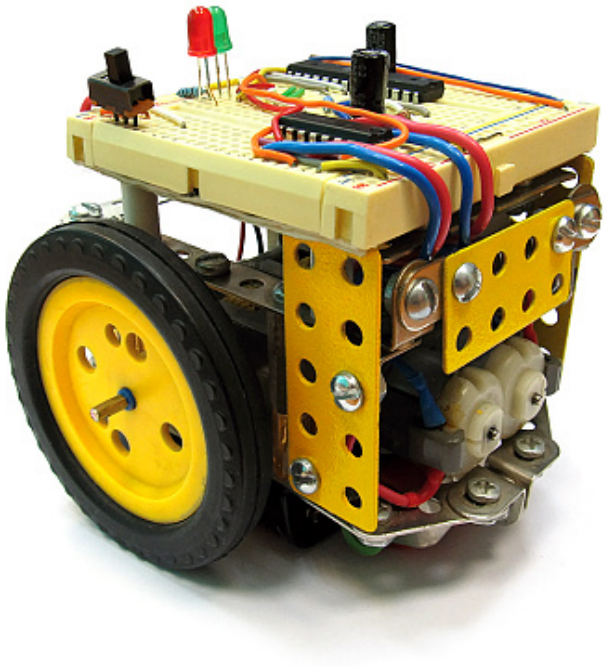
РОБОТ, ОБЪЕЗЖАЮЩИЙ ПРЕПЯТСТВИЯ

Как сделать робота без датчиков, объезжающего препятствия



В качестве продолжения экспериментов, описанных в статье "Простой робот на микроконтроллере" мы попробуем сделать робота, объезжающего препятствия. Особенностью робота будет то, что у него не будет датчиков в традиционном смысле, а определять препятствия робот будет при помощи измерения значений напряжения на электромоторах.

Когда робот упирается в препятствие, его колеса встречают сопротивление и начинают тормозиться. Моторы при этом пытаются вращать колеса и испытывают увеличение нагрузки. Нагрузка вызывает увеличение потребления тока электромоторами. В электрической цепи робота происходит просадка напряжения, связанная с увеличением потребления тока электромоторами.

Встретив на своем пути препятствие, робот будет "чувствовать" увеличение нагрузки на моторы. Алгоритм его движения в этом случае будет достаточно простым: робот будет отъезжать немного назад, поворачиваться и снова двигаться вперед, пытаясь таким образом объехать встретившееся препятствие.

Для того, чтобы измерить напряжение на моторах нам будет необходимо использовать аналого-цифровой преобразователь (АЦП, англ. Analog-to-digital converter, ADC), который

производит преобразование входного сигнала в численное представление. АЦП в Atmega8 десятибитный, то есть значение измеренного напряжения будет лежать в пределах от 0 до 1023 в численном выражении.

Для работы АЦП требуется источник опорного напряжения, относительно которого АЦП производит измерения. Напряжение, которое преобразуется АЦП, должно быть меньше опорного. Опорное напряжение можно подать на специальную ножку AREF микроконтроллера. Можно также использовать внутренний источник опорного напряжения на 2,56 вольта или использовать напряжение питания. Кроме того, у Atmega8 есть отдельные выводы для питания АЦП: AVCC (аналоговое питание) и AGND (аналоговая "земля"). Подключим AVCC и AREF к положительному полюсу источника питания, а AGND к общей "земле". Следует отметить, что на точность работы АЦП могут влиять наводки и помехи. В нашем учебном примере ими можно пренебречь, но в законченных устройствах следует предпринять меры, описанные в статье "СТАБИЛИЗАЦИЯ РАБОТЫ МИКРОКОНТРОЛЛЕРА".

Выводы PC0 (канал АЦП ADC0) и PC1 (канал АЦП ADC1) будем использовать для измерения напряжения. Подключим их через ограничительные резисторы с номиналом 150-220 Ом к тем выводам электромоторов, которые обеспечивают движение вперед при подаче на них положительного напряжения. Для избежания слишком больших просадок напряжения, приводящих к случайной перезагрузке микроконтроллера, подключим к выводам питания микросхем электролитические конденсаторы с номиналом около 470 мкф.

Для индикации используем два светодиода, которые подключим через ограничительные резисторы, например, к выводам PD6 и PD7. Выводы PC2 и PC3 используем для управления мотором M1, а выводы PB1 и PB2 для управления мотором M2.

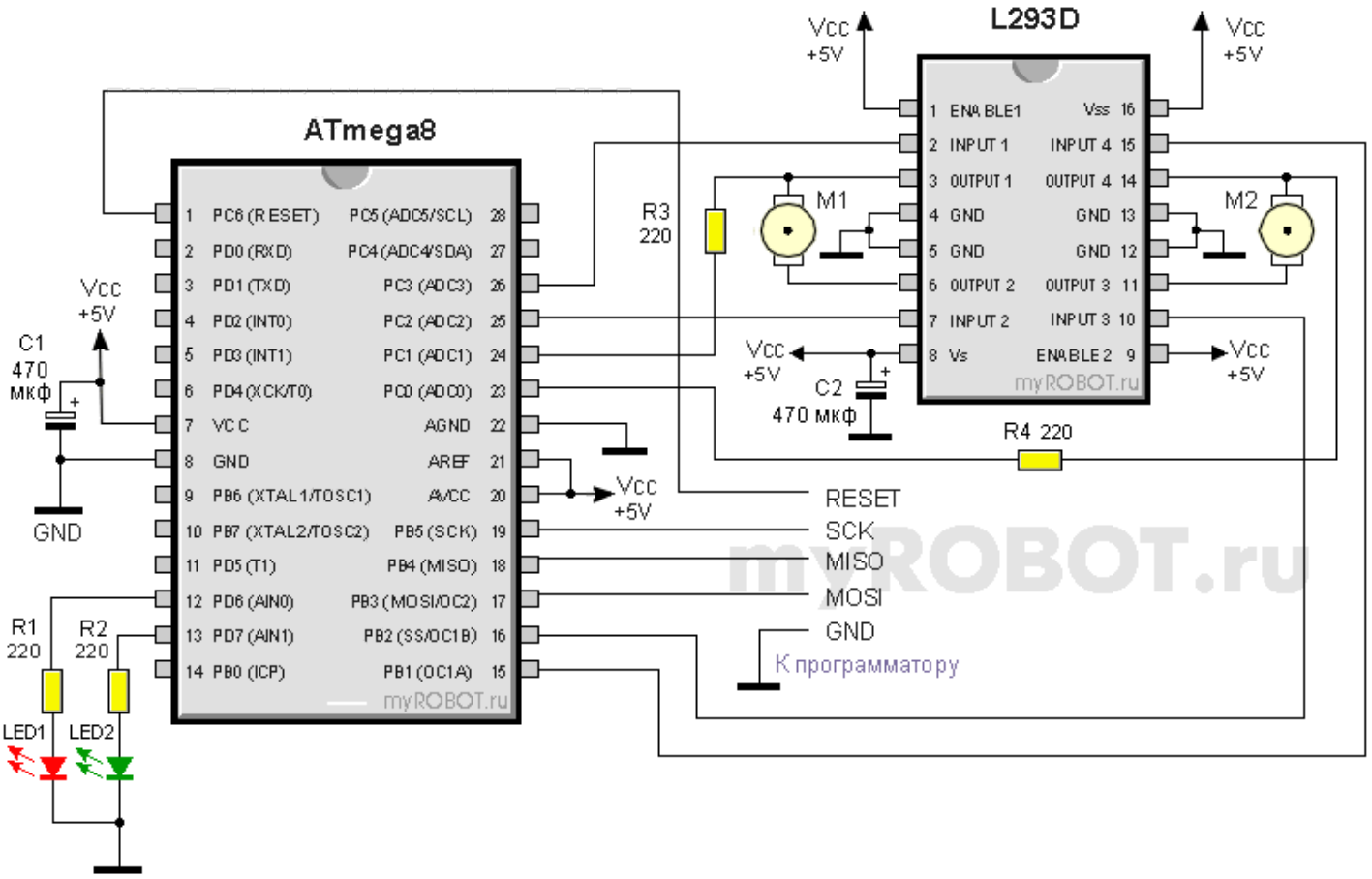
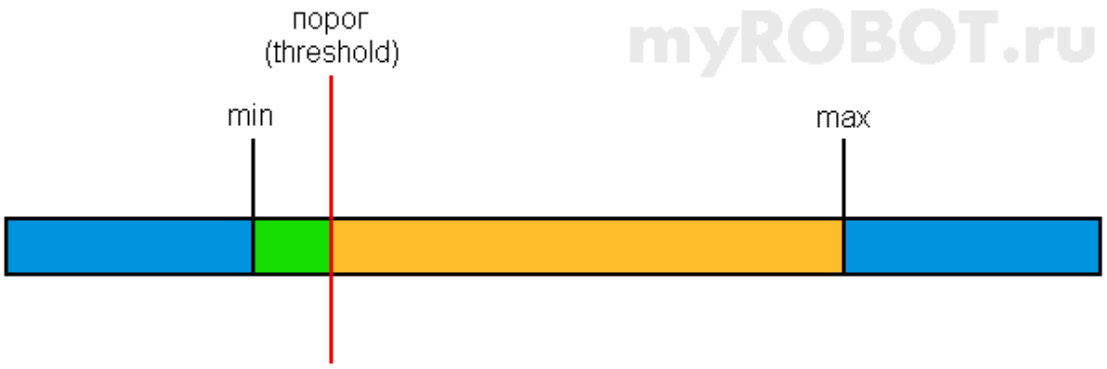


Схема робота, объезжающего препятствия

Комментарии к схеме робота

Выбор выводов для управления моторами продиктован в данном примере по большей части тем, что они находятся на одной стороне микроконтроллера и их подключение к микросхеме драйвера моторов L293D достаточно удобно нарисовать на схеме. В своей конструкции Вы можете использовать другой набор выводов как для светодиодов, так и для электромоторов. Выводы PC4 и PC5 оставлены незадействованными для того, чтобы при совершенствовании конструкции в дальнейшем у нас были в запасе как минимум два свободных канала АЦП (ADC4 и ADC5).

В общем виде программа для робота будет выглядеть следующим образом. Включаем моторы для движения вперед. Немного ждем, пока утихнет скачок напряжения, вызванный стартом моторов. Пока робот еще не столкнулся ни с одним препятствием, делаем серию опросов АЦП, чтобы найти минимальное и максимальное значение напряжения. В каждой серии мы будем определять напряжения на моторах M1 и M2 многократно, чтобы найти их средние значения. Поиск средних значений необходим, чтобы нивелировать помехи.



Вычислив минимальное и максимальное значение напряжения на каждом моторе, мы можем определить значение порога (threshold) для напряжения, при котором мы будем считать, что нагрузка на моторы возросла и перед роботом, скорее всего, возникло препятствие. Этот порог должен быть немного больше, чем среднеминимальное значение напряжения. Вычислим его по следующей формуле:

threshold = min + (max - min) / 20

Следует отметить, что величина, на которую делится дельта между max и min, может быть не 20, а, например, 10 или 15. Эта величина зависит от конкретных моторов, которые будут использоваться в конструкции робота, и подбирается опытным путем. Именно от этой величины будет зависеть чувствительность робота к препятствиям.

После того как все предварительные замеры и вычисления сделаны, мы можем начать бесконечный цикл основной части программы, в котором мы будем опрашивать АЦП, измеряющий напряжение на моторах, и если оно упадет ниже порога (threshold), то будем давать команды моторам для отъезда назад и небольшого поворота. Сторону для поворота будем определять сравнив напряжение на моторах M1 и M2: на моторе со стороны препятствия, падение напряжения обычно больше.

При написании программы воспользуемся набором макроопределений (подстановок) для более удобной работы с портами микроконтроллера. Макроопределение начинается с директивы #define, после которой написан идентификатор (буквосочетание) для того, чтобы препроцессор при компиляции заменил в тексте программы все вхождения этого буквосочетания на макроподстановку, которая написана после идентификатора. Если в макроподстановке несколько строк, то для их объединения в один блок используют знак обратного слэша \.

Для удобства работы с АЦП напишем функцию read_ADC, которая будет принимать номер канала АЦП и возвращать значение, полученное от преобразователя. Работа этой функции достаточно проста. Сначала нам необходимо в регистре ADMUX (ADC Multiplexer Select Register) указать номер канала (вывода микроконтроллера), для которого будет производиться преобразование (выбор канала АЦП осуществляется установкой битов MUX4..MUX0). Структура регистра ADMUX показана на рисунке.

РЕГИСТР ADMUX

7 бит	6 бит	5 бит	4 бит	3 бит	2 бит	1 бит	0 бит
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

выбор источника опорного напряжения		выравнивание результата в регистрах ADCH и ADCL
вход AREF	0	0
напряжение питания (AVCC)	0	1
зарезервировано	1	0
внутренний источник на 2.56 В	1	1
		0 - вправо
		1 - влево

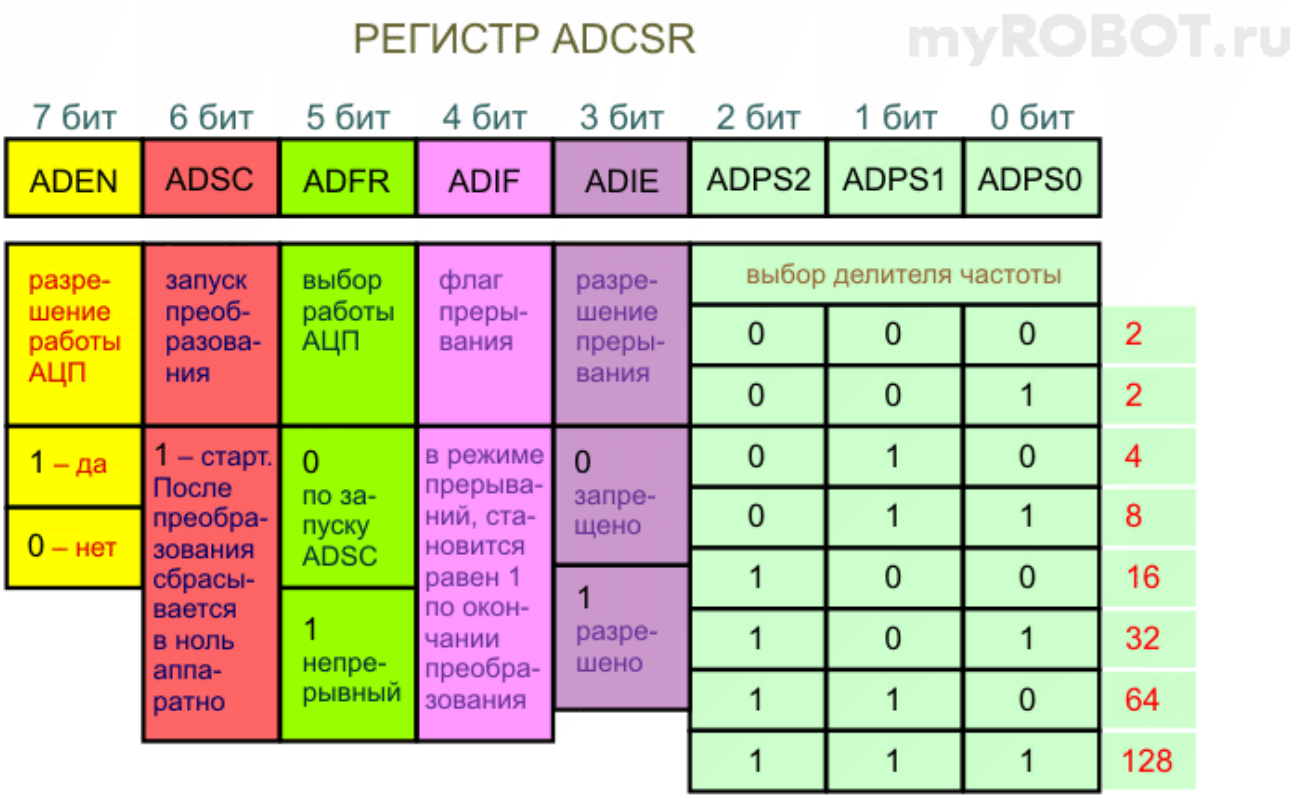
выбор канала АЦП					
0	0	0	0	0	ADC0
0	0	0	0	1	ADC1
0	0	0	1	0	ADC2
0	0	0	1	1	ADC3
0	0	1	0	0	ADC4
0	0	1	0	1	ADC5
0	0	1	1	0	ADC6
0	0	1	1	1	ADC7

В старших битах REFS1, REFS0 и ADLAR оставим нули, что будет соответствовать выбору источника опорного напряжения на выводе AREF и выравниванию результата в регистрах ADCH и ADCL по правому краю.

Как уже говорилось выше, АЦП в ATmega8 десятибитный, результат преобразования записывается в два восьмибитных регистра ADCH (для старших разрядов) и ADCL (для младших). При этом первым следует читать значение из регистра ADCL, а затем из регистра ADCH.

Если достаточно восьмибитной точности оцифровки аналогового сигнала, например в тех случаях, когда сигнал зашумлен, то можно отбросить два младших бита в получаемом результате. Сделать это можно, если в бите ADLAR регистра ADMUX установить 1 — результат будет выровнен влево и достаточно будет считывать только регистр ADCH, отбрасывая два младших бита, остающихся в регистре ADCL.

Теперь перейдем к регистру ADCSR (ADC Control and Status Register). Для того, чтобы начать преобразование необходимо установить единицу в бите ADSC. Структура регистра ADCSR показана на следующем рисунке.



После окончания преобразования бит ADSC будет сброшен к нулю и нам можно будет прочитать получившийся результат. Стоит добавить, что помимо регистров ADCH и ADCL, о которых речь шла выше, WinAVR предоставляет виртуальный шестнадцатибитный регистр ADCW, в который склеиваются данные из регистров ADCH и ADCL. Именно значение из регистра ADCW и будет возвращать наша функция.

В заключение описания работы с АЦП остановимся еще на двух важных установках, которые нам будет необходимо сделать перед началом преобразований в регистре ADCSR. По сути, эти установки являются инициализацией АЦП. Во-первых, выберем делитель частоты. Оптимальное значение частоты для АЦП лежит в пределах 50-200 КГц. Если наш микроконтроллер будет работать на частоте 1000000 Гц, то при делителе 8, частота получится равной 125000 Гц или 125 КГц, что будет хорошим выбором. Установим биты ADPS: ADPS2 = 0, ADPS1 = 1, ADPS0 = 1. Во-вторых, чтобы включить АЦП установим бит ADEN = 1. Все эти установки можно сделать одной командой **ADCSR = 0b10000011**, которую мы напишем в начале программы в разделе инициализаций.

```
/*****************************************************************************
ПРОГРАММА ДЛЯ РОБОТА, РЕАГИРУЮЩЕГО НА ПРЕПЯТСТВИЯ С ПОМОЩЬЮ АЦП :: MYROBOT.RU
*****************************************************************************/

#include <avr/io.h>
#include <util/delay.h>

// макроопределения для включения моторов
// вперед
#define FORWARD PORTC = 0b11111011;\
```

```
                PORTB = 0b11111011
// назад
#define BACK    PORTC = 0b11110111;\
                PORTB = 0b11111101
// вправо
#define RIGHT   PORTC = 0b11111011;\
                PORTB = 0b11111101
// влево
#define LEFT    PORTC = 0b11110111;\
                PORTB = 0b11111011

// макроопределения для включения светодиодов
#define RED     PORTD = 0b01111111
#define GREEN   PORTD = 0b10111111

//-------------------------------------------------------------------------------
// Функция для чтения АЦП (ADC)
//-------------------------------------------------------------------------------
unsigned int read_ADC(unsigned char channel)
{
        ADMUX = channel; // выбор канала АЦП (от 0 до 7),
                         // опорное напряжение берем с вывода AREF
        ADCSR |= _BV(ADSC); // запуск преобразования
        loop_until_bit_is_set(ADCSR,ADSC); // ждем, пока не будет сброшен ADSC
        _delay_us(1); // ждем обновления регистра данных
        return ADCW; // возвращаем результат
}

//-------------------------------------------------------------------------------
// Главная программа
//-------------------------------------------------------------------------------
int main(void)
{

        unsigned long  m1 = 0, m2 = 0; // переменные для записи показаний АЦП
                                       // для моторов M1 и M2
        unsigned int max1 = 0, max2 = 0, // переменные для записи
                                       // максимальных значений показаний АЦП
        min1 = 2000, min2 = 2000, // переменные для записи
                                       // минимальных значений показаний АЦП
        threshold1, threshold2, // переменные для порога срабатывания
                                       // при возникновении препятствия
        i, j; // переменные для счетчика цикла

        DDRB  = 0xff; // назначаем все линии порта B как выходы
        DDRD  = 0xff; // назначаем все линии порта D как выходы
        DDRC  = 0xfc; // назначаем линии порта C (PC0 и PC1 - входы)
        PORTD = 0xff; // включаем светодиоды

        FORWARD; // включаем моторы для движения вперед

        // инициализируем АЦП
        ADCSR = 0b10000011; // 7-й бит ADEN = 1 включает АЦП; ADPS2..0 = 011
                            //  коэффициент предделителя частоты = 8;

        _delay_ms(500); // ждем, пока утихнет скачок напряжения,
                        // вызванный стартом моторов

        // делаем 50 серий опросов АЦП для нахождения
        // минимальных и максимальных значений
        for (i = 1; i <= 50; i++)
        {
                for (j = 1; j <= 250; j++) // повторяем 250 раз
                {                                  // опрос АЦП для M1 и M2
                        m1 += read_ADC(PC1);
                        m2 += read_ADC(PC0);
```

```
                }

                // находим среднее арифметическое значений АЦП для M1 и M2
                m1 /= 250;
                m2 /= 250;

                // находим минимальные и максимальные значения
                if (m1 < min1) min1 = m1;
                if (m1 > max1) max1 = m1;
                if (m2 < min2) min2 = m2;
                if (m2 > max2) max2 = m2;
        }

        // вычисляем пороги срабатывания
        // при возникновении препятствия для моторов M1 и M2
        threshold1 = min1 + (max1 - min1) / 20;
        threshold2 = min2 + (max2 - min2) / 20;

        // бесконечный цикл
        for(;;) {

                m1 = 0;
                m2 = 0;

                // снимаем показания АЦП 250 раз
                for (i = 1; i <= 250; i++)
                {
                        m1 += read_ADC(PC1);
                        m2 += read_ADC(PC0);
                }

                m1 /= 250; // находим среднее арифметическое
                m2 /= 250;

                if (m1 < threshold1 || m2 < threshold2) { // сравниваем
                                // с порогом срабатывания на препятствие
                        BACK; // включаем моторы для отъезда назад

                        _delay_ms(1000); // ждем, пока пройдет отъезд назад

                        if (m1 < m2) //проверяем с какой стороны было препятствие
                        {
                                RIGHT; // включаем моторы для разворота вправо
                                RED; // включаем красный светодиод
                        }
                        else
                        {
                                LEFT; // включаем моторы для разворота влево
                                GREEN; // включаем зеленый светодиод
                        }

                        _delay_ms(300); // ждем, пока пройдет разворот

                        FORWARD; // включаем моторы вперед

                        _delay_ms(800); // ждем, пока утихнет скачок
                                // напряжения, вызванный реверсом моторов
                }

        } // закрывающая скобка бесконечного цикла

} // закрывающая скобка основной программы
```

Загрузим программу в микроконтроллер. (Как это сделать, см. в статьях: Makefile и компиляция программы; Простой программатор AVR; Первый проект на микроконтроллере AVR.) После чего, поставив робота на поверхность, подождем пока он столкнется с препятствием. Изменяя делитель (20) в формуле вычисления порога срабатывания, поэкспериментируем с чувствительностью робота. Так как недорогие моторы обычно имеют небольшой разброс характеристик, возможно, делители для моторов M1 и M2 придется подобрать разные.

Технология обнаружения препятствий, описанная в этой статье, не призвана заменить стандартные датчики, но может использоваться параллельно с ними. Это будет полезным дополнением к "чувствам" робота, особенно в те моменты, когда препятствие попадает в "слепую зону" основных датчиков.

Желаем Вам успехов!

Datasheets:

Описание микросхемы управления двигателем L293D (англ.).
L293D.pdf

Описание микроконтроллера ATmega8 (англ.).
ATmega8.pdf

Для просмотра необходим Adobe Acrobat Reader

STEP BY STEP

Статьи раздела РОБОТЫ

Все статьи курса

Простой робот на микроконтроллере (Часть 1). Управляем электромоторами.

Простой робот на микроконтроллере (Часть 2). Робот с фотодатчиком.

Простой робот на микроконтроллере (Часть 3). Робот для соревнований.

Простой робот на микроконтроллере (Часть 4). Как сделать робота, объезжающего препятствия.