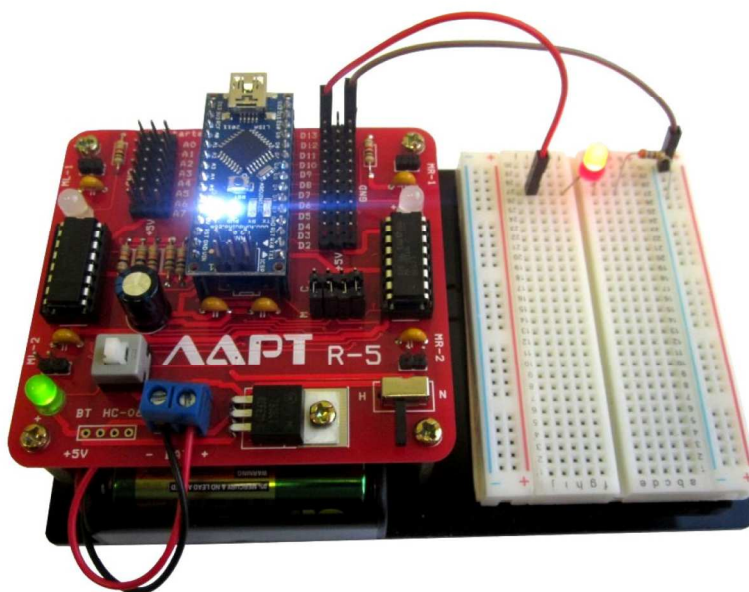
Макетная плата. Соединения отверстий.

Макетная плата – это поле с большим количеством крошечных отверстий. Проволочные выводы электронных компонентов (резисторы, диоды, транзисторы, медные проводники) вставляются в эти отверстия. Отверстия в верхних двух горизонтальных линиях и отверстия в нижних двух горизонтальных линиях соединены друг с другом. Отверстия в средней части макетной платы связаны между собой вертикально. Как показано на рисунке.



Резистор и светодиод подключаем по схеме.

1. Подключаем минус(GND) питания Ардуино к минусовой шине макетной платы. Для этого устанавливаем один конец проволоочной перемычки в гнездо GND на плате Ардуино, а второй конец перемычки устанавливаем в отверстие вдоль синей линии макетки.
2. Устанавливаем светодиод в макетку. Ножки светодиода вставляем в отверстия свободного поля так, чтобы длинная ножка находилась в отверстии левой части свободного поля, а короткая в отверстии правой части. К короткой ножке подключаем ножку резистора, вторую ножку резистора устанавливаем в отверстие вдоль синей линии. Затем проволоочной перемычкой соединяем АНОД светодиода и 13 контакт платы Ардуино.



Ранее в разделе «УСТАНОВКА ПО» мы уже загружали скетч MyBlink в Ардуино. Поэтому при включении питания светодиод на макетке должен мигать.

В качестве дополнительного эксперимента можно в скетче можно изменить номер контакта Ардуино и параметры мигания светодиода. Например:

```
void setup() {
    // initialize digital pin 10 as an output.
    pinMode(10, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(2000);            // wait for a second
    digitalWrite(10, LOW);  // turn the LED off by making the voltage LOW
    delay(500);             // wait for a second
}
```

Изменим немного нашу программу. Введем понятие ПЕРЕМЕННАЯ.

```
Int ledpin = 13;

void setup() {
  pinMode(ledpin, OUTPUT);
}

void loop() {
  digitalWrite(ledpin, HIGH);
  delay(1000);

  digitalWrite(ledpin, LOW);
  delay(1000);
}
```

Число 13 – это переменная. Мы можем изменить его, например, если нам будет необходимо подключить светодиод не к контакту 13, а к другому.

Что бы ввести переменную в программу её необходимо ОБЪЯВИТЬ.

Для этого мы пишем `int ledpin = 13;`

Слово `int` означает, что в качестве переменной мы будем использовать целое число.

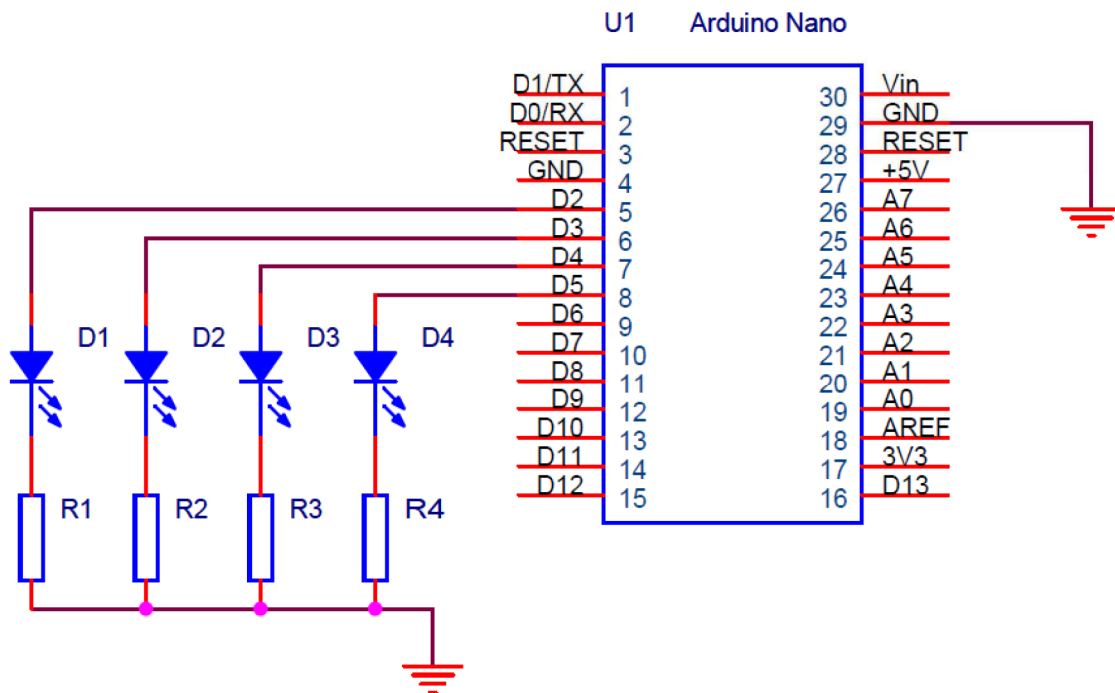
`Ledpin` – это имя нашей переменной. Его мы придумываем сами.

13 – это сама переменная.

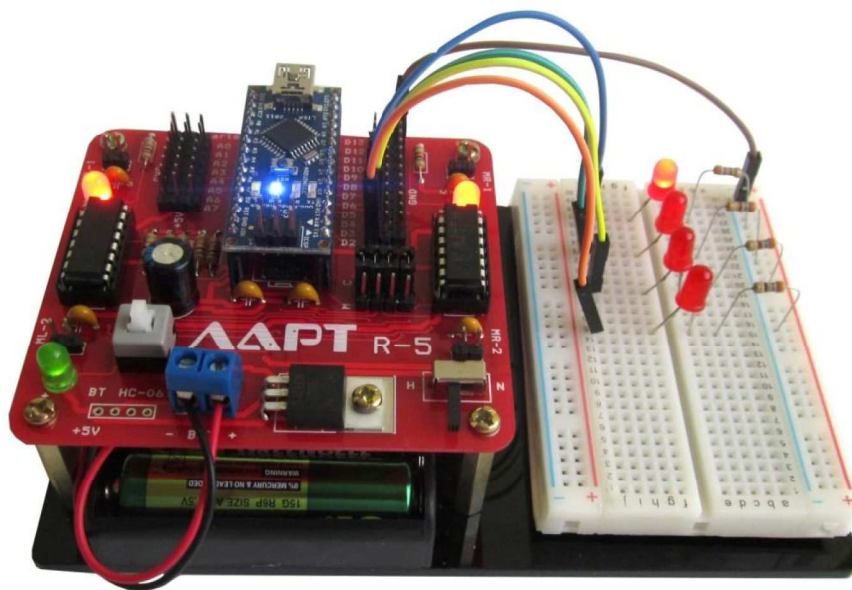
Программа стала удобнее для работы с ней.

Подключаем 4 светодиода

Для этого эксперимента соберем схему с 4-мя светодиодами и 4-мя резисторами



Резисторы R1-R4- 360 Ом



ПРИМЕР 1.

Создадим линейку поочередно мигающих светодиодов и используя не один, а четыре штуки. Заставим моргать светодиоды в бесконечном цикле

последовательности 1-2-3-4-1-2-3-4- ... с задержкой в одну секунду.

Прежде чем приступить к написанию кода, попробуем рассмотреть алгоритм работы устройства.

*Устанавливаем, что все контакты мы будем использовать как выходы
Создаем переменную для хранения длительности задержки в миллисекундах
Начинаем бесконечно повторяемый цикл
включить контакт 2, ждать установленное время задержки, выключить его
включить контакт 3, ждать установленное время задержки, выключить его
Повторяем эти операции для остальных выходов
Окончание бесконечного цикла*

```
int del=100; // переменная int led определяет время свечения светодиодов
```

```
void setup()
{
  // инициализируем пины Ардуино, как выходы платы :
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}
// далее пишем содержимое функции loop. Т.е. те операции, которые будут
//повторяться бесконечно, пока включено питание Ардуино
```

```
void loop()
{
  digitalWrite(2, HIGH); // включаем светодиод D1, подключенный к пину 2
  delay(del);           // ожидаем. Время ожидания указано в переменной del
  digitalWrite(2, LOW); // выключаем
  digitalWrite(3, HIGH); // включаем D2
  delay(del);           // ожидаем
  digitalWrite(3, LOW); // выключаем
  digitalWrite(4, HIGH); // включаем D3
  delay(del);           // ждем
  digitalWrite(4, LOW); // выключаем
  digitalWrite(5, HIGH); // включаем D4
  delay(del);           // ждем
  digitalWrite(5, LOW); // выключаем
}
```

Что бы изменить время свечения светодиодов нам достаточно изменить значение только в одном месте. Это переменную del

ПРИМЕР 2

В этом примере в качестве переменных мы установим номера пинов Ардуино и в алгоритм работы введем паузу после выключения светодиода.

```
int led1Pin = 2;
int led2Pin = 3;
```

```
int led3Pin = 4;
int led4Pin = 5;

void setup()
{
  // устанавливаем пины Ардуино, как выходы
  pinMode (led1Pin, OUTPUT);
  pinMode (led2Pin, OUTPUT);
  pinMode (led3Pin, OUTPUT);
  pinMode (led4Pin, OUTPUT);
}

void loop()
{
  digitalWrite (led1Pin, HIGH);
  delay (1000);
  digitalWrite (led1Pin, LOW);
  delay (1000);

  // сделать то же самое для остальных 3 -х светодиодов
  digitalWrite (led2Pin, HIGH);
  delay (1000);
  digitalWrite (led2Pin, LOW);
  delay (1000);

  digitalWrite (led3Pin, HIGH);
  delay (1000);
  digitalWrite (led3Pin, LOW); //
  delay (1000);

  digitalWrite (led4Pin, HIGH);
  delay (1000);
  digitalWrite (led4Pin, LOW);
  delay (1000);
}
```

ПРИМЕР 3

Мы можем этот скетч оставить так, как есть. Но на самом деле это не самый оптимальный вариант программы бегущего огня. Гораздо меньше места в памяти программ будет занимать скетч, если мы используем такую структуру, как цикл. Конструкция `for()` удобная для циклического повторения операций.

В нашем случае мы можем заменить

вот как мы запишем цикл:

```
for (int ledPin=2;ledPin<6;ledPin++)
{
  digitalWrite(ledPin, HIGH);
  delay(1000);
}
```

```
digitalWrite(ledPin, LOW);
delay(1000);
}
```

В первой строке мы инициализируем переменную `ledPin` в значение пина `2` и говорим Arduino, что нам нужно перебирать значения переменной начиная с `2` до значения, не превышающего `5` (`ledPin<6`). `ledPin++` говорит Ардуино увеличивать значение с каждым циклом на `1`. Затем Ардуино выполняет строки внутри цикла. Т.е. включает и выключает светодиод с паузой `1000` мсек. Операция повторяется с каждым циклом.

```
int led1Pin = 2;
int led2Pin = 3;
int led3Pin = 4;
int led4Pin = 5;

void setup() {
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);
  pinMode(led4Pin, OUTPUT);
}

void loop() {
  for (int ledPin=2;ledPin<6;ledPin++)
  {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
  }
}
```

ПРИМЕР 4

В этом примере мы напишем программу так, что бы «огонек» бегал в обе стороны.

```
int del=100; // переменная int led определяем время свечения светодиодов
void setup()
{
  // инициализируем пины Ардуино, как выходы платы :

  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}
// далее пишем содержимое функции loop. Т.е. те операции, которые будут
//повторяться бесконечно, пока включено питание Ардуино
void loop()
{
```

```
digitalWrite(2, HIGH); // включаем светодиод D1, подключенный к пину 2
delay(del);           // ожидаем. Время ожидания указано в переменной del
digitalWrite(2, LOW);  // выключаем
digitalWrite(3, HIGH); // Включаем D2
delay(del);           // ожидаем
digitalWrite(3, LOW);  // выключаем
digitalWrite(4, HIGH); // Включаем D3
delay(del);           // ждем
digitalWrite(4, LOW);  // выключаем
digitalWrite(5, HIGH); // включаем D4
delay(del);           // ждем
digitalWrite(5, LOW);  // выключаем
digitalWrite(4, HIGH); // включаем D3
delay(del);           // ждем
digitalWrite(4, LOW);  // выключаем
digitalWrite(3, HIGH); // включаем D2
delay(del);           // ждем
digitalWrite(3, LOW);  // выключаем
digitalWrite(2, HIGH); // включаем D1
delay(del);           // ждем
digitalWrite(2, LOW);  // выключаем
}
```

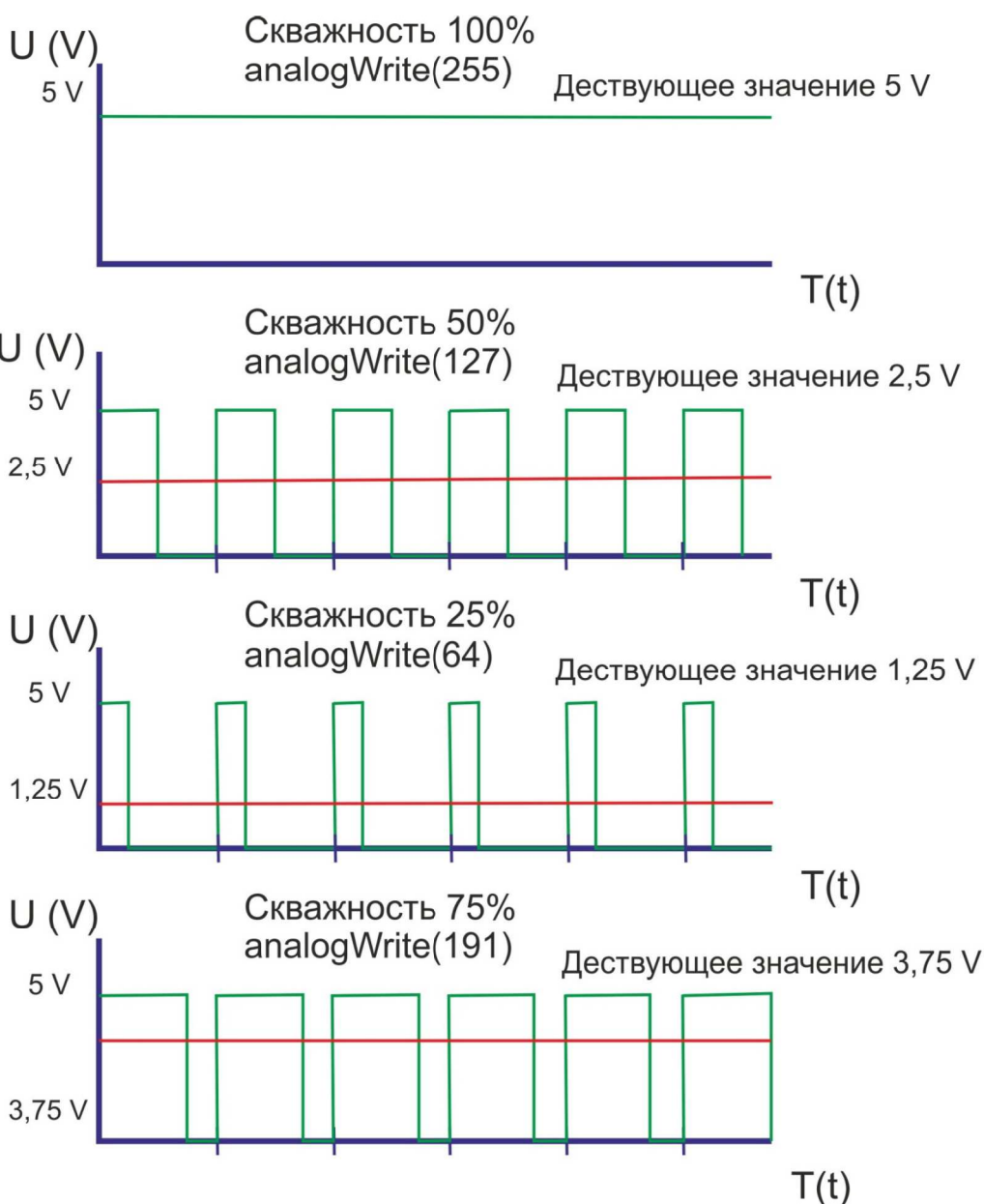

Регулировка яркости светодиода. ШИМ.

Иногда возникает потребность изменить яркость светодиода. Регулировка яркости светодиода осуществляется подачей на него не постоянного напряжения 5 Вольт, а последовательности прямоугольных импульсов различной длительности. Такой сигнал называется сигналом с широтно-импульсной модуляцией (ШИМ). На английском языке он пишется PWM.

Особенностью сигнала с ШИМ является то, что частота следования импульсов остается неизменной.

Метод широтно импульсной модуляции (ШИМ, англ. pulse-width modulation (PWM)) так же широко применяется для управления скоростью вращения электромоторов, путём изменения скважности импульсов, при постоянной частоте.

В контроллере Arduino есть специальные контакты, которые могут формировать сигнал с широтно-импульсной модуляцией. Это D3, D5, D6, D9, D11.



Сформировать сигнал ШИМ в контроллере Ардуино можно с помощью команды `analogWrite()`.

Синтаксис

`analogWrite(pin, value)`

Параметры

`pin`: порт вход/выхода на который подаем ШИМ сигнал. `value`: период рабочего цикла значение между 0 (полностью выключено) and 255 (сигнал подан постоянно).

ПРИМЕР 1

```
int ledPin = 9; // пин 9 Ардуино соединен со светодиодом
```

```
void setup()
```

```
{  
  pinMode(ledPin, OUTPUT); // инициализируем пин, как выход  
}
```

```
void loop() {
```

```
  analogWrite(ledPin, 255); // устанавливаем максимальное значение яркости  
  // светодиода  
  delay(1000); // ожидаем 1 секунду  
  digitalWrite(ledPin, LOW); // выключаем светодиод  
  delay(1000); // ждем секунду
```

```
  analogWrite(ledPin, 191); // устанавливаем примерно 75% яркости (191/255 ≈ 0.75)  
  delay(1000); // ожидаем 1 секунду  
  digitalWrite(ledPin, LOW); // выключаем светодиод  
  delay(1000); // ждем секунду
```

```
  analogWrite(ledPin, 40); // устанавливаем 15% яркости (40/255 ≈ 0.15)  
  delay(1000); // ждем секунду  
  digitalWrite(ledPin, LOW); // выключаем светодиод  
  delay(1000); // ждем секунду
```

```
  analogWrite(ledPin, 25); // устанавливаем 10% яркости (25/255 ≈ 0.1)  
  delay(1000); // ждем секунду  
  digitalWrite(ledPin, LOW); // выключаем светодиод  
  delay(1000); // ждем секунду  
}
```

ПРИМЕР 2

Теперь напишем код так, чтобы яркость плавно нарастала

Это можно сделать следующим образом добавляя по единице значение ШИМ повторяя практически один и тот же фрагмент кода:

```
analogWrite(ledPin, brightness);  
delay(5); //пауза  
brightness = brightness + 1;
```

Снова и снова (255 раз), увеличивая яркость на единицу каждый раз. Ниже пример такого кода:

```
int ledPin = 9; // пин 9 Ардуино подключаем к светодиоду  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT); // инициализируем пин как выход  
}  
  
void loop() {  
  int brightness = 0;  
  
  analogWrite(ledPin, brightness); // яркость = 0  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 1  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 2  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 3  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 4  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 5  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  analogWrite(ledPin, brightness); // яркость = 6  
  delay(5); // короткая пауза  
  brightness += 1;  
  
  //Повторяем до полного значения ШИМ = 255 (полная яркость)  
}
```

Но такая программа очень длинная, занимает много места в памяти Ардуино. Для решения этой проблемы мы можем использовать снова цикл `for()`.

В следующем скетче есть два цикла. Первый увеличивает значение ШИМ от (0) до максимальной яркости(255):

```
for (int brightness=0;brightness<256;brightness++)
{
  analogWrite(ledPin,brightness);
  delay(5);
}
```

Второй цикл от полной яркости до выключения светодиода:

```
for (int brightness=255;brightness>=0;brightness--)
{
  analogWrite(ledPin,brightness);
  delay(5);
}
```

Пауза(5) используется для того, чтобы уменьшить скорость изменения ШИМ.

Поэтому весь процесс занимает $5 \cdot 256 = 1280 \text{ms} = 1,28 \text{ сек}$

Мы используем `brightness--` , чтобы уменьшить значение яркости на 1 при каждом прохождении цикла. Цикл выполняется пока яркость ≥ 0 . Для того, что бы пройти весь диапазон яркости светодиода мы используем оператор $\geq$ вместо $>$.

```
int ledPin = 9;//пин 9 подключаем к светодиоду
```

```
void setup()
{
  pinMode(ledPin, OUTPUT);// инициализируем как выход
}
```

```
void loop()
{
  //увеличиваем значение ШИМ (0 to 255)
  for (int brightness=0;brightness<256;brightness++)
  {
    analogWrite(ledPin,brightness);
    delay(5);
  }
```

```
  delay(1000);// ожидаем 1 сек.
```

```
  //уменьшаем значение ШИМ (255 to 0)
  for (int brightness=255;brightness>=0;brightness--)
  {
    analogWrite(ledPin,brightness);
    delay(5);
  }
  delay(1000);//wait one second
}
```

Управление RGB светодиодом.

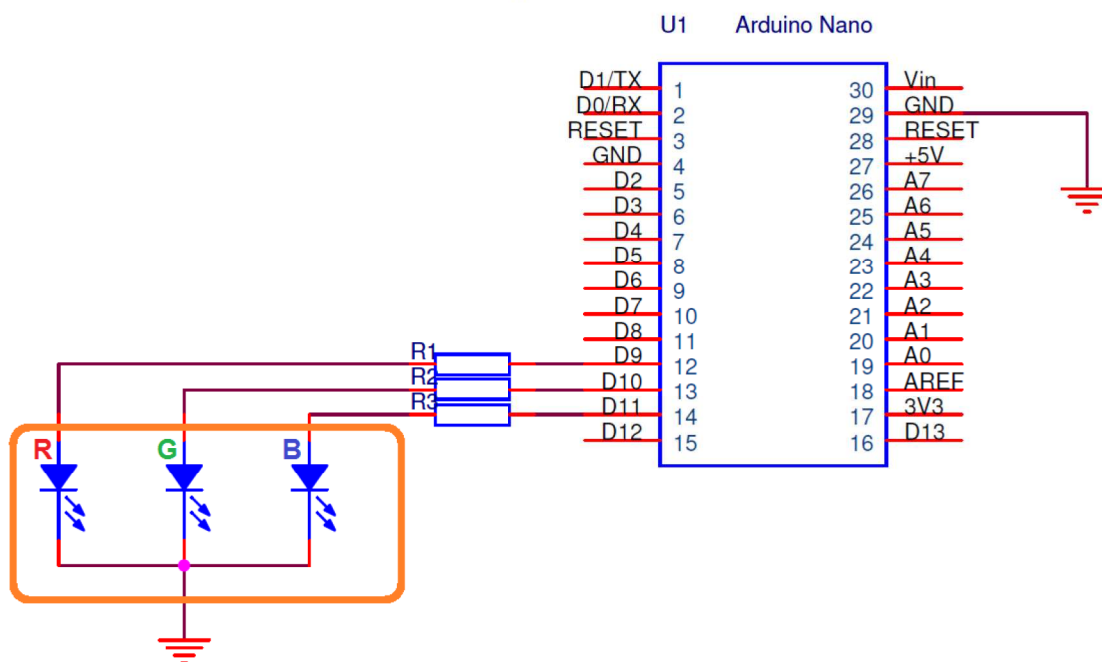
Трехцветный(RGB) светодиод выглядит так же, как и обычный, но внутри прозрачной колбы размещены кристаллы светодиодов красного(R), зеленого(G) и синего(B) свечения.

RGB светодиоды бывают с общим катодом и с общим анодом. В наших экспериментах мы будем использовать RGB светодиод с общим катодом.



Схема включения

Аноды светодиода мы подключим к пинам Ардуино, которые могут работать в режиме ШИМ (9,10,11). Таким образом, мы можем включать, выключать, изменять яркость отдельно каждого светодиода.



ПРИМЕР 1

Поочередно включаем встроенные светодиоды

```
int red = 9;
int green = 10;
int blue = 11;
```



```
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}

void loop()
{
  //включаем поочередно красный, зеленый, синий светодиоды
  digitalWrite(red, HIGH);
  delay(500);
  digitalWrite(red, LOW);
  delay(500);

  digitalWrite(green, HIGH);
  delay(500);
  digitalWrite(green, LOW);
  delay(500);

  digitalWrite(blue, HIGH);
  delay(500);
  digitalWrite(blue, LOW);
  delay(500);
}
```

ПРИМЕР 2

Затем с помощью функций [analogWrite\(\)](#) и [Random\(\)](#) для создания случайных уровней яркости для каждого светодиода. Таким образом, мы можем смешивать свечение светодиодов в различных пропорциях в зависимости от значения ИМ. И мы сможем получить большое разнообразие цветов ($255^3 = 16,581,375$ возможных цветов).

```
int red = 9;
int green = 10;
int blue = 11;
void setup()
{
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}
void loop()
{
  //выбираем случайный цвет
  analogWrite(red, random(256));
  analogWrite(blue, random(256));
  analogWrite(green, random(256));
  delay(1000); //ожидаем 1 секунду
}
```

ПРИМЕР 3

//RGB LED – плавное изменение цветов

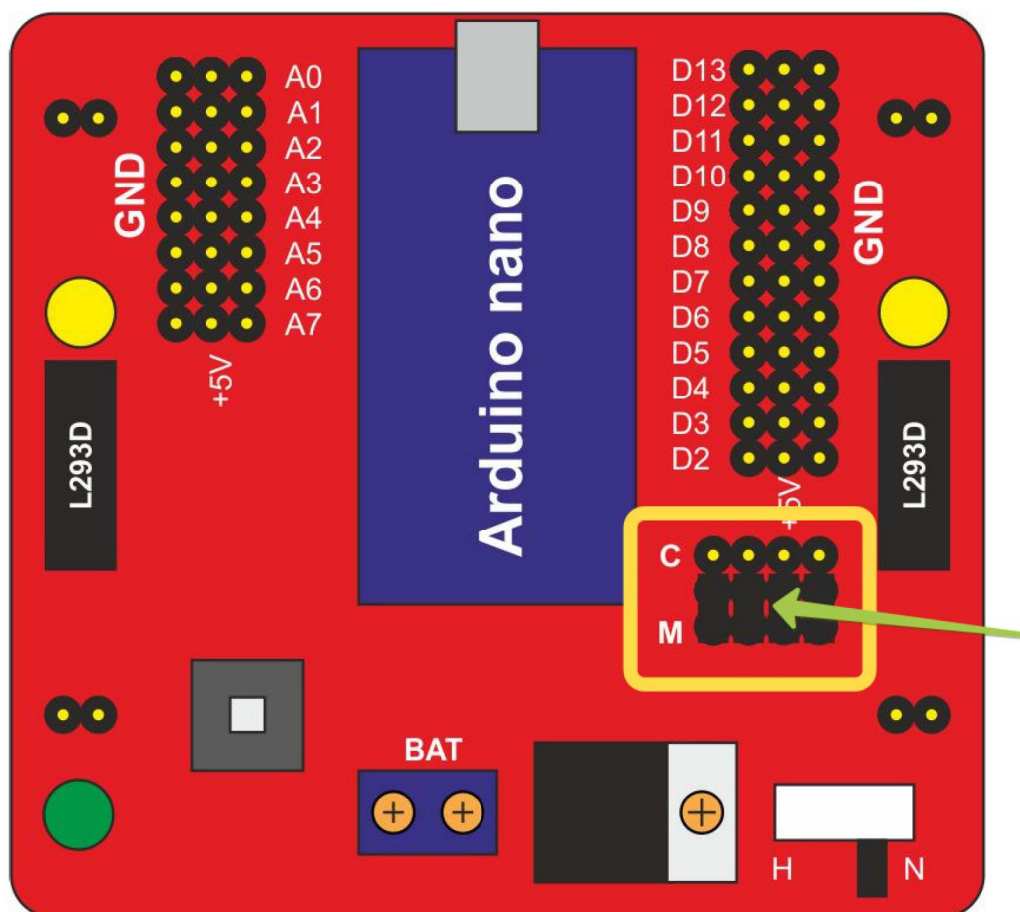
```
int red = 9;
int green = 10;
int blue = 11;
void setup(){
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);
}
void loop()
{
  for (int brightness=0;brightness<256;brightness++)
  {
    analogWrite(red, 255-brightness);
    analogWrite(green, brightness);
    delay(10);
  }
  for (int brightness=0;brightness<256;brightness++)
  {
    analogWrite(green, 255-brightness);
    analogWrite(blue, brightness);
    delay(10);
  }
  for (int brightness=0;brightness<256;brightness++)
  {
    analogWrite(blue, 255-brightness);
    analogWrite(red, brightness);
    delay(10);
  }
}
```

Этот скетч работает, но в нем также много повторяющихся строк. Можно упростить путем написания собственной вспомогательной функции, которая изменяет цвет от одного к другому:

```
void fader(int color1, int color2)
{
  for (int brightness=0;brightness<256;brightness++)
  {
    analogWrite(color1, 255-brightness);
    analogWrite(color2, brightness);
    delay(10);
  }
}
```

Движение робота вперед, назад.

Для работы платы R-5 с драйвером моторов необходимо джамперы установить в нижнее положение



Теперь уже приступаем непосредственно к программированию. Программа для контроллера Ардуино обычно состоит из 3-х частей.

В первой части, как в обычной математической задаче описываются исходные данные. Мы присваиваем названия, назначение, функционал контактов контроллера. Записываем, какие библиотеки мы будем использовать в ходе выполнения программы.

Во второй части с названием **Setup** пишем уже непосредственно код программы. Но этот код выполняется только один раз за время работы программы. Это необходимо для того, что бы запрограммировать выходы, входы контроллера на все время действия программы

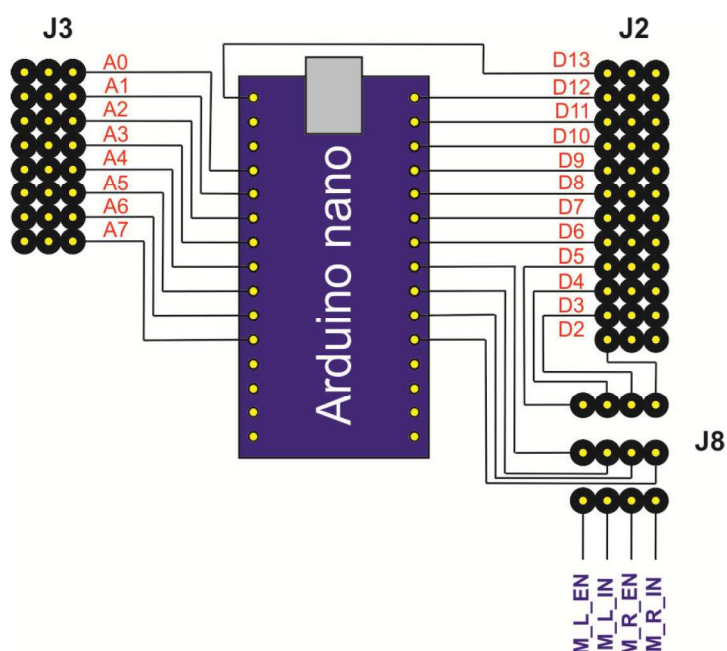
И третья часть программы **loop** представляет собой команды, которые исполняются процессором в течение всего периода работы процессора.

Рассмотрим простой пример кода для движения робота.

В первой части кода мы определяем, какими контактами платы Ардуино мы будем управлять драйвером и соответственно электромоторами, придумаем название команд, что бы нам было понятно их назначение.

В блоке R-5 контакты Ардуино жестко подключены к входам драйвера.

Давайте посмотрим на рисунок ниже:



Назначение входов драйвера электромоторов.

M_R_IN – вход драйвера, управляющий направлением вращения правого электромотора. Высокий уровень (HIGHT) – вращение вперед.

M_R_EN – вход драйвера, разрешающий вращение правого электромотора. Высокий уровень (HIGHT) разрешает вращение. При подаче на вход сигнала с широтно-импульсной модуляцией (ШИМ, PWM) вход управляет скоростью вращения.

M_L_IN – вход драйвера, управляющий направлением вращения левого электромотора. Высокий уровень (HIGHT) – вращение вперед.

M_L_EN – вход драйвера, разрешающий вращение левого электромотора. Высокий уровень (HIGHT) разрешает вращение. При подаче на вход сигнала с широтно-импульсной модуляцией (ШИМ, PWM) вход управляет скоростью вращения.

И у нас получается следующее:

Контакт Ардуино 2 – направление вращения правого мотора

Контакт Ардуино 3 – разрешение вращения правого мотора. При работе ШИМ – регулировка скорости вращения.

Контакт Ардуино 4 – направление вращения левого мотора.

Контакт Ардуино 5 – разрешение вращения левого мотора. При работе ШИМ – регулировка скорости вращения.

Пишем код.

```
#define DIR_R 2 // управлять направлением вращения правого мотора будем с
контакта 2
#define SPEED_R 3 // управлять разрешением вращения и скоростью вращения
правого //мотора будем с контакта 3
#define DIR_L 4 //управлять направлением вращения левого мотора будем с
контакта 4
#define SPEED_L 5 // управлять разрешением вращения и скоростью вращения
левого //мотора будем с контакта 5
```

```
//В этой части кода больше не будем задавать ни каких параметров
```

```
// приступаем ко второй части программы. Мы знаем, что в этой части кода
команды //исполняются только один раз
```

```
void setup()
{
    pinMode (DIR_R, OUTPUT); // Драйвер управляется выходными сигналами с
    Ардуино.
    //Поэтому мы определяем все контакты , как OUTPUT
    pinMode (SPEED_R, OUTPUT);
    pinMode (DIR_L, OUTPUT);
    pinMode (SPEED_L, OUTPUT);
}
```

```
// И в третьей части кода мы уже пишем алгоритм работы. Т.е. то, что наш робот
должен
```

```
//выполнять
```

```
void loop()
```

```
{
    digitalWrite (DIR_R, HIGH); // Команда digitalWrite устанавливает на контакте 2
    высокий
    //уровень. Для драйвера моторов это означает то, что мотор будет вращаться
    вперед
    digitalWrite (SPEED_R, HIGH); // высокий уровень на контакте 3 разрешает
    драйверу
    //вращать электромотор
    digitalWrite (DIR_L, HIGH);
    digitalWrite (SPEED_L, HIGH);
    delay(1000); // Вращаем 1 сек

    digitalWrite (DIR_R, HIGH);
    digitalWrite (SPEED_R, LOW); // Низкий уровень запрещает вращение моторов
    digitalWrite (DIR_L, HIGH);
}
```



```

digitalWrite (SPEED_L, LOW);
delay(1000);

digitalWrite (DIR_R, LOW); // Включаем низкий уровень и мотор должен вращаться
В
//обратную сторону
digitalWrite (SPEED_R, HIGH); //Разрешаем вращение мотора
digitalWrite (DIR_L, LOW);
digitalWrite (SPEED_L, HIGH);
delay(1000);

digitalWrite (DIR_R, LOW);
digitalWrite (SPEED_R, LOW); // Вращение запрещено
digitalWrite (DIR_L, LOW);
digitalWrite (SPEED_L, LOW);
delay(1000);
}

```

Скопируйте текст программы и вставьте его в Arduino IDE.

После копирования обязательно проверьте, что бы комментарии в каждой строчке начинались с двух символов //.

После проверки загрузите код в контроллер. Установите контроллер в блок R-5 и включите питание робота.

Наш робот, исполняя написанный выше скетч, должен в течение одной секунды ехать вперед, затем на секунду остановиться и начать движение назад. И так как команда **loop** выполняется постоянно, то робот будет выполнять эти команды пока включено питание.

Повороты робота влево, вправо.

Рассмотрим идеальный случай.

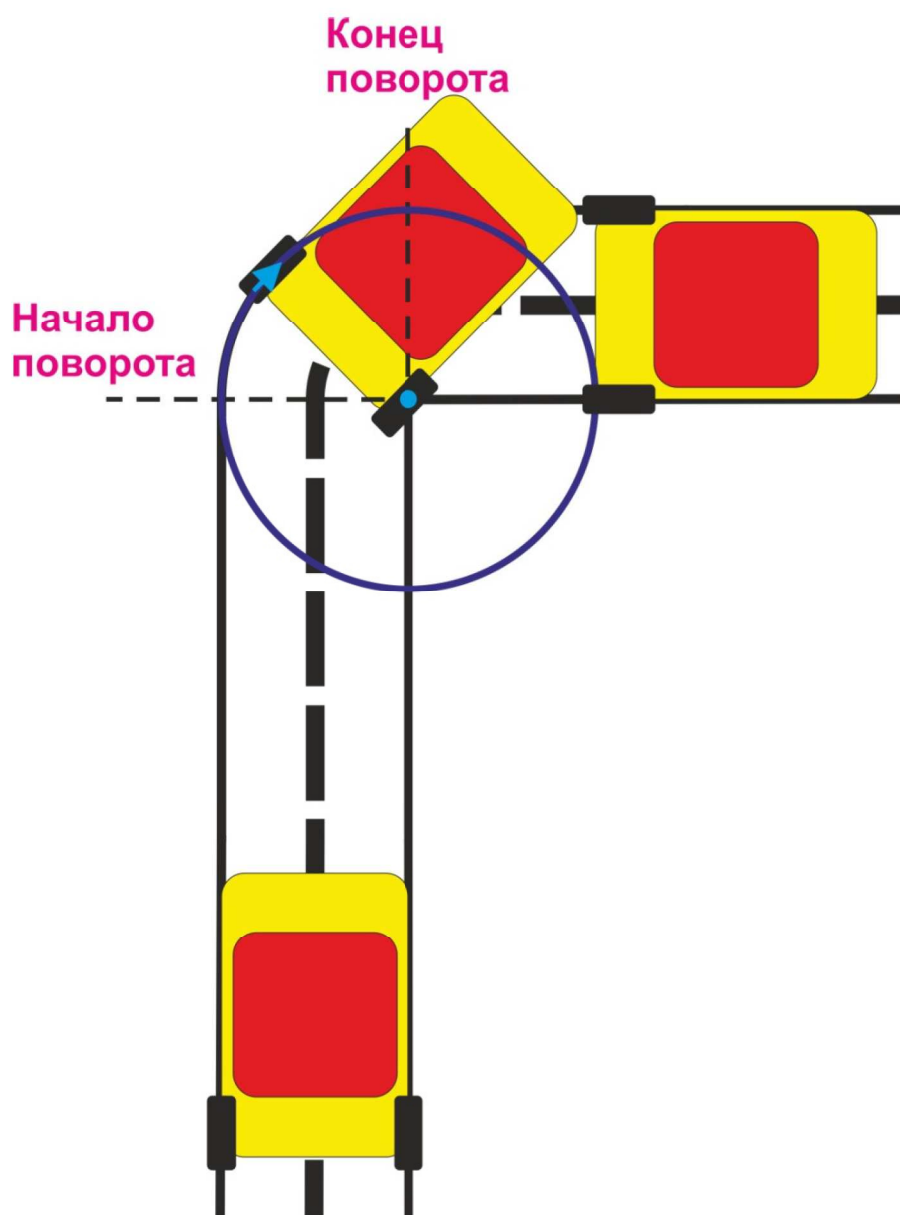
При этом колеса не проскальзывают по поверхности. Скорость движения стабильная.

При правильных расчетах робот точно выполняет все команды.

Вариант поворота №1.

Поворот осуществляется левым колесом. Правое колесо остановлено и является центром радиуса поворота.

Прежде, чем писать программу поворота, попробуем графически изобразить движение робота при повороте вправо на 90 градусов.



Из рисунка следует, что радиус поворота равняется расстоянию между центрами колес. И длина хода левого колеса равна одной четверти длины окружности.

Измерив расстояние между центрами колес на нашем роботе мы получим радиус окружности. Зная радиус мы сможем найти путь, который должно пройти левое колесо для поворота направо на 90 градусов.

Радиус окружности –
Длина пути левого колеса –
Время вращения левого колеса -

Пишем код.

```
#define DIR_R 2
#define SPEED_R 3
#define DIR_L 4
#define SPEED_L 5

void setup()
{
  pinMode (DIR_R, OUTPUT);
  pinMode (SPEED_R, OUTPUT);
  pinMode (DIR_L, OUTPUT);
  pinMode (SPEED_L, OUTPUT);
}

void loop()
{
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
  delay(1000); // Вращаем 1 сек

  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, LOW); // Низкий уровень запрещает вращение моторов
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, LOW);
  delay(2000); //останов 2 сек.

  //далее левое колесо вращаем 1 сек., правое колесо остановлено
  //после поворота направо робот сразу начинает движение вперед без паузы.
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, LOW);
  delay(1000); // Вращаем 1 сек

  //вращаем оба колеса 1 сек.
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
```

```
delay(1000); // Вращаем 1 сек
```

```
//вращаем правое колесо 1 сек., левое колесо остановлено  
digitalWrite (DIR_R, HIGH);  
digitalWrite (SPEED_R, LOW);  
digitalWrite (DIR_L, HIGH);  
digitalWrite (SPEED_L, HIGH);  
delay(1000); // Вращаем 1 сек.  
}
```

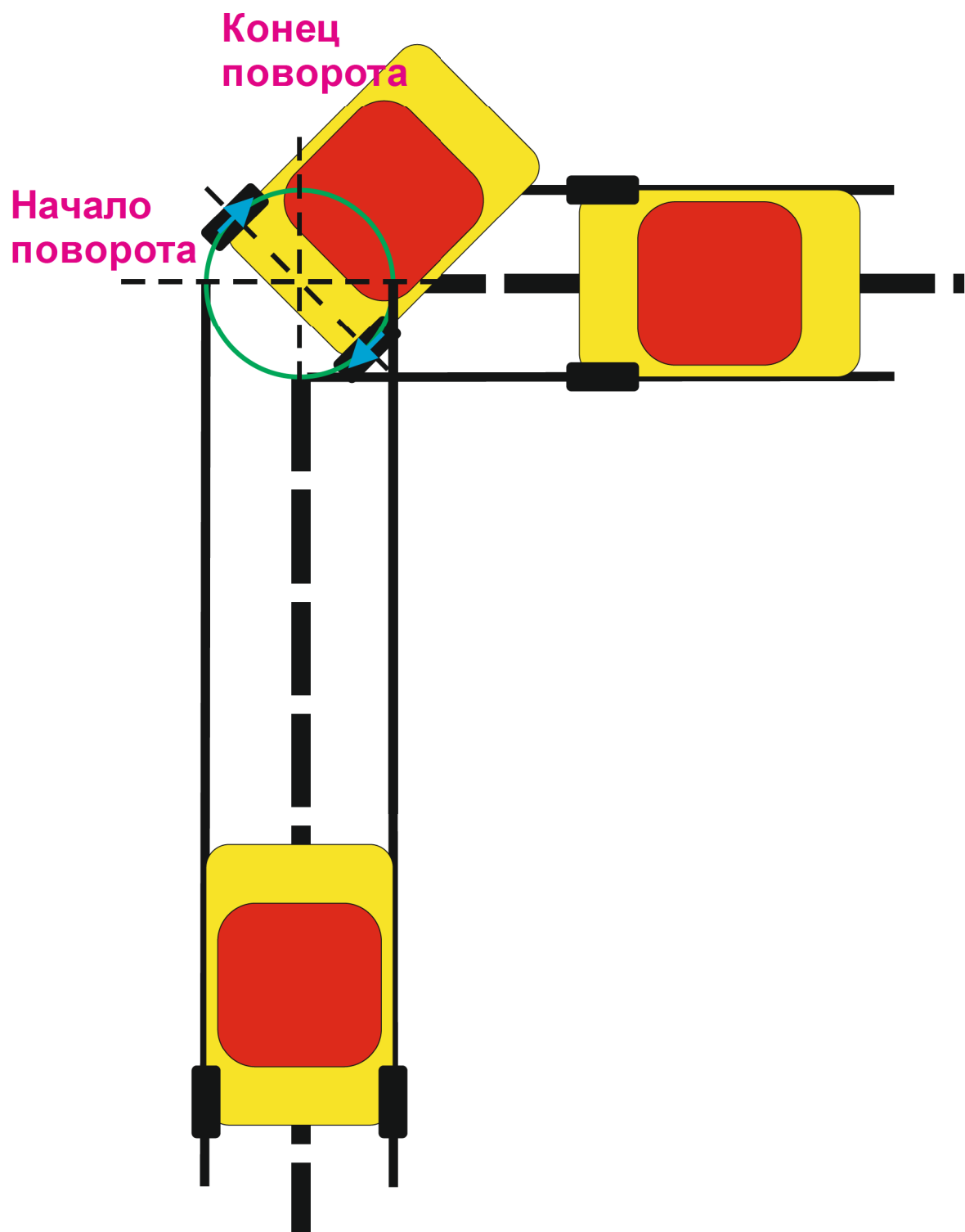
Скопируйте текст программы и вставьте его в Arduino IDE.
После проверки загрузите код в контроллер. Установите контроллер в блок R-5 и включите питание робота.

Вариант поворота №2

Во втором варианте для поворота мы будем вращать колеса в разные стороны. В этом случае значительно уменьшается радиус поворота. Робот разворачивается практически вокруг своей оси.

На рисунке видно, что радиус разворота равен половине расстояния между серединами колес.

Таким образом мы сможем посчитать время для поворота.
Половина расстояния между серединами колес (радиус поворота) –
Длина окружности поворота –
Четверть длины окружности –
Время прохождения участка поворота при средней измеренной скорости –



Пишем код.

```
#define DIR_R 2  
#define SPEED_R 3  
#define DIR_L 4  
#define SPEED_L 5
```

```
void setup()  
{  
  pinMode (DIR_R, OUTPUT);
```



```

pinMode (SPEED_R, OUTPUT);
pinMode (DIR_L, OUTPUT);
pinMode (SPEED_L, OUTPUT);

}

void loop()
{
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
  delay(1000); // Вращаем 1 сек

  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, LOW); // Низкий уровень запрещает вращение моторов
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, LOW);
  delay(2000); //останов 2 сек.

  //далее левое колесо вращаем 1 сек. вперед, правое колесо вращаем назад
  //после поворота направо робот сразу начинает движение вперед без паузы.
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, LOW);
  digitalWrite (SPEED_L, HIGH);
  delay(1000); // Вращаем 1 сек

  //вращаем оба колеса 1 сек.
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
  delay(1000); // Вращаем 1 сек

  //вращаем правое колесо 1 сек. назад, левое колесо вращаем вперед
  digitalWrite (DIR_R, LOW);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
  delay(1000); // Вращаем 1 сек
}

```

Скопируйте текст программы и вставьте его в Arduino IDE.
После проверки загрузите код в контроллер. Установите контроллер в блок R-5 и включите питание робота.

Измеряем максимальную скорость движения нашего робота.

Скорость транспортных средств измеряется величинами км/ч или м/с. Мы не сможем измерить скорость в км/ч, но мы можем попробовать измерить сколько метров проедет наш робот, скажем, за 10 секунд. А затем пересчитаем и в м/с и в км/ч. Это будет уже просто.

Для этого эксперимента используем код, который написали в предыдущем задании. Удалим из него часть, где робот движется в обратном направлении и установим время вращения моторов вперед 10 секунд. Можно поставить и меньшее время, если место испытаний ограничено. Но надо помнить, что чем большее время робот движется, тем точнее будет результат.

Пишем код.

```
#define DIR_R 2
#define SPEED_R 3
#define DIR_L 4
#define SPEED_L 5

void setup()
{
  pinMode (DIR_R, OUTPUT);
  pinMode (SPEED_R, OUTPUT);
  pinMode (DIR_L, OUTPUT);
  pinMode (SPEED_L, OUTPUT);
}

void loop()
{
  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, HIGH);
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, HIGH);
  delay(10000); // Вращаем 10 сек

  digitalWrite (DIR_R, HIGH);
  digitalWrite (SPEED_R, LOW); // Низкий уровень запрещает вращение моторов
  digitalWrite (DIR_L, HIGH);
  digitalWrite (SPEED_L, LOW);
  delay(5000); //останов 5 сек.
}
```

Скопируйте текст программы и вставьте его в Arduino IDE.

После проверки загрузите код в контроллер. Установите контроллер в блок R-5 и включите питание робота.

Контроллер Ардуино сам отсчитает 10 секунд. Нам останется только линейкой или рулеткой измерить пройденный путь.

На блоке R-5 есть переключатель **N – H**. Этот переключатель подает напряжение питания электромоторов. В положении **N** напряжение питания электромоторов подается со стабилизатора 5 Вольт. В этом положении скорость робота не очень высокая. Но стабильная. Не зависит от уровня разряда батареи. Поэтому все измерение скорости будет достаточно точным и стабильным во времени. В положении **H** напряжение питания подается непосредственно с батареи. И если установлены свежие качественные батареи, то напряжение составит $6 \times 1,5 = 9$ Вольт. В этом случае скорость робота будет выше. Но со временем из-за уменьшения заряда батареи скорость робота будет уменьшаться.

В нашем эксперименте пока не важна стабильность измерений скорости. Нам важнее измерить максимально возможную скорость движения робота. Поэтому измерение проведем, установив переключатель в положение **H**.

За 10 секунд робот проехал примерно 11 метров. Т.е скорость робота составила 1,1 м/с. Пересчитаем в километры и часы. Умножаем пройденный путь и время на 3600 и получаем скорость 3,96 км/ч.

Еще нам пригодится для будущих экспериментов знание такого параметра, как количество оборотов колеса в секунду. Зная скорость робота, количество оборотов в минуту, мы можем прокладывать путь робота, рассчитывать траекторию движения, места остановок, поворотов и др.

Для того, что бы вычислить этот параметр, нам необходимо рассчитать длину окружности колеса. Диаметр колеса составляет 42 мм.
Формула длины окружности:

$$L = \pi D$$

Подставляем значения.

$$L$$

$$= 3,14 \times 42.$$

Получаем, что длина окружности нашего колеса составляет 132 мм. Умножаем 11 метров, пройденных роботом за 10 секунд на 6 (в минуте 60 сек.). Затем делим на длину окружности колеса. $11000 \times 6 / 132 = 500$

Так же нам может пригодиться такой параметр, как время, за которое колесо сделает один оборот.

Скорость вращения колес – 500 об./мин.

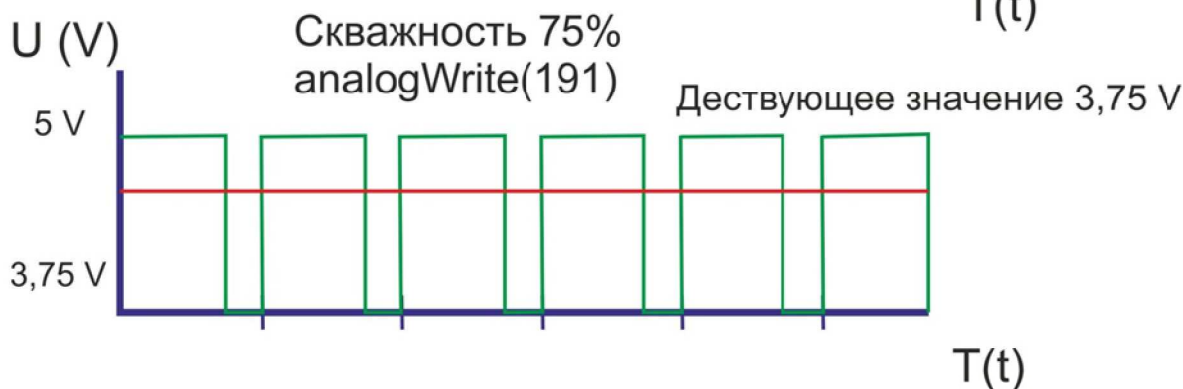
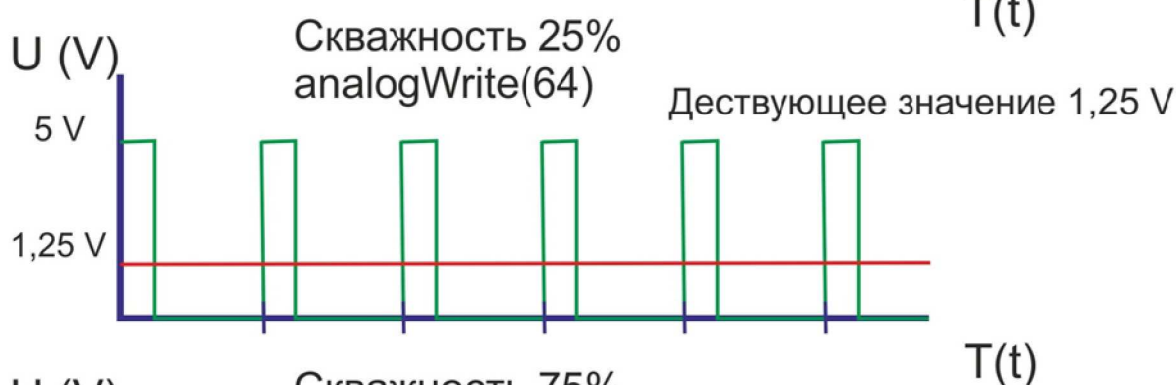
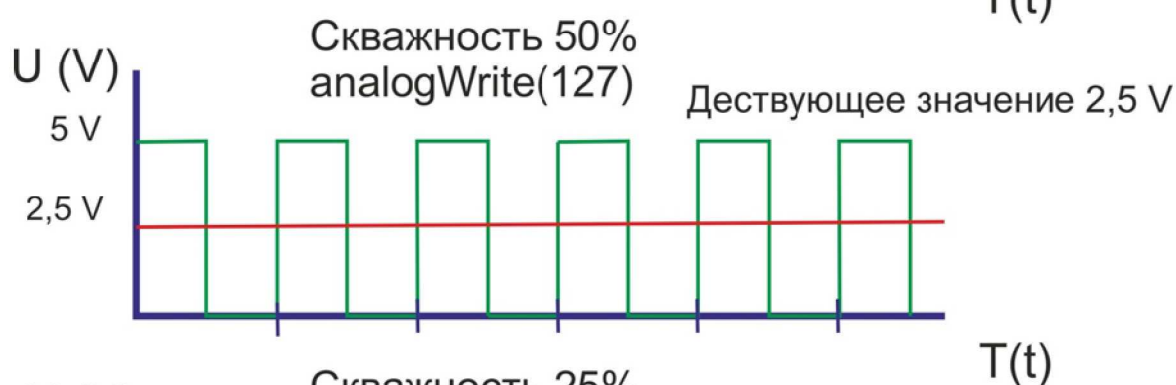
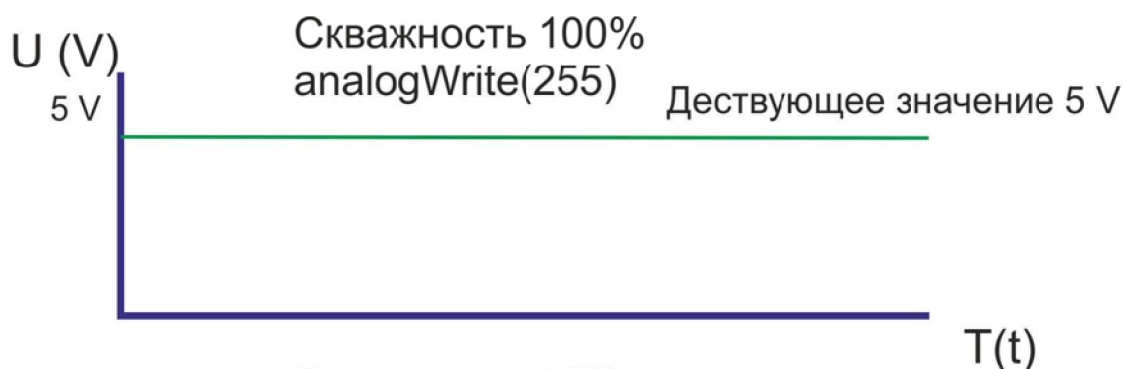
Длина окружности колеса – 132 мм.

Время, за которое колесо сделает один оборот (робот проедет 132 мм) – 120 миллисекунды.

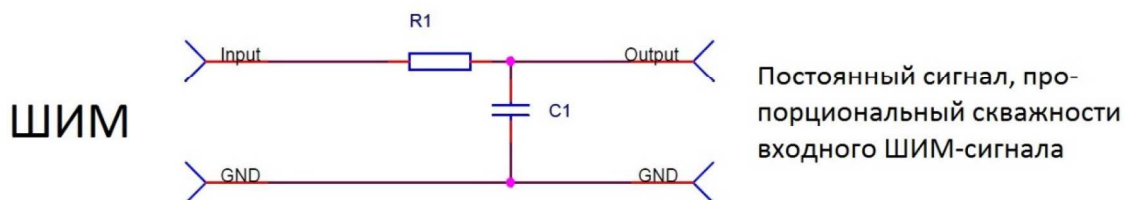
Теперь, зная эти кинематические характеристики нашего робота, мы сможем строить различные траектории. Рассчитывать остановки, повороты и пр. действия робота.

ШИМ. Регулировка скорости движения.

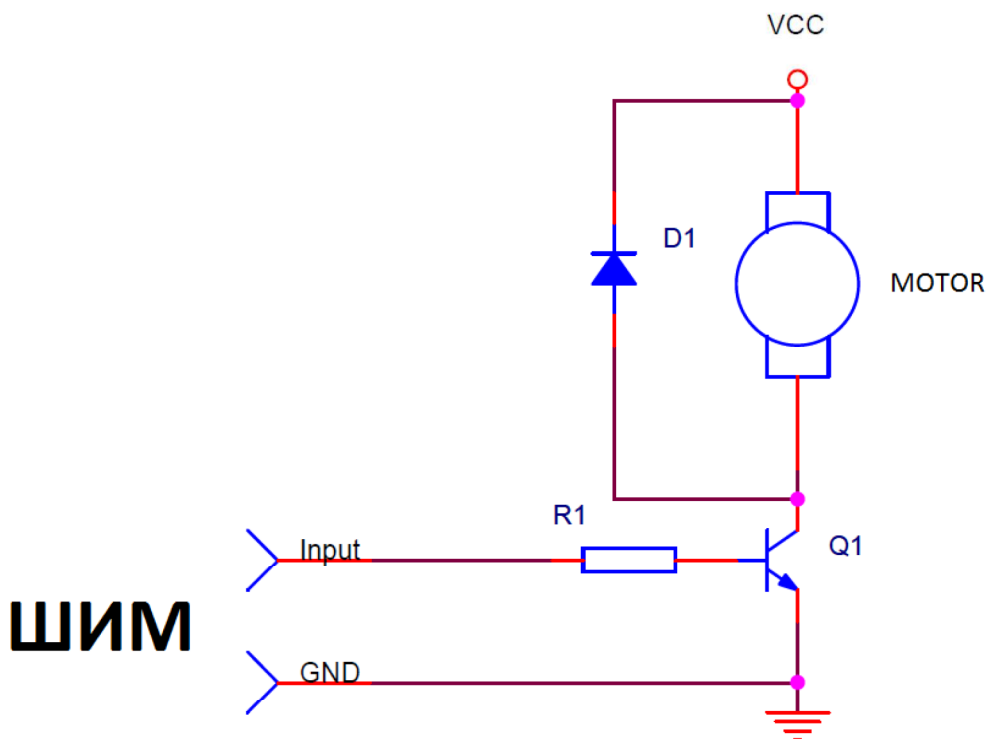
Метод широтно импульсной модуляции (ШИМ, англ. pulse-width modulation (PWM)) широко применяется для управления мощностью, подводимой к нагрузке, путём изменения скважности импульсов, при постоянной частоте.



Если на выходе устройства, генерирующего ШИМ сигнал установить фильтр низких частот (ФНЧ), то уровень постоянного напряжения на выходе фильтра будет определяться скважностью импульсов ШИМ. Фильтр не пропускает несущую частоту ШИМ. На выходе фильтра формируется сигнал с постоянной составляющей. Сам фильтр может состоять из простейшей интегрирующей RC цепи.



В нашем случае нагрузкой является электродвигатель. Он достаточно инерционен. Поэтому не требуется дополнительных фильтров. Таким образом, управляя длительностью «единицы» и «ноля» мы получаем необходимое нам значение аналогового сигнала, а в нашем случае напряжение питания электромотора.



Основным достоинством ШИМ-управления является то, что электронный ключ (обычно транзистор) работает в ключевом режиме, что заметно повышает

экономичность схемы, так как потери на активных элементах сводятся к минимуму.

В схеме управления электромотором с помощью ШИМ сигнала параллельно мотору необходимо устанавливать диод для защиты транзистора Q1 от индукционных всплесков, появляющимся в момент выключения и включения. Благодаря использованию диода, индукционный импульс разряжается через него и внутреннее сопротивление двигателя, защищая тем самым транзистор.

Для сглаживания всплесков питания между клеммами мотора, можно подключить конденсатор небольшой емкости (100nF). Это снизит помехи, создаваемые частыми переключениями транзистора Q1.

В контроллере Arduino есть специальные контакты, которые могут формировать сигнал с широтно-импульсной модуляцией. Это D3, D5, D6, D9, D11.

В блоке R-5 пины D3, D5 подключены к входам драйвера, отвечающих за регулировку скорости вращения электромоторов. Соответственно, вызвав функцию analogWrite() мы сможем изменять действующее значение напряжения питания электромоторов и соответственно изменять скорость движения робота.

Пишем код.

Пример 1. УСКОРЕНИЕ.

Последовательно увеличиваем значение ШИМ. С увеличением значения ШИМ скорость движения робота увеличивается.

```
#define DIR_R 2
#define SPEED_R 3
#define DIR_L 4
#define SPEED_L 5

void setup()
{
  pinMode (DIR_R, OUTPUT);
  pinMode (SPEED_R, OUTPUT);
  pinMode (DIR_L, OUTPUT);
  pinMode (SPEED_L, OUTPUT);
}

void loop()
{
  digitalWrite (DIR_R, HIGH);
  analogWrite (SPEED_R, 0); // Значение ШИМ равно нулю, двигатель не вращается
  digitalWrite (DIR_L, HIGH);
  analogWrite (SPEED_L, 0);
  delay(1000); // пауза 1 сек.

  digitalWrite (DIR_R, HIGH);
  analogWrite (SPEED_R, 70); // Значение ШИМ 70. Скорость низкая
  digitalWrite (DIR_L, HIGH);
```

```

analogWrite (SPEED_L, 100);
delay(2000);

digitalWrite (DIR_R, HIGH);
analogWrite (SPEED_R, 170); // Значение ШИМ 170. Скорость выше
digitalWrite (DIR_L, HIGH);
analogWrite (SPEED_L, 170);
delay(2000);

digitalWrite (DIR_R, HIGH);
analogWrite (SPEED_R, 255); // Значение ШИМ 255. Скорость максимальная
digitalWrite (DIR_L, HIGH);
analogWrite (SPEED_L, 255);
delay(2000);
}

```

Скопируйте текст программы и вставьте его в Arduino IDE.
После проверки загрузите код в контроллер. Установите контроллер в блок R-5 и включите питание робота.

Пример 2. ЗМЕЙКА.

Попеременно устанавливаем разное значение ШИМ для правого и левого колес. Соответственно колеса будут вращаться с разной скоростью, и наш робот будет двигаться змейкой.

```

#define DIR_R 2
#define SPEED_R 3
#define DIR_L 4
#define SPEED_L 5

void setup()
{
  pinMode (DIR_R, OUTPUT);
  pinMode (SPEED_R, OUTPUT);
  pinMode (DIR_L, OUTPUT);
  pinMode (SPEED_L, OUTPUT);
}

void loop()
{
  digitalWrite (DIR_R, HIGH);
  analogWrite (SPEED_R, 0);
  digitalWrite (DIR_L, HIGH);
  analogWrite (SPEED_L, 0);
  delay(1000); // пауза 1 сек.

  digitalWrite (DIR_R, HIGH);
  analogWrite (SPEED_R, 100);
}

```

```
digitalWrite (DIR_L, HIGH);  
analogWrite (SPEED_L, 255);  
delay(1000);
```

```
digitalWrite (DIR_R, HIGH);  
analogWrite (SPEED_R, 255);  
digitalWrite (DIR_L, HIGH);  
analogWrite (SPEED_L, 100);  
delay(1000);  
}
```

Пример 3. УСКОРЕНИЕ. ТОРМОЖЕНИЕ.

С помощью сигналов ШИМ мы можем плавно увеличивать и уменьшать скорость движения робота. Для этого нам необходимо воспользоваться оператором **for**. Конструкция, позволяющая нам последовательно увеличивать или уменьшать значение ШИМ, включает цикл **for (initialization; condition; increment)** и блок операторов, заключенный в скобки.

```
for (int i = 40; i < 255; i++)  
{  
    analogWrite (SPEED_R, i);  
    analogWrite (SPEED_L, i);  
    delay(50);  
}
```

Инициализация (Initialization) выполняется самой первой и один раз. Каждый раз в цикле проверяется **условие (condition)**, если оно верно, выполняется блок операторов и **приращение (increment)**, затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается.

Эта запись означает следующее:

К целочисленной переменной **i**, которая вначале равна 40 прибавляется **1 (i++)**, пока её значение меньше либо равно **255**. При каждом значении переменной **i** будут выполняться операции, записанные в фигурных скобках (тело цикла). Т.е. значение ШИМ каждые **50** мили секунд от значения **40** будет возрастать до значения **255**. Каждый раз в цикле проверяется условие **i<255**. Если условие, верно, то выполняется блок операторов и приращение значения **i**. Затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается. И наш робот начинает двигаться с постоянной максимальной скоростью.

Точно так же будет происходить и уменьшение скорости.

```
for (int i = 255; i > 0; i--)
```

Цикл **for** мы можем использовать так же и для записи однотипных команд. В блоке **setup** нам необходимо пины 2,3,4,5 определить, как выходы. И вместо команды для каждого пина, как мы делали ранее,

```
void setup()  
{  
    pinMode (DIR_R, OUTPUT);  
    pinMode (SPEED_R, OUTPUT);
```

```
pinMode (DIR_L, OUTPUT);  
pinMode (SPEED_L, OUTPUT);
```

пишем цикл, в котором в качестве переменной *i* указываем номера пинов.

```
void setup () {  
  for (int i=2; i<=5; i++) {  
    pinMode (i, OUTPUT);  
  }  
}
```

Пишем код.

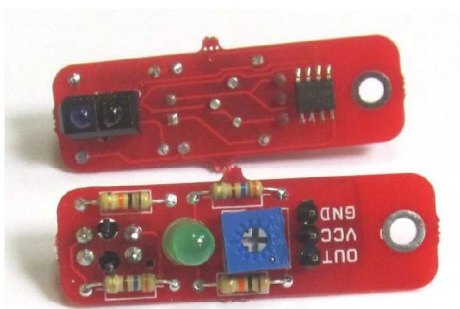
```
#define DIR_R 2  
#define SPEED_R 3  
#define DIR_L 4  
#define SPEED_L 5  
  
void setup () {  
  for (int i=2; i<=5; i++) {  
    pinMode (i, OUTPUT);  
  }  
  // укажем, что изначально моторы остановлены  
  digitalWrite (SPEED_R, LOW);  
  digitalWrite (SPEED_L, LOW);  
}  
  
void loop()  
{  
  digitalWrite (DIR_R, HIGH);  
  digitalWrite (DIR_L, HIGH);  
  for (int i = 40; i < 255; i++)  
  {  
    analogWrite (SPEED_R, i);  
    analogWrite (SPEED_L, i);  
    delay(50);  
  }  
  for (int i = 255; i > 0; i--)  
  {  
    analogWrite (SPEED_R, i);  
    analogWrite (SPEED_L, i);  
    delay(50);  
  }  
  digitalWrite (SPEED_R, LOW);  
  digitalWrite (SPEED_L, LOW);  
  delay(1000);  
}
```

Движение робота по черной линии.

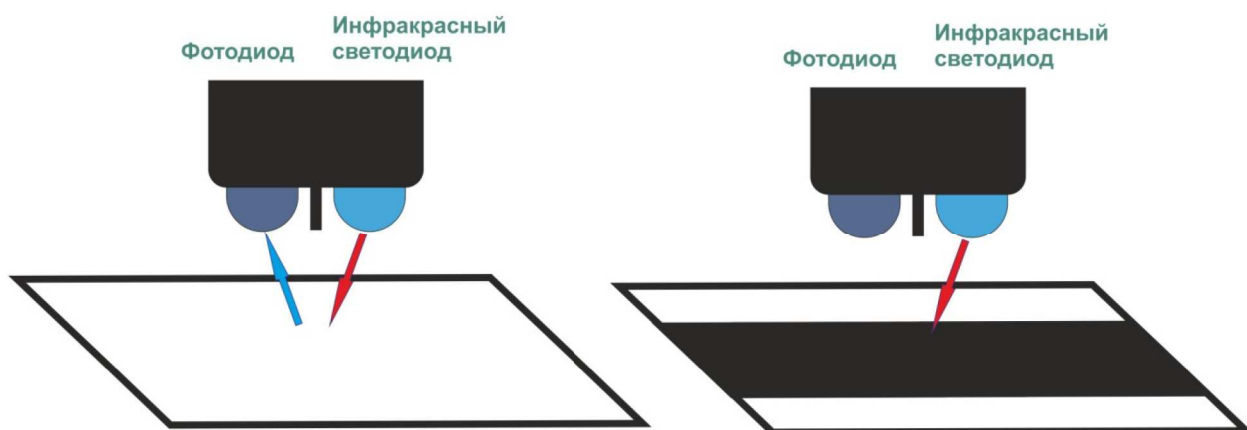
Движение робота по линии - самый популярный эксперимент начинающего роботостроителя.

Для того, что бы наш робот мог двигаться по полю с черной линией нам необходимы уже датчики, которые смогли бы различать белое и черное поля. Эти датчики называются датчиками линии. В их задачу входит определять наличие белого или черного поля и сообщать об этом контроллеру.

Внешний вид датчика



Принцип работы датчиков



Для определения белого или черного поля используется электронный компонент, состоящий из инфракрасного светодиода (ИК) и инфракрасного фотодиода. ИК светодиод излучает свет в инфракрасном диапазоне с длиной волны порядка 940 нанометра. Такой диапазон света используется для защиты от помех, создаваемых осветительными лампами, солнечным светом.

Рассмотрим два варианта возможных событий.

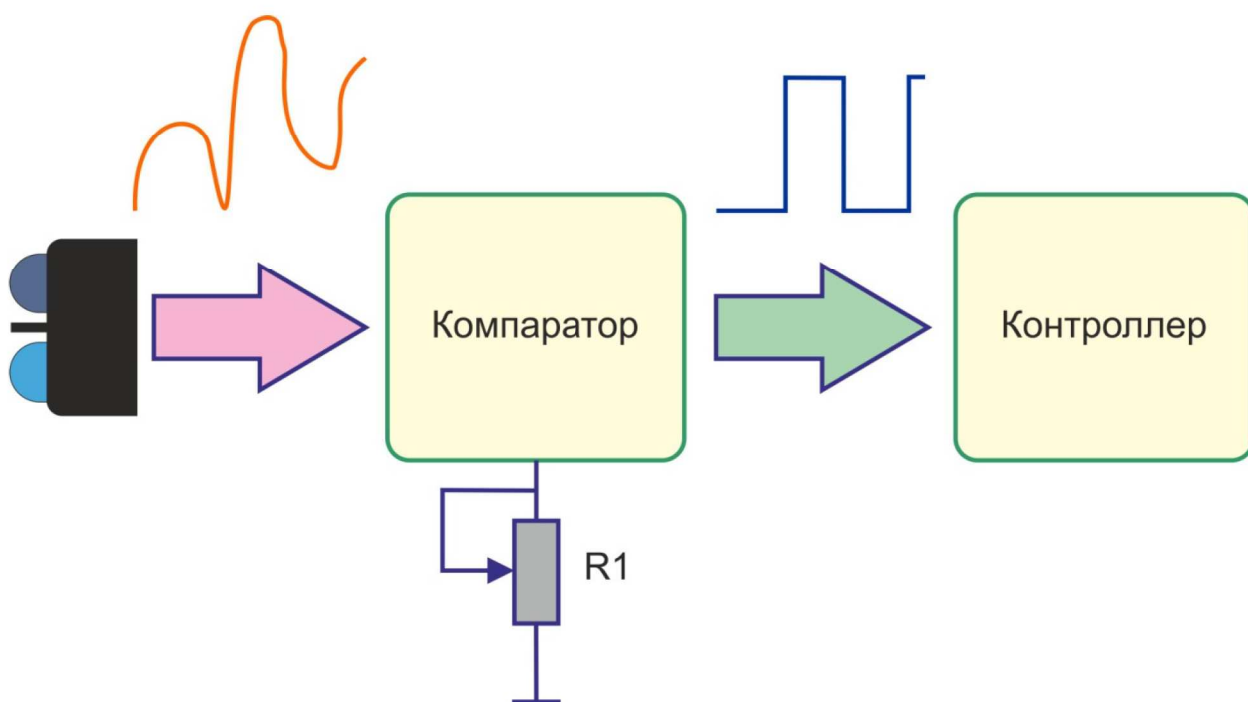
1. Датчик находится над белым полем.
Отраженный от белого поля свет попадает на фотодиод. Далее уже оцифрованный компаратором электрический сигнал поступает на вход

контроллера. Контроллер исходя из написанной программы воспринимает этот сигнал именно, как белое поле.

2. Датчик находится над черным полем.

Как мы знаем из физики, черное поле поглощает свет. Отражения не происходит. Такое состояние датчика контроллером воспринимается, как наличие черного поля.

На выходе фотодиода формируется хаотичный аналоговый сигнал. Для повышения помехозащищенности в схему датчика включен компаратор, который отсеивает помехи, ложные отражения и др. Порог срабатывания компаратора устанавливается резистором R1. С выхода компаратора мы получаем сигнал в цифровом виде.



Подключение датчиков

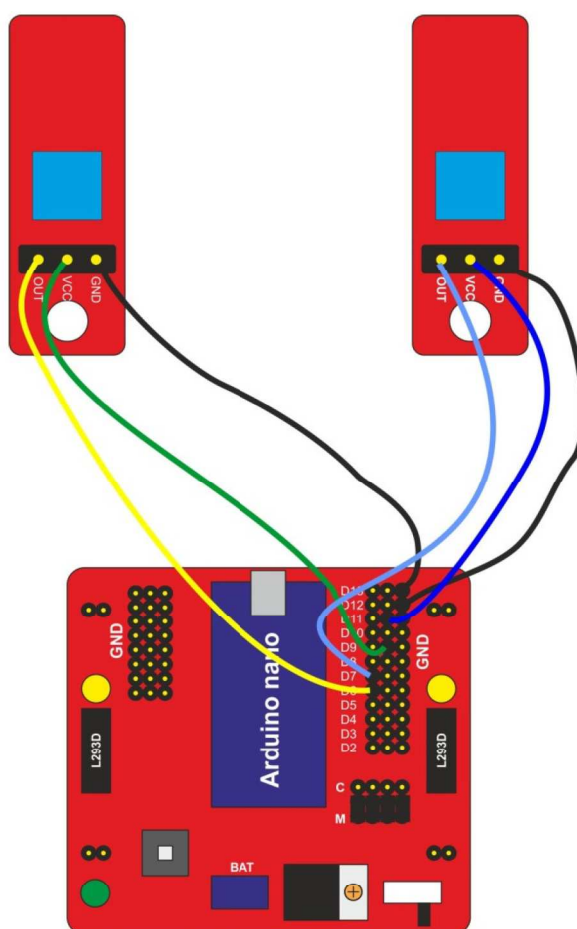
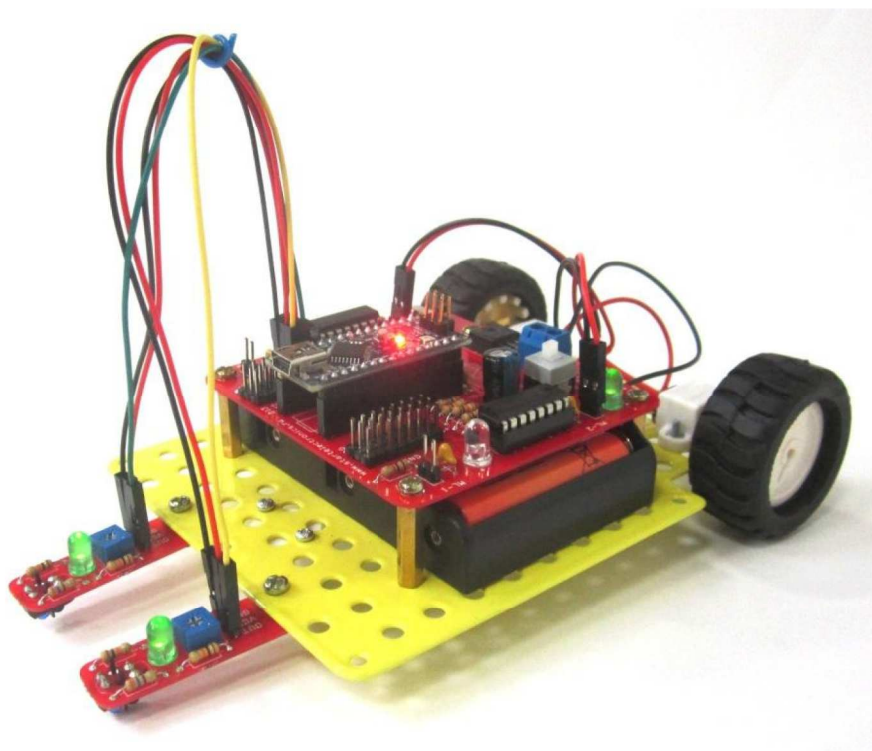
Устанавливаем датчики линии и подключаем их к блоку управления R-5. Контакты GND на датчиках соединяем проводками с контактами GND на блоке R-5 (в разъеме J3 правый ряд контактов). Контакт VCC на датчиках соединяем с контактами +5V на блоке R-5 (средний ряд контактов). Контакты OUT левого и правого датчиков подключаем к контактам разъема J3. К каким контактам подключим выходы датчиков, такие и надо будет нам указывать в нашей программе.

Помним, что контакты D2-D5 у нас заняты уже драйвером моторов.

Подключим датчики следующим образом.

Левый датчик – D6

Правый датчик – D7



Пишем код.

Перед написанием кода рассмотрим на рисунке ситуации, которые могут возникать при движении робота.

ШАГ1. Датчики робота «видят» белое поле. Контроллер принимает решение включить оба мотора и двигаться прямо.

ШАГ2. Робот правым датчиком наезжает на черную линию. Для выравнивания положения необходимо притормозить правое колесо.

ШАГ3. Робот наезжает левым датчиком на черную линию. Для выравнивания положения необходимо притормозить вращение левого колеса.



В этом эксперименте поворот в сторону линии сделаем следующим образом.

При наезде правым датчиком на черную линию мы останавливаем правый мотор.
Выравнивание хода осуществляется левым мотором.
При наезде левым датчиком на черную линию мы останавливаем левый мотор
Выравнивание хода осуществляется правым мотором.
Если оба датчика оказываются над черной линией, то робот продолжает движение в прямом направлении

```
#define LS 6    // левый датчик
#define RS 7    // правый датчик

#define DIR_R 2
#define SPEED_R 3
#define DIR_L 4
#define SPEED_L 5

void setup()
{
  pinMode(LS, INPUT);
  pinMode(RS, INPUT);
  pinMode(DIR_R, OUTPUT);
  pinMode(SPEED_R, OUTPUT);
  pinMode(DIR_L, OUTPUT);
  pinMode(SPEED_L, OUTPUT);
}
void loop()
{
  if(digitalRead(LS==HIGH) && digitalRead(RS==HIGH))    // Move Forward
  {
    digitalWrite(DIR_R, HIGH);
    digitalWrite(SPEED_R, HIGH);
    digitalWrite(DIR_L, HIGH);
    digitalWrite(SPEED_L, HIGH);
  }

  if((digitalRead(LS==LOW)) && digitalRead(RS==HIGH))    // Turn right
  {
    digitalWrite(DIR_R, HIGH);
    digitalWrite(SPEED_R, LOW);
    digitalWrite(DIR_L, HIGH);
    digitalWrite(SPEED_L, HIGH);
  }

  if(digitalRead(LS==HIGH) && (digitalRead(RS==LOW))    // turn left
  {
    digitalWrite(DIR_R, HIGH);
    digitalWrite(SPEED_R, HIGH);
    digitalWrite(DIR_L, HIGH);
    digitalWrite(SPEED_L, LOW);
  }
}
```

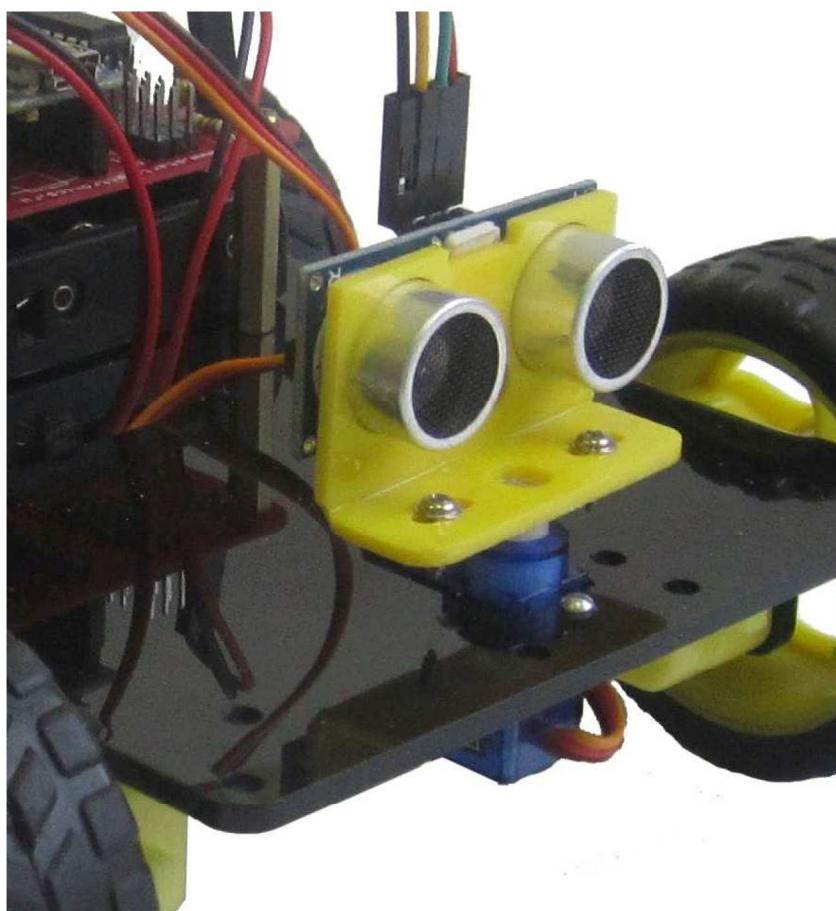
```
if(digitalRead(LS==LOW) && (digitalRead(RS==LOW))    // stop
{ digitalWrite(DIR_R , HIGH);
  digitalWrite(SPEED_R ,HIGH);
  digitalWrite(DIR_L , HIGH);
  digitalWrite(SPEED_L ,HIGH);
}
}
```

Подключение ультразвукового датчика HC-SR04



Датчик HC-SR-04 состоит из передатчика, приемника и схемы управления.

Для наиболее оптимального режима поиска препятствий датчик устанавливается на сервомотор SG90. Это позволяет искать и обнаруживать препятствия не только по направлению движения робота, но и с левой и с правой сторон от курса движения.



Подключение сервомотора, датчика к блоку управления R-5

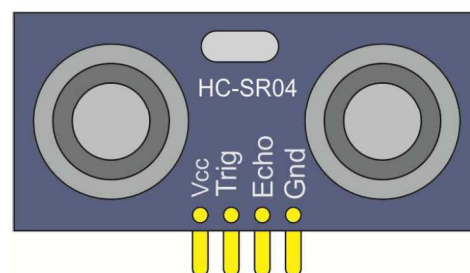
Назначение выводов:

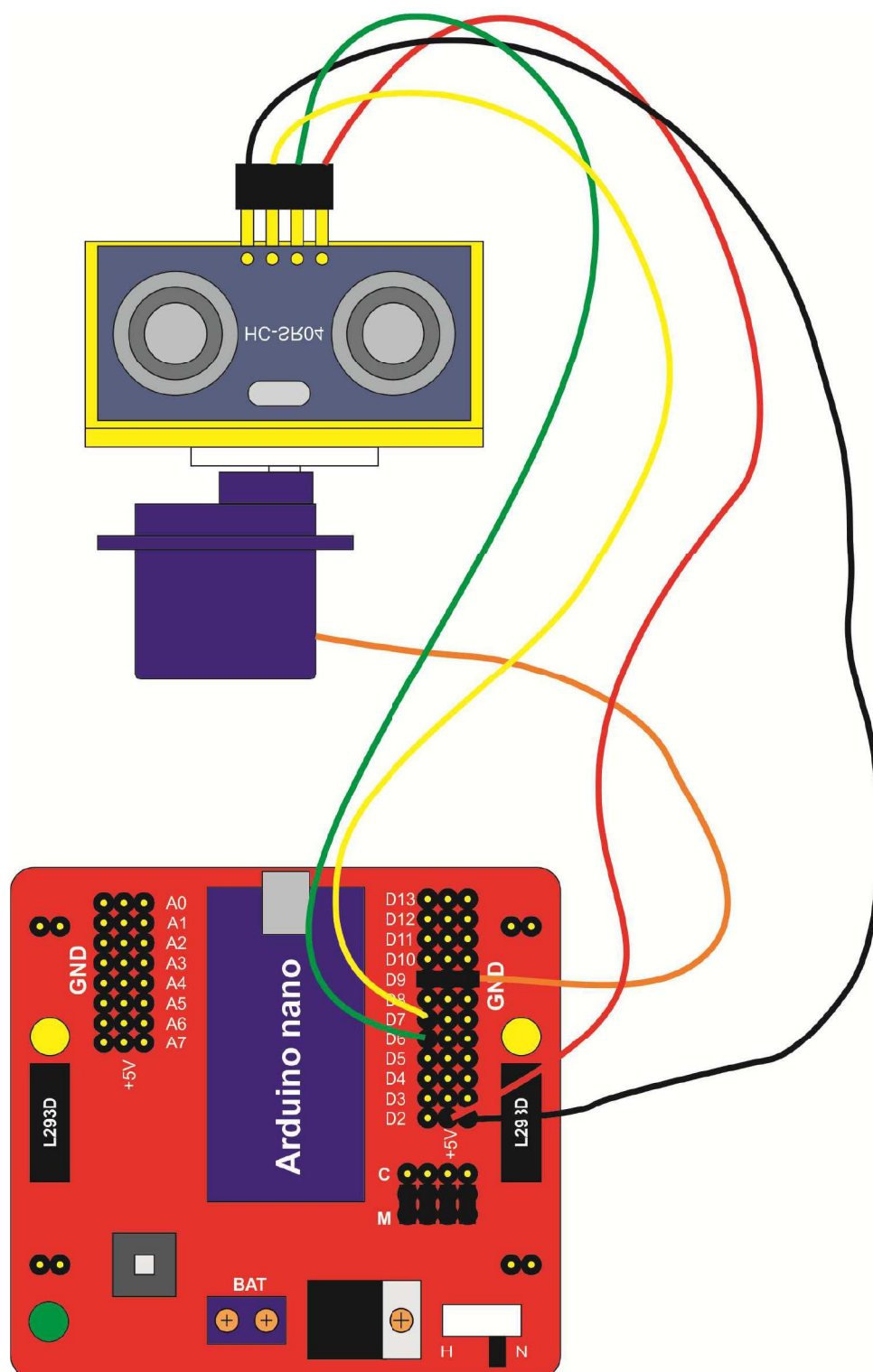
Gnd – «земля». «Минус» батареи питания.

Vcc – «плюс» батареи питания.

Trig – вход сигнала управления датчиком.

Echo – выход сигнала измерения расстояния.





Подключение сервомотора и УЗ датчика определяется кодом программы. В программе мы определили следующее:

Сервомотор подключается к контакту D9 (коричневый провод к контакту GND, желтый провод к D6).

Датчик HC-SR-06

Vcc – к любому контакту в среднем ряду

Trig – к контакту D6

Echo – к контакту D7

Gnd – к любому контакту в правом ряду

Код программы:

Для работы программы необходимо подключить две библиотеки.

```
#include <Servo.h>
#include <NewPing.h>
```

```
//Below are the symbolic constants. Instead of having to type in a non-sensical pin number each time
we want to do something we can write an easy to understand name which represents the pin, the
compiler will then replace the names with the numbers
```

```
#define MOTOR_R 2
#define SPEED_R 3
#define MOTOR_L 4
#define SPEED_L 5
#define USTrigger 6
#define USEcho 7
#define MaxDistance 100
#define LED 13
```

```
//Here we have created two 'objects', one for the servo and one for the ultrasonic sensor
Servo servo;
NewPing sonar(USTrigger, USEcho, MaxDistance);
```

```
//Below we are creating unsigned integer variables which we will use later on in the code. They are
unsigned as they will only have positive values
```

```
unsigned int duration;
unsigned int distance;
unsigned int FrontDistance;
unsigned int LeftDistance;
unsigned int RightDistance;
unsigned int Time;
unsigned int CollisionCounter;
```

```
void setup()      //This block happens once on startup
{
```

```
  Serial.begin(9600); //I have included the serial initialize but commented it out, if you want to debug
and print information to the serial monitor just uncomment
```

```
  //Here we are setting the pin modes. As we will be sending out signals from the pins we set them as
  outputs
```

```
  pinMode(MOTOR_R, OUTPUT);
  pinMode(SPEED_R, OUTPUT);
  pinMode(MOTOR_L, OUTPUT);
  pinMode(SPEED_L, OUTPUT);
  pinMode(LED, OUTPUT);
```

```
  servo.attach(9);          //The servo is attached to pin 9
}
```

```
void loop()              //This block repeats itself while the Arduino is turned on
{
```

```
  servo.write(90);        //Rotate the servo to face the front
```

```

scan();                                //Go to the scan function
FrontDistance = distance;               //Set the variable FrontDistance to the value of the distance
returned from the scan function
Serial.println("Front distance = ");
Serial.print(distance);
if(FrontDistance > 60 || FrontDistance == 0)    //If there is nothing in front of the robot within 15cm
or the distance value is 0 (which for the newping library means no ping was returned) then...
{
    moveForward();                       //Go to the moveForward function
}
else                                       //Else (if there is something in front of the robot within 40cm) then...
{
    CollisionCounter = CollisionCounter + 1;
    moveStop();                         //Go to the moveStop function
    navigate();
}
}

void moveForward()                      //This function tells the robot to go forward
{
    Serial.println("");
    Serial.println("Moving forward");
    //digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(MOTOR_R, HIGH);
    digitalWrite(SPEED_R, HIGH);
    digitalWrite(MOTOR_L, HIGH);
    digitalWrite(SPEED_L, HIGH);
}

void moveBackward()                    //This function tells the robot to move backward
{
    Serial.println("");
    Serial.println("Moving backward");
    digitalWrite(MOTOR_R, LOW);
    digitalWrite(SPEED_R, HIGH);
    digitalWrite(MOTOR_L, LOW);
    digitalWrite(SPEED_L, HIGH);
}

void moveLeft()                        //This function tells the robot to turn left
{
    Serial.println("");
    Serial.println("Moving left");
    digitalWrite(MOTOR_R, HIGH);
    digitalWrite(SPEED_R, HIGH);
    digitalWrite(MOTOR_L, LOW);
    digitalWrite(SPEED_L, HIGH);
}

void moveRight()                      //This function tells the robot to turn right
{

```

```

Serial.println("");
Serial.println("Moving right");
digitalWrite(MOTOR_R, LOW);
digitalWrite(SPEED_R, HIGH);
digitalWrite(MOTOR_L, HIGH);
digitalWrite(SPEED_L, HIGH);
}

void moveStop()                                //This function tells the robot to stop moving
{
    Serial.println("");
    Serial.println("Stopping");
    digitalWrite(SPEED_R, LOW);
    digitalWrite(SPEED_L, LOW);
    //digitalWrite(RightMotorForward, LOW);
    // digitalWrite(RightMotorBackward, LOW);
}

void scan()                                    //This function determines the distance things are away from the
ultrasonic sensor
{
    Serial.println("");
    Serial.println("Scanning");
    Time = sonar.ping();
    distance = Time / US_ROUNDTRIP_CM;
    delay(500);
}

void navigate()
{
    Serial.println("There's an obstacle!");
    servo.write(167);                          //Move the servo to the left (my little servos didn't like going to
180 so I played around with the value until it worked nicely)
    delay(1000);                               //Wait half a second for the servo to get there
    scan();                                    //Go to the scan function
    LeftDistance = distance;                   //Set the variable LeftDistance to the distance on the left
    Serial.println("Left distance = ");
    Serial.print(distance);
    servo.write(0);                           //Move the servo to the right
    delay(1000);                               //Wait half a second for the servo to get there
    scan();                                    //Go to the scan function
    RightDistance = distance;                  //Set the variable RightDistance to the distance on the right
    Serial.println("Right distance = ");
    Serial.print(distance);
    if(abs(RightDistance - LeftDistance) < 5)
    {
        moveBackward();                       //Go to the moveBackward function
        delay(200);                           //Pause the program for 200 milliseconds to let the robot reverse
        moveRight();                           //Go to the moveRight function
        delay(100);                           //Pause the program for 200 milliseconds to let the robot turn right
    }
    else if(RightDistance < LeftDistance)      //If the distance on the right is less than that on the left
    then...
    {

```

```
    moveLeft();           //Go to the moveLeft function
    delay(100);           //Pause the program for half a second to let the robot turn
}
else if(LeftDistance < RightDistance) //Else if the distance on the left is less than that on the
right then...
{
    moveRight();          //Go to the moveRight function
    delay(100);           //Pause the program for half a second to let the robot turn
}
}
```