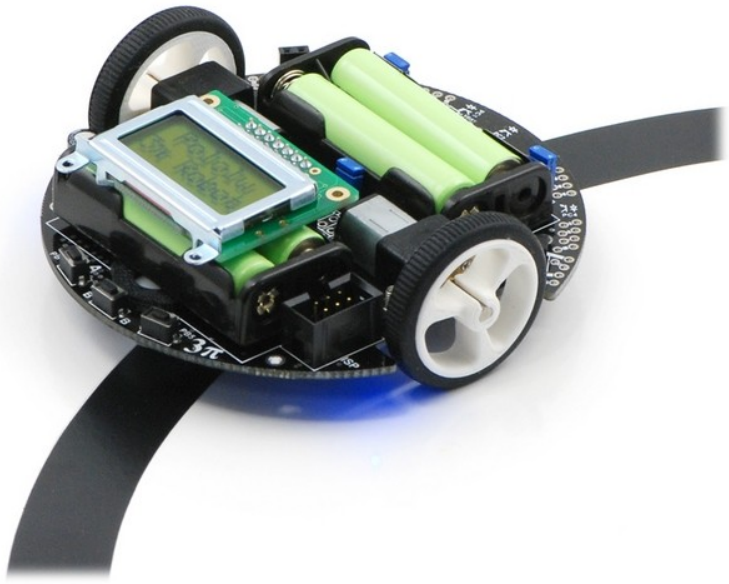
Руководство

Робот Pololu 3pi



Содержание

3. Начало работы с 3pi.....5

Работать с 3pi также просто как и вытаскивать его из коробки, вставлять батареи, и включать. Демонстрационная программа, загруженная в 3pi, быстро ознакомит вас с основными возможностями. Но прежде чем приступить к работе с роботом, ознакомьтесь с устройством 3pi.....5

.....6

.....6

Как видно из картинки, в комплект вместе с роботом поставляются 12 перемычек, которые возможно потом понадобятся и 4 светодиода (два из них можно припаять на обратную сторону платы управления роботом).....10

4.Первый контакт с 3pi.....11

4.c. Двигатель и коробка передач.....12

6.a. Загрузка и Монтаж C/C ++ Библиотека 21

6.b. Компиляция простой программы 22

7. Проект №1 в качестве примера.....25

7.a. О программе: езда по линии.....25

Теперь, когда Вы знаете как собрать простую программу для 3pi, то пора узнать как робот справляется с более сложными задачами. В этом проекте в качестве примера мы покажем вам, как 3pi может следовать по черной линии на белом фоне, с помощью своих датчиков и двигателей. Задача о следовании по линии – это самная простая и фундаментальная задача в программировании роботов. 25

7.b. Алгоритм следования по линии 26

Функция read_line() берет на себя чтение значений с датчиков и возвращает оценку положения робота относительно линии в виде числа между 0 и 4000. Значение 0 означает, что линия слева от датчика, а значение 1000 показывает, что линия непосредственно находится под датчиком, 2000 - линия непосредственно находится под датчиком 2 и так далее.....26

0–1000: робот далеко справа от линии. В этом случае, чтобы покинуть эту позицию, мы устанавливаем правильную частоту вращения двигателя в 100 и левую частоту вращения двигателя в 0. 26

1000–3000: робот возможно на линии. В этом случае мы заставляем оба двигателя ускоряться со значением 100, т.е. двигаться прямо вперед.....26

3000–4000: робот далеко слева от линии. В этом случае мы поворачиваемся резко направо, устанавливая частоту вращения правого двигателя в 0 и частоту вращения левого двигателя в 100.....26

7.c. Усовершенствованная версия программы езды по линии. ПИД-регуляторы.....27

Усовершенствованная версия программы для следования по линии для 3pi доступна в папке examples\atmegaxx8\3pi-linefollower-pid.....27

Метод, используемый в этой программе, известный как управление PID, решает проблему резкого движения робота. PID использует непрерывные функции, чтобы вычислить частоты вращения двигателей, так, чтобы неровное движение, при сходе с дистанции, было более мягким. PID (Proportional Integral Differential), что обозначает пропорциональная величина, интегральная величина, производная величина; это три входных значения, используемые в простой формуле, чтобы вычислить скорость для робота.....28

Пропорциональная величина характеризует следующее: если робот будет точно находиться на линии, то значение будет равно 0. Если робот будет слева от линии, то пропорциональная величина будет положительным числом, а справа от линии - отрицательным числом. 28

Интегральное значение записывает историю движения робота: это - сумма всех значений пропорциональной величины, которые регистрируются, с того момента как робот начал работать.....28

Производная величина – фиксирует изменения пропорционального значения. Вычисляется как разность последних двух пропорциональных значений.....28

1. Введение

Pololu 3pi робот это маленький, высокоэффективный, автономный робот, разработанный для участия в соревнованиях на следование по линии и прохождения лабиринта. Для того, чтобы робот смог передвигаться, ему понадобится 4 батарейки типа AAA (не включены в поставку). Также этот робот снабжен уникальной системой управления, которая работает с

двигателями в отрегулированном напряжении в 9.25 V. Скорость 3pi может достигать до 1 м/с, с помощью нее робот может совершать точные повороты и вращения.

Комплектация:

1. электромоторы
2. ИК-датчики (5 шт. на задней стороне робота)
3. ЖК-дисплей размером 8×2
4. Пищалка
5. 3 пользовательских кнопки

3pi размером 3.7 дюйма (9.5 см) в диаметре и весит 2.9 унции (83 g) без батарей. На борту 3pi присутствует микроконтроллер от фирмы Atmel ATmega328, впредь называем “ATmegaxx8”, тактовая частота которого может достигать 20 МГц.

Характеристики микроконтроллера ATmega328:

- 32 КБ Flash
- 2 КБ RAM
- 1 КБ EEPROM

Также 3pi совместим с популярной платформой разработки Arduino.

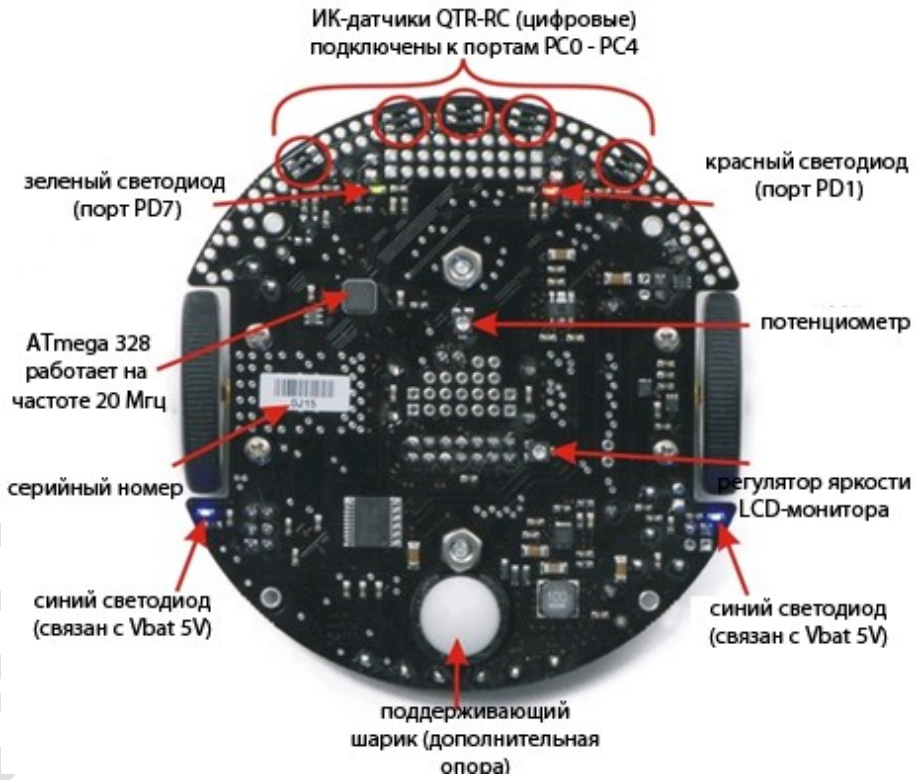
2. Меры предосторожности

3pi робот не предназначен для маленьких детей! Пользователи, не достигшие 13-15 лет должны использовать этот продукт только под наблюдением взрослых. Также, следует принять во внимание следующие меры предосторожности:

- Не пытайтесь программировать 3pi, если его батареи истощены или не заряжены, что может **надолго повредить** 3pi (зарядите сначала аккумуляторы). С помощью демонстрационных программ у вас появится возможность контролировать напряжение батарей. Вы можете использовать эту программу, чтобы знать, когда питание робота на исходе и его надо подзарядить, чтобы избежать фатальных последствий.
- 3pi содержит свинец, так что не пытайтесь его облизывать и мойте руки после работы.
- 3pi предназначен для использования в закрытом помещении на относительно плоских, гладких поверхностях. Избегайте управления на неровных поверхностях, которые могли бы повредить нижнюю сторону платы управления робота.
- Не допускайте контакта платы с проводящими поверхностями. Это может привести к короткому замыканию батарей и повреждению вашего робота, даже в выключенном состоянии.
- Защищайте его от электростатического электричества, которое может повредить бортовую электронику. Поднимая 3pi, вы должны сначала коснуться безопасной части робота, например колеса, двигателя, батареи, или края платы.
- Если Вы удаляете ЖК-дисплей, то запоминайте его первичное расположение. Если этого не сделать, то при неправильной установке ЖК-дисплей можно испортить.

3. Начало работы с 3pi

Работать с 3pi также просто как и вытаскивать его из коробки, вставлять батареи, и включать. Демонстрационная программа, загруженная в 3pi, быстро ознакомит вас с основными возможностями. Но прежде чем приступить к работе с роботом, ознакомьтесь с устройством 3pi.



3.a. Что понадобится для работы с 3pi

Следующие материалы, которые необходимы для того, чтобы начать работу с 3pi:

4 батареи AAA. Любые батареи (аккумуляторы) AAA будут работать, но мы рекомендуем батареи NiMH, которые можно перезаряжать.

AVR ISP программатор с 6-штыревым разъемом. Программатор **USB Pololu AVR** или **Atmel AVRISP**. 3pi имеет стандартный 6-штыревой программный разъем, в свою очередь, программатор должен иметь **6-жильный кабель ISP** для того, чтобы соединиться с устройством.

Настольный компьютер или ноутбук. 3pi может быть запрограммирован на Windows-, Mac- и Linux-компьютерах, но поддержка Pololu для Mac ограничена.

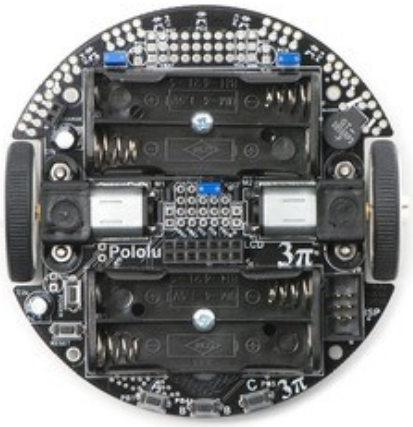
Для проведения исследований, возможно потребуются следующие принадлежности:

- Несколько больших листов белой бумаги
- Клейкая лента светлого цвета для того, чтобы скрепить листы в один большой лист
- 3/4" черная изолента, чтобы сделать линии для робота.

3.b. Включение 3pi

Перед началом работы с 3pi:

1. Нужно вставить четыре заряженные батареи AAA в держатели для батарей. Чтобы это сделать, нужно удалить ЖК-дисплей. Обратите внимание на ориентацию ЖК-дисплея (т.к. нужно потом правильно поставить его на место). После удаления ЖК- дисплея 3pi должен быть похож как на картинке справа.



2. Как только батареи установлены, установите ЖК-дисплей обратно. Удостоверьтесь, что каждый контакт входит в соответствующее гнездо.
3. Затем, нажмите кнопку "POWER" (расположена слева около ЖК-дисплея), чтобы включить 3pi. При включении должны светиться два синих индикатора питания на нижней стороне робота.

После выполнения шагов 1-3, ваш робот должен начать выполнять загруженную демонстрационную программу. Эта программа дает возможность управлять питанием робота и с помощью кнопки сброса «RESET» (расположенная рядом с «POWER») перезагружать программу.

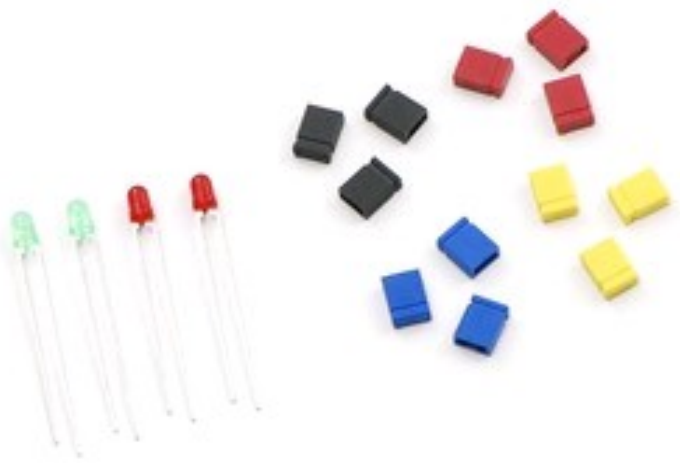
3.c. Использование демонстрационной программы

После включения 3pi запустится демонстрационная программа. Если вы слышите звуковой сигнал, но не видите текста на ЖК-дисплее, то возможно не отрегулирован контрастный потенциометр на нижней стороне платы 3pi. После калибровки робот должен успешно запуститься. Тогда нажмите кнопку B, чтобы попасть в главное меню. Нажмите C, чтобы перемещаться вперед или назад из главного меню, и нажимайте B, чтобы сделать выбор или выйти из предыдущего меню. Демонстрационная программа может выполнять семь различных действий, которые будут доступны из меню:

1. ***Battery:*** этот демонстрационный пример показывает напряжение батареи в милливольтах, которые должны быть выше 5000мВ (5.0 В) для полностью заряженных батарей. Удаление перемычки с ADC6 позволит отделить измерение напряжения батареи от аналогового входа.
2. ***LEDs:*** эта программа заставляет мигать красным и зеленым светодиодами на нижней стороне платы.
3. ***Trimpot:*** показывает положение потенциометра, который расположен на нижней стороне робота и выводит результаты в виде числа между 0 и 1023.
4. ***Sensors:*** показывает текущие значения с ИК-датчиков, используя гистограмму. Более крупные бары показывают более высокий коэффициент отражения.
5. ***Auto:*** удерживайте A или C, чтобы управлять двигателем на левой (правой) стороне, или если хотите, чтобы оба двигателя вращались, то удерживайте обе кнопки.
6. ***Music:*** демонстрирует способность 3pi воспроизводить музыку.
7. ***Timer:*** простой секундомер. Нажмите C, чтобы начать или остановить секундомер и, чтобы перезагрузить.

Исходный код для демонстрационной программы включен в библиотеку Pololu AVR C/C++ (в папке `examples\3pi-demo-program`).

3.d. Компоненты поставки



Как видно из картинки, в комплект вместе с роботом поставляются 12 перемычек, которые возможно потом понадобятся и 4 светодиода (два из них можно припаять на обратную сторону платы управления роботом)

4.Первый контакт с 3pi

4.a. Управление электропитанием

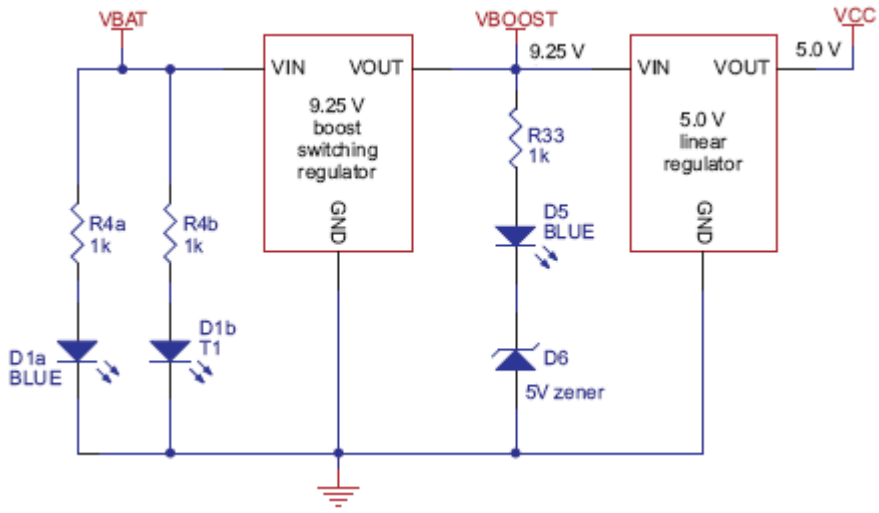
Этот подраздел посвящен управлению движением робота с помощью регуляторов. Такой принцип будет реализован в демонстрационных программах для езды по линии.

На борту нашего робота есть специальное устройство, ***регулятор напряжения***, он преобразовывает напряжение батареи в постоянное напряжение. В течение долгого времени, напряжение в 5 V было наиболее распространенным, используемым в цифровой электронике; его также называют уровнем TTL. Микроконтроллер и большая часть схемы в 3pi потребляют 5 V, таким образом, регулирование напряжения очень важно. Есть два основных типа регуляторов напряжения:

- ***Линейные регуляторы*** производят более низкое выходное напряжение, отдавая ненужную энергию в расход. Этот расточительный и неэффективный подход мы считаем не очень выгодным для конкретного случая (когда есть значительные различия между напряжениями входа и выхода, или для тех, которые требуют большого тока). Например, 15V напряжения батарей, которые отрегулированы к 5 V с линейным регулятором, потеряют две трети своей энергии в линейном регуляторе. Эта энергия превращается в тепло (нагрев компонентов), таким образом, линейные регуляторы часто нуждаются в больших теплоотводах, и их не стоит использовать в мощных системах.

- ***Переключающиеся регуляторы*** настраивают управление на высокую частоту, так что с помощью фильтрации можно было произвести поставку энергии с нужным напряжением. Пересылая поток электричества, с помощью переключения регуляторов этот способ является наиболее выгодным, чем линейные регуляторы, которые предназначены специально для высокого напряжения и больших изменений в нем. Кроме того, с помощью переключения регуляторов можно преобразовать низкие напряжения в более высокие! Ключевой компонент переключающегося регулятора – это ***катушка индуктивности***, которая хранит энергию и сглаживает поток; на 3pi, катушка индуктивности – это серый блок около поддерживающего шарика с маркировкой “100”.

Подсистему управления электропитанием, которая реализована в 3pi, показывает данная схема:

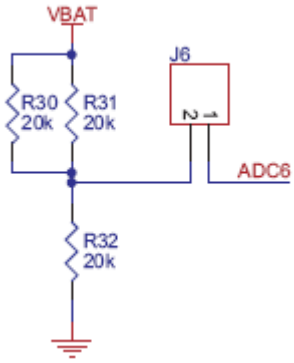


Напряжение батарей может измениться между 3.5 – 5.5 V (и даже к 6 V, если используются батареи (аккумуляторы) типа «alkalines»). Это означает, что невозможно просто отрегулировать напряжение до 5 V. Итак, в 3pi есть переключающийся регулятор, с помощью которого можно

добиться повышения напряжение батареи до 9.25 V (Vboost), и регулировать Vboost, а также понижать до 5 V (VCC). Вот для чего это необходимо. Vboost приводит в действие двигатели и ИК-светодиоды в датчиках линии, в то время как VCC используется для микроконтроллера и всех цифровых сигналов (т.к. 5V для них максимальное напряжение и если поднять, скажем, до 7V и не использовать регуляторы, то микроконтроллер может сгореть). Использование Vboost для двигателей и датчиков дает 3pi три уникальных преимущества перед типичными роботами, которые используют питание от батареи непосредственно:

- ***Во-первых***, более высокое напряжение значит больше энергии для двигателей, следовательно более высокая скорость.
- ***Во-вторых***, так как напряжение отрегулировано, двигатели будут управляться той же самой скоростью, в тот момент как напряжение будет понижаться от 5.5V до 3.5 V. Также вы можете использовать эту возможность, когда требуется калибровка для 90°-го поворота за определенное количество времени.
- ***В-третьих***, все пять из ИК - светодиодов могут быть приведены в действие последовательно так, чтобы они потребляли самое низкое количество энергии.

Другая интересная особенность этой системы состоит в том, что вместо того, чтобы постепенно исчерпать энергию как большинство роботов, 3pi будет работать при максимальной производительности, пока внезапно не выключится. Это может плохо закончится для робота. Таким образом, вы можете реализовать контроль напряжения батареи с помощью нехитрой схемы.



Данная схема (справа) контролирует напряжение батареи и конечно же встроена в 3pi. На ней три резистора, показанные на схеме справа, включают сепаратор напряжения, который производит напряжение, равное от 2/3 напряжения батареи, которое всегда будет ниже максимального напряжения аналогового входа микроконтроллера 5V. Например, в напряжении батареи 4.8V, напряжение батареи на порту ADC6 будет 3.2 V. Используя 10-битовое аналого-цифровое преобразование, 5V, после преобразования, будет соответствовать числу 1023, а 3.2 V числу 655. Чтобы преобразовать это в фактическое напряжение батареи, умножьте это число на 5000mV×3/2 и все это поделите на 1023. Данный принцип реализован в демонстрационной программе в функции `read_battery_millivolts_3pi()`:

```
unsigned int read_battery_millivolts_3pi()

{

      return readAverage(6,10)*5000L*3/2/1023;

}
```

4.c. Двигатель и коробка передач

Скорость вращения обычного электродвигателя обычно более тысяч вращений в минуту (оборот в минуту). ***Коробка передач*** – это система механизмов, которая выдает на выходе энергию большого вращающего момента малого быстродействия, которая намного больше подходит для управления роботом. Передаточное отношение ходовой части, используемое на 3pi, 30:1, что означает, что для каждых 30 поворотов вала двигателя, выходной вал поворачивается один раз.

Передаточное отношение :	**30:1**
Скорость	700 оборотов в минуту
Ток	60 миллиампер
Вращающий момент	6 унций·в
Поток	540 миллиампер

Характеристика	**Измерение в унциях, дюймах**	**Измерение в граммах, сантиметрах**
Радиус колеса	0.67 дюймов	18,961 см
Вес робота	7 унций (с батареями)	198,1 грамм
Производимая сила (2 двигателя), если двигаться вперед	2×6/0.67 = 18 унций	510,3 грамм

***** 1 дюйм = 2,54 см, 1 унция ≈ 28, 35 г**

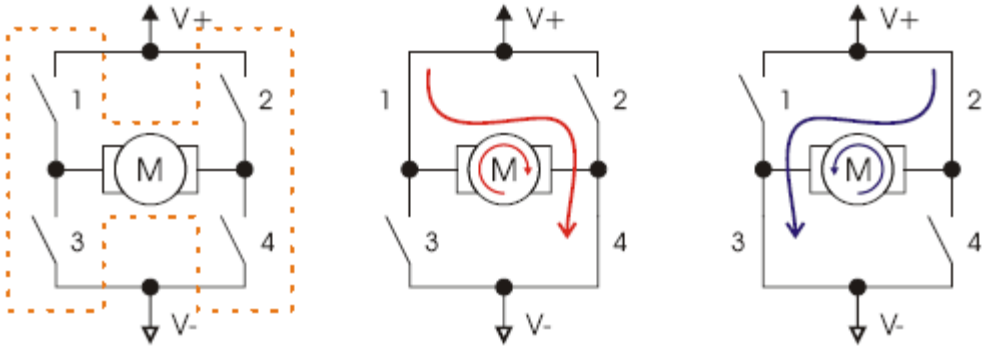
Просмотрев таблицу с данными о 3pi, сделаем вывод, что двигатели достаточно мощные, чтобы 3pi смог подняться по вертикальному наклону. Но данная работа ограничена трением шин: на достаточно крутом склоне скорость будет уменьшаться, в то время как наклон поверхности будет приблизительно 30-40°.

Управление скоростью двигателя

Важная особенность робота в том, что есть возможность регулировать скорость и направление передвижения робота по поверхности. Чтобы управлять направлением движения нужно лишь переключать полярность напряжения электродвигателей. Если у вас есть отдельно двигатели и 2-3 батареи, то можно провести небольшой эксперимент. Подключая двигатель к батареям с разными полярностями можно увидеть, что с изменением подключения направление движения колес меняется.

Но в роботе, такое переключение возможно с помощью Н-моста, который позволяет двигателю вращаться назад или вперед.

Вот диаграмма, которая показывает, как работает Н-мост:



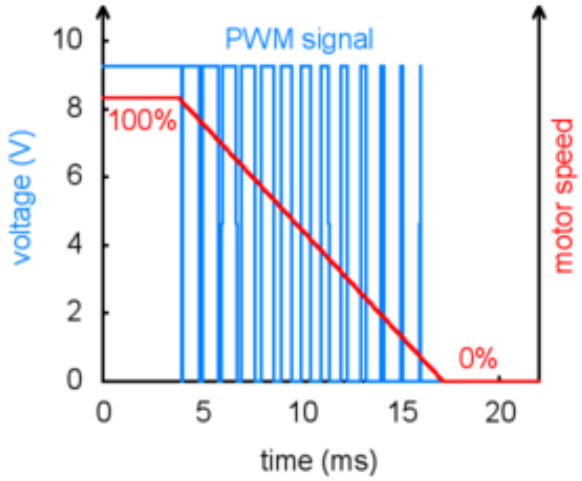
Пояснение*:** Если выключатели 1 и 4 закрыты (схема в центре), ток идет через двигатель слева направо, и мотор вращается вперед. Если же выключатели 2 и 3 закрыты, то двигатель вращается назад. H-мост может быть построен с механическими выключателями, но большинство роботов, включая 3pi, используют транзисторы, чтобы переключить поток с помощью электроники. H-мосты для обоих двигателей, установленных 3pi, встроены в микросхему – в ***драйвер двигателей, TB6612FNG. Эта схема подключена к портам микроконтроллера, которые и управляют выключателями через драйвер двигателей.

Мотор M1 управляются через порты PD5 и PD6:

PD5	PD6	1	2	3	4	M1
0	0	выкл.	выкл.	выкл.	выкл.	все моторы выключены (движения нет)
0	1	выкл.	вкл.	вкл.	выкл.	Движение вперед
1	0	вкл.	выкл.	выкл.	вкл.	Движение назад
1	1	выкл.	выкл.	вкл.	вкл.	все моторы выключены (движения нет)

Мотор M2 управляются через порты PD3 и PB3:

PD3	PB 3	1	2	3	4	M2
0	0	выкл.	выкл.	выкл.	выкл.	все моторы выключены (движения нет)
0	1	выкл.	вкл.	вкл.	выкл.	Движение вперед
1	0	вкл.	выкл.	выкл.	вкл.	Движение назад
1	1	выкл.	выкл.	вкл.	вкл.	все моторы выключены (движения нет)



Регулировка скорости достигнута благодаря быстрому переключению двигателей. Предположим, что PD6 принимает значение высокого логического уровня (в 5 V, также названным логической “1”), а порт PD5 между низким (0 V или “0”) и высоким. Драйвер двигателей переключится между направлением "вперед" и "стоп", заставляя M1 повернуться вперед со сниженной скоростью. Например, если PD6 будет в “1” две трети времени (67%-ый ***рабочий цикл***), то M1 будет вращаться с 67 % - ной мощностью от его максимальной скорости. Так как моторное напряжение – это серия пульса переменной ширины, то этот метод регулировки скорости называют ***широтно-импульсной модуляцией (PWM)***. В качестве примера PWM можно увидеть в таблице, где показано регулировка импульса от 100%-ого рабочего цикла до 0%, где частота вращения двигателя уменьшается с максимальной скорости до остановки.

В 3pi, регулировка скорости достигнута благодаря методу PWM, который можно реализовать с помощью средств микроконтроллера, которые связаны с внутренними таймерами Timer0 и Timer2. Это означает, что можно установить рабочий цикл PWM двух двигателей один раз и сохранять такое значение в течение всего цикла программы.

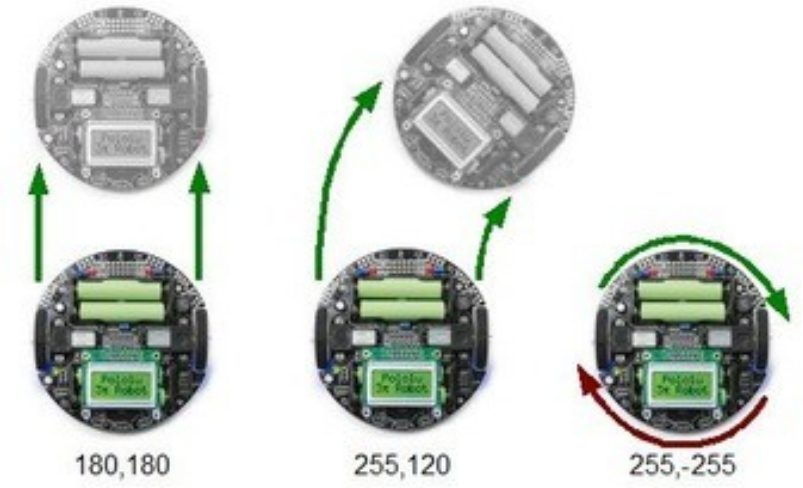
Функция `set_motors()` в библиотеке Pololu AVR позволяет устанавливать рабочий цикл, и эта функция использует 8-битовую точность: значение 255 соответствует 100%-ому рабочему циклу. Например, чтобы получить 67% на M1 и 33% на M2, вам понадобилось бы написать следующее:

```
set_motors(171,84);
```

Чтобы получить медленное уменьшение скорости двигателей с помощью PWM, понадобилось бы создать цикл с последовательным уменьшением параметров функции `set_motors()`.

Поворот с помощью дифференциального управления

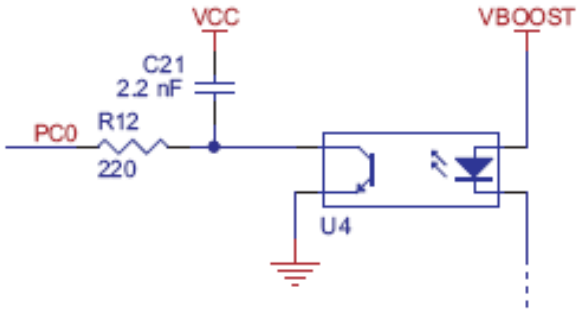
3pi имеет независимые двигатели и колесо на каждой стороне, что позволяет применить метод передвижения, названный ***дифференциальным управлением***. В предыдущем примере ***set_motors()***, левое колесо будет вращаться быстрее чем правое, так что робот будет ехать направо. Различие в скоростях определяет, насколько будет резкий поворот. Также, поворота можно достигнуть вращая одно колесо вперед, а другое назад. Такое вращение особенно эффективно для маневра робота с круглым корпусом.



5.d. Цифровые входы и датчики

У микроконтроллера есть входы, которые могут формироваться как цифровые (на вход либо “1” либо ”0”). Вот схема для одного из входов кнопки:

Если говорить о 3pi, то у него 5 цифровых ИК-датчиков. Они позволяют роботу узнавать темная поверхность или светлая. Внизу представлена простейшая схема датчика, который подключен к поту PC0 и находится внизу платы управления слева:



Элемент датчика, который получает информацию - это фототранзистор, показанный слева половине U4, который связан последовательно с конденсатором C21. Отдельная связь принуждает через резистор R12 прикреплять порт PC0. Также следует помнить, что цифровые входы AVR могут повторно формироваться как цифровые выходы. Цифровой выход представляет напряжение 5V или 0V, в зависимости от того, установлен ли он в 1 или 0 программой. Напряжение на выходе поочередно устанавливается в 5V и 0V. А конденсатор хранит временный результат, так, чтобы когда вход читался как “1”, до тех пор пока большая часть сохраненного напряжения не перешла через фототранзистор.

Когда цифровой вход остается в “1”, тогда можно говорить о том, что сенсор находится на белой поверхности, и “0” на черной. Функция `read_line_sensors()` в библиотеке Pololu AVR переключает порт как описано выше и возвращает время для каждого из этих пяти датчиков. Вот упрощенная версия кода функции, которая считывает сигналы с датчиков:

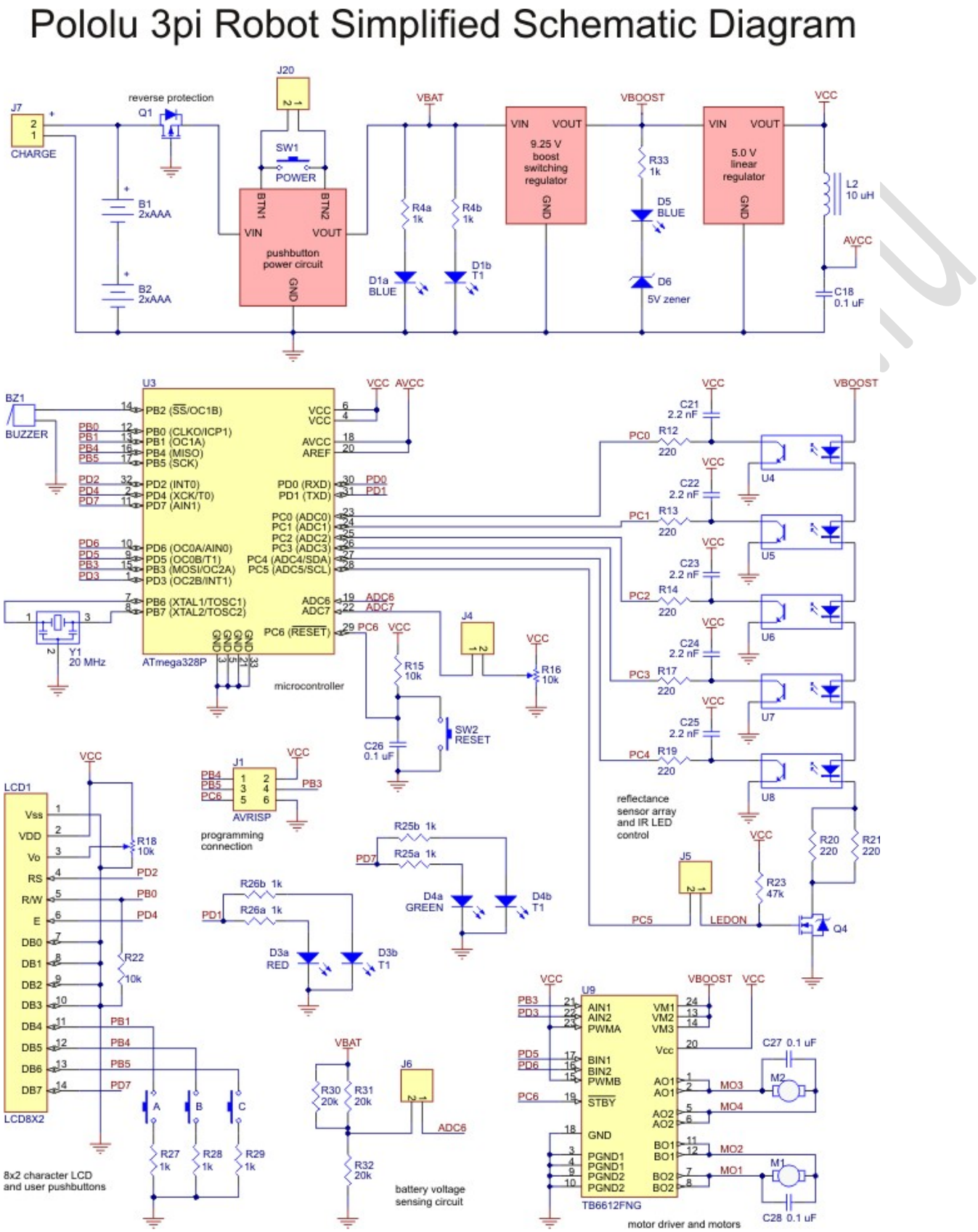
```
time = 0;

last_time = TCNT2;

while (time < _maxValue)

{

// Keep track of the total time.

// This implicity casts the difference to unsigned char, so

// we don't add negative values.

unsigned char delta_time = TCNT2 - last_time;

time += delta_time;

last_time += delta_time;

// continue immediately if there is no change

if (PINC == last_c)

continue;

// save the last observed values
```

```
last_c = PINC;

// figure out which pins changed

for (i = 0; i < _numSensors; i++)

{

if (sensor_values[i] == 0 && !(*_register[i] & _bitmask[i]))

sensor_values[i] = time;

}

}
```

Эту часть кода можно найти в файле `src\PololuQTRSensors\PololuQTRSensors.cpp`. В коде используется таймер TCNT2, который является специальным регистром в AVR и используется для того, чтобы подсчитать непрерывные сигналы с конденсатора, с шириной шага 0.4 мс. После обнаружения перехода от 1 до 0 на одном из датчиков, код определяет, какой датчик изменил свое значение и записывает эти данные в массив значений `sensor_values[i]`.

5.e. 3pi упрощенная схема

Чтобы лучше понимать работу 3pi следует изучить его схему:



6. Программирование Ваш 3pi

Чтобы заставить робота 3pi выполнять другие программы понадобится USB - программатор AVR. Чтобы запрограммировать вашего робота понадобится программатор и специальное программное обеспечение. Для пользователей Windows мы рекомендуем:

1. **WinAVR**, свободный, общедоступный набор средств разработки для микроконтроллеров семейства AVR, включая GNU компилятор GCC для C/C ++.
2. **AVR Studio**, свободная интегрированная среда проектирования Atmel, которая работает со свободно распространяемым компилятором GCC WinAVR C/C ++. AVR Studio включает программное обеспечение для AVR ISP, которое позволит вам загружать свои программы в 3pi.
3. **Pololu AVR C/C ++ Library**, которая облегчает использование 3pi.
4. Драйвера для программатора USB Pololu.

Предупреждение: не пытайтесь запрограммировать 3pi, если его батареи истощены или не заряжены (удостоверьтесь, что аккумуляторы полностью заряжены).

6.a. Загрузка и Монтаж C/C ++ Библиотека

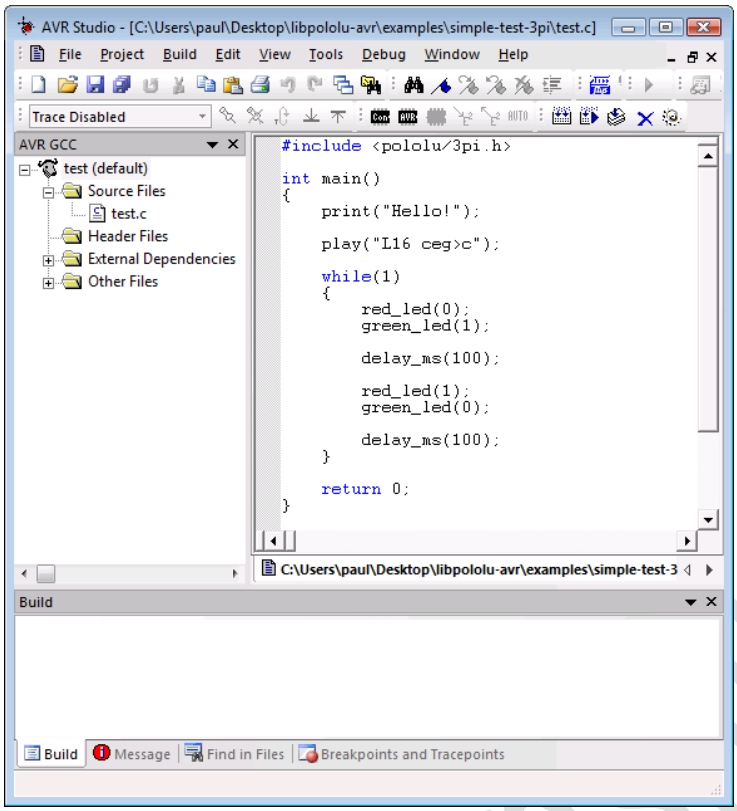
Библиотека Pololu C/C ++ AVR облегчает использование и изучение преимуществ 3pi; библиотека используется во всех примерах следующих разделов. Мы рекомендуем загрузить библиотеку. Вы можете загрузить инсталлятор, который автоматически установит библиотеки 3pi.

6.b. Компиляция простой программы

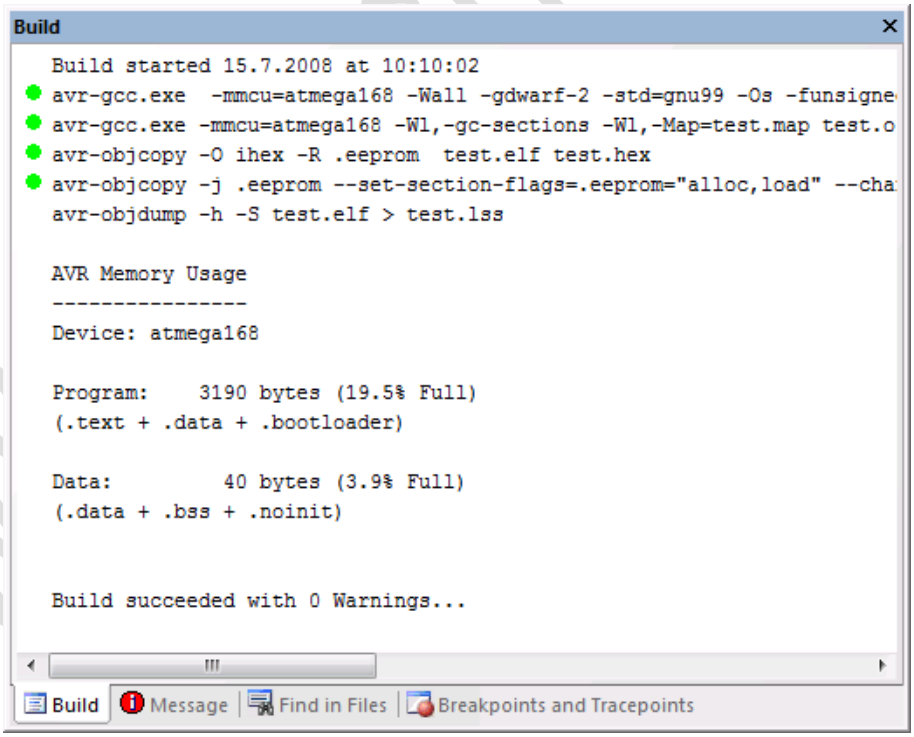
Очень простая демонстрационная программа для 3pi доступна в папке `examples\atmega328p\simple-test` (или `examples\atmega168\simple-test` если у Вас есть более старый робот 3pi с ATmega168). Папка `examples` может быть найдена в корневом каталоге установки библиотеки Pololu AVR C/C++, которая является `C:\libpololu-avr\` по умолчанию. Эта демонстрационная программа использует несколько основных команд из библиотеки Pololu AVR:

```
#include <pololu/3pi.h>

int main()
{
    print("Hello!");

    play("L16 ceg>c");

    while(1)
    {
        red_led(0);
        green_led(1);

        delay_ms(100);

        red_led(1);
        green_led(0);

        delay_ms(100);
    }

    return 0;
}
```

В папке `simple-test` найдите и щелкните два раза на файле `simple-test.aps`, и проект должен открыться автоматически в AVR Studio.



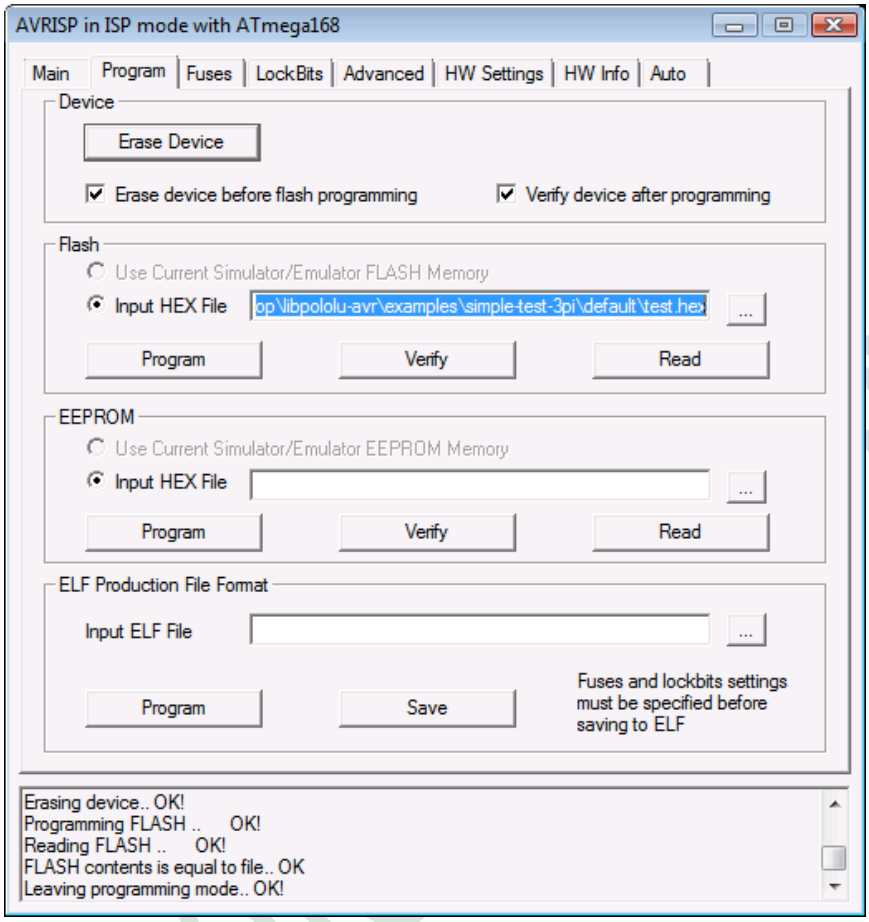
Чтобы собрать программу, нажимаем Build или **F7**. Предупреждения или ошибки будут обозначены желтыми и красными точками, как на картинке, указанной ниже. Если сборка программы проходит успешно, появится сообщение “ Build succeeded with 0 Warnings… ”, в результате должен создаться файл `test.hex` в папке `examples\atmegaxx8\simple-test\default`.



Соедините программатор с компьютером и с портом ISP 3pi, и включите 3pi. Если Вы используете программатор USB Pololu AVR, то должен загореться желтый светодиод, что свидетельствует о том, что программатор готов.

Выберите **Tools > Program AVR > Connect,** чтобы соединиться с программатором. Для программатора USB Pololu AVR или другого программатора USB Orangutan, есть варианты по умолчанию “STK500 или AVRISP”, которые подходят для нашего программатора, затем нажимаем на **Connect**.

Чтобы загрузить `test.hex` во флэш-память мы будем использовать AVRISP. Затем “Program” и программа загрузится на 3pi.



Если процесс прошел успешно, то вы должны услышать короткую мелодию и увидеть сообщение “Hello!” на ЖК-дисплее, и должны замигать светодиоды. Если Вы слышите мелодию и видите, что светодиод мигает, но ничто не появляется на ЖК-дисплее, удостоверьтесь, что ЖК-дисплей правильно подключен к 3pi, и попытайтесь отрегулировать контраст, используя маленький потенциометр на нижней стороне 3pi (около поддерживающего шарика).

7. Проект №1 в качестве примера

7.a. О программе: езда по линии

Теперь, когда Вы знаете как собрать простую программу для 3pi, то пора узнать как робот справляется с более сложными задачами. В этом проекте в качестве примера мы покажем вам, как 3pi может следовать по черной линии на белом фоне, с помощью своих датчиков и двигателей. Задача о следовании по линии – это самная простая и фундаментальная задача в программировании роботов.

7.b. Алгоритм следования по линии

Простая программа для езды по линии находится в папке `examples\atmegaxx8\3pi-linefollower`. Код демонстрирует множество различных особенностей 3pi, включая датчики линии, двигатели, ЖК-дисплей и пищалку. У программы есть две фазы.

Первая фаза программы - инициализация и фаза калибровки, которая обрабатывается функцией ***intitialize()***. Эта функция вызывается однажды, в начале программы и ее цель заботится о следующих шагах:

1. ***pololu_3pi_init*** **(*2000*)** - настройка 3pi, с прерыванием датчика в 2000×0.4мс = 800мс. Это означает, что значение датчика изменятся от 0 (абсолютно белая поверхность) к 2000 (абсолютно черная поверхность), где величина 2000 указывает, что конденсатор датчика отработал 800 мс.
2. Показатель напряжения батареи, который был получен с помощью функции ***read_battery_millivolts()***. Контроль напряжения очень важная способность для робота. Ведь всегда необходимо знать сколько времени робот еще может проработать.
3. ***Калибровка датчиков.*** Это происходит следующим образом: робот поворачивается направо и налево и остается на линии (получает минимальные и максимальные значения от датчиков) с помощью функции ***calibrate_line_sensors()***. Минимальные и максимальные значения сохраняются в RAM-памяти. Это позволяет функции ***read_line_sensors_calibrated()*** вернуться к возвращаемым значениям, которые могут изменяться от 0 до 1000 для каждого датчика. Функция ***read_line()*** функция, используемая в коде также, зависит от результатов калибровки.
4. Значения откалиброванных датчиков отображаются в гистограмме. Эту возможность демонстрирует функция ***lcd_load_custom_character()*** вместе с ***print_character()***, чтобы точно знать, работают ли датчики линии должным образом перед началом работы робота.
5. Чтобы контролировать действия робота, нужно пользоваться кнопкой “B”. Иначе, если этого не сделать, то вы можете просто не поймать его. Для этого используется функция ***button_is_pressed()***, которая дает роботу команду **“Ждать”** до тех пор, пока кнопка “B” не была нажата.

Во второй фазе программы, 3pi установит частоту вращения двигателя (благодаря данным с датчиков). Эти шаги, далее, контролируют движение робота, чтобы тот смог постоянно оставаться на линии. Следующие шаги происходят в бесконечном цикле while(1), который будет повторяться много раз, пока робот не будет выключен или перезагружен.

Функция ***read_line()*** берет на себя чтение значений с датчиков и возвращает оценку положения робота относительно линии в виде числа между 0 и 4000. Значение 0 означает, что линия слева от датчика, а значение 1000 показывает, что линия непосредственно находится под датчиком, 2000 - линия непосредственно находится под датчиком 2 и так далее.

1. Значения, возвращенные функцией ***read_line()***, разделены на три группы:

   0–1000: робот далеко справа от линии. В этом случае, чтобы покинуть эту позицию, мы устанавливаем правильную частоту вращения двигателя в 100 и левую частоту вращения двигателя в 0.

   1000–3000: робот возможно на линии. В этом случае мы заставляем оба двигателя ускоряться со значением 100, т.е. двигаться прямо вперед.

   3000–4000: робот далеко слева от линии. В этом случае мы поворачиваемся резко направо, устанавливая частоту вращения правого двигателя в 0 и частоту вращения левого двигателя в 100.

7.c. Усовершенствованная версия программы езды по линии. ПИД-регуляторы

Усовершенствованная версия программы для следования по линии для 3pi доступна в папке `examples\atmegaxx8\3pi-linefollower-pid`

Метод, используемый в этой программе, известный как управление PID, решает проблему резкого движения робота. PID использует непрерывные функции, чтобы вычислить частоты вращения двигателей, так, чтобы неровное движение, при сходе с дистанции, было более мягким. ***PID (Proportional Integral Differential)***, что обозначает ***пропорциональная величина, интегральная величина, производная величина***; это три входных значения, используемые в простой формуле, чтобы вычислить скорость для робота.

Пропорциональная величина характеризует следующее: если робот будет точно находиться на линии, то значение будет равно 0. Если робот будет слева от линии, то пропорциональная величина будет положительным числом, а справа от линии - отрицательным числом.

Интегральное значение записывает историю движения робота: это - сумма всех значений пропорциональной величины, которые регистрируются, с того момента как робот начал работать.

Производная величина – фиксирует изменения пропорционального значения. Вычисляется как разность последних двух пропорциональных значений.

Вот раздел кода, который вычисляет входные значения PID:

```
// Get the position of the line.  Note that we *must* provide
// the "sensors" argument to read_line() here, even though we
// are not interested in the individual sensor readings.


unsigned int position = read_line(sensors,IR_EMITTERS_ON);

// The "proportional" term should be 0 when we are on the line.
int proportional = ((int)position) - 2000;

// Compute the derivative (change) and integral (sum) of the
// position.
int derivative = proportional - last_proportional;
integral += proportional;

// Remember the last position.
last_proportional = proportional;
```

Каждое из этих входных значений обеспечивает различный вид информации. Следующий шаг - простая формула, которая комбинирует все значения в одну переменную, которая используется для определения частоты вращения двигателей:

```
// Compute the difference between the two motor power settings,
// m1 - m2.  If this is a positive number the robot will turn
// to the right.  If it is a negative number, the robot will
// turn to the left, and the magnitude of the number determines
// the sharpness of the turn.
int power_difference = proportional/20 + integral/10000 + derivative*3/2;

// Compute the actual motor settings.  We never set either motor
// to a negative value.
const int max = 60;
if(power_difference > max)
    power_difference = max;
if(power_difference < -max)
    power_difference = -max;

if(power_difference < 0)
    set_motors(max+power_difference, max);
```

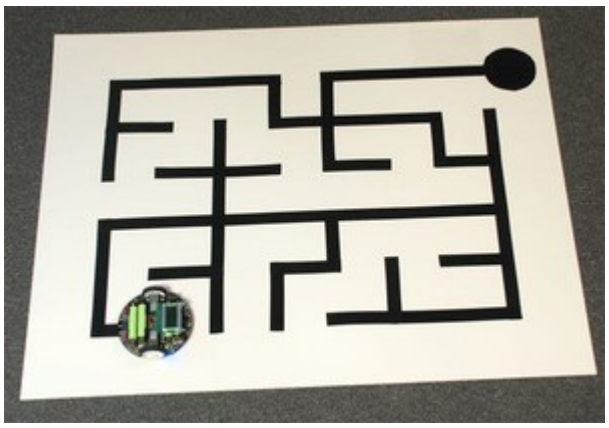
```
else
    set_motors(max, max-power_difference);
```

Значения 1/20, 1/10000, и 3/2 являются корректируемыми параметрами, которые определяют, как 3pi будет реагировать на линию. Они были выбраны произвольно. В целом увеличение этих параметров реализовано в переменной ***power_difference***. Большие параметры, вызывают более сильные реакции, в то время как уменьшение их, делает реакции более слабыми. В этом примере реализована возможность, которая дает двигателям максимальную скорость 100, которая является безопасным значением. Как только скорректированы параметры (для скорости 100), попытайтесь увеличить еще и скорость.

8.

Следующий уровень программирования роботов – это решение задачи перемещения по лабиринту. В этом разделе робот 3pi научится перемещаться по путям с крутыми поворотами, справляться с тупиками и пересечениями. Советуем вам сделать полигон для робота, который будет состоять из черной линии. Он будет исполнять роль лабиринта. В таких соревнованиях робот должен перемещаться как можно быстрее вдоль линии от старта до финиша (цели), отслеживая пересечения, которые встречаются на его пути. Обычно, в таких соревнованиях роботу будет дано несколько попыток для прохождения лабиринта.

Одна особенность у данного алгоритма это отсутствие циклов. Т.е. у робота не будет возможности повторно посетить любую точку на лабиринте без возвращения той же дорогой. Решение этого типа лабиринта намного проще, чем решение циклического лабиринта, так как такая простая стратегия позволяет исследовать весь лабиринт. Мы будем говорить о той стратегии в следующем разделе.



В лабиринтах будут использоваться только прямые линии, что сделает решение задачи прохождения более простой и понятной для первого знакомства. Здесь, главной целью является изучение базовых алгоритмов. А, как известно, изучение чего-то начинается с простого.

8.b. Работа с многофайловыми C - проектами в AVR Studio.

Исходный код для решение задачи прохождения лабиринта доступен в папке `examples\atmegaxx8\3pi-mazesolver`.

Примечание: Arduino-совместимая версия этой программы может быть загружена как часть Pololu Arduino Library, где Arduino-проект представляет собой единственный файл.

Эта программа более сложная, чем те примеры, которые были представлены ранее. И с целью упрощения понимания концепции проект был разделен на несколько файлов, что существенно облегчает работу с проектом. Например, файл ***turn.c*** содержит только единственную функцию, которая позволяет выполнять повороты в пересечениях:

```
#include <pololu/3pi.h>

// Turns according to the parameter dir, which should be 'L', 'R', 'S'
// (straight), or 'B' (back).
void turn(char dir)
{
```

```
    switch(dir)
    {
    case 'L':
        // Turn left.
        set_motors(-80,80);
        delay_ms(200);
        break;
    case 'R':
        // Turn right.
        set_motors(80,-80);
        delay_ms(200);
        break;
    case 'B':
        // Turn around.
        set_motors(80,-80);
        delay_ms(400);
        break;
    case 'S':
        // Don't do anything!
        break;
    }
}
```

Чтобы получить доступ к этой функции от других файлов C, мы нуждаемся в “заголовочном файле”, который называют ***turn.h***. Заголовочный файл просто содержит единственную строку кода:

```
void turn(char dir);
```

Эта строчка объявляет функцию ***turn()***. Чтобы получить доступ к данной строчке, в каждый файл должна быть добавлена следующая строка:

```
#include "turn.h"
```

Файл ***segment.c***, также содержит единственную функцию, ***follow_segment()***, которая будет управлять движением 3pi по линии, пока он не достигнет пересечения или финиша. Вот функция:

```
void follow_segment()
{
    int last_proportional = 0;
    long integral=0;

    while(1)
    {
    // Normally, we will be following a line.  The code below is
    // similar to the 3pi-linefollower-pid example, but the maximum
    // speed is turned down to 60 for reliability.

    // Get the position of the line.
        unsigned int sensors[5];
        unsigned int position = read_line(sensors,IR_EMITTERS_ON);

    // The "proportional" term should be 0 when we are on the line.
        int proportional = ((int)position) - 2000;
```

```
    // Compute the derivative (change) and integral (sum) of the
    // position.
        int derivative = proportional - last_proportional;
        integral += proportional;

   // Remember the last position.
        last_proportional = proportional;

   // Compute the difference between the two motor power settings,
   // m1 - m2.  If this is a positive number the robot will turn
   // to the left.  If it is a negative number, the robot will
   // turn to the right, and the magnitude of the number determines
   // the sharpness of the turn.
        int power_difference = proportional/20 + integral/10000 +
derivative*3/2;

   // Compute the actual motor settings.  We never set either motor
   // to a negative value.
        const int max = 60; // the maximum speed
        if(power_difference > max)
            power_difference = max;
        if(power_difference < -max)
            power_difference = -max;

        if(power_difference < 0)
            set_motors(max+power_difference,max);
        else
            set_motors(max,max-power_difference);

        // We use the inner three sensors (1, 2, and 3) for
        // determining whether there is a line straight ahead, and the
        // sensors 0 and 4 for detecting lines going to the left and
        // right.

        if(sensors[1] < 100 && sensors[2] < 100 && sensors[3] < 100)
        {
            // There is no line visible ahead, and we didn't see any
            // intersection.  Must be a dead end.
            return;
        }
        else if(sensors[0] > 200 || sensors[4] > 200)
        {
            // Found an intersection.
            return;
        }

    }
}
```

Процедура помещения файлов C и заголовочных файлов в проект AVR Studio очень проста. В левом столбце экрана есть раздел **“Исходные файлы”** и **“Заголовочные файлы”**. Щелкните правой кнопкой мыши и выберите опцию добавления или удаления файлов из списка. По завершении работы над проектом, AVR Studio автоматически скомпилирует все файлы C, которые помещены в проект и в результате появится единственный .hex файл.

8.c. Реализация принципа «левой руки» для прохождения лабиринта

Правило ***«левой руки»*** состоит в том, что робот идет, постоянно «держась рукой» за левую стену. При этом на каждом шаге выбирается левый из существующих коридоров. Если найден коридор слева по ходу движения робота, то он выбирает этот коридор (поворачивает на 90^0 влево).

Если на этом шаге коридор слева отсутствует, но найден коридор впереди, робот идет вперед. Если нет коридоров слева и спереди, но есть коридор справа, робот поворачивает на 90^0 направо.

Если коридоров нет (робот оказался в тупике), то он поворачивает на 180^0 и идет в обратном направлении. Происходит поворот на 180^0. Для нахождения выхода из лабиринта точка входа в него не должна находиться на пути зацикленного маршрута. В этом случае, если на каждом шаге, при выборе самого левого из возможных коридоров путь замыкается и робот возвращается в исходную точку, то он снова и снова будет идти этим маршрутом, и не сможет найти выход. Но робот сам по себе не сможет попасть в такой цикл.

Далее программа:

```
// This function decides which way to turn during the learning phase of
// maze solving.  It uses the variables found_left, found_straight, and
// found_right, which indicate whether there is an exit in each of the
// three directions, applying the "left hand on the wall" strategy.
char select_turn(unsigned char found_left, unsigned char found_straight,
unsigned char found_right)
{
    // Make a decision about how to turn.  The following code
    // implements a left-hand-on-the-wall strategy, where we always
    // turn as far to the left as possible.
    if(found_left)
        return 'L';
    else if(found_straight)
        return 'S';
    else if(found_right)
        return 'R';
    else
        return 'B';
}
```

8.d. Loop(s)

Стратегия данной программы выражена в файле ***solve.c***. Также важно еще отслеживать путь робота, который будет определяться как массив, который состоит из 100 значений; это будут значения, используемые в функции ***turn()***. Мы также должны отслеживать текущую длину пути для того, чтобы знать, куда поместить значения в массиве.

```
char path[100] = "";
unsigned char path_length = 0; // the length of the path
```

Наша функция ***main loop()*** содержится в функции ***maze_solve()***, которая вызывается после калибровки в файле ***main.c***. Эта функция выполняет два главных цикла – первый, который управляет решением лабиринта. Второй цикл – это цикл, который управляет решением лабиринта повторно. Вот код:

```
// This function is called once, from main.c.
void maze_solve()
{
    while(1)
    {
```

```
        // FIRST MAIN LOOP BODY
        // (when we find the goal, we use break; to get out of this)
    }

    // Now enter an infinite loop - we can re-run the maze as many
    // times as we want to.
    while(1)
    {
        // Beep to show that we finished the maze.
        // Wait for the user to press a button...

        int i;
        for(i=0;i<path_length;i++)
        {
            // SECOND MAIN LOOP BODY
        }

        // Follow the last segment up to the finish.
        follow_segment();

        // Now we should be at the finish!  Restart the loop.
    }
}
```

Первый цикл должен решать, как повернуть, и как сделать запись поворота в массив. Чтобы передать правильные аргументы в ***select_turn()***, мы должны точно знать, по какому принципу робот пересекает перекрестки. Обратите внимание на то, что есть специальное исключение для того, чтобы найти конец лабиринта. Следующий код работает вполне прилично, по крайней мере на медленных скоростях, которые мы используем:

```
// FIRST MAIN LOOP BODY
follow_segment();

// Drive straight a bit.  This helps us in case we entered the
// intersection at an angle.
// Note that we are slowing down - this prevents the robot
// from tipping forward too much.
set_motors(50,50);
delay_ms(50);

// These variables record whether the robot has seen a line to the
// left, straight ahead, and right, whil examining the current
// intersection.
unsigned char found_left=0;
unsigned char found_straight=0;
unsigned char found_right=0;

// Now read the sensors and check the intersection type.
unsigned int sensors[5];
read_line(sensors,IR_EMITTERS_ON);

// Check for left and right exits.
```

```
if(sensors[0] > 100)
    found_left = 1;
if(sensors[4] > 100)
    found_right = 1;

// Drive straight a bit more - this is enough to line up our
// wheels with the intersection.
set_motors(40,40);
delay_ms(200);

// Check for a straight exit.
read_line(sensors,IR_EMITTERS_ON);
if(sensors[1] > 200 || sensors[2] > 200 || sensors[3] > 200)
    found_straight = 1;

// Check for the ending spot.
// If all three middle sensors are on dark black, we have
// solved the maze.
if(sensors[1] > 600 && sensors[2] > 600 && sensors[3] > 600)
    break;

// Intersection identification is complete.
// If the maze has been solved, we can follow the existing
// path.  Otherwise, we need to learn the solution.
unsigned char dir = select_turn(found_left, found_straight, found_right);

// Make the turn indicated by the path.
turn(dir);

// Store the intersection in the path variable.
path[path_length] = dir;
path_length ++;

// You should check to make sure that the path_length does not
// exceed the bounds of the array.  We'll ignore that in this
// example.

// Simplify the learned path.
simplify_path();

// Display the path on the LCD.
display_path();
```

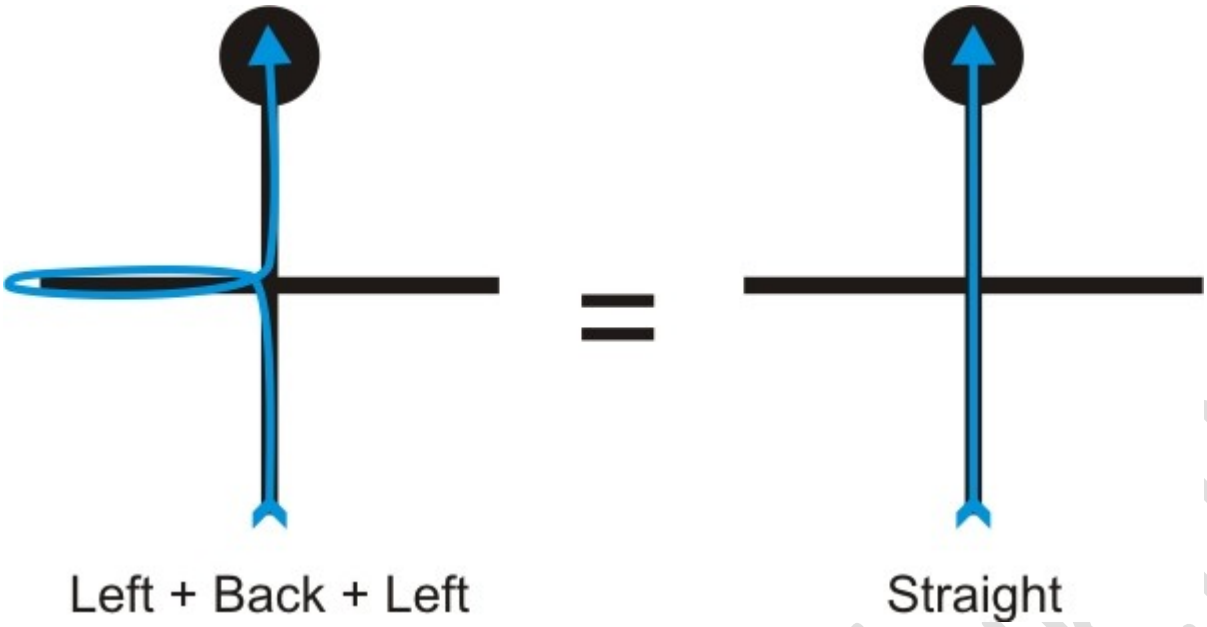
Мы обсудим требования к функции ***simplify_path()*** в следующей главе. Но перед этим давайте посмотрим на второй главный бесконечный цикл, который весьма простой. Все, что мы делаем, это двигаемся к следующему пересечению и повороту, согласно нашим подсчетам. После выполнения последнего зафиксированного поворота робот будет недалек от конца, это помогает определить функция ***follow_segment()***.

```
// SECOND MAIN LOOP BODY
follow_segment();
```

```
// Drive straight while slowing down, as before.
set_motors(50,50);
delay_ms(50);
set_motors(40,40);
delay_ms(200);

// Make a turn according to the instruction stored in
// path[i].
turn(path[i]);
```

После каждого поворота длина зафиксированного пути увеличивается на 1. Если в лабиринте, например, будет длинный зигзагообразный проход без выходов со стороны, то будет видно, что последовательность как ‘RLRLRLRL’ появляется на ЖК-дисплее робота.

Считайте последовательность “LBL”, где встречается. Это происходит, если есть левый поворот, который отклоняет нашего робота от прямого пути и немедленно приводит к тупику. После поворота на 90 °, затем на 180 °, и опять на 90 ° робот, в результате снова возвращается на верный путь. Путь может быть упрощен (может вообще не быть поворотов): единственный ‘S’. Следующая диаграмма показывает два функционально эквивалентных пути от начала до конца:



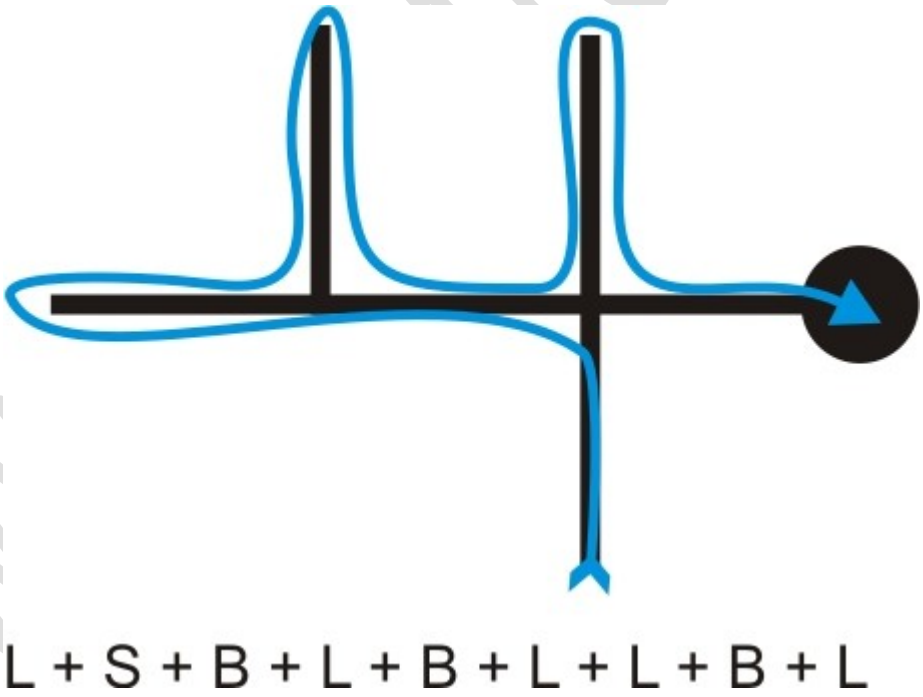
Другой пример - Т-пересечение с тупиком слева: ‘LBS’. Повороты - 90 °, 180 ° и 0°. Тогда последовательность будет заменена на единственное ‘R’.

Фактически, всякий раз, когда у нас есть последовательность как ‘xBx’, мы можем заменить все три поворота поворотом, соответствующим полному углу, устраняя разворот и ускоряя решение лабиринта. Вот код, который позволяет решать поставленную задачу:

```
// Path simplification.  The strategy is that whenever we encounter a
// sequence xBx, we can simplify it by cutting out the dead end.  For
// example, LBL -> S, because a single S bypasses the dead end
// represented by LBL.
void simplify_path()
{
    // only simplify the path if the second-to-last turn was a 'B'
    if(path_length < 3 || path[path_length-2] != 'B')
        return;

    int total_angle = 0;
    int i;
    for(i=1;i<=3;i++)
    {
        switch(path[path_length-i])
        {
        case 'R':
            total_angle += 90;
            break;
        case 'L':
            total_angle += 270;
            break;
        case 'B':
            total_angle += 180;
            break;
        }
    }

    // Get the angle as a number between 0 and 360 degrees.
```

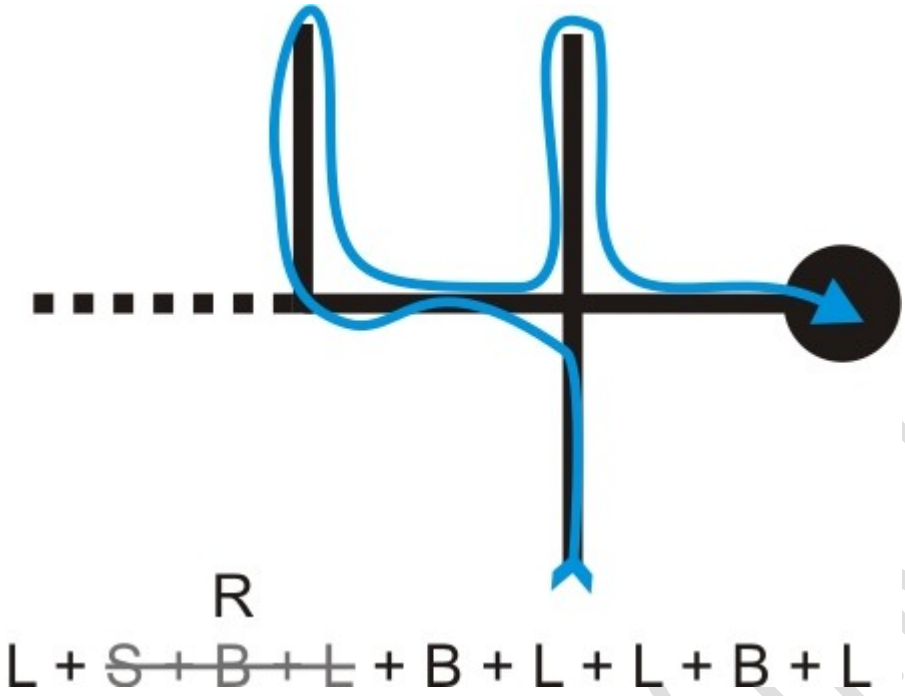
```
    total_angle = total_angle % 360;

    // Replace all of those turns with a single one.
    switch(total_angle)
    {
    case 0:
        path[path_length - 3] = 'S';
        break;
    case 90:
        path[path_length - 3] = 'R';
        break;
    case 180:
        path[path_length - 3] = 'B';
        break;
    case 270:
        path[path_length - 3] = 'L';
        break;
    }

    // The path is now two steps shorter.
    path_length -= 2;
}
```

Замечание: в коде есть последовательности, с которыми никогда не должен сталкиваться робот, который поворачивается направо, как ‘RBR’. Этот путь, согласно коду, должен быть заменен на ‘S’.



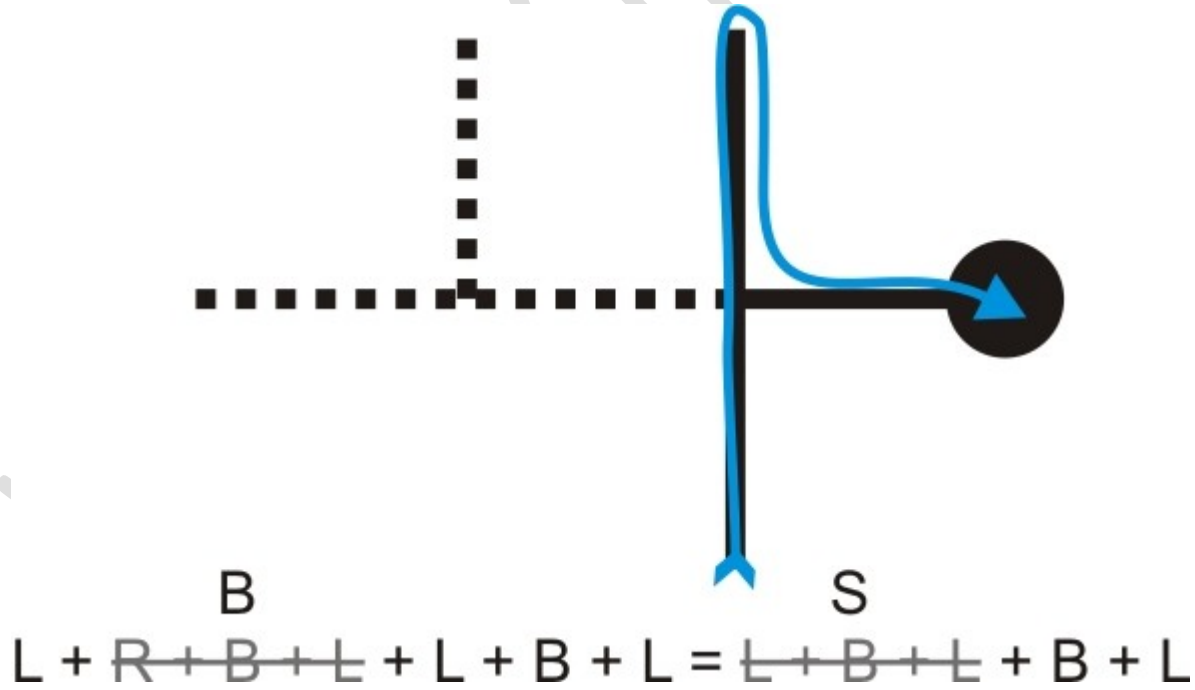
Вышеупомянутый алгоритм на картинке (вверху) позволяет полностью исследовать лабиринт при поиске конца лабиринта. Наша цель состоит в том, чтобы теперь уменьшить количество шагов в этом списке, чтобы добиться наикратчайшего пути от начала до конца, избавляясь от всех тупиков. Проблема состоит в том, чтобы выполнить это сокращение, используя минимальное количество памяти. Это достаточно сложная задача, которую нам придется разобрать все по шагам:

1. сократите первый тупик, поскольку его можно спокойно вычислить



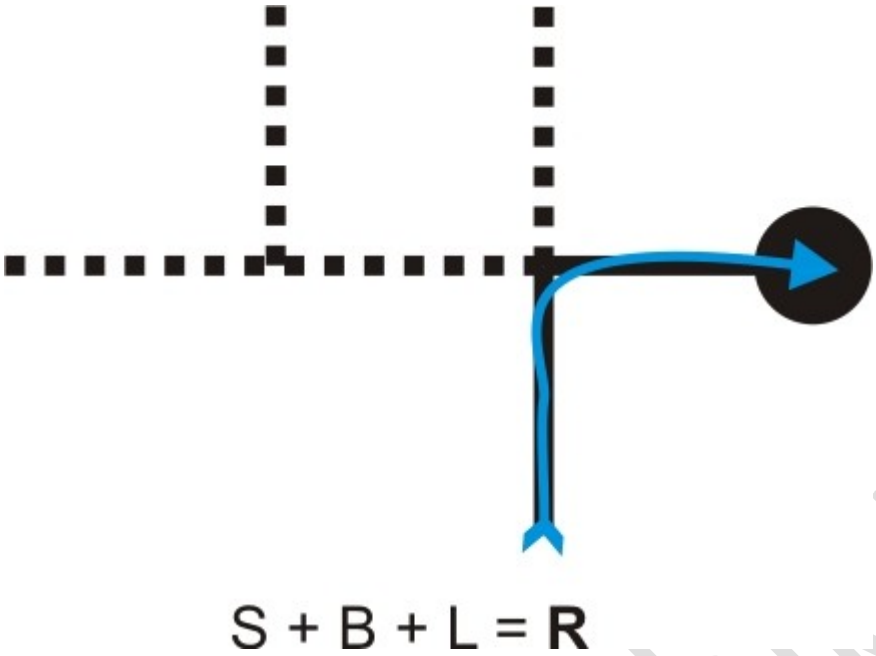
Когда мы встречаемся с первым пересечением после нашего первого предыдущего действия, мы знаем, что мы в тупике, который может быть удален из нашего списка действий. В этом случае, наша новая последовательность выглядит так ‘SBL’, и схема показывает, что эта последовательность может быть упрощена в единственный правый поворот ‘R’.

2. сократим остальную часть этого тупикового ответвления



3. затем заканчиваем с последовательностью 'RBL', которую можно уменьшить до ‘B’, и объединим это со следующим действием, чтобы получилась последовательность ‘LBL’, которую уменьшим до ‘S’.

4. сократим заключительное тупик, чтобы оставить кратчайший путь



S + B + L = **R**

Последний тупик дает нам последовательность ‘SBL’, который уменьшает до sigle правого поворота ‘R’. Наш список действия - теперь просто ‘R’ и представляет кратчайший путь от начала до конца.

Список произведенных действий:

1. L
2. LS
3. LSB
4. LSBL => LR *(сокращение происходит здесь)*
5. LRB
6. LRBL => LB *(сокращение происходит здесь)*
7. LBL => S *(сокращение происходит здесь)*
8. SB
9. SBL => R *(сокращение происходит здесь)*

8.f. Улучшение кода

После того, как вы разобрались с задачей прохождения лабиринта, мы попытаемся улучшить нашу программу.

1. Увеличим скорость движения по линии.
2. Улучшение управления с помощью PID.
3. Увеличение скорости.
4. Опознание ситуаций, когда робот потерялся.
5. Корректировка скорости.

Основным решением будет реализация расчета длины каждого сегмента пути, пройденного роботом. В коде, мы бы обнуляли бы таймер тогда, когда робот будет на пересечении и затем снова бы восстановил бы его, когда 3pi проходит на следующем пересечении. Поскольку программа хранит в массиве количество посещаемых пересечений, она также будет хранить и длину пройденного сегмента пути:

```
{L,                 S,                  S,                  R,                  L,...}
{3, 3, 6, 5, 8,...}
```

Нижний массив сохраняет значения пройденного пути там, где робот проходил пересечения. Они показывают степень длительности (т.е. как долго робот преодолевал путь от предыдущего до последующего пересечения). Значения не должны превышать

255. Это второе ограничение означает, что значения могут быть сохранены в массиве символов без знака (т.е. время каждого сегмента приводит в рабочее состояние всего один байт памяти), который помогает экономить память. У ATmega168 есть всего 1024 байта RAM, поэтому очень важно научиться эффективно использовать память. Но если посмотреть на то, что ATmega328 предоставляет нам 2048 байта RAM, то делаем вывод, что здесь больше возможностей для использования ресурсов для хранения данных.

Общий алгоритм 3pi:

1. Пока нет пересечений, двигаемся вперед с большой скоростью. Если увидим пересечение, то поворачиваем.
2. В противном случае, на текущем сегменте пути двигайтесь с большой скоростью до тех пор, пока не истекло время T, затем скорость будет постепенно замедляться и опять возвращаться к нормальной скорости, пока не будет обнаружено следующее пересечение.

В противном случае управляйте текущим значением T, которое может быть получено путем вычисления функции, которая использует ранее измеренный сегмент пути. Для коротких сегментов T отрицательный, и 3pi просто едет с нормальной скоростью на всем сегменте. Для более длинных сегментов T положительно и 3pi увеличивает скорость. Но когда замедляется тогда, когда необходимо замедлиться так, чтобы безопасно обработать пересечение.

Также можно использовать энкодеры, чтобы измерять длины сегментов. Тогда можно было бы использовать синхронизацию на 3pi. Однако это не лучший вариант, так как из-за его системы питания, которая использует отрегулированное напряжение для двигателей, это приведет к очень очевидным результатам. С более традиционной системой питания уменьшилась бы частота вращения двигателя, и поскольку батареи разряжаются, то синхронизация привела бы к неожиданным результатам.

Подсказка: Как только начинаете увеличивать решающую скорость, производительность становится зависима от тяги шин. К сожалению, уменьшение тяги в течение долгого времени, шины поднимают пыль и грязь при движении. Поэтому у такого робота должны быть такие шины, которые имели бы хорошее сцепление, с помощью которых можно было бы решить проблему проскальзывания на поворотах.