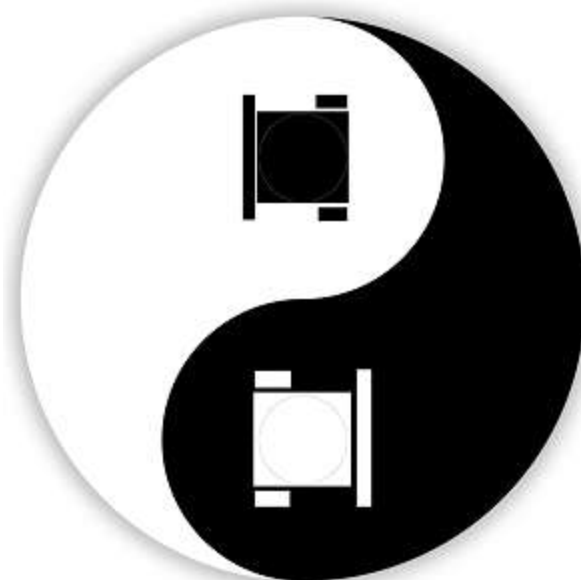


Введение



Книга задумана как информационное пособие для начинающего робоспортсмена в одной из наиболее популярных дисциплин робоспорта – «Соревнования сумороботов 10x10» и альтернатива крупноблочному «кубиковому» созданию роботов, чрезвычайно широко распространенному на территории всего мира.

Автору показалось, что эта альтернатива нужна для того, чтобы НАСТОЯЩИЙ РОБОСПОРТСМЕН мог не только мыслить в концепциях готовых блоков и их ограничений, но еще уметь создавать эти блоки своими руками по СВОЕМУ ЗАМЫСЛУ, делая роботов более эффективными и, параллельно, получая больше знаний и умений в смежных областях, связанных с трудом СВОИМИ РУКАМИ И ГОЛОВОЙ.

Подбирать материалы, пилить, паять, компоновать и травить (в хорошем, паяльном смысле этого слова), делать формы и лить в них металл и многие другие знания и умения, главным из которых будет умение верить в свои силы – для РОБОСПОРТСМЕНА все должно быть возможным и доступным.

Книга написана для широкого круга школьников, студентов, начинающих энтузиастов робототехники и робоспортсменов, всегда юных пенсионеров.

Цель книги – рассказать о том, как сделать робота для соревнований сумо «10x10» по регламентам Robotchallenge.org на основе, пожалуй, самого распространенного и доступного по цене и функционалу микропроцессорного решения Arduino (Funduino, DFRduino и других).

Еще одной целью книги будет погружение читателя в мир робототехники и робоспорта, что может стать залогом для успешного будущего читателя в профессиональных областях, связанных с робототехникой.

Книга не претендует на научную полноту и непогрешимую истинность предложенных решений. Скорее она – катализатор для размышлений и собственных решений в области спортивного роботостроения – в этом ее Заветная Цель.

Урок 1. «Мозг робота». Как он работает. Как его программировать. Учимся подавать первые признаки жизни – мигаем светодиодом.



«Мозги» робота, также как и мозги человека, служат для достижения Основной цели этих разных и одновременно одинаковых субъектов. В случае с сумороботом Основная цель проще – достичь победы в поединке. (В случае, если робота будет проектировать девушка – вероятно еще и красиво достичь победы J).

В нашем роботе и книге в качестве «думающего органа» используется открытая микропроцессорная платформа Arduino, включающая в себя «железную» и программные части. Основными преимуществами данной платформы является ее широкая распространенность, относительная дешевизна, огромное количество «клонов» (DFRduino, SeeDuido, Lillipad и многие прочие), огромное сообщество последователей в сети интернет, применимость платформы во вполне серьезных «взрослых» проектах автоматизации в «умных» домах, промышленности, быту.

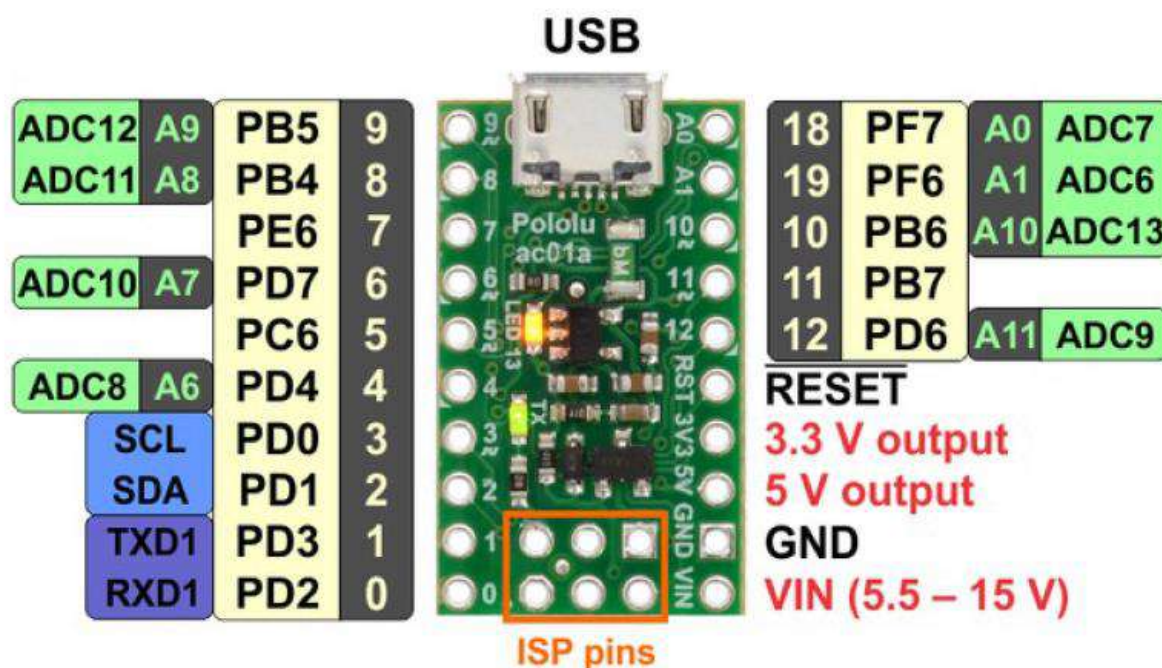
Для построения нашего робота мы выбрали микропроцессорную плату A-Star 32U4 Micro за ее миниатюрность и отличный функционал, соотношение цены и качества.

Как работают мозги нашего будущего робота?

Что же могут наши «мозги»? Очень многое для столь миниатюрной платы. Получать и обрабатывать информацию от «органов чувств» робота – датчиков

дистанции, управлять исполнительными устройствами робота – «ногами и руками» (двигателями). Датчики и управляемые устройства подключаются к контактам платы A-Star 32U4 Micro. Внутри микропроцессора уже есть программа, которая позволяет подключить плату к компьютеру или планшету с установленным программным обеспечением – интегрированной средой разработки Arduino (подробная информация на сайте www.arduino.cc) для «загрузки» сценариев обработки сигналов с датчиком и реакции на них Ваших исполнительных устройств. Данные сценарии называются программами и мы научимся их писать.

Начинаем разбираться с контактами A-Star 32U4 Micro. Официальная документация содержит картинку, которая дает возможность понять, как подключить контакты датчиков исполнительных устройств к плате.



Все просто. Желтые цифры на сером фоне – номера цифровых контактов, которые могут быть использованы в Вашей программе. Зеленые цифры с буквой «А» на сером фоне – номера аналоговых контактов.

Резонный вопрос — цифровой контакт 19 и аналоговый контакт А1 совпадают. Ответ также прост – решите, что будет подключено к данному контакту и программно задайте данному контакту либо **цифровой** либо **аналоговый** статус.

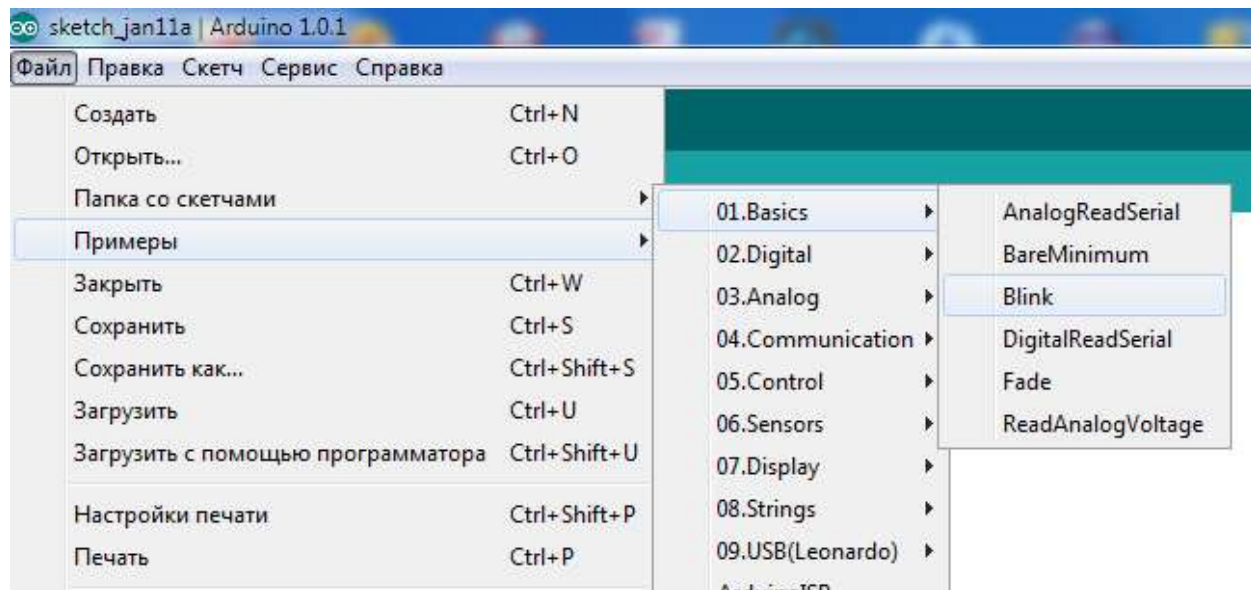
Питание от батарейки подключается к контакту VIN – плюс (от 5.5 до 15 вольт) и контакту GND – минус.

5V output и 3.3V output – контакты для питания датчиков нашего робота. Контакты ISP и SCL SDA TXD1 и RXD1 – искатели приключений сами смогут найти информацию в документации. 😊

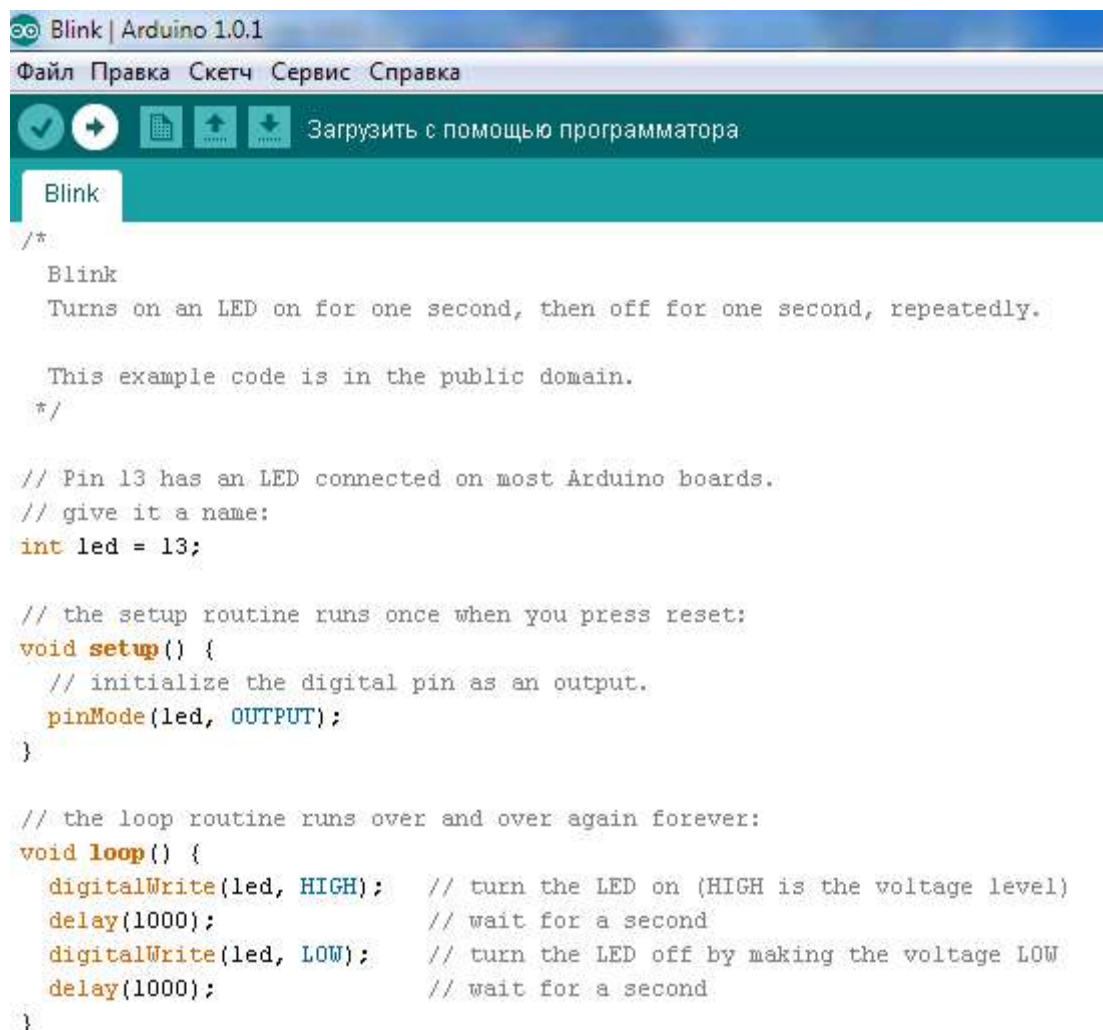
Обратите внимание – на плате есть светодиоды – одним из которых (LED) мы сейчас помигаем миру с целью заявить о своих правах на будущие победы на рингах.

Надеюсь, у Вас получилось установить среду разработки Arduino с сайта www.arduino.cc на Ваш компьютер или планшет. Подробные инструкции для установки последней версии среды, ссылки для скачивания среды для различных операционных систем, вся последняя информация о Arduino есть на этом сайте.

Итак, открываем интегрированную среду разработки Arduino на вашем компьютере (планшете) и выбираем из примеров скетч Blink. [Скетчем в сообществе Arduino называется программа или сценарий поведения Вашего робота, который будет записан в микропроцессор и затем исполнен.]



Видим следующий текст:



Давайте разберемся в структуре скетча. Все что представлено серым цветом – это комментарии к скетчу для Вас – не для «мозгов» платы.

Для «мозгов» платы в теле скетча мы объявляем что переменной `led` будет соответствовать 13 вывод платы, который имеет тип «`int`» от слова `integer` – целое число.

```
int led = 13;
```

В теле скетча есть две процедуры `setup()` и `loop()`. Процедура `setup()` выполняется один раз при включении питания платы и обычно в ней содержится объявление контактов платы и инициализация этих контактов). В частности в теле процедуры (между фигурными скобками { и }) расположена следующая строка:

```
pinMode(led, OUTPUT);
```

Эта строка инициализирует контакт 13 на вывод данных – т.е. к данному контакту будет присоединено исполнительное устройство (светодиод в нашем случае).

Процедура `loop()` интересна тем, что все, что мы располагаем в ее теле будет исполняться «по кругу» — от первой до последней строки и затем опять от первой до последней строки и так пока ваши батарейки не «сядут». В теле нашей процедуры `loop()` расположены 4 строки с командами для микропроцессора платы:

```
digitalWrite(led, HIGH);
```

```
delay(1000);
```

```
digitalWrite(led, LOW);
```

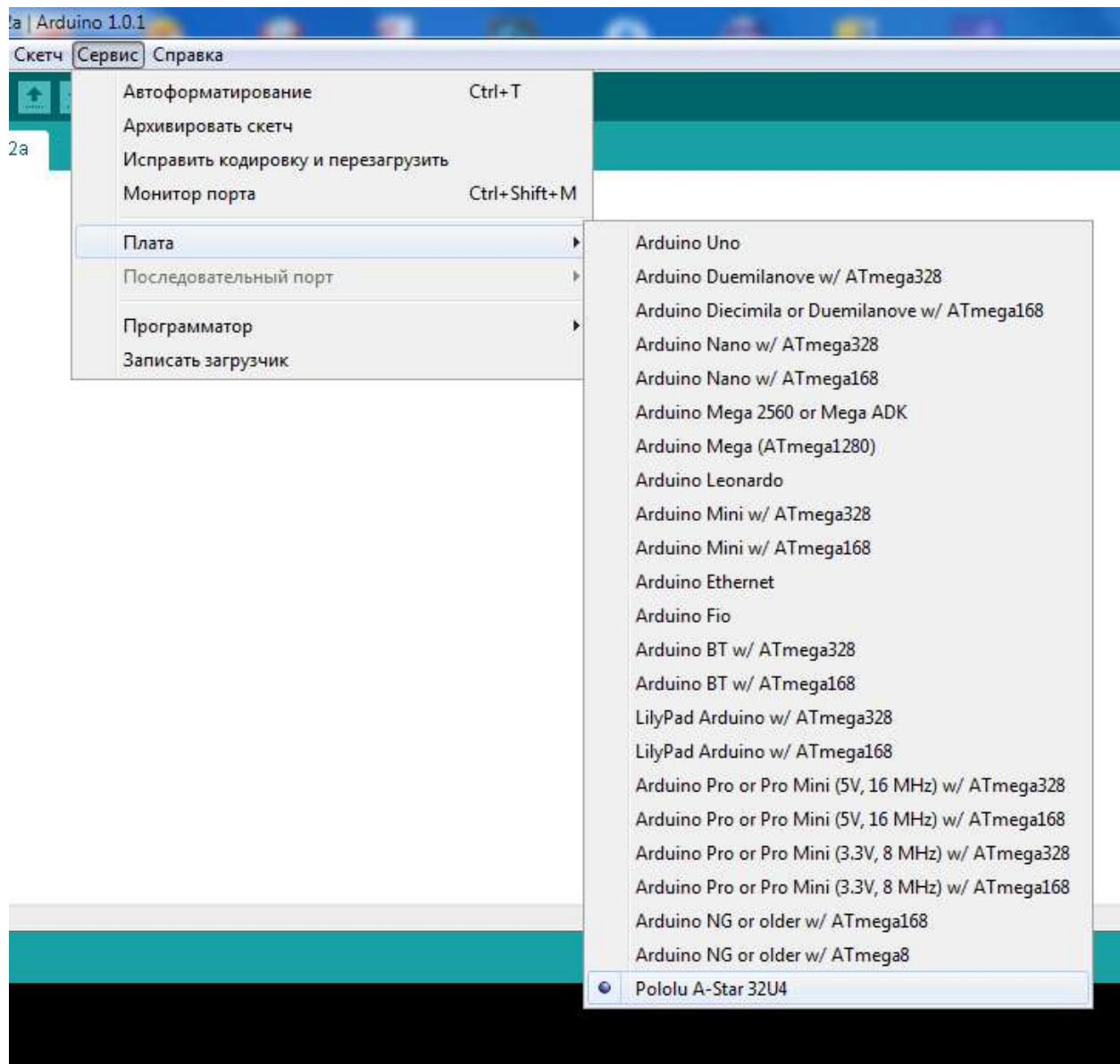
```
delay(1000);
```

Команда `digitalWrite(led, HIGH)` указывает нашему микропроцессору дать на контакт 13 (переменная `led`) высокий уровень напряжения (`HIGH`)

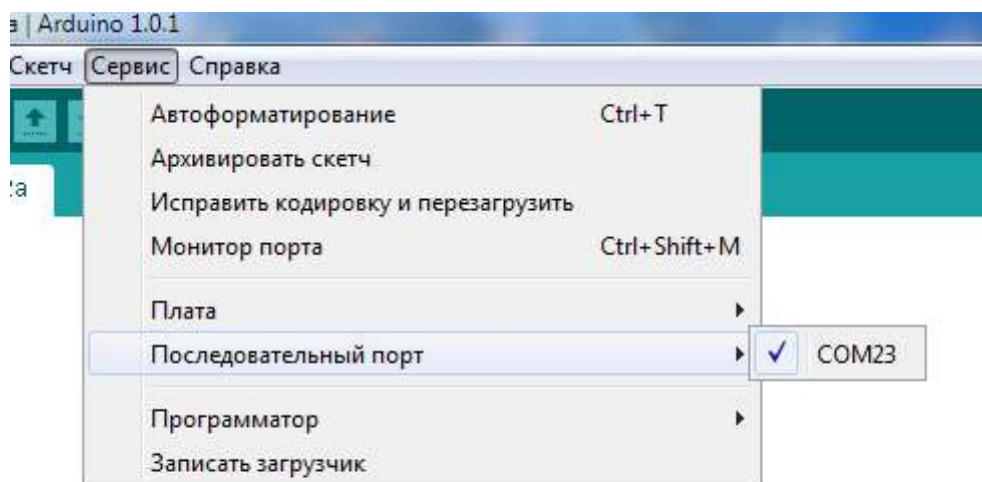
Следующая за ним команда `delay(1000);` задерживает все действия микропроцессора на 1000 миллисекунд или 1 секунду.

Следующая команда `digitalWrite(led, LOW)` снижает напряжение на контакте 13 до низкого (почти 0 вольт) и после нее опять по команде `delay(1000);` опять ожидаем секунду. Затем исполнение опять передается первой строке в теле процедуры `loop` и программа повторяется.

Загрузим данную программу (круглая кнопка со стрелкой вправо в панели меню) в нашу плату предварительно подключив ее кабелем в USB разъем порта и посмотрим, что получилось. Перед загрузкой обязательно убедитесь, что Вы выбрали именно плату Pololu A-Star 32U4 [По умолчанию, драйверов этой платы пока нет в стандартной установке Ардуино. Для ее установки необходимо скачать драйверы USB и установить их на Ваш компьютер (пункт 6 по порядку — драйверы, затем директорию в ардуино — Успеха!)

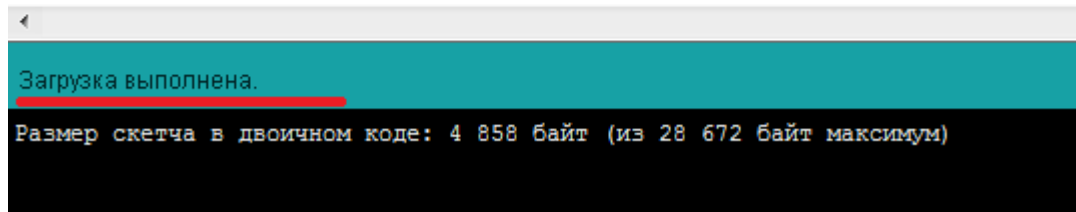


Выбрали COM – порт



Выполнили загрузку:

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



После окончания загрузки плата начинает мигать светодиодом с частотой 1 секунда. – мы научились «подмигивать» миру!!!

Вопросы для проверки:

1. Какие основные типы контактов на микропроцессорной плате A-Star 32U4 Вы знаете? Для чего они используются?
2. Что такое скетчи? Для чего они нужны?
3. Какая интегрированная среда разработки используется для создания скетчей для Arduino роботов? Как загрузить программу в Вашу микропроцессорную плату?
4. Какие основные процедуры должны обязательно быть в скетче для Arduino?

Урок 2. «Ноги» робота. Двигатели. Как ими управлять.



=-

С «мозгами» робота мы более менее прояснили ситуацию – научились «подмигивать» этому миру – давайте учиться ходить. Роботы они как дети 😊 .

Ходить мы будем с использованием моторов, конечно.

Немного по поводу моторов в целом. В спортивной сумо-робототехнике очень важны миниатюрные и мощные моторы, которые бы могли не просто перемещать массу робота (наиболее часто встречающиеся категории «мини» — 500 грамм в размере 10 x 10 см, «микро» -100 грамм в размере 5x5см, и «супер» — 3 кг в размере 20x20 см), но и вытолкнуть чужого робота с такой-же массой, который противостоит Вам, за пределы ринга. Для этого необходимо подбирать моторы, исходя из будущей тактики проведения боя вашим Роботом, количества колес, ширины и материала шин колес, типа робота (гусеничный или колесный), многих других параметров, которые будут Вам очень важны по мере совершенствования Вами Ваших знаний и умений в спортивной робототехнике.

Сейчас наиболее распространенным типом моторов в категории «мини» сумо (в которой мы строим робота), являются микромоторы высокой мощности Pololu (www.pololu.com)

Motor Type	Stall Current @ 6 V	No-Load Speed @ 6 V	Stall Torque @ 6 V	 Gearbox Shaft Only
high-power (HP)	1600 mA	6000 RPM	2 oz-in	5:1 micro HP
		3000 RPM	4 oz-in	10:1 micro HP
		1000 RPM	9 oz-in	30:1 micro HP
		625 RPM	15 oz-in	50:1 micro HP
		400 RPM	22 oz-in	75:1 micro HP
		320 RPM	30 oz-in	100:1 micro HP
		200 RPM	40 oz-in	150:1 micro HP
		140 RPM	50 oz-in	210:1 micro HP
		120 RPM	60 oz-in	250:1 micro HP
		100 RPM	70 oz-in	298:1 micro HP
		32 RPM	125 oz-in	1000:1 micro HP

Эти моторы обеспечивают достаточно большое усилие (от 0.4 кг/см до 4.4 кг/см) и достаточно высокое количество оборотов от 32 до 6000.

Помните – чем меньше оборотов – тем выше мощность. В таблице самые мощные моторы имеют 32 оборота и самые слабые – 6000 оборотов.

Для нашего класса минисумороботов оптимально выбирать моторы от 100 до 1000 оборотов.

Есть аналоги этих моторов восточных производителей, которые Вы сможете найти у крупнейшего продавца современности www.alibaba.com. В России достаточно большое количество магазинов, которые продают подобные моторы и доставляют их (www.electronshtik.ru, www.chipdip.ru, www.amperka.ru)

Выбор колес (их размеров (диаметра и ширины), материала, количества, для робота также очень важная задача.

Есть основные правила:

1. Чем больше диаметр колеса, тем менее оборотистый двигатель должен быть у Вас
2. Чем шире колеса и более мягкая резина у колес, тем робот становится менее маневренным и точно управляемым (в варианте четыре и более колес), но зато более сильным в ситуации «клинча» и выталкивания противника за ринг.
3. Колеса должны жестко крепиться на валу двигателей – нагрузки очень высоки и возможность потерять колесо в бою и проиграть тем выше, чем слабее крепление колес.

Обычно для роботов сумо покупают колеса в тех-же магазинах, что и двигатели, либо изготавливают сами на основе стандартных комплектов, печатью на 3d принтерах и литьем силикона, вытачиванием на токарном станке и прочими способами.

Мы будем использовать колеса собственного изготовления из силикона специально рассчитанные для выбранных двигателей (видео инструкция по их изготовлению). Также эти колеса доступны в магазине toborobot.ru.

Перед подключением двигателей к нашей микропроцессорной плате давайте проверим, совместимы ли они друг с другом, сможет ли «мозг» управлять «ногами» напрямую?

Итак, первое что мы смотрим в документации к микропроцессорной плате AStar 32U4 какой максимально допустимый ток (сила тока) возможна для снабжения потребителей (в том числе наших «ног»-двигателей?

На странице 7 документации мы можем найти следующую информацию:

USB power input: The A-Star can be powered from the USB 5 V bus voltage (VBUS) if it is connected to a USB cable. A resettable PTC fuse on VBUS makes it less likely for the A-Star (and the connected computer or other device) to be damaged if too much current is drawn from the USB connection.

VIN power input: The A-Star can be powered from VIN if you connect a 5.5 V to 15 V power supply (such as a battery or wall power adapter) to the VIN and GND pins, with the positive terminal connected to VIN. It will draw power from VIN only if USB is disconnected.

When powering the A-Star 32U4 Micro from VIN, a minimum voltage of 5.5 V is required to ensure that the board's 5 V supply is stable. Even if power is being provided to the A-Star via USB, connecting a voltage higher than 0 V but lower than 5.5 V to VIN is not recommended, as this can interfere with the power selection circuit and cause the 5 V line to drop (potentially triggering a brown-out reset).

5V power output: This pin provides access to the board's 5 V supply, which comes from either the USB 5 V bus voltage or a low-dropout (LDO) regulator on VIN, depending on which power source is connected. The regulator can supply up to 100 mA, although some of this is used by the board itself (typically about 25 mA) or used to provide current for the GPIO pins or 3.3 V power output (see below).

3V3 power output: This pin gives access to the output of the internal 3.3 V regulator inside the ATmega32U4. The microcontroller uses this regulated voltage for USB signaling, but up to about 50 mA is available for powering external circuits or devices.

When the A-Star 32U4 Micro is being powered through VIN, the sum of the 5V output current, 3V3 output current, GPIO output current, and current used by the board itself should not exceed the 100 mA that the regulator can provide.

Главное для нас следующее:

Регулятор питания самой платы может обеспечить на все выводы (включая питание 5V и 3.3V для других устройств не более 100mA (при этом потребленное самой платы составляет примерно 25mA) что еще уменьшает возможный бюджет для остальных потребителей.

Вторым шагом будет изучение, какая сила тока необходима для питания двигателей. Также обращаемся к документации по выбранным нами моторам (на сайте www.pololu.com):

Motors and Gearboxes » Pololu Metal Gearmotors » Micro Metal Gearmotors »
50:1 Micro Metal Gearmotor HP



This gearmotor is a miniature (0.94" x 0.39" x 0.47"), high-power brushed DC motor w popular 12 mm gearmotors. These units have a 0.365"-long, 3 mm-diameter D-shaped ou

Key specs at 6 V: 625 RPM and 100 mA free-run, 15 oz-in (1.1 kg-cm) and 1.6 A stall.

Итак, читаем, что при 6 вольтах напряжения питания мотор, крутясь свободно (без нагрузки), потребляет 100mA, а под нагрузкой в момент остановки целых 1,6A или 1600mA. Это значит, что просто двигаясь по рингу робот будет потреблять примерно 300mA на один мотор, а выталкивая противника с ринга 500-900mA на один мотор. Вывод однозначный – микропроцессорная плата не сможет даже начать прокручивать один мотор если подсоединить мотор к плате напрямую – просто сгорит.

Вопросы для проверки:

1. Как связаны обороты двигателей с мощностью?
2. Каков общий бюджет по току микропроцессорной платы AStar 32U4?
3. Подумайте, почему происходит увеличение потребления тока моторами при выталкивании соперников с ринга?
4. Подумайте, кроме силы тока, какие электрические параметры важны при подборе комплектующих для робототехнического проекта?

Урок 3. Усиливаем «ноги робота». Драйвер двигателей.



Что делать в ситуации когда моторам необходимо много тока, а у микропроцессорной платы его просто нет? Правильно — «звать друзей» J.

Какие возможны варианты?

1. Применять реле
2. Применять транзисторы
3. Применять специальные микросхемы –драйверы моторов

Первый и второй путь более сложные с точки зрения миниатюризации и уменьшения количества контактов на пайке, поэтому сразу пойдем третьим путем.

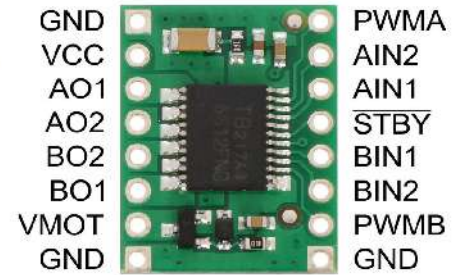
Итак – нам необходима специальная микросхема-драйвер двигателей, которая поможет нам справиться с токами, необходимыми для питания моторов робота.

Таких драйверов достаточно много в зависимости от типов моторов, используемого напряжения и силы тока.

Мы выберем оптимальный драйвер для таких моторов, как у нас – TB6612FNG (www.pololu.com). Этот драйвер, основанный на одноименной микросхеме может управлять двумя моторами сразу, что нам и необходимо. В [документации](#) к нему мы можем найти необходимые для нас параметры.

Features and specifications

- Dual-H-bridge motor driver: can drive two DC motors or one bipolar stepper motor
- Recommended motor voltage (VMOT): 4.5 V to 13.5 V (can operate down to 2.5 V with derated performance)
- Logic voltage (VCC): 2.7 V to 5.5 V
- Output current maximum: 3 A per channel
- Output current continuous: 1 A per channel (can be paralleled to deliver 2 A)
- Maximum PWM frequency: 100 kHz
- Built-in thermal shutdown circuit
- Filtering capacitors on both supply lines
- Reverse-power protection on the motor supply



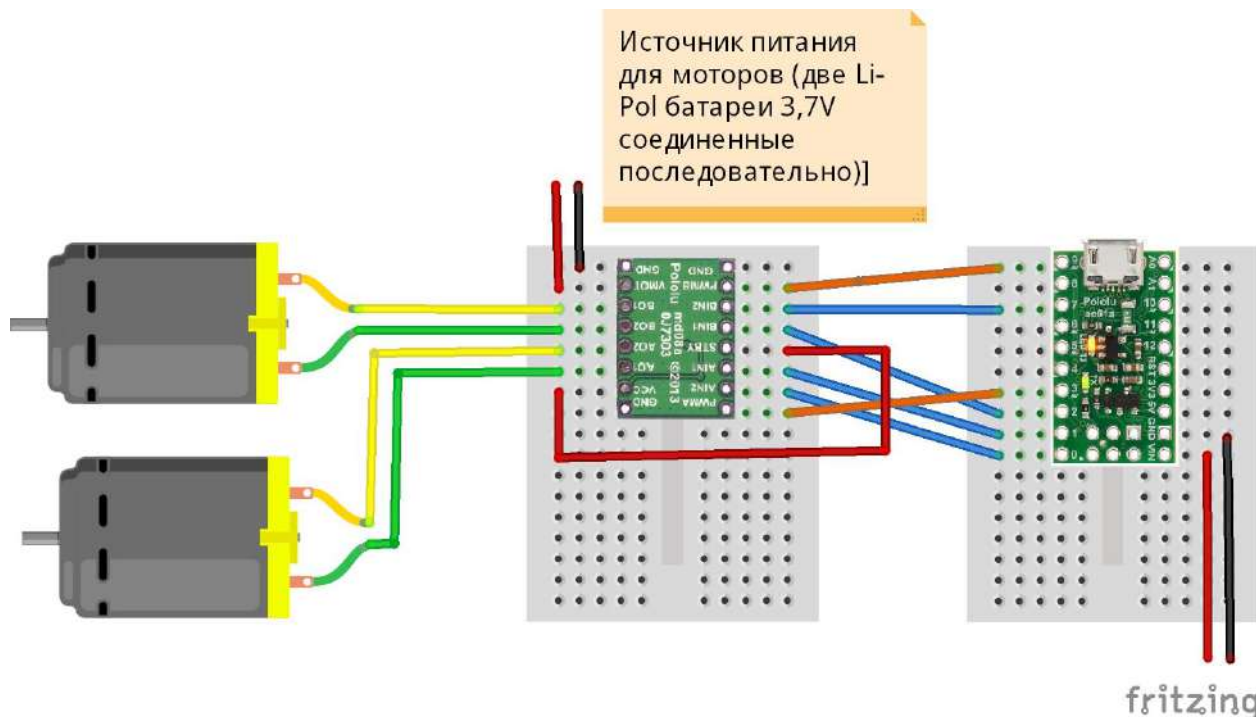
Real-world power dissipation considerations

The TB6612 motor driver used on the carrier board has a peak current rating of 3 A per channel. The peak ratings are for quick transients (e.g. when a motor is first turned on), and the continuous rating of 1 A is dependent on various conditions, such as the ambient temperature. The actual current you can deliver will depend on how well you can keep the motor driver cool. The carrier's printed circuit board is designed to draw heat out of the motor driver chip, but performance can be improved by adding a heat sink.

This product can get hot enough to burn you long before the chip overheats. Take care when handling this product and other components connected to it.

На данной картинке приведены основные характеристики драйвера моторов TB6612FNG и описание контактов для его подключения к микропроцессорному контроллеру и моторам. Итак, драйвер может обеспечить моторы напряжением питания от 4,5V до 13,5V. Обеспечить продолжительный ток до 1A на каждый из 2-х двигателей (нам надо 300-900mA) и пиковый до 3A (нам надо максимум 1,6A на каждый при полной блокировке двигателей). В таком варианте наши «мозги» будут целы, «ноги» сыты. Обратите внимание, что микросхема может нагреваться при нагрузках – всегда проверяйте ее температуру, если суморобот проводил долгий и сложный бой.

Ниже представлен вариант схемы подключения микропроцессорной платы AStar 32U4 и драйвера двигателей TB6612FNG с моторами Pololu HP:



Обратите внимание – мы подсоединяем к контактам PWMA и PWMB драйвера двигателей контакты микропроцессорной платы 3 и 9 соответственно, помеченные специальным значком «~». Этот значок говорит нам, что данные цифровые выходы микропроцессорной платы имеют возможность передавать PWM или ШИМ (шиотно-импульсно модулированный) сигнал. Не вдаваясь в **теоретизирование** просто скажем, что позже, в программе, мы сможем управлять скоростью вращения двигателей, используя ШИМ или PWM (pulse-width modulation).

Итак, соединили все компоненты, как указано на схеме – сейчас необходимо написать скетч, который бы смог крутить моторы в двух направлениях – прямом и обратном, с разными скоростями.

Для того, чтобы приступить к программированию, необходимо разобраться, как же подавать сигналы с контактов микропроцессорной платы №0, №1, №2, №3, №7, №9 на соответствующие входы драйвера двигателей PWMA, AIN1, AIN2 и PWMB, BIN1, BIN2 для того, чтобы моторы заработали.

Для решения этой задачи обратимся к рекомендации по использованию драйвера двигателей TB6612FNG. В ней на странице 4 мы найдем управляющую таблицу, с помощью которой сможем понять логику работы драйвера.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

В таблице указано, что при подаче на IN1 (AIN1 или BIN1) (столбец 1, IN1) логического нуля — сигнала «LOW» («L»), а на IN2 (AIN2 или BIN2 (столбец 2) логической единицы — сигнала «HIGH» («H»), и на контакт PWM (PWMA или PWMB) сигнала «HIGH» («H»), мотор будет крутиться против часовой стрелки («CCW»), а при изменении сигналов на контактах IN1 и IN2 наоборот будет крутиться по часовой стрелке («CW»). Скорость вращения моторов будет задаваться ШИМ (PWM) сигналом на контактах PWMA и PWMB.

Итак, давайте попробуем написать скетч, который будет управлять моторами.

Для начала вращения моторов против часовой стрелки нам необходимо написать следующие команды в скетче:

```
//для мотора, подключенного к контактам AO1 и //AO2
digitalWrite(1, LOW);
digitalWrite(0, HIGH);
analogWrite(3, 255);
```

```
delay(1000);
```

заставит его вращаться против часовой стрелки ровно одну секунду с максимальной скоростью, а исполнение команд

```
digitalWrite(1, HIGH);
```

```
digitalWrite(0, LOW);
```

```
analogWrite(3, 255);
```

```
delay(1000);
```

заставят его вращаться по часовой стрелке ровно одну секунду.

Кажется все.

Нет.

Не все.

Во первых, после вращения одну секунду что произойдет с мотором? Мотор продолжит крутиться или остановится? А мы ему дали команду круться или остановиться? Нет. Такие ситуации в программировании моторов нежелательны, так как в разной ситуации разные моторы могут вести себя совсем непредсказуемо – в одном случае он продолжит вращение, в другом просто остановится... Эта неопределенность поможет Вам безусловно увидеть в работе элементы собственного сознания и даже своенравия... J

Прекратим шутить и вернемся к вопросу.

Как остановить моторы? Смотрим в таблицу – чтобы остановить мотор необходимо подать на оба ввода драйвера логическую единицу – сигнал «HIGH» («H») и на PWM – любой сигнал.

```
digitalWrite(1, HIGH);
```

```
digitalWrite(0, HIGH);
```

```
analogWrite(3, LOW);
```

```
delay(10);
```

Теперь мотор остановится. Зачем нужна небольшая задержка — delay(10);? – для снятия инерции крутящихся моторов и более надежного торможения – мы живем в инерциальном мире. Чем больше задержка – тем сильнее торможение.

Для второго мотора, подключенного к выводам драйвера BO1 и BO2. все приведенные выше команды также справедливы при замене номеров контактов микропроцессорной платы, подключенной к вводам драйвера двигателей, отвечающим за этот мотор.

Обратим внимание на контакт драйвера двигателей STBY (standby). При подаче на него сигнала «HIGH» («H») все функции драйвера двигателя будут доступны, при сигнале «LOW» драйвер будет находиться в спящем режиме.

Для упрощения и с учетом того, что бои сумороботов проходят очень скоротечно мы сразу подключили данный контакт к +5V, питающим саму микросхему драйвера двигателей – подали постоянный сигнал «HIGH».

Как изменить скорость вращения двигателей?

Вот здесь нам и пригодится упомянутое выше свойство контактов микропроцессорной платы передавать PWM или ШИМ сигнал. Вспомним о том, что мы использовали контакты, которые имеют такое свойство – передавать PWM

сигнал. Как Вы уже заметили мы использовали команду `analogWrite(pin, value)`; для формирования ШИМ (PWM) сигнала.

В этой команде значение `value` может находиться в диапазоне от 0 до 255. При этом 0 в `analogWrite` равнозначен LOW в `digitalWrite`, 255 равнозначно HIGH.

Для первого мотора исполнение команд

```
digitalWrite(1, LOW);
```

```
digitalWrite(0, HIGH);
```

```
analogWrite(3, 125);
```

```
delay(1000);
```

будет соответствовать вращению мотора против часовой стрелки примерно с половинной скоростью от максимальной (без нагрузки). Обратите внимание, что вместо 255 в команде `analogWrite` также можно использовать системную константу HIGH или LOW интегрированной среды разработки Arduino.

Таким образом программа, которая вращает моторы вперед-назад с частотой 1 секунда и половинной скоростью, будет выглядеть следующим образом:

```
//Скетч 2
```

```
//крутим моторы вперед назад с частотой 1 //секунда с половинной скоростью
```

```
//переменные для моторов pololu через драйвер //TB6612FNG
```

```
#define PWMA 3
```

```
#define AIN1 1
```

```
#define AIN2 0
```

```
#define BIN1 2
```

```
#define BIN2 7
```

```
#define PWMB 9
```

```
void setup()
```

```
{
```

```
pinMode(PWMA,OUTPUT);
```

```
pinMode(AIN1,OUTPUT);
```

```
pinMode(AIN2,OUTPUT);
```

```
pinMode(PWMB,OUTPUT);
```

```
pinMode(BIN1,OUTPUT);
```

```
pinMode(BIN2,OUTPUT);
```

```
}
```

```
void loop()
```

```
{
```

```
//запускаем оба мотора против часовой стрелки
```

```
digitalWrite(AIN1, LOW);
```

```
digitalWrite(AIN2, HIGH);
```

```

analogWrite(PWMA, 125);
digitalWrite(BIN1, LOW);
digitalWrite(BIN2, HIGH);
analogWrite(PWMB, 125);
delay(1000);
//останавливаем моторы
digitalWrite(AIN1,HIGH);
digitalWrite(AIN2,HIGH);
digitalWrite(PWMA,LOW);
digitalWrite(BIN1,HIGH);
digitalWrite(BIN2,HIGH);
digitalWrite(PWMB,LOW);
delay(10);
//запускаем оба мотора в другую сторону
digitalWrite(AIN1, HIGH);
digitalWrite(AIN2, LOW);
analogWrite(PWMA, 125);
digitalWrite(BIN1, HIGH);
digitalWrite(BIN2, LOW);
analogWrite(PWMB, 125);
delay(1000);
}

```

Что в этом скетче появилось нового для нас:

Директивы *#define* – они назначают константам AIN1, AIN2, BIN1, BIN2, PWMA, PWMB (которые мы можем затем использовать в программе в качестве замены номеров контактов микропроцессорной платы) номера соответствующих выводов микропроцессорной платы.

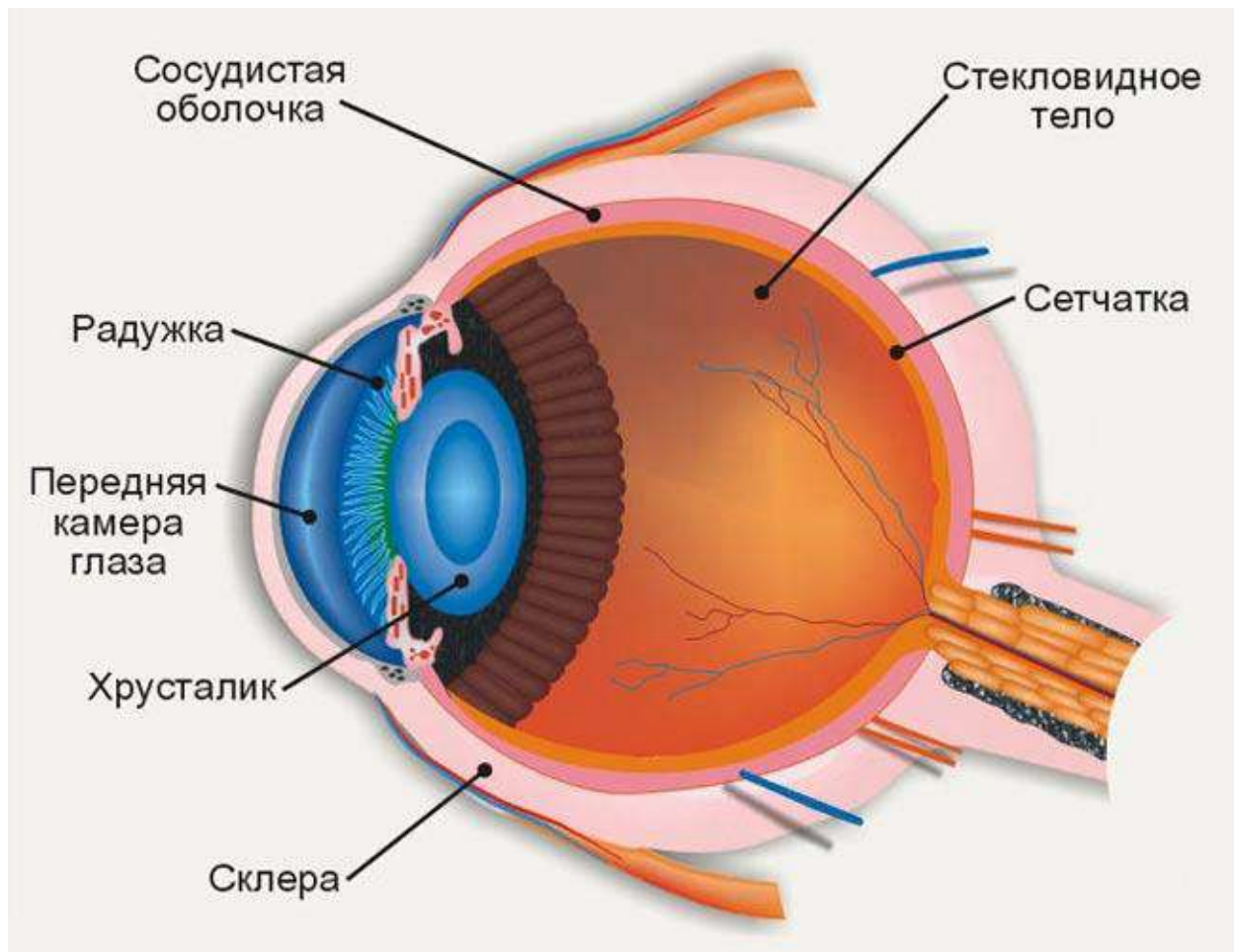
В процедуре *setup()* появились функции *pinMode(pin, state)*, где *pin* – номер соответствующего контакта микропроцессорной платы и *state* имеет значение либо OUTPUT (вывод) – мы управляем драйвером, либо INPUT (ввод данных – пригодится нам при получении информации с датчиков в следующих темах).

Пример работы скетча моторов

Вопросы для проверки:

1. Зачем нужен драйвер двигателей?
2. В чем отличие команд digitalWrite от analogWrite
3. Как изменить скорость вращения моторов нашего робота?
4. Подумайте, в какую сторону должны вращаться моторы робота, расположенные по его правой и левой стороне для того чтобы робот а) ехал вперед, б) назад, в) поворачивал направо, г) поворачивал налево.*

Урок 4. «Зрение» робота. Сенсоры расстояния.



Зрение как свойство глаз различать предметы и мозга распознавать это различие и заставить по разному реагировать наши органы движения в зависимости от того, тигра мы увидели или кошку – важнейшее приобретение эволюции организмов (по Дарвину) или божественной сущности (религиозный подход), проявляющейся в человеке.

Человек конструирует роботов по своему образу и подобию («мозг» и «ноги» робота мы уже рассмотрели и научились ими управлять) –теперь очередь за зрением.

Безусловно, когда мы говорим о зрении, в качестве сенсора(датчика, органа) зрения, человек подразумевает камеру, передающую изображение. Для обработки роботом полученного с камеры изображения сейчас существует достаточно много специализированных решений и программных библиотек. Но для практической реализации этих решений требуются значительные вычислительные ресурсы и математический аппарат, реализованный в программных средствах.

Еще раз оценим – готова ли наша миниатюрная микропроцессорная плата AStar 32U4 справиться с этой сложной задачей, а мы с Вами потратить годы на изучение и создание матаппарата и программирование с отладкой? Вероятно пока нет.

Как же быть в этом случае? Как нам найти соперника на ринге?

Давайте оценим нашу задачу и выберем оптимальные методы ее решения. Ринг представляет собой плоский круг черного цвета размером 77 сантиметров в диаметре с белой полосой по краю, за которую робот не должен заезжать.

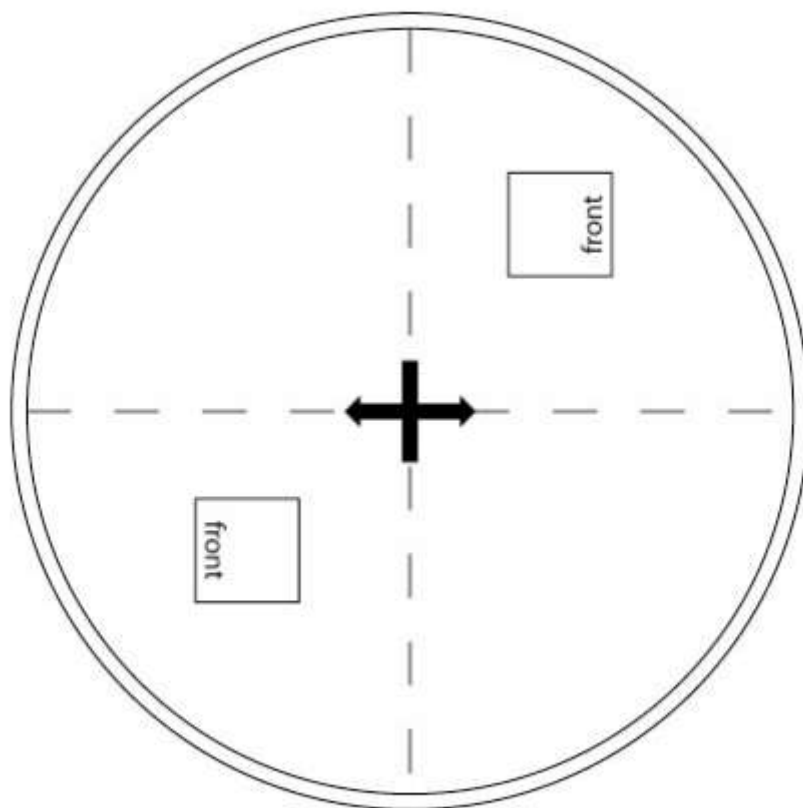


Рисунок 1: Размещение роботов

Задачей робота является найти на этом ринге соперника и вытолкнуть его с ринга, используя Самую Хитрую Стратегию в Мире (если робот не быстр и силен) или грубую физическую силу (если робот Самый Сильный и Быстрый).

Таким образом, видим, что Мир Робота ограничен и его задача проста (по сравнению с нашим миром и нашей целью безусловно). Видимо и зрение ему необходимо более простое.

От философских вопросов бытия вернемся к практическим.

Для реализации функции поиска соперника на ринге отлично зарекомендовали себя инфракрасные сенсоры и чуть хуже ультразвуковые сенсоры препятствий.

С учетом того, что размеры ринга меньше метра датчики должны покрывать соответствующие дистанции.

Чаще всего в реальных роботах, побеждающих на соревнованиях, используются ИК датчики Sharp (www.pololu.com):



ИК датчик Sharp GP2Y0A21SK0F (10см-80см) или визуально похожий на него GP2Y0A41SK0F (4см-30см). Эти датчики рассчитывают расстояние до объекта перед ними.



ИК датчики расстояния Sharp GP2Y0D805, GP2Y0D810, GP2Y0D815 на 5, 10 и 15 см соответственно. Эти датчики просто фиксируют, что объект появился не дальше 5, 10, и 15 сантиметров соответственно (без расчета реального расстояния до объекта).



Более современными датчиками являются GP2Y0A60SZ, которые рассчитывают расстояние от 10 до 150 сантиметров (два варианта – с подключением на 5V и 3V соответственно), но эти датчики почти в два раза дороже, недостаточно «близоруки» и чересчур «дальнозорки».

Ультразвуковые сенсоры расстояния это обычно восточные аналоги Parallax PING (HC-SR04, 05 и другие)

Они обычно рассчитаны на получение данных на расстоянии 4см-3 метра.



Несмотря на свои казалось бы отличные характеристики по расстоянию эти датчики имеют и свои недостатки, связанные с физическими свойствами измерения расстояния с помощью звука.

Давайте сравним.

ИК датчик для измерения расстояния использует инфракрасный свет – то есть это световой поток, невидимый для человеческого глаза. Свет имеет гораздо

меньший чем звуковая волна коэффициент рассеивания в воздухе да и еще на столь малых расстояниях, которые мы рассматриваем. Кроме того, свет распространяется со скоростью света, простите за тавтологию. Со ссылкой на авторитет Альберта Эйнштейна скажу, что это самая высокая скорость в нашем мире.

Таким образом, например датчик GP2Y0A21SK0F имеет возможность примерно за 38-48 миллисекунд ([смотрим документацию](#)), тратя всего 30mA тока измерить достаточно точное расстояние до робота-соперника.

Ультразвуковой датчик (еще его называют сонар) использует свойство звука распространяться в различных средах (в частности в воздухе) и отражаться от предметов, а по времени, необходимому на преодоление расстояния до предмета и от него, рассчитать расстояние. Но для реального использования сонаров есть несколько «но».

Первое «но» — звуковая волна менее точно сфокусирована по сравнению со световой, второе — скорость звука в воздухе «всего» 331 м/с и поэтому сонар более «медленный» с точки зрения измерений, третье — сонар более крупный, что увеличивает робота в размерах а значит повышает шанс его обнаружить, четвертое — сонар обычно требует 2 контакта микропроцессорной платы для управления в отличие от ИК, который обходится одним линком с микропроцессорной платой. Пятое — так как роботы при перемещении качаются, переваливаются, сонар чаще «зацепляет» поверхность ринга и может принимать ее за соперника. Вот поэтому сонары менее часто распространены. Для [информации ссылка на документацию](#).

Итак, мы выбираем для нашего робота ИК датчики Sharp GP2Y0A60SZLF в качестве органов «зрения».

Вопрос — сколько таких датчиков выбирать? — также очень важен.

У человека два глаза. Благодаря этому факту мы имеем стереозрение, позволяющее нам определять расстояние и запас, если один глаз откажет... 😊

Многие говорят что имеют «третий глаз» позволяющий им избегать неприятностей... 😊

На практике сумороботостроители применяют 1-6 датчиков — «глаз» робота. Очень часто — два спереди, и по одному справа и слева. Выбор конструкции всегда за Вами.

Что необходимо обязательно учитывать?

Измерения расстояния одним из наиболее распространенных датчиков GP2Y0A21SK0F (10-80см), занимают примерно 50 миллисекунд. Выбранный нами датчик измеряет расстояние примерно за 25 миллисекунд. Часто роботостроители измеряют расстояние одним датчиком несколько раз для того, чтобы надежно быть уверенным, что найден противник, а не просто закралась ошибка измерения. Посчитаем, что если робот использует 2 последовательных измерения для датчика, то ему надо примерно 100 миллисекунд. Много это или мало? Для сравнения — реальный бой роботов обычно длится 2 секунды. Десять измерений всеми датчиками — глазами....

Еще раз повторимся, что выбор оптимального количества «глаз» всегда за робоспортсменом-конструктором робота. Кроме того, никто не отменял механизмы оптимизации программ — когда вы можете «закрывать» некоторые глаза на время (не измерять ими расстояния). Например — передние «глаза» надежно «видят»

соперника и боковые датчики на время не используются для измерений – в Мире Робота всего один соперник на ринге во время боя, к счастью... J

Мы с Вами выберем 3 датчика – по одному спереди, справа и слева.

Перейдем к вопросу, как измерять расстояние до соперника? Для этого давайте изучим характеристики наших «глаз».

Обращаемся к **документации** датчика GP2Y0A60SZLF.

Feature summary

- Operating voltage:
 - 5V version: 2.7 V to 5.5 V
 - 3V version: 2.7 V to 3.6 V
- Average current consumption: 33 mA (typical)
- Distance measuring range: 10 cm to 150 cm (4" to 60")
- Output type: analog voltage
- Output voltage differential over distance range:
 - 5V version: 3.0 V (typical)
 - 3V version: 1.6 V (typical)
- Update period: 16.5 ± 4 ms
- Enable pin can optionally be used to disable the emitter and save power
- Size without header pins: 33 mm × 10.4 mm × 10.2 mm (1.3" × 0.41" × 0.4")
- Weight without header pins: 2.5 g (0.09 oz)



5V version



0 Ω
resistor

3V version



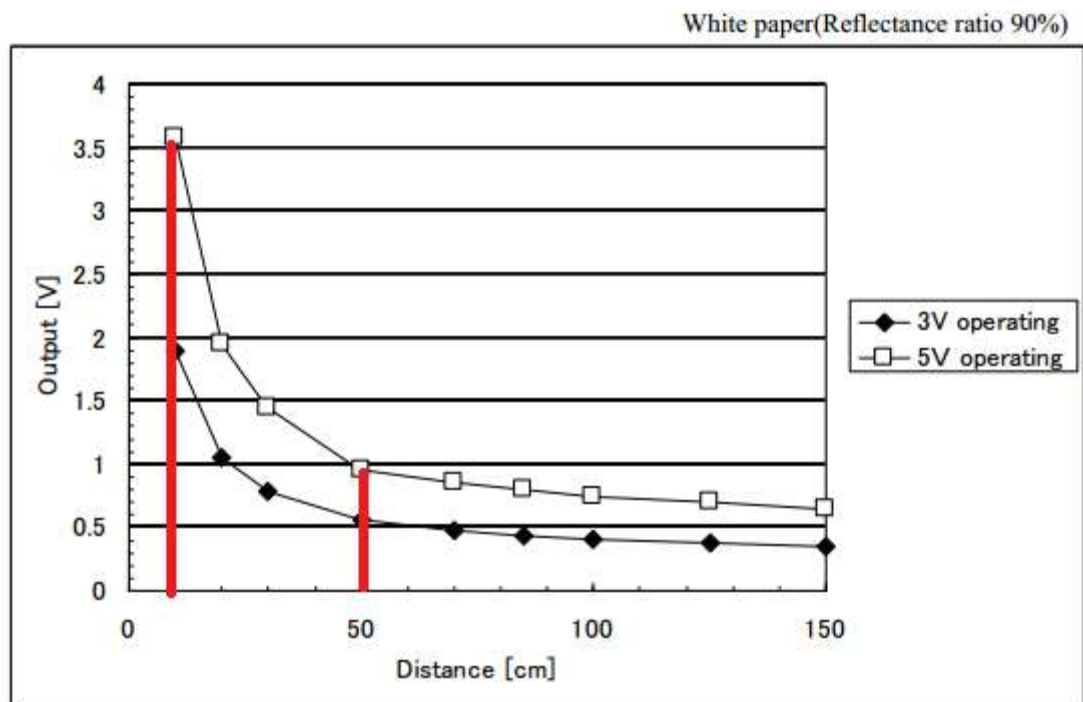
not popl

Обратите внимание на подчеркнутые красным параметры:

1. Датчик аналоговый – т.е. мы с Вами можем на аналоговом контакте микропроцессорной платы получить число от 0 до 1024, которое будет характеризовать расстояние по определенной зависимости, которую мы с Вами рассмотрим чуть позже.
2. Потребление тока датчиком составляет около 33mA – учтем это в нашем бюджете тока.
3. Напряжение питания – 2.7V до 5.5V (рекомендуемое от 4.5V до 5.5V). Обязательно учтите, что это очень важный параметр для «глаз» Вашего робота. Стабильное напряжение без просадок и пиков – очень важно для точных измерений расстояний датчиком. Для примера приведем опыт из человеческого спорта. Предположим спортсмен бежит дистанцию на время. В начале дистанции если он не сильно волнуется, его давление крови и пульс в норме. В конце дистанции обычно его пульс значительно повышен, давление также и обычно глаза «затуманены», угол зрения значительно снижен... Также и у робота – если его «пульс» и «давление» (а это примерные аналоги напряжения и потребляемого тока) мы будем держать в заданном диапазоне, а еще лучше абсолютно стабильными, то и измерения дистанции его «глазами» — датчиками будут абсолютно стабильны.

Давайте изучим зависимость между аналоговыми значениями и дистанцией до измеряемого объекта.

На пятой странице документации сенсора мы видим следующий график зависимости напряжения на аналоговом выходе датчика от расстояния до измеряемого объекта:



Обратите внимание – производитель говорит, что его датчик измеряет расстояние в диапазоне от 10 см до 150 сантиметров.

Кроме того, давайте обратим внимание на график после правой красной черты (после 50 сантиметров) – как мы видим, напряжение на аналоговом выходе будет меняться не слишком значительно при значительном изменении расстояния. Предположим разница в напряжении между измеренным расстоянием в 60 сантиметров и 70 сантиметров составляет менее 100mV.

С учетом рассмотренных особенностей данного сенсора сделаем следующие выводы для реальных условий боя между минисумороботами с использованием данных сенсоров:

1. Надежно измерить расстояние можно в диапазоне от 10 до 50 сантиметров – этого вполне достаточно для боя минисумороботов.
2. Для стабилизации напряжения на датчике расстояния необходимо применять специальные мероприятия (рассмотрим чуть позже в уроке 6).

Теперь перейдем к скетчу.

Итак, сигнал мы получаем аналоговый, поэтому будем применять функцию `analogRead(pin);`

Эта функция конвертирует напряжение от 0V до 5V на контакте (pin) в числовое значение от 0 до 1023 или примерно 0.0049V на единицу. Таким образом если сенсор измерит расстояние 50 сантиметров до соперника, то на аналоговом выводе сенсора, соединенном с контактом микропроцессорной платы, появится напряжение (смотрим по рисунку) 1.0V. А функция `analogRead(номер контакта)` вернет нам значение $1/0.0049 \sim 204$.

Для подключения 3-х сенсоров расстояния будем использовать аналоговые контакты микропроцессорной платы A0, A1(они помечены на самой плате) и аналоговый порт A8(он находится на контакте 8 микропроцессорной платы AStar32U4).

Итак, наш скетч для «прозрения» нашего суморобота выглядит следующим образом:

```

//объявляем переменные, содержащие значения //аналоговых портов
int front_eye = A0;
int left_eye = A1;
int right_eye = A8;
void setup(){
//инициализируем терминальный вывод
Serial.begin(9600);
}
void loop(){
//выводим данные о значениях расстояний в окно
//терминала в одну строку
Serial.print(«front_distance = «);
Serial.print(analogRead(front_eye));
Serial.print(» left_distance = «);
Serial.print(analogRead(left_eye));
Serial.print(» right_distance = «);
Serial.println(analogRead(right_eye));
//немного задерживаем вывод данных, чтобы было //удобнее смотреть
delay(500);
}

```

Какие новые функции мы видим в данном скетче?

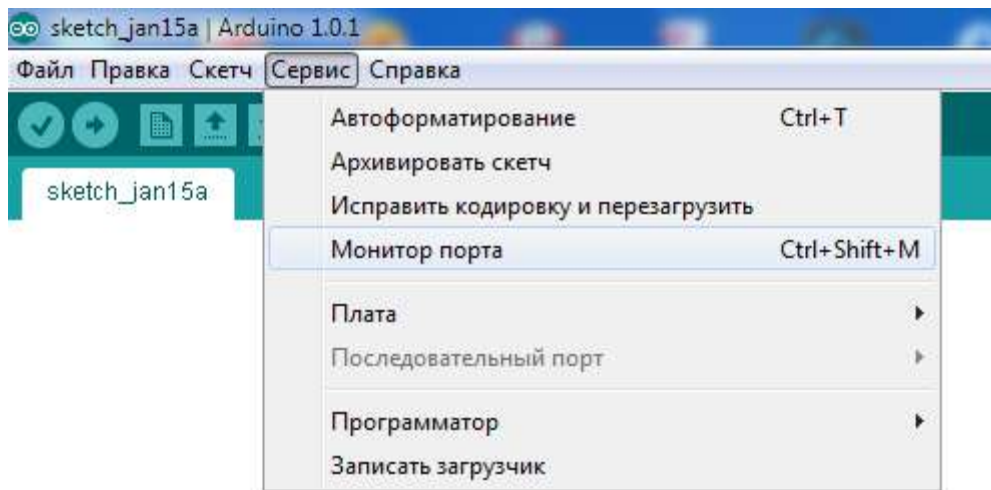
Функции работы с портом, к которому подключена микропроцессорная плата.

`Serial.begin(“скорость порта”);` — инициализация порта для обмена данными между микропроцессорной платой и компьютером.

`Serial.print(“что-то”);` — вывод в этот порт данных какой-то строки, заключенной в кавычках или значения функции `analogRead(номер контакта)`.

`Serial.println(“что-то”);` — вывод строки с переходом на новую строку.

Загружаем скетч в микропроцессорную плату, затем, не отключаем кабель от платы, открываем окно терминала:



Затем в окне терминала мы видим результат работы скетча:

```
COM23

front_distance = 94 left_distance = 181 right_distance = 218
front_distance = 95 left_distance = 128 right_distance = 269
front_distance = 93 left_distance = 107 right_distance = 264
front_distance = 95 left_distance = 86 right_distance = 306
front_distance = 101 left_distance = 67 right_distance = 330
front_distance = 182 left_distance = 42 right_distance = 336
front_distance = 182 left_distance = 47 right_distance = 332
front_distance = 323 left_distance = 26 right_distance = 329
front_distance = 721 left_distance = 58 right_distance = 322
front_distance = 422 left_distance = 47 right_distance = 319
front_distance = 311 left_distance = 174 right_distance = 323
front_distance = 112 left_distance = 172 right_distance = 299
front_distance = 114 left_distance = 149 right_distance = 266
front_distance = 115 left_distance = 139 right_distance = 268
front_distance = 115 left_distance = 125 right_distance = 266
front_distance = 126 left_distance = 110 right_distance = 269
front_distance = 115 left_distance = 93 right_distance = 278
front_distance = 116 left_distance = 81 right_distance = 255
front_distance = 117 left_distance = 59 right_distance = 239
```

Для человека не очень удобно переводить полученное значение в вольты, а затем лезть каждый раз документацию и сопоставлять вольты с расстоянием – работу же все равно и проще составить таблицу соответствия расстояния измеренному значению функцией `analogRead()`.

Таблица для датчика GP2Y0A60SZLF и плоской поверхности, напечатанной на 3D принтере из ABC пластика.

Аналоговое значение	Расстояние (см)
790	5
816	6
793	7

748	8
701	9
640	10
546	12
464	15
380	20
355	25
333	30

Обратите внимание, что различные поверхности под разными углами имеют различную отражательную способность, что будет вносить коррективы в реальную зависимость расстояния от аналогового измерения датчика.

Еще одно замечание по таблице – датчик GP2Y0A60SZLF рассчитан для измерения расстояния от 10 до 150 см. Мы с Вами попробовали измерить меньшие расстояния, и это получилось – отметим, что на расстоянии 5 см и 7 см показания датчика примерно равны. Значит эффективно померить расстояние можно примерно до 6 см.

Почему нам лучше не переводить аналоговое значение в расстояние? Во время боя все ресурсы робота надо направлять только для достижения победы и потеря времени для перевода значения функции `analogRead()` в расстояние в сантиметрах может вам вылиться в поражение от соперника, который оптимизирует вычислительные ресурсы. Более подробно о путях совершенствования спортивного искусства поговорим на 9 уроке.

Вопросы для самопроверки:

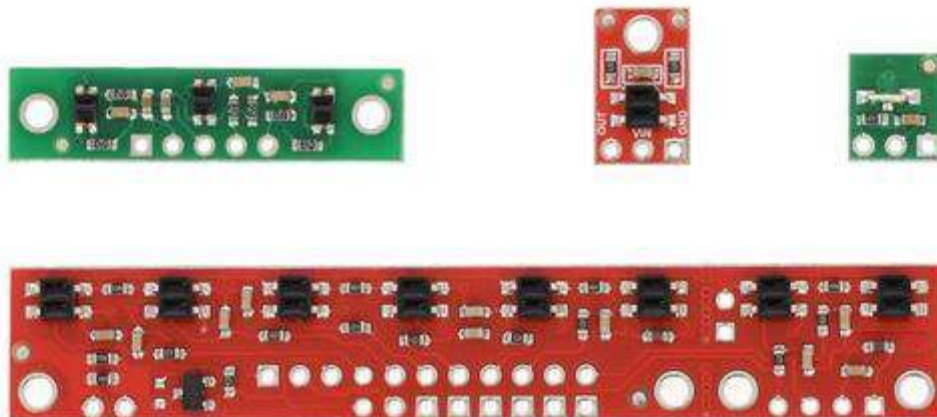
1. Инфракрасный свет против ультразвука – поразмышляйте сами – что оптимальнее использовать для «зрения» робота.
2. Возвращаемое значение функции `analogRead(pin)`; — расскажите подробнее о связи между значением этой функции, напряжением и расстоянием в сантиметрах.
3. Подумайте, какие еще технологии, которые могут заменить «глаза» сумороботу Вы можете предложить использовать?
4. Какие решения можно применить для того, чтобы спрятаться от «зрения» соперника в пределах правил боев роботов Robotchallenge.org?

Урок 5. «Зрение» робота. Сенсоры черного и белого цвета ринга для боев сумороботов.



Мы с Вами разобрались, как суморобот сможет найти соперника в бою. Но это только часть «зрения», необходимая для победы. Вторая часть «зрения» должна быть направлена «под ноги» суморобота – по правилам поединка робот, выехавший или вытолкнутый за пределы ринга – проиграл. Как вы заметили ранее – в материалах урока №4, ринг черный, а по краю ринга проходит белая полоса. Таким ринг сделан именно для того, чтобы суморобот смог различить край (границу между черным рингом и белой полосой по краю) боевого поля и совершить действия, препятствующие выходу за эту границу.

Для различения такой границы нам необходим специализированный датчик. Существует достаточно большое количество производителей таких датчиков - для поиска их в интернете надо ввести «датчик линии», «датчик уровня серого», «датчик цвета». Мы остановимся на наиболее распространенных, компактных и недорогих, представленных на сайте www.pololu.com (или на сайтах www.amperka.ru, www.electronshtik.ru)



QTR sensor size comparison. Clockwise from top left: QTR-3RC, QTR-1RC, QTR-L-1RC, QTR-8RC.

Как видите, датчики представлены в различных исполнениях – по 8, 3 и по 1 сенсору. 8 и 3 сенсора на одной плате обычно используются для проектов роботов, следующих по линии. Для проектов сумороботов наиболее удобны датчики QTR-1A и QTR-1RC



QTR-1A возвращает аналоговый сигнал, QTR-1RC – цифровой.

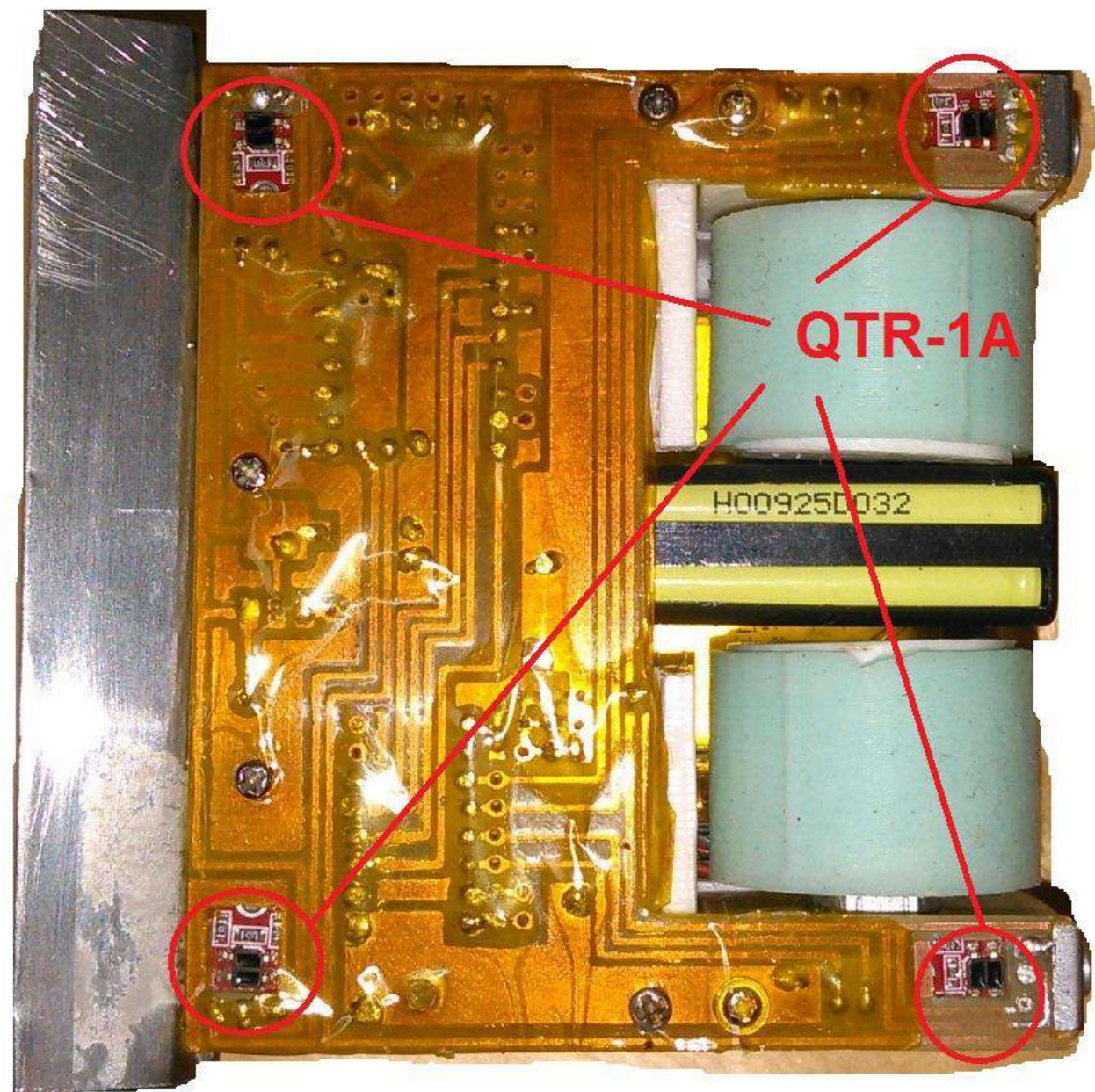
Сколько таких датчиков нам необходимо для суморобота и как их располагать? Оптимально расположить датчики максимально близко к краю по углам робота ПЕРЕД колесами для того, чтобы он мог распознать край и вовремя отъехать от него и колесо не висело в воздухе, за границей ринга – это снижает устойчивость робота и повышает шансы проигрыша.

Сколько таких датчиков? Минимально один спереди по центру. В этом случае движение робота по рингу должно происходить только передним ходом – чтобы различить границу и не вылететь с ринга. Оптимально – 4 сенсора по краям робота или 3 (два сенсора спереди по краям и один сзади по центру) – такой робот будет иметь больше возможностей в тактическом превосходстве – сможет перемещаться в любые стороны.

В нашем роботе с двумя колесами мы будем использовать два датчика QTR-1A черного/белого, расположенные спереди.

Для примера приведены примеры компоновки датчиков — изображения снизу реальных роботов:

Робот с четырьмя датчиками QTR-1A по углам робота.



Робот с двумя датчиками QTR-1A спереди (робот гусеничный и имеет хорошую центровку массы, тактика движения по рингу только вперед).



Как прочесть данные с датчиков линии?

Как всегда обратимся к [документации](#) по выбранному датчику QTR-1A:

Specifications

- Dimensions: 0.3" x 0.5" x 0.1" (without optional header pins installed)
- Operating voltage: 5.0 V
- Supply current: 17 mA
- Output format: analog voltage
- Output voltage range: 0 to supplied voltage
- Optimal sensing distance: 0.125" 3 mm
- Maximum recommended sensing distance: 0.25" 6 mm



Итак, отметим, что датчик питается от напряжения 5V и потребляет примерно 17mA и возвращает аналоговое значение напряжения в зависимости от цвета

поверхности. Его надо располагать оптимально на расстоянии 3мм от поверхности и не более 6мм от ринга.

Для этого датчика также написана **специальная библиотека**, которую можно использовать в Arduino проектах.

Подключим контакты OUT наших датчиков QTR-1A с аналоговыми контактами A6 и A11 (контакты 4 и 12 микропроцессорной платы A-Star 32U4), контакты VIN к напряжению питания 5V, контакты GND к общей шине (земле).

Для чтения данных с датчиков будем использовать обычную функцию чтения аналогового сигнала `analogRead(pin)` как в предыдущем уроке. Помним, что эта функция возвращает значения от 0 до 1023.

Скетч будет максимально похож на программу предыдущего урока – ведь это практически такое-же «зрение» — просто направленное вниз. J

Итак:

//объявляем переменные, содержащие значения //аналоговых портов, подключенных к датчикам //QTR-1A

```
int left_bw = A6;
```

```
int right_bw = A11;
```

```
void setup(){
```

```
//инициализируем терминальный вывод
```

```
Serial.begin(9600);
```

```
}
```

```
void loop(){
```

```
//выводим данные о значениях датчиков в окно //терминала в одну строку
```

```
Serial.print("left side color = ");
```

```
Serial.print(analogRead(left_bw));
```

```
Serial.print("      right side color = ");
```

```
Serial.println(analogRead(right_bw));
```

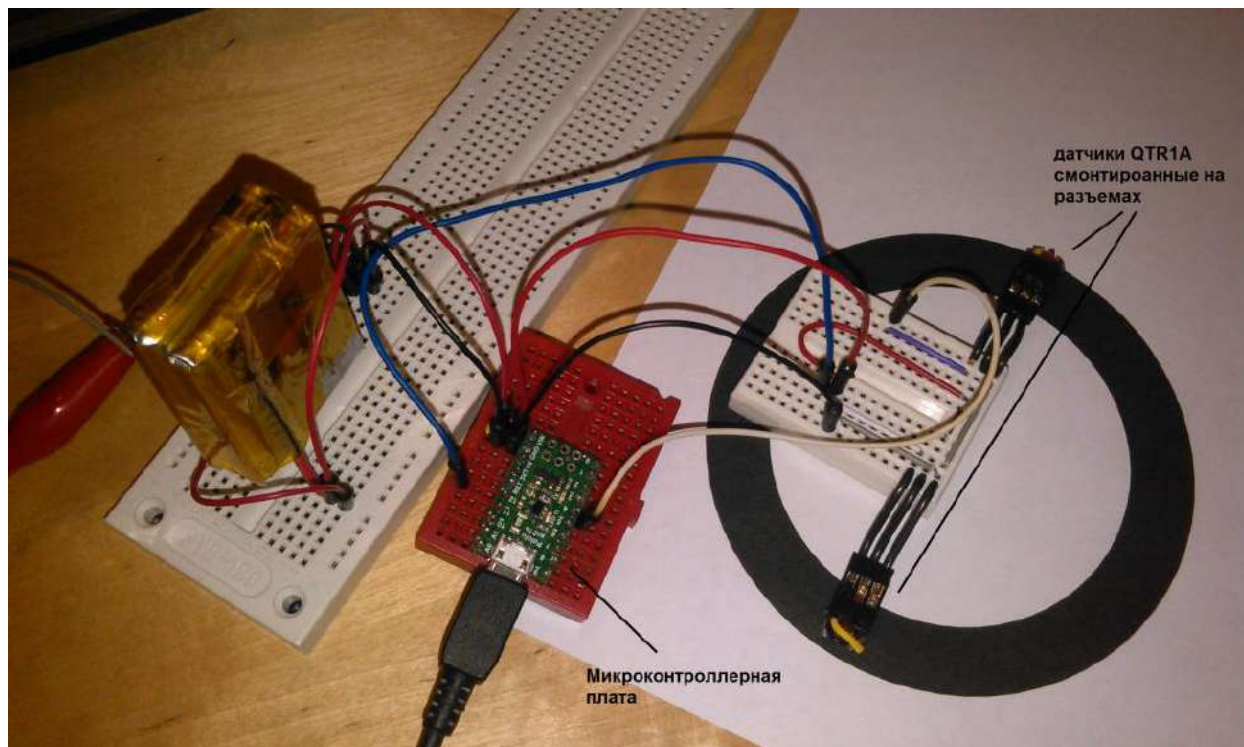
```
//немного задерживаем вывод данных, чтобы было //удобнее смотреть
```

```
delay(500);
```

```
}
```

Предполагаю, что в данном скетче Вам уже все понятно и объяснения будут лишними.

Загрузим скетч в микропроцессорную плату и также запустим монитор порта для представления данных. Теперь датчики станем передвигать так, чтобы они попеременно оказывались над белой и черной частями листа.



Экран терминала:

```
COM23

left side color = 42    right side color = 36
left side color = 43    right side color = 34
left side color = 39    right side color = 35
left side color = 41    right side color = 35
left side color = 41    right side color = 34
left side color = 42    right side color = 35
left side color = 860    right side color = 706
left side color = 861    right side color = 702
left side color = 862    right side color = 704
left side color = 857    right side color = 707
left side color = 43    right side color = 617
left side color = 40    right side color = 37
left side color = 41    right side color = 35
left side color = 788    right side color = 662
left side color = 619    right side color = 33
left side color = 800    right side color = 32
left side color = 836    right side color = 28
left side color = 845    right side color = 27
left side color = 851    right side color = 28
left side color = 851    right side color = 26
left side color = 845    right side color = 28
left side color = 863    right side color = 36
left side color = 60    right side color = 725
left side color = 51    right side color = 42
left side color = 44    right side color = 680
left side color = 42    right side color = 697
left side color = 41    right side color = 702
left side color = 42    right side color = 701
```

На экране терминала мы видим, что когда датчики попадают на белую поверхность листа значения становятся ближе к нулю и наоборот – при попадании датчиков на черную поверхность листа значения ближе к 1024.

Видео с примером работы скетча

Предлагаю воспользоваться этим свойством датчика QTR-1A и для различения границы ринга сформулировать следующее условие с известным допущением:

«Если возвращаемое значение функции `analogRead(pin)` меньше значения 400 считаем поле под датчиком QTR-1A условно белым (светлая поверхность), если больше 400, то условно черным (темная поверхность)».

Модернизируем скетч:

//объявляем переменные, содержащие значения //аналоговых портов, подключенных к датчикам //QTR-1A

```
int left_bw = A6;
int right_bw = A11;
void setup(){
  //инициализируем терминальный вывод
  Serial.begin(9600);
}
void loop(){
  if(analogRead(left_bw) < 400) {
    Serial.print("under left sensor – light surface");
  }
  else {
    Serial.print("under left sensor – dark surface ");
  }
  if(analogRead(right_bw) < 400) {
    Serial.print(" under right sensor – light surface");
  }
  else {
    Serial.print(" under right sensor – dark surface");
  }
  Serial.println(" ");
  delay(500);
}
```

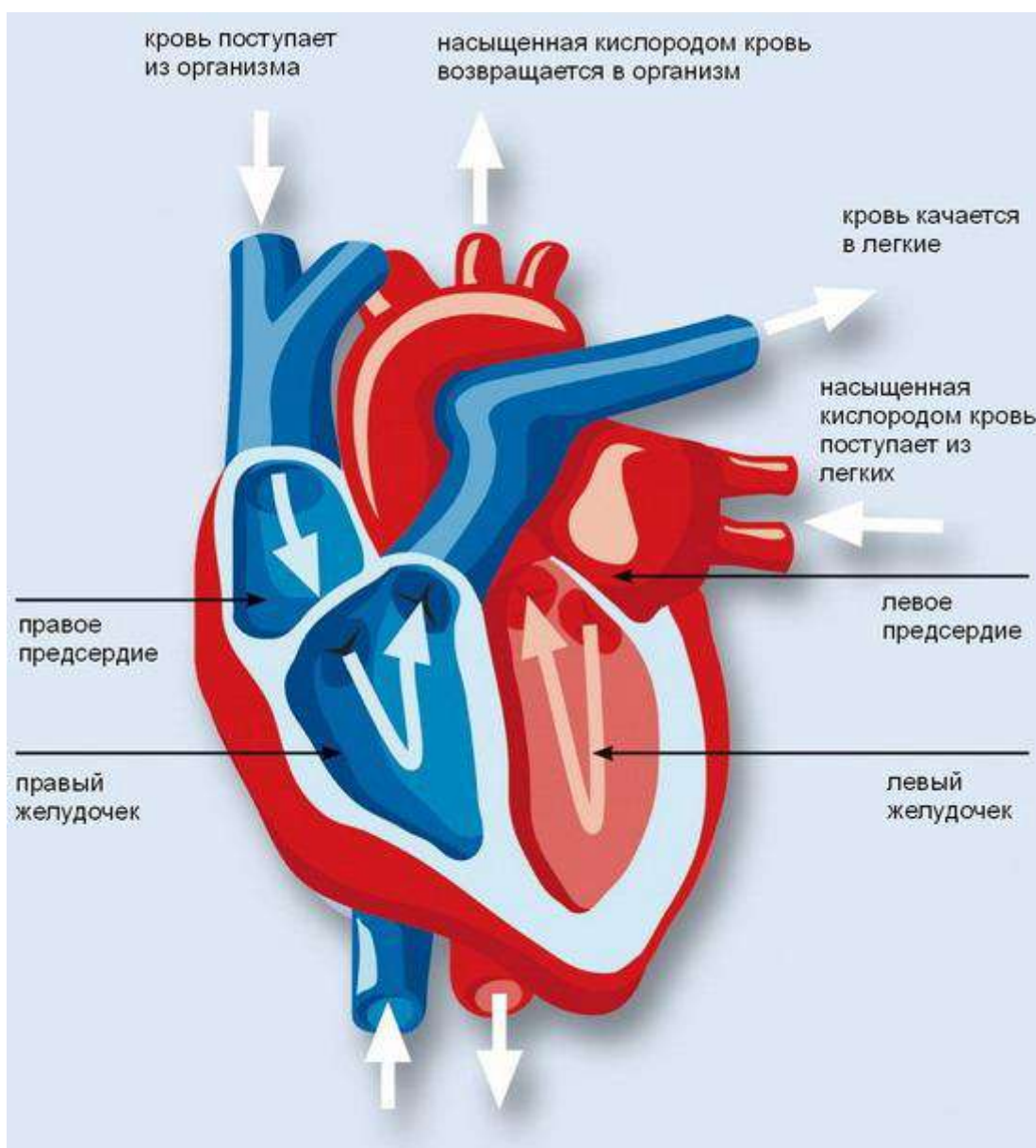
Загрузите скетч в микропроцессорную плату и попробуйте перемещать датчики над темным и светлым полями.

Обратим внимание на новую управляющую конструкцию условия `if(условие верно){действия если условие верно}else{действия если условие неверно}`.

Вопросы для самопроверки:

1. В чем основное отличие ИК – датчиков дистанции от ИК – датчиков линии?
2. Подумайте, какое количество датчиков линии оптимально для круглого суморобота (такие также бывают)?*

Урок 6. Правильное «питание» – основа правильного поведения робота. Выбор аккумуляторов. Контроллер питания DC-DC.



Наш суморобот становится все сложнее – он получил «мозг», «сильные ноги», «зрение» для поиска соперника и для различения границы ринга.

На протяжении всех предыдущих уроков мы всегда рассматривали электрические вопросы - напряжения и потребления тока микропроцессорной платой, моторами, и датчиками.

Полагаю, пришло время более детально разобраться в этом не слишком погружаясь в теоретические вопросы электричества.

Вы не поверите, но человечество повсеместно использует электричество уже более 100 лет, но многие теоретические вопросы не нашли единого толкования до сих пор – природа образования электрического потенциала, скорость его передачи по различным проводникам и многое другое....

Мы постараемся, также как человечество, не теоретизировать, а сразу успешно перейти к практическим вопросам использования.

Тем не менее, для лучшего понимания, давайте разберемся, что такое напряжение источника питания. Если попытаться сделать сравнение с понятными нам явлениями, то напряжение можно представить как разность высоты между началом реки (плюсом источника питания) и местом ее выхода в море (минусом источника питания). Уровень высоты моря, как вы помните из школьного курса географии, равен 0 метров. А вот от высоты начала реки зависит и потенциал самой реки. Чем выше начало реки, тем они быстрее и более бурные. Сравним горные реки и реки, текущие на равнине. Так и у источника питания – чем выше напряжение, тем «сильнее» потенциал источника питания.

Теперь давайте разбираться с силой тока. Если постараться представить понятие силы тока, то оптимальным для понимания будет водяной поток реки и понятие ширины этого водяного потока. Если для питания датчиков и микропроцессорной платы нам нужны «ручейки воды» от 17mA (датчик QTR-1A) до 30mA (датчик дистанции), то для моторов нам нужны «широкие потоки воды» в объеме 300mA-900mA и «разливы воды при половодье» до 1600mA, если движение колес будет заблокировано без отключения на них напряжения.

Поэтому важно также понятие «бюджета» по силе тока – «ширине и объему потока воды» который будет бежать по нашим «рекам и ручейкам» — проводам до каждого устройства.

Также верно провести аналогию по потреблению тока каждым устройством как необходимостью получить именно этот «объем воды» в каждую единицу времени. И если устройство не сможет получить объем воды в результате «узких ручейков» — проводов или «недостатка воды в реке», то оно будет работать неправильно или по-простому «сбоить».

Итак какой бюджет по току («какой суммарный объем воды в реке в единицу времени») необходим для питания наших электронных компонентов?

Таблица для источника тока на 5 вольт:

Устройство	Кол-во	Потребление	Суммарно
A-Star 32U4	1	25mA*	25mA
GP2Y0A21SK0F	3	30mA	90mA
QTR-1A	2	17mA	34mA
Итого			149mA

Обычно желательно увеличивать в два раза требуемый бюджет по току для учета разряда источника питания, нелинейности и пиков потребления в различных режимах. Таким образом, для электронных компонентов, желательно иметь источник питания напряжением 5 вольт с силой тока 250-300mA.

Для моторов необходимо оценить пиковые нагрузки при выталкивании соперника с ринга – как мы помним, они примерно 600-900mA и умножить их на количество двигателей – у нас их 2. Итого получим требуемый источник питания с напряжением около 6 вольт и силой тока разряда 1200-1800mA. Обычно литий-полимерные элементы могут выдержать ток разряда до 3 номиналов. Также существуют быстроразрядные литий-полимерные аккумуляторы – у них ток разряда может доходить и до 10 номиналов! Например, поставив литий-полимерную батарею номиналом 1000mA, вы сможете обеспечить максимальный ток разряда в 3000mA.

Итак, для наших моторов, по нашим подсчетам, оптимальным будет литий-полимерный аккумулятор с номиналом по току на 600mA – 1200mA (1800mA — 3600mA на максимальном токе разряда). Если поставить источник с худшими параметрами, то при выталкивании моторы не смогут развивать достаточную максимальную мощность.

Какие еще параметры (кроме напряжения и силы тока), важны при выборе батарей для роботов сумо?

Размер и вес – оптимальность их сочетания в зависимости от конфигурации робота и расположения всех его компонентов.

Удобство при зарядке и долговечность. Обычно цикл боев на турнире сумо-роботов состоит из 5 — 6 серий по 2-3 поединка. Надо помнить об удобстве перезарядки аккумуляторов или емкости батареи для прохождения такого цикла.

Типы батарей, используемых сейчас в минисумороботах – в основном Li-Pol или Li-Po (литий полимерные) и LiFePol(литий-ферум полимерные). NiMH(никельметалгидридные) используются все реже и реже – по мере удешевления литиевых аккумуляторов.

Недостаток литиевых аккумуляторов – необходимость использовать специализированные источники зарядки для избежания пожара и взрыва. С другой стороны будем помнить, что такие источники питания сейчас повсеместно распространены в коммуникаторах и прочих массовых электронных устройствах и они взрываются не очень часто. J

Итак, нам желательно сделать так, чтобы все наши электронные устройства (микропроцессорная плата, драйвер двигателя, датчики) могли быть подключены к стабильному (не допускающему снижение ниже 5V и повышение выше 5V) источнику питания.

Почему возникают скачки напряжения? Попробуйте подключить параллельно аккумулятору питания вольтметр (лучше стрелочный), затем подключите к аккумулятору мотор. Обратите внимание, что в момент подключения двигателя напряжение будет кратковременно меняться – «скакать» — это связано с тем, что для запуска двигателя необходимы значительные ресурсы по току – превосходящие в 2-3 раза затраты тока на вращение двигателя после запуска. Вот этих скачков и боится электроника.

При скачках напряжения выше допустимых в документации параметров, наши электронные устройства будут работать нестабильно – скажем микропроцессорная плата может «зависать» или перезагружаться, датчики расстояния будут выдавать некорректное расстояние – робот будет неуправляемым своей программой – «немного не в себе». J

Как раз для исключения скачков и получения необходимого напряжения питания электронных компонентов и используется DC-DC преобразователь,

который обеспечивает стабильное напряжение 5V и необходимую силу тока от источника питания – аккумуляторов робота.

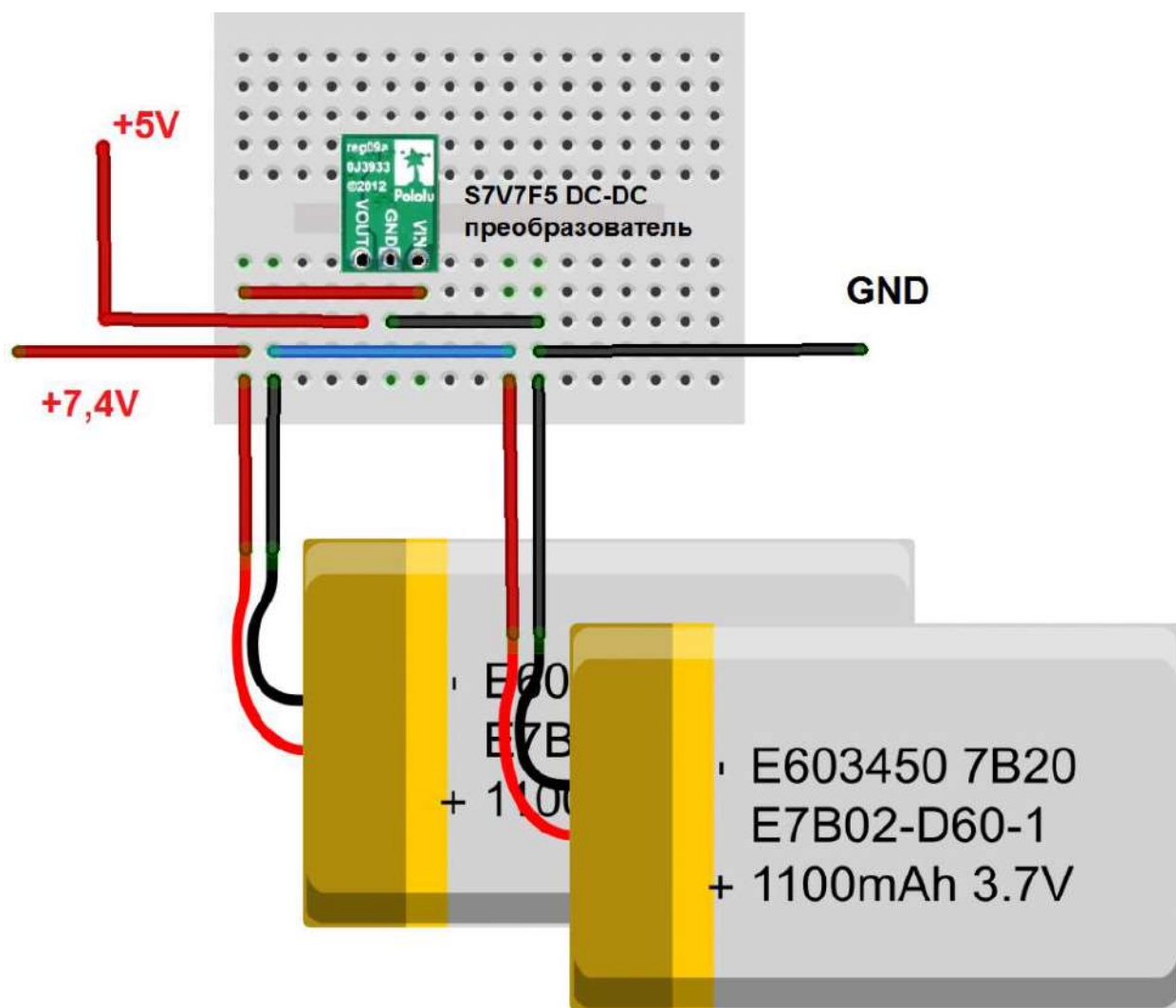
Давайте зафиксируем, что мы с Вами получили:

1. Для питания электронных компонентов нам надо 5V и 300mA
2. Для двигателей желательно 6V и 1200-1800 mA.

У нас есть литий-полимерные батареи с напряжением питания 3,7 V и 1100mA. Как из них получить необходимые нам параметры по напряжению для питания двигателей и электроники?

Известно, что при последовательном соединении одинаковых по напряжению и току источников питания (плюс одного элемента с минусом другого элемента) на сборке мы получим батарею с двойным номиналом по току и той-же силой тока, что и номинал каждого элемента. При параллельном соединении (минус с минусом и плюс с плюсом) напряжение останется тем-же, а сила тока батарейной сборки увеличится.

Используем последовательный метод сборки батарей из двух аккумуляторных элементов и получим сборку с напряжением 7,4V и силой тока 1100mA. Подключив к батарейной сборке преобразователь напряжения DC-DC от pololu.com модель S7V7F5 получим следующий источник питания (рисунок ниже)



Данная схема позволяет нам получить напряжение для питания двигателей – 7,4V и стабилизированное напряжение для питания электроники – 5V.

Обратимся к **параметрам DC-DC преобразователя** для того, чтобы уточнить его основные характеристики:

Pololu 5V Step-Up/Step-Down Voltage Regulator S7V7F5



The S7V7F5 switching step-up/step-down regulator efficiently produces 5 V from input voltages between 2.7 V and 11.8 V. Its ability to convert both higher and lower input voltages makes it useful for applications where the power supply voltage can vary greatly, as with batteries that start above but discharge below 5 V. The very compact (0.35" × 0.475") module has a typical efficiency of over 90% and can supply up to 1 A when stepping down and about 500 mA when stepping up.

Обратим внимание, что данный регулятор – универсальный – он может отдавать 5V от входного напряжения в 2,7V до 11,8V и от 500mA до 1000mA силы тока, что для нашего бюджета электроники в 150mA с избытком.

Для двигателей мы используем чуть большее, чем оптимальное, напряжение питания (не 6V а 7,4V) – что допустимо для данных моторов и находится в «зоне комфорта» для них.

Вопросы для проверки:

1. С чем сравнивают и напряжение и силу тока для понимания сути явления?
2. Почему появляются «скачки» напряжения и как их можно устранить?
3. Какие аккумуляторные элементы лучше использовать в сумороботе и почему? От каких факторов зависит выбор элементов питания?

Урок 8. Программа действий сумо-робота. Вытолкнем противника «в лоб» грубой силой.



«Тело» нашего сумо-робота готово и отдельные его компоненты «мозг» — микроконтроллерная плата, «глаза» — сенсоры расстояния и черного/белого, «сильные ноги» — двигатели и драйвер управления ими, «кровь и регулятор давления крови» — батарея и преобразователь напряжения DC-DC, «кровеносная система» — провода и макетные платы, скелет – база, отвал, бампер, крепежные элементы – нормально функционируют.

Функционируют отдельно друг от друга.

Что сделает робота «человеком»?

Божественный Замысел.

Безусловно, в человеческом понимании этого сложного философского понятия. J

Этим Замыслом будет программа действий робота в «мире» робота – на ринге (дохё) для соревнований роботов-сумо.

Что он должен будет сделать в краткий миг своего «просыпания» на поле боя перед новым «сном» после выключения кнопки питания – все зависит от Вас – вашей Идеи, воплощенной в программу действий робота.

Чем более точно Вы продумали поведение робота – тем более он будет успешен в своей «жизни». Какие «знания» вы ему дали, чему Вы его обучили – только это он постарается воплотить в жизнь.

От лирики перейдем к практике. J

Давайте начнем с простого замысла – Вашему роботу надо будет просто «в лоб» вытолкнуть соперника с ринга сразу после старта поединка.

Для того, чтобы трансформировать эту идею в программу надо разобраться с окружающими Вашего робота обстоятельствами и его поведением в этих обстоятельствах.

Начнем разбираться. По **правилам проведения поединков RobotChallenge.org** сумо роботы устанавливаются в пределах указанной при жеребьевке четверти ринга задней частью робота по направлению друг к другу.

Обратите внимание на правила боя – это фактически «законы физики» для робота. Их нельзя нарушать.

Итак, что должен наш сумо-робот сделать при такой расстановке (предположим он стоит в левой нижней четверти ринга)?

1. Сразу после старта поединка повернуться направо примерно на 135 градусов
2. Увидеть соперника передним датчиком расстояния.
3. Если увидел, то начать движение на него и вытолкнуть с ринга.

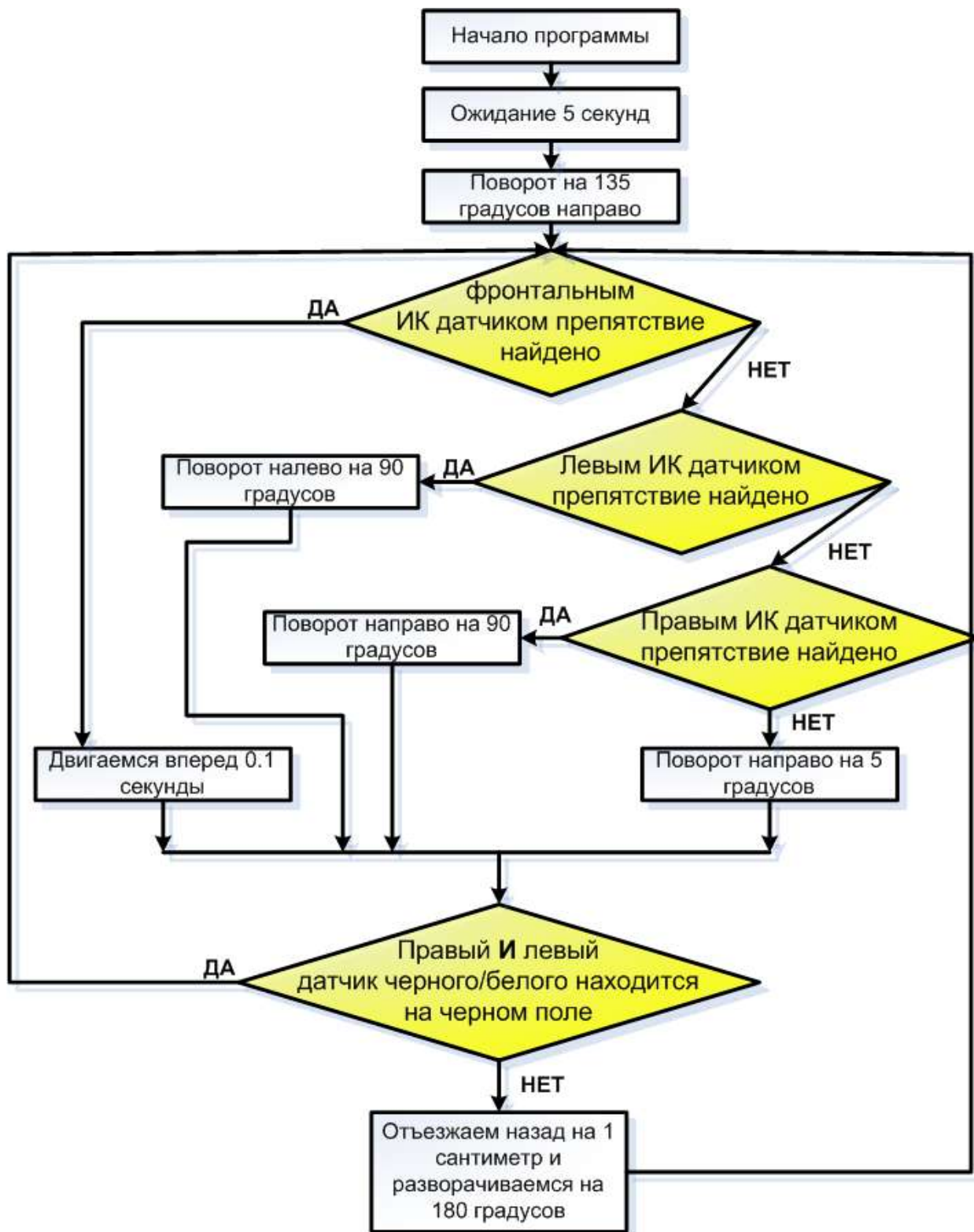
Для простоты сейчас не рассматриваем случаи, когда расстановка роботов отличается от вышеприведенной (другие четверти ринга, роботы ближе или дальше друг от друга).

С пунктом номер один все примерно понятно. Крутим левый двигатель вперед, правый одновременно назад. Время работы двигателей экспериментально подбираем так, чтобы робот повернулся примерно на 135 градусов.

Второй пункт с задачей увидеть соперника давайте разбирать подробнее. Первое – на каком расстоянии будем искать соперника? Помним, что ринг для минисумо имеет диаметр 77 сантиметров и выбранные нами датчики отлично измеряют расстояние от 10 до 50 сантиметров. Итак вывод простой – с учетом расстановки роботов противника лучше искать на расстоянии до 30 сантиметров от себя – будет меньше ошибок поиска.

Третий пункт также непросто. Итак – нашли и выталкиваем соперника – с этим все понятно. А если не увидели его на этом месте (он начал движение и ушел на другую позицию пока мы поворачивали робота предположим) – что будем делать? Давайте попробуем повернуться направо на небольшой угол и еще раз поискать соперника. Все эти действия будем делать если датчики черного/белого не выехали на белое поле.

Создадим блок схему программы поведения робота:



Разберем блок-схему.

Начало программы – момент, когда Вы включили питание робота.

Ожидание 5 секунд – для того, чтобы вы с соперником смогли отойти от ринга перед началом действий робота. В профессиональных соревнованиях роботы включаются по команде с пульта судей с использованием специального **стартового модуля**.

Затем начинаем исполнять логику, которую с Вами разобрали чуть ранее. На блок схеме желтый ромб означает оператор условия (if(условие){если условие

выполнено}else{если условие не выполнено}}, прямоугольник – оператор исполнения.

Обратите внимание – робот едет вперед при обнаружении противника 0.1 секунды, затем проверяет – не выкатился ли за пределы ринга и затем опять идет на оператор проверки – видит ли фронтальный датчик соперника.

Также обратим внимание, что робот будет искать соперников до момента его выключения кнопкой (переключателем) – из постоянного цикла нет программного выхода.

Итак, соберем программно нашего робота в единое целое – глаза, моторы и задуманное блок схемой поведение.

Скетч программы:

```
//Скетч для простого выталкивания соперника с ринга
//робот сразу после старта поворачивает на 135 градусов
//и если не находит соперника, то крутится направо
//подключаем «ноги»
//определяем соответствие контактов микроконтролера
//и контактам драйвера двигателей TB6612FNG
#define PWMA 3
#define AIN1 1
#define AIN2 0
#define BIN1 2
#define BIN2 7
#define PWMB 9
//включаем «глаза» для поиска соперника
//объявляем переменные, содержащие значения
//аналоговых портов микроконтроллера, подключаемых
//к ИК датчикам Sharp
int front_eye = A0;
int left_eye = A1;
int right_eye = A8;
//включем «глаза» для определения, на ринге мы находимся
//или выехали за него
//объявляем переменные, содержащие значения
//аналоговых портов, подключенных к датчикам
//QTR-1A
int left_bw = A6;
int right_bw = A11;
//процедура инициации портов
```

```

void setup()
{
  //инициируем переменные моторов
  pinMode(PWMA,OUTPUT);
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWMB,OUTPUT);
  pinMode(BIN1,OUTPUT);
  pinMode(BIN2,OUTPUT);
  //ждем 5 секунд
  delay(5000);
  // делаем поворот на 135 градусов
  //левый мотор по направлению движения робота
  //(против часовой стрелки)
  digitalWrite(AIN1, LOW);
  digitalWrite(AIN2, HIGH);
  analogWrite(PWMA, 255);
  //Правый мотор против направления движения робота
  //(против часовой стрелки)
  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, HIGH);
  analogWrite(PWMB, 255);
  delay(100); // длительность задержки определяет
  //угол поворота робота
  //и подбирается экспериментально в зависимости
  //от веса робота, моторов и колес
  //останавливаем моторы
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);
  digitalWrite(PWMA,LOW);
  digitalWrite(BIN1,HIGH);
  digitalWrite(BIN2,HIGH);
  digitalWrite(PWMB,LOW);
  delay(10);
}
void loop()
{

```

```

//проверяем наличие препятствия фронтальным датчиком
//на расстоянии ближе 40 см
//(значение analogRead() — 333 по таблице)
if(analogRead(front_eye) > 333){
//левый мотор по направлению движения робота
//(против часовой стрелки)
digitalWrite(AIN1, LOW);
digitalWrite(AIN2, HIGH);
analogWrite(PWMA, 255);
//Правый мотор по направлению движения робота
//(по часовой стрелке)
digitalWrite(BIN1, HIGH);
digitalWrite(BIN2, LOW);
analogWrite(PWMB, 255);
delay(100); // длительность задержки 0.1 секунда
}
else if(analogRead(left_eye) > 333){// если соперник
//не найден спереди то проверяем слева
//если соперник слева найден поворачиваем налево
//на 90 градусов
//левый мотор против направлению движения
//робота (по часовой стрелки)
digitalWrite(AIN1, HIGH);
digitalWrite(AIN2, LOW);
analogWrite(PWMA, 255);
//Правый мотор по направлению движения
//робота (по часовой стрелки)
digitalWrite(BIN1, HIGH);
digitalWrite(BIN2, LOW);
analogWrite(PWMB, 255);
delay(60); // длительность задержки
//определяет угол поворота робота
//и подбирается экспериментально в
//зависимости от веса робота, моторов и колес
}
else if(analogRead(right_eye) > 333){// если
//соперник не найден спереди и слева то проверяем справа

```

```

//если соперник справа найден поворачиваем
//направо на 90 градусов
//левый мотор по направлению движения робота
//(против часовой стрелки)
digitalWrite(AIN1, LOW);
digitalWrite(AIN2, HIGH);
analogWrite(PWMA, 255);
//Правый мотор против направления движения робота
//(против часовой стрелки)
digitalWrite(BIN1, LOW);
digitalWrite(BIN2, HIGH);
analogWrite(PWMB, 255);
delay(30); // длительность задержки определяет
//угол поворота робота
//и подбирается экспериментально в зависимости
//от веса робота, моторов и колес
}
else
{
  // если не соперник не найден спереди и слева
  //и справа то поворачиваем направо на 5 градусов
  //левый мотор по направлению движения робота
  //(против часовой стрелки)
  digitalWrite(AIN1, LOW);
  digitalWrite(AIN2, HIGH);
  analogWrite(PWMA, 255);
  //Правый мотор против направления движения
  //робота (ппротив часовой стрелки)
  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, HIGH);
  analogWrite(PWMB, 255);
  delay(10); // длительность задержки определяет
  //угол поворота робота
  //и подбирается экспериментально в зависимости
  //от веса робота, моторов и колес
  //останавливаем моторы
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);

```

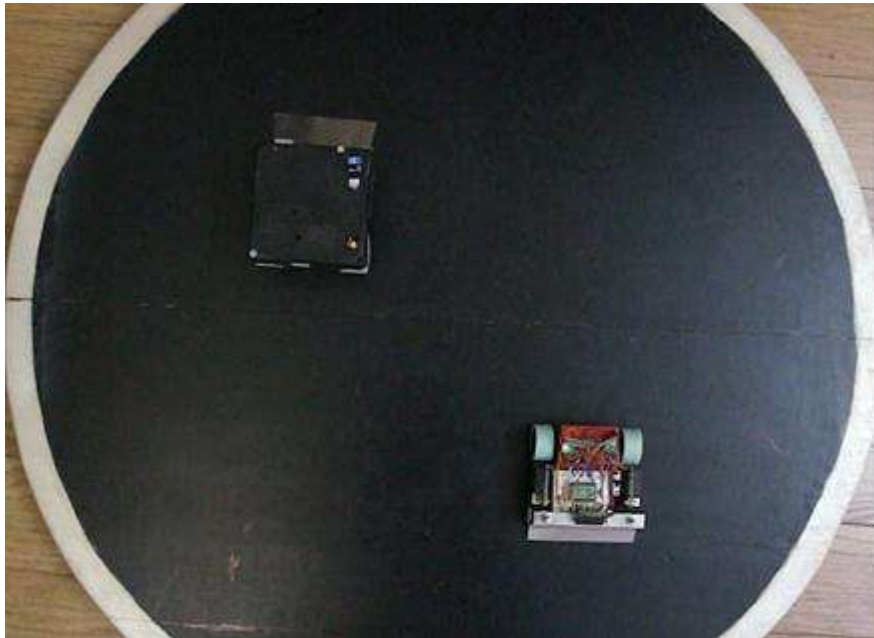
```
digitalWrite(PWMA,LOW);
digitalWrite(BIN1,HIGH);
digitalWrite(BIN2,HIGH);
digitalWrite(PWMB,LOW);
delay(30);
}
//проверяем что робот не выскочил за пределы ринга
//оба датчика на черном поле
if((analogRead(left_bw) > 400)&&(analogRead(right_bw) > 400)){
//ничего не делаем
}
else // выехали на белое поле одним или обоими датчиками
{
//Отъезжаем на 5 сантиметров
//левый мотор против направления движения робота
//(по часовой стрелки)
digitalWrite(AIN1, HIGH);
digitalWrite(AIN2, LOW);
analogWrite(PWMA, 255);
//Правый мотор против направления движения робота
//(против часовой стрелки)
digitalWrite(BIN1, LOW);
digitalWrite(BIN2, HIGH);
analogWrite(PWMB, 255);
delay(100); // длительность задержки определяет
//расстояние отъезда назад
//и подбирается экспериментально в зависимости
//от веса робота, моторов и колес
// разворачиваемся на 180 градусов в правую сторону
//левый мотор по направлению движения робота
//(против часовой стрелки)
digitalWrite(AIN1, LOW);
digitalWrite(AIN2, HIGH);
analogWrite(PWMA, 255);
//Правый мотор против направления движения робота
//(против часовой стрелки)
digitalWrite(BIN1, LOW);
```

```
digitalWrite(BIN2, HIGH);  
analogWrite(PWMB, 255);  
delay(130); // длительность задержки определяет  
//угол поворота робота  
//и подбирается экспериментально в зависимости  
//от веса робота, моторов и колес  
}  
//конец цикла loop — идем на начало цикла и проверяем расстояние спереди  
}
```

Все операторы в данном скетче нами уже изучались, поэтому трудностей в его понимании, надеюсь, не возникнет.

Приведем фотографии, сделанные из видеозаписи поединка с неподвижным соперником, с использованием данной программы поведения робота.

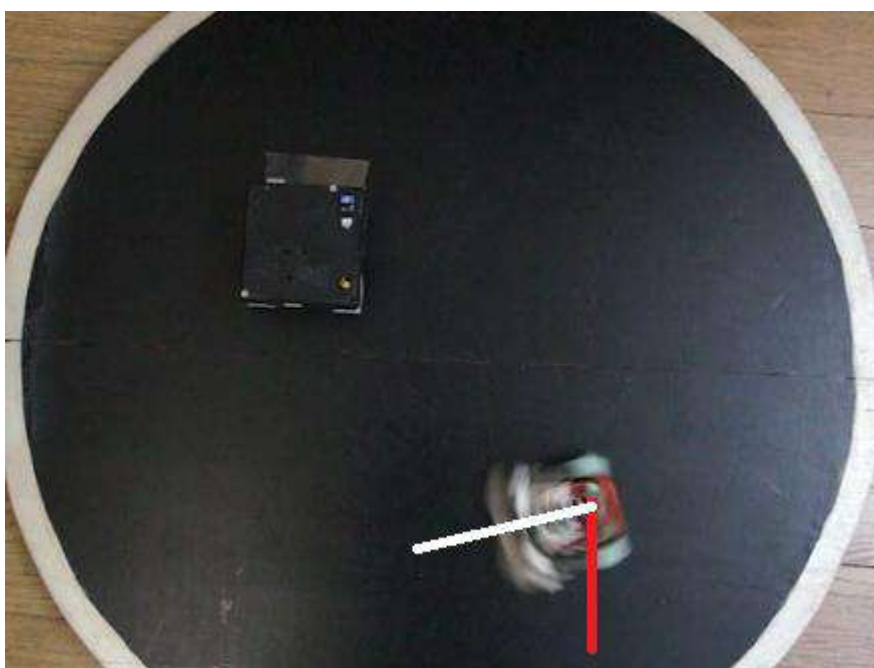
Начальная позиция.



Робот начинаем поворачиваться на 135 градусов направо (красная черта – начало отсчета, белая черта – текущее положение).



Продолжение поворота



Окончание поворота.



Обнаружен соперник. Движение вперед.



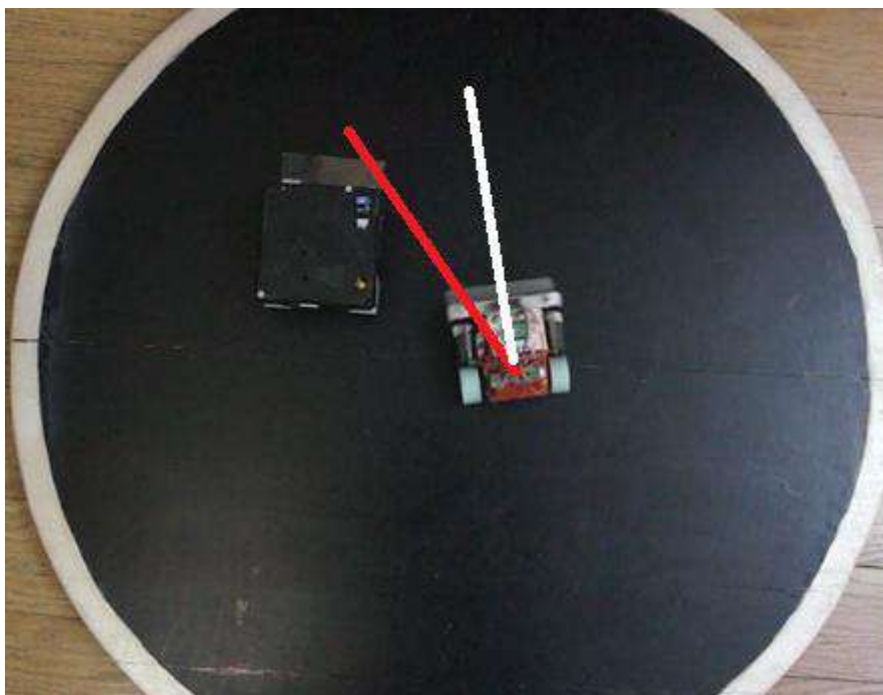
Соперник утерян из вида при движении вперед.



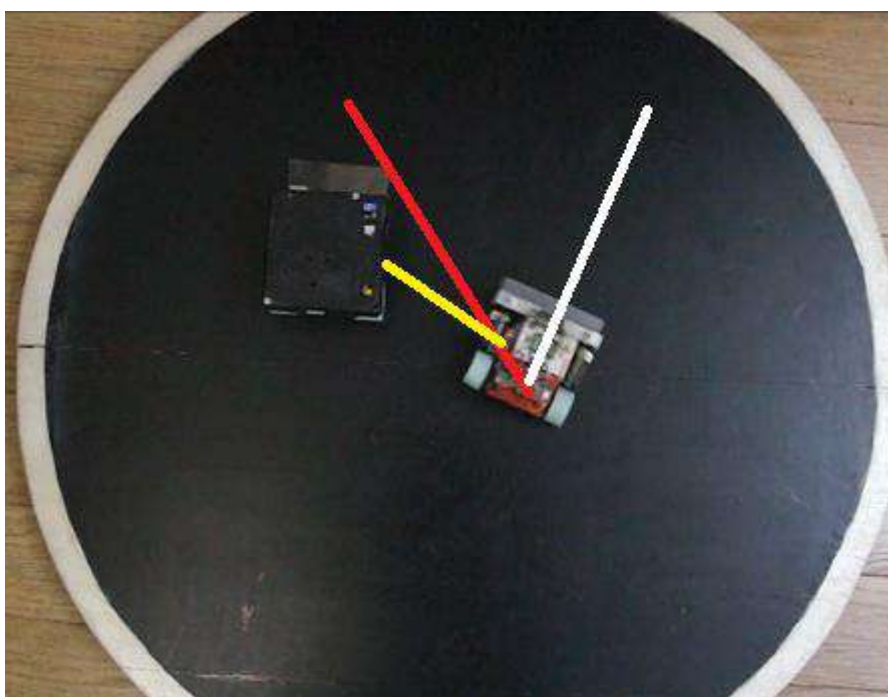
Начало движения направо.



Продолжение движения направо.



При продолжении движения направо соперник найден левым ИК датчиком (желтая линия).



Разворот налево на 90 градусов.



Соперник обнаружен передним ИК датчиком.



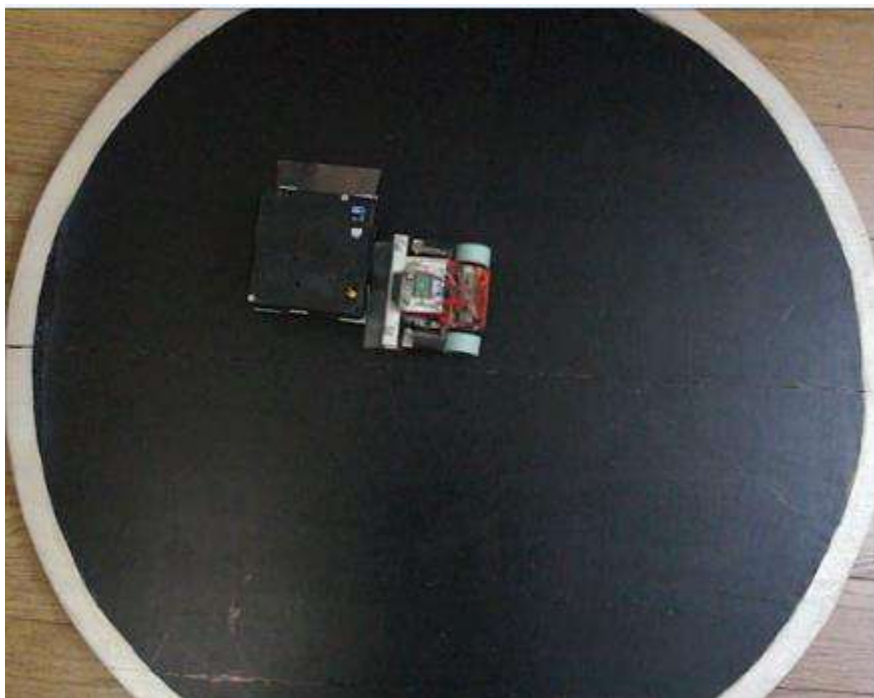
Движемся вперед к сопернику.



Врезаемся в него.



Пытаемся вытолкнуть.



Вопросы для проверки:

1. Программа действий робота. Почему важно продумать все нюансы программы?
2. Правила боев роботов сумо. Какие важные детали Вы запомнили?
3. Блок схема программы. Какие элементы блок схем Вы знаете? Удобно ли Вам использовать блок схему для отработки программы действий робота?
4. Подумайте, как можно оптимизировать скетч? Какие одинаковые блоки операторов в программе Вы увидели?

Урок 9. «Шаолинь» стилей боев. Двигаемся к «Седьмому дану».
Боевое искусство – пути и направления развития.



Мы создали робота и научили его выталкивать противника с ринга. Пока неподвижного соперника.

Но, насколько мне известно, неподвижные противники нечасто встречаются в реальных боях.

Как и в любом искусстве и спорте, ваш успех зависит от ваших умений, воплощенных в работе, искусства опередить и обхитрить соперника, предвосхитить его поведение и провести прием, который поразит его в самое «сердце».

Вес робота, параметры моторов, характеристики аккумуляторов, сила сцепления колес с поверхностью и стиль поведения робота должны быть оптимально сбалансированы между собой. Они будут диктовать поведение робота на ринге.

Можно выделить минисумороботов со стандартным поведением (маневрирование, направленное на выталкивание соперника передним отвалом) и нестандартных минисумороботов (любые остальные типы роботов), имеющих нестандартную уникальную тактику поведения.

Условно стандартных минисумо роботов можно разделить на тяжелых и облегченных. Тяжелые обычно максимально близки к разрешенным 500 граммам и обладают чуть меньшей динамикой разгона, чем облегченные, которые весят около 300 грамм и значительно более динамичны, что позволяет им лучше маневрировать и значительно быстрее набирать скорость, так как сила, затрачиваемая ими на преодоление инерции значительно меньше.

Также, оптимальное использование набранной инерции робота перед столкновением с соперником – ключ к победе. Практически это значит, что Вашему роботу желательно разогнаться перед столкновением с соперником и использовать инерцию движения соперника для его выталкивания с ринга – ударять не «в лоб», а в боковую часть или в заднюю часть робота, которые незащищены отвалом и значительно более уязвимы.

Перейдем к созданному нами роботу и обсудим, какие пути развития для него возможны. Прежде всего оценим существующие характеристики:

1. Робот получился скоростным и имеет возможность развиваться в сторону облегченных роботов (необходимо добавить груз массой около 100-150 грамм);
2. Робот имеет 3 «глаза», что позволяет ему одновременно контролировать значительную зону ринга (примерно 270 градусов с учетом возможного маневрирования);
3. Робот использует 2 силиконовых колеса, расположенные в задней части робота, что позволяет ему иметь хорошее сцепление и маневренность для осуществления поворотов;
4. Робот имеет датчики черного/белого только спереди, что ограничивает его маневрирование задним ходом во избежание неконтролируемого покидания ринга;
5. Робот имеет отвал в передней части, который имеет возможность быть дополнительно нагруженным для повышения прижимаемости к рингу и «подцепа» соперника.

После объективной оценки робота, переходим к решению о его возможных путях развития.

1. Оптимально – облегченный робот (тяжелые роботы обычно четырехколесные).

2. Поведение робота – скоростные маневры с преимущественным движением вперед, поиск уязвимых мест соперника и быстрые атаки.

Теперь оценим сам момент старта боя с точки зрения уязвимости робота. Когда, по-вашему, робот наиболее уязвим? Правильно, также как и самолет — «на взлете», то есть на старте.

Давайте разбираться почему. Все очень просто – именно на старте Ваш противник имеет Ваши точные координаты на ринге и использует самый совершенный (пока) компьютер – голову робоспортсмена – для оценки размещения своего робота на ринге с целью сделать стартовый маневр для оптимального выталкивания Вас с ринга.

Наш стартовый маневр сейчас – поворот направо на 135 градусов. Для его успешности робоспортсмену важно точно выставить робота на ринге, чтобы после старта программы и поворота он сразу увидел соперника и, разогнавшись, вытолкнул его с ринга.

С учетом уровня проводящихся сейчас боев на серьезных соревнованиях, такой маневр очень прост и успешен против совсем новичков, делающих первые шаги в минисумо робоспорте.

Для успешности, ваша основная задача сразу после старта боя — иметь несколько заранее отработанных и запрограммированных маневров ухода со стартовой позиции с одновременным определением маневра и позиции соперника и поиска и расчета его вектора движения для расчета своего маневра для удара в его боковую или заднюю часть.

Сложно? Безусловно непросто.

Как разрабатываются и программируются стартовые маневры?

Прежде всего начинают с разбора видеозаписей боев потенциальных соперников или лучших роботов. В замедленном воспроизведении фиксируется поведение изучаемого робота и его маневры, определяется частота употребления этих маневров и статистика их использования.

Затем моделируется поведение своего робота для противодействия данным маневрам и выбора нескольких оптимальных тактик стартовых маневров.

Лучше иметь несколько различных отработанных маневров и запрограммировать их использование в зависимости от соперника, с которым предстоит сражаться.

Мы намеренно не рассматриваем в данной книге возможные программные и аппаратные реализации успешных примеров маневрирования.

Приведя примеры можно «убить» творчество робоспортсмена. Повести Вас проторенной тропой, не дав вам, возможно, найти более «короткую и простую» дорогу через увлекательные «джунгли» робоспорта, не дав возможности Вам стать Мастером, разработавшим собственную школу «единоборств» роботов.

Приведем лишь возможные примеры направлений реализации технологически успешных решений роботов сумо на базе Arduino.

1. Использование EEPROM Arduino и модулей памяти для формирования «памяти» робота и отработки маневров;

2. Использование нескольких процессоров Arduino для распараллеливания процессов вычислений либо управления роботом;

3. Использование дополнительных модулей (акселерометров, компасов, гироскопов, запчастей от компьютерных «мышек») для вычисления собственного текущего положения на ринге и расчета маневров.

Урок получился очень коротким, но он открывает Вам двери к собственному творчеству, поиску и ошибкам, удачам и поражениям, радости и горечи – всему, что связано с совершенствованием себя и Ваших роботов – ДАО РОБОТА СУМО.