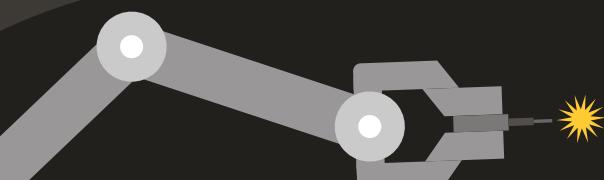
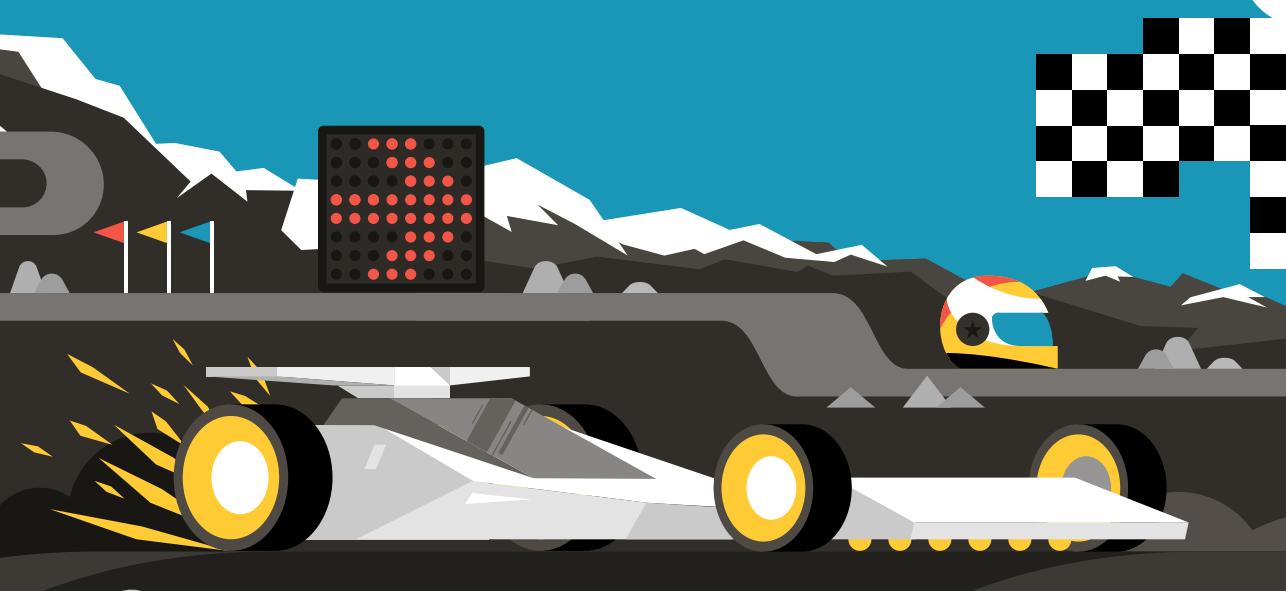


ДРАГСТЕР

ГОНОЧНЫЙ БОЛИД НА ARDUINO



DRAGSTER.AMPERKA.RU

ПРИВЕТ!

Перед тобой набор «Драгстер».

Драгстер – гоночный болид с большими задними колёсами.

Такие используются для заездов на время на длинных прямых трассах. В этом наборе ты будешь программировать собственный гоночный болид, чтобы научить его побеждать в соревнованиях по езде вдоль линии.

Проходи эксперименты последовательно, один за другим.

Так будет гораздо проще освоить программирование на языке C++ в среде Arduino. Кроме того, так ты не пропустишь важную информацию, необходимую для прохождения набора.

**увидимся
на финиш**

СОДЕРЖАНИЕ

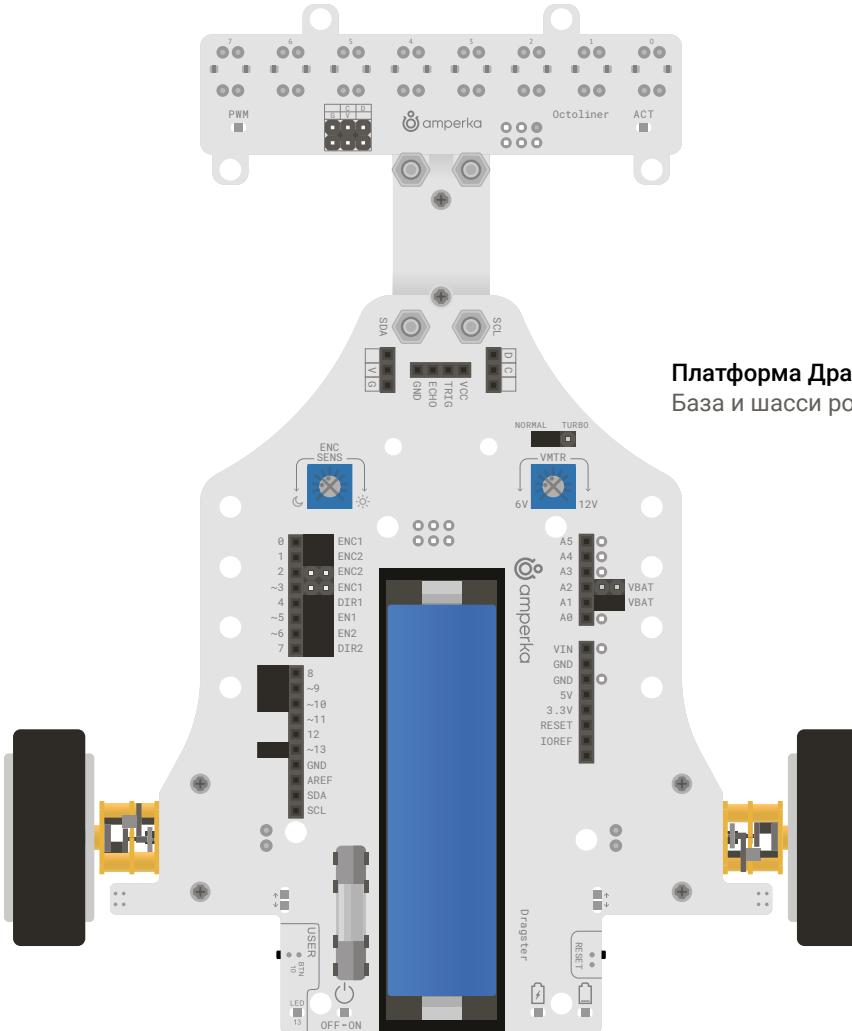
Гараж	4
Драгстер	6
Iskra Neo	8
Питание и зарядка	10
Установка Arduino IDE	12
Эксперименты	15

№1 Запуск	16
№2 Индикация	20
№3 Ключ на старт	22
№4 Прогрев двигателей	26
№5 Змейка	30
№6 Тюнинг. Библиотеки	32
№7 Стоп-сигнал	34
№8 Поворотники	36
№9 Реклама спонсоров	38
№10 Уровень заряда	44
№11 Энкодер	50
№12 Умные энкодеры	52
№13 Одометр	56
№14 Спидометр	56
№15 Тюнинг. Домашняя трасса	68
№16 Сканер дорожного полотна	60
№17 Драг-рейсинг	64
№18 Следование по линии	66
№19 Тюнинг. ПИД-регулятор	72
№20 Предсказания	74
№21 Накопление	76

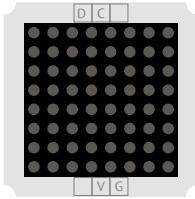


E!

ГАРАЖ

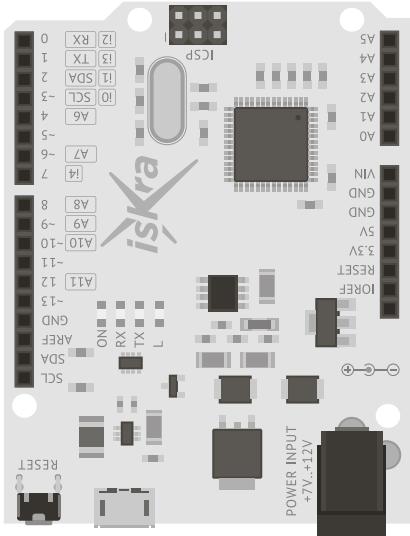


Платформа Драгстер
База и шасси робота.



Iskra Neo

Мозг робота. Аналог Arduino Leonardo, но способна выдать больший ток для работы модулей.

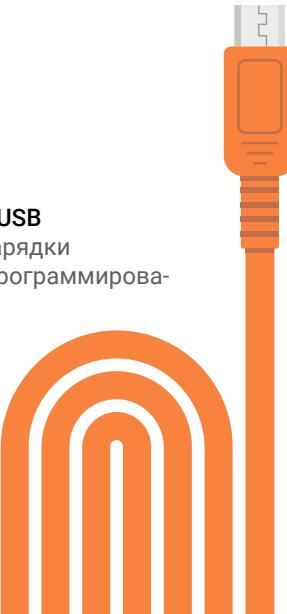


Светодиодная матрица 8×8

Выводит символы, знаки
и картинки.

Кабель micro-USB

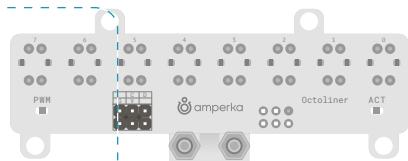
Служит для зарядки
Драгстера и программирова-
ния Iskra Neo.



ДРАГСТЕР

Ручка чувствительности энкодеров

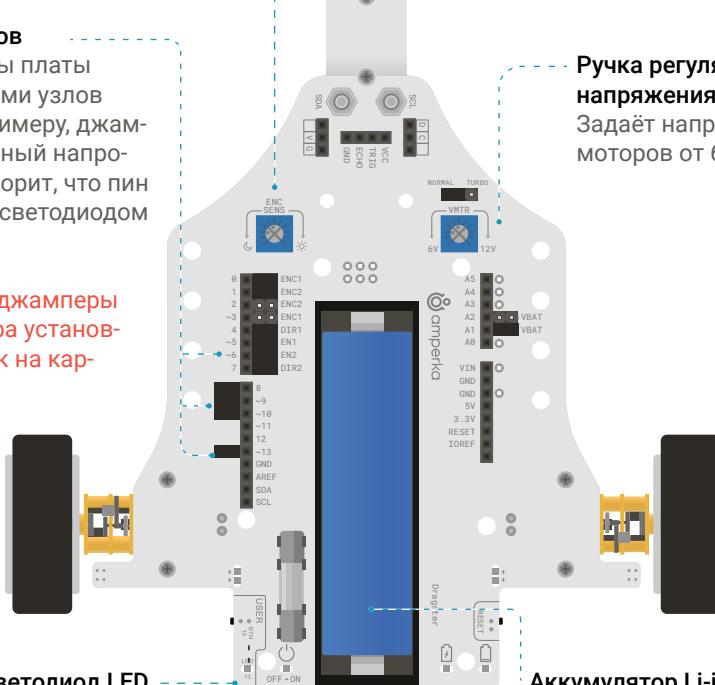
Позволяет энкодерам стабильно работать в помещениях с разной освещённостью.



Джамперы пинов

Соединяют пины платы Iskra Neo с пинами узлов Драгстера. К примеру, джампер, установленный напротив пина 13, говорит, что пин 13 соединён со светодиодом Драгстера.

⚡ Убедись, что джамперы твоего Драгстера установлены так же, как на картинке.



Кнопка BTN и светодиод LED

Используются для удобства во время отладки кода или во время гонок.

Ручка регулятора напряжения

Задаёт напряжение питания моторов от 6 до 12 вольт.

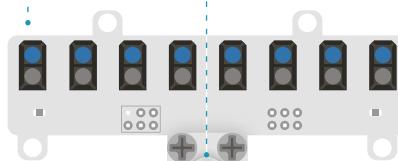
Аккумулятор Li-ion

В форм-факторе 18650. Выдаёт напряжение от 3,2 до 4,2 вольта.

Zelo-модуль

8 датчиков линии

Помогает работе двигаться вдоль линии и не терять её.

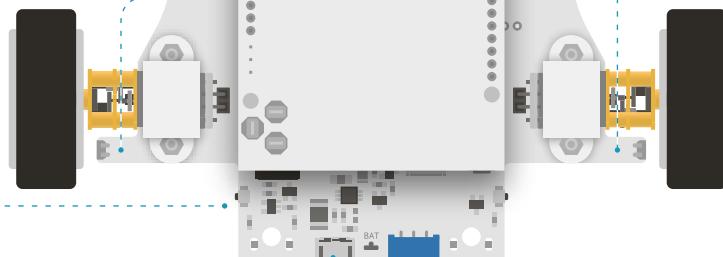


Кнопка RESET

Это кнопка управления защищой аккумулятора. Защита автоматически отключает Драгстера, если батарея в опасности.

Если защита сработала и Драгстер отключился, устрани проблему и нажми на эту кнопку, чтобы перезапустить защиту.

 А ещё эту кнопку нужно нажимать каждый раз после того, как ты вставил батарею!



Разъём зарядки

Подойдёт любой кабель micro-USB и блок питания.

Переходная плата с шаровой опорой

Передаёт сигнал и питание на модуль датчиков линии, а также служит Драгстеру третьей точкой опоры.

Плата Iskra Neo

Мозг Драгстера. Обрати внимание, у неё есть собственный разъём micro-USB специально для загрузки кода!

Энкодеры ENC1, ENC2

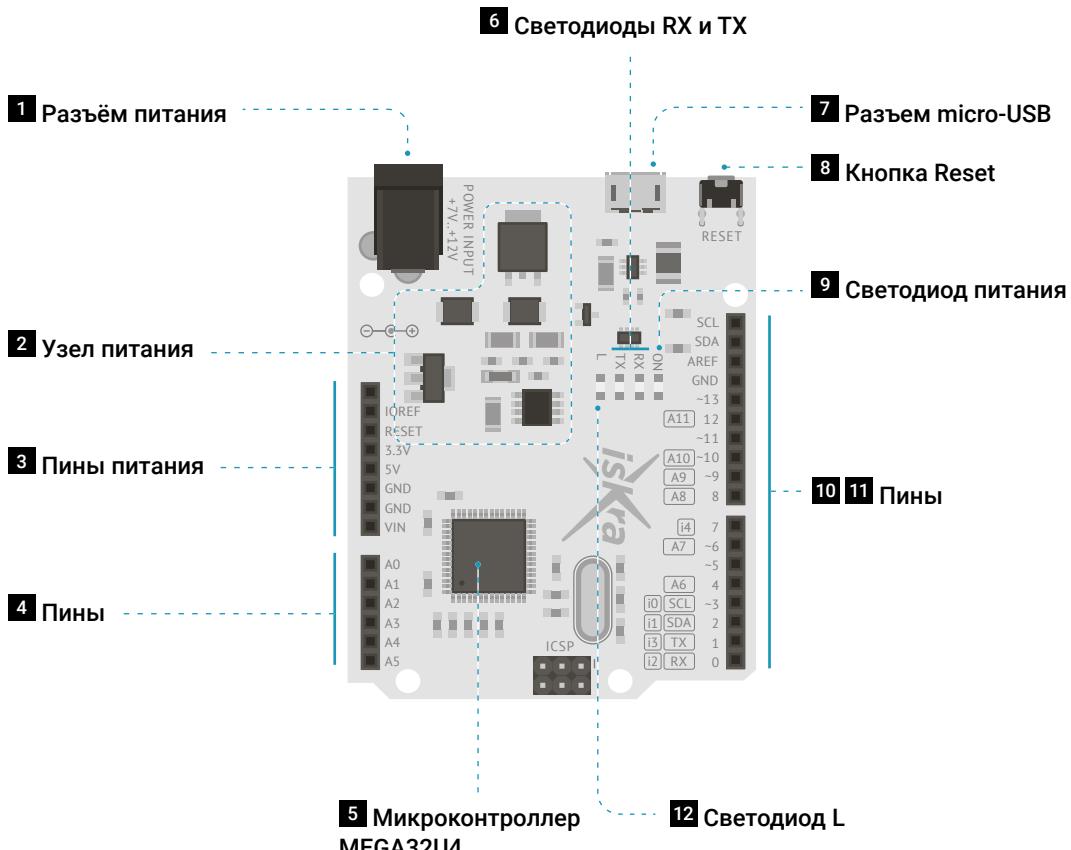
Позволяют контролировать скорость и пройденный путь. Датчики работают на отражении инфракрасного света от колёс.

Тумблер питания

Включает и выключает робота.

ISKRA NEO

Iskra Neo – плата для работы с мозгом цифрового устройства – микроконтроллером. На плате предусмотрены разъёмы, контактные колодки, светодиоды и узел питания.



1 Разъём питания

Круглый разъём (бочка, Barrel Jack) для питания платы напряжением от 7 до 12 вольт.

2 Узел питания

Понижает входное напряжение платы до 5 вольт, чтобы микроконтроллер не вышел из строя от перенапряжения. Также узел выдаёт 3,3 вольта для питания некоторых модулей.

3 Пины питания:

- **VIN** — пин входного источника питания с напряжением 7...12 вольт;
- **GND** — земля, общий провод, ноль;
- **5V, 3.3V** — пины источников питания 5 и 3,3 вольта;
- **RESET** — пин сброса питания и программы;
- **IREF** — пин опорного напряжения аналогового сигнала.

4 Пины A0 ... A11

Предназначены для работы с аналоговыми датчиками. Умеют считывать напряжение от 0 до 5 вольт и преобразовывать его в цифровой вид.

5 Микроконтроллер MEGA32U4

Вычислительный блок, исполняющий код. Металлические контакты по бокам корпуса разведены к источникам питания, светодиодам индикации, кнопке сброса и к пинам ввода-вывода.

6 Светодиоды RX и TX

Включаются и мигают, когда плата обменивается данными с компьютером по кабелю micro-USB.

7 Разъём micro-USB

Соединяет плату с компьютером для программирования (прошивки), отладки кода и подачи питания.

8 Кнопка RESET

Перезагружает микро-контроллер и запускает программу заново.

9 Светодиод питания

Горит, когда на плату подано напряжение.

10 Пины со знаком ~

(тильда): **3, 5, 6, 9, 10, 11, 13**

Способны выдавать ШИМ-сигнал, то есть очень быстро переключаться, не отнимая ресурсов микроконтроллера.

11 Пины прерываний i0 ... i4

Позволяют следить за напряжением и мгновенно реагировать на его изменение.

4 10 11

Пины ввода-вывода

С их помощью Iskra Neo общается с внешними модулями: драйвером моторов, матрицей 8×8, энкодерами и датчиками линии.

12 Светодиод L

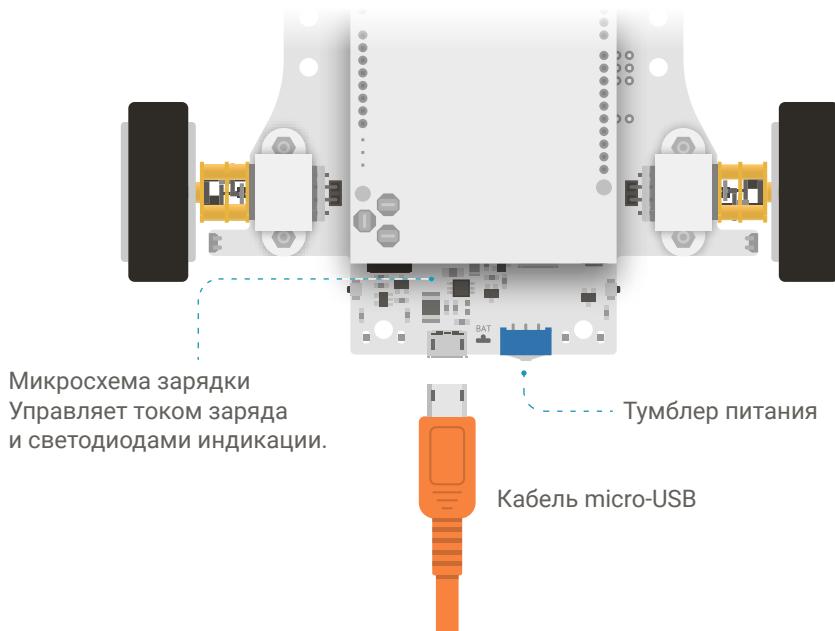
Соединён с пином 13 микроконтроллера. Им можно управлять напрямую из программы.

ПИТАНИЕ И ЗАРЯДКА

Драгстер питается от литиевого аккумулятора с напряжением от 3,2 до 4,2 вольта. Это напряжение повышается на преобразователе до уровня, заданного перемычкой Turbo и ручкой регулятора VMTR. Повышенное напряжение питает двигатели и Iskra Neo.

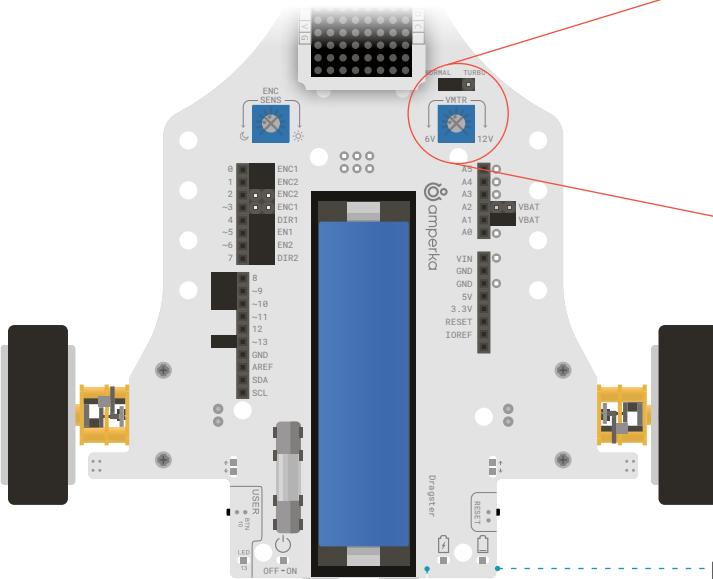
На плате Драгстера предусмотрена микросхема заряда-разряда литиевого аккумулятора. Подключи робота к блоку питания кабелем micro-USB и заряжай, как обычный телефон или другой гаджет!

Рекомендуем выключать Драгстер во время зарядки аккумулятора. Так он наполнит баки гораздо быстрее.



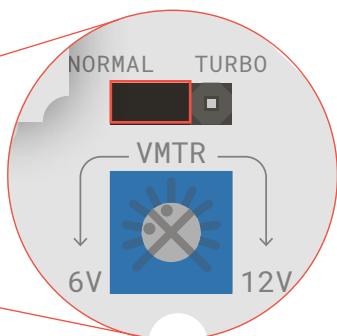
Кто не любит быстрой езды? Мы добавили режим Turbo специально для соревнований и сорвиголов! Включить его можно с помощью джампера Normal/Turbo. В положении перемычки Turbo можно регулировать напряжение на моторах с помощью подстроечного резистора.

 Мы рекомендуем установить перемычку в положение Normal. Режим Turbo предназначен для участия в соревнованиях. В этом режиме существенно увеличивается износ моторов и их редукторов, от чего они выходят из строя. Используйте режим Turbo на свой страх и риск, этот режим не является штатным для Драгстера. (Но один раз попробовать можно)



Индикатор питания
Горит синим, когда переключатель установлен в On.

Индикатор зарядки
Мигает жёлтым во время зарядки аккумулятора.



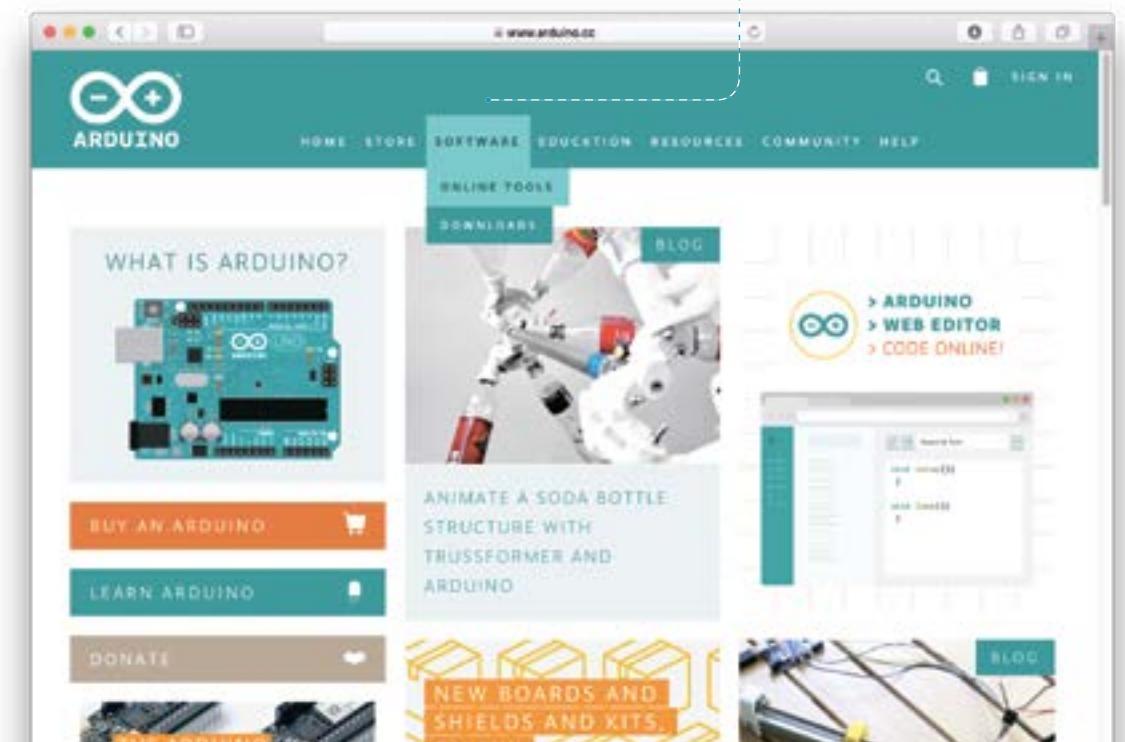
Обрати внимание, на серой ручке есть две крошечные точки. Они указывают, в каком положении находится потенциометр.

Индикатор разряда
Горит красным, когда уровень заряда низок.

УСТАНОВКА ARDUINO IDE

Arduino IDE – среда разработки программ для Arduino и Iskra Neo. Скачай и установи её на свой компьютер.

- 1 Открой браузер на своём компьютере и перейди на сайт arduino.cc.
- 2 Перейди на страницу DOWNLOADS во вкладке SOFTWARE.



- 3 Выбери версию Arduino IDE для своей операционной системы.



- 4 Нажми JUST DOWNLOAD («просто скачать»). Начнётся скачивание файла.



5 Запусти скачанный файл и следуй инструкции установщика.

6 Открой Arduino IDE.



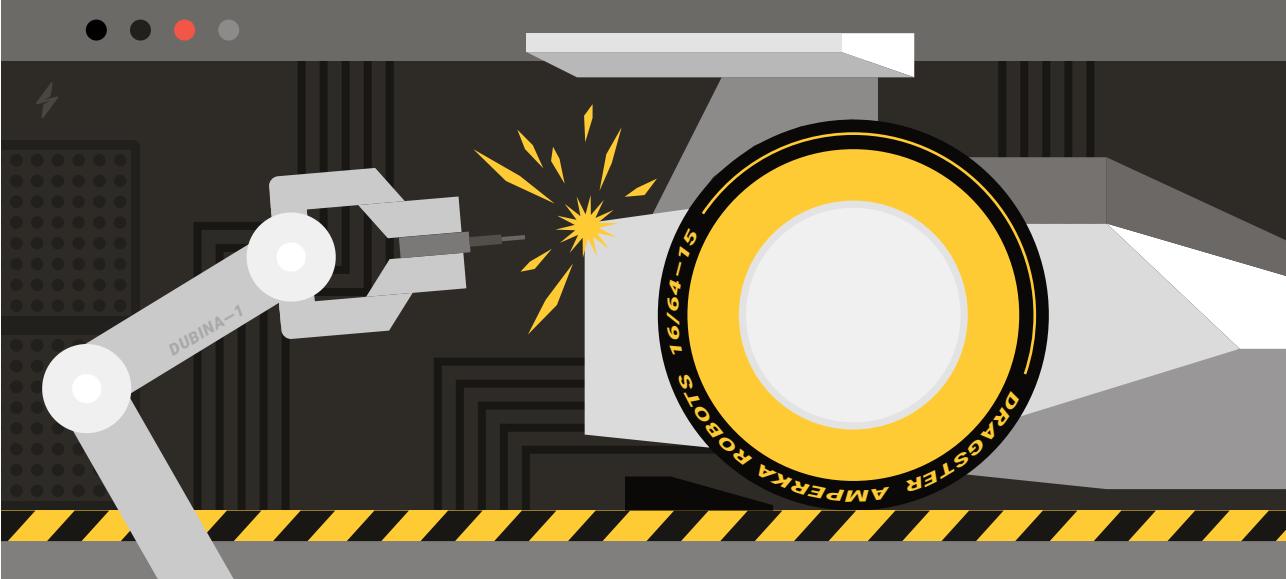
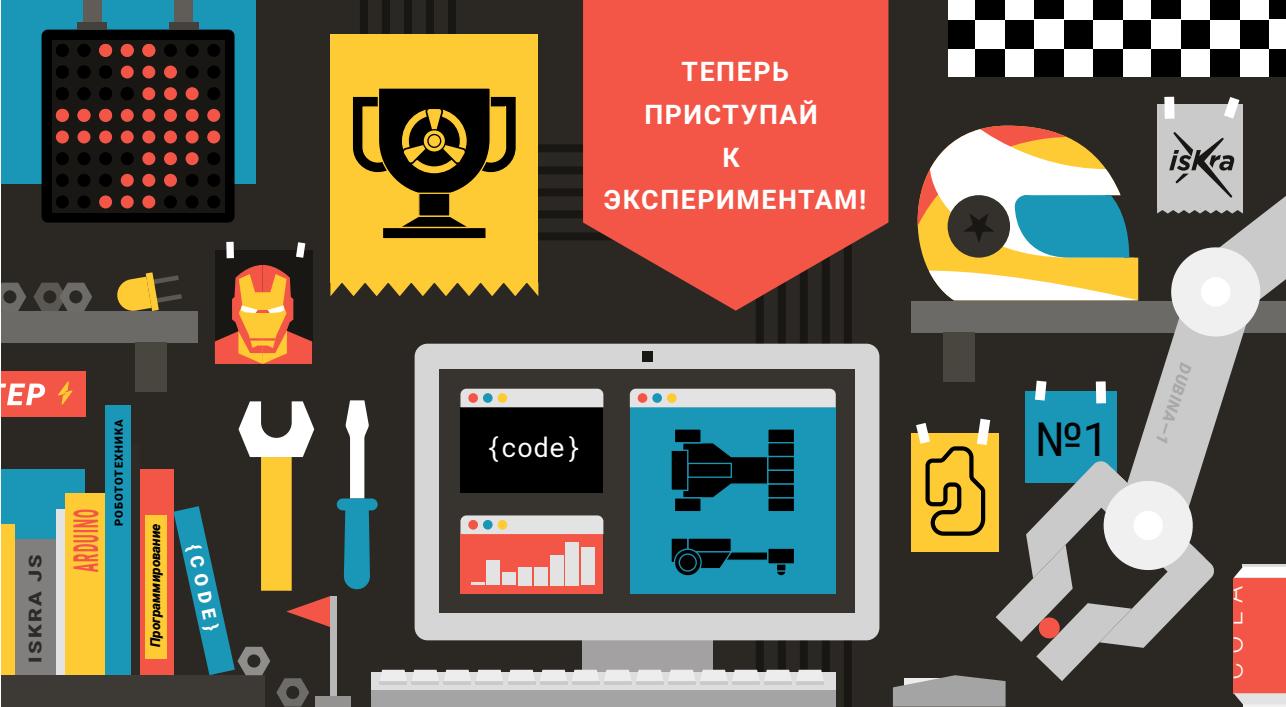
The screenshot shows the Arduino IDE interface. The title bar reads "sketch_sct20a | Arduino 1.8.7". The main code editor window contains the following C++ code:

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

void loop() {
  // put your main code here, to run repeatedly:
}
```

The code editor has a light gray background with syntax highlighting for keywords like "void", "setup", and "loop". The Arduino IDE interface includes standard window controls (minimize, maximize, close) at the top left, and a toolbar with various icons above the code editor. A status bar at the bottom displays the text "Arduino (C:\Users\Denis\Documents\Arduino\sketch_sct20a\sketch_sct20a.ino)".

ТЕПЕРЬ
ПРИСТУПАЙ
К
ЭКСПЕРИМЕНТАМ!

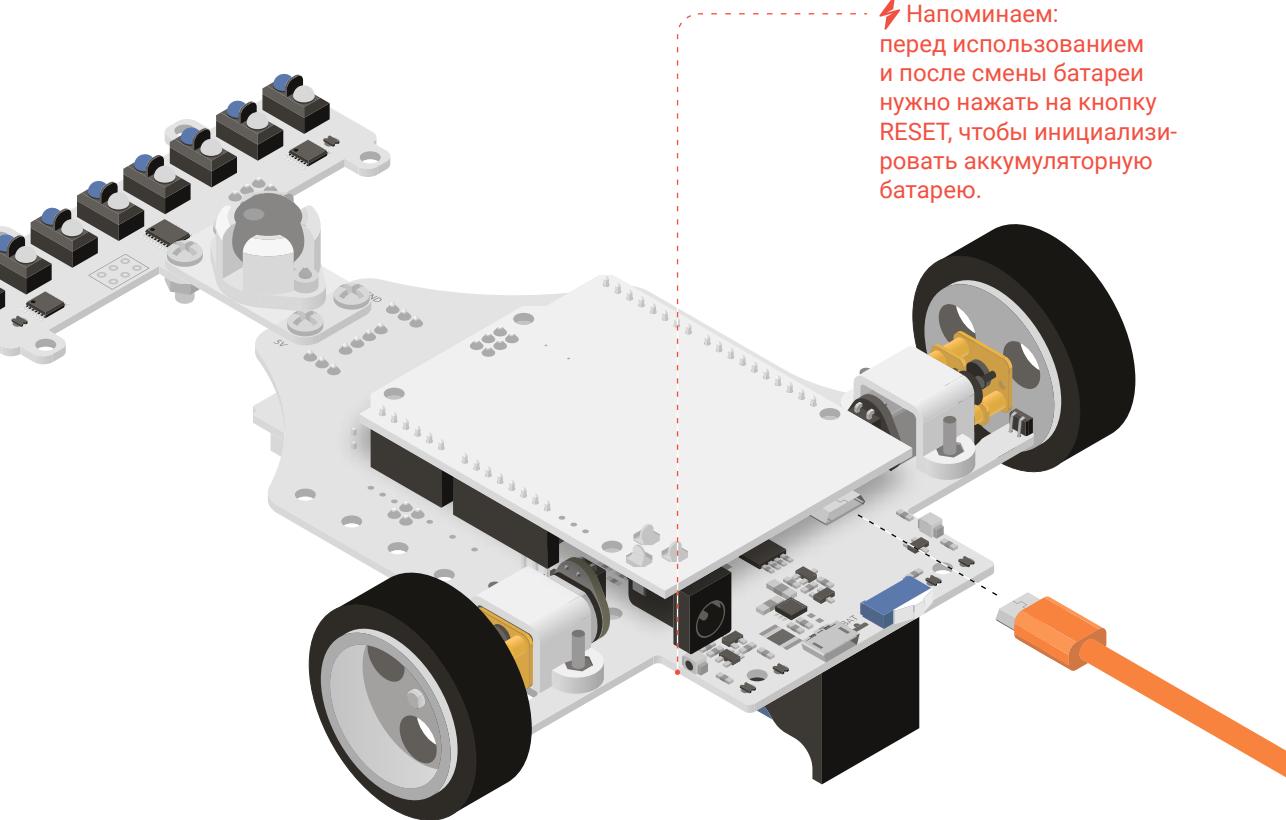


ЗАПУСК

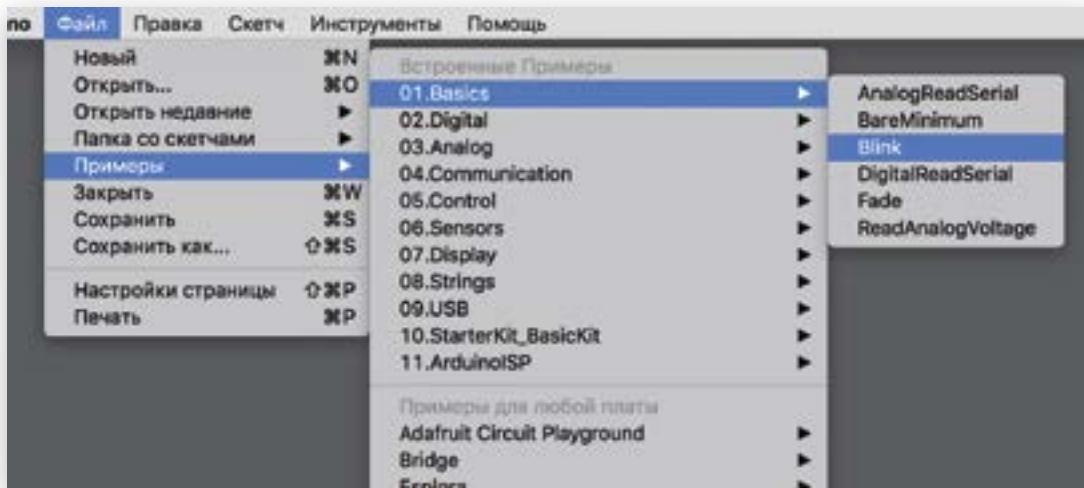
Начнём с простого: зажжём светодиод и убедимся, что плата работает, а код загружается.

- 1 Соедини плату Iskra Neo с компьютером кабелем micro-USB.

⚡ Напоминаем:
перед использованием
и после смены батареи
нужно нажать на кнопку
RESET, чтобы инициализи-
ровать аккумуляторную
батарею.



- 2 Открой Arduino IDE. В верхнем меню выбери
Файл → Примеры → 01.Basics → Blink.



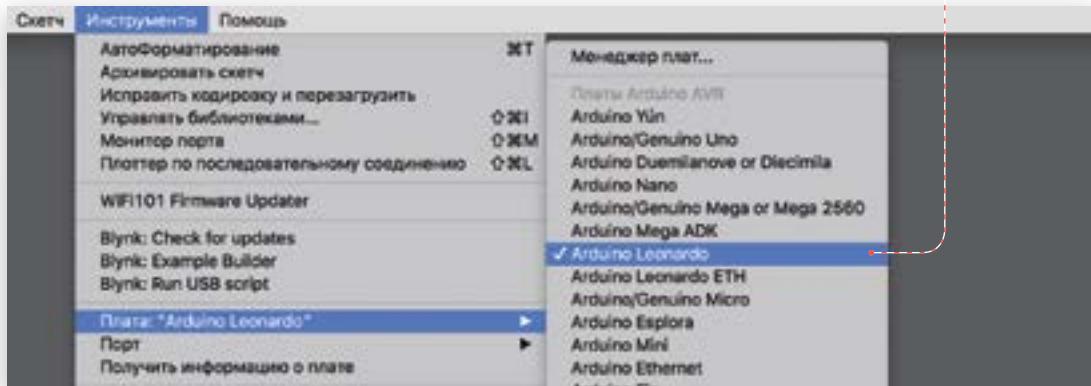
- 3 В окне редактора ты увидишь код программы мигания
светодиодом на 13-м пине.

```
1 void setup() {
2     pinMode(LED_BUILTIN, OUTPUT);
3 }
4
5 void loop() {
6     digitalWrite(LED_BUILTIN, HIGH);
7     delay(1000);
8     digitalWrite(LED_BUILTIN, LOW);
9     delay(1000);
10 }
```

4 Укажи среди разработки плату, с которой работаешь.

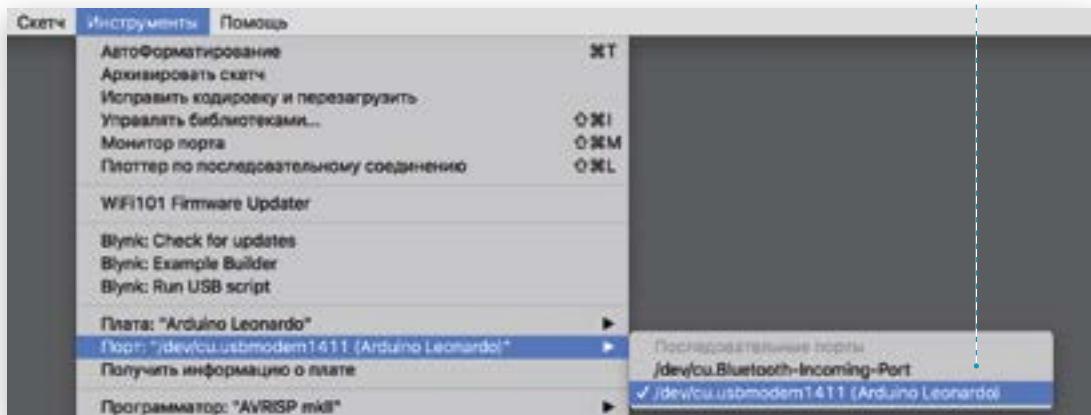
Для этого выбери в меню Инструменты → Плата → Arduino Leonardo.

Iskra Neo – аналог Arduino Leonardo, поэтому укажи именно её.



5 Убедись, что Порт выбран правильно.

Открой Инструменты → Порт. Выбери пункт, у которого в скобках указано Arduino Leonardo.

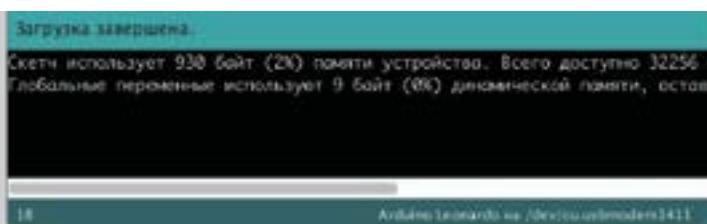


6 Жми кнопку Загрузка ➔.



7 Среда разработки начнёт компилировать код (переводить с человеческого языка на машинный).

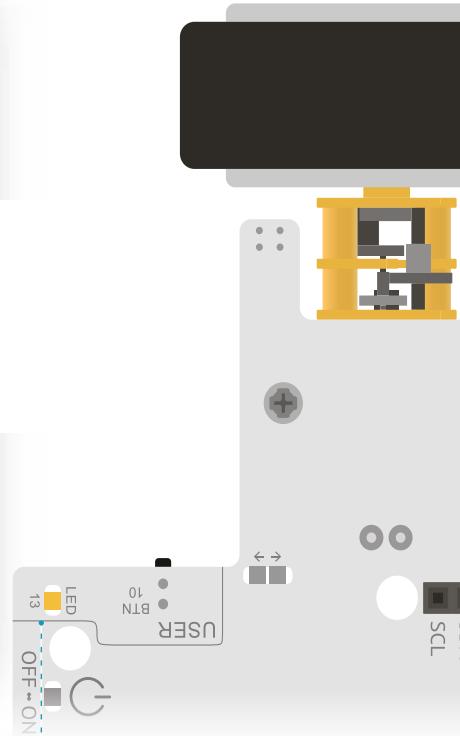
8 Если в коде нет ошибок, Arduino IDE попытается загрузить его в Iskra Neo.



Когда всё получится, код сразу начнёт исполняться на Iskra Neo. Светодиод будет мигать раз в две секунды.

Здорово, когда всё работает!

Но готовыми примерами пользоваться неспортивно. Разберёмся, как работает код, и научимся создавать свои собственные программы.



Если что-то пошло не так, открой сайт dragster.amperka.ru. Там ты найдёшь способы решения возникших проблем!

ИНДИКАЦИЯ

Разберёмся с кодом прошлого эксперимента и немного изменим его.

Программы в мире Arduino называются скетчами (англ. sketch – эскиз, набросок).

Взгляни на код. Он состоит из двух частей: **setup** (настройка) и **loop** (петля). Обе части называются функциями. Когда Iskra Neo запускается, она сперва один раз выполняет функцию **setup**, а затем раз за разом повторяет функцию **loop**, словно попала в петлю и не может из неё выбраться.

```
1 void setup() {
2   ...
3 }
4
5 void loop() {
6   ...
7 }
```

```
1 void setup() {
2   pinMode(LED_BUILTIN, OUTPUT);
3 }
```

4 Переменная **LED_BUILTIN** указывает, что нужно использовать pin, подключённый к встроенному светодиоду «L» на Iskra Neo (pin 13).

1 Функция **setup** выполняется один раз при старте программы.

2 **loop** продолжает выполняться раз за разом, пока на Iskra Neo подаётся питание.

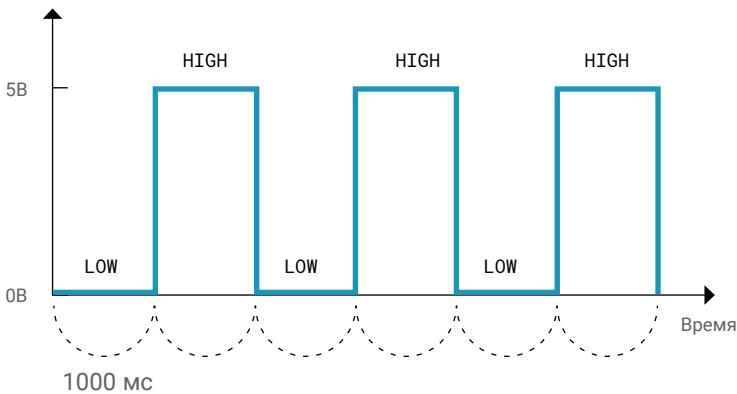
3 Функция **pinMode()** подготавливает pin к работе. В скобках через запятую указываются параметры функции. Первый параметр, **LED_BUILTIN**, задаёт номер пина. Второй параметр, **OUTPUT**, говорит, что pin необходимо настроить на выход.

```
4 void loop() {  
5     digitalWrite(LED_BUILTIN, HIGH);  
6     delay(1000);  
7     digitalWrite(LED_BUILTIN, LOW);  
8     delay(1000);  
9 }
```

5 Функция `digitalWrite()` задаёт напряжение на пине `LED_BUILTIN`. Высокое напряжение (5 вольт) устанавливается параметром `HIGH`, а низкое (0 вольт) – параметром `LOW`.

6 Функция `delay` (задержка) прерывает выполнение кода на количество миллисекунд, указанное в скобках.

Напряжение на пине `LED_BUILTIN`



ЗАДАНИЕ

Измени время задержки `delay`. Пусть светодиод горит 0,5 секунды (500 миллисекунд) и гаснет на 2 секунды. Загрузи обновлённый код в `Iskra Neo` и проверь результат.

КЛЮЧ НА СТАРТ

Научимся включать и выключать светодиод по нажатию кнопки.

В отличие от светодиода, кнопка сама устанавливает напряжение на пине Iskra Neo: либо 0, либо 5 вольт. Микроконтроллер считывает напряжение и использует результат в программе.



Кнопка отпущена

Iskra Neo прочитает на пине логическую 1.

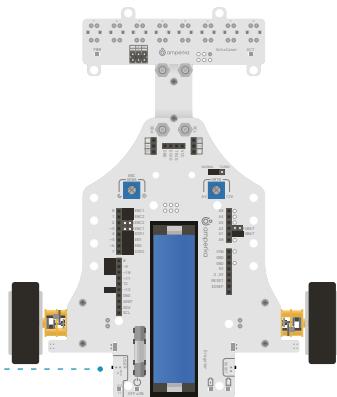


Кнопка нажата

Iskra Neo прочитает на пине логический 0.

Пока кнопка нажата, светодиод должен гореть, иначе — должен быть выключен. То есть светодиод будет менять состояние по условию.

Условие — это ветвление программы на два действия: «Если выполняется условие А, сделай действие Б, иначе — сделай действие В». В языке программирования такие ветвления оформляются конструкцией `if ... else ...`.



```
7 if ( ) {  
8   digitalWrite(LED_BUILTIN, HIGH);  
9 } else {  
10  digitalWrite(LED_BUILTIN, LOW);  
11 }
```

1 Условие **if** (если).

2 Действие (ветвь), если условие выполняется.

3 Действие, если условие не выполняется.

Кнопка на Драгстере подключена к pinu 10. Напряжение на пине считывается функцией **digitalRead()**. Она возвращает результат в виде числа: единицы — когда уровень высокий, и нуля — когда уровень низкий.

```
7 if (digitalRead(10) == LOW) {  
8   digitalWrite(LED_BUILTIN, HIGH);  
9 } else {  
10  digitalWrite(LED_BUILTIN, LOW);  
11 }
```

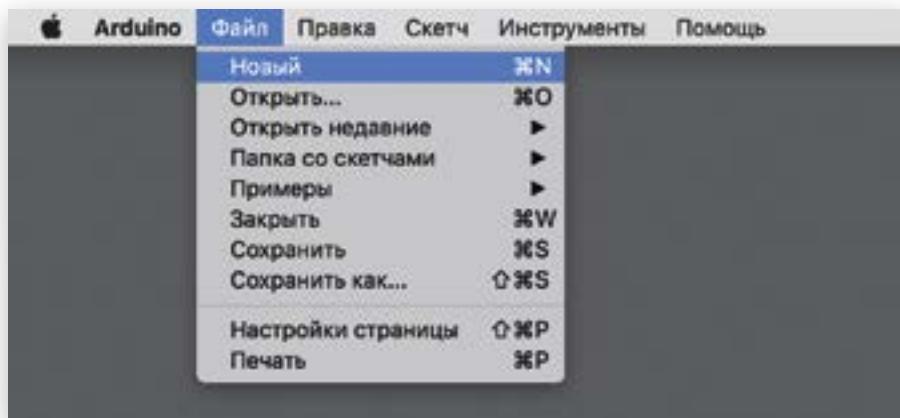
5 Если значение функции равно **LOW** (кнопка нажата), подаём на pin 13 высокое напряжение, иначе — низкое.

Не забываем подготовить пины в функции **setup**.

```
1 void setup() {  
2   pinMode(10, INPUT);  
3   pinMode(LED_BUILTIN, OUTPUT);  
4 }
```

6 Настраиваем 10-й pin Iskra Neo на вход. Для этого в функцию **pinMode** вторым параметром передаём **INPUT**.

- 1 Открой новый скетч в Arduino IDE (Файл → Новый).



- 2 Допиши функции `setup` и `loop`.

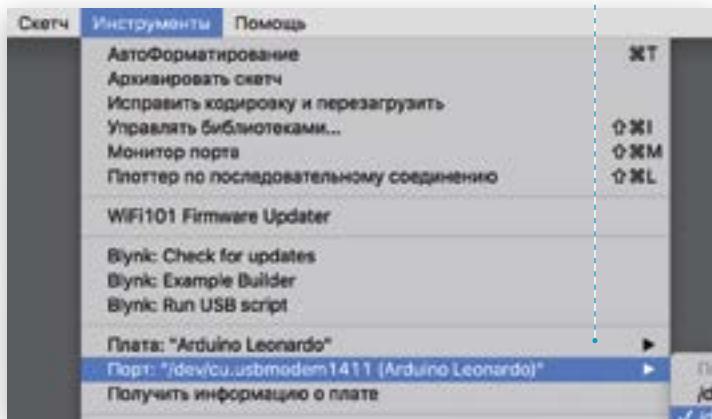
A screenshot of the Arduino IDE showing a sketch titled "sketch_nov06a". The code in the editor is as follows:

```
sketch_nov06a | Arduino 1.8.7

1 void setup() {
2   pinMode(10, INPUT);
3   pinMode(LED_BUILTIN, OUTPUT);
4 }
5
6 void loop() {
7   if (digitalRead(10) == LOW) {
8     digitalWrite(LED_BUILTIN, HIGH);
9   } else {
10    digitalWrite(LED_BUILTIN, LOW);
11  }
12 }
```

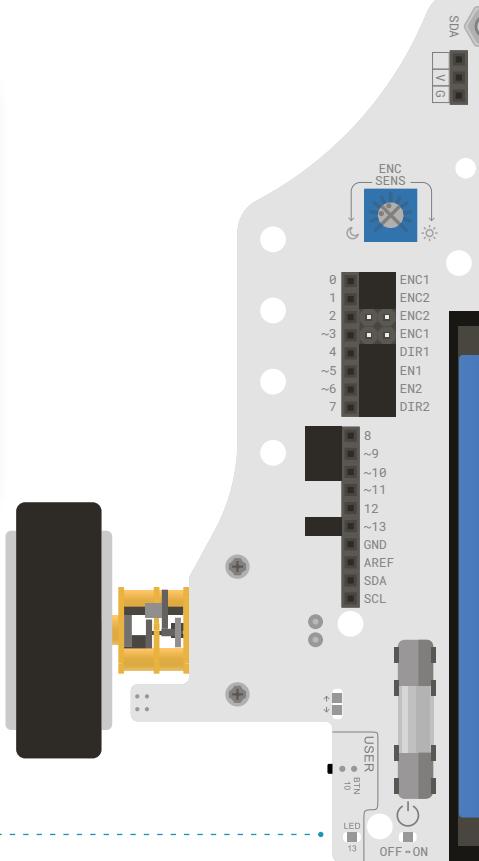
The code defines a setup function that initializes pins 10 and the built-in LED as inputs and outputs respectively. It also defines a loop function that reads the state of pin 10 and toggles the built-in LED based on its value.

- 3 Подключи Iskra Neo к компьютеру. Открой меню Инструменты и проверь, что Плата и Порт выбраны правильно.



- 4 Загрузи код в Iskra Neo ➔.

- 5 Понажимай на кнопку. Светодиод должен реагировать на нажатия.

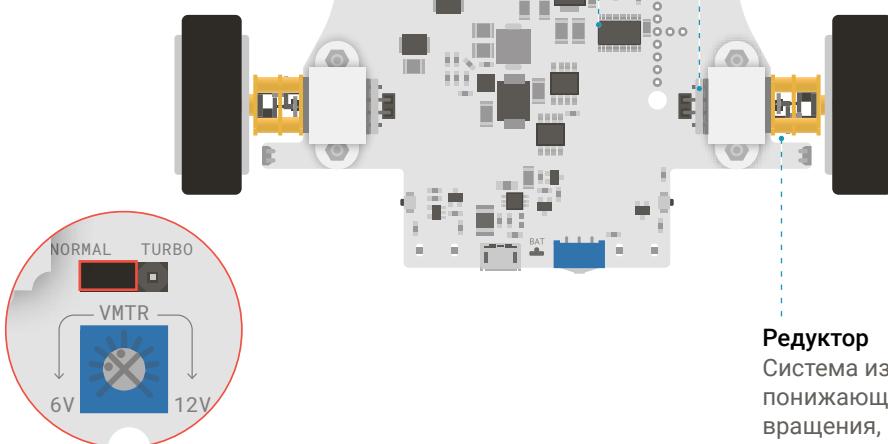


ПРОГРЕВ ДВИГАТЕЛЕЙ

Запустим робота. Для начала медленно, чтобы случайно не сломать его.

Драйвер двигателей

Коллекторные моторы потребляют большой ток, способный сжечь микроконтроллер. Чтобы этого не произошло, используют усилители тока. Драйвер — усилитель тока, способный переключать направление вращения двигателя.



 Убедись, что перемычка турбо-режима стоит в состоянии Normal.

Коллекторный двигатель

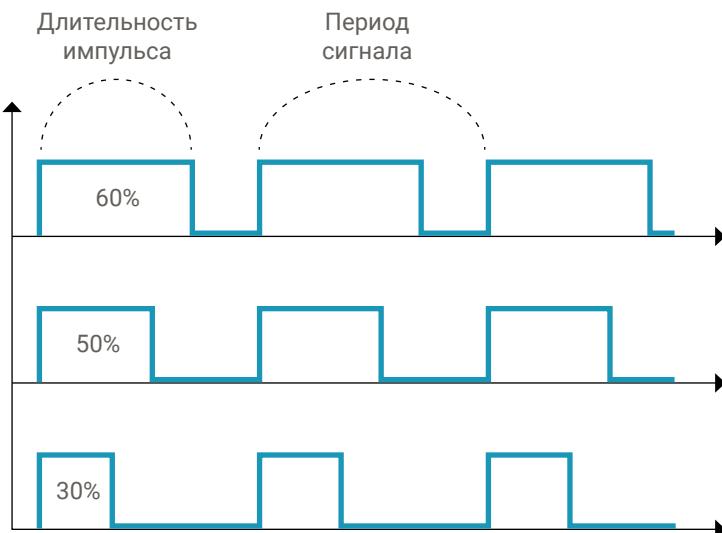
Ещё называют щёточный. Ток передаётся на вал через щётки. Они быстро выходят из строя, если подавать на них высокое напряжение. Поэтому не рекомендуется устанавливать напряжение выше 7 вольт на регуляторе.

Редуктор

Система из шестерёнок, понижающая скорость вращения, но повышающая момент (усилие на валу двигателя). Без редуктора мотору не хватит сил сдвинуть робота с места.

Драйвер на роботе управляет двумя двигателями.
Направление вращения задаётся пинами DIR1 и DIR2.
Для управления скоростью используется ШИМ на пинах EN1 и EN2.

ШИМ – широтно-импульсная модуляция – быстрое переключение напряжения таким образом, чтобы его среднее значение было пропорционально длительности импульсов.



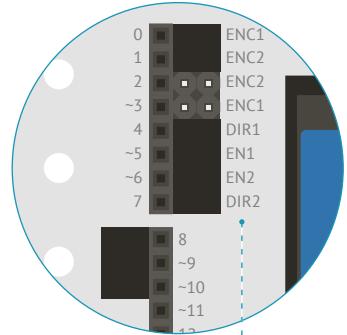
 Мощность, передаваемая на двигатели, пропорциональна длительности импульса.

Доля длительности импульса от периода всего сигнала называется «коэффициент заполнения» и может быть либо от 0 до 1, либо 0% до 100%.

В среде Arduino ШИМ задаётся функцией `analogWrite(номер_пина, значение)` в диапазоне от 0 до 255.

1 Открой новый скетч в Arduino IDE и напиши код.

```
1 void setup() {  
2     pinMode(4, OUTPUT);  
3     pinMode(5, OUTPUT);  
4     pinMode(6, OUTPUT);  
5     pinMode(7, OUTPUT);  
6  
7     digitalWrite(4, HIGH);  
8     digitalWrite(7, HIGH);  
9 }  
10  
11 void loop() {  
12     analogWrite(5, 64);  
13     analogWrite(6, 64);  
14     delay(1000);  
15  
16     analogWrite(5, 0);  
17     analogWrite(6, 0);  
18     delay(1000);  
19 }
```



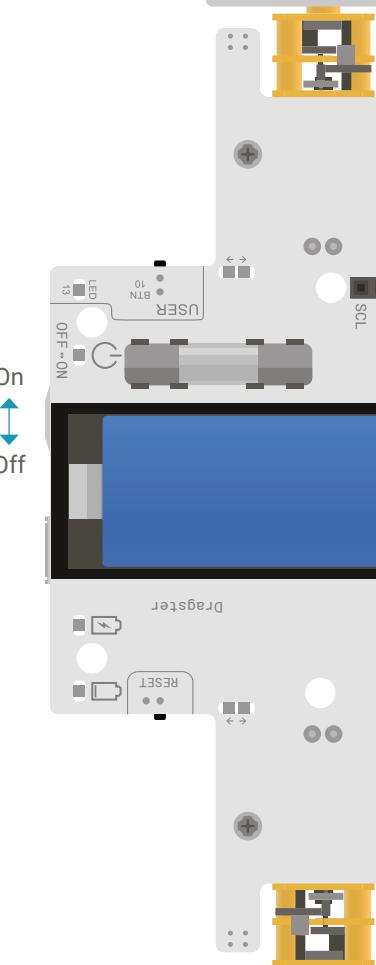
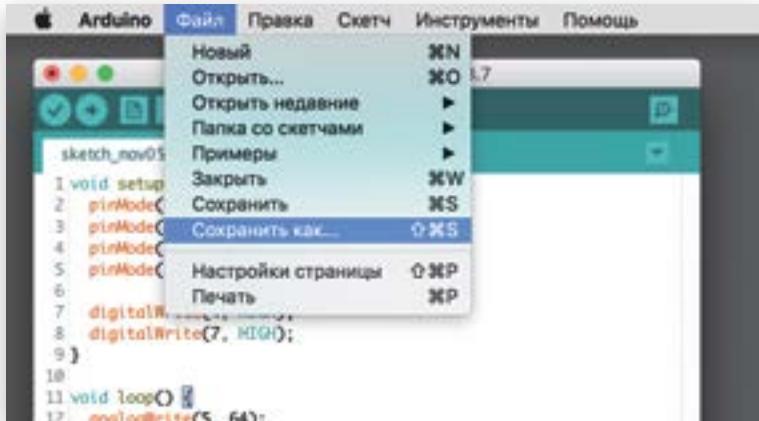
1 Настраиваем пины **DIR** и **EN** на выход. Номера пинов указаны на плате Драгстера.

2 Задаём направление вращения двигателей. **HIGH** – в одну сторону, **LOW** – в другую.

4 Задаём коэффициент заполнения импульсов примерно 25%, Драгстер начнёт медленно двигаться.

5 Отключаем ШИМ, заставляя робота остановиться.

- 2** Сохрани код под именем *Preparation.ino* (Файл → Сохранить как...).



ЗАДАНИЕ

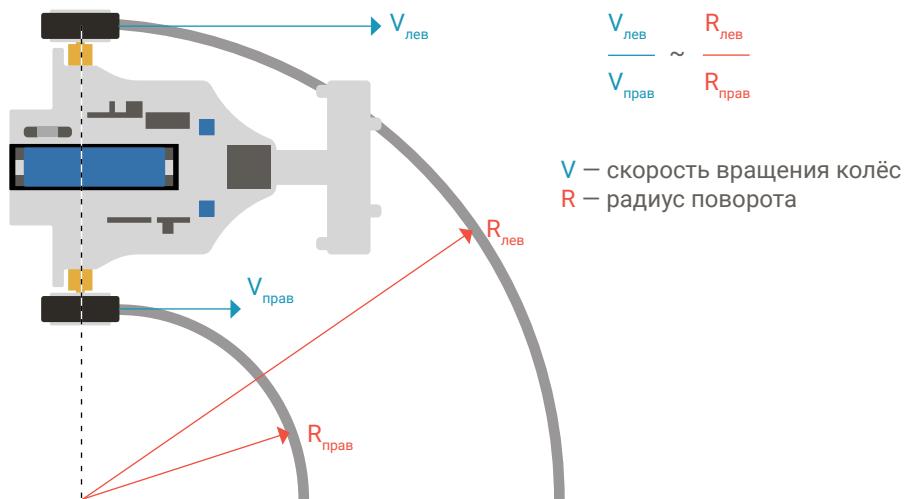
Подбери значения (HIGH, LOW) на пинах 4 и 7 таким образом, чтобы оба колеса двигались вперед.

ЗМЕЙКА

Научим робота поворачивать и двигаться по кругу.

Чтобы заставить робота сделать поворот, достаточно задать колёсам разную скорость.

Рассчитать траекторию движения можно по упрощённой формуле:



- 1 Потренируйся проходить повороты, запиши код и загрузи его в Iskra Neo.
- 2 Опусти робота на пол, отсоединив от кабеля. Робот должен начать движение по окружности.
- 3 Повтори эксперимент с поворотом в другую сторону.

```

1 #define DIR1 4      '
2 #define DIR2 7      '
3 #define EN1  5      '
4 #define EN2  6      '
5
6 void setup() {
7     pinMode(DIR1, OUTPUT);      '
8     pinMode(EN1,  OUTPUT);      '
9     pinMode(EN2,  OUTPUT);      '
10    pinMode(DIR2, OUTPUT);      '
11
12    digitalWrite(DIR1, HIGH);
13    digitalWrite(DIR2, HIGH);
14
15    analogWrite(EN1, 45);      '
16    analogWrite(EN2, 90);      '
17 }
18
19 void loop() {      '
20
21 }
```

1 Используем директивы `#define` для хранения пинов под заданным именем. Это позволяет не запоминать пины, а использовать их понятное обозначение во всей программе.

2 Устанавливаем пины на выход, задавая их имена вместо чисел.

3 Задаём разную скорость двигателей. Робот будет двигаться по окружности.

4 Оставляем цикл `loop()` пустым. Он будет раз за разом повторять *ничего*, пока не отключится питание.

ЗАДАНИЕ

Попробуй научить робота ездить змейкой: влево, вправо, влево, вправо и так далее. Используй алгоритм ниже. Расставь на полу стаканчики вдоль прямой линии и устрой слалом.

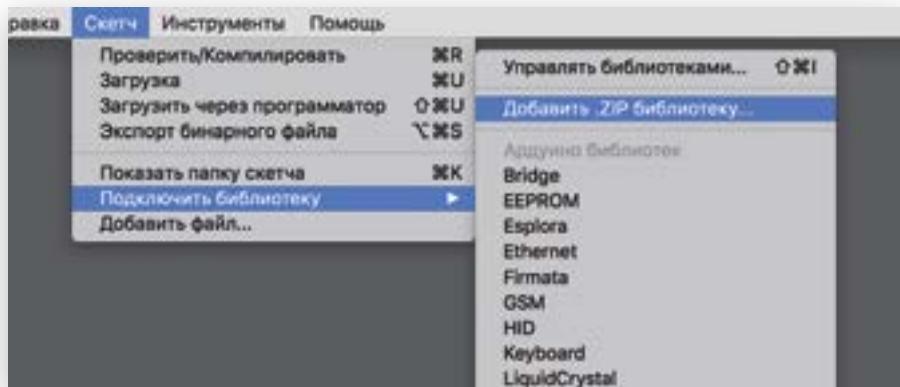
- ➊ Настроить пины `DIR1`, `DIR2`, `EN1`, `EN2` на выход.
- ➋ Задать направление вращения двигателей на пинах `DIR1`, `DIR2`.
- ➌ Задать двигателям скорость для движения вперёд и влево.
- ➍ Подождать N секунд.
- ➎ Задать двигателям скорость для движения вперёд и вправо.
- ➏ Подождать N секунд.

ТЮНИНГ. БИБЛИОТЕКИ

Проведём оптимизацию кода: сделаем его короче и проще. В этом помогут библиотеки.

Чтобы не писать код каждый раз заново, его выносят в отдельные файлы — библиотеки. Огромное количество готового кода уже написано другими людьми, и с помощью библиотек его можно легко использовать в своих программах. Установим библиотеку *Dragster* для управления двигателями робота.

- 1 Скачай zip-архив *Dragster.zip* на сайте dragster.amperka.ru в разделе *Библиотека Dragster*.
- 2 Открой Arduino IDE и добавь библиотеку через меню Скетч → Подключить библиотеку → Добавить .ZIP библиотеку...



3 В окне выбора файла открой скачанный zip-архив и нажми *Open*. Библиотека готова к использованию!

4 Напиши простой скетч движения змейкой.

```
1 #include "Dragster.h"
2
3 Dragster robot(MMAX_16_OHM);
4
5 void setup() {
6     robot.begin();
7 }
8
9 void loop() {
10    robot.drive(110, 50);
11    delay(1000);
12    robot.drive(50, 110);
13    delay(1000);
14 }
```

```
6 robot.begin(SWAP_RIGHT);
6 robot.begin(SWAP_LEFT);
6 robot.begin(SWAP_BOTH);
```

1 Подключаем библиотеку **Dragster**. Для этого используется директива **#include** и указывается заголовочный файл библиотеки **Dragster.h**.

2 Заводим объект **robot**, через него будем общаться с функциями библиотеки. **MMAX_16_OHM** – константа, которая указывает, какими именно моторами оснащён твой Драгстер. Не забывай добавлять её, иначе Драгстер будет ехать медленно или даже испортит моторы.

3 Функция **begin** объекта **robot** выполняет настройку пинов **Iskra Neo** для работы с Драгстером.

4 Используем встроенную функцию **drive** объекта **robot**. Первым параметром задаём скорость левого колеса, а вторым – правого в диапазоне от -255 до 255.

5 Если на твоём Драгстере колёса врачаются не в ту сторону, передай в функцию **robot.begin()** дополнительный параметр **SWAP_RIGHT**, **SWAP_LEFT** или **SWAP_BOTH**.

ЗАДАНИЕ

Передай в функцию **drive()** отрицательные значения, убедись, что колёса начнут вращаться в другую сторону.

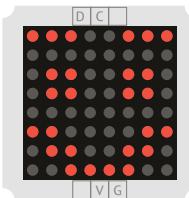
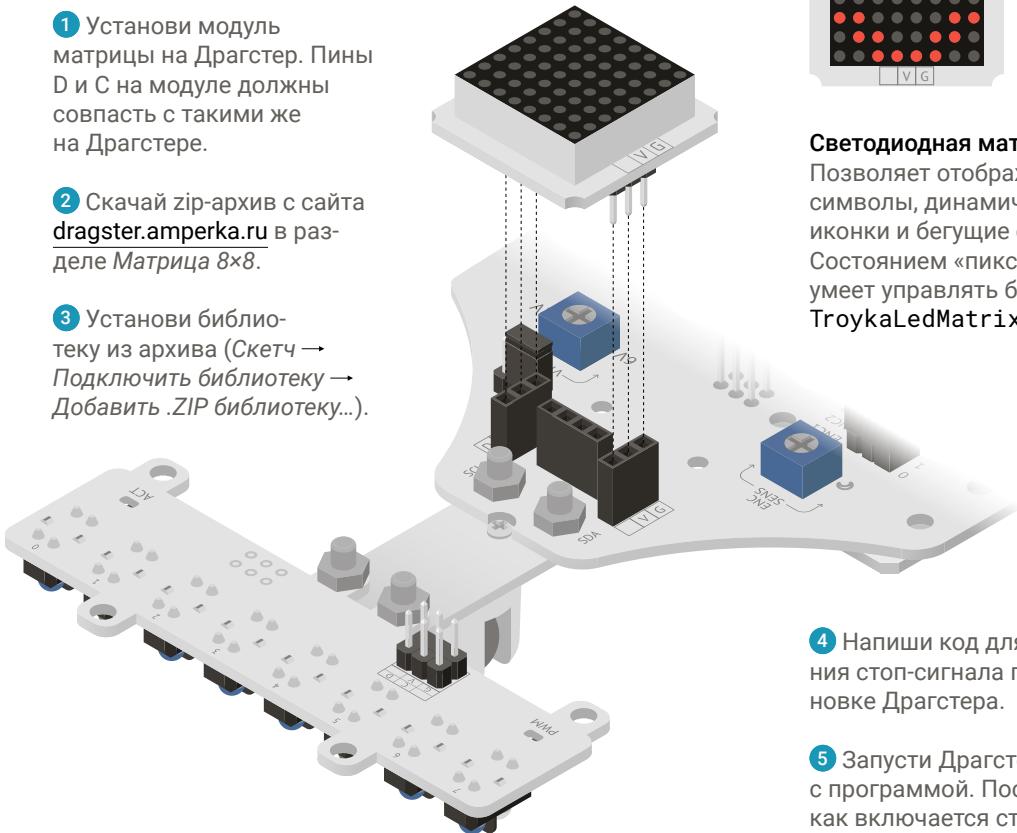
СТОП-СИГНАЛ

Добавим Драгстеру динамическую индикацию на светодиодной матрице. Будет полезно отображать информацию при отладке.

1 Установи модуль матрицы на Драгстер. Пины D и C на модуле должны совпасть с такими же на Драгстере.

2 Скачай zip-архив с сайта dragster.amperka.ru в разделе Матрица 8x8.

3 Установи библиотеку из архива (Скетч → Подключить библиотеку → Добавить .ZIP библиотеку...).



Светодиодная матрица 8x8

Позволяет отображать символы, динамические иконки и бегущие строки. Состоянием «пикселей» умеет управлять библиотека `TroykaLedMatrix`.

4 Напиши код для включения стоп-сигнала при остановке Драгстера.

5 Запусти Драгстер с программой. Посмотри, как включается стоп-сигнал при остановке.

```

1 #include "Dragster.h"
2 #include "TroykaLedMatrix.h" •
3
4 Dragster robot(MMAX_16_OHM);
5 TroykaLedMatrix matrix; •
6
7 uint8_t stop[] = {      •
8     0b11111111,
9     0b11111111,
10    0b11111111,
11    0b11111111,
12    0b11111111,
13    0b11111111,
14    0b11111111,
15    0b11111111,
16 };
17
18 void setup() {
19     robot.begin();
20     matrix.begin(); •
21 }
22
23 void loop() {
24     matrix.clear(); •
25     robot.drive(120, 120);
26     delay(1000);
27
28     matrix.drawBitmap(stop); •
29     robot.drive(0, 0);
30     delay(1000);
31 }
```

1 Подключаем библиотеку `TroykaLedMatrix`.

2 Заводим объект `matrix`. Он будет отвечать за передачу пикселей на матрицу.

3 Создаём массив пикселей матрицы: 8 строк по 8 битов. Для наглядности записываем строки в бинарном виде. Заполняем биты единицами, означающими «включить пиксель».

4 Инициализируем матрицу, то есть запускаем функцию настройки, встроенную в библиотеку.

5 Очищаем экран матрицы.

6 Рисуем на матрице массив из пикселей. Получится сплошной квадрат.

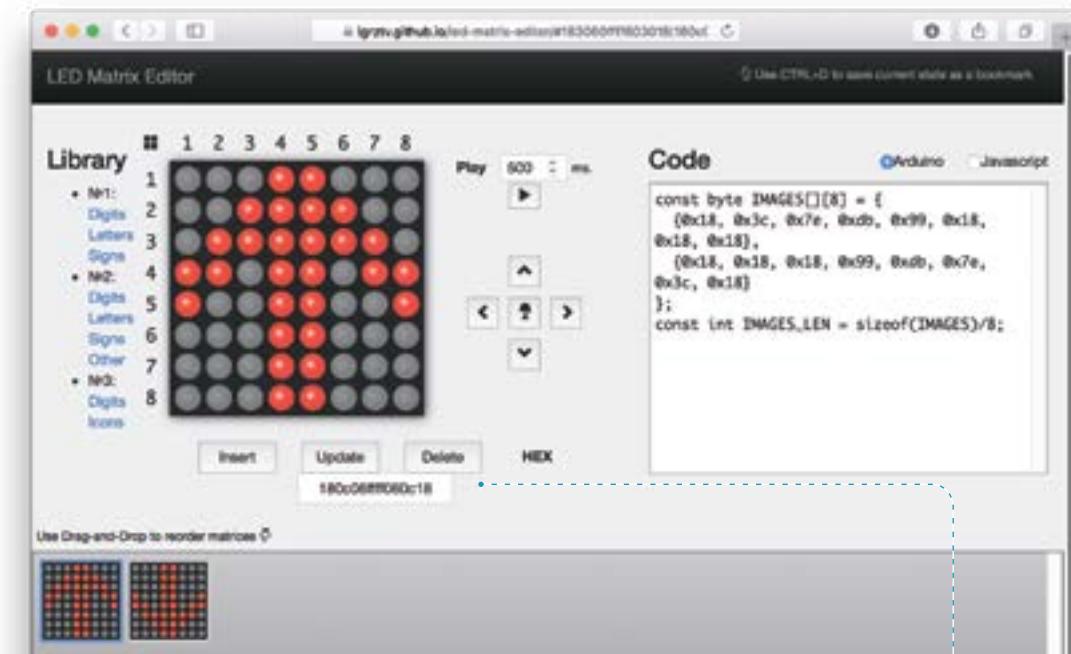
ЗАДАНИЕ

Поэкспериментируй с символами на матрице. В массиве `stop` составь из единиц и нулей стрелку, крестик или плюс.

ПОВОРОТНИКИ

Выведем графику посложнее. Создадим несколько изображений и будем переключать по необходимости.

- 1 Зайди на сайт dragster.amperka.ru и перейди по ссылке на редактор изображений. Нарисуй свои иконки двух стрелок: влево и вправо.



- 2 Скопируй получившийся код в свой скетч. Это массив изображений **IMAGES** (массив массивов строк).

Чтобы добавить изображение в массив, пользуйся кнопками Insert (добавить), Update (обновить) и Delete (удалить).

- 3 Добавь Драгстеру индикацию стрелками, указывающими направление поворота.

```
1 #include "Dragster.h"
2 #include "TroykaLedMatrix.h"
3
4 Dragster robot(MMAX_16_0HM);
5 TroykaLedMatrix matrix;
6
7 const byte IMAGES[][8] = {
8     { 0x18, 0x3c, 0x7e, 0xdb, 0x99, 0x18, 0x18, 0x18 },
9     { 0x18, 0x18, 0x18, 0x99, 0xdb, 0x7e, 0x3c, 0x18 } }
10 };
11 const int IMAGES_LEN = sizeof(IMAGES) / 8;   ···
12
13 void setup() {
14     robot.begin();
15     matrix.begin();
16 }
17
18 void loop() {
19     matrix.drawBitmap(IMAGES[0]);
20     robot.drive(110, 60);
21     delay(1000);
22
23     matrix.drawBitmap(IMAGES[1]);
24     robot.drive(60, 110);
25     delay(1000);
26 }
```

1 Массив IMAGE из двух изображений. Каждое из них содержит массив строк матрицы.

2 Константа IMAGE_LEN хранит длину массива IMAGES – количество изображений. Сейчас это не понадобится.

3 Выводим изображение стрелки вправо и поворачиваем направо.

4 Меняем изображение на стрелку влево и поворачиваем налево.

ЗАДАНИЕ

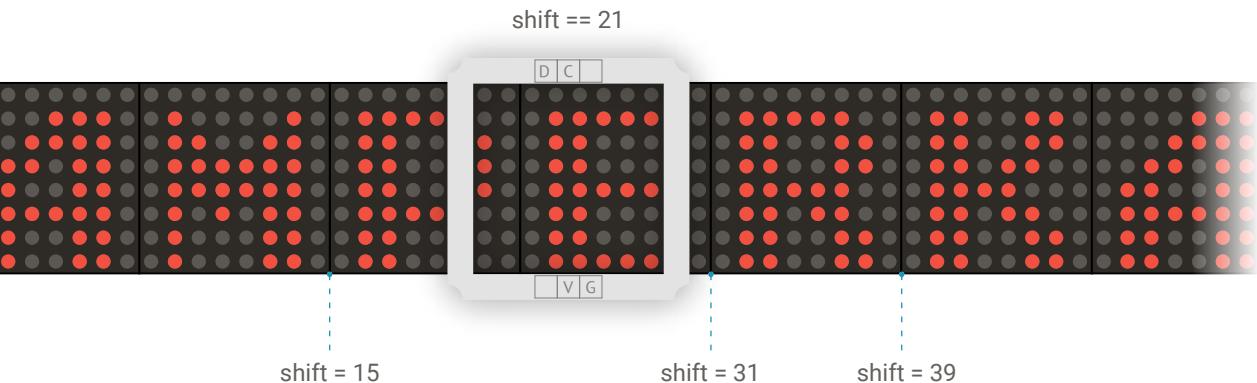
Вернись в редактор иконок и поверни изображения. Посмотри библиотеки символов в левой части редактора.

РЕКЛАМА СПОНСОРОВ

Выведем на светодиодную матрицу длинную строку.

Длинные строки не помещаются целиком на матрицу, поэтому будем их отображать в виде бегущей строки. Для этого создадим массив изображений, где в каждом изображении будет свой символ строки.

В библиотеке *TroykaLedMatrix* есть готовая функция отображения бегущей строки – `marquee`. Она принимает в качестве параметров массив изображений, длину массива и величину сдвига строки. Увеличивая или уменьшая сдвиг, выбираем квадрат, который необходимо отобразить в текущий момент.



- 1 Создай скетч для отображения бегущей строки.

```
1 #include "TroykaLedMatrix.h"
2
3 TroykaLedMatrix matrix;
4
5 const byte IMAGES[][8] = {
6     {0x30, 0x78, 0xcc, 0xcc, 0xfc, 0xcc, 0xcc, 0x00},
7     {0xc6, 0xee, 0xfe, 0xfe, 0xd6, 0xc6, 0xc6, 0x00},
8     {0xfc, 0x66, 0x66, 0x7c, 0x60, 0x60, 0xf0, 0x00},
9     {0xfe, 0x62, 0x68, 0x78, 0x68, 0x62, 0xfe, 0x00},
10    {0xfc, 0x66, 0x66, 0x7c, 0x6c, 0x66, 0xe6, 0x00},
11    {0xe6, 0x66, 0x6c, 0x78, 0x6c, 0x66, 0xe6, 0x00},
12    {0x30, 0x78, 0xcc, 0xcc, 0xfc, 0xcc, 0xcc, 0x00},
13    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
14};
15 const int IMAGES_LEN = sizeof(IMAGES)/8;
16
17 byte shift = 0;  -----
18
19 void setup() {
20     matrix.begin();
21 }
22
23 void loop() {
24     delay(70);
25     matrix.marquee(IMAGES, IMAGES_LEN, shift++);
26     if (shift == IMAGES_LEN * 8) {   -----
27         shift = 0;
28     }
29 }
```

- 1 Создаём 8 символов для бегущей строки.

2 Переменная `shift` будет отвечать за сдвиг символов в строке.

3 Функция `marquee` отображает часть бегущей строки `IMAGES` длиной `IMAGES_LEN` со сдвигом `shift`. Сразу после вызова функции оператор `++` увеличит переменную `shift` на 1.

4 Если величина `shift` достигла длины строки, обнуляем переменную.

ЗАДАНИЕ

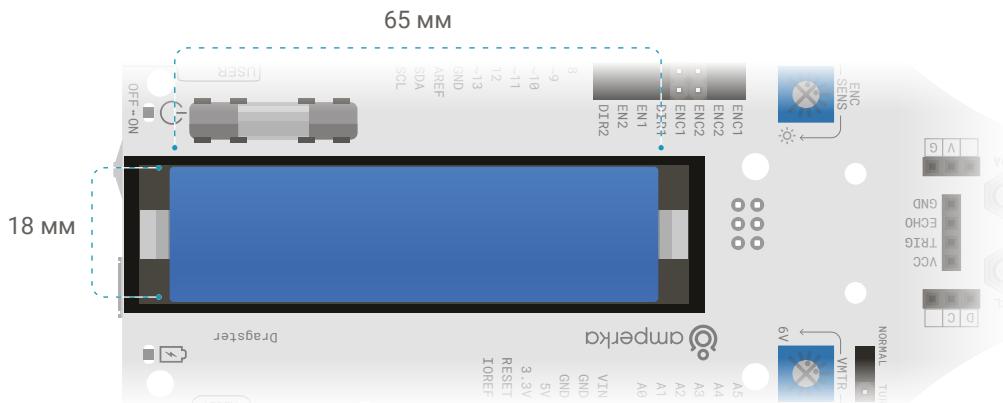
Запусти строку в другую сторону. Для этого измени условие проверки переменной `shift` и замени оператор `++` на `--`.

№10 УРОВЕНЬ ЗАРЯДА

Измерим напряжение аккумулятора и выведем значение в Serial и на матрицу.

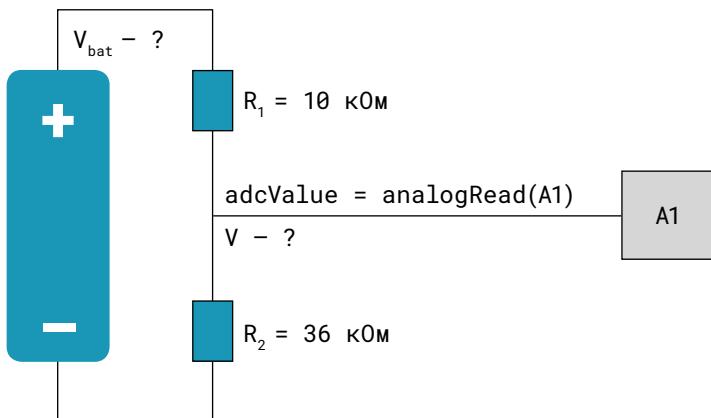
Драгстер работает от литий-ионного (Li-ion) аккумулятора. Аккумулятор выдаёт напряжение 4,2 вольта при полной зарядке и 3,2 вольта при полном разряде. Литиевые аккумуляторы быстро выходят из строя, если напряжение опускается ниже 3,2 вольта. Поэтому очень важно следить за уровнем заряда и не допускать падения напряжения.

Литий-ионный аккумулятор в Драгстере имеет популярный типоразмер 18650. Это означает, что его диаметр равняется 18 миллиметрам, а длина – 650 десятых миллиметра, то есть 65 миллиметров.



Для контроля заряда аккумулятора на Драгстере предусмотрен резисторный делитель напряжения, или просто делитель. Это элемент электрической цепи из двух последовательных резисторов, который пропорционально уменьшает входное напряжение. Это сделано для плат, которые не могут измерять напряжение выше 3,3 вольта.

 Подробнее про делитель можно узнать в разделе Электричество на сайте wiki.amperka.ru.



Напряжение аккумулятора вычисляется по формуле:

$$V_{\text{bat}} = \frac{V \cdot (R_1 + R_2)}{R_2}$$

Напряжение V между резисторами получим с аналогового входа A1 функцией `analogRead(A1)`. Она возвращает результат в диапазоне от 0 до 1023. Преобразуем это значение в напряжение.

$$V = \text{adcValue} \cdot \frac{5,0}{1023,0}$$

Умножаем измеренное значение с пина A1 на отношение максимального напряжения в вольтах к максимальному аналоговому значению.

1 Напиши скетч измерения напряжения аккумулятора.

```
1 #include "TroykaLedMatrix.h"
2
3 TroykaLedMatrix matrix;
4
5 byte volume[8] =
6     { 0x80, 0xc0, 0xe0, 0xf0, 0xf8, 0xfc, 0xfe, 0xff };
7 byte diagram[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
8 float R1 = 10000;
9 float R2 = 36000;
10
11 void setup() {
12     matrix.begin();
13     pinMode(A1, INPUT);
14 }
15
16 void loop() {
17     delay(200);
18     int adcValue = analogRead(A1);
19     float V = adcValue * 5.0 / 1023.0;
20     float Vbat = V * (R1 + R2) / R2;
21
22     int matrixValue = (Vbat - 3.2) * 8;
23     for (int i = 0; i < 8; i++) {
24         if (i < matrixValue) {
25             diagram[i] = volume[i];
26         } else {
27             diagram[i] = 0x00;
28         }
29     }
30     matrix.drawBitmap(diagram);
31 }
```

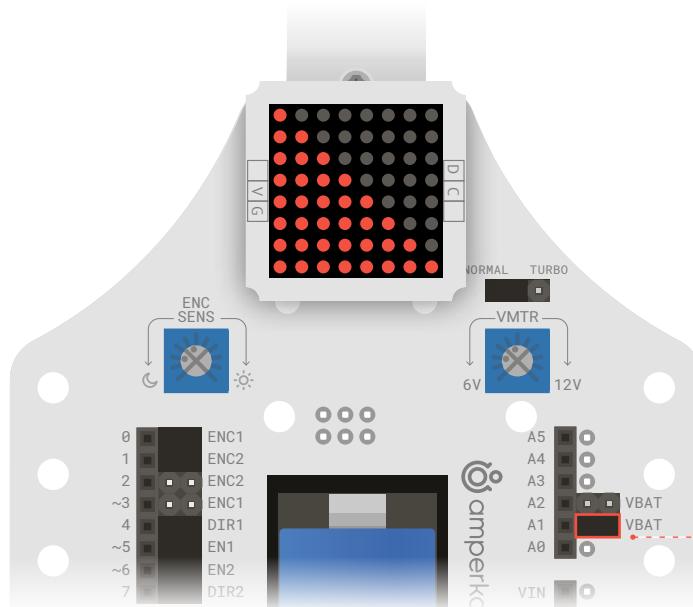
1 Заводим массив для хранения элементов «лесенки», обозначающей уровень заряда аккумулятора.

2 Определяем сопротивления R1 и R2.

3 Считываем аналоговое значение с пина A1, преобразуем по формулам и выводим в Монитор порта.

4 Преобразуем напряжение в столбики на матрице, выводим изображение.

2 Включи робота и посмотри на матрицу. В зависимости от уровня заряда аккумулятора, матрица будет заполняться столбиками разной высоты. Если горят все столбики — аккумулятор полностью заряжен. Если не горит ни один — ставь Драгстер на зарядку.



⚡ Убедись, что перемычка установлена именно так:
на A1!

Если у тебя возникнут подозрения, что аккумулятор разряжен, запусти скетч и проверь напряжение.

ЗАДАНИЕ

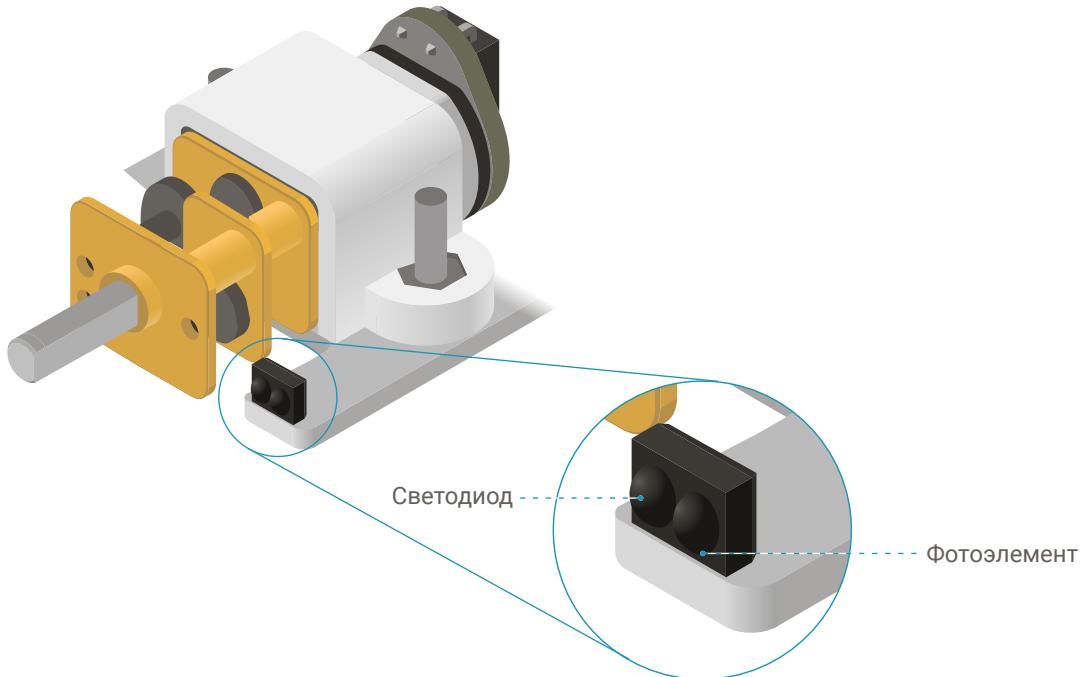
Измени содержимое массива

```
volume[8] =  
{ 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
```

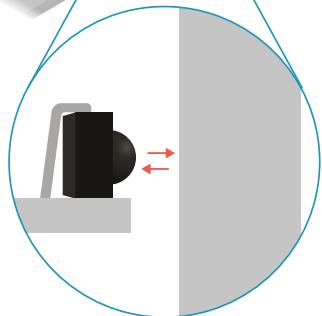
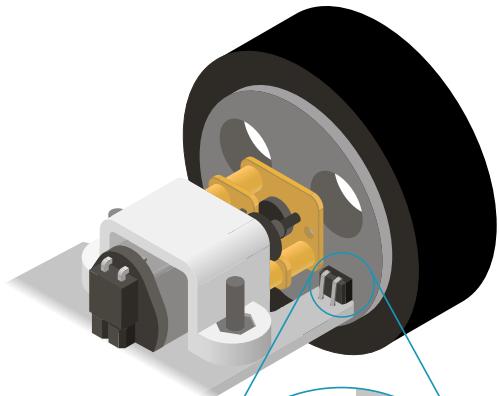
№11 ЭНКОДЕР

Научимся определять пройденный путь колеса.
Это необходимо для навигации, точных поворотов
и контроля скорости.

Для этой цели на Драгстере предусмотрены оптические энкодеры — датчики, работающие на отражении инфракрасного света. Светодиод излучает свет в инфракрасном диапазоне, а чувствительный фотоэлемент пытается его уловить в отражении от предметов.

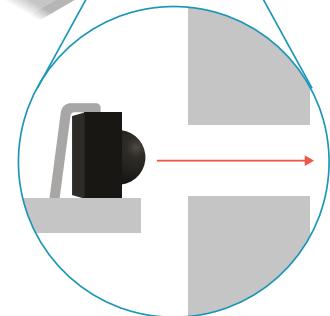
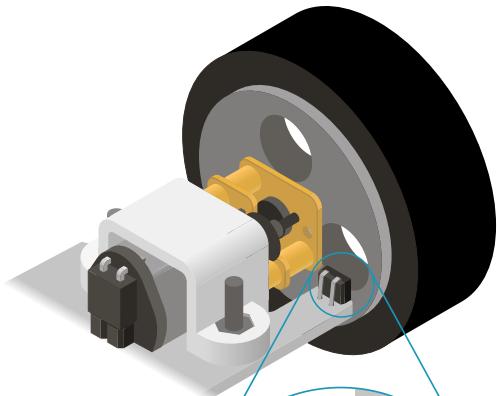


Если на фотоэлемент возвращается достаточное количество света, на пине ENC будет логическая единица, иначе – логический ноль. С точки зрения Iskra Neo энкодер ничем не отличается от кнопки.



Свет отражается от диска и возвращается на фотоэлемент.

`digitalRead (0) == 1`



Свет проходит сквозь отверстие и не возвращается к энкодеру.

`digitalRead (0) == 0`

💡 Также свет хорошо отражается от белых поверхностей и плохо – от чёрных.

Посмотрим, как выглядит сигнал энкодера.

Будем выводить значение на печать через последовательный порт – Serial. Он умеет обмениваться данными с компьютером через кабель USB.

- 1 Напиши скетч и загрузи его в плату.

```
1 #include "Dragster.h"
2
3 #define ENC1 0
4
5 Dragster robot(MMAX_16_OHm);
6
7 void setup() {
8     pinMode(ENC1, INPUT);
9     robot.begin();
10    robot.drive(120, 0);
11
12    Serial.begin(115200);
13 }
14
15 void loop() {
16     Serial.println(digitalRead(ENC1));
17     delay(100);
18 }
```

1 Подключаем библиотеку Dragster.h и задаём имя пину энкодера.

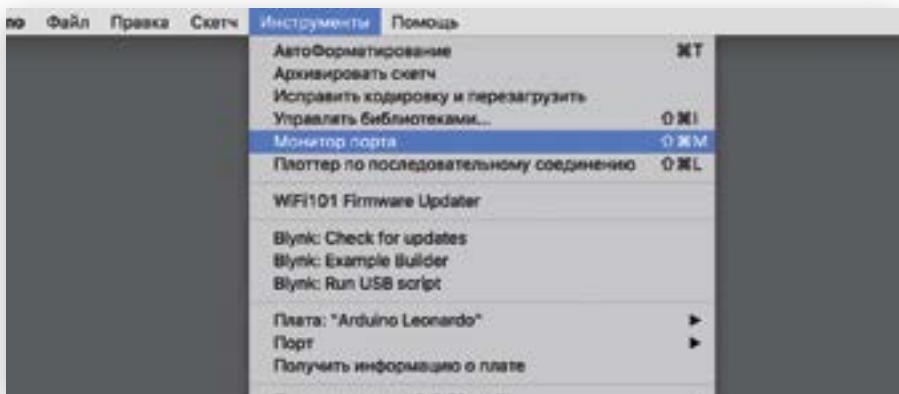
2 Устанавливаем пин энкодера на вход.

3 Запускаем последовательный порт. В скобках функции `begin` указываем скорость передачи данных – 115 200 бит в секунду.

4 Каждые 100 миллисекунд выводим на печать значение пина ENC1.

 Не отсоединяй Драгстер от компьютера!

- 2 Открой Монитор порта в меню Инструменты.



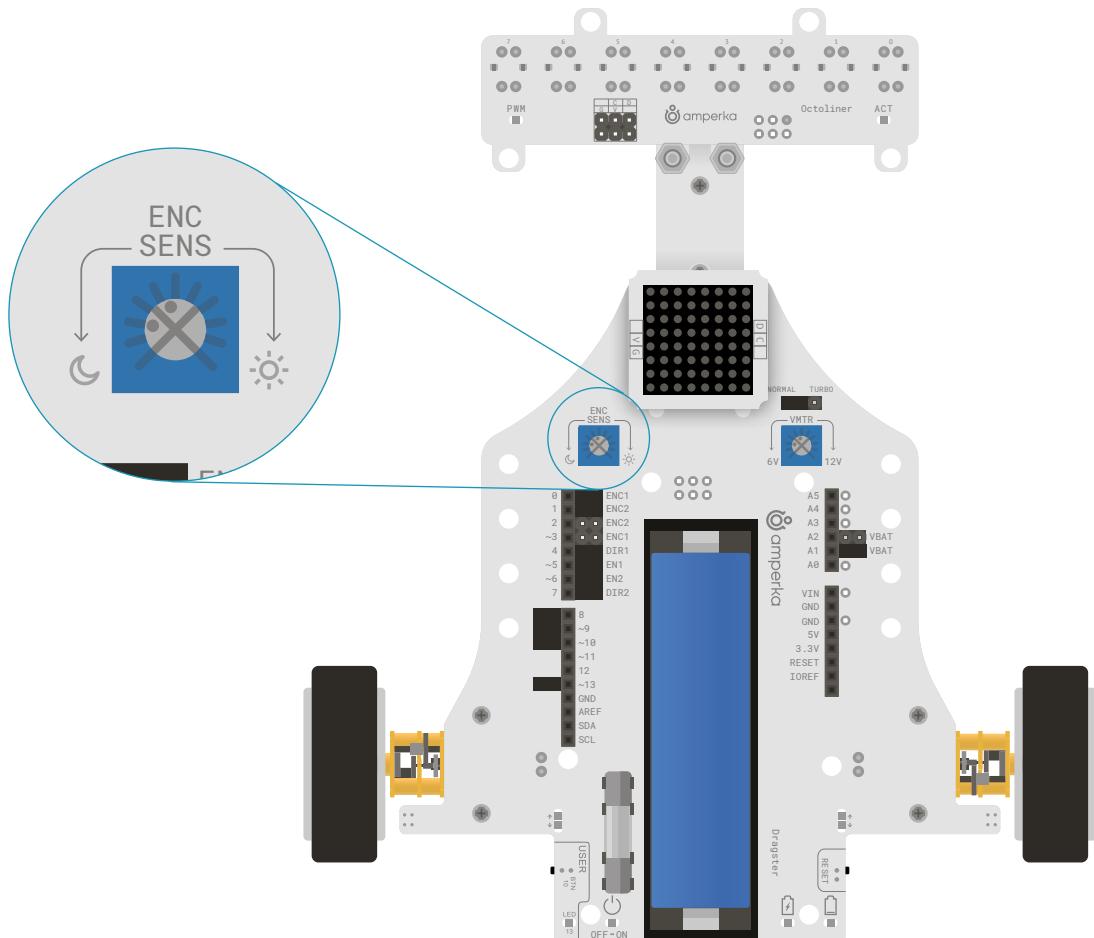
- 3 Появится окно с выводом сообщений. Установи параметр скорости передачи таким же, как в коде: 115 200.



- 4 Немного замедли колесо рукой. Обрати внимание, что изменится в выводе Монитора.

В Мониторе будут появляться данные с энкодера: 1 или 0. При вращении колеса значения должны меняться. Если этого не происходит, отрегулируй чувствительность энкодеров потенциометром ENC SENS.

Чувствительность фотоэлементов энкодеров зависит от условий освещённости в помещении. Отрегулировать чувствительность поможет потенциометр ENC SENS. Подбери подходящее положение ручки для твоих условий.



5 Добавь в код обработку второго энкодера.

```
1 #include "Dragster.h"
2
3 #define ENC1 0
4 #define ENC2 1
5
6 Dragster robot(MMAX_16_0HM);
7
8 void setup() {
9     Serial.begin(115200);
10    pinMode(ENC1, INPUT);
11    pinMode(ENC2, INPUT);
12    robot.begin();
13    robot.drive(120, 120);
14 }
15
16 void loop() {
17     Serial.print(digitalRead(ENC1));
18     Serial.print(" ");
19     Serial.println(digitalRead(ENC2));
20     delay(100);
21 }
```

1 Добавляем имя второго энкодера и номер его пина на Iskra Neo.

2 Запускаем колёса.

3 Изменяем вывод в Монитор. Теперь числа будут появляться в две колонки, разделенные символом пробела.

ЗАДАНИЕ

Добавь в проект светодиод на 13-м пине. Пусть он загорается и гаснет вместе с изменением напряжения на энкодере.

УМНЫЕ ЭНКОДЕРЫ

Научимся использовать энкодеры через прерывания.

Прерывание (англ. interrupt) – внутренний сигнал микроконтроллера, прерывающий основную программу. Используется для немедленной обработки важного события, например фиксирования изменения состояния пина.

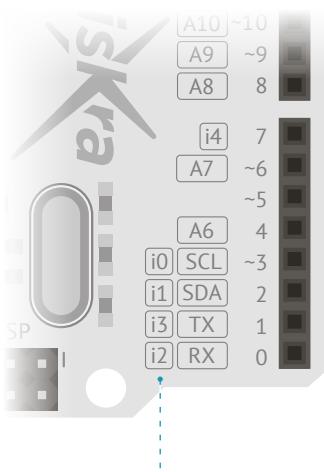
Прерывания полезны при работе с энкодерами. Если ждать, когда изменится состояние энкодера, может уйти много времени впустую. Если считывать состояние между основными задачами, можно пропустить изменения сигналов. Прерывания решают эту проблему.

Прерывание может произойти в любой момент исполнения цикла `loop`.



Библиотека *Dragster* умеет настраивать микроконтроллер на работу с пинами прерываний. Остается только написать собственную функцию-обработчик. Она будет вызываться автоматически каждый раз, когда напротив энкодера будут меняться полоски.

- 1 Напиши скетч и загрузи его в плату.



На плате Iskra Neo пины прерываний отмечены буквой *i* с номером.

```

1 #include "Dragster.h"
2
3 Dragster robot(MMAX_16_OHM);
4 int counter = 0;
5
6 void leftEncoder() {
7     counter++;
8 }
9
10 void rightEncoder() {
11 }
12
13
14 void setup() {
15     robot.begin();
16     robot.encodersBegin(leftEncoder, rightEncoder);
17     robot.drive(60, 0);
18
19     Serial.begin(115200);
20 }
21
22 void loop() {
23     Serial.println(counter);
24
25     delay(200);
26 }
```

1 Определяем переменную для подсчёта прерываний.

2 Создаём собственную функцию с увеличивающимся счётчиком прерываний для левого энкодера.

3 Функцию для правого энкодера оставим пустой. Пока достаточно одного левого энкодера.

4 Функция `encodersBegin` настраивает пины и прерывания. В скобках указываем функцию обработки.

5 Пять раз в секунду выводим на печать количество сработавших прерываний.

2 Открой Монитор порта и посмотри, как увеличивается счётчик энкодера.

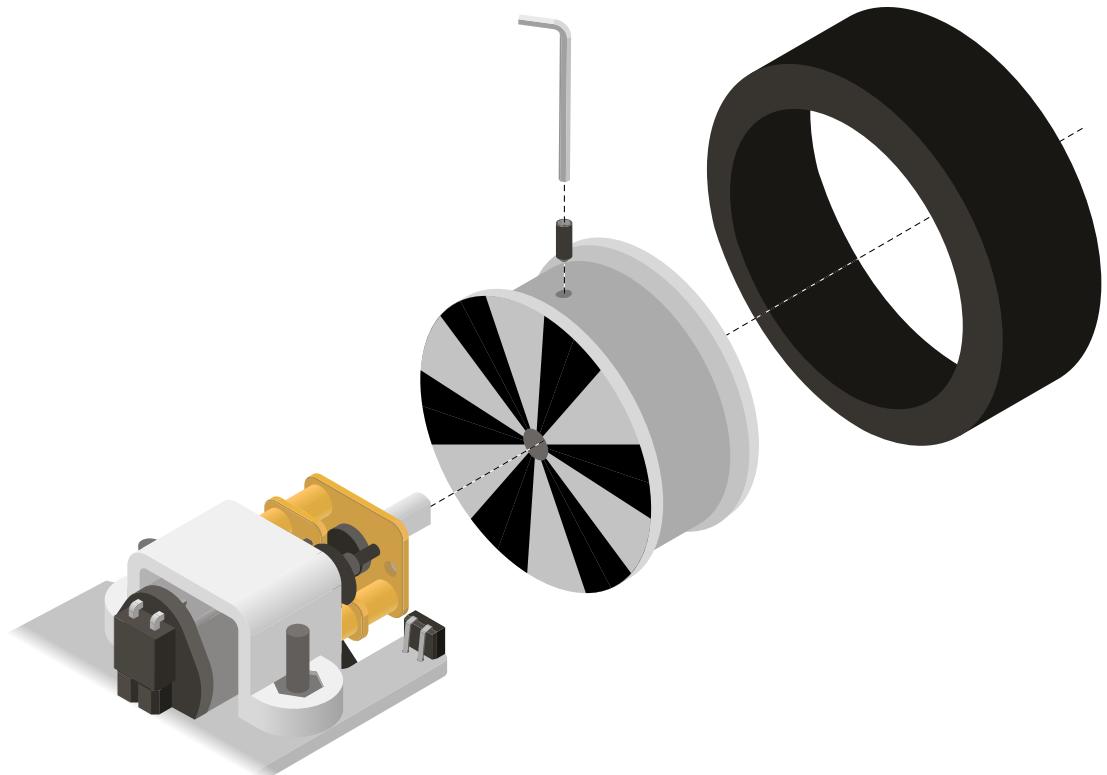
ЗАДАНИЕ

Вернись к эксперименту «Реклама спонсоров». Измени программу так, чтобы переменная `shift` увеличивалась по прерыванию от энкодера. Так ты сможешь регулировать скорость бегущей строки!

ОДОМЕТР

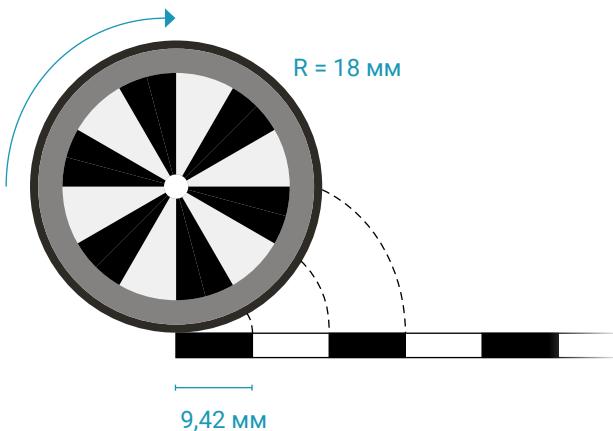
Будем измерять путь, пройденный Драгстером. Потребуются оба энкодера и совсем чуть-чуть математики.

- 1 Наклей на колёса стикеры с чёрно-белыми секторами, чтобы увеличить точность.

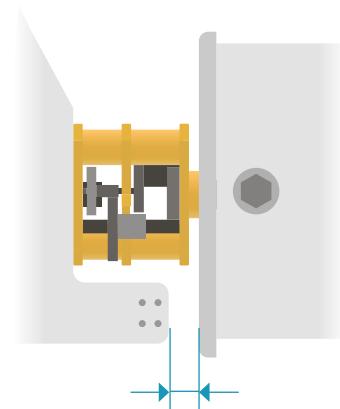


Теперь энкодер сможет замечать повороты колеса гораздо чаще, 12 раз за один оборот колеса. Зная длину окружности колеса и количество секторов, можно легко подсчитать, сколько проезжает колесо за поворот на один сектор.

$$S = \frac{2\pi R}{N_{\text{секторов}}} = \frac{2 \cdot 3,14 \cdot 18 \text{ мм}}{12} = 9,42 \text{ мм}$$



Чтобы узнать полный пройденный путь Драгстером, необходимо взять среднее значение пройденного пути каждого колеса.



Расстояние от датчика до колеса влияет на работу энкодера. Подбери оптимальное расстояние для твоих условий.

2 Напиши скетч и загрузи его в плату.

```
1 #include "Dragster.h"
2
3 Dragster robot(MMAX_16_0HM);
4 float left = 0;
5 float right = 0;
6
7 void leftEncoder() {
8     left = left + 9.42;
9 }
10
11 void rightEncoder() {
12     right = right + 9.42;
13 }
14
15 void setup() {
16     robot.begin();
17     robot.encodersBegin(leftEncoder, rightEncoder);
18     robot.drive(100, 120);
19     Serial.begin(115200);
20 }
21
22 void loop() {
23     Serial.print(left);
24     Serial.print(" ");
25     Serial.print(right);
26     Serial.println();
27     delay(20);
28 }
```

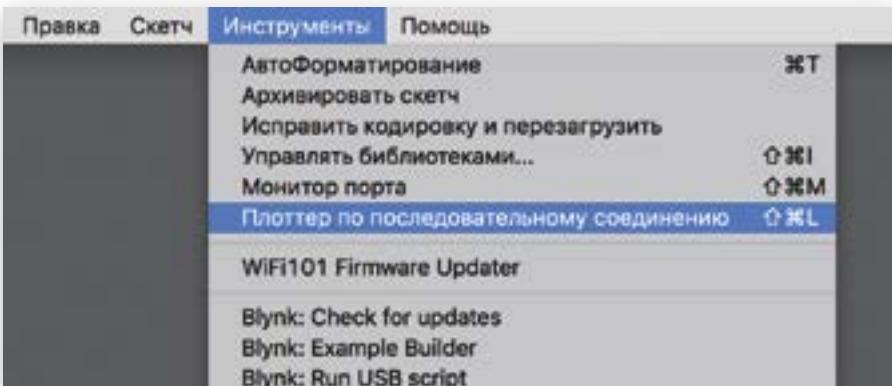
1 Объявляем переменные для хранения пройденного расстояния обоими колёсами. Тип **float** означает число с плавающей точкой, то есть действительное.

2 Накапливаем в каждом обработчике пройденное расстояние. Iskra Neo умеет хранить максимум 2 знака после дробной точки.

3 Последовательно выводим в одну строку: пройденное расстояние левым колесом, пробел, затем расстояние правого колеса. По пробельному символу Плоттер понимает, что числа в строке относятся к разным графикам.

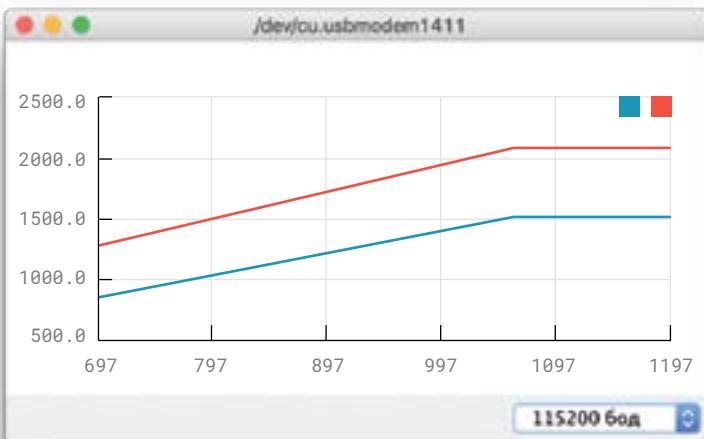
4 Печатаем символ перевода строки.

3 Открой Плоттер по последовательному соединению в меню Инструменты.



4 Откроется окно с графиком. Выставь скорость соединения 115 200 бод.

5 Запусти Драгстер, не отсоединяя его от кабеля. Смотри, как растут графики пройденного расстояния.



ЗАДАНИЕ

Добавь в проект кнопку сброса пройденного расстояния. По нажатию она должна обнулять переменные left и right.

СПИДОМЕТР

Будем замерять скорость колёс и выводить на светодиодную матрицу.

В этом снова помогут энкодеры. Они замеряют пройденный путь, и ты уже умеешь это делать, а скорость измеряется в метрах в секунду. Чтобы вычислить скорость по пройденному пути, необходимо разделить расстояние на затраченное время.

 Скорость Драгстера зависит от множества факторов: трения поверхности, износа внутренних частей двигателя и редуктора, массы робота, напряжения питания и многих других.

- 1 Измени скетч предыдущего эксперимента.

1 Используем одну и ту же функцию обработки прерываний энкодеров. Сейчас нам не важна точная скорость, поэтому просто складываем значения с каждого колеса без дополнительной обработки.

```

1 #include "Dragster.h"
2 #include "TroykaLedMatrix.h"
3
4 Dragster robot(MMAX_16_OHM);
5 TroykaLedMatrix matrix;
6 byte diagram[8];
7
8 int speed = 0;
9 void handler() {
10     speed++;
11 }
12
13 void setup() {
14     robot.begin();
15     robot.encodersBegin(handler, handler);   ······
16     robot.drive(40, 80);
17     matrix.begin();
18 }
19
20 void loop() {
21     for (int i = 0; i < 8; i++) {
22         if (speed > i) {
23             diagram[i] = matrix.map(8 - (i + 1), 0, 8);
24         } else {
25             diagram[i] = 0;
26         }
27     }
28     matrix.drawBitmap(diagram);
29     speed = 0;      ······
30     delay(150);    ······
31 }
```

2 Передаём в функцию `encodersBegin` два одинаковых параметра `handler`.

3 В цикле для каждого столбца матрицы отрисовываем индикатор скорости. «Закрашиваем» все столбики, высота которых численно ниже скорости. Функция `matrix.map(val, min, max)` преобразует число `val` в диапазон от `min` до `max`, а затем переводит результат в высоту столбика.

4 Каждые 150 миллисекунд обнуляем значение скорости.

ЗАДАНИЕ

Поставь Драгстер на пол и запусти. В зависимости от скорости будет меняться заполнение столбцов на матрице.

ТЮНИНГ. ДОМАШНЯЯ ТРАССА

Сделаем из подручных материалов гоночную трассу.

Понадобятся:



2 листа
ватмана
формата А1



Чёрная
изолента
или гуашь



Двусторонний
или обычный
скотч

1 Разложи на полу или на столе 2 листа ватмана так, чтобы они перекрывали друг друга на 5 сантиметров вдоль длинной стороны.

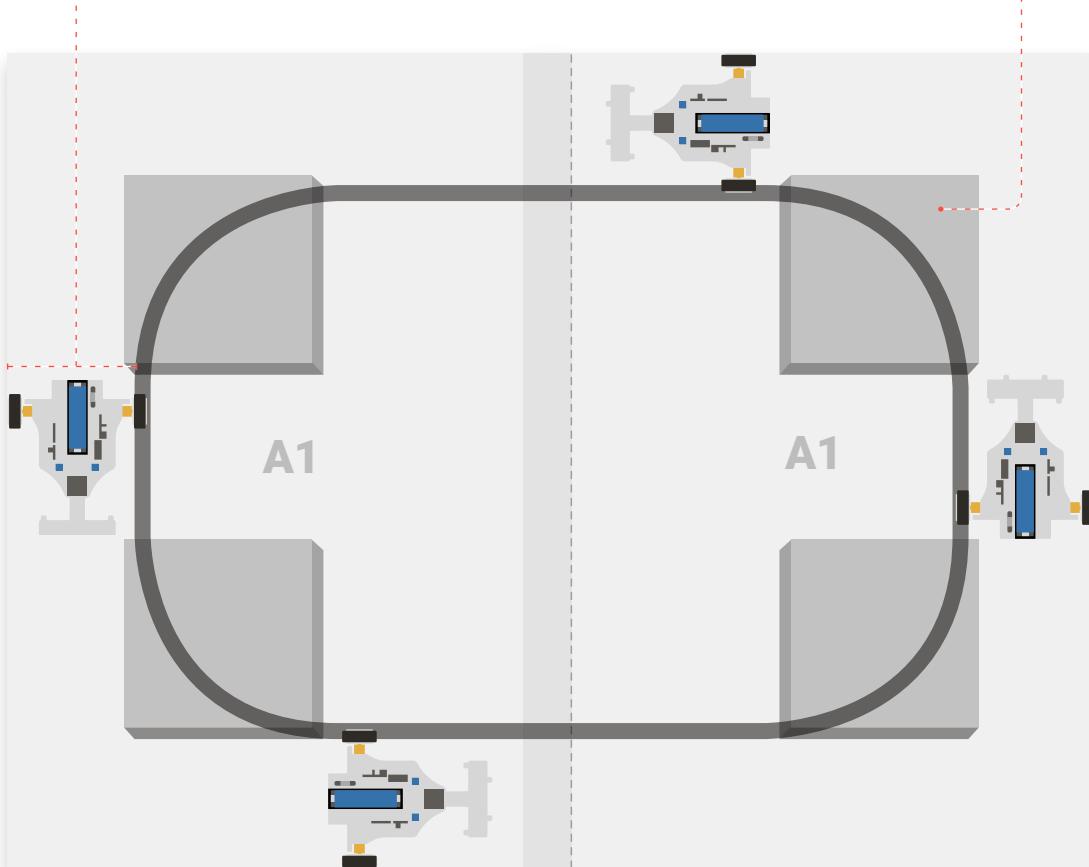
2 Соедини листы скотчем.

3 Возьми чёрную изоленту и наклей линию для Драгстера.
Соблюдай два условия:

- линия не должна приближаться к краю поля ближе чем на ширину Драгстера.
- повороты должны быть чуть больше коробки набора.

 Линия не должна приближаться к краю поля ближе чем на ширину Драгстера.

 Повороты должны быть чуть больше коробки набора.



Нахлест
5 см

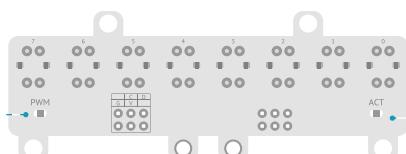
СКАНЕР ДОРОЖНОГО ПОЛОТНА

Будем сканировать гоночную поверхность домашней трассы с помощью датчика линии.

ZELO-МОДУЛЬ 8 ДАТЧИКОВ ЛИНИИ – ОКТОЛАЙНЕР

Светодиод PWM

Показывает мощность испускаемого света всеми датчиками. Мощность задаётся программно.

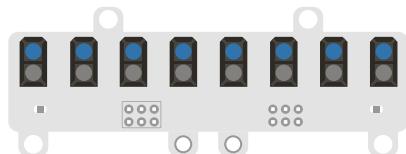


Светодиод ACT

Отображает передачу данных между модулем и Iskra Neo.

8 оптических датчиков

Каждый оптический датчик работает на отражении света от поверхности.



Контакты питания

Датчики передают модулю информацию не только в цифровом виде, но и в аналоговом. То есть они измеряют возвращённый свет не только в двоичном виде, но и могут сообщить о его интенсивности в диапазоне от 0 до 4095 единиц.

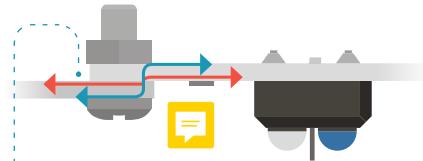


Для передачи питания и данных используются контактные площадки печатных плат и металлические винты с гайками.

КАЛИБРОВКА



Чувствительность оптических датчиков и отражающая способность каждой трассы отличаются. Чтобы контроллер и Октолайнер правильно понимали друг-друга, откалибруем чувствительность датчиков.



1 Скачай с сайта dragster.amperka.ru zip-архив с библиотекой `Octoliner` и установи её в Arduino IDE.

2 Загрузи код сканера в Iskra Neo.

```
1 #include "TroykaLedMatrix.h"
2 #include "Octoliner.h"      •••••
3
4 Octoliner octoliner;      •••••
5 TroykaLedMatrix matrix;
6
7 byte diagram[8];
8
9 void setup() {
10     matrix.begin();
11     octoliner.begin();
12     octoliner.setSensitivity();      •••••
13 }
14
15 void loop() {
16     for (int i = 0; i < 8; i++) {
17         int analogValue = octoliner.analogRead(i);      •••••
18         diagram[i] = matrix.map(analogValue, 0, 1023); •••••
19     }
20     matrix.drawBitmap(diagram);
21 }
```

5 Переводим цифровое значение напряжения в высоту столбика на матрице.

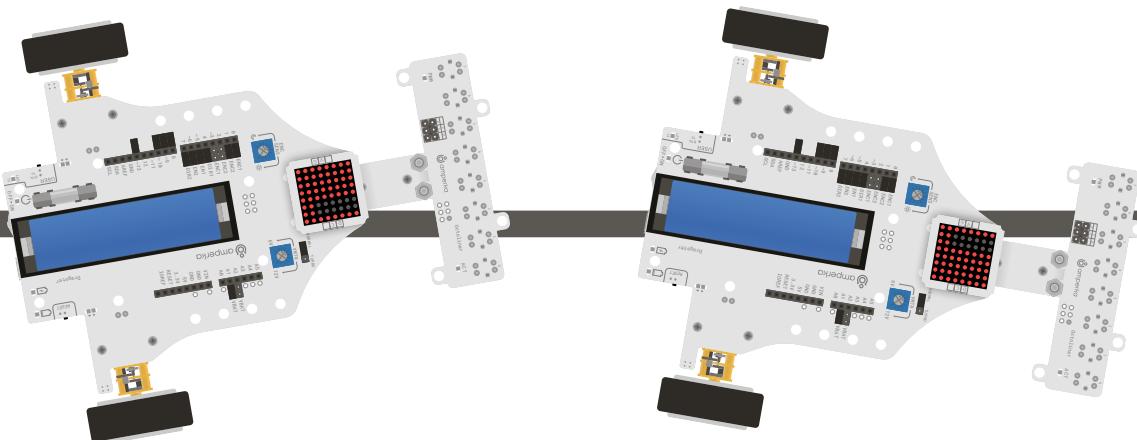
1 Подключаем библиотеку `Octoliner.h`.

2 Объявляем объект `octoliner` для работы с датчиком.

3 Запускаем датчик линии. В функцию `begin` передаём значение чувствительности датчиков в каждом из восьми датчиков модуля. Она задаётся в диапазоне от 0 до 255. Сначала впиши любое значение наугад.

4 Запрашиваем у модуля значение на каждом аналоговом приёмнике. Функция `octoliner.analogRead` вернёт целое число в диапазоне от 0 до 1023. Мы намеренно написали 1023 вместо 4095, чтобы разницу было лучше видно. В дальнейшем используй значение 4095.

- 3 Поставь робота на трассу и запусти его. Столбики матрицы должны отображать количество отражённого света на приёмниках.
- 4 Запусти код несколько раз с разным значением чувствительности в функции `octoliner.setSensitivity()`. Подбери такое значение, при котором столбики матрицы, которые соответствуют датчикам находящимся над чёрной линией, будут показывать минимальное значение – один или два «кружка».



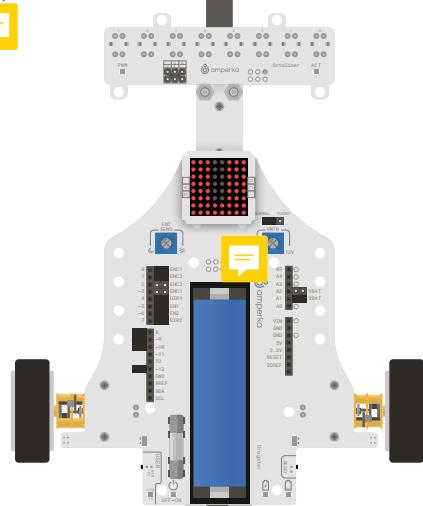
АВТОМАТИЧЕСКАЯ КАЛИБРОВКА

Калибровку нужно делать всякий раз при смене
Октолайнера и в случае, если робот плохо «видит» новую
трассу. Плохо — это например так:

Для того, чтобы не подбирать значение чувствительности
вручную, мы сделали функцию автокалибровки.

од для калибровки на следующей странице. Загрузи его
в Iskra Neo. Код длинный и непростой: можно не лениться
и напечатать его самостоятельно, но можно и скопиро-
вать с нашей вики dragster-a2.amperka.ru.

```
1 #include "Octoliner.h"
2 #include "TroykaLedMatrix.h"
3
4 const byte images[][8] = { // *1
5     {0x38, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x38},
6     {0x10, 0x30, 0x10, 0x10, 0x10, 0x10, 0x10, 0x38},
7     {0x38, 0x44, 0x04, 0x04, 0x08, 0x10, 0x20, 0x7c},
8     {0x38, 0x44, 0x04, 0x18, 0x04, 0x04, 0x44, 0x38},
9     {0x04, 0x0c, 0x14, 0x24, 0x44, 0x7c, 0x04, 0x04},
10    {0x7c, 0x40, 0x40, 0x78, 0x04, 0x04, 0x44, 0x38},
11    {0x38, 0x44, 0x40, 0x78, 0x44, 0x44, 0x44, 0x38},
12    {0x7c, 0x04, 0x04, 0x08, 0x10, 0x20, 0x20, 0x20},
13    {0x38, 0x44, 0x44, 0x38, 0x44, 0x44, 0x44, 0x38},
14    {0x38, 0x44, 0x44, 0x44, 0x3c, 0x04, 0x44, 0x38},
15    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
16    {0x38, 0x44, 0x44, 0x08, 0x10, 0x00, 0x10, 0x10}
17 };
18 byte result[5][8]; // *2
19 const int IMAGES_LEN = sizeof(images) / 8;
20 const int RESULT_LEN = sizeof(result) / 8;
21 TroykaLedMatrix matrix;
```

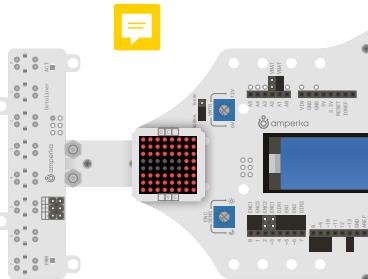


№16 Сканер дорожного полотна

```

22 Octoliner octoliner;
23 byte shift = 0;
24 byte getX_(byte c) { return c / 100;}
25 // *3
26 byte get_X_(byte c) { return (c - floor(c / 100) * 100)
27 / 10; } //
28 byte get__X(byte c) { return (c - floor(c / 10) * 10);
29 }      //
30
31 void copyPattern(byte position, byte value) { // *4
32   for(byte i = 0; i < 8; i++) {
33     result[position][i] = images[value][i];
34   }
35 }
36 void setup() {
37   byte optimal;
38   Serial.begin(115200);
39   matrix.begin();
40   octoliner.begin();
41   delay(1000);
42   while( digitalRead(10) != 0); // *5
43   matrix.drawBitmap(images[11]); // *6
44   octoliner.optimizeSensitivityOnBlack(); // *7
45   optimal = octoliner.getSensitivity();
46   Serial.print(optimal); // *8
47   copyPattern(0,10); copyPattern(1,10); // *9
48   copyPattern(2,getX_(optimal)); // *10
49   copyPattern(3,get_X_(optimal)); //
50   copyPattern(4,get__X(optimal)); //
51 }
52 void loop() {
53   delay(70);
54   matrix.marquee(result, RESULT_LEN, shift++); // *11
55   if(shift == RESULT_LEN * 8) { shift = 0; } //
56 }
```

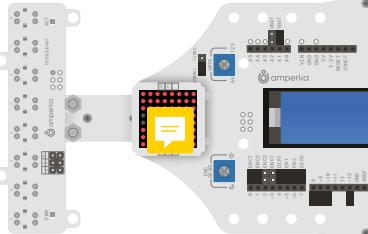
```
57 // 1. Шрифт для матричного индикатора с цифрами от 0 до  
58 // 9, пробелом и знаком вопроса.  
59 // 2. Буфер для отображения результата на матричном  
60 // индикаторе.  
61 // 3. Функции для извлечения сотен, десятков и единиц из  
62 // трехзначного числа.  
63 // 4. Функция copyPattern() переносит шаблон цифры из  
64 // буфера шрифта в буфер отображения.  
65 // 5. Ожидаем нажатия кнопки USER BTN на Драгстере.  
66 // 6. Рисуем на индикаторе знак вопроса.  
67 // 7. Ищем оптимальное значение чувствительности  
68 // октолайнера.  
69 // 8. Выводим результат на компьютер в монитор порта.  
70 // 9. Вставляем в буфер результата два пробела.  
71 // 10. Вставляем в буфер результата три цифры (сотни,  
72 // десятки и единицы).  
73 // 11. Прокручиваем бегущую строку из буфера результата.
```



Октолайнер хорошо откалиброван, значение в octoliner.setSensitivity() верное.

Сперь включи Драгстер и поставь его на трассу, так, чтобы один или два датчика были над линией, и нажми на кнопку USER BTN.

На матрице отобразится вопросительный знак, и начнется процедура калибровки. Это займет около 8 секунд. По окончании на матрице появится трёхзначное число – значение чувствительности. Запиши или запомни это число. В дальнейшем указывай это значение в команде octoliner.setSensitivity().



Нужно провести калибровку

НАСТАЛО ВРЕМЯ ГОНОК!

ДРАГСТЕР

START

isKra

СТАРТ

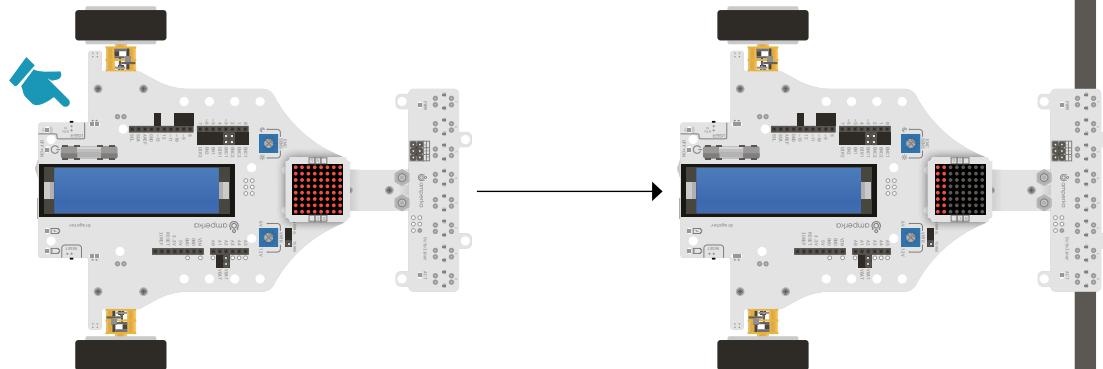
ДРАГ-РЕЙСИНГ

Напишем программу для гонок по прямой линии.

Будем давать старт по кнопке с задержкой
и останавливаться, пересекая финишную черту.

- 1 Запиши код алгоритма.
- 2 Поставь Драгстер на белое поле внутри домашней трассы.
- 3 Нажми кнопку на Драгстере. У тебя будет секунда, чтобы
убрать руку после нажатия. Робот устремится вперёд.

Как только один из датчиков увидит чёрную линию,
робот остановится.



```
1 #include "Dragster.h"
2 #include "Octoliner.h"
3 #include "TroykaLedMatrix.h"
4
5 Dragster robot(MMAX_16_OHM);
6 TroykaLedMatrix matrix;
7 Octoliner octoliner;
8
9 byte diagram[8];
10
11 void setup() {
12     robot.begin();
13     matrix.begin();
14     octoliner.begin();
15     octoliner.setSensitivity(208); -----
16 }
17
18 void loop() {
19     for (int i = 0; i < 8; i++) {
20         int adcValue = octoliner.analogRead(i);
21         diagram[i] = matrix.map(adcValue, 0, 1024);
22         if (diagram[i] < 4) {
23             robot.drive(0, 0);
24         }
25     }
26     matrix.drawBitmap(diagram);
27
28     if (robot.readButton() == 0) {
29         delay(1000);
30         robot.drive(100, 100);
31     }
32 }
```

- 1 Здесь и в дальнейшем указывай в команде `octoliner.setSensitivity()` значение полученное при калибровке ([эксперимент 16](#)).

2 Повторяем цикл из предыдущего эксперимента для обновления данных и отрисовки на матрице.

3 Добавляем проверку каждого значения. Если хотя бы одно из них меньше половины от максимума, останавливаем робота.

ЗАДАНИЕ

Пусть в конце заезда на светодиодной матрице появляется финишный флаг. Нарисуй его в редакторе изображений, скопируй массив в код и выводи сразу после остановки двигателей.

4 Если кнопка нажата, делаем паузу на секунду и даём старт роботу.

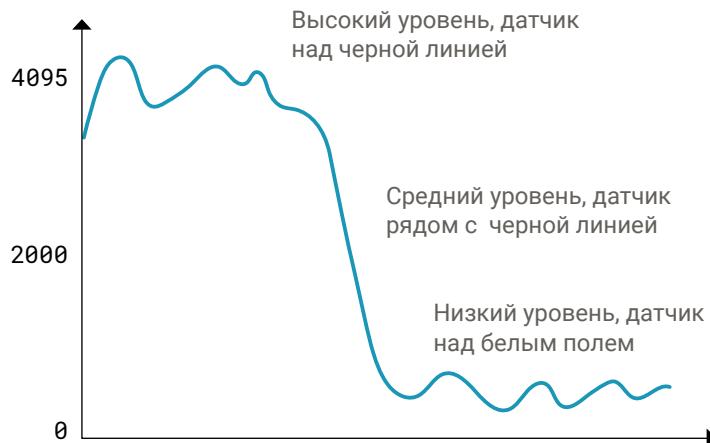
СЛЕДОВАНИЕ ПО ЛИНИИ

Научим робота следовать по линии. Сделать сразу идеально не получится, поэтому будем плавно улучшать качество алгоритма.

Начнём со сканирования поверхности в поисках чёрной линии.

- 1 Считываем напряжение на каждом датчике в аналоговом виде.
- 2 Чем выше аналоговое значение, тем выше вероятность, что под датчиком находится чёрная линия.

Аналоговое значение



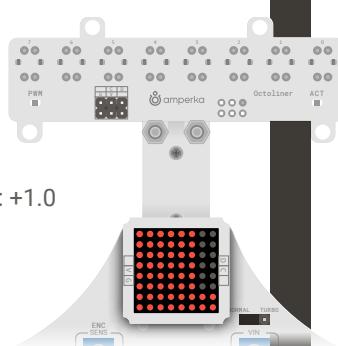
Представим положение чёрной линии относительно робота в виде одного числа от -1.0 до +1.0.

Назовём это число ошибкой.

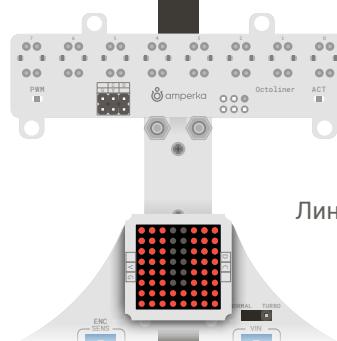
Если чёрная линия ровно под роботом, ошибки нет и число равно 0.0.



Линия слева: -1.0



Линия справа: +1.0



Линия в центре: 0.0

Превратим информацию с восьми датчиков в одно значение.
Для этого есть несколько формул, воспользуемся одной из оптимальных.

- 3 Просуммируем все значения датчиков.
- 4 Получим взвешенную сумму датчиков. Для этого каждое значение умножим на его вес, то есть на расстояние от центра робота, и сложим все значения вместе.
- 5 Разделим взвешенное значение на общую сумму. Получим одно-единственное число, обозначающее положение робота относительно линии.

Код вычисления ошибки по описанной формуле.

```
1 float weight[] =  
2     { -1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.75, 1};  
3 float sum = 0;  
4 float sumWeighted = 0;  
5 float error = 0;  
6  
7 for (int i = 0; i < 8; ++i) {  
8     adcValue = octoliner.analogRead(i);  
9     sum += adcValue;  
10    sumWeighted += sumWeighted * weight[i];  
11}  
12 if (sum != 0) {  
13     error = sumWeighted / sum;  
14}
```

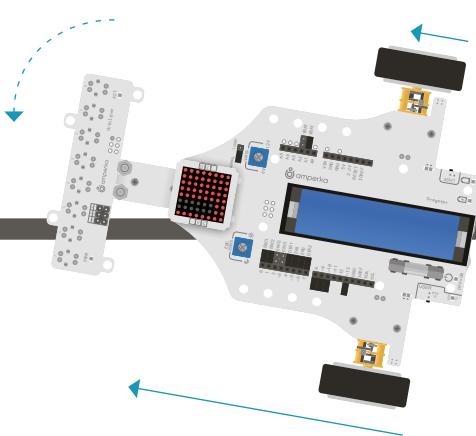
1 Создаём переменные для промежуточных результатов. Массив `weight` хранит веса датчиков, `sum` накапливает сумму значений, `sumWeighted` – взвешенную сумму. Переменная `error` хранит конечный результат отклонения робота от линии.

2 Накапливаем сумму и взвешенную сумму.

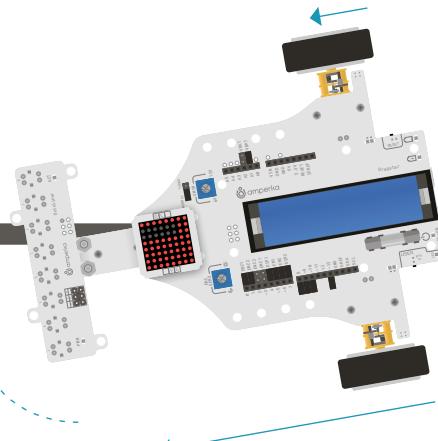
3 Получаем значение ошибки. Перед делением одного числа на другое обязательно проверяем, что делитель не равен нулю!

Зная ошибку, будем пропорционально менять скорость двигателей, чтобы робот мог вернуться на линию.

Увеличиваем скорость правого колеса



Замедляем правое колесо



Замедляем левое колесо

Увеличиваем скорость левого колеса

К основной скорости движения робота добавляем ошибку `error`, умноженную на пропорциональный коэффициент `Kp`. Он определяет силу влияния ошибки на скорость каждого двигателя. Чем больше `Kp`, тем значительней изменится скорость при той же ошибке.

```
1  float Kp = 0.3;  
2  • robot.driveF(0.35 * (1 - error * Kp), 0.35 * (1 + error * Kp));  
...
```

1 Используем функцию `driveF` вместо `drive`. Она позволяет задавать скорость колёс от `-1` до `+1`. Так

гораздо удобнее работать с ошибкой, ведь она также изменяется от `-1` до `+1`!

```

1 #include "Dragster.h"
2 #include "Octoliner.h"
3 #include "TroykaLedMatrix.h"
4
5 Dragster robot(MMAX_16_OHM);
6 TroykaLedMatrix matrix;
7 Octoliner octoliner;
8
9 byte diagram[8];
10 float weight[] = { -1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.75, 1 };
11 float Kp = 2;
12
13 void setup() {
14     robot.begin();
15     matrix.begin();
16     octoliner.begin();
17     octoliner.setSensitivity(208);
18 }
19
20 void loop() {
21     float error = 0;
22     float sum = 0;
23     float sumWeighted = 0;
24
25     for (int i = 0; i < 8; i++) {
26         int adcValue = octoliner.analogRead(i);
27         diagram[i] = matrix.map(adcValue, 0, 1024);
28         sum += adcValue;
29         sumWeighted += adcValue * weight[i];
30     }
31     if (sum != 0.0) {
32         error = sumWeighted / sum;
33     }
34     matrix.drawBitmap(diagram);
35
36     robot.driveF(0.35 * (1 - error * Kp), 0.35 * (1 + error * Kp));
37 }
```

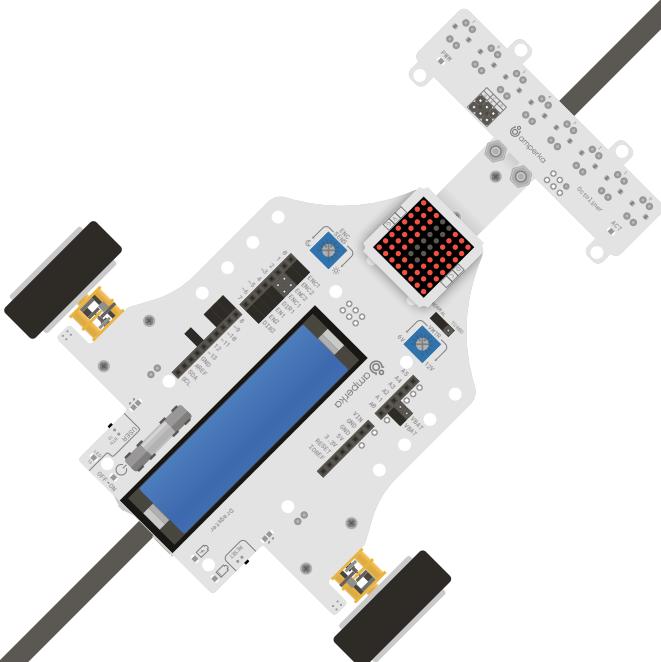
1 Переносим объявление переменных наверх для порядка.

2 Запускаем робота в полислы и изменяем скорость вращения колёс пропорционально ошибке.

- 1 Запиши полный скетч и загрузи его в Драгстер.
- 2 Запусти робота вдоль линии. На маленькой скорости и плавных поворотах он будет устойчиво следовать по маршруту, а на крутых виражах роботу может не хватать усилия для поворота.

Если увеличить пропорциональный коэффициент, робот станет лучше проходить резкие повороты, но его начнёт раскачивать. При слишком большом значении K_p он может вылететь с трассы даже на прямых участках.

- 3 Подбери оптимальное значение K_p для своего Драгстера.



ЗАДАНИЕ

Попробуй такие значения коэффициентов, не меняя скорости:

0.5

1

1.5

2

8

Запиши наблюдаемый результат напротив каждого значения.

ТЮНИНГ. ПИД-РЕГУЛЯТОР

Установим специальную библиотеку для управления роботом.

- 1 Зайди на сайт dragster.amperka.ru, скачай библиотеку PID_dragster и установи её.

ПИД-регулятор – алгоритм управления роботом по отклонению от чёрной линии. Алгоритм принимает *отклонение* (ошибку, *error*) и вычисляет скорость, на которую стоит ускорить одно колесо и замедлить другое. Это изменение скорости называется *управляющим воздействием* (*output*).

Для Драгстера с Октолайнером ошибка должна быть равна нулю (робот едет ровно вдоль линии).

ПИД – аббревиатура по первым буквам слагаемых регулятора: *пропорциональной, интегральной и дифференциальной*.

- Пропорциональная составляющая умножает текущую ошибку на коэффициент. Точно так же, как мы это делали в предыдущем эксперименте.
- Интегральная составляющая убирает накопление ошибки с течением времени.
- Дифференциальная составляющая «предсказывает» будущую ошибку и помогает противодействовать заранее.

- 2 Перепиши скетч прошлого эксперимента с использованием библиотеки.

- 3 Запусти Драгстер на поле с чёрной линией. Он должен стабильно держаться маршрута.

- 1 Подключаем библиотеку PID_dragster.h для работы с ПИД-регулятором.

```

1 #include "Dragster.h"
2 #include "Octoliner.h"
3 #include "PID_dragster.h"
4
5 Dragster robot(MMAX_16_0HM);
6 Octoliner octoliner;
7
8 int lineData[8];
9
10 float speed = 0.4;
11 float KP = 1;
12
13 double output;
14 PID regulator(&output, KP, 0, 0);
15
16 void setup() {
17     robot.begin();
18     octoliner.begin();
19     octoliner.setSensitivity(208);
20 }
21
22 void loop() {
23     for (int i = 0; i < 8; i++) {
24         lineData[i] = octoliner.analogRead(i);
25     }
26     double error = octoliner.trackLine(lineData);
27     regulator.compute(error);
28     robot.driveF(speed * (1 - output), speed * (1 + output));
29 }

```

ЗАДАНИЕ

Добавь в скетч отображение дорожного полотна под датчиком линии, как в эксперименте 18. Сделай так, чтобы чёрная линия была всегда различима на матрице.

2 Задаем значения скорости в диапазоне от 0 до 1 и пропорционального коэффициента – от 0 до 3. Так проще работать с коэффициентами при подборе параметров.

3 Настраиваем ПИД-регулятор. Переменная **output** – выходное значение регулятора, на эту величину необходимо изменить скорость двигателей. Применяем только пропорциональный коэффициент, остальные пока не используем.

4 Обновляем информацию о текущей ошибке. Записываем в переменную **error**. Вызываем функцию ПИД-регулятора для обновления выходного значения **output**.

5 Используем **output** в функции **robot.driveF()**.

ПРЕДСКАЗАНИЯ

Добавим дифференциальную составляющую в ПИД-регулятор и увеличим скорость движения Драгстера.

```

1  #include "Dragster.h"
2  #include "Octoliner.h"
3  #include "PID_dragster.h"
4
5  Dragster robot(MMAX_16_ОHM);
6  Octoliner octoliner;
7
8  int lineData[8];
9
10 float speed = 0.5;      • - - -
11 float KP = 1;           • - - -
12 float KD = 1;           • - - -
13
14 double output;
15 PID regulator(&output, KP, 0, KD);      • - - -
16
17 void setup() {
18     robot.begin();
19     octoliner.begin();
20     octoliner.setSensitivity(208);
21 }
22
23 void loop() {
24     for (int i = 0; i < 8; i++) {
25         lineData[i] = octoliner.analogRead(i);
26     }
27     double error = octoliner.trackLine(lineData);
28     regulator.compute(error);
29     robot.driveF(speed * (1 - output), speed * (1 + output));
30 }
```

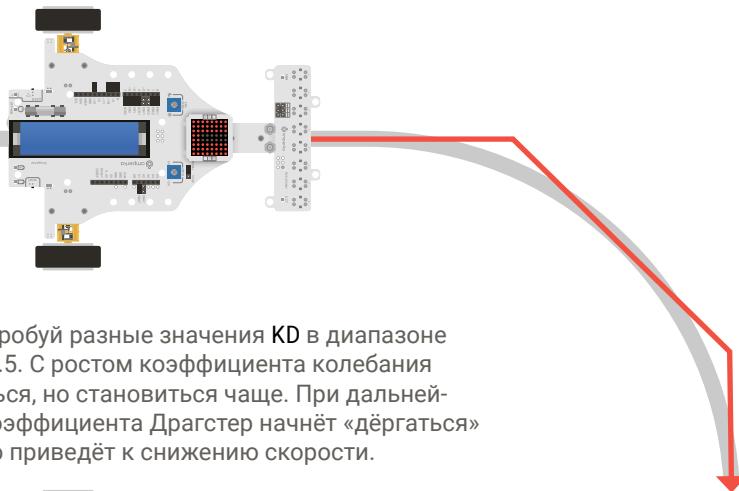
1 Немного увеличиваем скорость.

2 Добавляем дифференциальный коэффициент. Он ускорит ответную реакцию на возникающую ошибку.

3 Не забываем передать переменную в регулятор.

1 Запусти Драгстер. Теперь колебания вокруг чёрной линии станут мельче, но чаще.

2 Постепенно увеличивай KP, пока робот не станет проходить повороты, будто движется по многоугольнику.



3 После этого попробуй разные значения KD в диапазоне от 0 до 2 с шагом 0.5. С ростом коэффициента колебания должны уменьшаться, но становиться чаще. При дальнейшем увеличении коэффициента Драгстер начнёт «дёргаться» слишком часто, что приведёт к снижению скорости.



ЗАДАНИЕ

Экспериментируй с настройками KP и KD, пока Драгстер не станет двигаться вдоль линии с постоянной мелкой дрожью.

НАКОПЛЕНИЕ

Отрегулируем баланс регулятора с помощью интегральной составляющей.

```

1  #include "Dragster.h"
2  #include "Octoliner.h"
3  #include "PID_dragster.h"
4
5  Dragster robot(MMAX_16_ОHM);
6  Octoliner octoliner;
7
8  int lineData[8];
9
10 float speed = 0.55;   .-
11 float KP = 0.5;
12 float KD = 0.5;
13 float KI = 0.0;      .-
14
15 double output;
16 PID regulator(&output, KP, KI, KD);   .-
17
18 void setup() {
19     robot.begin();
20     octoliner.begin();
21     octoliner.setSensitivity(208);
22 }
23
24 void loop() {
25     for (int i = 0; i < 8; i++) {
26         lineData[i] = octoliner.analogRead(i);
27     }
28     double error = octoliner.trackLine(lineData);
29     regulator.compute(error);
30     robot.driveF(speed * (1 - output), speed * (1 + output));
31 }
```

Интегральная составляющая устраняет накопление ошибки с течением времени.

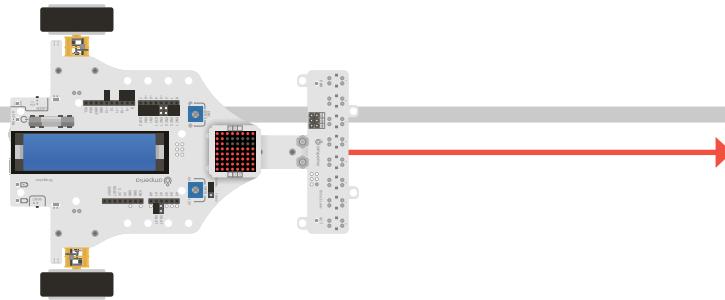
В Драгстере эта составляющая заключает в себе неточности работы Октолайнера и двигателей, учитывает повороты.

1 Увеличиваем скорость Драгстера.

2 Добавляем переменную для интегрального коэффициента. Сначала с нулевым значением, затем будем плавно увеличивать от заезда к заезду.

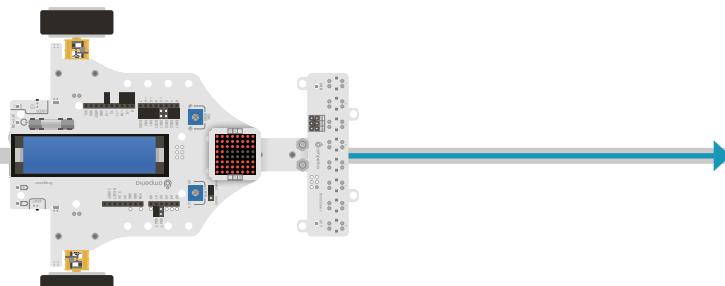
3 Заменяем 0 на переменную интегральной составляющей.

1 Запусти Драгстер с твоими коэффициентами KP и KD из предыдущего эксперимента. Плавно увеличивай коэффициент KI на 0.2. При увеличении KI Драгстер начнёт придерживаться линии на небольшом расстоянии от середины Октолайнера.

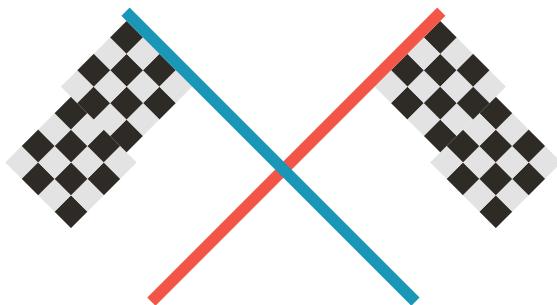


 Интегральную составляющую удобно использовать, чтобы выровнять скорость двигателей. Из-за конструктивных особенностей редукторов, моторы по-разному вращаются вперёд и назад. Поэтому у колёс может быть разная скорость при одинаковых токах и напряжениях.

При разной скорости колёс и неправильно подобранным KI Драгстер будет двигаться со смещением относительно линии.



Тот же случай, но с правильным коэффициентом KI.



ФИНИШ!



амперка

 Помощь на форуме:
forum.amperka.ru

 Электронная версия книги:
dragster.amperka.ru

 Руководства и инструкции:
wiki.amperka.ru

 Видеоканал:
youtube.com/AmperskaRU

 Новые платы и модули:
amperka.ru

 vk.com/amperkaru

 facebook.com/amperka.ru

 instagram.com/amperkaru

 twitter.com/amperka

Дизайн: студия КЛАСТЕР
www.clusterstudio.ru



амперка

ДРАГСТЕР