# РОБОНЯША

Твой первый робот

Амперка

Электронная версия robot.amperka.ru

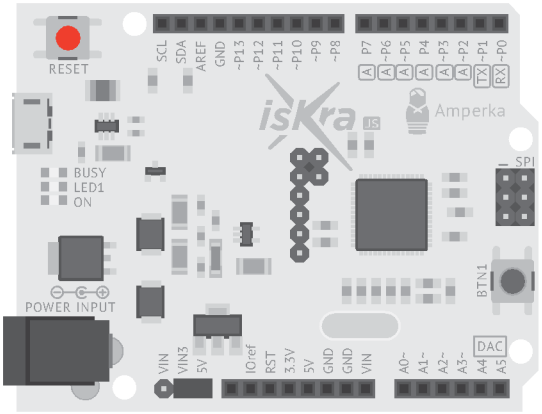
**РОБОНЯША — НАСТОЯЩИЙ РОБОТ. ОН СПОСОБЕН ВЫПОЛНЯТЬ КУЧУ РАЗНЫХ ЗАДАНИЙ В ЗАВИСИМОСТИ ОТ ЗАЛОЖЕННОЙ ПРОГРАММЫ. В ЭТОМ НАБОРЕ ТЕБЯ ЖДУТ 12 ЭКСПЕРИМЕНТОВ, В КОТОРЫХ ТЫ СОБЕРЁШЬ СВОЕГО РОБОНЯШУ И НАУЧИШЬСЯ ПИСАТЬ ДЛЯ НЕГО ПРОГРАММЫ.**

Программы для Робоняши будем писать на языке программирования JavaScript. Писать на JavaScript очень легко, особенно используя готовые *библиотеки*.

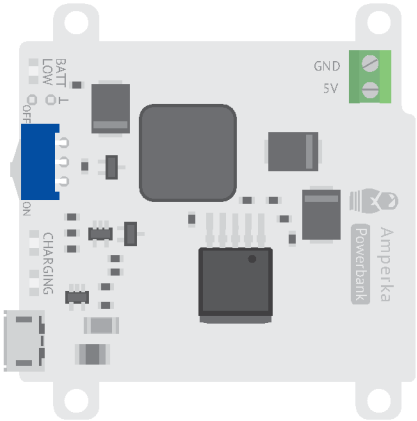
Будем вместе учиться программированию, конструированию и электронике.

# СОДЕРЖАНИЕ

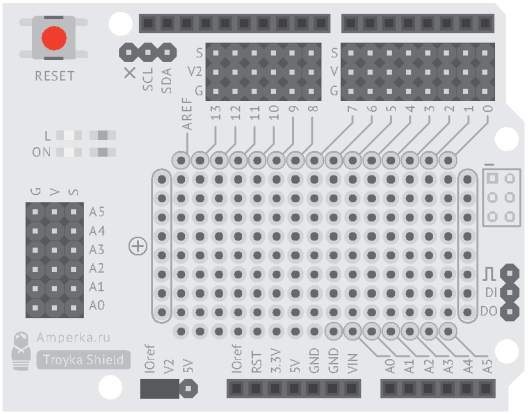
# ЭЛЕМЕНТЫ В НАБОРЕ



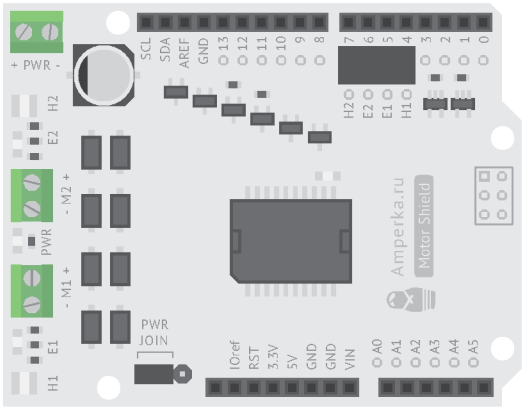

**Плата Iskra JS**
Мозг Робоняши. Выполняет программы робота




**Плата Power Bank**
Позволяет роботу ездить без подключения к компьютеру



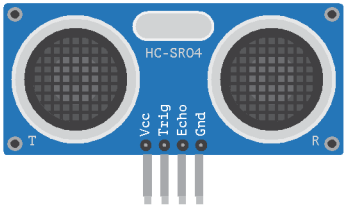

**Плата расширения Troyka Shield**
К ней подключают модули



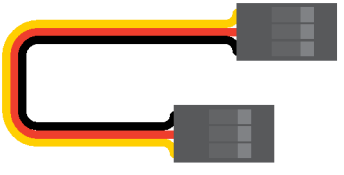

**Плата расширения Motor Shield**
Управляет двигателями

**Датчики линии цифровые (×2)**
Видят светлые предметы перед собой

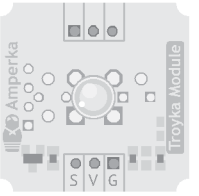
**Датчики линии аналоговые (×2)**
Такие же, как цифровые, но ещё определяют оттенки серого



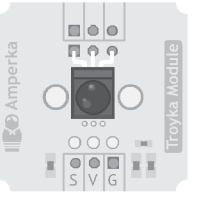

**Ультразвуковой дальномер**
Измеряет расстояние до предметов



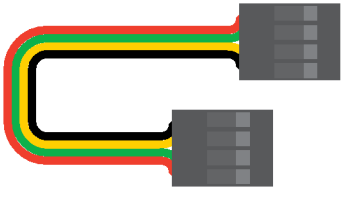
**Трёхпроводные шлейфы (×6)**
Соединяют Troyka-модули с Troyka Shield




**Troyka-модуль светодиод**
Светит по команде Iskra JS

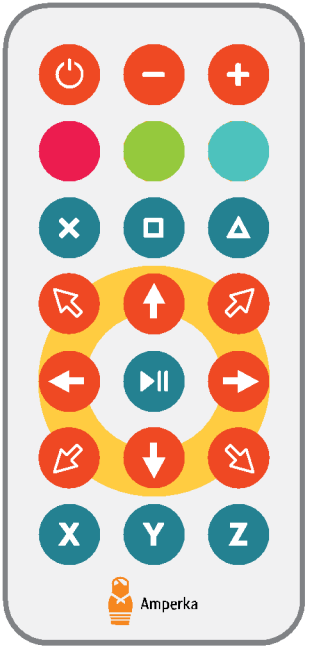



**Troyka-модуль ИК-приёмник**
Принимает команды ИК-пульта

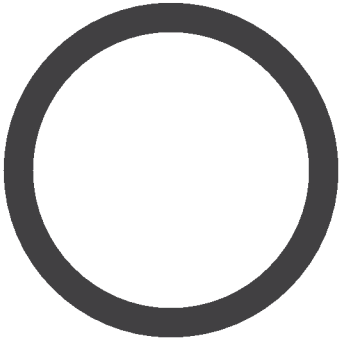


**Четырёхпроводной шлейф**
Подключает ультразвуковой дальномер

**Кабель micro-USB**
Соединяет Iskra JS с компьютером

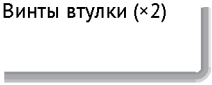



**ИК-пульт**



Шины для колёс (×2)

Винты втулки (×2)



**6-гранный ключ**
Для закручивания установочного винта



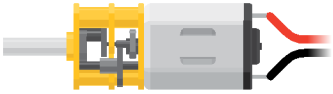
Винт (×28)



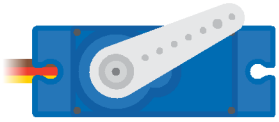
**Шаровые опоры (x2)**
Делают робота устойчивее



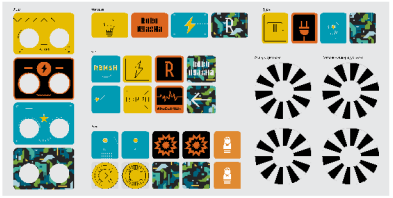
**Втулки на вал мотора (×2)**
Фиксируют колёса на валу двигателя



**Мотор-редуктор (×2)**
Приводят робота в движение



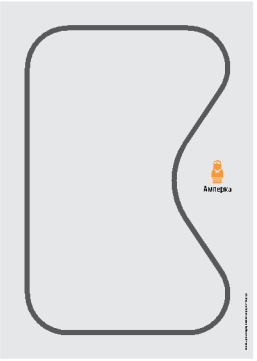
**Микросервопривод с комплектом качелек**
Поворачивается к заданному углу



**Стикеры**
Для красоты и работы некоторых датчиков

**Отвёртка крестовая**
Не раз пригодится во время сборки

**Нож для резки #Структора**




**Карта с чёрной линией**

**#Структор (×4)**
Конструктор для сборки корпуса

# ISKRA JS

Iskra JS — это маленький компьютер, мозг твоего устройства. Он умеет измерять напряжение, умеет его выдавать. Этого достаточно, чтобы взаимодействовать с внешним миром: считывать всевозможные сенсоры, выдавать команды на реле, моторы, светодиоды, дисплеи.

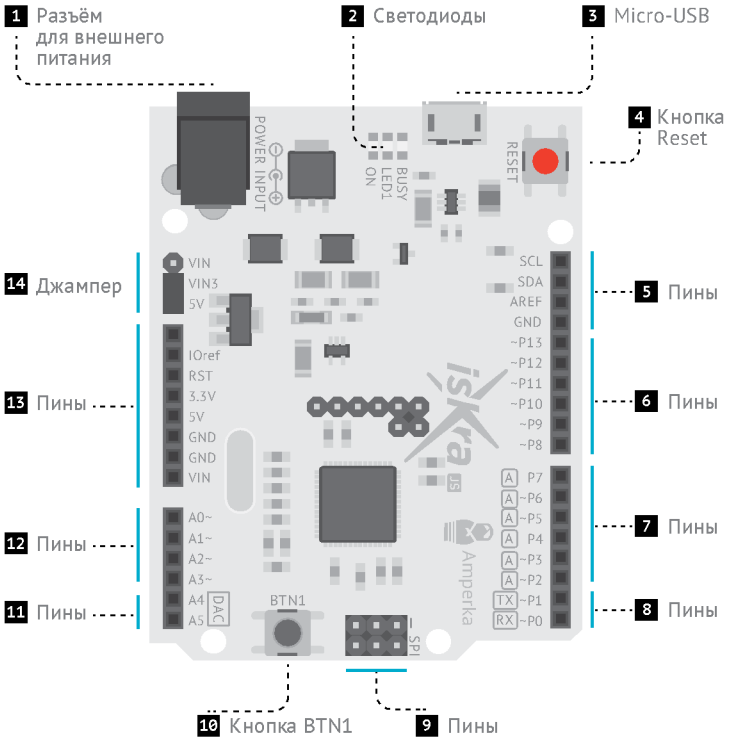
Чтобы получилось законченное устройство, нужно описать его поведение. Это поведение ты можешь запрограммировать на языке JavaScript, загрузить программу в плату и таким образом получить уникальный самостоятельный гаджет!

**1**
Разъём для питания платы от розетки или аккумулятора.

**2**
Светодиоды:
• ON — горит, когда на плату поступает питание;
• LED1 — можешь использовать по собственному усмотрению;
• BUSY — включается, когда микроконтроллер производит вычисления или передачу данных.

**3**
Разъём micro-USB – через него загружают программу, передают данные обратно на компьютер и питают плату.

**4**
Кнопка RESET — перезагружает плату, последняя сохранённая программа начинает исполняться с начала.



5 6 7 8 9 11 12 13

Колодки с контактами вдоль длинных сторон — это входы и выходы платы. Их называют *пинами*.

6 7 8 11 12

Пины P0...P13 и A0...A5 используют для взаимодействия с модулями, платами расширения и другими устройствами. Их называют *портами GPIO* (General Purpose Input and Output) или просто *портами*.

Порты GPIO в режиме входа могут определять, есть ли на них напряжение 3,3 вольта или его нет. В режиме выхода порты могут либо выдавать 3,3 вольта, либо выдавать ноль.

7 11 12

Пины A0...A5 и те, что отмечены A, умеют измерять точную величину входного напряжения. К ним подключают аналоговые сенсоры, о которых ты вскоре узнаешь.

6 7 8 12

Те, что отмечены ~ (тильда), умеют переключать выходное напряжение между 0 и 3,3 вольта тысячи раз в секунду. Это используют, чтобы влиять на яркость светодиодов, скорость моторов, мощность магнитов и т. п.

5 6 9 11

Пины DAC, RX, TX, SPI, SDA, SCL обладают дополнительными функциями, которые нужны при работе с некоторыми модулями.

10

Кнопка BTN1 — можешь использовать по собственному усмотрению.

13

Пины питания:

- 3.3V и IORef выдают ровные и родные для Iskra JS 3,3 вольта.
- 5V выдаёт ровные 5 вольт для модулей, которым не хватает штатных 3,3 вольта.
- VIN выдаёт входное напряжение как есть. То, что приходит через разъём USB или разъём внешнего питания.
- GND — земля платы, точка отсчёта напряжения, 0 вольт.

14

*Джампер,* который можно перекидывать между двумя положениями:

- VIN3+5V — обычный режим.
- VIN3+VIN — питание от низковольтного аккумулятора.

### ТРЮК

Если зажать BTN1 на момент нажатия RESET, плата после перезагрузки не начнёт исполнять программу, а перейдёт в режим ожидания перепрошивки: светодиоды LED1 и BUSY будут попеременно мигать. Если после этого нажать BTN1 ещё раз, плата перейдёт в обычный режим, но не будет исполнять программу, сохранённую в флеш-памяти. Это полезно, если ты написал такой алгоритм, который «вешает» плату.

# ПЕРВЫЙ ЗАПУСК

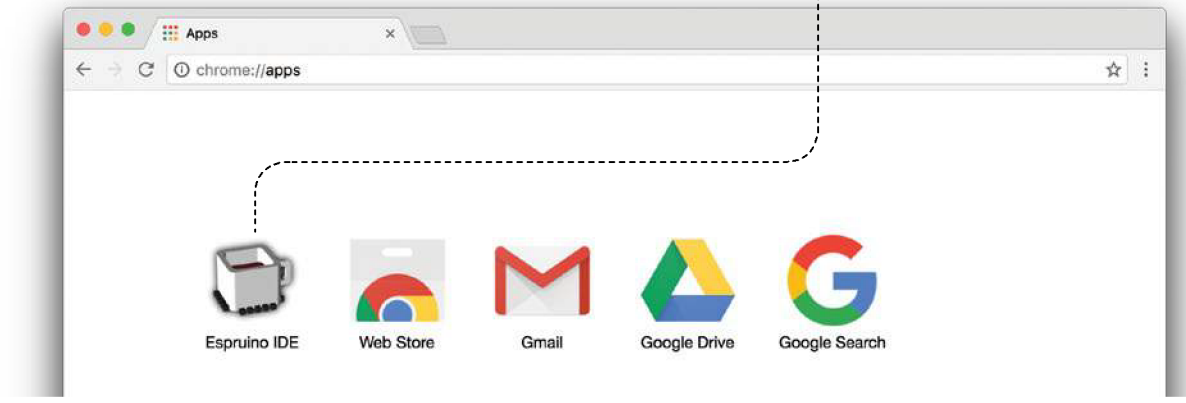
Ни один робот не работает без внутренней программы. Давай для начала потренируемся её создавать. Программы пишут в специальных редакторах кода. Мы будем писать код в Espruino IDE, этот редактор умеет загружать написанную программу в Iskra JS.

1 Если у тебя ещё не установлен Google Chrome, установи его: google.com/chrome. Не беспокойся, если пользуешься другим браузером: привычки менять не придётся. Google Chrome нужен лишь как платформа для среды программирования.

2 Установи Espruino IDE с сайта wiki.amperka.ru/js:ide.

Выполни 3 шага в разделе «Установка и Настройка».

3 Запусти среду. Перейди на вкладку приложений Google Chrome (chrome://apps) и кликни по иконке приложения.




**ТРЮК**

Кликни правой кнопкой по иконке приложения и создай его ярлык на рабочем столе, чтобы запускать приложение напрямую, без запуска браузера Google Chrome.

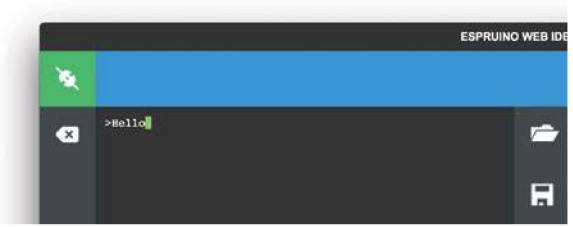
4 Подключи Iskra JS через кабель micro-USB к своему компьютеру, нажми на кнопку соединения и выбери порт.
— *COMx* на Windows;
— */dev/cu.usbmodemXXXX* на Mac OS;
— */dev/ttyACMx* на Linux.
Всё, мы готовы к работе.


Левая панель — это окно *консоли*. Сюда ты можешь на лету посылать инструкции JavaScript, плата их будет исполнять и посылать результат обратно на компьютер.

Правая панель — это окно *программы*. Здесь ты работаешь над кодом. Клик по кнопке заставит среду загрузить код программы в плату, а та начнёт его исполнять.

# НЕМНОГО О JAVASCRIPT

JavaScript — популярный язык программирования. Он простой, выразительный и гибкий. JavaScript обычно используют для создания веб-приложений, но его также может исполнять и твоя Iskra JS.




5 Смелее, попробуй! В окне консоли (та, что слева в IDE) введи `"Hello!"` и нажми *Enter*, чтобы отправить строку на плату:

```
>"Hello!"
```

Мгновение спустя, ты увидишь ответ платы:

```
="Hello!"
```

Мы поздоровались. Это означает, что всё настроено правильно и мы готовы к чему-то более серьёзному.

## АРИФМЕТИКА

JavaScript знает об арифметике. Введи выражение и нажми *Enter*:

```
>15 + 13 - 3
=25
```

Ура! Плата получила твою инструкцию, исполнила её и вернула результат «25» обратно. Давай ещё:

```
>5 + (4 - 1) * 3 / 2
=9.5
```

## ПЕРЕМЕННЫЕ

Чуть усложним. Добавим *переменных*. Переменная — это значение, которому мы дали имя. Чтобы создать новую переменную, используй слово `var` (от англ. *variable*).

```
>var x = 3
=3
>var y = 4
=4
>x * x + y * y + 5
=30
>x = y + 1
=5
>x * x + y * y + 5
=46
```

Переменные могут хранить не только числа, но и строки, логические значения, функции, составные объекты. Обо всём этом ты узнаешь в своё время.

## ФУНКЦИИ

А сейчас давай попробуем сделать что-то специфичное для платы:

```
>getTime()
=1425.66402724404
```

Мы вызвали встроенную *функцию* с именем `getTime`. Она возвращает время в секундах, прошедшее с момента включения платы. Попробуй вызвать её несколько раз и посмотри, как меняется результат.

## ОБЪЕКТЫ И МЕТОДЫ

Теперь попробуем порулить электричеством!

```
>LED1.write(1)
=undefined
```

А сейчас переведи свой взгляд на плату: светодиод LED1 горит! Давай выключим:

```
>LED1.write(0)
=undefined
```

Теперь попробуй сам помигать светодиодом.

Разберёмся, что здесь происходит. Мы обращаемся к встроенной переменной-объекту `LED1` и вызываем *метод* `write` с *параметром* `1`... Обо всём по порядку.

Метод — это функция, которая работает с объектом, на котором её вызвали.


В скобках мы передаём методу параметры. У `LED1.write` параметр один — значение, которое определяет, нужно ли выключить светодиод (ноль) или включить (не ноль).

Читай инструкцию целиком так:
«Эй, светодиод1, запиши-ка мне единичку».

И наконец, ответ — `undefined`. В JavaScript это означает «пусто», «ничего», «дырка от бублика». Функция `getTime` возвращала нам время, а `LED1.write` не возвращает ничего, поэтому мы видим `undefined`.

## ПОПРОБУЕМ ЕЩЁ?

`BTN1` — встроенный объект, который олицетворяет кнопку на Iskra JS. Метод `read` считывает текущее состояние кнопки. Параметры ему не нужны.

```
>BTN1.read()
=false
```

Метод вернул `false`. Это логическое значение, которое означает «ложь», «нет», «ноль»: кнопка не нажата.

Теперь зажми кнопку и выполни инструкцию ещё раз:

```
>BTN1.read()
=true
```

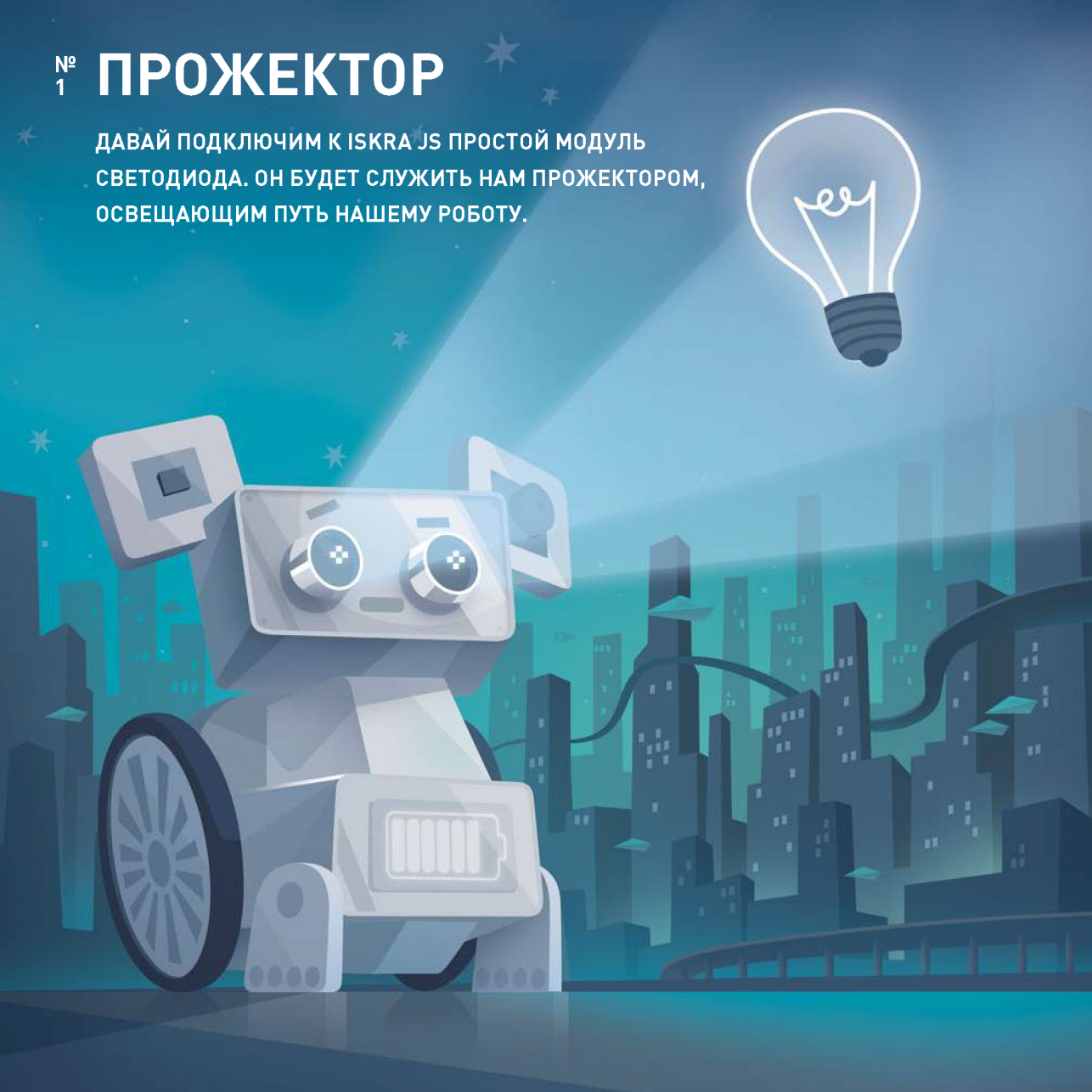
О! Теперь мы видим `true`. Это «истина», «да», «единица»: кнопка зажата.

### ЕЩЁ БОЛЬШЕ ФУНКЦИЙ

Встроенных функций и методов много. У них разные параметры. Как ими пользоваться, что они делают, ты можешь узнать на js.amperka.ru.

# № 1 ПРОЖЕКТОР

ДАВАЙ ПОДКЛЮЧИМ К ISKRA JS ПРОСТОЙ МОДУЛЬ СВЕТОДИОДА. ОН БУДЕТ СЛУЖИТЬ НАМ ПРОЖЕКТОРОМ, ОСВЕЩАЮЩИМ ПУТЬ НАШЕМУ РОБОТУ.



## ОБ ЭЛЕКТРИЧЕСТВЕ

Любому электронному компоненту для работы нужно питание. Питание подводят по двум проводам. Помнишь, как выглядит розетка? По одному проводу электричество втекает, по другому — вытекает.

*Плюсом* называют тот провод, по которому ток втекает. Этот провод обычно делают ● красным. Ещё этот провод называют +V (англ. *voltage* – напряжение), или Vcc, или просто *питанием*, или же сколькими-то вольтами (например, родными для Iskra JS 3,3 вольта).

*Минусом* называют провод, по которому ток вытекает. Этот провод также называют *землёй* (англ. *ground* — земля, сокращённо GND) или *нулём вольт*. Чаще всего его делают ● чёрным.

> Не путай *землю* устройства и планету Земля. В слаботочных устройствах эти понятия никак не связаны.

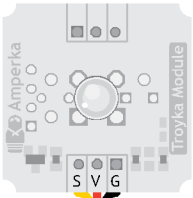
Электричество передают не только по проводам, но и по дорожкам на плате, через разъёмы и даже по воздуху. Поэтому вместо слова «провод» также употребляют слово «линия»: линия питания, линия земли.

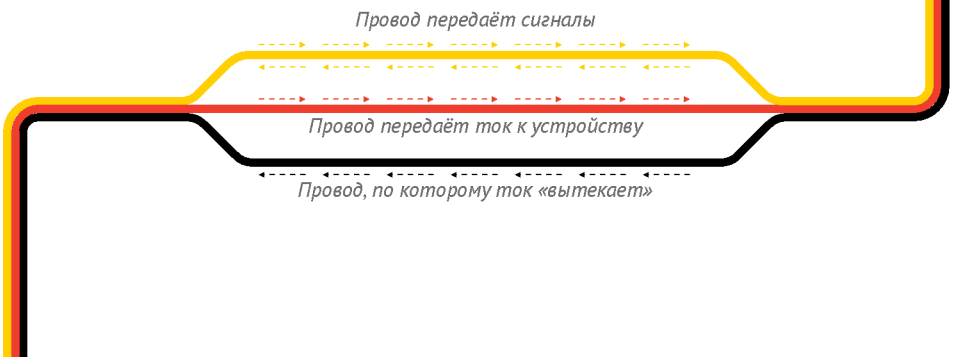
## СИГНАЛЫ

Просто включить устройство — хорошо, но как с ним можно общаться? Электронные устройства передают друг другу информацию, используя различные сигналы. Сигналы передаются по отдельным проводам — сигнальным. В большинстве модулей ты увидишь третий, *сигнальный,* ● жёлтый провод.

● G – земля (чёрный провод)
● V – питание (красный провод)
● S – сигнал (жёлтый провод)

Светодиод и трёхпроводной шлейф

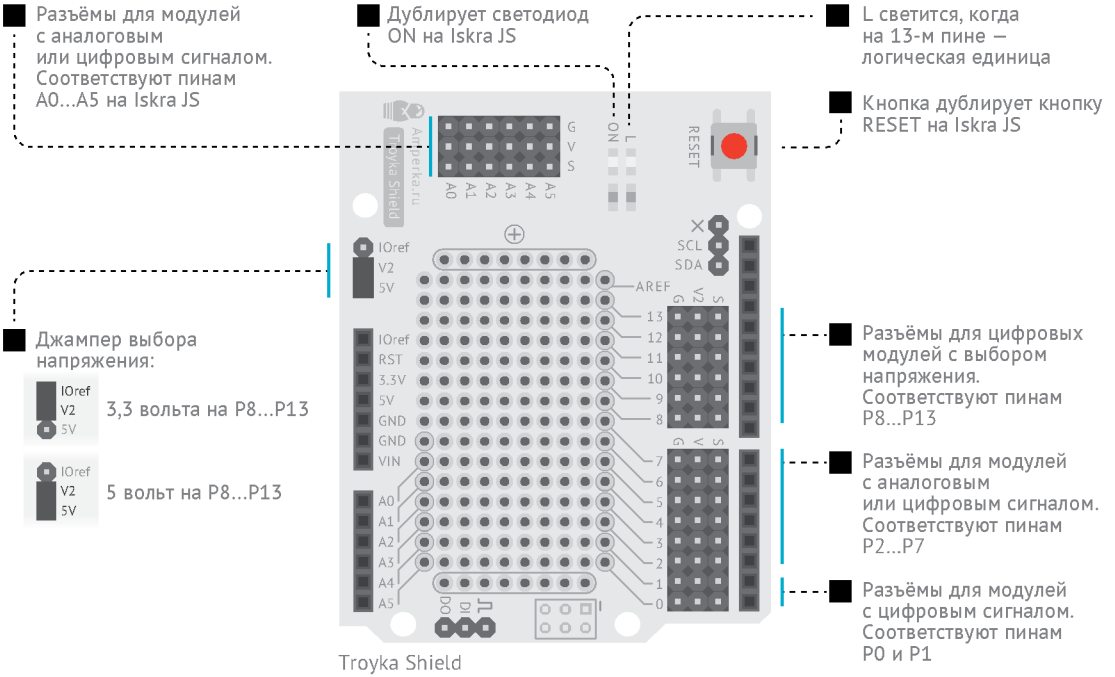


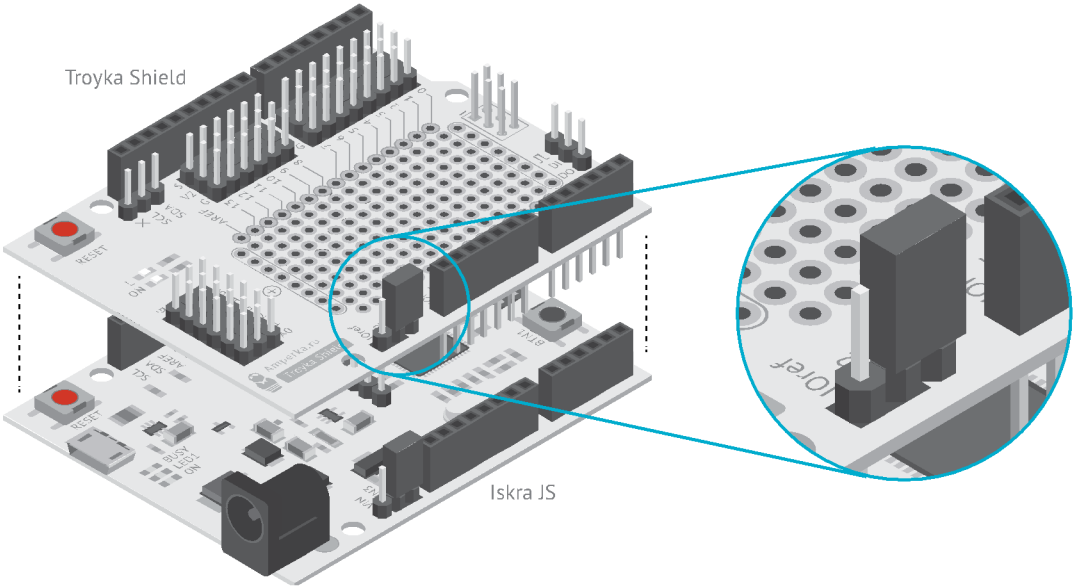
## TROYKA SHIELD

На Iskra JS — 20 сигнальных пинов, а пинов питания всего несколько. Когда ты захочешь подключить десяток-другой сенсоров и модулей, придётся думать о том, как разветвить линии GND и 3.3V: питание ведь нужно каждому модулю. Чтобы об этом не думать, есть Troyka Shield.

Это плата расширения для Iskra JS, которая каждый пин P0...P13 и A0...A5 превращает в тройной разъём «земля-питание-сигнал». Теперь достаточно соединить разъём на Troyka Shield и модуль трёхпроводным шлейфом, и он готов к работе.



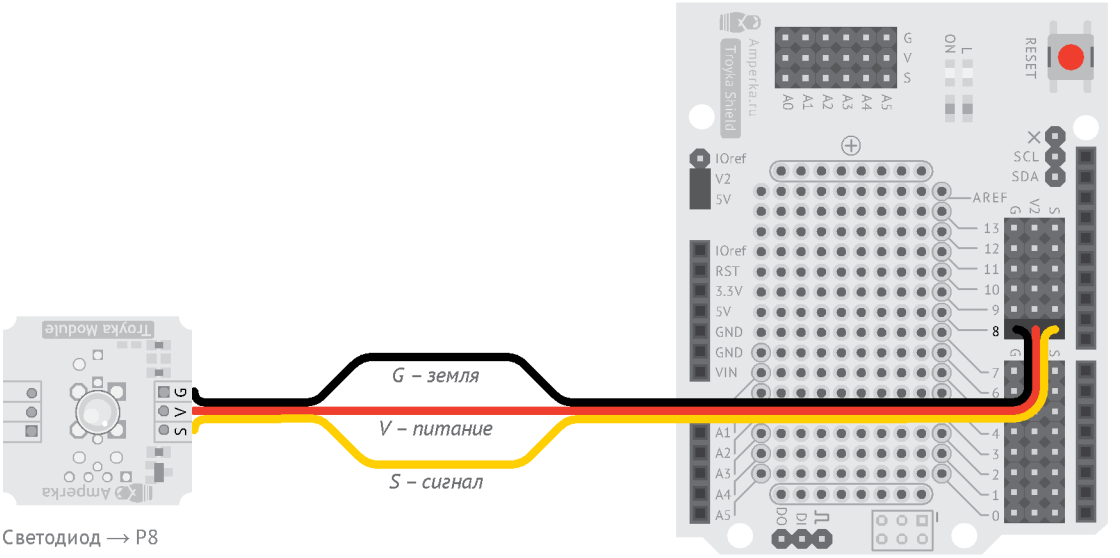
6 Вставь Troyka Shield в пины Iskra JS сверху. Ты получишь единое устройство. Для соединения с компьютером, как и раньше, используй USB-порт на Iskra JS. А для подключения модулей используй разъёмы на Troyka Shield.




## НАПРЯЖЕНИЕ ПИТАНИЯ

По линиям, отмеченным *V*, Troyka Shield подводит родные для Iskra JS 3,3 вольта. Но некоторым модулям недостаточно этого напряжения для работы. Поэтому на пинах P8...P13 предусмотрено альтернативное напряжение V2. В наших проектах мы будем использовать ультразвуковой дальномер и сервопривод, которым требуется 5 вольт, поэтому установи джампер в положение V2+5V.

**7** Модуль светодиода необходимо подключить в разъём P8 на плате Troyka Shield, как показано на рисунке.




Светодиод ⟶ P8

Troyka Shield + Iskra JS

8 Создай программу для включения прожектора. Напиши целиком этот код в правой панели Espruino IDE и кликни кнопку «загрузить»


```
1  var projector = require('@amperka/led').connect(P8);
2  projector.turnOn();
```

a Создаём переменную с именем `projector` и говорим, что это светодиод, который подключён к пину P8. `require` означает, что мы хотим использовать модуль. `'@amperka/led'` — что это модуль управления светодиодом.

`connect(P8)` — что он подключён к восьмому пину.

Мы сохраняем объект-светодиод в переменной, которой дали имя `projector`. Придумай другое имя, если хочешь.

b У объекта-светодиода вызываем метод `turnOn`, чтобы включить его.

У каждого модуля есть свой набор методов. Подробнее о всех методах смотри на сайте js.amperka.ru.

**ПОДУМАЙ САМОСТОЯТЕЛЬНО**

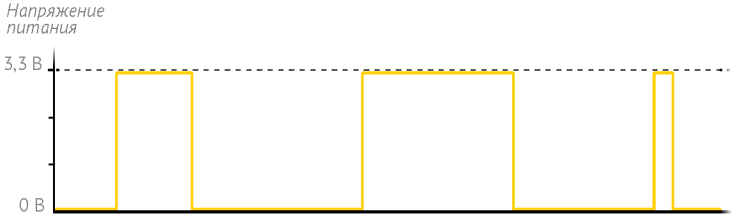
Попробуй зажечь прожектор с других пинов (P0, P1, P2 и т. д.).

Если не получилось, вернись на стр. 10 и проверь, всё ли правильно сделал.

# № 2 СИГНАЛЬНАЯ КОЛОННА

ПРИВЛЕЧЁМ К РОБОТУ ПОБОЛЬШЕ ВНИМАНИЯ, ДАДИМ ЗНАТЬ ВСЕМ ОКРУЖАЮЩИМ О ПРИБЛИЖЕНИИ РОБОТА. СДЕЛАЕМ ДЛЯ ЭТОГО ПРОЖЕКТОР МИГАЮЩИМ! БУДЕМ ТО ВКЛЮЧАТЬ, ТО ВЫКЛЮЧАТЬ ЕГО.

## ЦИФРОВЫЕ СИГНАЛЫ

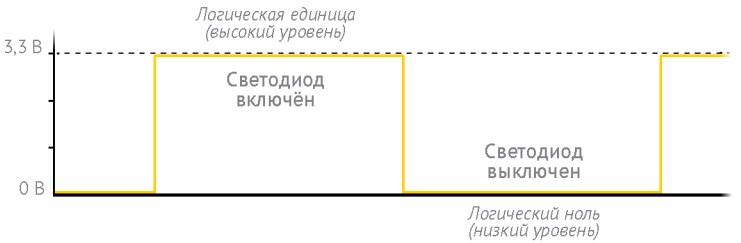



Периодически включать и выключать прожектор можно с помощью *цифровых сигналов*. В цифровых сигналах величина напряжения в любой момент времени равна либо 0 вольт, либо напряжению питания.

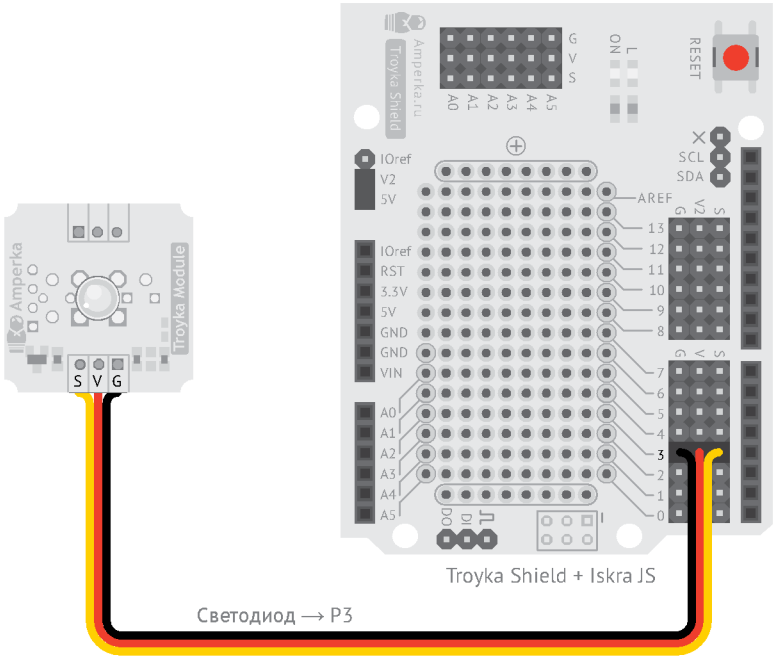
Напряжение питания на сигнальной линии при этом называют *логической единицей*, а ноль вольт — *логическим нулём.*

При помощи цифровых сигналов можно общаться с коллекторным двигателем, сервоприводом, ИК-приёмником, светодиодом, цифровым датчиком линии, дальномером.

Цифровые модули можно подключать к любым портам Iskra JS.



**9** Теперь подключи светодиод к пину P3.




```
var notice = require('@amperka/led')
  .connect(P3);

notice.blink(0.1, 0.9);
```

**a** Для стройности можно разбивать выражения на несколько строк.

Чтобы код было легче читать, отбивай дополнительные строки двумя пробелами. Главное, не забывай ставить «;» (точку с запятой) в конце.

**b** Заставим светодиод мигать! Пусть он светится одну десятую секунды, а затем гаснет на девять десятых. И так по кругу. Именно это делает метод `blink`.

Если не передать в `blink` второй параметр, светодиод моргнёт всего один раз.

## ПОРАБОТАЕМ С КОНСОЛЬЮ

Изменить период мигания светодиода можно и без перезагрузки кода. Воспользуемся консолью в среде Espruino Web IDE. Напиши в левой панели строчку:

```
>notice.blink(1, 2);
```

И нажми кнавишу *Enter*. Светодиод начнёт мигать раз в три секунды. Попробуй задать другие значения для функции `blink()`.


### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Измени темп мигания светодиода. Пускай мигает 4 раза в секунду.

№ 3

# СЕНСОРНЫЙ ВЫКЛЮЧАТЕЛЬ

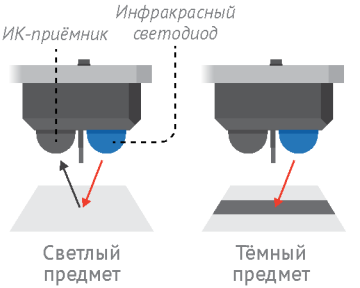
НАУЧИМСЯ ВКЛЮЧАТЬ И ВЫКЛЮЧАТЬ ПРОЖЕКТОР С ПОМОЩЬЮ СЕНСОРНОЙ КНОПКИ. ДАВАЙ ПРИСПОСОБИМ ДЛЯ ЭТИХ ЦЕЛЕЙ ЦИФРОВОЙ ДАТЧИК ЛИНИИ. ПОЛУЧИТСЯ ОПТИЧЕСКИЙ БЕСКОНТАКТНЫЙ ВЫКЛЮЧАТЕЛЬ.

## ЦИФРОВОЙ ДАТЧИК ЛИНИИ

Датчик линии состоит из инфракрасных светодиода и приёмника. Светодиод излучает свет, а приёмник пытается этот свет уловить. Если рядом с датчиком поставить светлый предмет, то свет от диода отразится и приёмник его уловит.

Если же перед датчиком поставить тёмный предмет или вовсе ничего не поставить, свет от диода не вернётся и приёмник не сможет ничего уловить.

Инфракрасный (ИК) свет невидим для глаза, поэтому для проверки работоспособности датчика на плате датчика предусмотрен индикатор отражённого света.



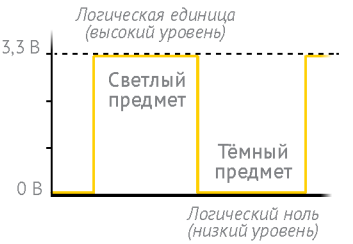


**ВНИМАНИЕ!**

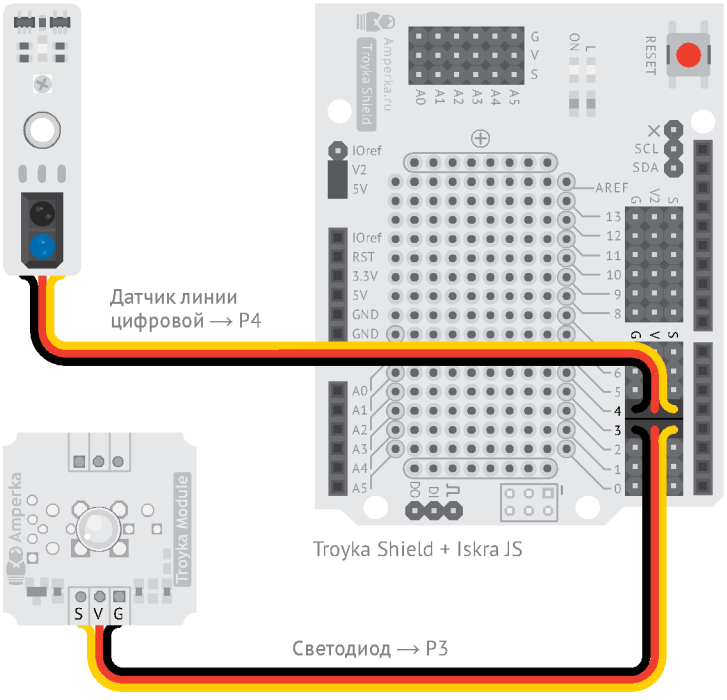
Цифровой и аналоговый датчики линии очень похожи. Отличить их можно по одной микросхеме. В цифровом датчике линии она есть, а в аналоговом датчике линии её нет. Сейчас нам потребуется именно цифровой датчик.

Датчик линии, как и любой другой датчик, *посылает* сигналы в Iskra JS, а та их *считывает*.

Если приёмник «видит» светлый предмет, датчик посылает логическую единицу. Если приёмник «не видит», на сигнальном проводе будет логический ноль.



**10** Цифровой датчик линии необходимо подключить к пину P4, написать новый код и для запуска нажать ↑ в среде IDE.




**a** Создаём переменную с именем `button` и говорим, что это объект — цифровой датчик линии (`'@amperka/digital-line-sensor'`), подключённый (`connect`) к пину P4.

**b** Слово `function` говорит о том, что мы заводим новую функцию.

*Функция* — это блок кода, у которого есть имя. Выражения внутри неё выполняются, когда кто-нибудь эту функцию вызывает. Функции заводят, чтобы использовать один и тот же код снова и снова.

После слова `function` следует желаемое имя. Мы назвали её `myCoolButtonHandler`.

В круглых скобках перечисляются параметры. У нас их нет, поэтому там пусто, но скобки обязательны. И, наконец, в фигурных скобках следует тело функции, т. е. инструкции, которые будут исполняться при вызове.

```
var led = require('@amperka/led').connect(P3);

var button = require('@amperka/digital-line-sensor')
  .connect(P4);

function myCoolButtonHandler() {
  led.toggle();
}

button.on('white', myCoolButtonHandler);
```

**c** Метод `toggle` переключает «включённость» светодиода:
- если он был выключен — включает;
- если включён — выключает.

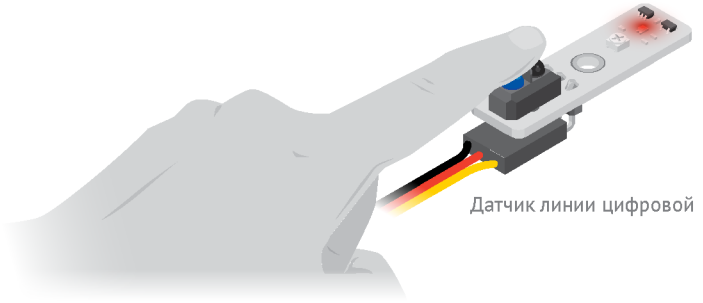
**d** Некоторые объекты генерируют события. В определённые моменты они как бы говорят: «Эй, у меня случилось что-то важное». Мы можем подписаться на событие, чтобы всякий раз, когда оно происходит, вызывать одну из наших функций. У цифровых датчиков линии есть событие `white`. Оно происходит в момент, когда датчик «видит» светлый предмет перед собой.

Подпишемся на нажатие. Для подписки на события у объектов есть метод `on`. Есть он и у объекта-датчика. Первым параметром передаём имя события (`white`), вторым — функцию, которую мы хотим исполнять при наступлении события. Мы используем функцию `myCoolButtonHandler`, которую создали ранее. Все события, которые есть у цифровых датчиков линии, ищи на сайте wiki.amperka.ru/js:digital-line-sensor.

### ТРЮК

Код программы можно не набирать вручную: его можно скопировать с сайта robot.amperka.ru.

Если приёмник улавливает отражённый свет, загорается красный индикатор.



Датчик линии цифровой

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

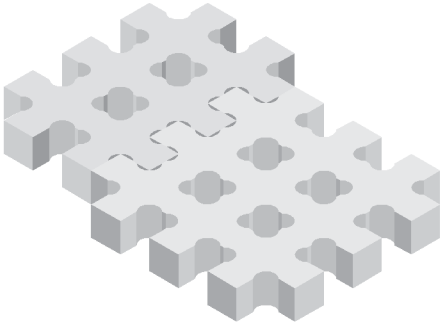
Сделай так, чтобы нажатие на кнопку включало на 1 секунду светодиод, а после он выключался сам. Если не можешь придумать решение, внимательно перечитай описание к предыдущему проекту.

# #СТРУКТОР

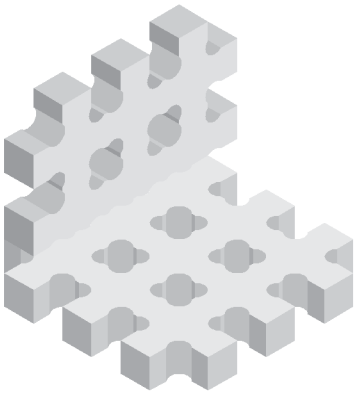
МЫ УЖЕ СДЕЛАЛИ НЕСКОЛЬКО УСТРОЙСТВ. ПОЧЕМУ БЫ НЕ СДЕЛАТЬ ДЛЯ НИХ КОРПУС? КОРПУСА МОЖНО ИЗГОТАВЛИВАТЬ РАЗЛИЧНЫМИ СПОСОБАМИ, НАПРИМЕР СОБИРАТЬ ИЗ КОНСТРУКТОРА.

## КОНСТРУКТОР ДЛЯ КОРПУСА

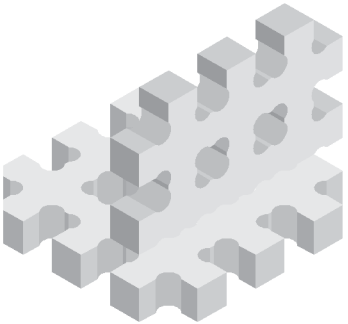
#Структор — это решётчатый конструктор из ПВХ.
#Структор можно склеивать, ломать и резать.



Детали можно соединять
в плоскость.



Можно ставить стенки.

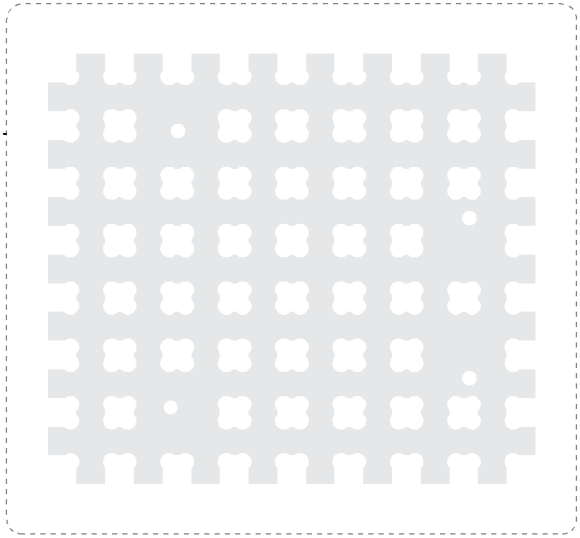


А можно скреплять накрест.

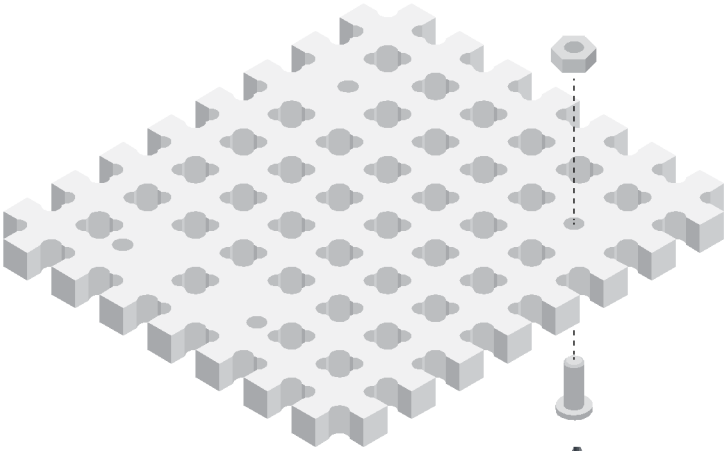
### СТРОЙ ЧТО УГОДНО

Появилась грандиозная идея, но не хватает деталей? Закажи дополнительные на сайте amperka.ru.

11 Воспользуйся канцелярским ножом из набора, чтобы отделить детали #Структора от квадратной плашки. Тебе понадобится самая большая деталь #Структора в наборе.

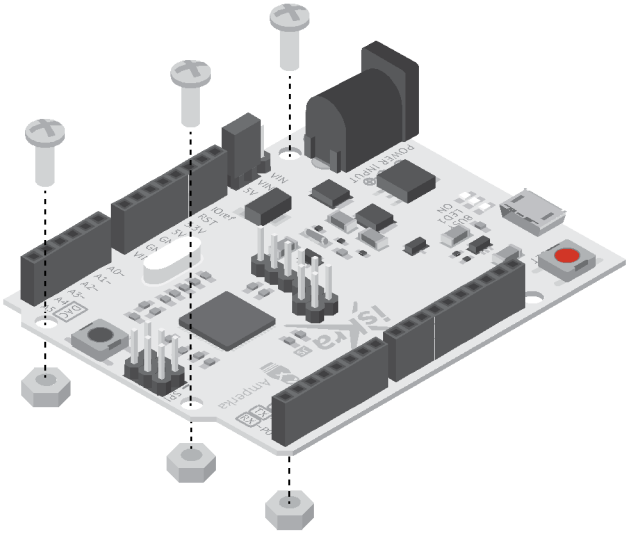


12 С помощью отвёртки закрути винт в #Структор до самой шляпки. Затем накрути гайку на выступающий кончик винта.

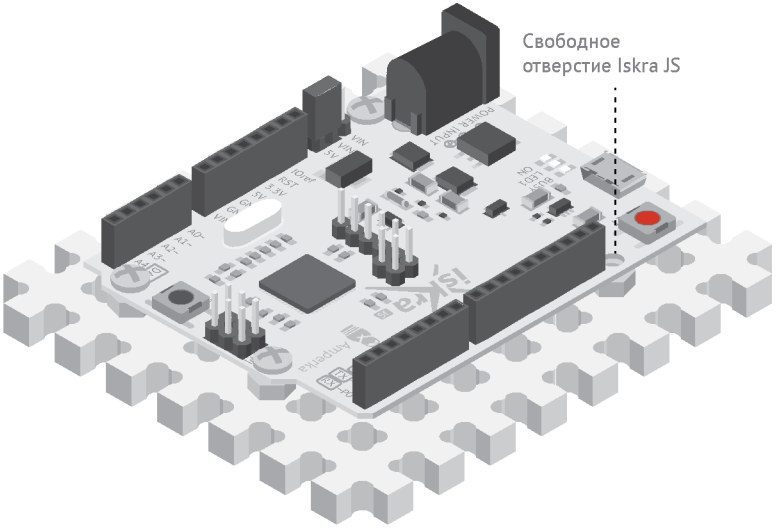


Отвёртка

13 Iskra JS крепится тремя винтами. Одно отверстие остаётся свободным.



14 Кончики винтов вкрути в #Структор.



№ 4

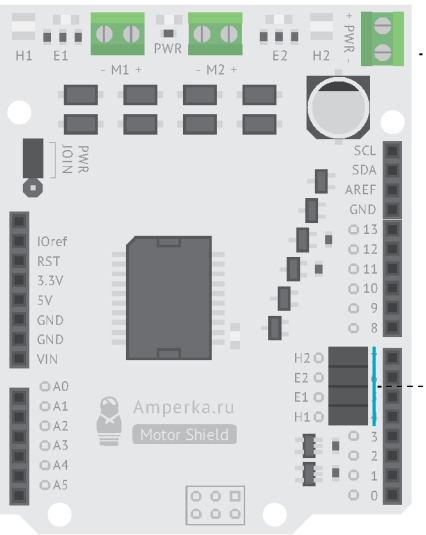
# МИКСЕР

ГЛАВНАЯ ЧАСТЬ ЛЮБОГО РОБОТА — ДВИГАТЕЛЬ. ПОДКЛЮЧИМ ДВИГАТЕЛЬ К УСТРОЙСТВУ И НАУЧИМСЯ ИМ УПРАВЛЯТЬ.

## MOTOR SHIELD

Ток, потребляемый двигателями, слишком большой, чтобы подключать их напрямую к Iskra JS. Для использования двигателей существуют специальные усилители (*драйверы*), позволяющие пропускать через себя необходимые токи.

Motor Shield — именно такой драйвер для *коллекторных двигателей*. Он позволяет управлять скоростью и направлением вращения двигателей с напряжением до 24 вольт.

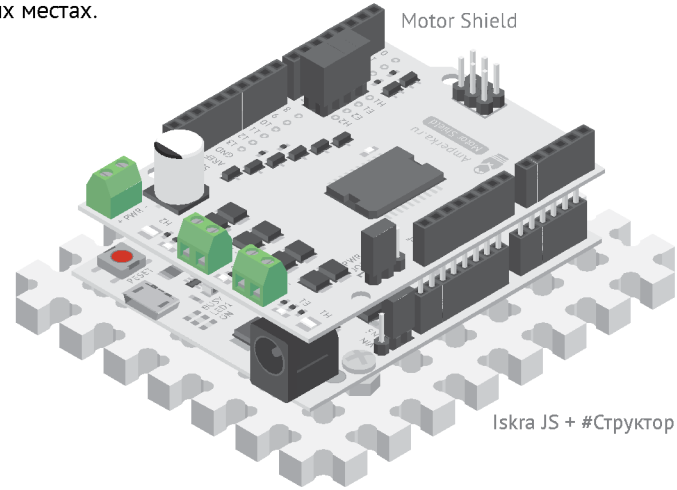



Motor Shield

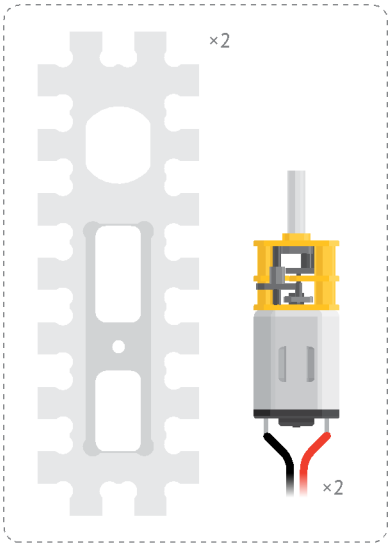
15 Установи джампер в положение PWR JOIN.

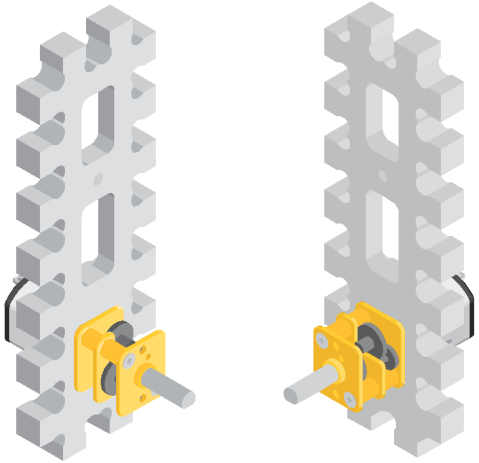
Пины 4, 5, 6 и 7, используемые для *управления* скоростью и направлением вращения моторов, подключены через джамперы. Джамперы позволяют поменять управляющие пины, если в устройстве какие-то из этих пинов уже заняты. Нам менять пины не потребуется, поэтому оставляем джамперы на своих местах.

**16** На Iskra JS можно установить самые разнообразные платы расширения. Сейчас нам потребуется Motor Shield, поэтому сними Troyka Shield с Iskra JS и установи на его место Motor Shield.



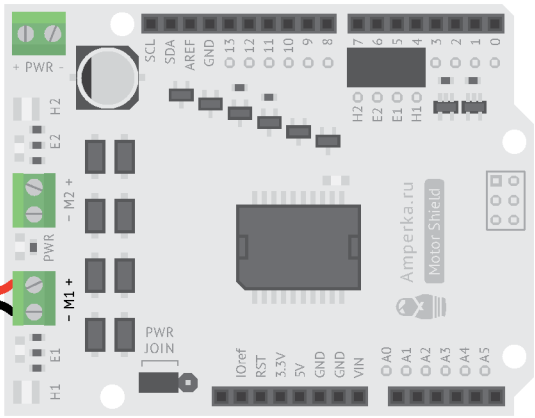

**17** Закрепи оба двигателя в #Структоре.



**18** С помощью отвёртки подключи красный провод к «+», а чёрный провод подключи к «–». Если поменять провода местами, двигатель начнёт крутиться в другую сторону.

Левый двигатель




Motor Shield + Iskra JS

```
1  var motor = require('@amperka/motor');
2  var leftMotor = motor.connect(motor.MotorShield.M1);
3  leftMotor.write(-0.85);
```

**a** Используем переменную `motor`, хранящую подключаемый модуль `'@amperka/motor'`. Функцию `connect()` вызываем не сразу, это позволит не подключать модуль второй раз для второго двигателя. Таким образом, модули могут подключаться всего один раз для нескольких одинаковых объектов.

**b** Говорим, что левый двигатель подключён к Motor Shield, к клеммам M1.

**c** Включаем мотор «назад» на 85 % мощности. Функция `write()` принимает значения от –1 до 1 для указания скорости и направления вращения двигателя. 1 — полный вперёд, –1 — полный назад.

## ПОДУМАЙ САМОСТОЯТЕЛЬНО

- Попробуй задать разные скорости для двигателя.
- Повтори эксперимент, добавив правый двигатель. Подключи его к клеммам M2.

№ 5

# ОДОМЕТР

СКОЛЬКО МЕТРОВ ПРОЕХАЛ РОБОТ?
НА ЭТОТ ВОПРОС ОТВЕТИТ ОДОМЕТР.
ДАВАЙ СОБЕРЁМ ЕГО!

## ЭНКОДЕР

Главный элемент одометра — *энкодер*. Энкодер используют, когда хотят измерить угол поворота колеса или вала. Энкодер фиксирует небольшие углы поворота. Сложив вместе небольшие углы, получим количество полных оборотов колеса.

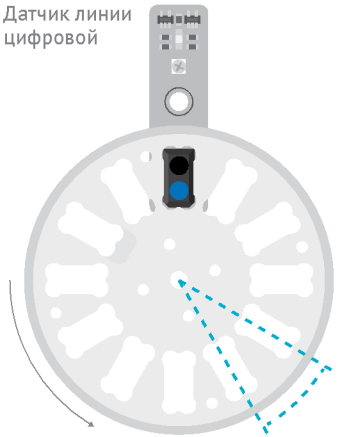
За один полный оборот колеса робот проезжает расстояние, равное *длине окружности* колеса. Длина окружности колеса зависит от радиуса колеса и вычисляется по формуле:

$$S = 2\pi \times R$$

где $\pi$ — математическая *константа*, равная числу ~3,14159.

Умножив длину окружности на количество оборотов колеса, получим общее пройденное роботом расстояние.

Будем использовать в качестве энкодера цифровой датчик линии. В колесе Робоняши есть 12 отверстий для работы цифрового датчика линии. При вращении колеса датчик линии «видит» каждое из двенадцати отверстий и сообщает Iskra JS, что колесо повернулось на одну двенадцатую часть полного оборота.

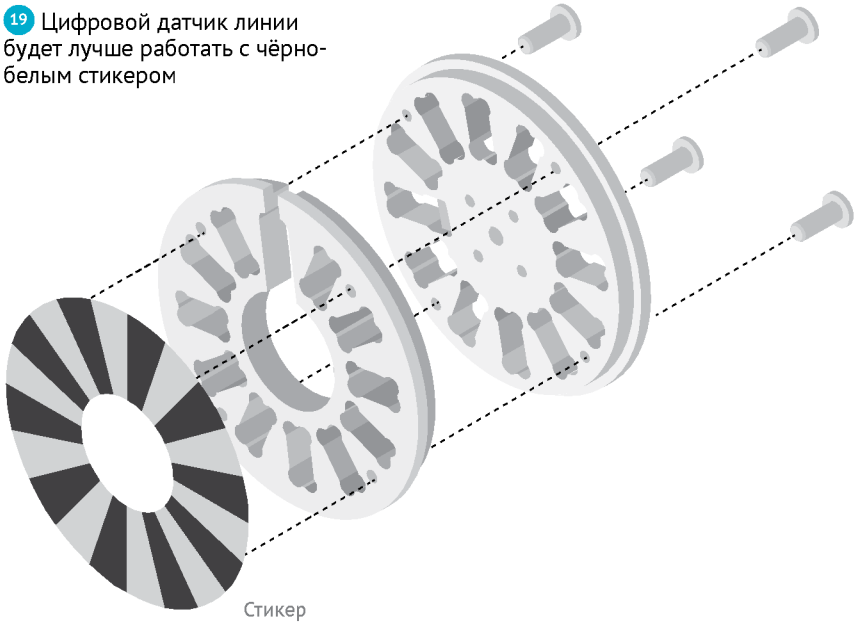



Посмотри эксперимент «Сенсорный выключатель», в нём мы подносили собственный палец, чтобы датчик линии подавал сигнал на Iskra JS. Теперь роль твоего пальца будут выполнять отверстия в колесе.

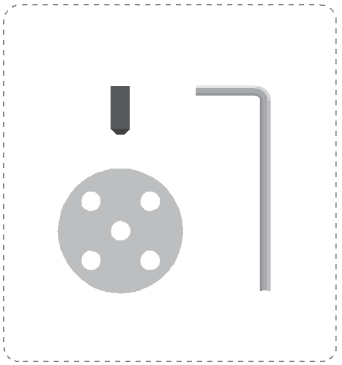
Один оборот колеса Робоняши — это 12 поворотов на небольшой угол, каждый из которых будет зафиксирован цифровым датчиком линии — энкодером.

Колесо делится отверстиями на 12 равных частей.

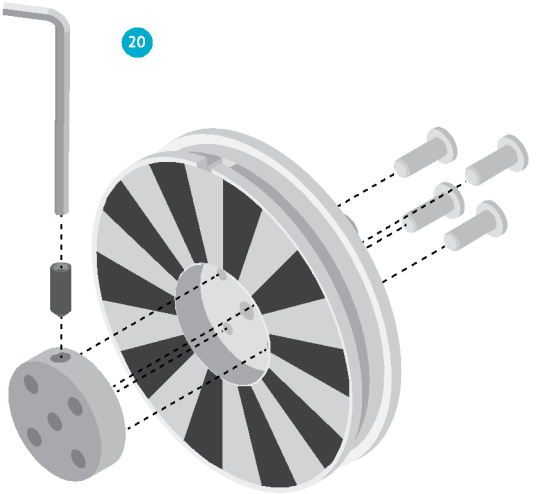
19 Цифровой датчик линии будет лучше работать с чёрно-белым стикером



Стикер

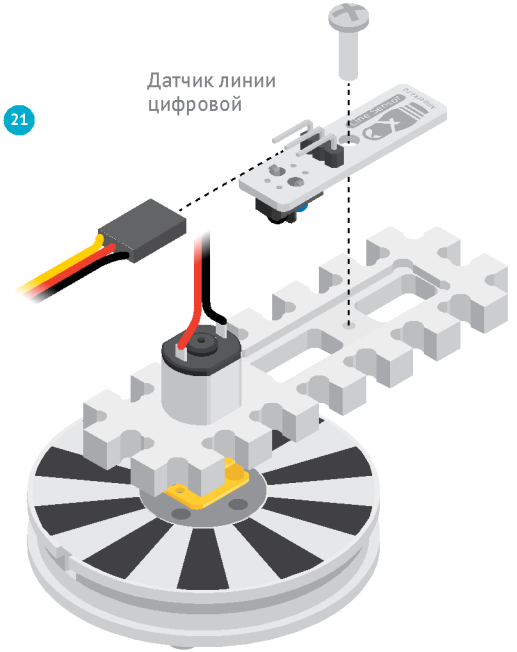


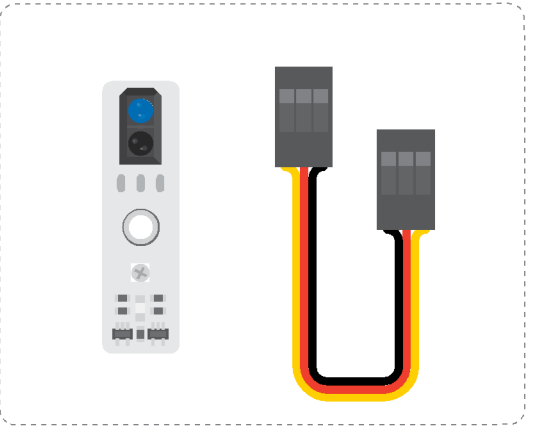
20



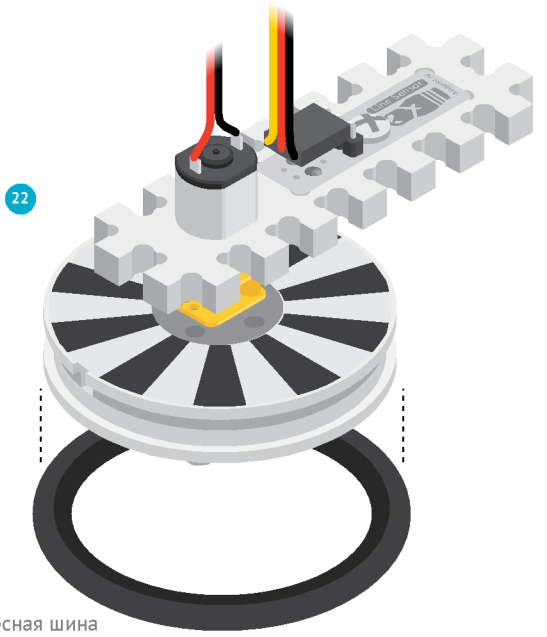
21

Датчик линии
цифровой



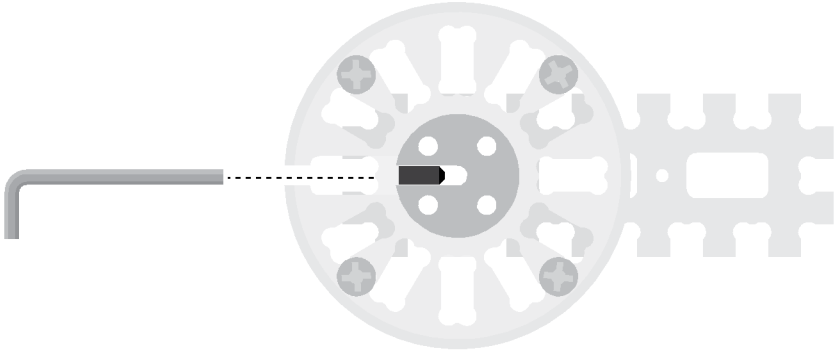


22

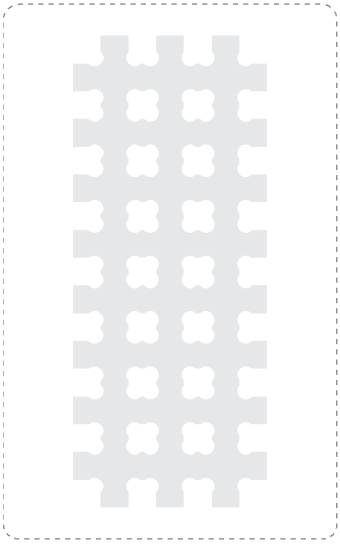


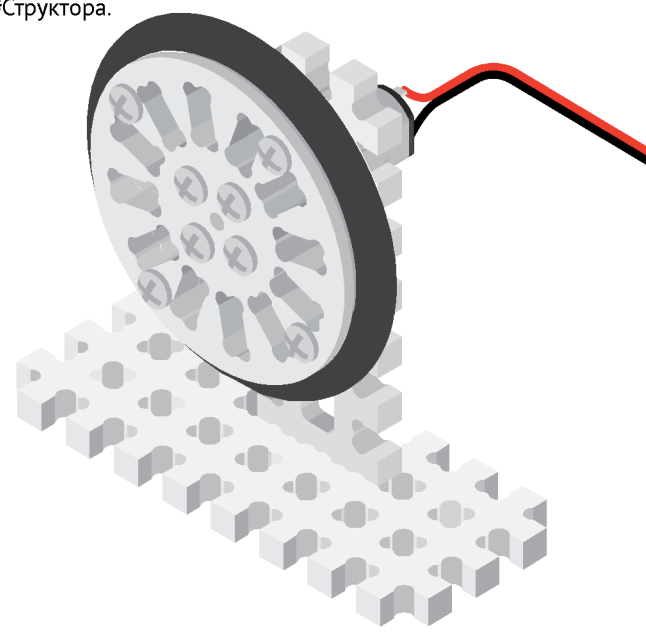
Колёсная шина

23 Винт на металлической втулке выполняет роль штифта. Штифт фиксирует втулку на валу двигателя. Без штифта колесо будет прокручиваться и отваливаться.

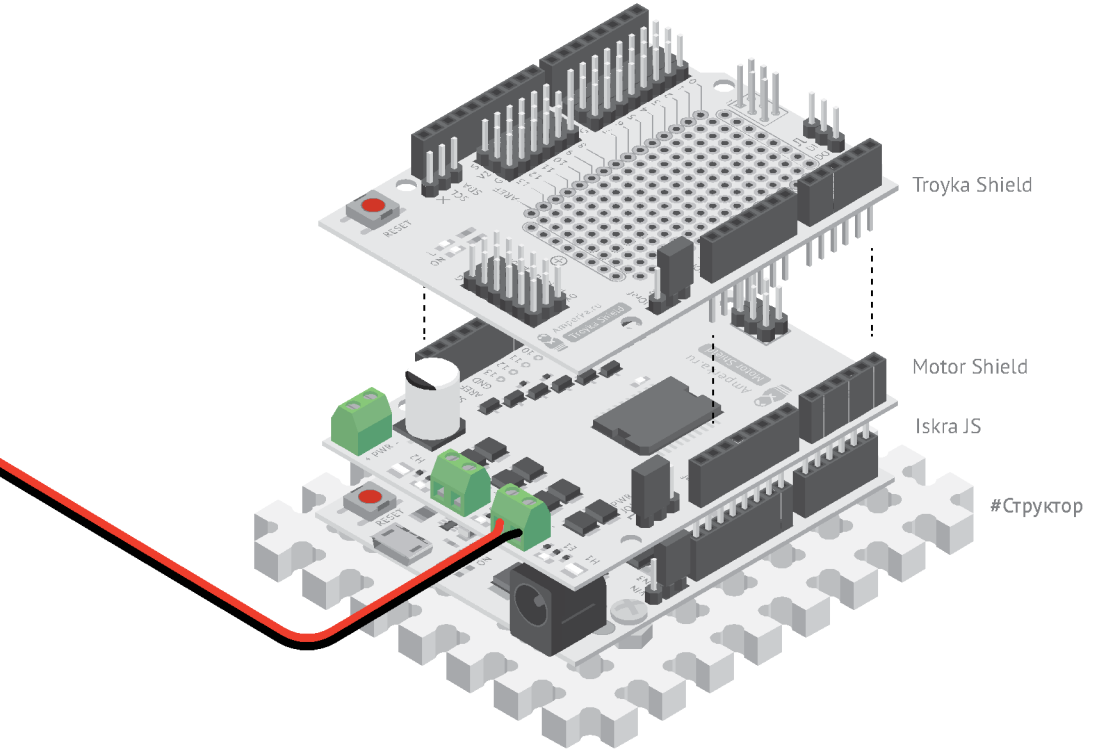


24 Установи колесо на временную подставку из #Структора. Так будет удобнее проводить эксперимент.

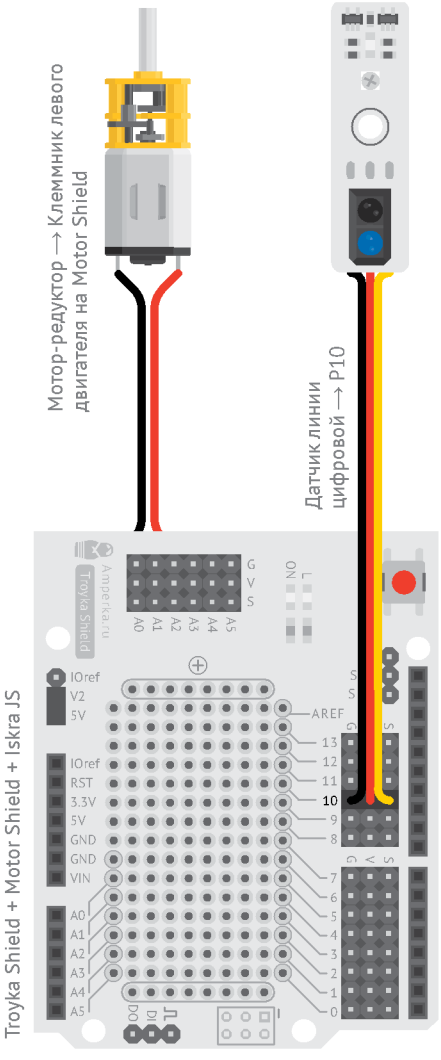




25 Установи Troyka Shield на Motor Shield.



**26** Присоедини датчик линии цифровой.



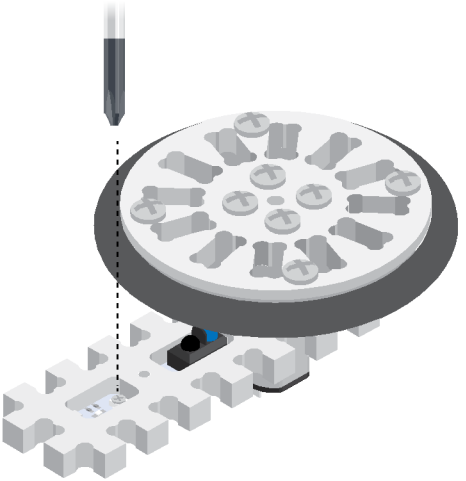

**a** Создадим переменную `RADIUS`, в которой будем хранить радиус колеса. Колесо Робоняши имеет радиус 32 миллиметра. В переменной `WHEEL_LENGTH` будем хранить значение длины окружности колеса. Длина окружности и радиус колеса — величины постоянные. Они не изменятся в процессе выполнения программы, поэтому запишем их заглавными символами. Ими будем обозначать все неизменяемые величины — константы.

Колесо вместе с резиновой шиной имеет длину окружности 201 миллиметр. Можешь проверить: возьми нитку и оберни один раз вокруг колеса. Длина этой нитки будет равна длине окружности колеса.

**b** Создадим переменную `STEP` для хранения значения шага поворота колеса. За один оборот колеса энкодер 12 раз повстречает отверстия, поэтому шаг в 12 раз меньше длины окружности колеса. Шаг, как и длина окружности колеса, величина неизменяемая, поэтому пишем её заглавными символами.

```
var motor = require('@amperka/motor');
var leftMotor = motor.connect(motor.MotorShield.M1);
leftMotor.write(-0.85);

var encoder = require('@amperka/digital-line-sensor')
  .connect(P10);

var RADIUS = 32;
var WHEEL_LENGTH = 2 * Math.PI * RADIUS;
var STEP = 1 / 12;

var revolutions = 0;
encoder.on('black', function() {
  revolutions = revolutions + STEP;
  var distance = revolutions * WHEEL_LENGTH;
  print(distance, 'mm');
});
```

**27** *Очень аккуратно* покрути резистор отвёрткой, чтобы настроить расстояние срабатывания датчика. Добейся, чтобы индикатор загорался напротив белой полоски стикера и гас напротив чёрной. Если при вращении колеса индикатор горит непрерывно, подкрути резистор влево. Если не горит вовсе — подкрути вправо.



c Создадим переменную `revolutions`, в которой будем запоминать количество оборотов колеса. Во время каждого срабатывания энкодера количество оборотов колеса будет увеличиваться на величину шага.

d Энкодер срабатывает каждый раз, когда «видит» отверстие в колесе. Энкодер будет говорить нам, что колесо повернулось ещё на шаг. Найдём общее расстояние `distance`, умножив количество оборотов на длину окружности колеса. А функцией `print()` выведем в консоль расстояние в миллиметрах.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Выведи значение пройденного расстояния в метрах. В одном метре 1000 миллиметров, значит, значение переменной `distance` нужно разделить на 1000.

# № 6 СПИДОМЕТР

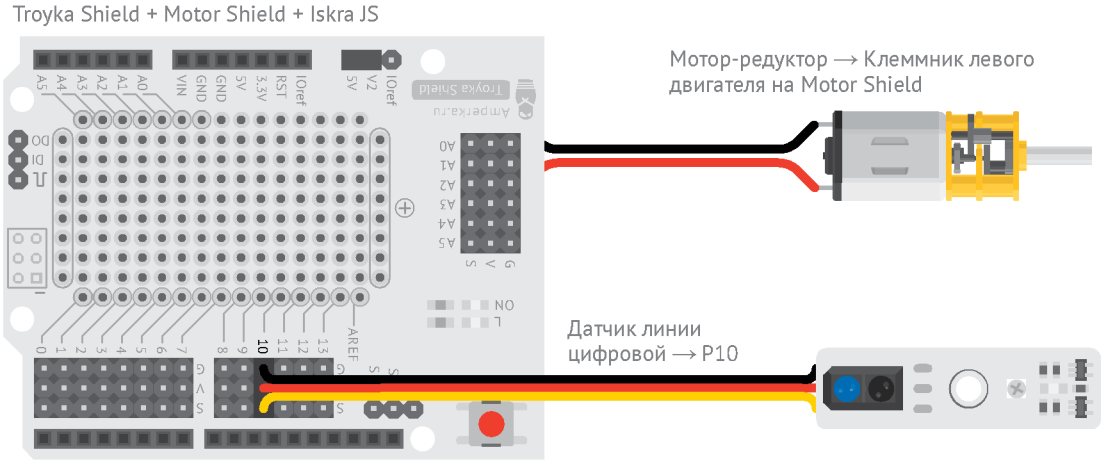
ЧАСТО, ПОМИМО ПРОЙДЕННОГО РАССТОЯНИЯ, ИНТЕРЕСНО ЗНАТЬ, С КАКОЙ СКОРОСТЬЮ ДВИЖЕТСЯ РОБОТ. СОБЕРЁМ СПИДОМЕТР.

## ИЗМЕРЯЕМ СКОРОСТЬ

Скорость — это величина, показывающая, сколько *метров* проезжает робот *за секунду*. Мы уже умеем подсчитывать пройденный путь. Для получения значения скорости V нужно *разделить* кусочек пройденного пути *S* на время *t*, за которое этот кусочек был пройден. Получается известная формула:

**V = S / t**

Её-то мы и будем использовать для наших расчётов.



```
var motor = require('@amperka/motor');
var leftMotor = motor.connect(motor.MotorShield.M1);
leftMotor.write(-0.85);

var encoder = require('@amperka/digital-line-sensor')
  .connect(P10);

var RADIUS = 32;
var WHEEL_LENGTH = 2 * Math.PI * RADIUS;
var STEP = WHEEL_LENGTH / 12;
var speed = 0;

var lastTime = getTime();
encoder.on('black', function() {
  var deltaTime = getTime() - lastTime;
  speed = STEP / deltaTime / 1000;
  lastTime = getTime();

  print(speed.toFixed(2), 'm/s');
});
```

**a** Заведём переменную `lastTime` для хранения времени предыдущего замера скорости.

**b** В переменную `deltaTime` будем записывать разницу во времени между замерами.

**c** Замеряем текущую скорость. Для этого нужно разделить пройденный путь за один шаг оборота колеса на время, за которое робот преодолел этот шаг. Полученный результат делим на 1000, так как хотим получить скорость в метрах в секунду, а не в миллиметрах в секунду.

**d** Метод `toFixed()` отрезает знаки после запятой в дробных числах. Параметр в скобках указывает, сколько знаков оставить после запятой. Указываем 2, то есть оставляем 2 знака после запятой.

**28** Скорость колеса зависит от трения поверхности, по которой едет колесо. Если поверхность не меняется, скорость остаётся примерно одинаковой. Попробуй руками замедлить скорость вращения колеса. Ты увидишь, как выводимое в консоль значение начнёт уменьшаться.

## ВЫВОД ПЕРЕМЕННЫХ В КОНСОЛЬ

Обычно для вывода значений переменных в консоль мы используем функцию `print()`. Обрати внимание, как часто при этом появляются новые данные в консоли. Это не всегда удобно. Есть и другие способы вывести значение переменных. Удали строчку с функцией `print()` и попробуй пару трюков:

• 1-й способ. Допиши в конце кода пару строк, выводящих в консоль значение скорости каждую секунду.

```
setInterval(function() {
  print(speed.toFixed(2), 'm/s');
}, 1000);
```

• 2-й способ. В процессе работы устройства набери в консоли имя переменной, значение которой хочешь узнать. Нажми клавишу Enter на клавиатуре – плата вернёт значение переменной. Можешь использовать целые выражения, например такое:

```
>print(speed.toFixed(2), 'm/s');
```

Нажми клавишу «вверх» на клавиатуре, последнее введённое выражение автоматически подставится в строку консоли.
Так не придётся набирать его заново.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Собери второе колесо. Подключи его к клеммам M2. Второй датчик линии подключи к разъёму P9. Повтори эксперименты «Одометр» и «Спидометр» со вторым колесом.

№ 7

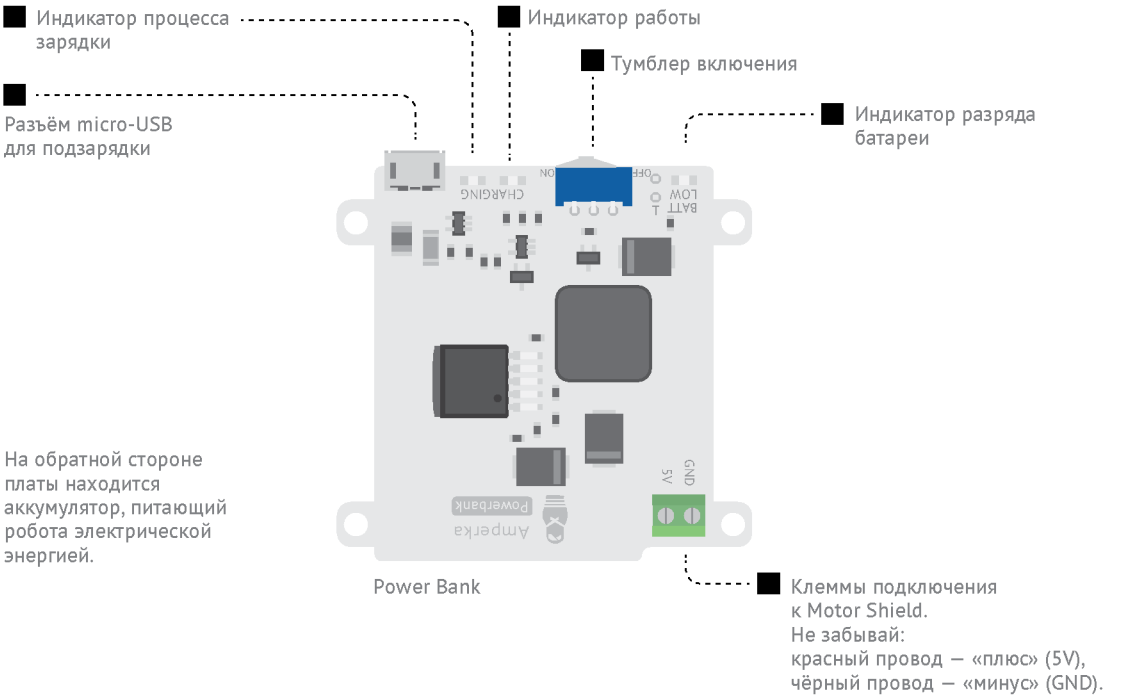
# МАРСОХОД

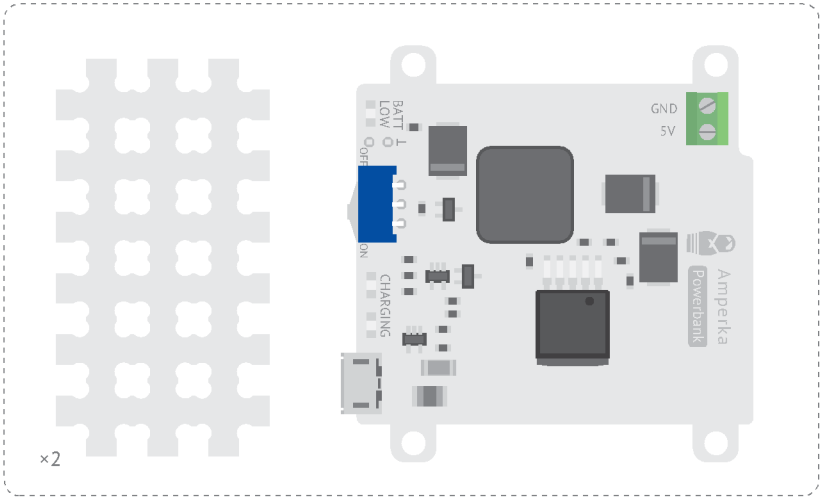
**СОБЕРЁМ, НАКОНЕЦ, РОБОТА!
НИ ОДИН ЕЗДЯЩИЙ РОБОТ НЕ ОБХОДИТСЯ
БЕЗ АВТОНОМНОГО ПИТАНИЯ И ПУЛЬТА
УПРАВЛЕНИЯ.**

## POWER BANK

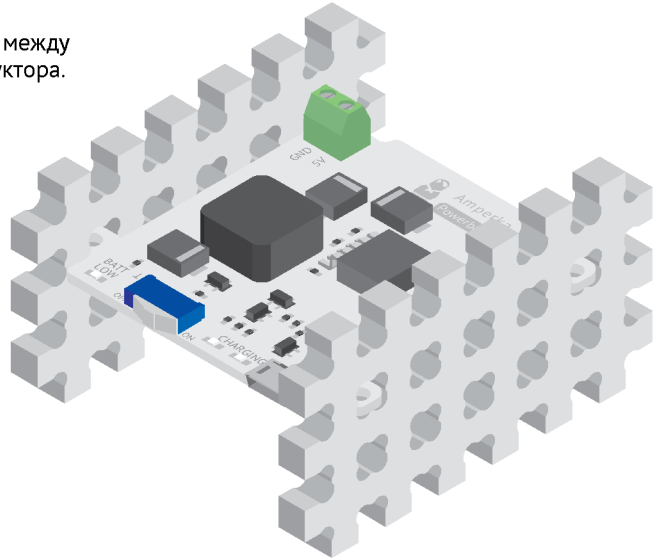
Плата Power Bank служит источником автономного питания Робоняши. С ней тебе не потребуется вести длинные провода от компьютера к роботу.

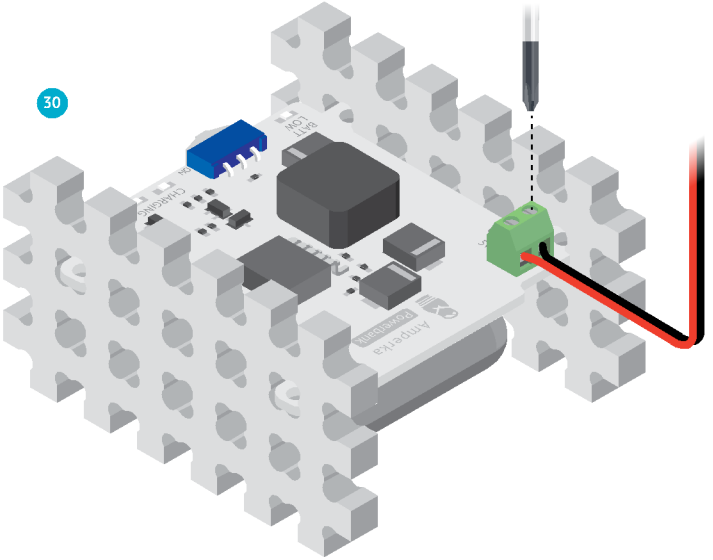
Можешь заряжать Power Bank от компьютера или от адаптера USB, который используешь для зарядки своего смартфона. При этом нет необходимости вынимать Power Bank из переднего отсека Робоняши.

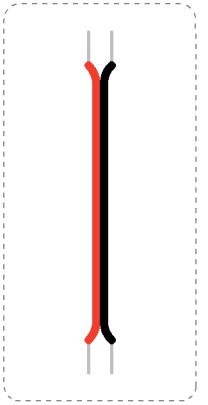


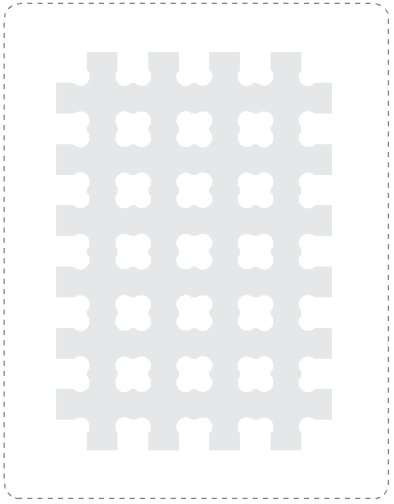


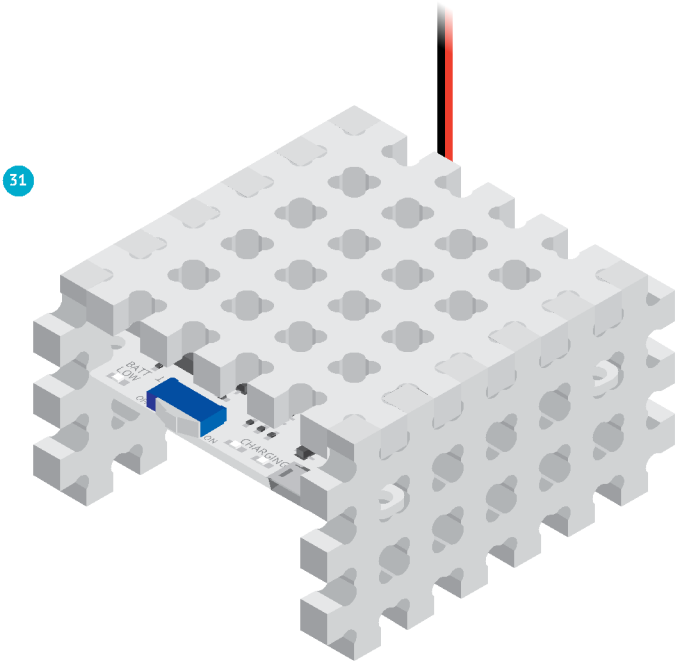

29 Закрепи Power Bank между двумя пластинами #Структора.

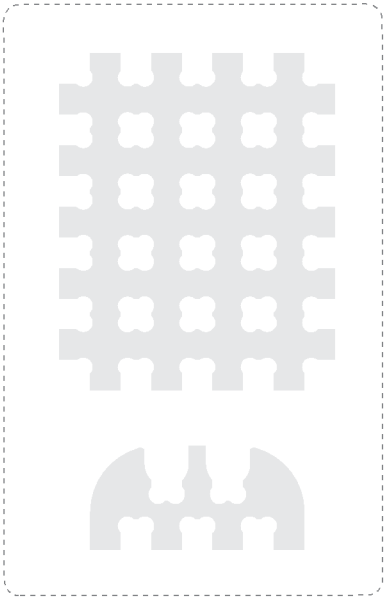



30
BATT LOW
CHARGING
Ampeka
Компания

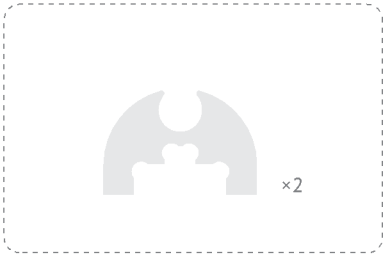





31
BATT LOW
CHARGING

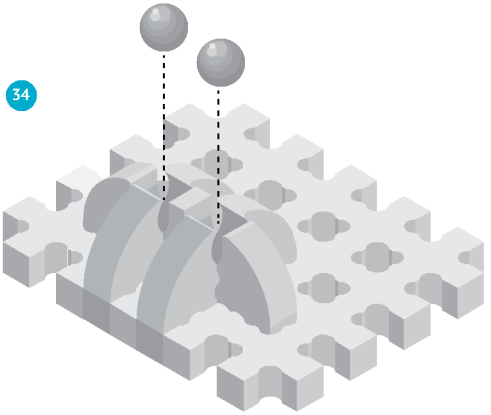


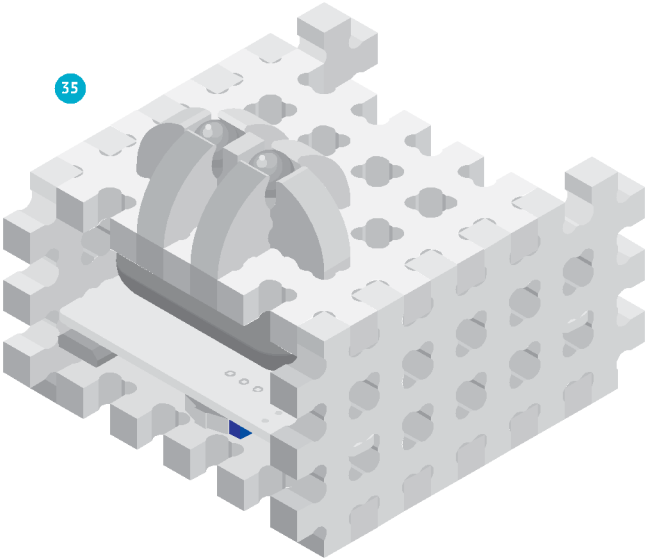
32

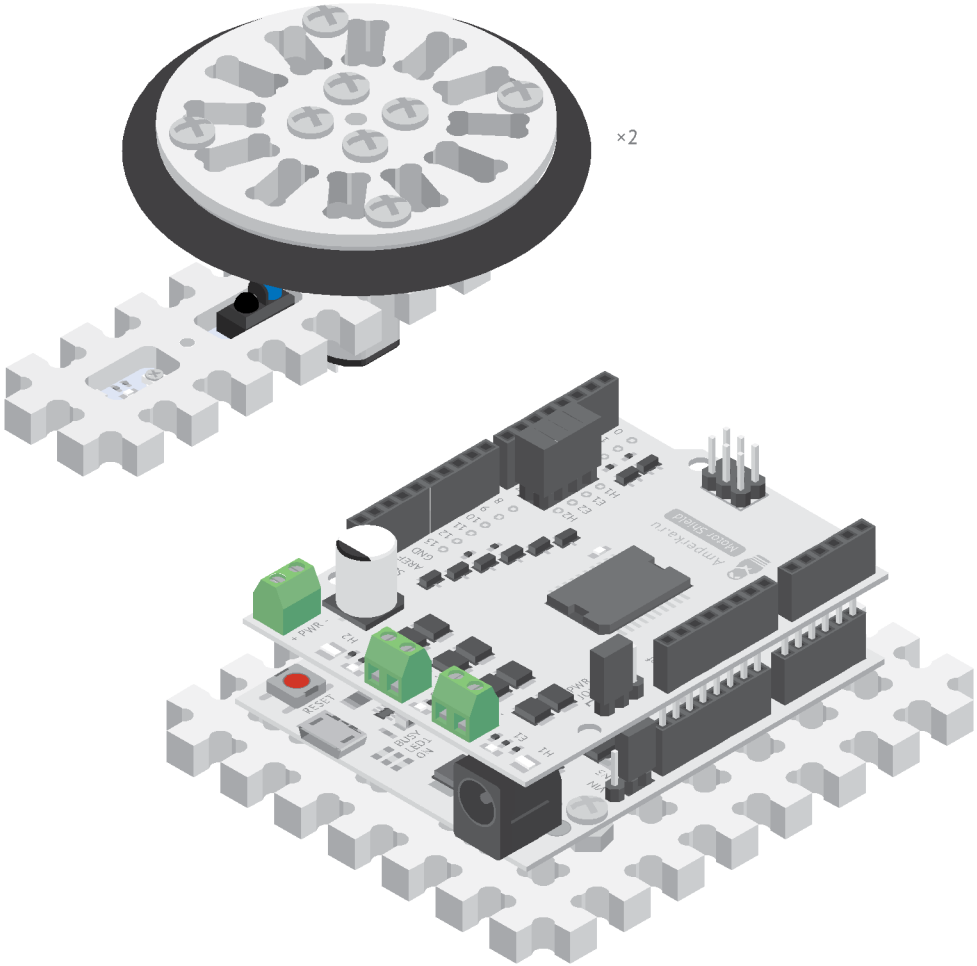



33

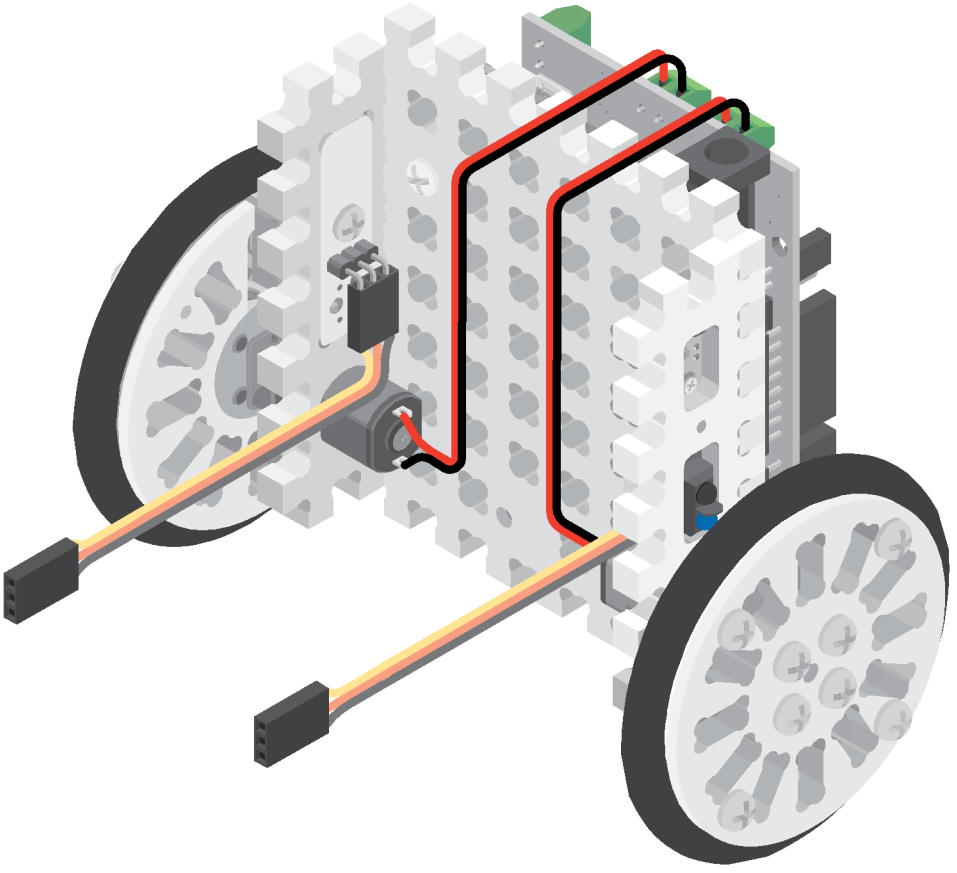
×2

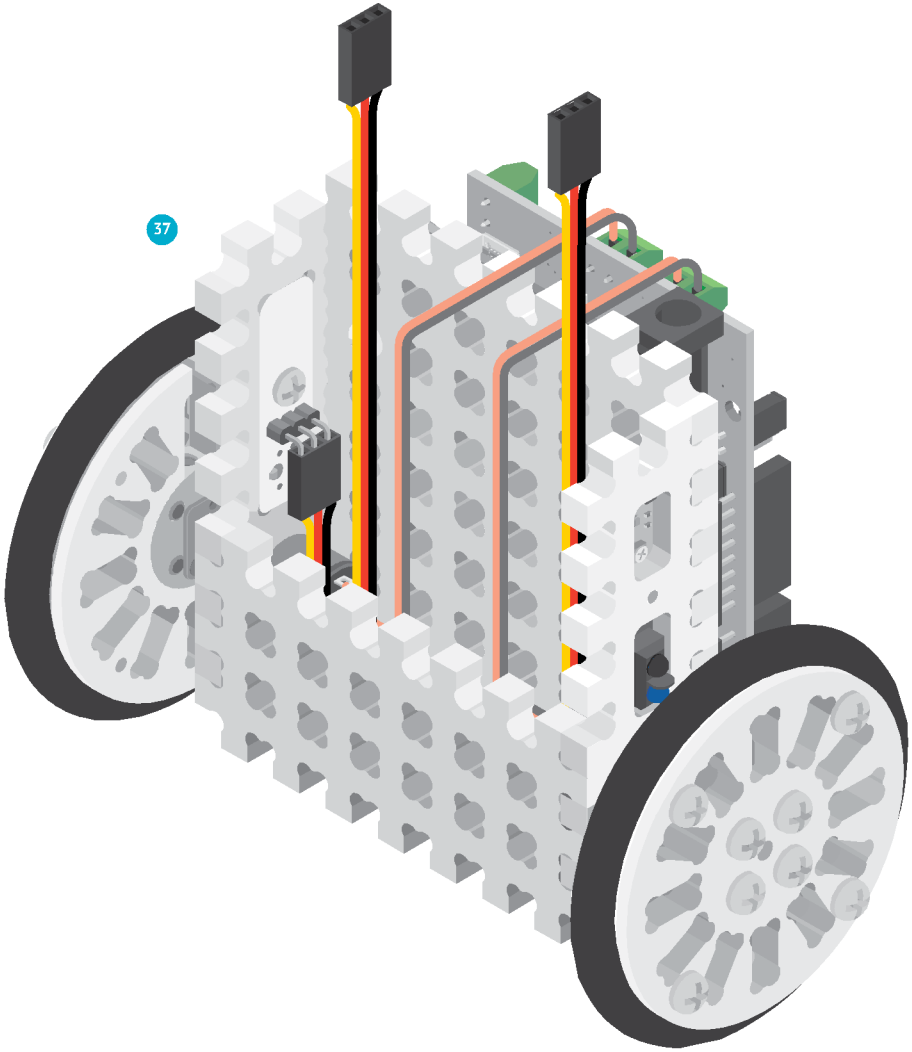


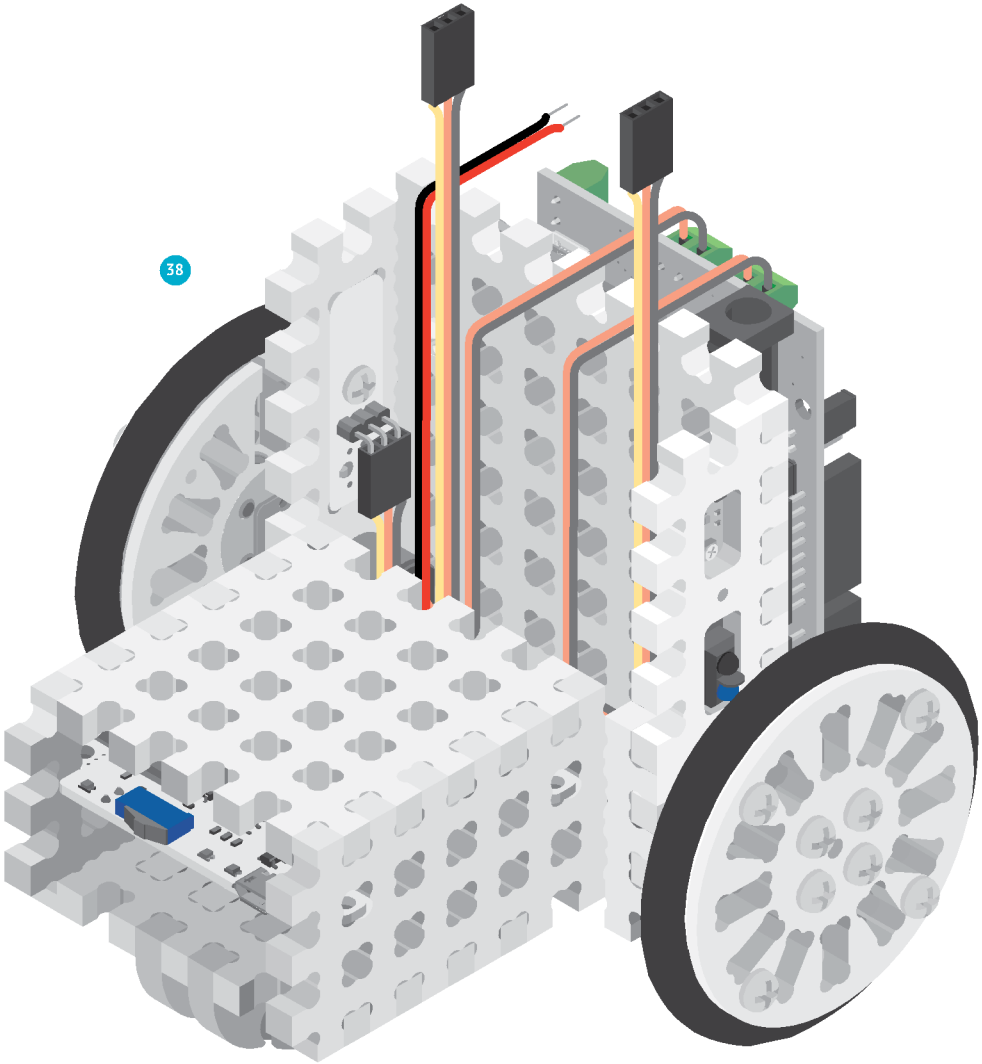




×2
Ampelka.ru
RESET
+ PWR -

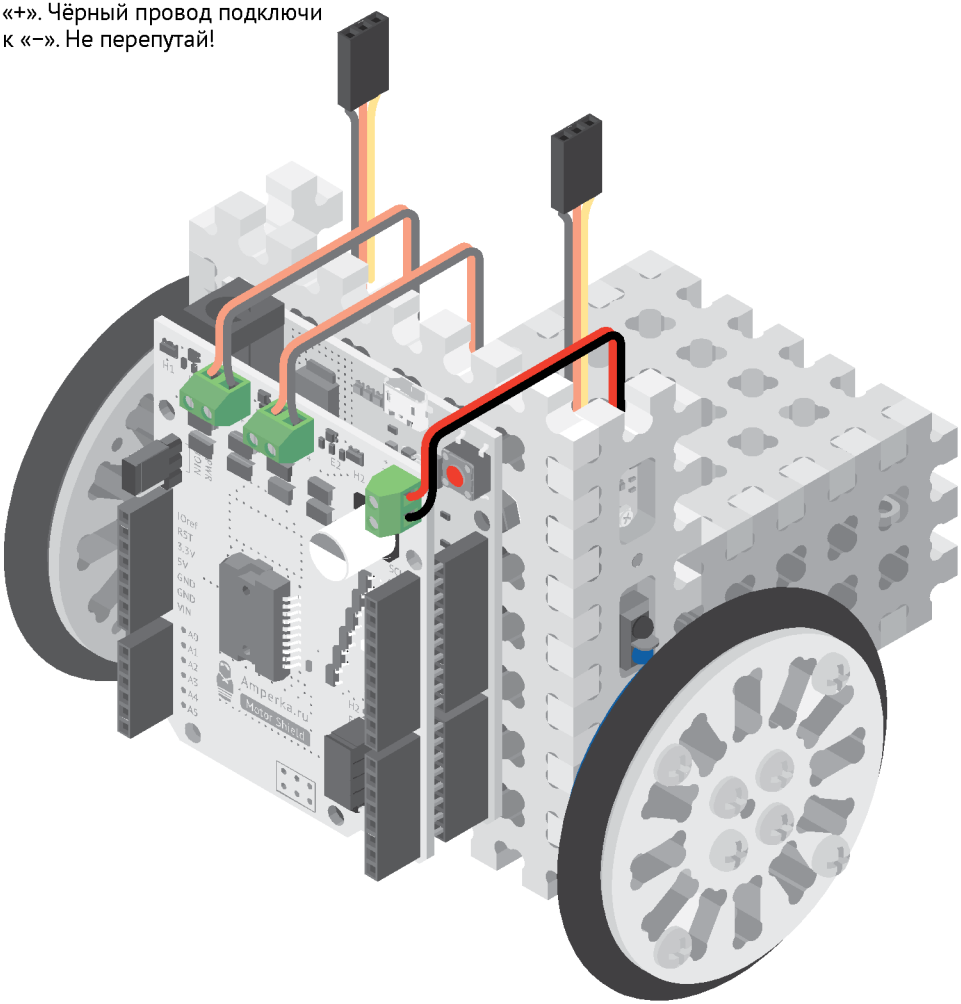
36 Правый двигатель подключи к клеммам M2. Левый двигатель подключи к клеммам M1.



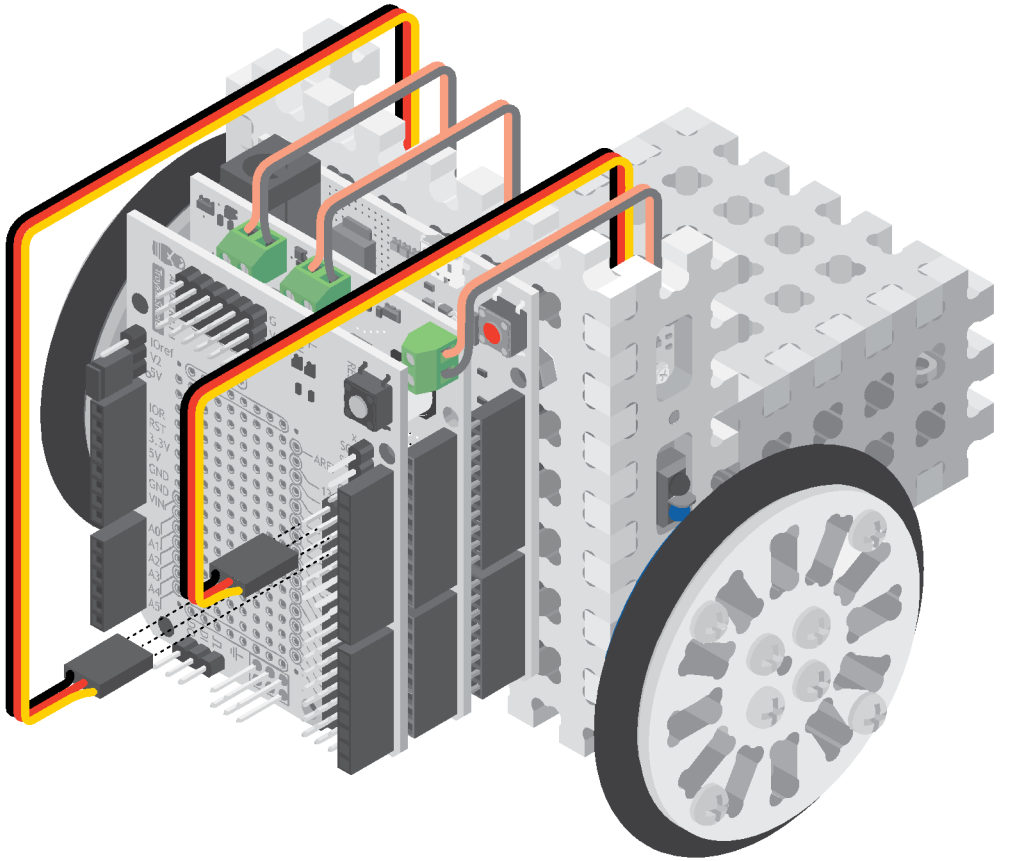

37


38

39 Красный провод питания Power Bank подключи к клемме «+». Чёрный провод подключи к «–». Не перепутай!



40 Установи на Motor Shield плату Troyka Shield. Подключи к Troyka Shield датчики линии: разъёмы подключаются к P10 (правый) и к P9 (левый).

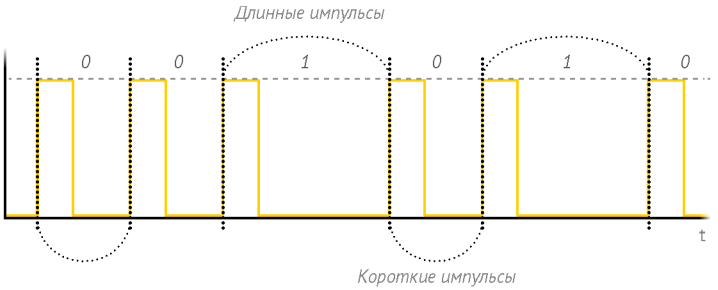


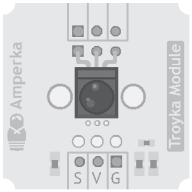
Давай запустим робота и поуправляем им.

Научим его слушаться наших команд. Будем управлять с помощью пульта на инфракрасных лучах. Точь-в-точь как телевизор или кондиционер.

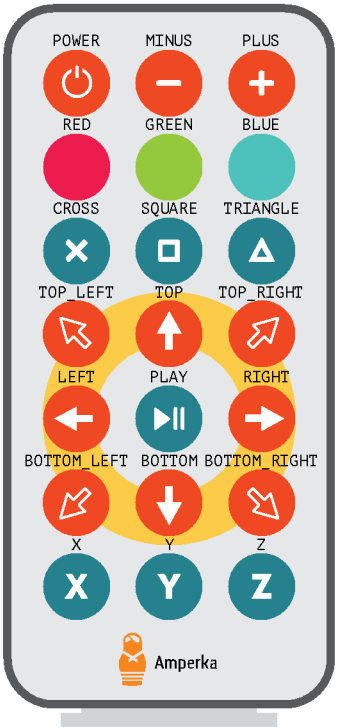
Пульт ИК при нажатии кнопки излучает кодированную посылку, а приёмник принимает её. Кодированный сигнал — это последовательность импульсов разной длительности. Ноль — короткий сигнал, единица — длинный. Приёмник получает кодированный сигнал и передаёт его на Iskra JS.

Каждая кнопка на пульте имеет свой собственный код.



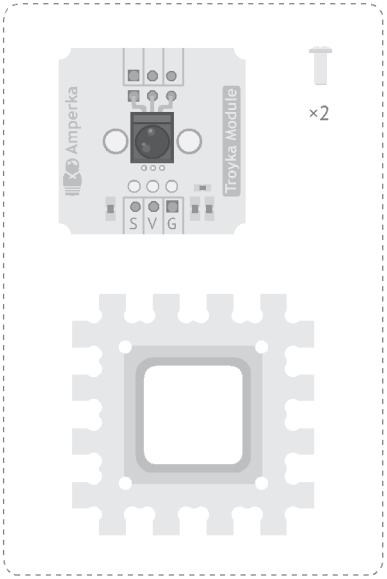



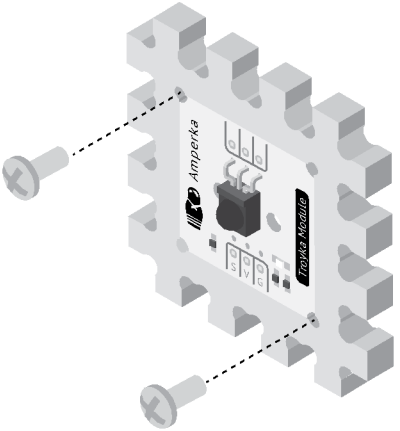




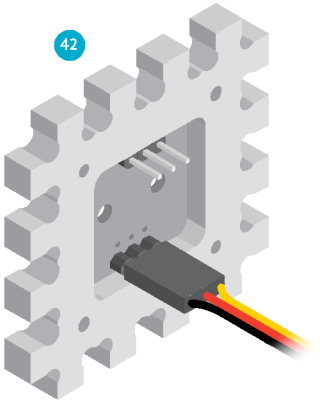

Вытяни язычок, пульт заработает.

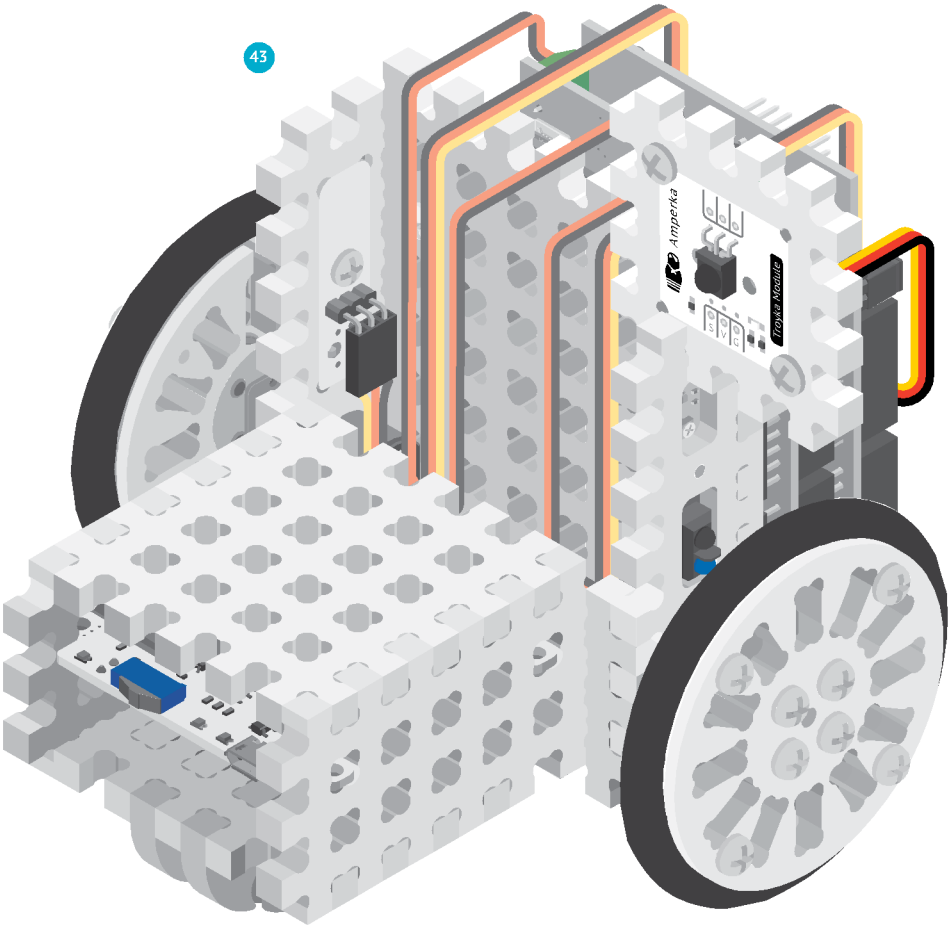
41 Установи ИК-приёмник на пластину #Структора.



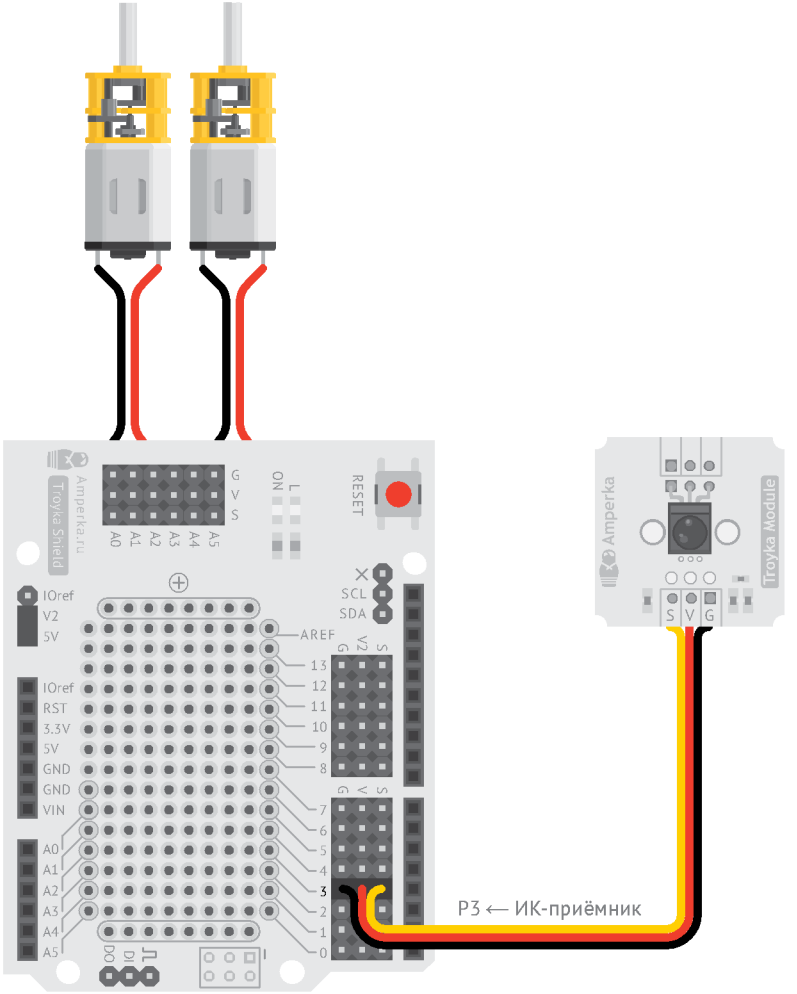




42




43
Amperka
Troyka Module
S
V
G

44 Во всех следующих экспериментах будем подключать мотор-редукторы именно так: левый в клеммы M1, правый в клеммы M2. Хорошо?




Iskra JS + Motor Shield + Troyka Shield

```
var marsohod = require('@amperka/robot-2wd')
  .connect();

var receiver = require('@amperka/ir-receiver')
  .connect(P3);

receiver.on('receive', function(code) {
  if (code === receiver.keys.TOP) {
    marsohod.go({l: 1, r: 1});
  }
  if (code === receiver.keys.POWER) {
    marsohod.stop();
  }
});
```

**a** Для двухколёсного робота существует специальная библиотека `'@amperka/robot-2wd'`. В ней находятся необходимые функции управления двигателями, подключёнными через Motor Shield. Библиотека знает, что левый двигатель подсоединён к клеммам M1, а правый — к M2.

**b** Используем переменную `receiver` для подключаемой библиотеки `'amperka/ir-receiver'`, которая будет обрабатывать команды с инфракрасного пульта.

**d** `if` — это условное выражение. Их используют, чтобы сделать в коде ветвление, когда в зависимости от некоего условия выполняется либо один код, либо другой. Условие записывается в круглых скобках. Интерпретатор проверяет, истинно ли оно или ложно. Если выражение истинно, он выполняет код в фигурных скобках, следующих прямо за условием. Если выражение ложно, код в этих фигурных скобках он пропустит. Для записи условия можно использовать операторы сравнения, логические операторы «и», «или», «не».

## ВАЖНО!

Не забывай заходить на сайт js.amperka.ru, когда видишь новые библиотеки. Там находится их полное описание.

| *Оператор* | *Значение* |
|---|---|
| `a < b` | a меньше b |
| `a <= b` | a меньше либо равно b |
| `a > b` | a больше b |
| `a >= b` | a больше либо равно b |
| `a === b` | a равно b |
| `a !== b` | a не равно b |
| `a < b && a >= c` | a меньше b, и a больше либо равно c |
| `a < b \|\| a >= c` | a меньше b, или a больше либо равно c |
| `a` | a истинно или не ноль |
| `a \|\| b` | a истинно либо не ноль, или b истинно либо не ноль |
| `!a && !b` | a ложно или ноль, и b ложно или ноль |

## 45 Научим марсоход поворачивать.

```
var SPEED = 0.5;

var marsohod = require('@amperka/robot-2wd')
  .connect();

var receiver = require('@amperka/ir-receiver')
  .connect(P3);

receiver.on('receive', function(code) {
  if (code === receiver.keys.TOP) {
    marsohod.go({l: SPEED, r: SPEED});
  }
  if (code === receiver.keys.BOTTOM) {
    marsohod.go({l: -SPEED, r: -SPEED});
  }
  if (code === receiver.keys.LEFT) {
    marsohod.go({l: 0, r: SPEED});
  }
  if (code === receiver.keys.RIGHT) {
    marsohod.go({l: SPEED, r: 0});
  }
});
```

**a** Заведём переменную `SPEED` для хранения скорости и будем её использовать вместо постоянного написания значения `0.5`. Если захотим задать другое значение, достаточно будет изменить число в одной строчке, а не искать его по всему коду.

**b** Увеличим число функций марсохода. Если на ИК-пульте нажата кнопка `TOP`, едем прямо. Если нажата `BOTTOM`, едем назад. `LEFT` — налево, `RIGHT` — направо.

**c** Объект `receiver` может подписываться на событие `'receive'`, возникающее всякий раз, когда ИК-приёмник получает команду от ИК-пульта. Событие `'receive'` передаёт код команды в виде переменной `code`. Зная код команды, можно задать алгоритм движения робота.

**e** Встроенная в модуль функция `go()` принимает в качества аргумента скорость левого и правого колеса. Скорость передаётся в виде объекта, состоящего из двух параметров: `l` и `r`. `l` — скорость левого колеса, `r` — правого. Скорость можно задать в пределах от −1 до 1 для каждого колеса.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Добавь к роботу светодиод: закрепи его в панели #Структора и подсоедини к пину P2. Пускай кнопка `PLAY` регулирует его включение и выключение.

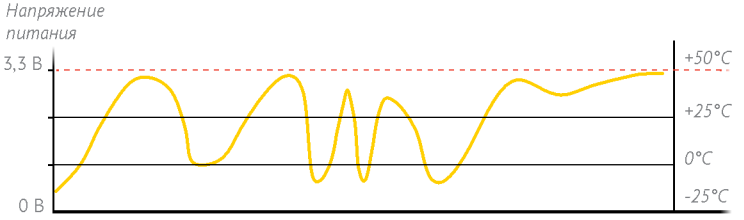
№ 8

# ЧИСТЮЛЯ

ДОБАВИМ НЕМНОГО АВТОНОМНОСТИ. РОБОТЫ-ПЫЛЕСОСЫ, НЕСМОТРЯ НА КАЖУЩУЮСЯ СЛОЖНОСТЬ, УСТРОЕНЫ ДОВОЛЬНО ПРОСТО. У НИХ ЕСТЬ БАМПЕР С ДАТЧИКАМИ, ЧТОБЫ ЧУВСТВОВАТЬ ПРЕПЯТСТВИЯ, А ТАКЖЕ ИНФРАКРАСНЫЕ ДАТЧИКИ СНИЗУ, ЧТОБЫ НЕ ПАДАТЬ С ПОДИУМОВ И ЛЕСТНИЦ. МЫ МОЖЕМ С ЛЁГКОСТЬЮ СДЕЛАТЬ РОБОТА, НЕ ПАДАЮЩЕГО СО СТОЛА.

## АНАЛОГОВЫЕ СИГНАЛЫ

Мы уже умеем работать с цифровыми датчиками линии, но сейчас нам больше подойдут аналоговые датчики линии. Аналоговыми они называются потому, что выходной сигнал у них аналоговый, а не цифровой.




В отличие от цифровых сигналов, аналоговые могут принимать любое значение в диапазоне от 0 вольт до напряжения питания.

Микроконтроллер может считать напряжение на сигнальной линии и с помощью нехитрой арифметики вычислить значение. Например, температуру с термометра в градусах Цельсия.

Аналоговые модули можно подключать к портам A0…A5 или P2…P7.

## ДОСТОИНСТВА И НЕДОСТАТКИ АНАЛОГОВЫХ СИГНАЛОВ

**+** Просто создавать, просто считывать, просто декодировать.

**–** Сложно обеспечить высокую точность и широкий диапазон передаваемых значений.

**–** При передаче по длинным проводам сигнал набирает помехи из радиоволн, показания искажаются.

## ДОСТОИНСТВА И НЕДОСТАТКИ ЦИФРОВЫХ СИГНАЛОВ

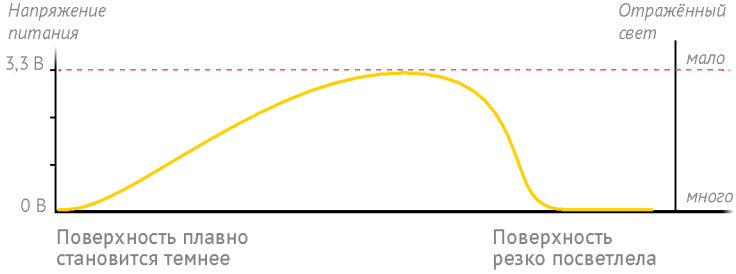
**–** Сложно кодировать и декодировать.

**+** Значения передаются в первозданном виде: с произвольной точностью и в произвольном диапазоне.

**+** Приёмник округляет напряжение до логической единицы и нуля, поэтому помехи в длинных проводах — не проблема.

Аналоговый датчик линии очень похож на своего цифрового брата.

Аналоговый датчик умеет не только определять чёрное и белое, но и различать оттенки серого. Это даёт возможность точно контролировать процесс перехода границы от чёрного к белому и наоборот.




Выходным результатом работы сенсора является аналоговый сигнал. Чем светлее поверхность под сенсором, тем меньше его выходное напряжение.


**ВНИМАНИЕ!**

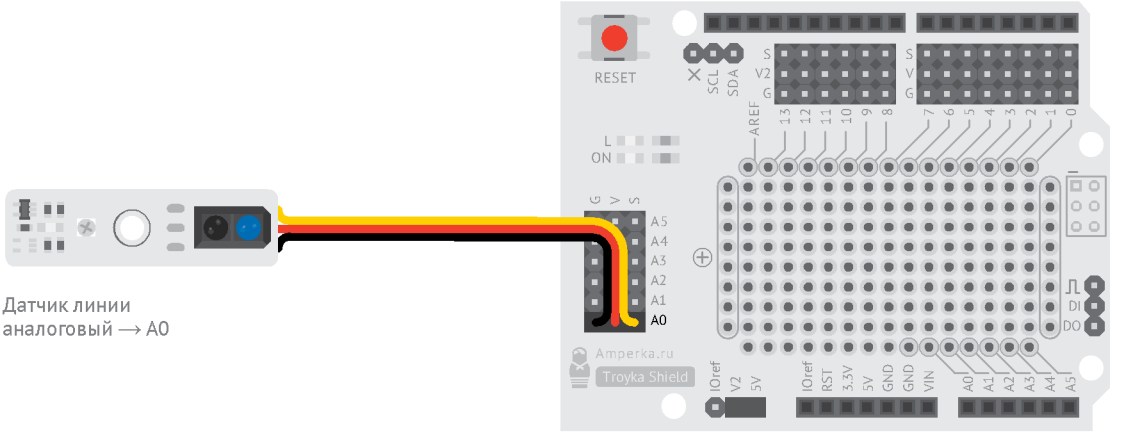
Помнишь, как отличить цифровой датчик линии от аналогового? Отличить их можно по одной микросхеме. В цифровом датчике линии она есть, а в аналоговом датчике линии её нет.

a Библиотека для работы с аналоговым датчиком линии называется `'@amperka/analog-line-sensor'`. Подключим датчик к аналоговому пину A0.

b Создадим функцию `signal` для вывода в консоль значения напряжения датчика в вольтах.

## НЕМНОГО ПОТРЕНИРУЕМСЯ

**46** Чтобы понять, как работать с аналоговыми сигналами, напишем небольшую программу. Будем каждые 100 миллисекунд читать значение напряжения на сигнальном пине датчика линии.




Датчик линии
аналоговый ⟶ A0

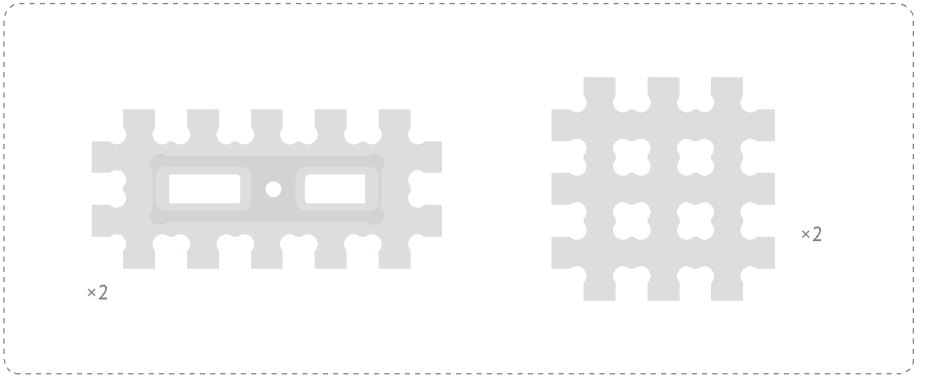
Troyka Shield + Motor Shield + Iskra JS

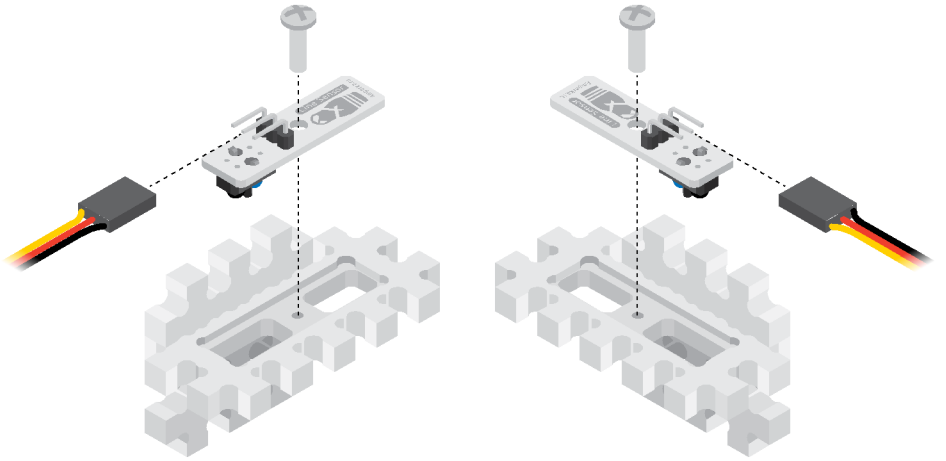
```
var analogSensor = require('@amperka/analog-line-sensor')
    .connect(A0);

var showSignal = function() {
  print('signal:', analogSensor.read('V'), 'volts');
};

setInterval(showSignal, 100);
```

**47** Поводи датчиком возле светлых и тёмных предметов. Обрати внимание, теперь значение напряжения меняется плавно от 0 вольт до напряжения питания — 3,3 вольта. Покрути резистор, чтобы добиться полного диапазона значений.

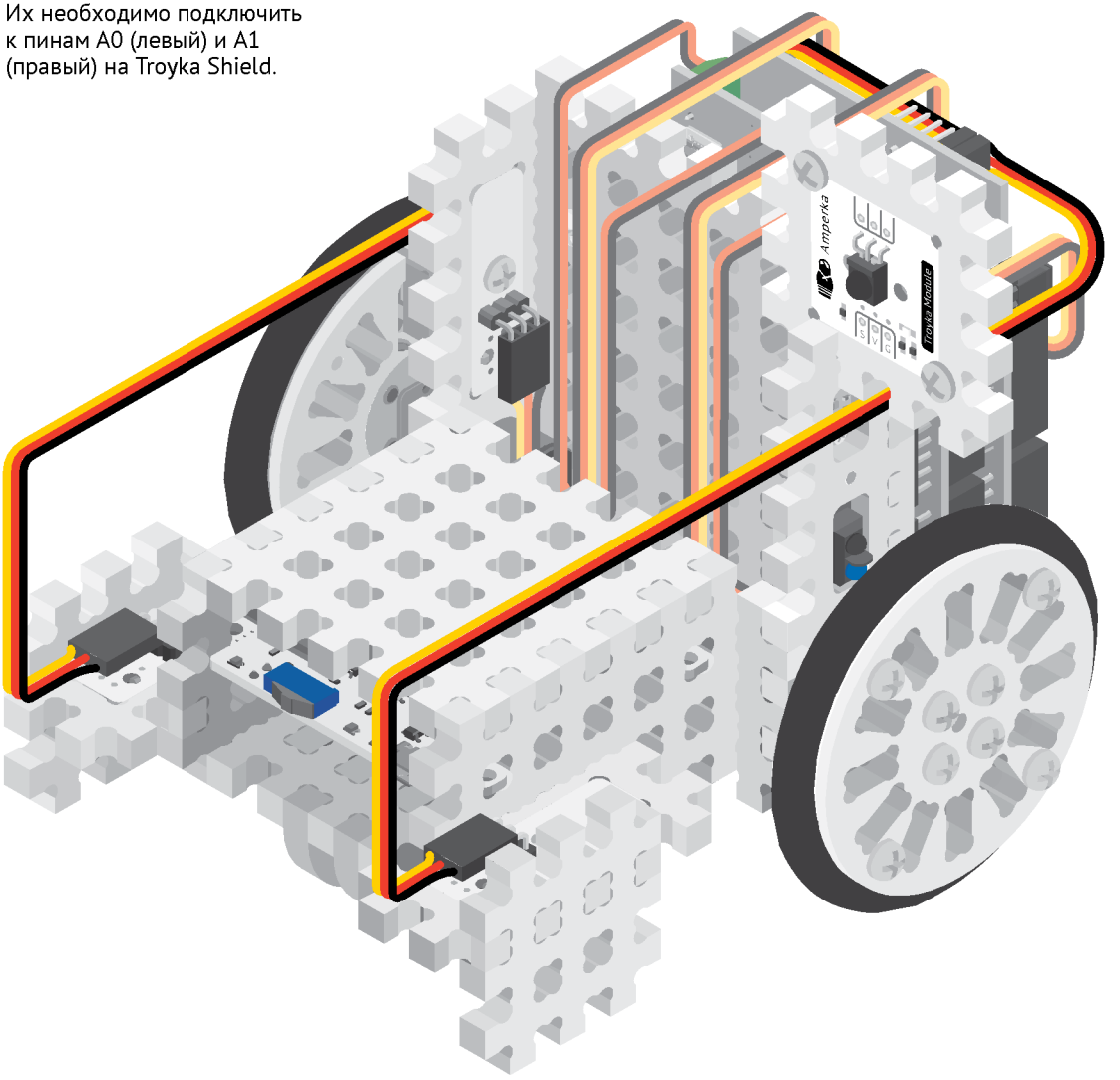
**C** Чтобы что-то происходило периодически и до бесконечности, в JavaScript есть функция `setInterval`. Первым параметром она принимает функцию, которую нужно вызывать периодически, вторым — интервал в миллисекундах. Будем вызывать созданную функцию `signal` с интервалом 100 миллисекунд.

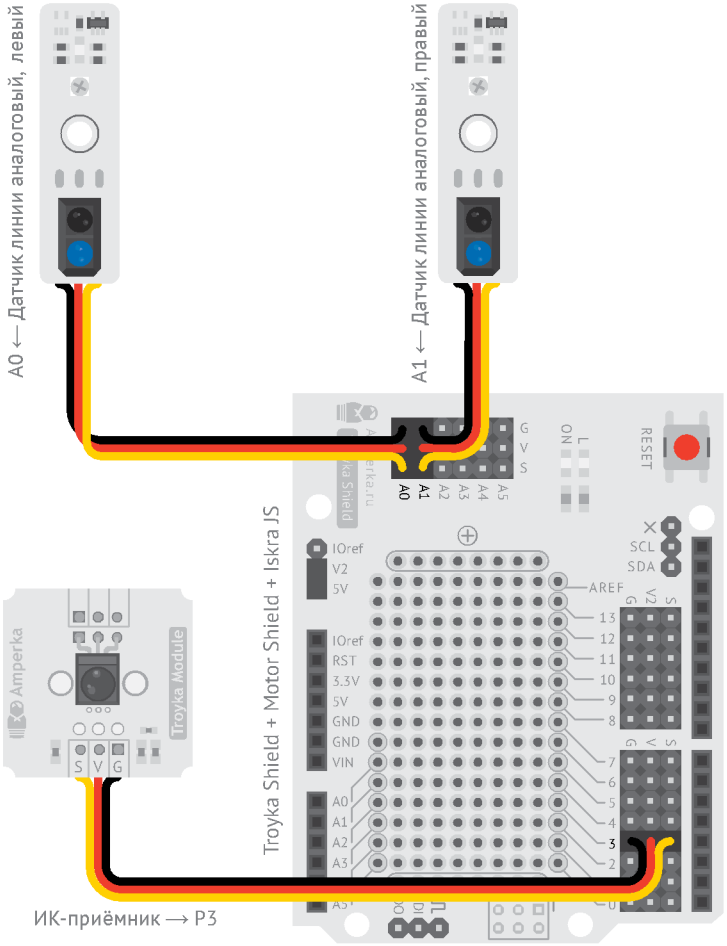
48 Установи пару аналоговых датчиков на Робоняшу.



49 Расположи один датчик слева, а другой справа. Их необходимо подключить к пинам A0 (левый) и A1 (правый) на Troyka Shield.

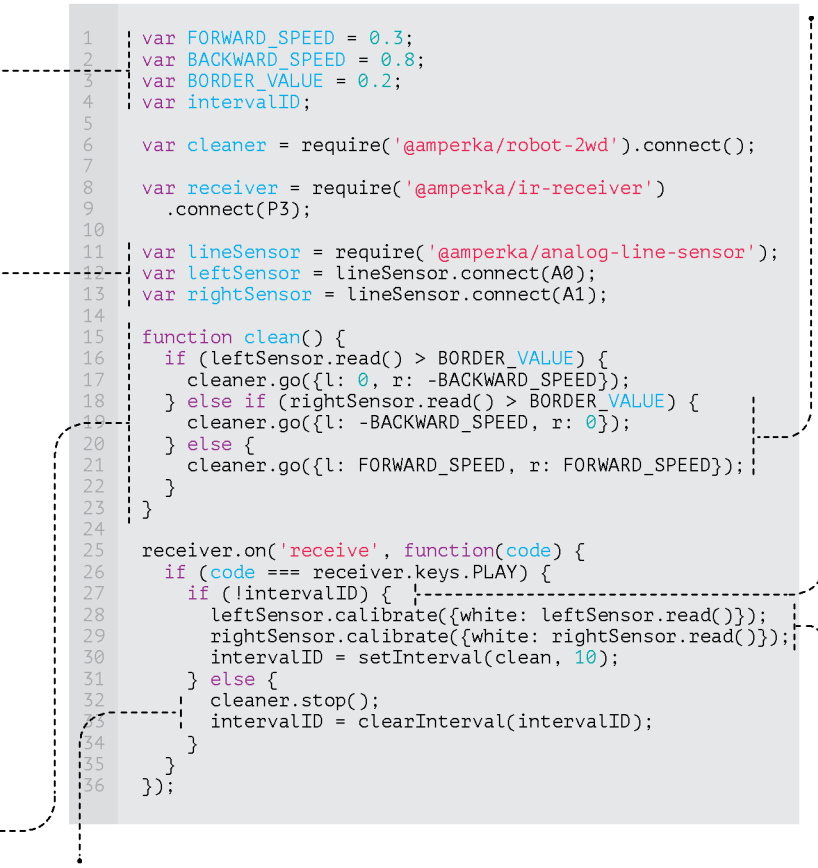





**a** Заведём несколько констант. Константой `FORWARD_SPEED` зададим скорость движения вперёд, а `BACKWARD_SPEED` — назад. Сделаем скорость вперёд меньше, чем назад, чтобы робот успевал среагировать на приближение к краю стола. Константа `BORDER_VALUE` определяет пороговое значение для аналогового датчика линии. Переменная `intervalID` хранит номер интервальной функции `setInterval()`. По этому номеру можно отключить выполнение функции `setInterval()`. Зададим ей начальное значение `null`. Ключевое слово `null` обозначает «отсутствие объекта».

**b** Подключим библиотеку `'@amperka/analog-line-sensor'` для работы с аналоговыми датчиками линии. Левый датчик подключим к аналоговому пину A0, правый — к пину A1.

**c** Создадим функцию `clean()`, ею будем проверять, не приблизился ли робот к краю стола. Если во время уборки робот приблизился одной стороной к краю стола, останавливаем двигатель с этой стороны. Двигатель с другой стороны включаем в обратном направлении.



```
var FORWARD_SPEED = 0.3;
var BACKWARD_SPEED = 0.8;
var BORDER_VALUE = 0.2;
var intervalID;

var cleaner = require('@amperka/robot-2wd').connect();

var receiver = require('@amperka/ir-receiver')
  .connect(P3);

var lineSensor = require('@amperka/analog-line-sensor');
var leftSensor = lineSensor.connect(A0);
var rightSensor = lineSensor.connect(A1);

function clean() {
  if (leftSensor.read() > BORDER_VALUE) {
    cleaner.go({l: 0, r: -BACKWARD_SPEED});
  } else if (rightSensor.read() > BORDER_VALUE) {
    cleaner.go({l: -BACKWARD_SPEED, r: 0});
  } else {
    cleaner.go({l: FORWARD_SPEED, r: FORWARD_SPEED});
  }
}

receiver.on('receive', function(code) {
  if (code === receiver.keys.PLAY) {
    if (!intervalID) {
      leftSensor.calibrate({white: leftSensor.read()});
      rightSensor.calibrate({white: rightSensor.read()});
      intervalID = setInterval(clean, 10);
    } else {
      cleaner.stop();
      intervalID = clearInterval(intervalID);
    }
  }
});
```

**d** `else` — это ветвь, которая выполняется, только если условие в `if` было ложным. Ветку `else`, если она не нужна, можно к `if` не добавлять. В нашем примере она нужна. Обрати внимание, как можно объединять несколько условий с помощью конструкции `else if`.

**e** При каждом нажатии на кнопку PLAY на ИК-пульте проверяем переменную `intervalID`. Если она не равна `null`, будем запускать уборку.

Во время уборки будем вызывать функцию `clean` каждые 10 миллисекунд, чтобы робот проверял «стол под ногами». 100 раз в секунду — более чем достаточно, чтобы не упасть со стола.

**f** Перед каждым запуском уборки задаём датчикам линии уровень белого, которому соответствует поверхность стола. Делает это функция `calibrate()`.

**g** В ветке `else` будем останавливать робота функцией `stop()`. Кроме того, нужно остановить и функцию `setInterval()`. Это делает функция `clearInterval()`, которая принимает параметром идентификатор интервала, который нужно остановить.

**50** Всё готово! Поставь Робоняшу на светлый стол, а затем нажми кнопку PLAY на пульте.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Попробуй изменить значения FORWARD и BACKWARD, посмотри, как меняется поведение Робоняши возле края стола.

№ 9

# СЛЕДОПЫТ

ОЧЕНЬ ЧАСТО ОТ РОБОТОВ ТРЕБУЕТСЯ ДОЕХАТЬ ИЗ ОДНОГО МЕСТА В ДРУГОЕ, ЧТОБЫ ДОСТАВИТЬ ГРУЗ. ДЛЯ ТАКОЙ ЗАДАЧИ РОБОТУ ТРЕБУЕТСЯ ЗНАТЬ МАРШРУТ ДВИЖЕНИЯ.

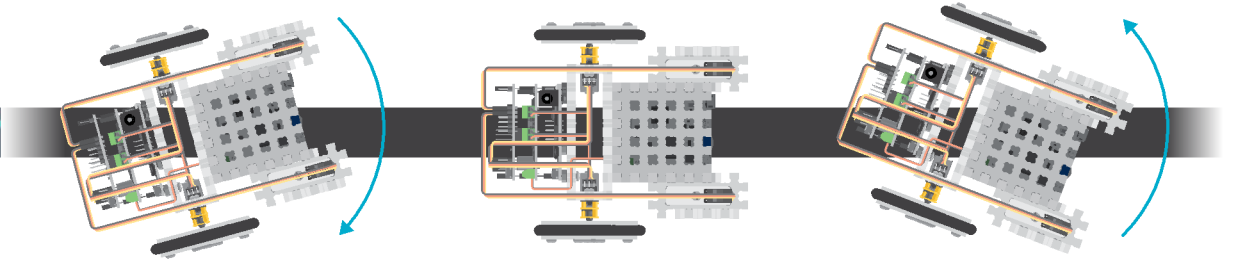
## ДАТЧИК ЛИНИИ

«Показать» маршрут можно довольно просто: нарисовать хорошо заметную на всём пути линию и научить робота ей следовать. Часто устраивают целые соревнования, чей робот лучше проедет по чёрной линии. Давай научим и нашего робота, а заодно подготовимся к участию в соревнованиях.

Будем отслеживать траекторию двумя аналоговыми датчиками линии. Она всегда должна находиться между ними. Если робот отклонится от траектории, один из датчиков увидит линию. Чтобы вернуться на неё, нужно немного изменить скорость колёс.

*Ускоряем левое колесо*

*Замедляем правое колесо*



*Замедляем левое колесо*

*Ускоряем правое колесо*

Если отклоняемся от линии, нужно вернуться на неё.

Всё отлично! Продолжаем ехать вдоль линии.

Отклонились в другую сторону. Нужно немного повернуть, чтобы снова выровняться.

Эту задачу может решить специальный алгоритм — ПИД-регулятор.

## ПИД-РЕГУЛЯТОР

Это алгоритм управления. Существуют целые устройства, которые только и занимаются тем, что управляют какой-нибудь физической величиной. Например, ПИД-регулятор в холодильнике поддерживает температуру внутри на заданном уровне.

Мы будем использовать ПИД-регулятор для управления движением робота. Алгоритм будет постоянно измерять значения напряжения двух аналоговых датчиков линии и вычитать их друг из друга.

Если робот движется по линии, значения датчиков будут одинаковыми и их разность будет равна нулю. Если один из датчиков увидит чёрную линию, разность изменится и станет больше или меньше нуля.

Значение разности, отличающееся от нуля, назовём *ошибкой*. Чтобы убрать ошибку, роботу нужно немного повернуть, изменив скорость двигателей. ПИД-регулятор — это формула, по которой вычисляется, *на сколько* необходимо изменить скорость двигателей.

Формула регулятора состоит из трёх слагаемых:

**output = P + I + D**

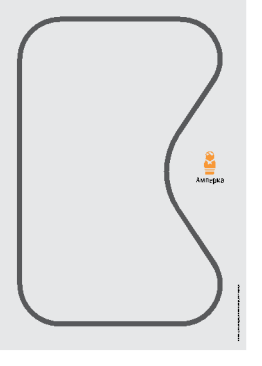
P – *пропорциональное* слагаемое. Оно следит за величиной ошибки и пропорционально изменяет скорость двигателей, чтобы убрать её.
I — *интегральное* слагаемое. Во время движения робота небольшая ошибка может накапливаться. Это слагаемое вносит свой вклад в исправление общей накопленной ошибки.
D — *дифференциальное* слагаемое. Оно позволяет увеличить скорость реакции робота на ошибку. Слишком большое значение этого слагаемого заставит робота чрезмерно быстро реагировать на линию.

Каждое слагаемое может давать разный вклад в общую сумму: одно больше, другое меньше. Вклад каждого слагаемого в общую сумму регулируется коэффициентами. Общая формула с учётом коэффициентов выглядит так:

**output = $k_p$ × P + $k_i$ × I + $k_d$ × D**

Настройка ПИД-регулятора заключается в подборе этих коэффициентов. Если подобрать правильно, робот будет плавно ездить вдоль линии. Если подобрать неправильно, робот начнёт болтаться из стороны в сторону и «слетит» с дистанции.

**51** В качестве пути используй карту из набора (или нарисуй свою).



**a** Подключаем библиотеку ПИД-регулятора `'@amperka/pid'`. Библиотека не работает с модулями напрямую, поэтому она не содержит функцию `connect()`. Вместо неё используем функцию `create()`, которая принимает параметром объект с настройками ПИД-регулятора.

**b** `target` — требуемое значение контролируемой величины, `kp` — пропорциональный коэффициент, `ki` — интегральный коэффициент, `kd` — дифференциальный коэффициент (коэффициенты подобраны опытным путём). `outputMin`, `outputMax` — минимальное и максимальное значения скорости двигателей. Запускаем функцию `run()`.

**c** Функция повторяет переданный в неё код каждые 0,02 секунды.

**d** Вычисляем ошибку: разность между значениями левого и правого датчиков линии.

```
var SPEED = 0.3;

var detective = require('@amperka/robot-2wd')
  .connect();

var lineSensor = require('@amperka/analog-line-sensor');
var leftSensor = lineSensor.connect(A0);
var rightSensor = lineSensor.connect(A1);

var lineFollower = require('@amperka/pid').create({
  target: 0,
  kp: 6,
  ki: 0.1,
  kd: -1,
  outputMin: -SPEED,
  outputMax: SPEED
});

lineFollower.run(function() {
  var right = rightSensor.read();
  var left = leftSensor.read();
  var error = left - right;
  var output = lineFollower.update(error);
  detective.go({
    l: SPEED + output,
    r: SPEED - output
  });
}, 0.02);
```

**e** Обновляем значения ПИД-регулятора. С помощью этих значений алгоритм определяет, как скорректировать скорость левого и правого колеса. Записываем в переменную `output` вычисленное значение.

**f** Задаём новую скорость для колёс. Скорректированное значение прибавляем к скорости левого колеса и вычитаем из скорости правого.

## ПОДУМАЙ САМОСТОЯТЕЛЬНО

Задай коэффициентам ki и kd нулевое значение. Посмотри, как себя будет вести Робоняша на линии, если установить коэффициент kp равным 1. Попробуй установить значения kp равными 2, 5, 10 и 20. Сравни результаты.

№ 10

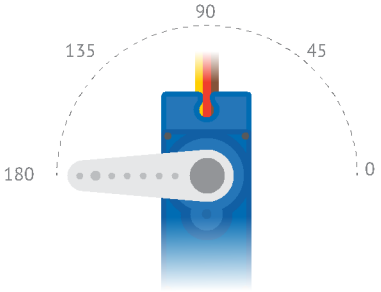
# НЕХОЧУХА

СКОРО МЫ СОБЕРЁМ РОБОНЯШУ ПОЛНОСТЬЮ. ПРЕЖДЕ ЧЕМ СОБРАТЬ ГОЛОВУ, НАУЧИМСЯ ПОВОРАЧИВАТЬ ШЕЮ, ЧТОБЫ РОБОНЯША МОГ СМОТРЕТЬ ИЗ СТОРОНЫ В СТОРОНУ.

## ПОВОРОТ ГОЛОВЫ

Для этого нам потребуется двигатель, но обычные двигатели вращаются непрерывно с заданной скоростью. Для головы это не подходит, она должна поворачиваться влево или вправо только до заданного угла.

Если прикрепить к двигателю энкодер (смотри эксперимент «Одометр»), можно повернуться на заданный угол и остановиться. Получится особый тип привода — сервопривод. Хорошо, что в наборе уже есть готовый сервопривод, давай им воспользуемся.



Сервопривод



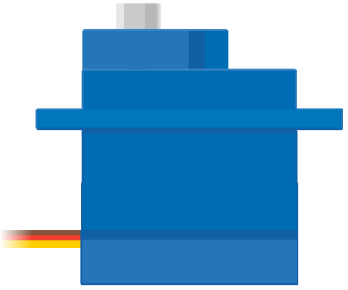
В комплекте с сервоприводом также идёт набор качалок. С их помощью можно удобно прикреплять детали к сервоприводу.

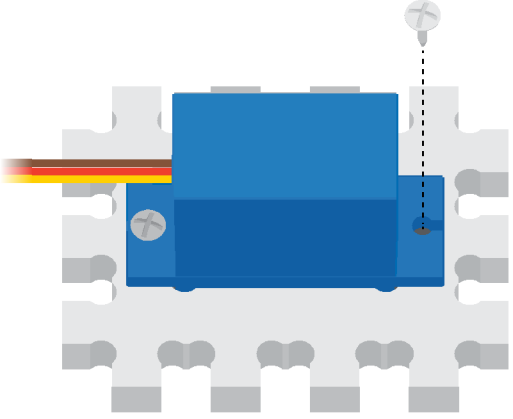
Сервопривод — это двигатель с энкодером в едином корпусе. Сервопривод (или просто — серво) не умеет вращаться непрерывно, зато он умеет устанавливать свой вал на заданный угол. Например, мы можем задать угол в 34°. Серво переведёт вал на этот угол и будет удерживать его. Сервопривод может поворачиваться на угол от 0° до 180°.

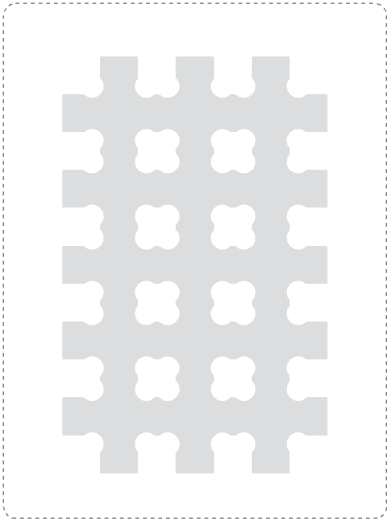
52



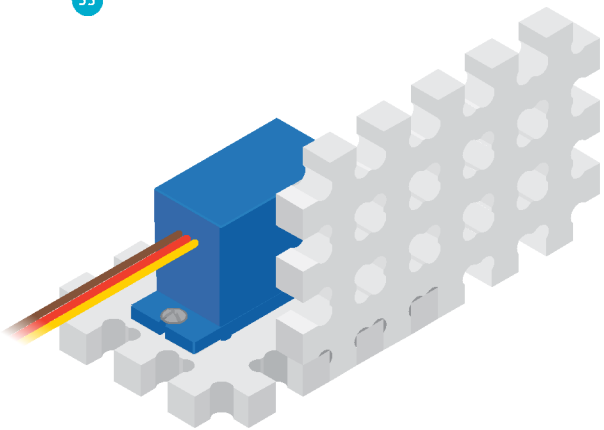
×2

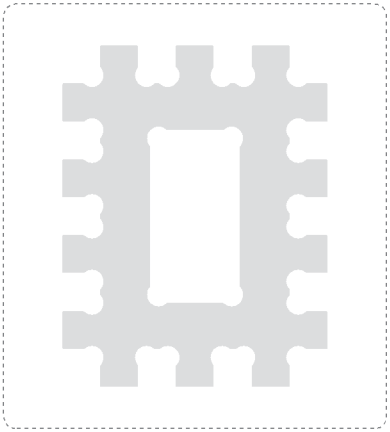




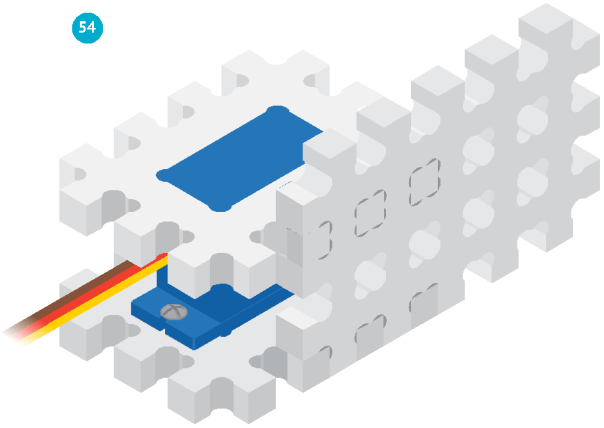


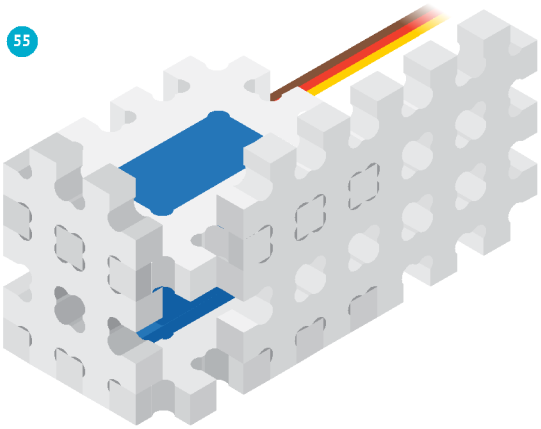
53

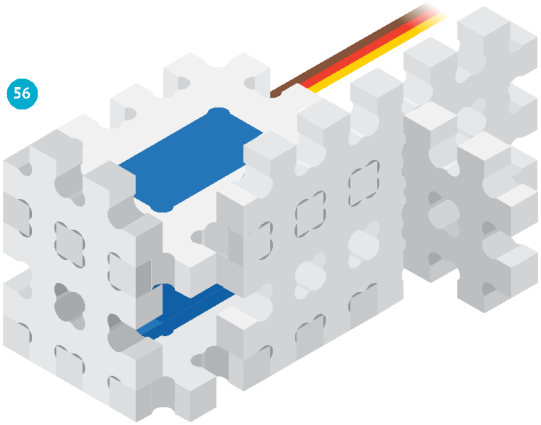


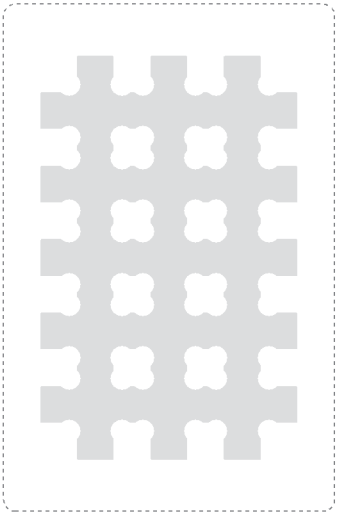


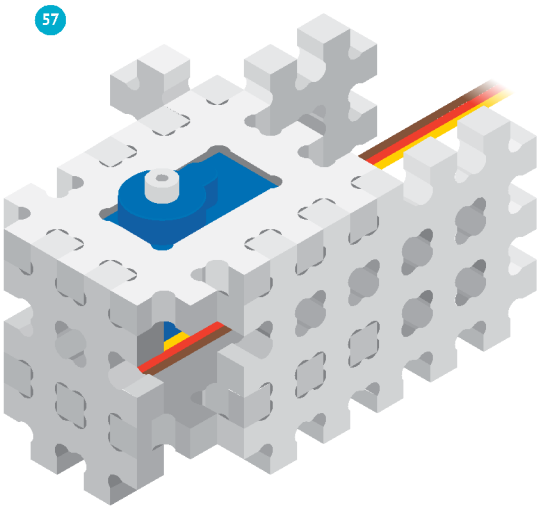
54

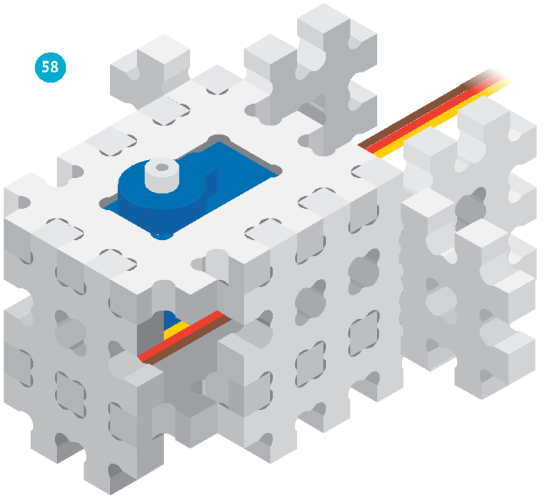



55


56

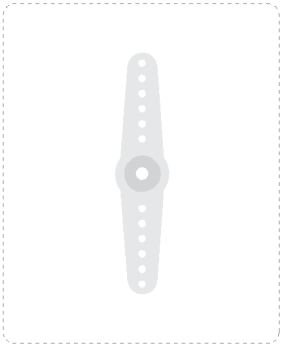


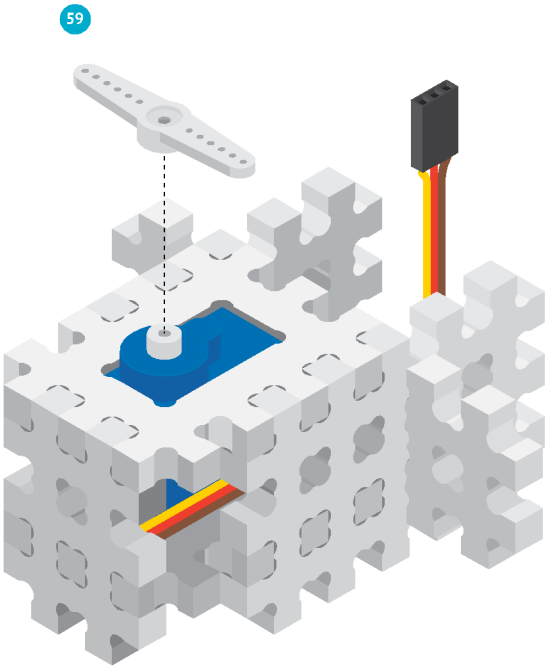

57

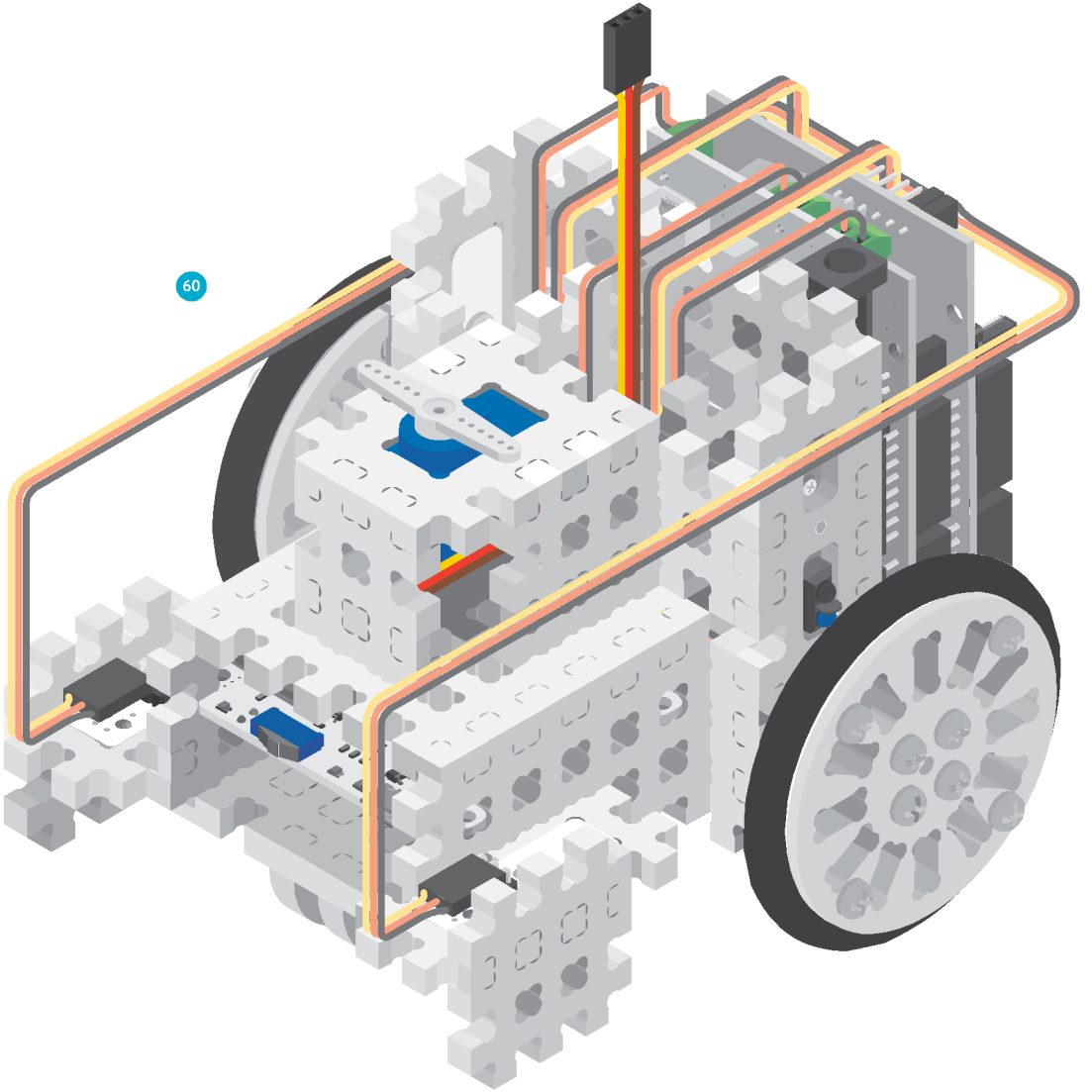

58

## ВАЖНО!

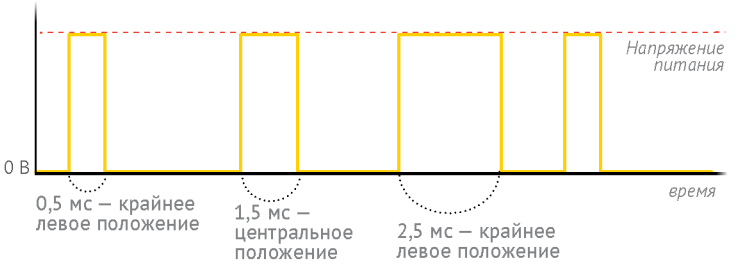
Вал сервопривода имеет небольшие засечки. Такие же засечки есть и на качалке. Качалку можно закрепить на валу в разных начальных положениях.



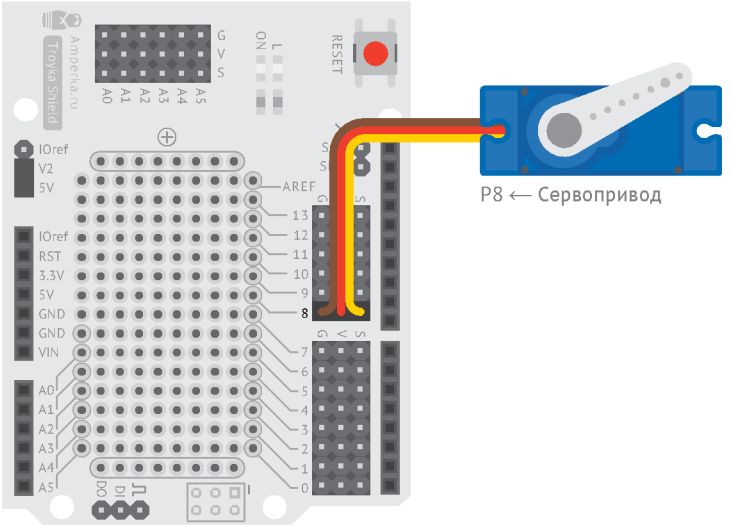



60

Сервопривод управляется цифровыми импульсами различной длительности. Длительность импульса определяет угол, на который сервопривод должен повернуть. Импульсы передаются по сигнальной линии каждые 20 миллисекунд.




Для управления сервоприводом с помощью Iskra JS существует библиотека `'@amperka/servo'`. В ней уже заданы длительности импульсов для поворота на заданные углы.




Troyka Shield + Iskra JS

```
var neck = require('@amperka/servo').connect(P8);

var angle = 90;
var STEP = 5;

setInterval(function() {
  if (angle <= 30 || angle >= 150) {
    STEP = -STEP;
  }
  angle = angle + STEP;
  neck.write(angle);
}, 100);
```

**a** С помощью библиотеки `'@amperka/servo'` подключаем наш сервопривод к пину P8.

**b** Задаём первоначальный угол серво на 90° и шаг поворота серво. Пусть шаг будет 5°. Поэкспериментируй с разными значениями.

**c** Когда вал повернётся максимально влево или максимально вправо, меняем знак шага с плюса на минус и наоборот. Так мы заставим серво изменить направление поворота.

**d** Прибавляем к установленному углу 1 шаг. Если значение шага отрицательное, угол будет уменьшаться.

**e** Функция `write()` устанавливает угол вала сервопривода.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Добавь к эксперименту ИК-приёмник и ИК-пульт. Добейся, чтобы угол поворота качалки управлялся кнопками «+» и «–» на пульте.

№ 11

# ПРИЛИПАЛА

СДЕЛАЕМ ИЗ РОБОТА ВЕРНОГО ДОМАШНЕГО ПИТОМЦА, КОТОРЫЙ БУДЕТ ВЕЗДЕ ХОДИТЬ ПО ПЯТАМ. ЗАДЕЙСТВУЕМ ДЛЯ ЭТОГО УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР.

## УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР

Робот будет замерять расстояние до твоей ладони и стараться постоянно удерживаться на заданном расстоянии от неё, не слишком близко и не слишком далеко.

Ультразвуковой дальномер, как и датчик линии, имеет приёмник и передатчик. Только они не инфракрасные, а ультразвуковые. Ультразвук (УЗ) — это звук очень высокой частоты. Этот УЗ-датчик работает на частоте 40 килогерц, а человек может слышать звук до 20 килогерц.

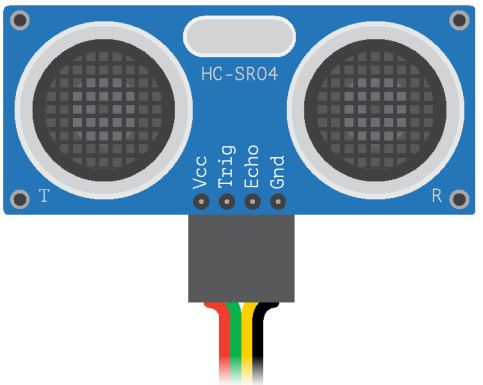
Передатчик генерирует звуковые импульсы, а приёмник слушает эхо. По времени распространения звуковой волны туда и обратно можно определить расстояние до объекта.

Звук распространяется в воздухе со скоростью примерно 340 м/с. Зная скорость звука V и время возвращения эха t, можно подсчитать расстояние до объекта по уже знакомой формуле:

$$S = V \times t / 2$$

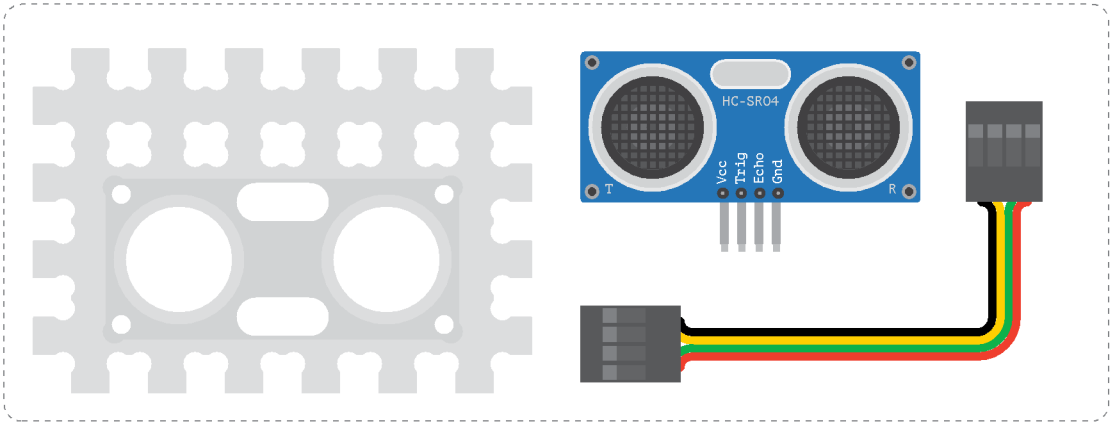
Звук проходит расстояние от датчика до объекта и обратно, поэтому реальное расстояние делим пополам.

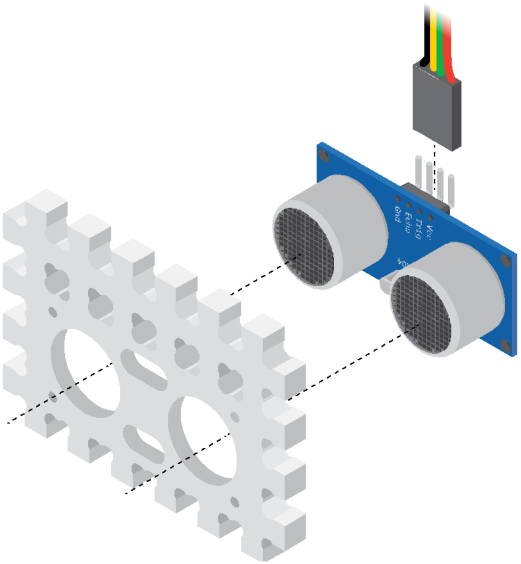
УЗ-дальномеры хороши тем, что могут работать даже при ярком солнце, ведь они совсем нечувствительны к свету.

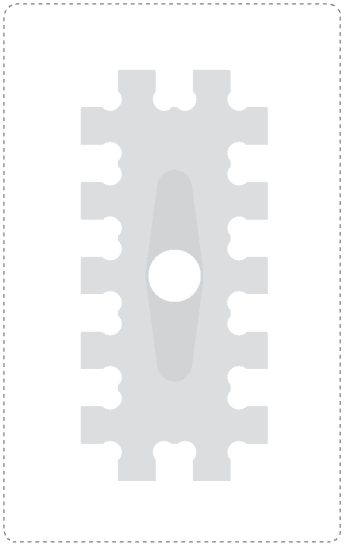


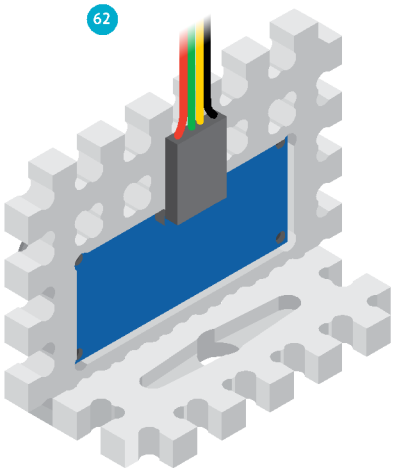
УЗ-дальномер, благодаря своей форме, отлично подходит на роль глаз Робоняши.

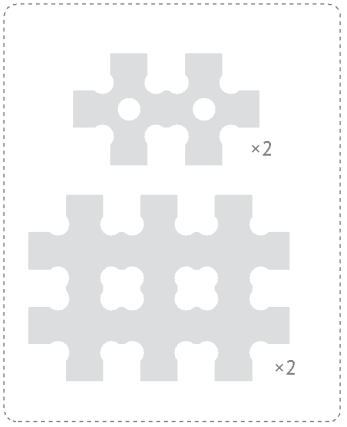
61 Размести датчик на голове робота.

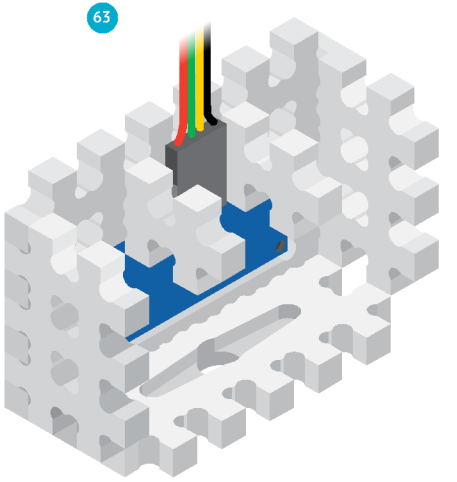





62


×2
×2


63

## УСТАНАВЛИВАЕМ ГОЛОВУ

На этом этапе сборки важно, чтобы голова Робоняши смотрела прямо при повороте серво на 90°. Как ты уже знаешь, серво умеет поворачиваться на угол от 0° до 180°. 90° — как раз середина этого диапазона. Значит, после сборки Робоняша сможет поворачивать голову и влево, и вправо одинаково хорошо.

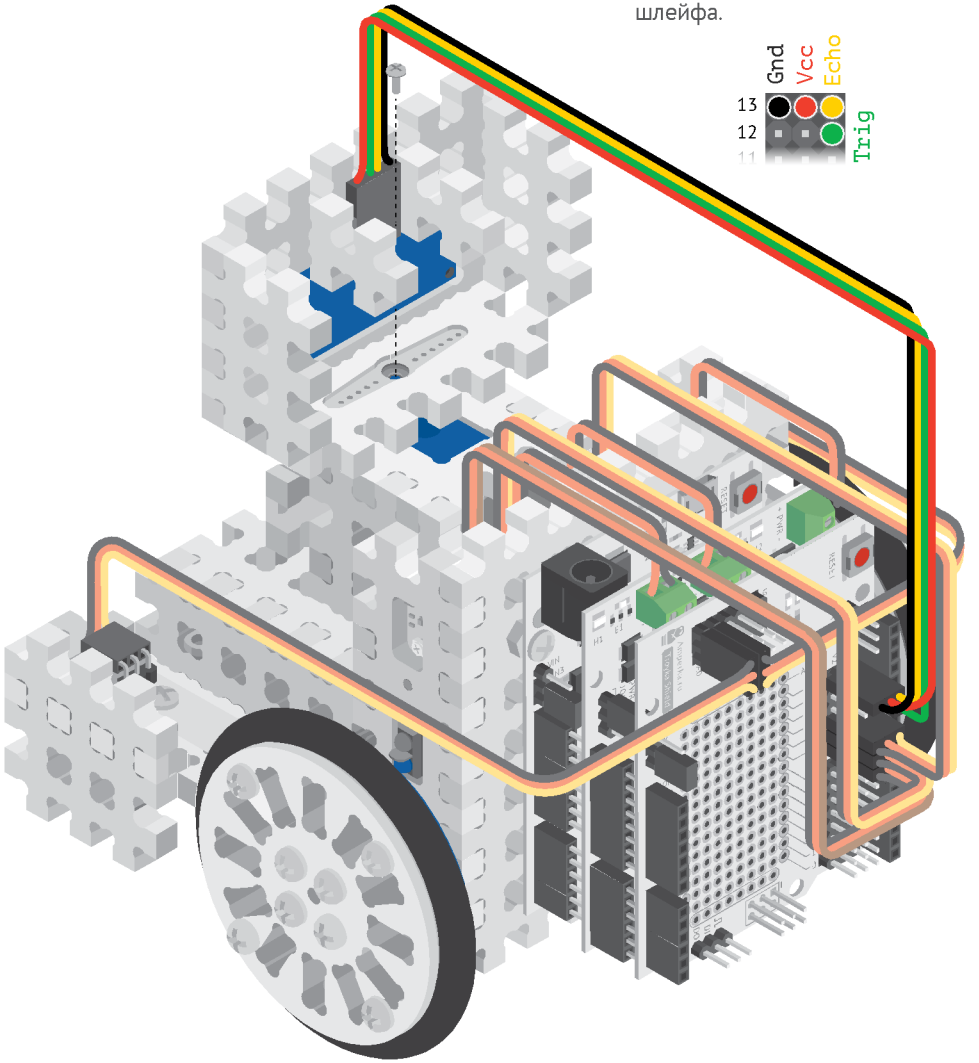
```
1   var neck = require('@amperka/servo').connect(P8);
2
3   neck.write(90);
```

a Подключаем библиотеку для управления сервоприводом.

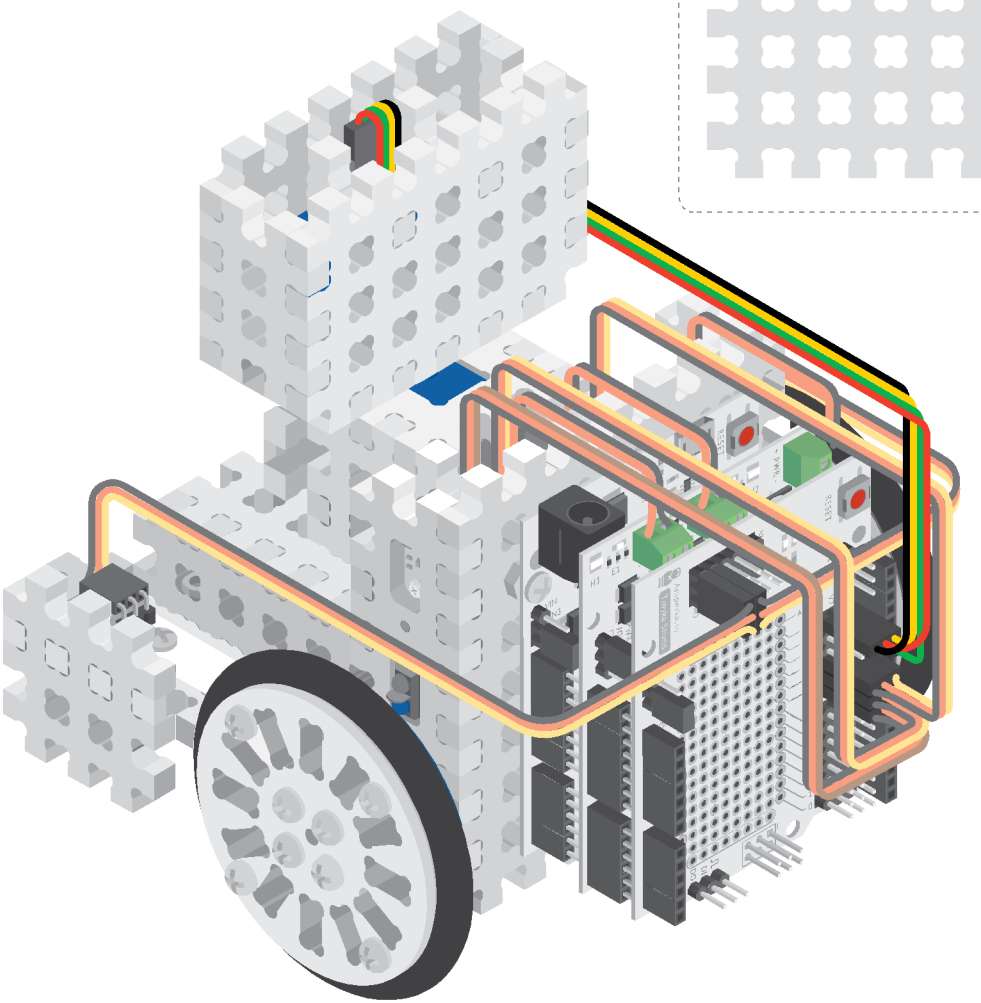
b Устанавливаем угол вала сервопривода на 90 градусов.

64 Не отключая Робоняшу от питания, соедини детали по схеме и установи на Робоняшу. Следи, чтобы голова смотрела строго вперёд.

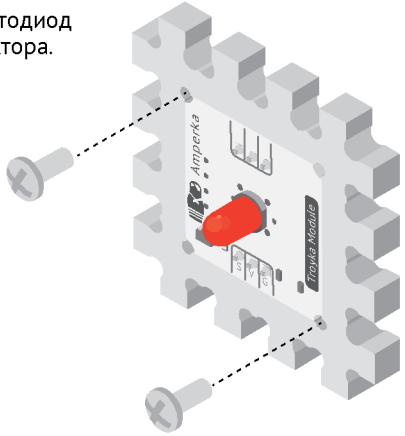
Следи за правильным подключением четырёхпроводного шлейфа.



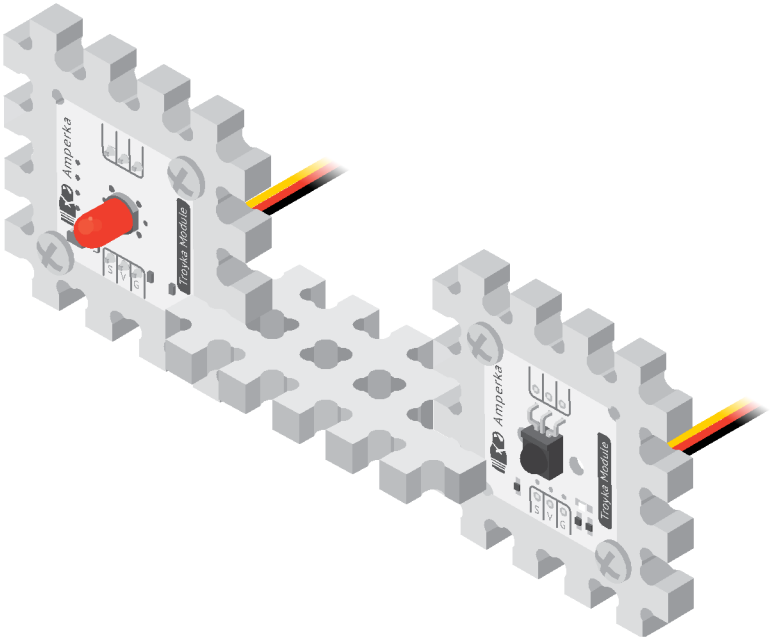
65 Четырёхпроводной шлейф дальномера вдень в нижнее правое отверстие в голове и закрой задней стенкой.



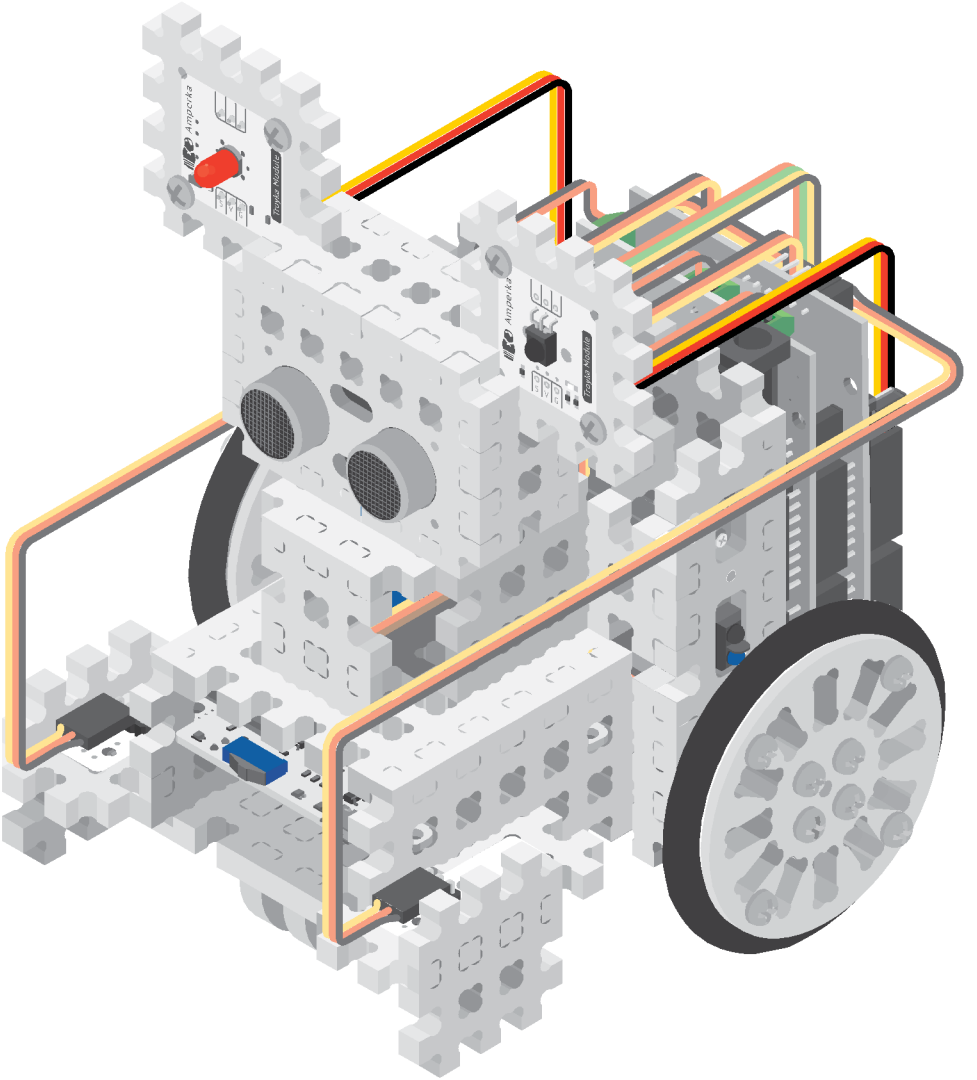
66 Закрепи светодиод в панели #Структора.



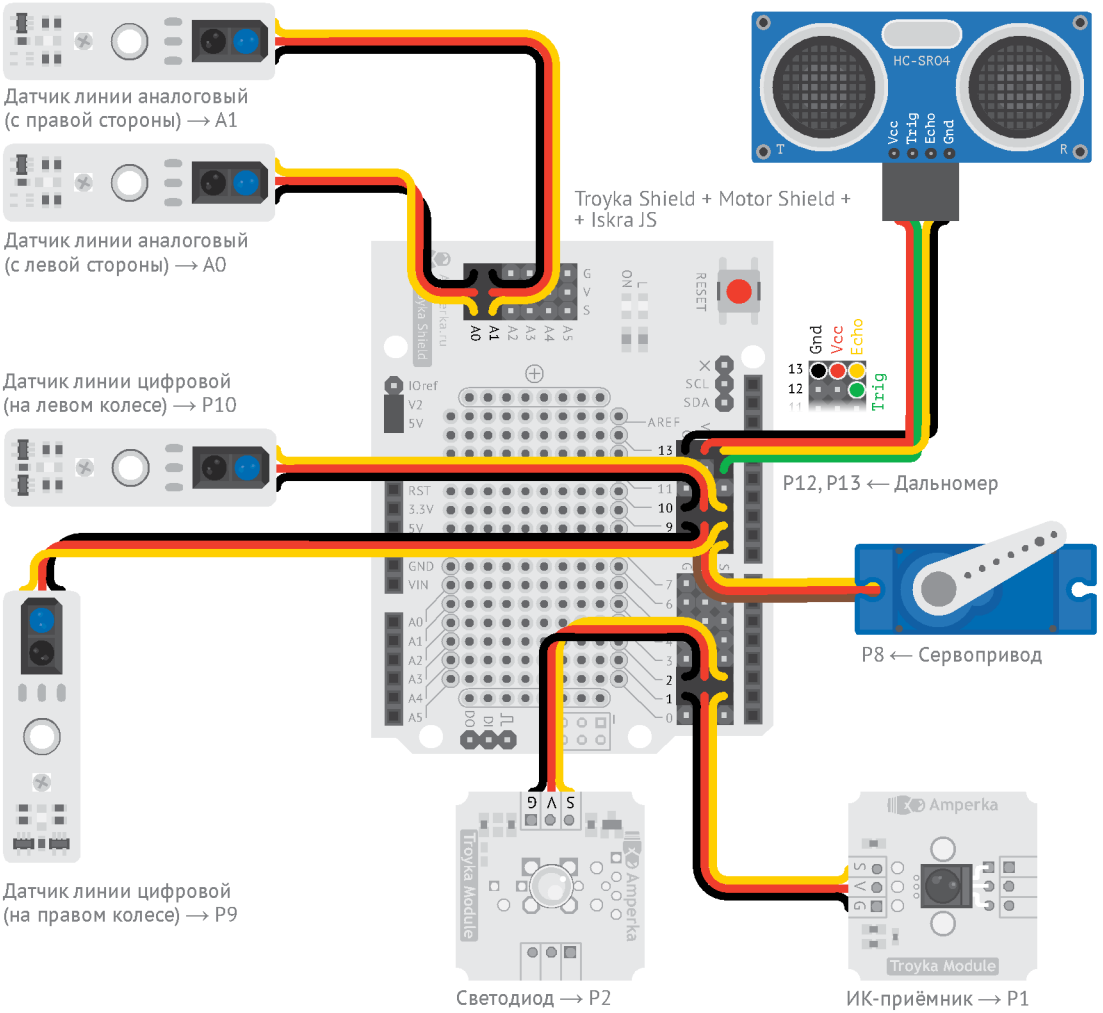

67



68 Ура! Готово! Робоняша полностью собран.



69 Подключи дальномер. Проверь остальные соединения.



**a** Константами `DISTANCE_MIN` и `DISTANCE_MAX` будем задавать минимальное и максимальное расстояние до ладони в сантиметрах.

**b** Библиотека для работы с УЗ-датчиком называется `'@amperka/ultrasonic'`. В функцию `connect` передаём объект с указанием пинов, к которым подключены сигнальные пины датчика. `trigPin` — пин управления передатчиком, `echoPin` — сигнальный пин приёмника.

**c** Если расстояние до руки больше допустимого, робот едет вперёд. Если меньше — едет назад. Если расстояние в заданных пределах — робот просто стоит.

```
var SPEED = 0.5;
var DISTANCE_MIN = 10;
var DISTANCE_MAX = 14;

var sticker = require('@amperka/robot-2wd')
  .connect();

var ultrasonic = require('@amperka/ultrasonic').connect({
  trigPin: P12,
  echoPin: P13
});

var check = function(distance) {
  if (distance > DISTANCE_MAX) {
    sticker.go({l: SPEED, r: SPEED});
  } else if (distance < DISTANCE_MIN) {
    sticker.go({l: -SPEED, r: -SPEED});
  } else {
    sticker.stop();
  }
};

setInterval(function() {
  ultrasonic.ping(function(error, value) {
    if (!error) {
      check(value);
    }
  }, 'cm');
}, 100);
```

d Функция `ping()` объекта `ultrasonic` измеряет расстояние до объекта. Результат измерения передаётся в функцию в виде переменной `value`. Единицы измерения расстояния указываются во втором аргументе функции `ping()`. Мы сейчас использовали сантиметры, но также есть и другие величины: миллиметры и метры.

e Иногда произвести измерения не удаётся. Например, расстояние до предмета слишком велико или измерения выполняются слишком часто. В этом случае переменная `error` не будет пустой, а будет содержать информацию о характере ошибки.

После каждого измерения всегда проверяй `error` на наличие ошибок.

**ПОДУМАЙ САМОСТОЯТЕЛЬНО**

Поэкспериментируй со значением констант: задай скорость Робоняши выше, а `DISTANCE_MIN` и `DISTANCE_MAX` меньше.

№ 12

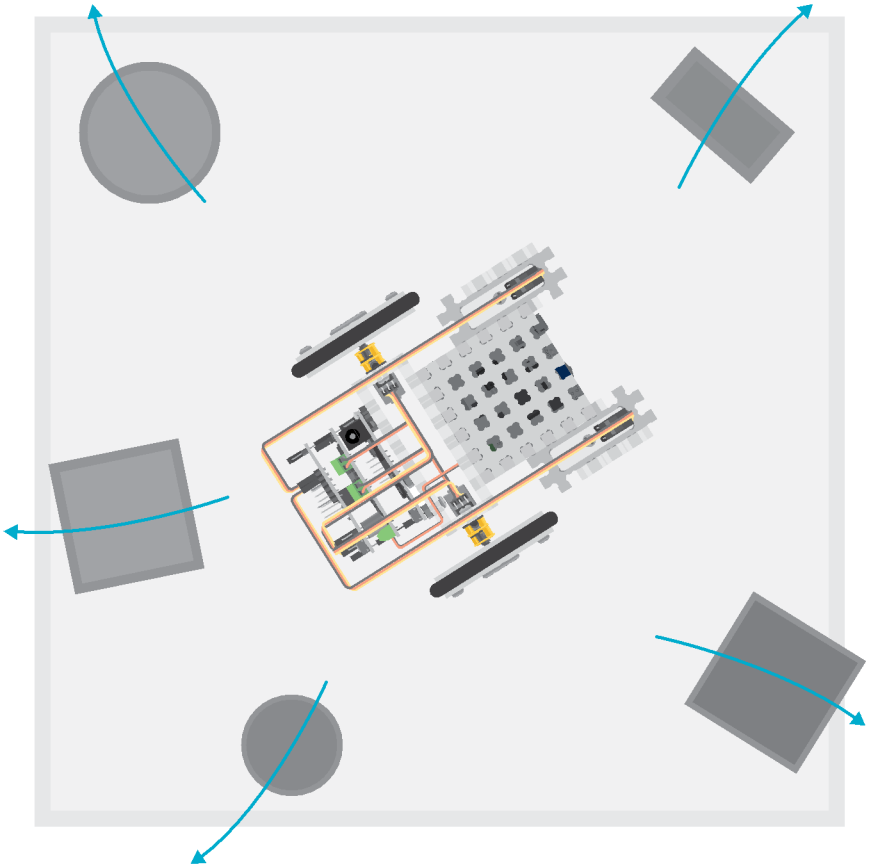
# РОБО-СУМО

РОБО-СУМО — ПОПУЛЯРНЫЙ ВИД СОРЕВНОВАНИЙ ПО РОБОТОТЕХНИКЕ. В ЭТОМ СОСТЯЗАНИИ УЧАСТНИКАМ НЕОБХОДИМО ПОДГОТОВИТЬ АВТОНОМНОГО РОБОТА, СПОСОБНОГО ВЫТАЛКИВАТЬ ПРОТИВНИКА ЗА ПРЕДЕЛЫ РИНГА.

## РИНГ

В этом состязании участникам необходимо подготовить автономного робота, способного выталкивать робота-противника за пределы ринга. Давай сделаем из Робоняши настоящего сумоиста!

Будем выталкивать предметы за пределы стола. Подойдут алюминиевые банки, картонные коробки — любые небьющиеся предметы — или Робоняша другого сумоиста.



a Заводим константы, чтобы задать скорость двигателей в режимах работы Робоняши: FORWARD — во время движения вперёд, выталкивая предметы со стола; BACKWARD — для движения назад, чтобы не упасть; ROTATE — во время вращения Робоняши вокруг себя в поисках предметов.

b Переменная BORDER_VALUE задаёт уровень аналогового сигнала датчика линии для обнаружения границ стола. MAX_DISTANCE определяет максимальное расстояние, на котором Робоняша будет реагировать на предметы.

```
var FORWARD = 0.5;
var BACKWARD = 0.8;
var ROTATE = 0.2;
var BORDER_VALUE = 0.5;
var MAX_DISTANCE = 50;

var sumoist = require('@amperka/robot-2wd')
  .connect();
var ultrasonic = require('@amperka/ultrasonic').connect({
  trigPin: P12,
  echoPin: P13
});
var lineSensor = require('@amperka/analog-line-sensor');
var leftSensor = lineSensor.connect(A0);
var rightSensor = lineSensor.connect(A1);

var caution = false;

var detectBorder = function() {
 if (leftSensor.read() < BORDER_VALUE) {
  caution = true;
  sumoist.go({l: 0, r: -BACKWARD});
 } else if (rightSensor.read() < BORDER_VALUE) {
  caution = true;
  sumoist.go({l: -BACKWARD, r: 0});
 } else {
  caution = false;
 }
};
setInterval(detectBorder, 10);

var scan = function() {
 ultrasonic.ping(function(error, value) {
  if (!error && value < MAX_DISTANCE) {
   if (!caution) {
    sumoist.go({l: FORWARD, r: FORWARD});
   }
  } else {
   sumoist.go({l: ROTATE, r: -ROTATE});
  }
 }, 'cm');
};
setInterval(scan, 100);
```

c Заведём переменную `caution`, которой присвоим значение `true`, когда Робоняша окажется у края стола. Эта переменная будет регулировать приоритет действий робота. Робоняша рискует свалиться вниз? Сдаём назад, не обращая внимания на предметы поблизости.

d Функция `detectBorder` опрашивает датчики линии и проверяет, не приблизился ли робот к краю стола. Если это произошло, робот начнёт двигаться назад, чтобы не упасть. При этом переменной `caution` будет присвоено значение `true`. Она как бы говорит: «Эй, сейчас Робоняша пытается отъехать от края, не мешайте ему!»

e УЗ-дальномер измеряет расстояние до предметов. Если оно оказывается меньше `MAX_DISTANCE`, робот едет в сторону предмета, пытаясь его столкнуть. Если робот оказывается у края стола, переменная `caution` будет равна `true`. В этом случае команда движения вперёд не сработает.

### ПОДУМАЙ САМОСТОЯТЕЛЬНО

Добавь в программу управление светодиодом. Пусть он включается на секунду, когда Робоняша выполняет движение назад от края стола.

# ВООБРАЖАЛА

МЫ ПРОШЛИ 12 ЭКСПЕРИМЕНТОВ, РОБОНЯША ПОЛНОСТЬЮ СОБРАН И МОЖЕТ ВЫПОЛНЯТЬ ВСЯЧЕСКИЕ ЗАДАНИЯ. ТЕПЕРЬ ТЫ МОЖЕШЬ ПРИДУМЫВАТЬ СВОИ ПРОЕКТЫ!

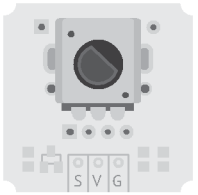
## ЧТО ДАЛЬШЕ?

Подумай над идеей этого эксперимента самостоятельно. Вспомним, что мы уже делали:

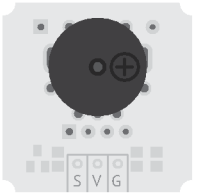
- включали и выключали светодиод;
- управляли роботом через ИК-пульт;
- замеряли скорость и пройденное расстояние;
- ездили вдоль чёрной линии;
- учили Робоняшу не падать на пол;
- следовали по пятам за рукой;
- сталкивали их со стола;
- мотали головой в разные стороны.

## АПГРЕЙДЫ

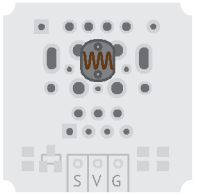
Робоняшу можно оснащать дополнительными модулями. Источники звука, датчики освещённости, шума, температуры, считыватели NFC-карт, Wi-Fi модули и многое другое — всё это можно найти на сайте amperka.ru.



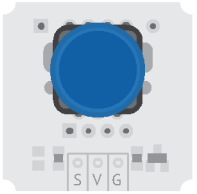

**Потенциометр.**
Сообщает о повороте ручки



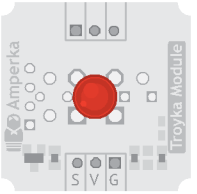

**Зуммер.**
Издаёт звуки



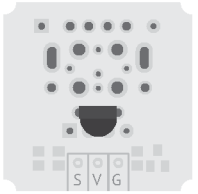

**Датчик освещённости.**
Измеряет яркость света




**Кнопка.**
Сообщает о нажатии




**Светодиод.**
Светит, бывает разного цвета



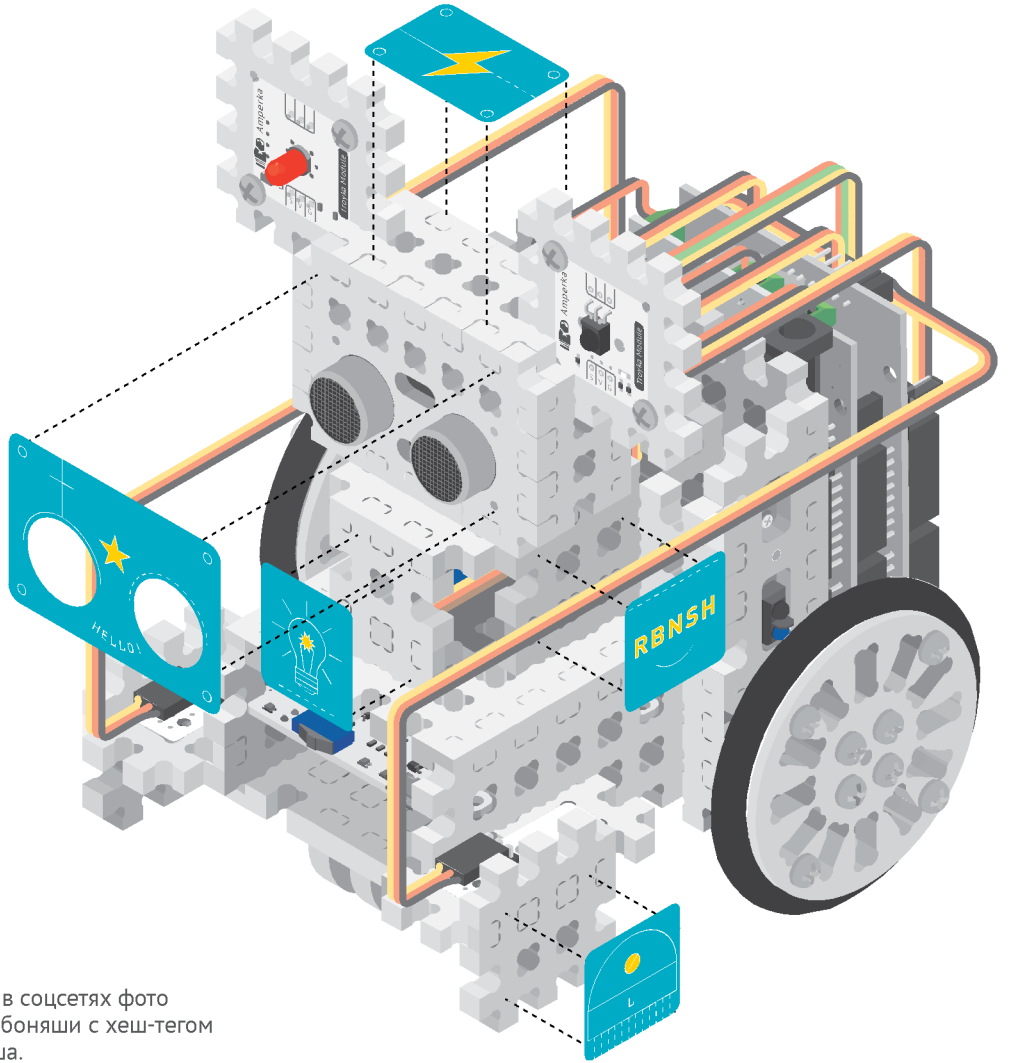

**Термометр.**
Измеряет температуру воздуха

## ПРИДУМАЙ СВОЮ ЗАДАЧУ ДЛЯ РОБОНЯШИ

Робот должен делать что-то полезное и интересное или решать проблему и упрощать повседневную жизнь. Воспользуйся идеями ниже или придумай свою собственную.

| *Идея проекта* | *Что делает* |
|---|---|
| Робот-охранник | Робот-охранник поворотом головы осматривает территорию. Если видит нарушителя границы, включает прожектор и едет на задержание. |
| Продвинутый следопыт | Ездит по линии, как в эксперименте «Следопыт». Коэффициенты ПИД-регулятора настраиваются кнопками на ИК-пульте. |
| Танцор | Выполняет движения вперёд, назад, развороты и повороты головой в заданной последовательности. Для красоты можно мигать светодиодом. |
| Музыкант | Издаёт звуки во время танца.<br>Дополнительно потребуется Troyka-модуль зуммер. |
| Bluetooth-робот | Управляется через смартфон по беспроводной связи bluetooth.<br>Дополнительно потребуется Troyka-модуль bluetooth. |
| ИК-танк | Стреляет в другого Робоняшу ИК-лучами. Дополнительно потребуется ИК-светодиод и друг с Робоняшей. |

# СТИКЕРЫ




Размести в соцсетях фото
своего Робоняши с хеш-тегом
#робоняша.

# СПРАВОЧНИК

## СВЕТОДИОД

```
var led = require('@amperka/led').connect(P1);
```

• `led.turnOn()` — включить;
• `led.turnOff()` — выключить;
• `led.toggle()` — если выключен — включить, если включён — выключить;
• `led.blink(0.2, 0.8)` — мигать: 0,2 секунды гореть, 0,8 секунд не гореть;
• `led.blink(0.2)` — мигнуть 1 раз на 0,2 секунды;
• `led.brightness(0.42)` — установить яркость на 42 %.

## ИК-ПРИЁМНИК

```
var ir = require('@amperka/ir-receiver').connect(P3);
```

Вызвать функцию при нажатии кнопки на ИК-пульте:

```
ir.on('receive', function(code, repeat) {
 if (code === ir.keys.TOP) {
  …
 }
});
```

## ДАТЧИК ЛИНИИ ЦИФРОВОЙ

```
var bumper = require('@amperka/digital-line-sensor')
 .connect(P9);
```

• `bumper.read()` — возвращает `'black'`, если под датчиком чёрный цвет или пустота. Возвращает `'white'`, если под датчиком — белый цвет;
• `bumper.on('black', function() { … })` — вызвать функцию при появлении чёрного цвета или пустоты;
• `bumper.on('white', function() { … })` — вызвать функцию при появлении белого цвета.

## ДАТЧИК ЛИНИИ АНАЛОГОВЫЙ

```
var bumper = require('@amperka/digital-line-sensor')
 .connect(A0);
```

• `bumper.calibrate({ black: 0.2, white: 0.8 })` — установить крайние значения белого и чёрного;
• `bumper.read()` — возвращает значение уровня серого в диапазоне 0,0 … 1,0.

## ДВУХКОЛЁСНАЯ ПЛАТФОРМА

```
var robot = require('@amperka/robot-2wd').connect();
```

• `robot.stop()` — остановка робота;
• `robot.go({l: -0.5, r: 0.85})` — задать скорость двигателям: левый — назад на 50 %, правый — вперёд на 85 %.

## УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР

```
var sonic = require('@amperka/ultrasonic')
 .connect({trigPin: P12, echoPin: P11});
```

Измерить значение и вызвать функцию с результатом:

```
sonic.ping(function(err, value) {
 if (err) {
  print('ERROR:', err);
 } else {
  print('The distance is:', value, 'millimeters');
 }
}, 'mm');
```

## СЕРВОПРИВОД

```
var servo = require('@amperka/servo').connect(P8);
```

`servo.write(120)` — повернуть в положение 120°.

# СОДЕРЖАНИЕ

Амперка

Помощь на форуме:
forum.amperka.ru

Электронная версия книги:
robot.amperka.ru

Руководства и инструкции:
wiki.amperka.ru

Ютьюб канал:
youtube.com/AmperkaRU

Новые платы и модули:
amperka.ru

Дизайн: студия «Кластер»
clusterstudio.ru

vk.com/amperkaru

facebook.com/amperka.ru

instagram.com/amperkaru

twitter.com/amperka


Амперка