федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет «Московский институт электронной техники»

Лабораторные работы по курсу «Робототехника» для 10 класса

Москва 2021

Лабораторная работа №1 «Изучение интерфейса RobotC»

Цель

Изучение интерфейса и основных возможностей для программирования роботов в RobotC.

Краткие теоретические сведения

Среда RobotC предназначена для разработки и отладки программного кода, позволяющего управлять робототехническими устройствами на базе микроконтроллера VEX IQ и VEX CORTEX (рис.1).

Рисунок 1: Микроконтроллер VEX CORTEX

Подключение всех внешних устройств (датчики, моторы) происходит к определенным портам микроконтроллера, которые подписаны в соответствии с назначением: MOTOR (моторы), ANALOG (аналоговые датчики), DIGITAL (цифровые датчики) и UART1/UART2/I2C (порты для подключения внешних устройств по интерфейсу UART и I2C) (рис.2):

1. порты для подключения аналоговых датчиков;
2. порты для подключения цифровых датчиков;
3. порты для подключения внешних устройств по интерфейсу UART и I2C;
4. порты для подключения моторов.

Обратите внимание: в один микроконтроллер можно подключить до 8 аналоговых, 12 цифровых датчиков и 10 моторов одновременно. Это открывает широкие возможности для разработки роботов, которые могут решать достаточно сложные задачи.

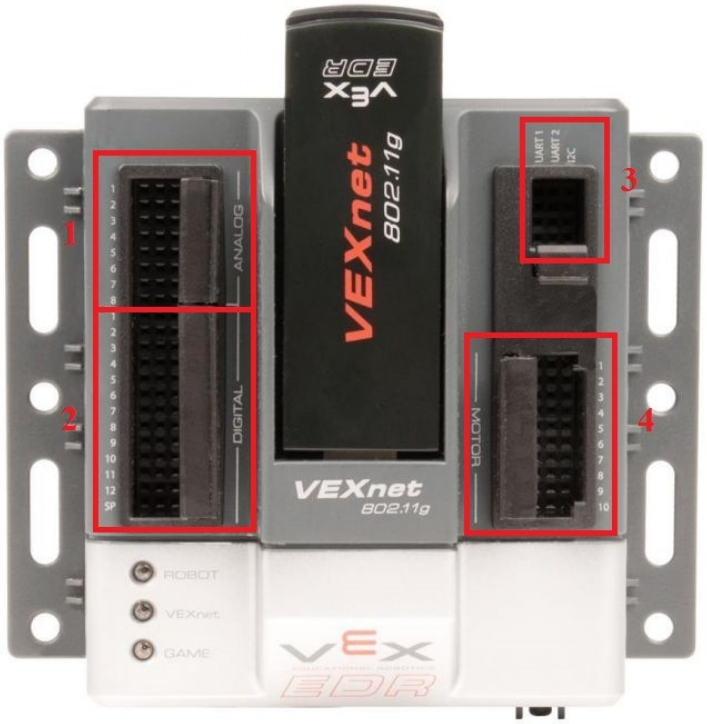

Рисунок 2: Порты микроконтроллера VEX CORTEX

В рамках данного лабораторного практикума рассматриваются основные аспекты создания робототехнических устройств на базе конструктора VEX EDR, основанного на применении микроконтроллера VEX CORTEX.

Внешний вид среды RobotC представлен на рисунке 3.

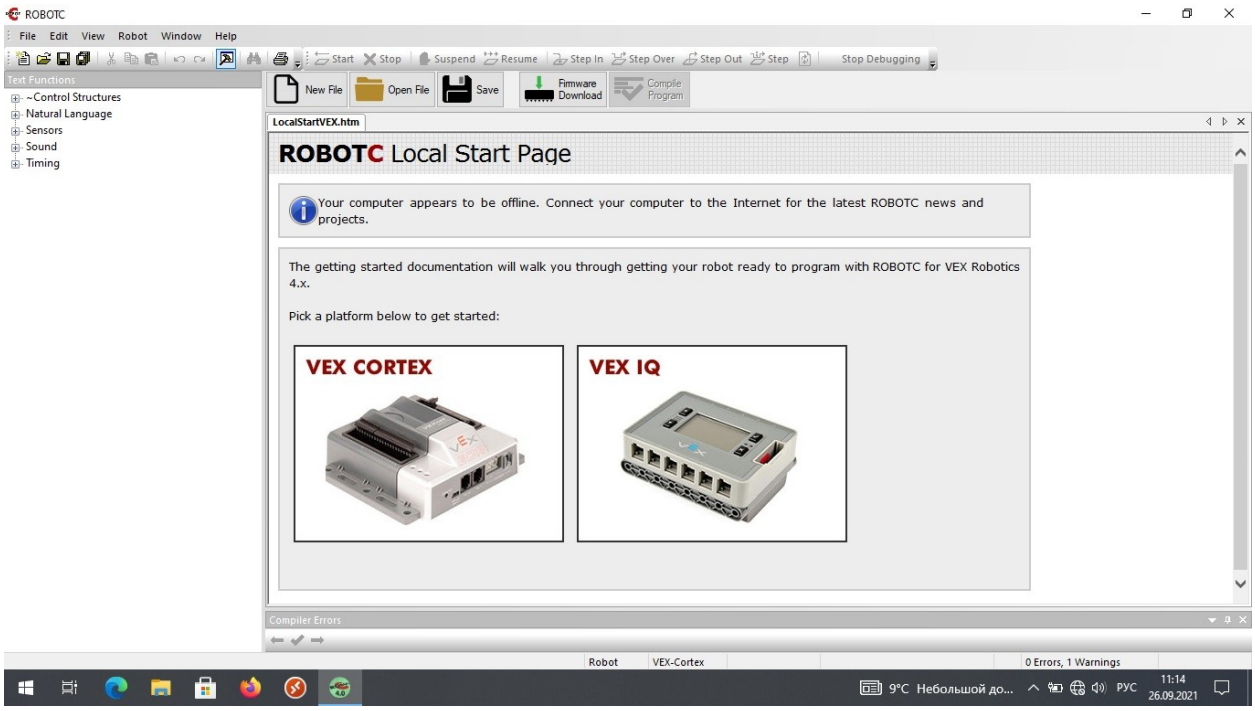

Рисунок 3: Стартовое окно RobotC

Основные элементы интерфейса выделены на рисунке 4 и состоят из следующих разделов:

1. главное меню;
2. пиктограммы наиболее часто используемых пунктов меню;
3. библиотека встроенных функций;
4. окно программного кода;
5. окно сообщений.

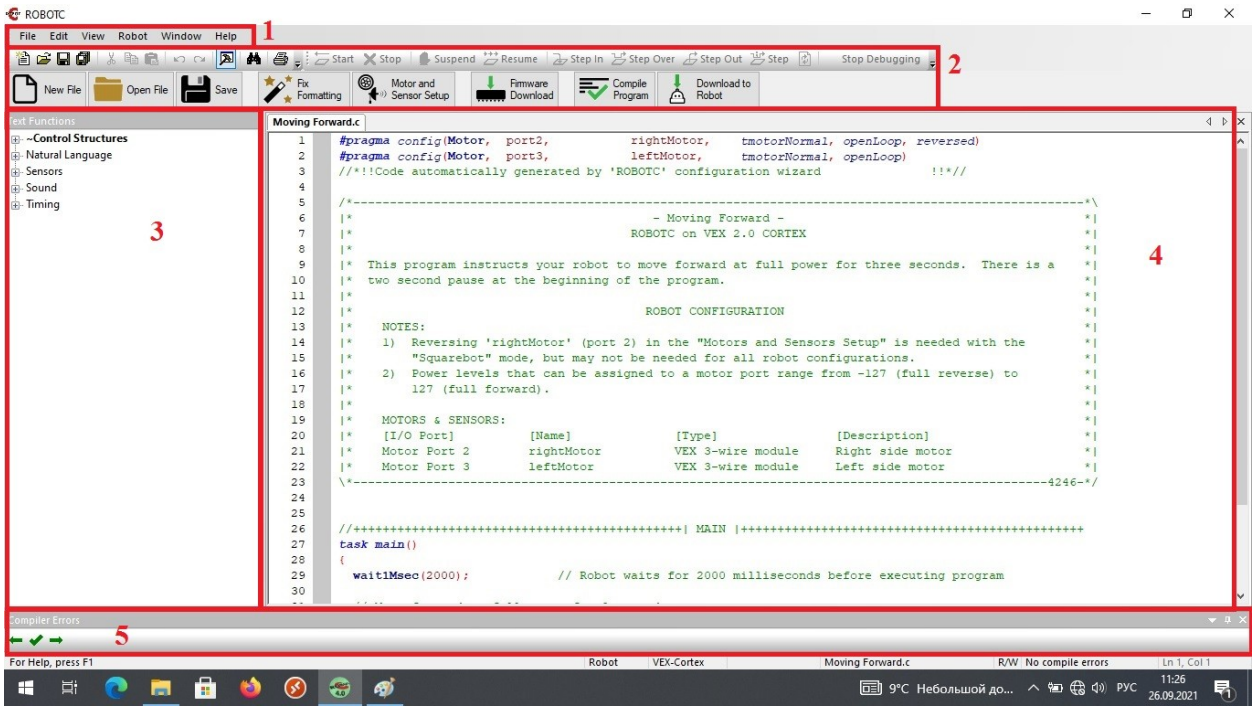

Рисунок 4: Основные элементы интерфейса RobotC

Первое, с чего начинается программирование робота – установка загрузчика в микроконтроллер (рис.5). Загрузчик – системное программное обеспечение, обеспечивающее загрузку операционной системы непосредственно после включения микроконтроллера и начальной загрузки. Загрузчик операционной системы обеспечивает необходимые средства для диалога с пользователем, в противном случае становится невозможным последующая прошивка микроконтроллера разработанной пользователем программы.

Следующий шаг также является подготовительным и заключается в том, что необходимо выбрать платформу (тип используемого микроконтроллера) (рис.6) Robot → Platform Type → VEX 2.0 Cortex.

Наконец, последним подготовительным шагом является выбор способа последующего управления роботом (рис.7): по USB-кабелю – USB Only, с помощью джойстика – Competition (VEXnet) или одновременно по USB-кабелю и с помощью джойстика (VEXnet or USB).

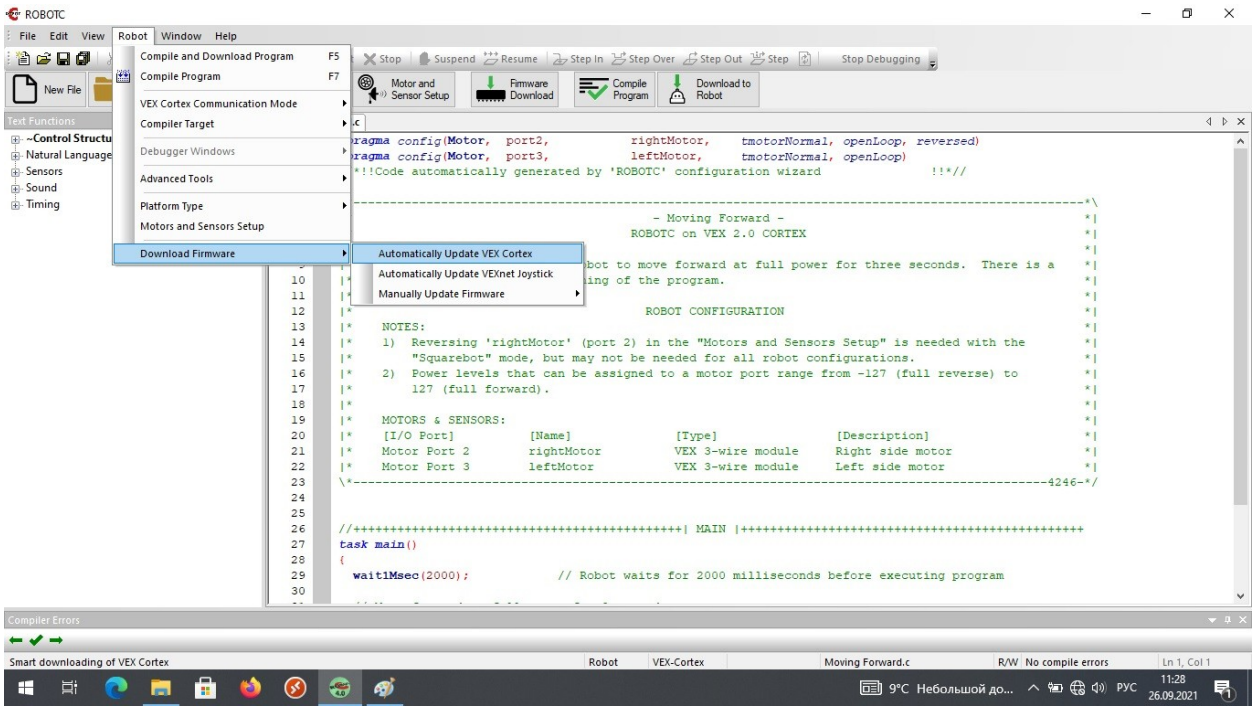

Рисунок 5: Установка загрузчика в микроконтроллер

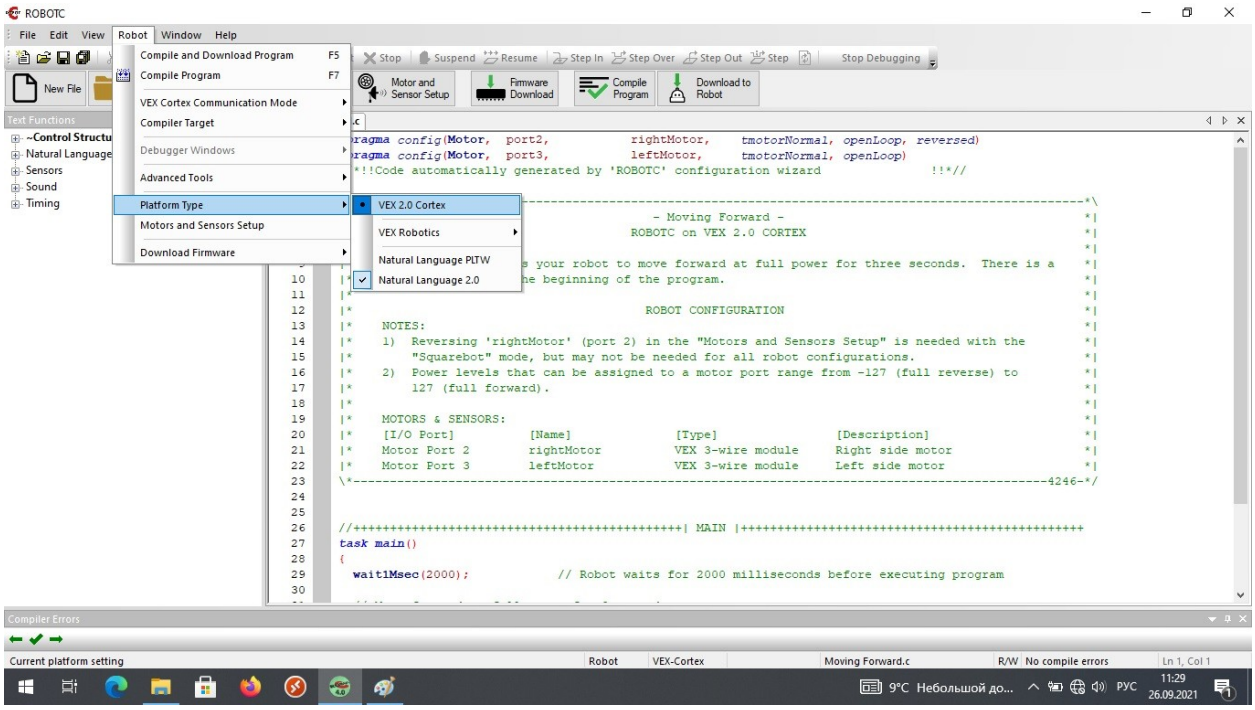

Рисунок 6: Выбор платформы для программирования

Особенностью управления роботом по USB-кабелю является то, что в этом случае есть возможность запуска отладчика кода – можно устанавливать контрольные точки, для приостановки выполняемой программы, выполнять программу по шагам и видеть содержимое всех глобальных переменных и подключенных датчиков. Такая возможность является очень полезной,

поскольку в случае разработки сложного робота неизбежно возникает необходимость отладки кода и поиска причин возникающих ошибок – соответственно, крайне важно при этом видеть содержимое всех переменных и датчиков.

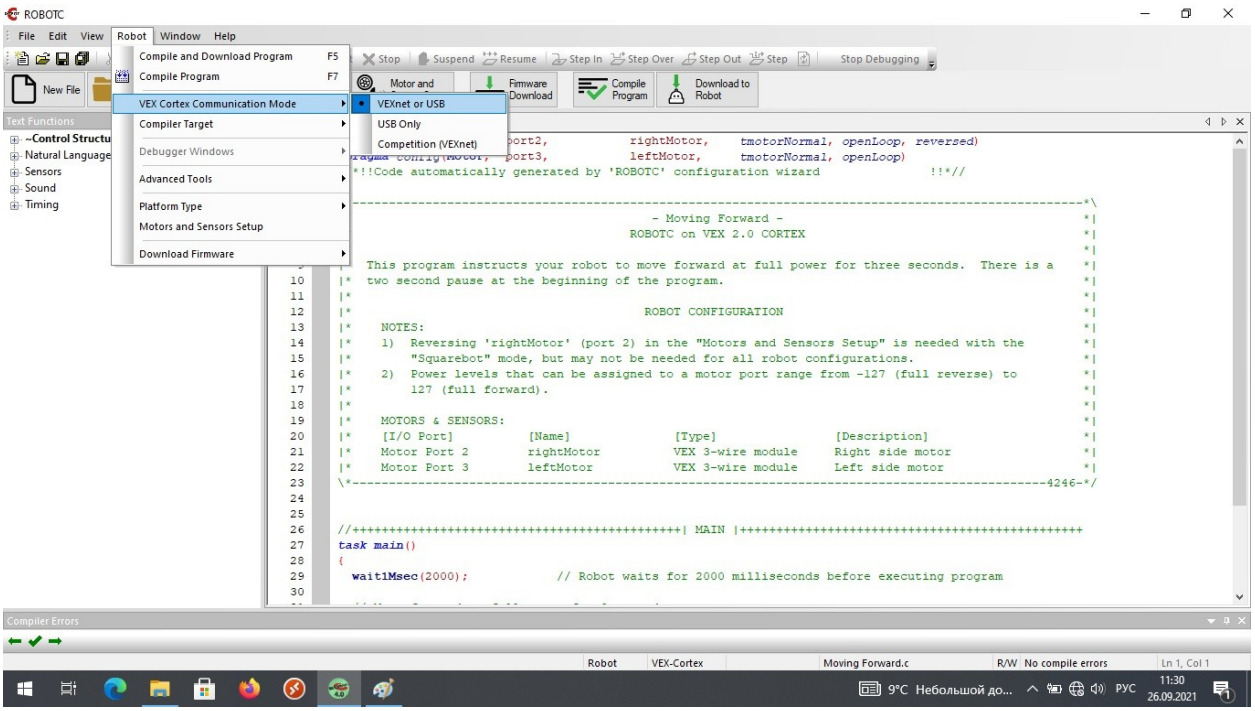

Рисунок 7: Выбор способа управления роботом

Теперь необходимо выполнить настройку подключения различных датчиков к роботу – для того, чтобы компилятор «понимал», куда именно подключен тот или иной датчик. Разумеется, одновременно в процессе настройки необходимо физически подключить используемые датчики в соответствующие порты микроконтроллера. Для настройки подключения моторов и датчиков в RobotC используется меню «Motors and Sensors Setup», которое доступно через пиктограммы наиболее часто используемых пунктов меню. Данное меню имеет четыре вкладки: «Standard Models», «Motors», «VEX 2.0 Analog Sensors 1-8» и «VEX Cortex Digital Sensors 1-12». Рассмотрим эти вкладки более подробно.

Первая вкладка – «Standard Models» (рис.8) предназначена для выбора одной из перечисленных в выпадающем списке конфигураций робота. Список конфигураций достаточно обширен и включает в себя, в том числе, ту модель, с

которой нам предстоит работать на протяжении всего лабораторного практикума – RVW Clawbot. В разделе «Model Description» перечислены все датчики и моторы с указанием портов, в которые они должны быть физически подключены.

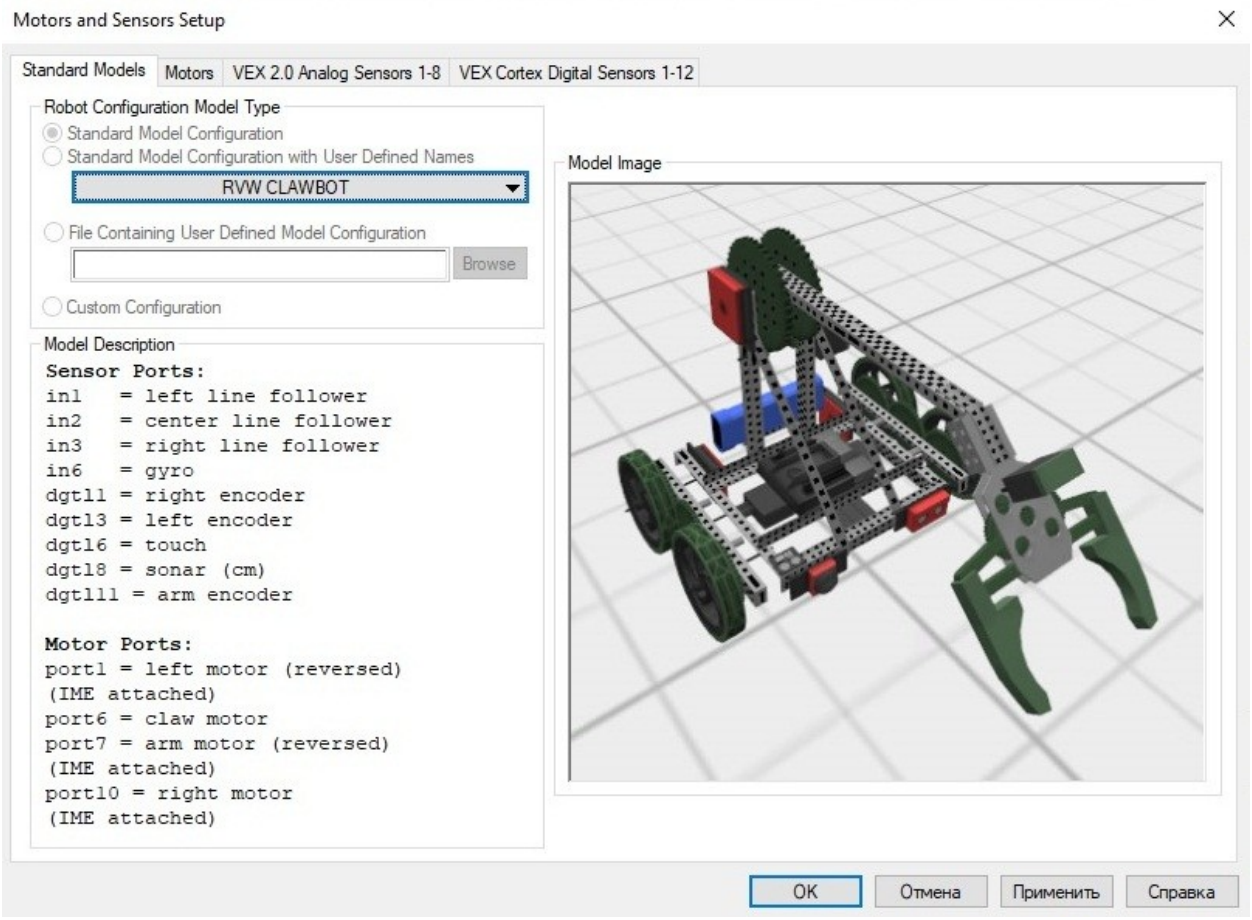

Рисунок 8: Меню «Motors and Sensors Setup», вкладка «Standard Models»

Следующая вкладка – «Motors» (рис.9) предназначена для подключения моторов к роботу. Для подключения мотора необходимо сначала выбрать порт микроконтроллера, к которому собираемся сделать подключение: всего у нас в распоряжении 10 портов – port1…port10.

На следующем шаге необходимо задать имя для мотора (например, для порта «port3» задано имя мотора «leftMotor»). Имя выбирается произвольно, но желательно заранее выбрать некоторую логику, чтобы впоследствии по имени мотора можно было понять его основное предназначение и местоположение. При написании программы можно обращаться к мотору двумя способами: по

имени и по номеру порта. Учитывая, что количество подключенных моторов достигает 10, то гораздо удобнее использовать первый вариант - по имени.

Наконец, на последнем шаге необходимо из раскрывающего списка выбрать тип мотора: в наборе конструктора VEX EDR поставляется только один тип – VEX 393 – соответственно, при выполнении заданий данного лабораторного практикума следует выбирать именно этот мотор. Также, обратите внимание: справа от типа мотора есть переключатель «Reversed». В случае его использования мотор будет вращаться в противоположную сторону. Это очень удобно, когда необходимо достигнуть согласованности в работе моторов – например, для управления базой робота используется два мотора, и они расположены зеркально друг относительно друга. Соответственно, для того, чтобы обе оси колес вращались в одну и ту же сторону (при движении вперед) необходимо, чтобы физически один мотор вращал свою ось по часовой стрелке, а другой – против нее.

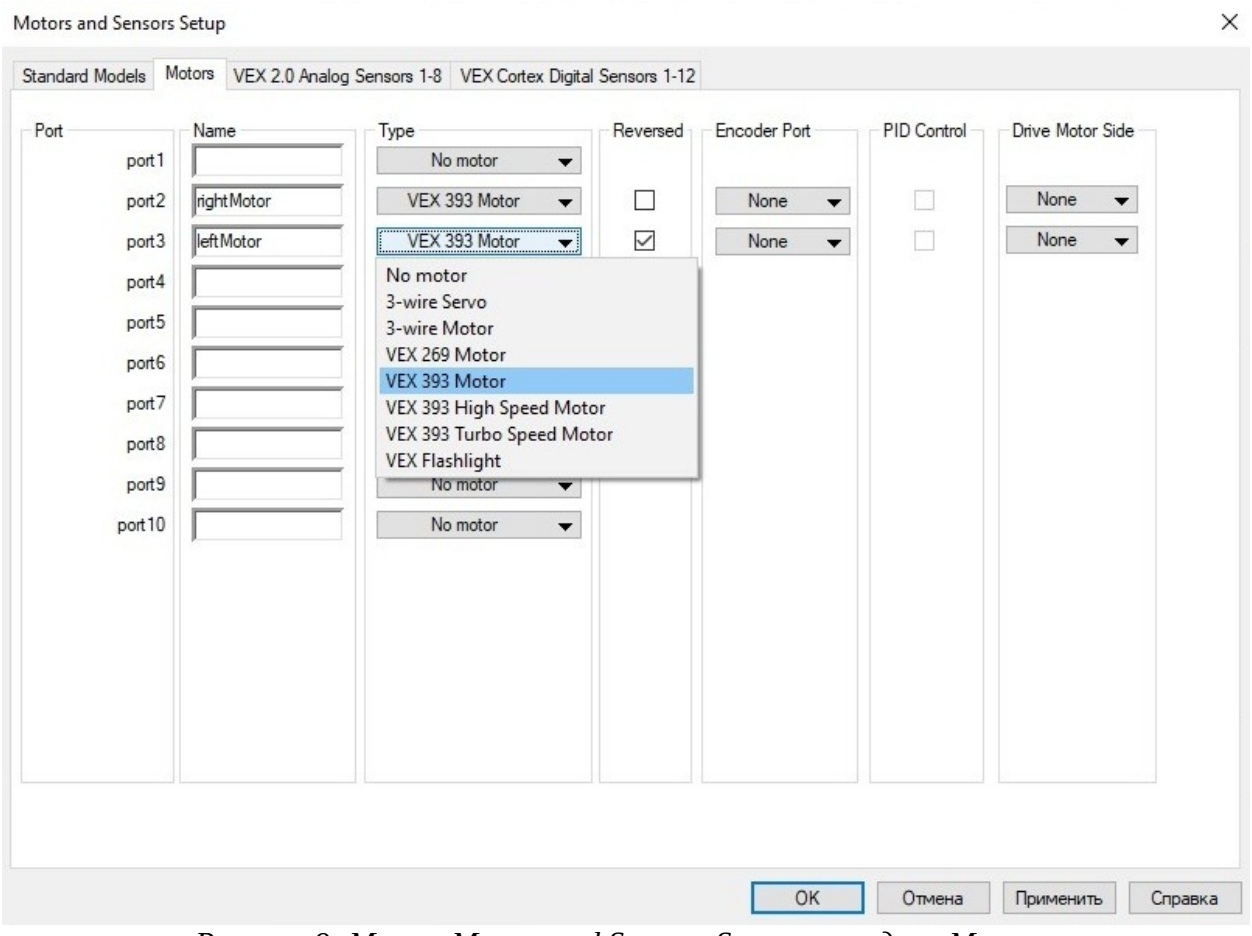

Рисунок 9: Меню «Motors and Sensors Setup», вкладка «Motors»

Следующая вкладка - «VEX 2.0 Analog Sensors 1-8» (рис.10) предназначена для подключения аналоговых датчиков. Так же, как и в случае с подключением моторов необходимо выбрать один из портов: in1…in8. На следующем шаге необходимо задать имя для датчика и выбрать его тип из раскрывающегося списка. Для выбора доступны такие датчики, как: потенциометр (Potentiometer), датчик освещенности (Light Sensor), датчик линии (Line Follower), гироскоп (Gyro Sensor) и акселерометр (Accelerometer). Работа с каждым из перечисленных датчиков представляет собой отдельную задачу и большинство из них будет нами рассмотрено в следующих лабораторных работах.

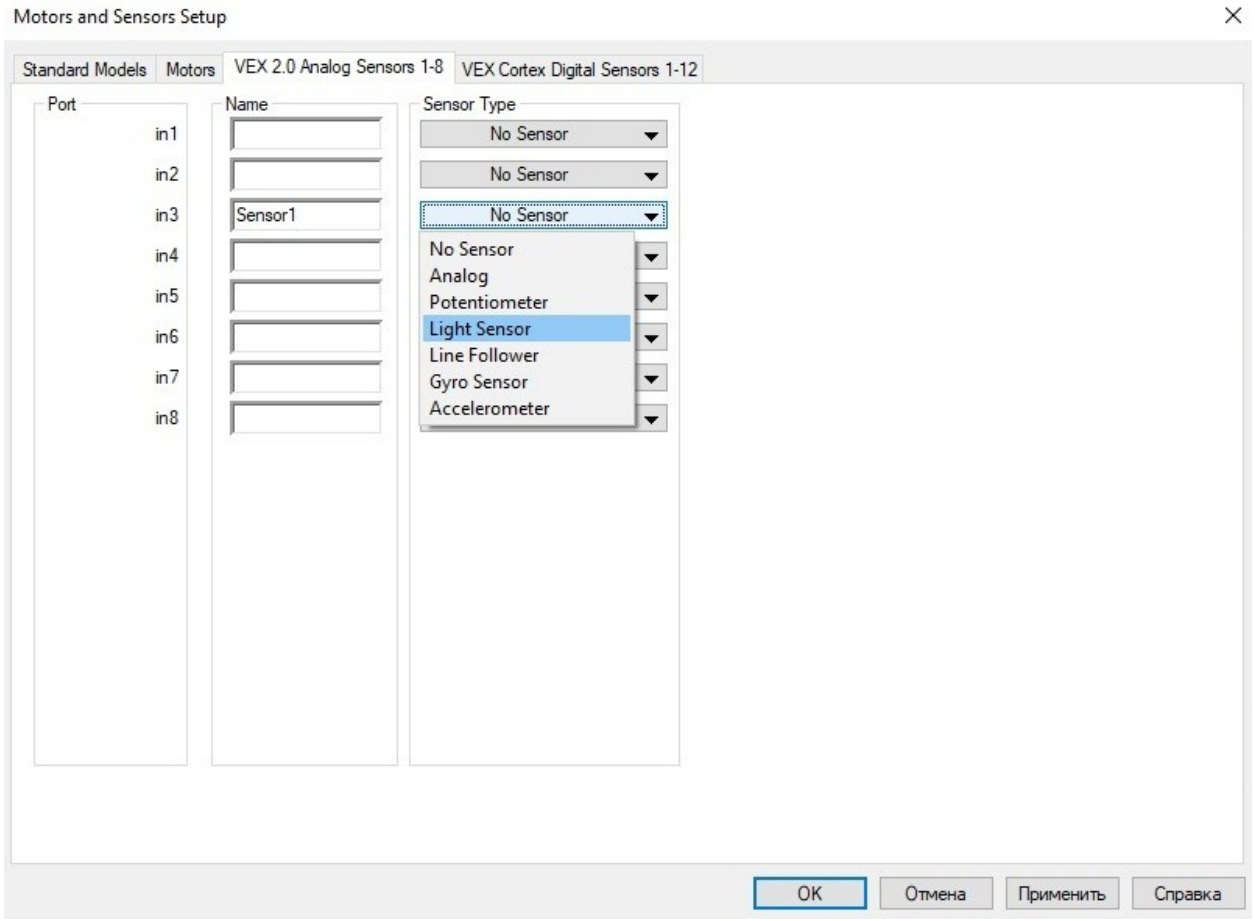

Рисунок 10: Меню «Motors and Sensors Setup», вкладка «VEX 2.0 Analog Sensors 1-8»

Наконец, последняя вкладка – «VEX Cortex Digital Sensors 1-12» (рис.11) предназначена для подключения цифровых датчиков. Так же, как и в случае с подключением моторов и аналоговых датчиков необходимо выбрать один из портов: dgtl1…dgtl12. На следующем шаге необходимо задать имя датчика и выбрать его тип из раскрывающегося списка. Для выбора доступны такие

датчики, как: кнопка (Bumper Switch или Limit Switch: с точки зрения программирования обыкновенная кнопка и концевой переключатель представляют собой один и тот же элемент), оптический энкодер (Quadrature Encoder), встраиваемый энкодер (Encoder (Single Wire)), светодиод (VEX LED), цифровой вход/выход (Digital In/Out), цифровой высокоомный резистор (Digital High Impedance) и ультразвуковой дальномер (SONAR). Работа с каждым из перечисленных датчиков представляет собой отдельную задачу и большинство из них будет нами рассмотрено в следующих лабораторных работах.

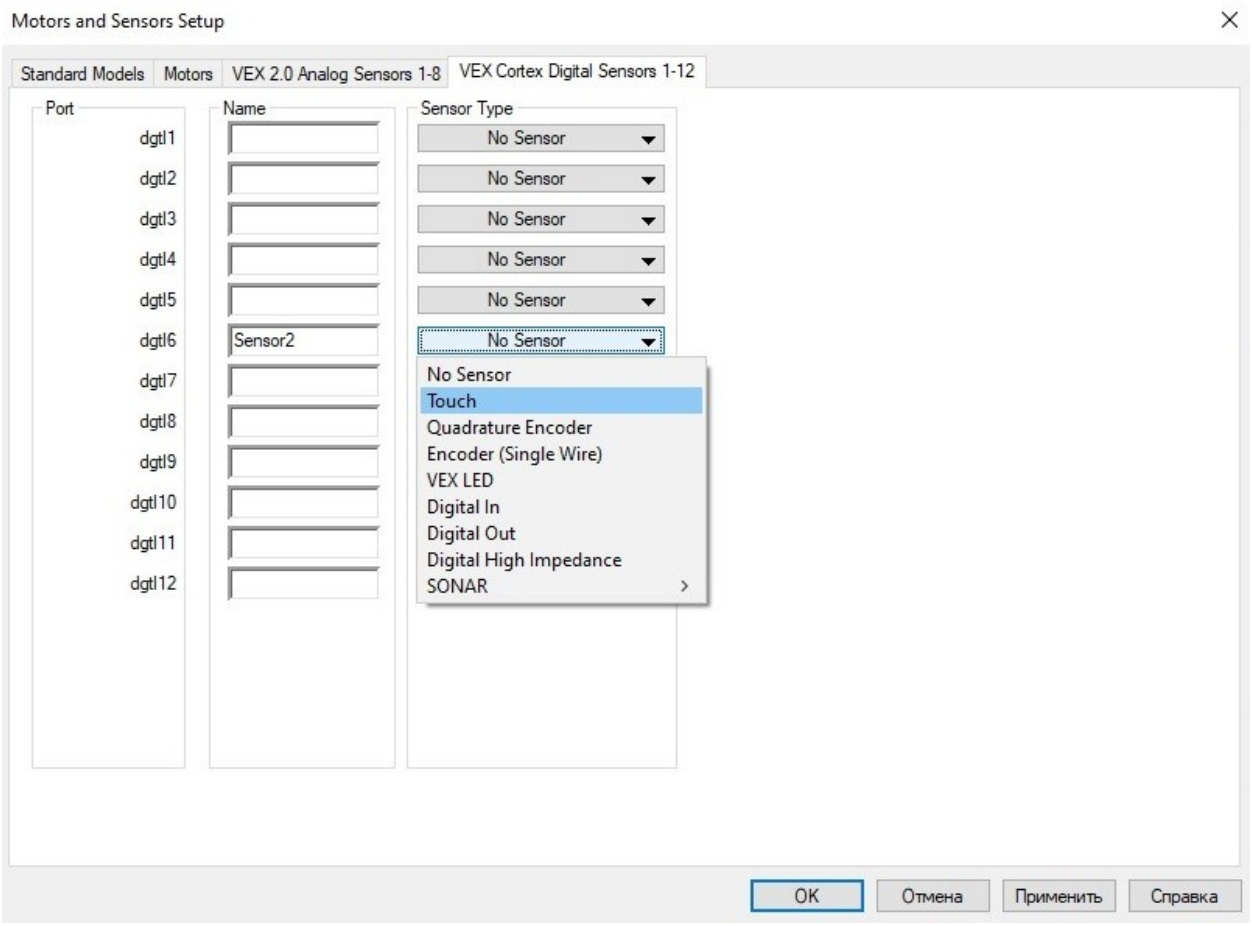

Рисунок 11: Меню «Motors and Sensors Setup», вкладка «VEX Cortex Digital Sensors 1-12»

После завершения настройки подключения следует нажать кнопку «Применить» и затем «ОК».

В результате, в окне программного кода автоматически появятся строки, содержащие описание подключенных датчиков на языке RobotC (рис.12). Эту часть программы не следует изменять вручную т.к. она формируется

автоматически и «привязана» к настройкам, выставленным пользователем в меню «Motors and Sensors Setup».

После того, как выполнена настройка подключения датчиков и моторов можно приступать к написанию кода. Как правило, используются разнообразные встроенный функции, уровень сложности которых определяется выбранным уровнем программирования: Basic, Expert или Super User (рис.13).

В библиотеке встроенных функций (рис.14) приведены все функции, доступные для использования. После наведения курсора мыши на название функции появляется подсказка, с помощью которой можно понять, каким образом данная функция вызывается, и какие аргументы она использует.

Moving Forward.c

```
1  #pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop, reversed)
2  #pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop)
3  //*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
4
5  /*---------------------------------------------------------------------------------------------------*\
6  |*                                       - Moving Forward -                                          *|
7  |*                                     ROBOTC on VEX 2.0 CORTEX                                      *|
8  |*                                                                                                   *|
9  |*  This program instructs your robot to move forward at full power for three seconds.  There is a    *|
10 |*  two second pause at the beginning of the program.                                                *|
11 |*                                                                                                   *|
12 |*                                       ROBOT CONFIGURATION                                         *|
13 |*    NOTES:                                                                                         *|
14 |*    1)  Reversing 'rightMotor' (port 2) in the "Motors and Sensors Setup" is needed with the       *|
15 |*        "Squarebot" mode, but may not be needed for all robot configurations.                      *|
16 |*    2)  Power levels that can be assigned to a motor port range from -127 (full reverse) to        *|
17 |*        127 (full forward).                                                                        *|
18 |*                                                                                                   *|
19 |*    MOTORS & SENSORS:                                                                              *|
20 |*    [I/O Port]          [Name]              [Type]               [Description]                     *|
21 |*    Motor Port 2        rightMotor          VEX 3-wire module    Right side motor                  *|
22 |*    Motor Port 3        leftMotor           VEX 3-wire module    Left side motor                   *|
23 \*---------------------------------------------------------------------------------------------4246-*/
24
25
26 //++++++++++++++++++++++++++++++++++++++++++++++| MAIN |++++++++++++++++++++++++++++++++++++++++++++++
27 task main()
28 {
29   wait1Msec(2000);           // Robot waits for 2000 milliseconds before executing program
30
```

Рисунок 12: Описание подключенных датчиков (выделено рамкой)

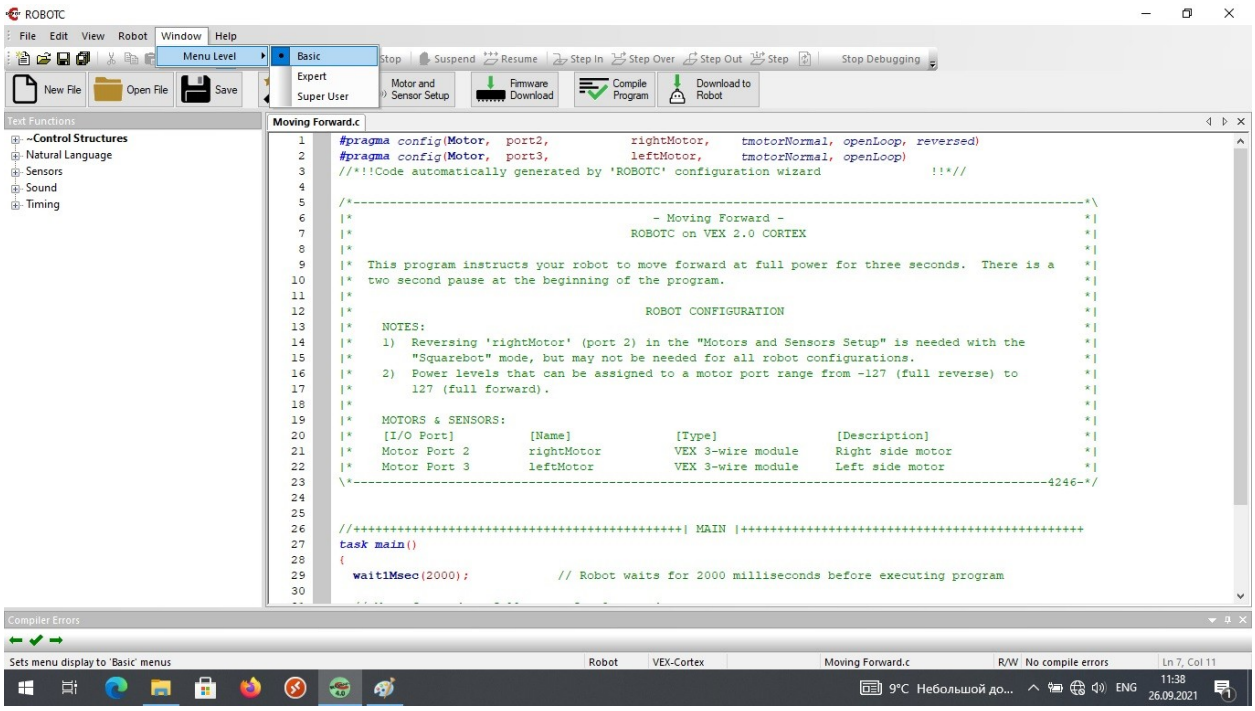

Рисунок 13: Выбор уровня сложности программы

После того, как программа написана, необходимо нажать кнопку «Compile Program» и исправить возникшие ошибки. Затем можно выполнить «прошивку» микроконтроллера робота: для этого, предварительно, нужно выбрать объект для программирования (рис.15) – физический робот, собранный из конструктора («Physical Robot») или виртуальный робот, работающий в симулятора («Virtual Worlds»). В первом случае, после выбора соответствующего меню, необходимо нажать кнопку «Download to Robot». Во втором случае откроется дополнительное окно, в котором необходимо выбрать конфигурацию робота и поле, на котором он будет выполнять написанную программу (рис.16).

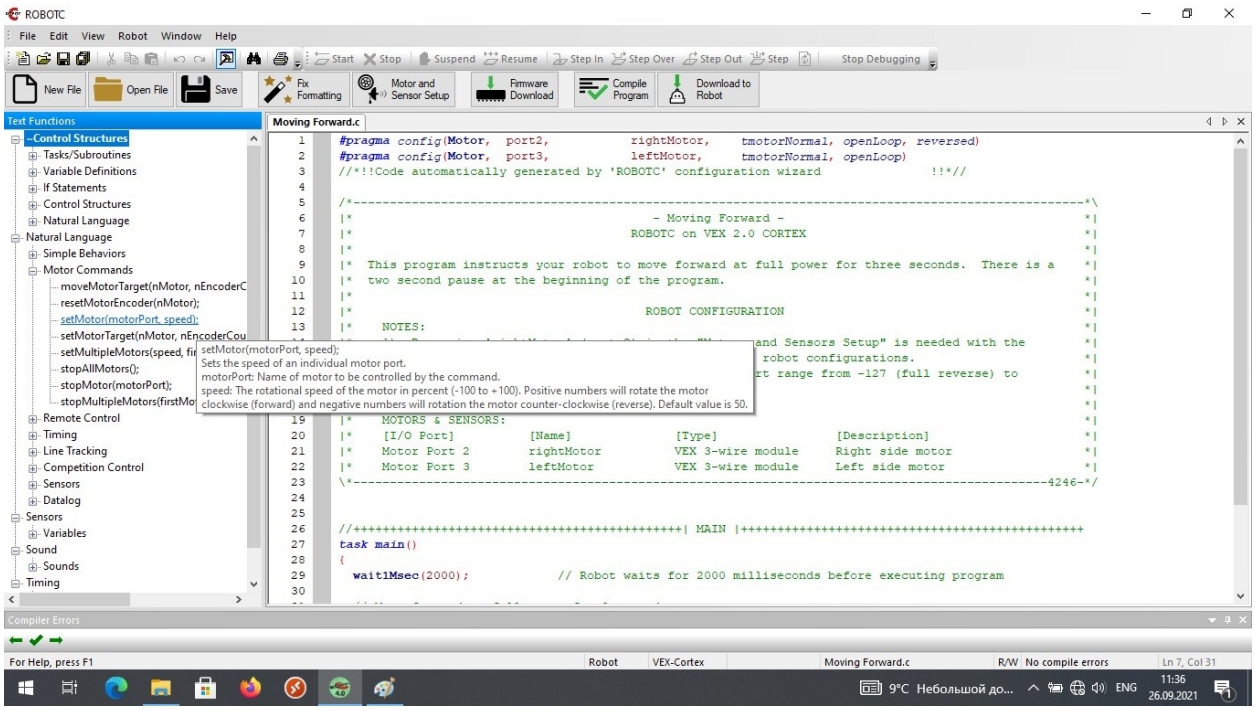

Рисунок 14: Выбор встроенной функции из библиотеки

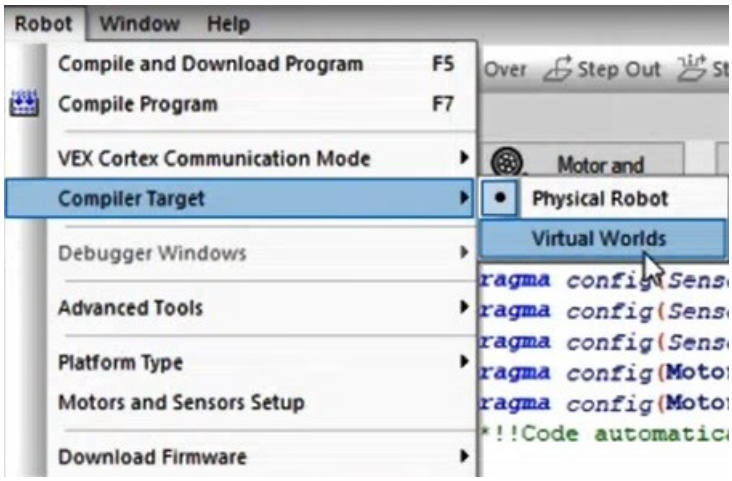

Рисунок 15: Выбор робота для программирования

Также, для написания программы могут быть полезны готовые примеры, реализующие известные робототехнические задачи (рис.17). Эти программы поставляются вместе со средой RobotC и могут быть использованы в качестве «заготовок» для будущей программы.

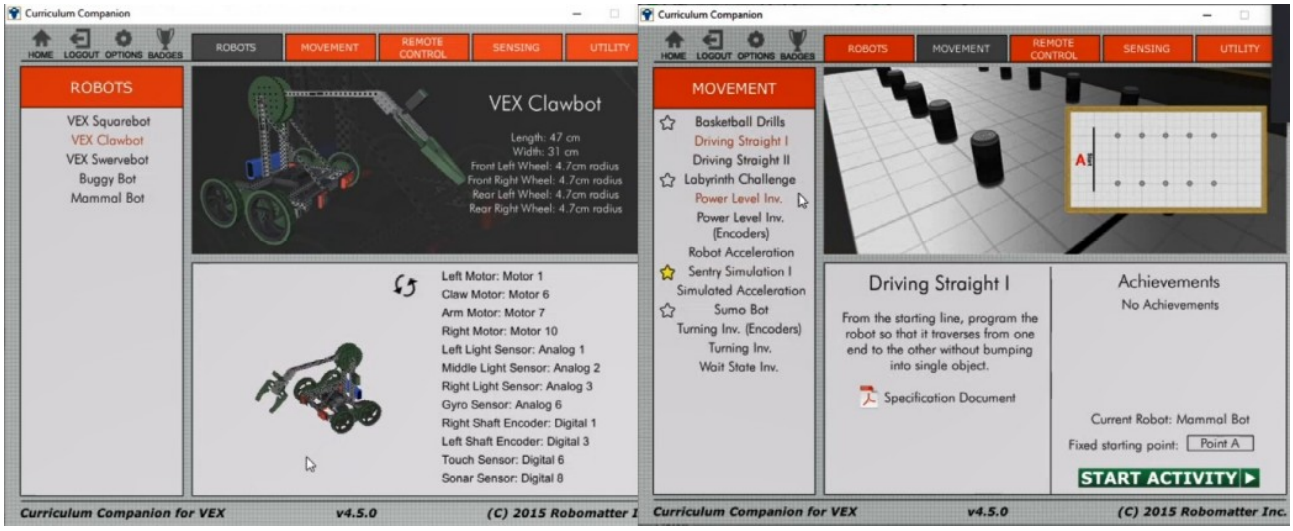

Рисунок 16: Выбор конфигурации робота и поля

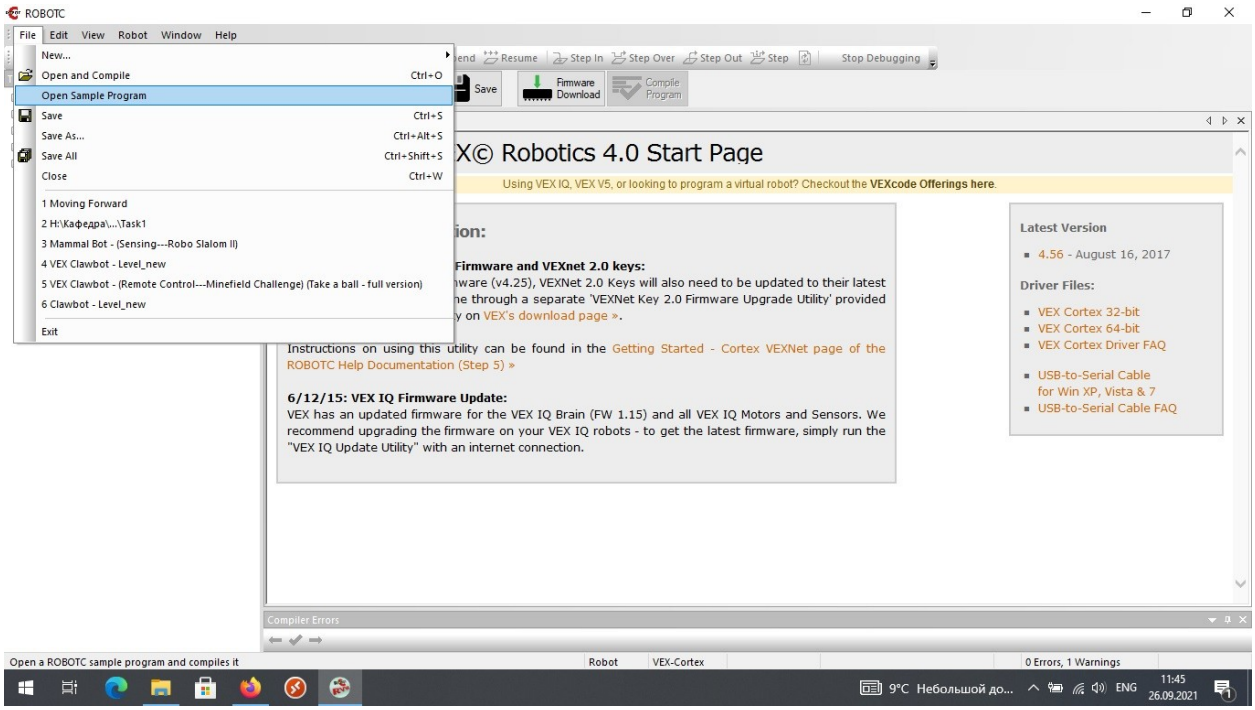

Рисунок 17: Расположение папки с готовыми программами

Задание

Откройте из папки готовых программ «Basic Movement» программу «Moving Forward», проверьте правильность подключения датчиков и моторов, после чего скомпилируйте программу и убедитесь в отсутствии ошибок.

Порядок выполнения работы

1. Запустите RobotC.
2. Откройте папку с готовыми программами.

3. Выберите папку «Basic Movement».
4. Выберите программу «Moving Forward».
5. Проверьте правильность подключения моторов и датчиков в меню «Motors and Sensors Setup».
6. Выберите базовый уровень программирования.
7. Выберите тип микроконтроллера.
8. Скомпилируйте программу и убедитесь в отсутствии ошибок.

Содержание отчета по работе

Отчет должен содержать снимки экрана выполнения этапов задания.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №2 «Создание функций на языке RobotC»

Цель

Изучение правил создания и синтаксиса пользовательских функций на языке RobotC.

Краткие теоретические сведения

Функция – это фрагмент программного кода, к которому можно обратиться из другого места программы. По сути, функция представляет собой небольшую программу, которая может многократно вызываться внутри основной части кода. Это обуславливает определенное удобство – при отсутствии функции потребовалось бы многократно повторять одинаковую последовательность строк, что неизбежно загромождало бы программу и приводило к ухудшению ее целостного восприятия.

В том случае, если целью работы функции является выполнение некоторых вычислений над полученными ранее данными, она принимает эти данные, и, соответственно, после выполнения преобразования, выдает результат, доступный для обработки последующей частью кода. Если же целью работы функции является управление некоторыми механизмами или узлами (например, мотором), то такая функция может и не иметь выходного результата т.к. в этом случае результатом ее работы является физическое действие, а не математический расчет. В обоих случаях результат работы функции представляет собой некоторое возвращаемое значение.

Моторы, используемые в нашем курсе, выглядят так, как показано на рисунке 1.

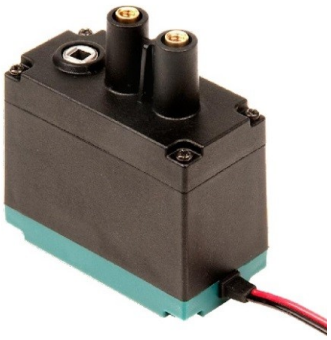

Рисунок 1: Мотор VEX393

У микроконтроллера VEX CORTEX присутствует 10 портов для подключения моторов: 2 – напрямую, 8 – через драйвер (рис.2).

Рисунок 2: Драйвер для мотора VEX393

Функции на языке RobotC имеют следующую структуру:

тип_возвращаемого_значения имя_функции (**тип_аргумента_1** аргумент_1, **тип_аргумента_2** аргумент_2, … , **тип_аргумента_N** аргумент_N)

{

тело функции

}

В качестве типа возвращаемого значения указывают один из следующих вариантов:

void – в том случае, если функция не возвращает никакого значения;

int – в том случае, если функция возвращает целочисленное значение;

float – в том случае, если функция возвращает значения с плавающей запятой (дробные числа);

double – в том случае, если функция возвращает значения с двойной точностью относительно типа float (по сути, в данном случае выделяется в 2 раза больший объем памяти для хранения такого вида данных).

Имя функции создается пользователем произвольно с использованием латиницы (русский шрифт использовать нельзя). Не существует однозначных требований к имени функции – это целиком находится в зоне ответственности пользователя, за исключением того, что имя не должно начинаться с цифры и не должно содержать пробелов. Очевидно, что одним из возможных подходов для создания имени функции является его «функциональность» т.е. разумно давать такие имена, которые будет давать возможность стороннему человеку понять основное назначение данной функции.

Имена для аргументов функции так же задаются произвольно, на усмотрение пользователя. Так же, как и в случае с именем, разумно, давать короткие, значимые имена т.к. далее все эти аргументы будут использоваться в теле функции.

Тело функции представляет собой программный код, реализующий саму функцию посредством стандартных операций над объявленными выше аргументами. По сути, здесь пользователь создает ту часть кода, которую он планирует многократно вызывать в основной программе.

Для того, чтобы вызвать функцию, выполнить ее, необходимо написать ее имя в требуемой части кода, далее в скобках перечислить передаваемые ей аргументы, в том порядке, в котором они в ней объявлены.

Например, создадим функцию для движения робота Clawbot вперед и назад. Такая функция выглядит следующим образом:

```
void moving (int LeftMotor, int RightMotor, int time)
{
setMotor (port1, LeftMotor);
setMotor (port6, RightMotor);
  wait1Msec (time);
setMotor (port1, 0);
setMotor (port6, 0);
  wait1Msec (30);
}
```

Именем функции является слово «moving», что сразу дает понимание стороннему человеку об основном назначении функции – движение. Поскольку данная функция не выполняет каких-то вычислений, то она не возвращает никакого значения и, соответственно, в качестве типа возвращаемого значения указан тип «void». Функция moving имеет 3 аргумента: LeftMotor (скорость левого мотора), RightMotor (скорость правого мотора) и time (длительность движения). Все перечисленные аргументы являются целочисленными (нет смысла задавать дробную скорость вращения мотора или дробное значение длительности движения), поэтому все аргументы имеют тип int.

Тело функции состоит из следующих строк, заключенных в фигурные скобки:

```
setMotor (port1, LeftMotor);
setMotor (port6, RightMotor);
wait1Msec (time);
```

Как было сказано выше, тело функции представляет собой короткую программу, использующую стандартные (встроенные функции). Это такие функции, которые уже поставляются производителем в составе библиотеки внутри среды программирования RobotC. В нашем случае используются следующие стандартные функции: setMotor и wait1Msec. Рассмотрим их более подробно.

Функция setMotor принимает на вход 2 аргумента – номер порта, к которому физически подключен мотор, и скорость мотора, задаваемую пользователем. В качестве номера порта указывается не число, а целиком слово «port» с последующим указанием номера – «port1» и «port6» в нашем случае. Разумеется, перед запуском программы мы должны включить левый мотор робота в первый порт микроконтроллера, а правый мотор робота в шестой порт микроконтроллера. В качестве скорости мотора следует указать целое число в диапазоне от -125 до +125. Знак «+» или «-» будет указывать на направление вращения мотора (вперед или назад). Поскольку мы вызываем встроенную функцию setMotor внутри нашей (пользовательской) функции moving, то вместо числа мы указываем введенный ранее аргумент: LeftMotor (для указания скорости левого мотора) и RightMotor (для указания скорости правого мотора). Наконец, для того, чтобы указать компилятору, в течение какого времени мотору необходимо вращать ось, используется встроенная функция wait1Msec, которая принимает на вход всего лишь один аргумент – время в миллисекундах. Фактически, эта функция реализует задержку той длительности, которая была указана пользователем. После вызова перечисленных функций необходимо «отпустить» мотор, чтобы робот остановился и компилятор «понимал», что работа с моторами завершена. Для этого мы вызываем еще раз функции setMotor, но в качестве скорости указываем «0» - эта команда даст возможность компилятору однозначно «понять», что мотор нужно остановить. И завершает функцию команда wait1Msec с небольшой задержкой, значение которой составляет 30 мс. В принципе, можно указать любое другое близкое значение – идея в том, чтобы компилятор «успел» отреагировать на вызов функции setMotor с нулевым значением скорости – визуально мы не увидим никаких отличий, но программа в этом случае будет работать более корректно и мы сможем избежать потенциальных сбоев.

Для того, чтобы вызвать (выполнить) созданную пользователем функцию, необходимо записать внутри основной (выполняемой) функции всего лишь

одну строку, Также, как это делается для вызова библиотечных функций в примере выше:

имя_функции (значение_аргумента_1, значение_аргумента_2, … , значение_аргумента_N);

Обратите внимание, каждая строка выполняемого кода должна завершаться знаком «;», Кроме структур, заключающих код в фигурные скобки {}, например, после void moving (int LeftMotor, int RightMotor, int time) точка с запятой не ставится. Также, как и после условных операторов и циклов, о которых речь пойдет дальше.

В случае с нашим примером вызов функции moving происходит следующим образом:

```
moving (50, 50, 600);
```

Такая запись будет означать для компилятора следующее: запустить функцию moving со значением аргумента LeftMotor, равным 50, и со значением аргумента RightMotor, так же равным 50, в течение 600 миллисекунд. В результате, робот Clawbot будет ехать вперед со скоростью, равной 50 (безразмерных единиц) в течение 600 миллисекунд и после этого остановится.

Соответственно, мы можем вызывать эту функцию столько раз, сколько будет нужно в пределах основной программы, но, если сама функция состоит из 6 строк, то нам не придется каждый раз их повторять – нам потребуется всего лишь одна строка. Это очень удобно и не загромождает основную программу.

Пример текста готовой программы:

```
#pragma config(Sensor, dgtl1, Bumper, sensorDigitalIn)
#pragma config(Motor, port1, MotorLeft, tmotorVex393_HBridge, openLoop, driveLeft)
#pragma config(Motor, port6, MotorRight, tmotorVex393_MC29, openLoop, reversed, driveRight)
// сроки выше являются конфигурационными
//*!!Code automatically generated by 'ROBOTC' configuration wizard!!*//
```

```
void moving(int LeftMotor, int RightMotor, int time)
{
  setMotor(port1, LeftMotor);
  setMotor(port6, RightMotor);
  wait1Msec(time);
}
task main()
{
  moving(50,50,600);
}
```

После запуска симулятора можно увидеть выполнение написанной программы – движение робота Clawbot вперед (рис.3).

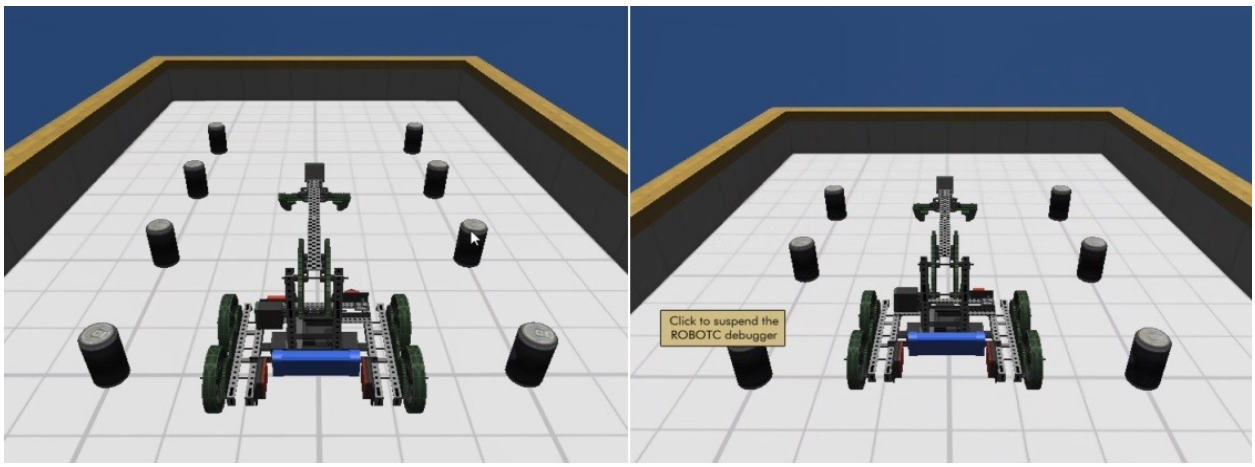

Рисунок 3: Пример выполнения программы в симуляторе

Задание

Создайте пользовательскую функцию, которая позволит роботу Clawbot совершить поворот направо на 90^0.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup» (рис.4): укажите в качестве имени правого мотора «RightMotor», а в качестве имени левого мотора «LeftMotor», заполнив поля напротив соответствующего номера порта. Номера портов следует взять из схемы робота (рис.5). В данной работе используется поле Driving Straight I (рис.6).
3. Напишите функцию, реализующую поворот робота Clawbot направо. Подумайте, чем будет отличаться такая программа от той, что была рассмотрена в разделе «Краткие теоретические сведения»? В каких

направлениях нужно вращать оси моторов, чтобы робот начал разворачиваться в какую-либо сторону?

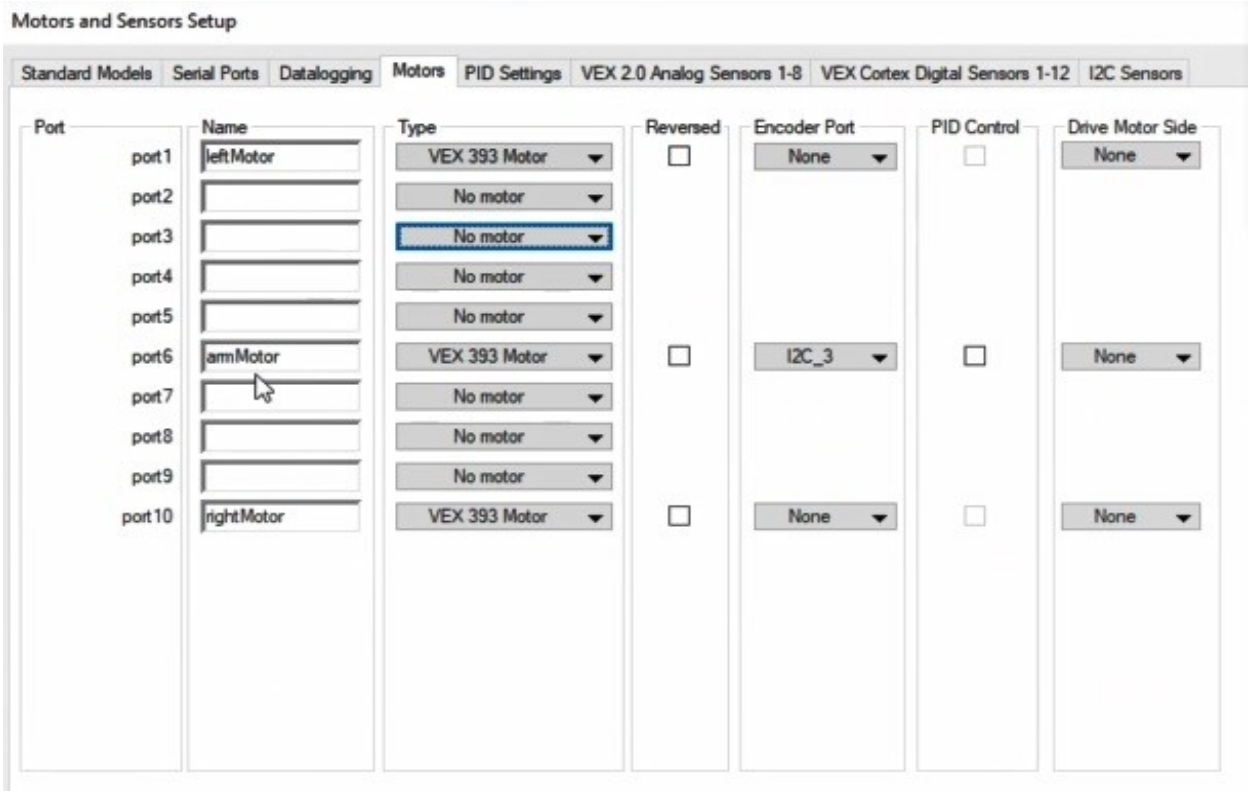

Рисунок 4: Подключение моторов в окне Motor and Sensor Setup

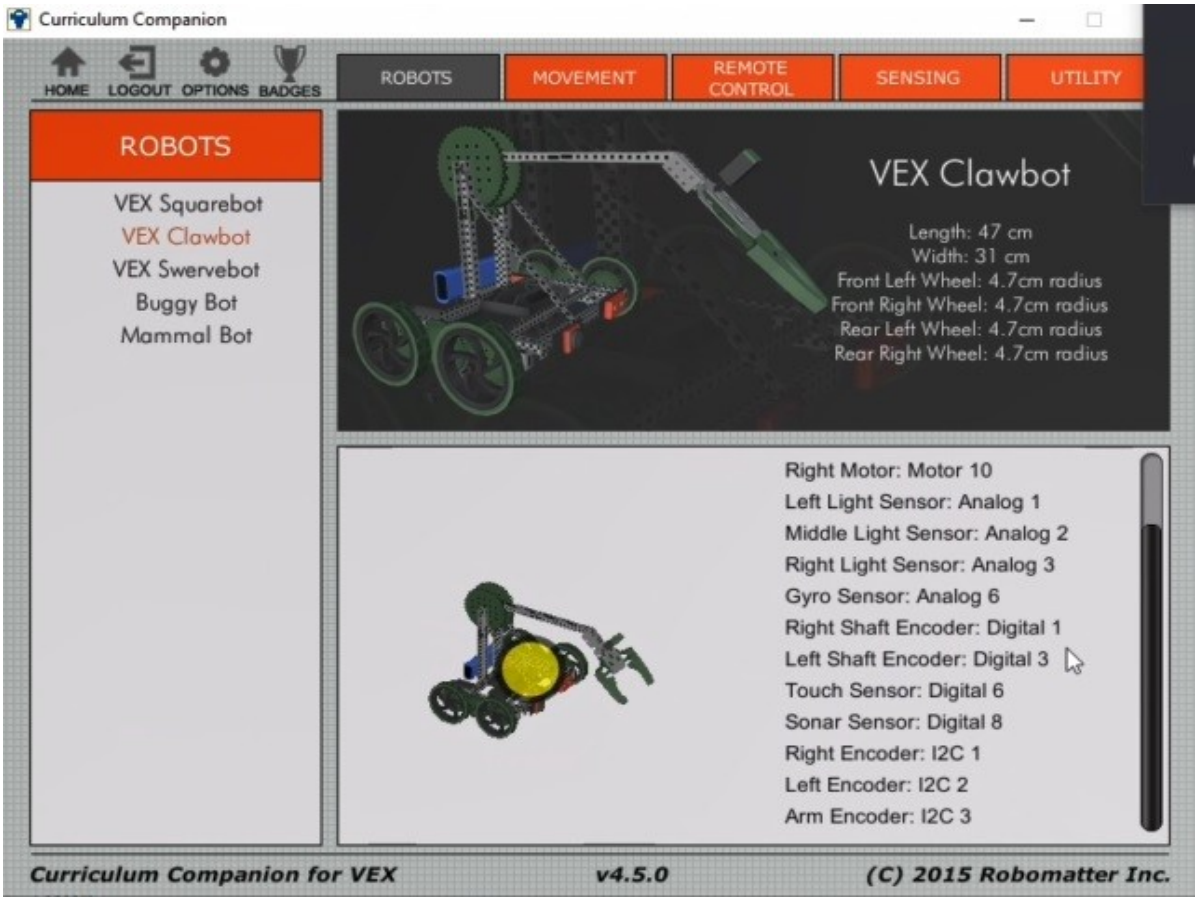

Рисунок 5: Схема подключения датчиков робота Clawbot

Рисунок 6: Схема поля для робота

4. Вызовите написанную функцию в теле основной программы (task main)
5. Запустите симулятор и проверьте работу функции.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.

4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №3 «Использование датчиков в RobotC: датчик касания»

Цель

Изучение правил подключения и использования датчиков расстояния в RobotC.

Краткие теоретические сведения

Датчиком является самостоятельное устройство, выполняющее измерение и последующее преобразование некоторой физической величины в электрический сигнал. Полученный сигнал можно использовать для управления движением робота с целью реализации автономного режима его работы. В данном режиме предполагается, что робот не управляется пользователем напрямую, а самостоятельно, на основе заранее созданной программы, принимает решения об изменении своего поведения в зависимости от различных ситуаций. Так может быть построен робот, который умеет самостоятельно объезжать препятствия. Распознавание препятствий осуществляется с помощью нескольких видов датчиков, которые можно условно поделить на контактные и бесконтактные. Контактные датчики контактируют с объектом измерения, например, представляют собой обыкновенную кнопку (Bumper Switch) (рис.1), либо концевой переключатель (Limit Switch) (рис.2).

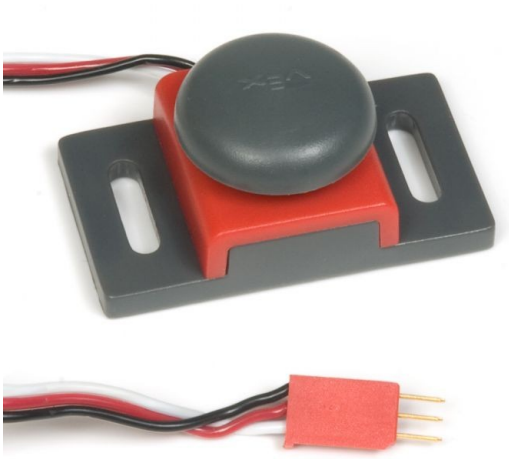

Рисунок 1: Кнопка Bumper Switch

В случае наезда на препятствие, происходит нажатие на кнопку (или рычаг переключателя) и микроконтроллер получает сигнал логической «1». В противном, случае пока это не произошло, микроконтроллер получает сигнал логического «0». Это дает возможность автоматического принятия решения роботом об изменении своих действий при поступлении того или иного сигнала: если поступает сигнал логического «0», то движение продолжается без изменений, если поступает сигнал логической «1», то необходимо остановиться и, например, отъехать назад, так как робот наткнулся на препятствие.

Рисунок 2: Концевой переключатель
Limit Switch

Вторым видом датчиков являются бесконтактные устройства, которые позволяют распознать препятствие без физического контакта. Наиболее характерным примером является ультразвуковой дальномер (рис.3), который, в отличие от кнопки формирует сигнал, имеющий не два значения, а целый диапазон, причем само значение в данном случае будет пропорционально измеренному до препятствия расстоянию. Это позволяет объезжать преграды еще до того, момента, как произойдет касание.

Рисунок 3: Ультразвуковой дальномер
Ultrasonic Range Finder

Использование датчиков начинается с их подключения к роботу (виртуального и физического). Под виртуальным подключением понимается предварительная настройка программы в меню «Motor and Sensors Setup» (рис.4).

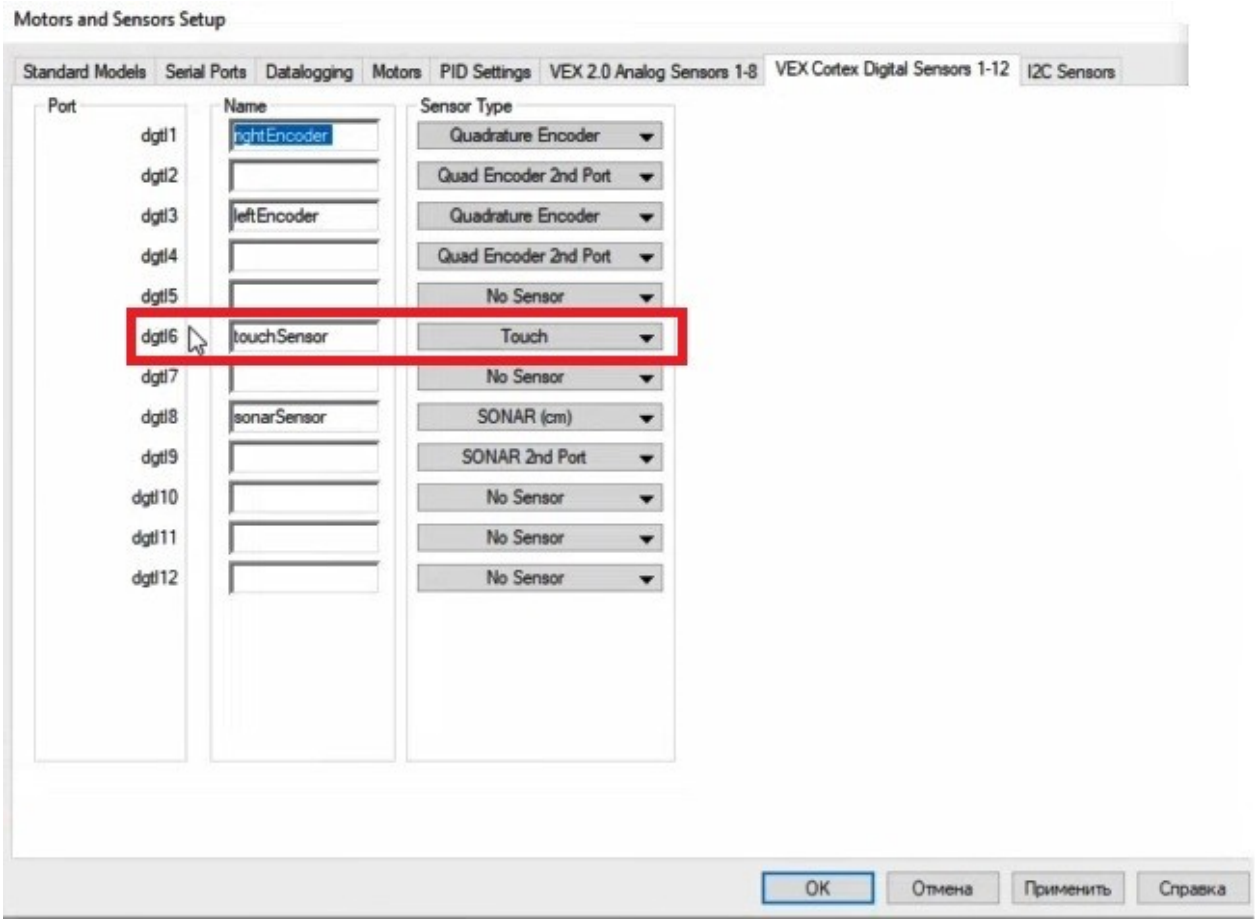

Рисунок 4: Настройка подключения датчика

Во вкладке «VEX Cortex Digital Sensors 1-12» мы выбираем тот порт, в который включен используемый нами датчик (в данном случае – dgtl6), вводим произвольное имя датчика (в данном случае – «touchSensor») и выбираем из раскрывающегося списка тип датчика (в данном случае используется датчик Bumper Switch, поэтому выбираем тип «Touch»). Затем нажимаем кнопку «Применить» и «ОК». После этого программа готова к тому, чтобы использовать данный датчик внутри основного кода.

На языке RobotC обращение к датчику происходит следующим образом. Значение, передаваемое датчиком, хранится в ячейке массива SensorValue, причем, индекс массива совпадает с номером порта, к которому подключен датчик. Таким образом, программа, которая проверяет состояние кнопки Bumper Switch (нажата/не нажата) выглядит следующим образом:

```
if (SensorValue[dgtl6] == 0)
{
```

```
//        выполнение_первого_действия, если SensorValue[dgtl6] равен 0;
}
else
{
  // в противном случае выполнение_второго_действия;
}
```

Аналогично, программа, которая измеряет расстояние с помощью ультразвукового дальномера, выглядит очень похоже:

```
if (SensorValue[dgtl8] < 30)
{
  //выполнение_первого_действия, если SensorValue[dgtl6] меньше 30;
}
else
{
  //в противном выполнение_второго_действия;
}
```

Обратите внимание: в первом случае (работа с кнопкой Bumper Switch) мы проверяем, нажата она или нет – в этом случае используется знак «двойного равно» («==»). Этот знак отличается от «одиночного равно» («=») тем, что «двойное равно» имеет своей целью «спросить» у датчика – равны ли его показания тому числу, которое стоит справа от знака? В то же время, «одиночное равно» имеет своей целью присвоить переменной («положить» в переменную) то значение, которое стоит справа от знака. Во втором случае (работа с ультразвуковым дальномером) мы проверяем, насколько далеко робот находится от препятствия (в данном примере расстояние должно быть меньше 30) – тогда происходит выполнение_первого_действия (например, движение вперед). В тот момент, когда условие перестанет выполняться – а это может случиться только тогда, когда робот приблизится на определенное расстояние к препятствию (в данном случае это расстояние равно 30) – происходит выполнение_второго_действия (например, остановка и разворот).

Работа с кнопкой происходит следующим образом. Если кнопка Bumper Switch, подключенная в порт dgtl6 не нажата (if (SensorValue[dgtl1] == 0)), то происходит выполнение_первого_действия. В противном случае т.е. когда

кнопка нажата (а это возможно только в случае физического наезда на препятствие) происходит выполнение_второго_действия, под которым можно понимать, например, разворот робота в противоположную сторону.

Рассмотрим пример программы, которая решает следующую задачу. Робот Clawbot движется вперед до тех пор, пока не натолкнется на стену. В этом случае он должен остановиться и откатиться назад.

Текст готовой программы:

```
#pragma config(Sensor, dgtl1, Bumper, sensorDigitalIn)
#pragma config(Motor, port1, MotorLeft, tmotorVex393_HBridge, openLoop,
driveLeft)
#pragma config(Motor, port6, MotorRight, tmotorVex393_MC29, openLoop,
reversed, driveRight)
//*!!Code automatically generated by 'ROBOTC' configuration wizard!!*//
void moving(int LeftMotor, int RightMotor, int time)
{
 setMotor(port1, LeftMotor);
 setMotor(port6, RightMotor);
 wait1Msec(time);
}
task main() // основной код всегда содержит функцию main()
{
 while(1) // цикл работающий бесконечное количество раз
 {
        if(SensorValue[dgtl6] == 0)
        {
                moving(50,50,600);
        }
        else
        {
                moving(-50,-50,600);
        }
 }
}
```

После запуска симулятора можно увидеть выполнение написанной программы (рис.5).

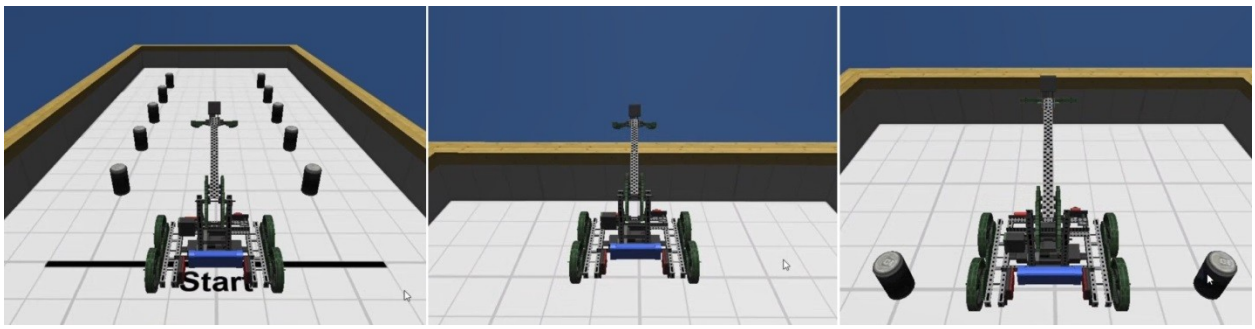

Рисунок 5: Пример работы программы

Задание

Скорректируйте программу так, чтобы робот после наезда на препятствие развернулся на 180°.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup»: подключите кнопку Bumper Switch (в симуляторе она называется «Touch Sensor») в соответствии с моделью робота Clawbot (рис.6). В данной работе следует выбрать поле Driving Straight I (рис.7).
3. Напишите программу, которая позволит роботу после наезда на препятствие не просто откатиться назад, а развернуться на 180°. Подумайте, чем будет отличаться такая программа от той, что была рассмотрена в разделе «Краткие теоретические сведения»?
4. Запустите симулятор и проверьте работу программы.

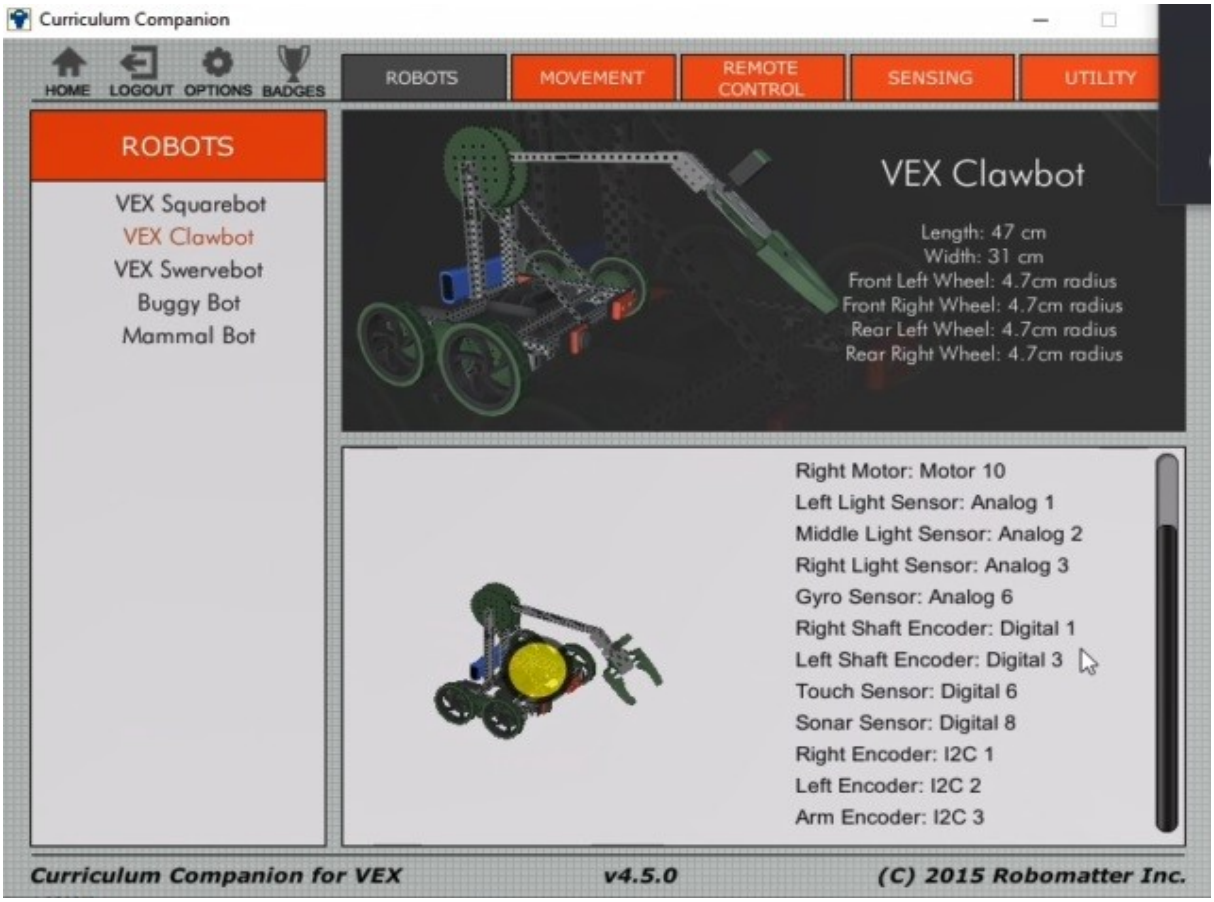

Рисунок 6: Модель робота Clawbot

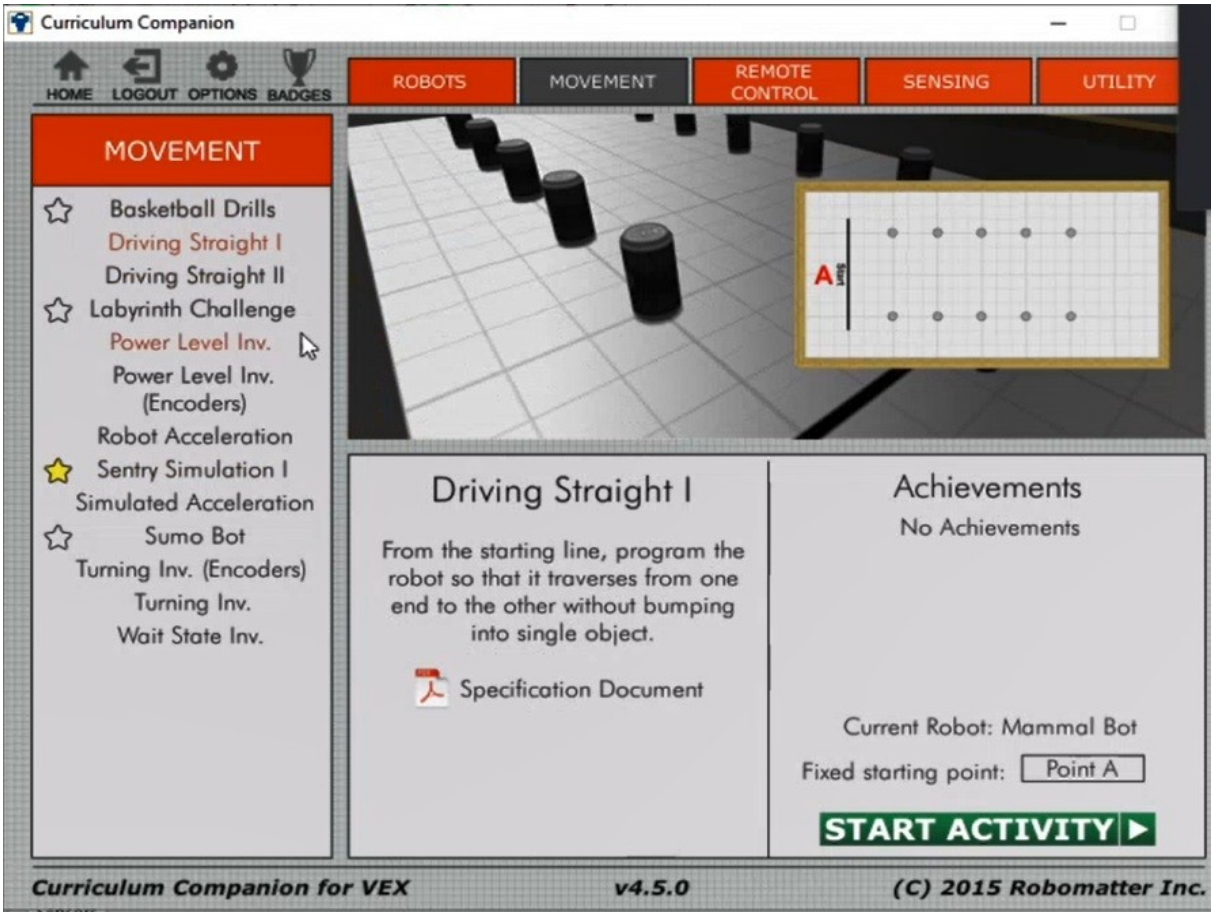

Рисунок 7: Схема поля для робота

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №4 «Использование датчиков в RobotC: энкодер»

Цель

Изучение правил подключения и использования энкодеров в RobotC.

Краткие теоретические сведения

Энкодер – это устройство, которое работает совместно с мотором и имеет своей целью измерение угла поворота оси мотора. Для чего может понадобиться такое измерение? Если нам необходимо, чтобы робот проехал на точно определенное расстояние, то можно пойти двумя путями. Первый (простой) путь заключается в том, что мы попросту вызываем функцию движения робота (наподобие той, что мы рассматривали во второй лабораторной работе) в течение определенного времени, устанавливаемого функцией задержки wait1Msec – мы можем косвенно задать необходимое нам расстояние через время движения робота. В этом случае принято говорить, что «робот управляется по задержке». Такой способ является самым простым и может быть полезен при решении легких задач, которые не требуют точного результата. Робот, действительно, проедет на нужное нам расстояние, но погрешность будет достаточно высокой т.е. запуская робота каждый раз с помощью одной и той же программы, мы каждый раз будем получать результаты (пройденное расстояние), величина которых будет отличаться друг от друга очень существенно. В этом и заключается главный недостаток такого способа. Для его устранения появляется второй способ: в тот мотор, которым хотим управлять точно, встроено специальное устройство, позволяющее измерить угол поворота оси мотора – энкодер. Данное устройство имеет две разновидности – встраиваемый энкодер (рис.1) и оптический энкодер (рис.2).

Рисунок 1: Мотор VEX393 со встроенным энкодером

Рисунок 2: Оптический энкодер

На рисунке 1 показан мотор с уже встроенным энкодером – по сути, у мотора снимается верхняя крышка и вместо нее вставляется другая крышка, внутри которой, собственно, и расположен энкодер. Поэтому внешне такой мотор будет выглядеть практически так же, как и мотор без энкодера. Иначе обстоит дело с оптическим энкодером: он представляет собой самостоятельное устройство, которое «нанизывается» на ось мотора (и для этого у него сделано специальное отверстие в центре корпуса). Управление обоими типами энкодеров происходит похожим образом, но, конечно же, есть определенные особенности, которые мы рассмотрим далее.

Так же, как и при работе с датчиками, перед началом программирования необходимо явным образом подключить энкодер. Если используется встроенный энкодер, то выполняется дополнительная настройка подключения

мотора (рис.3). Физически к микроконтроллеру подключается только один энкодер (первый) – все последующие энкодеры подключаются друг к другу последовательно и в этом случае необходимо указать в меню «Motors and Sensors Setup» соответствующий номер порта: I2C_2, I2C_3 и т.д. Если же используется оптический энкодер, то он подключается так же, как и цифровой датчик (рис.4).

Обращение к встроенному энкодеру происходит с помощью функции nMotorEncoder(имя_мотора). Данная функция возвращает значение, равное текущему показанию энкодера – то, что нам нужно: значение угла поворота оси мотора. Нужно отметить, что возвращаемое значение не имеет размерности угла – его можно рассматривать как безразмерный параметр, значение которого пропорционально углу (но никак не равно ему). Соответственно, располагая информацией даже о таком, безразмерном, значении угла можно решать самые разнообразные задачи по управлению мотором. Обратите внимание: при обращении к встроенному энкодеру мы используем имя мотора – как такового имени у самого энкодера не существует!

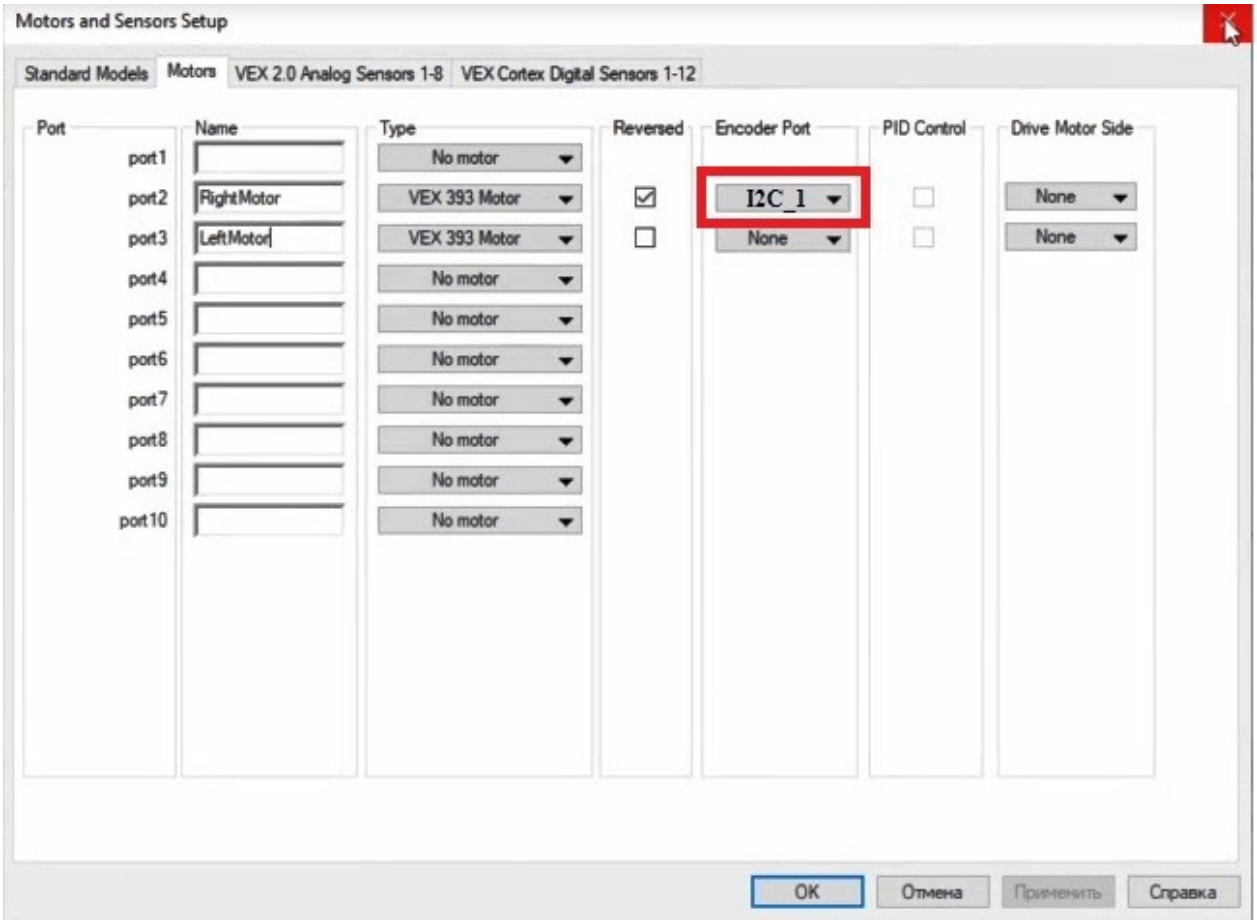

Рисунок 3: Подключение встроенного энкодера

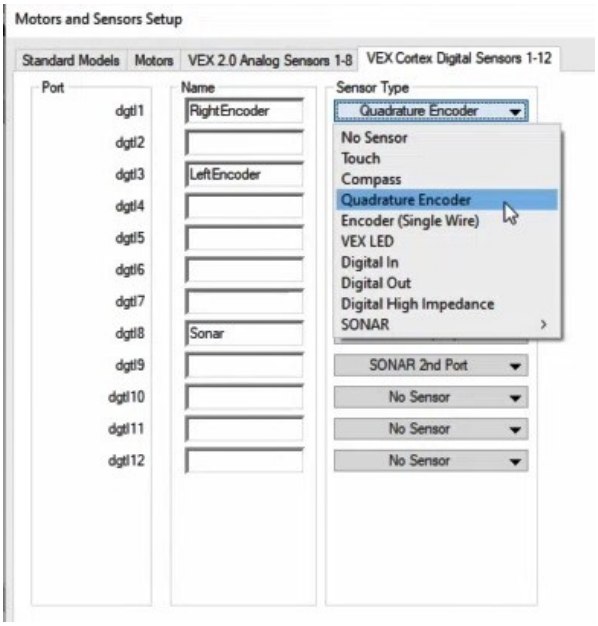

Рисунок 4: Подключение оптического энкодера

Обращение к оптическому энкодеру происходит через другую функцию – Sensor Value(имя_энкодера). В данном случае энкодер рассматривается в

качестве датчика, поэтому у него появляется отдельное имя (которое мы заранее задали в меню «Motors and Sensors Setup»). Соответственно, такой датчик возвращает значение угла поворота оси мотора – опять же, безразмерное значение.

Далее мы используем полученное с энкодера значение для решения задач робототехники: строим циклы, формируем ветвления и пр. В качестве примера рассмотрим следующую программу.

```
#pragma config(I2C_Usage, I2C1, i2cSensors)
#pragma config(Sensor, I2C_1, sensorQuadEncoderOnI2CPort, , AutoAssign)
#pragma config(Sensor, I2C_2, sensorQuadEncoderOnI2CPort, AutoAssign)
#pragma config(Motor,  port1, MotorLeft, tmotorVex393_HBridge, openLoop,
driveLeft, encoderPort, I2C_1)
#pragma config(Motor,  port10, MotorRight, tmotorVex393_HBridge,
openLoop, driveRight, encoderPort, I2C_2)
//*!!Code automatically generated by 'ROBOTC' configuration wizard!!*//
void moving(int LeftMotor, int RightMotor, int time, int angle)
{
  nMotorEncoder[port1] = 0;
  while(nMotorEncoder[port1] < angle)
  {
          setMotor(port1, LeftMotor);
          setMotor(port10, RightMotor);
          wait1Msec(time);
  }
setMotor(port1, 0);
setMotor(port10, 0);
}

task main()
{
  moving(50, 50, 600, 2000);
}
```

В описанной выше программе мы создали функцию moving, которая принимает на вход четыре аргумента: LeftMotor (скорость левого мотора), RightMotor (скорость правого мотора), time (задержка, длительность работы моторов), angle (угол поворота оси мотора). В принципе, такая функция работает почти так же, как функция moving, описанная в предыдущих

лабораторных работах. Отличие заключается в том, что мы используем встроенный энкодер с целью измерения угла поворота оси мотора, а вместе с ним – с целью обеспечения контроля расстояния, пройденного роботом. В этом случае мы имеем возможность существенно повысить точность измерения пройденного расстояния по сравнению с работой «по задержке». Работа с энкодером начинается с того, что мы обнуляем его показания – ведь до начала работы мы не можем знать, в каком положении находится энкодер и нам крайне важно быть уверенными в том, что стартовой точкой для работы с энкодером является именно 0, а не какое-то другое значение:

```
nMotorEncoder[port1] = 0;
```

Обратите внимание: при обращении к энкодеру мы не используем имя мотора (хотя это разрешено). Мы используем альтернативный способ обращения – через номер порта, к которому подключен мотор.

Следующим шагом мы создаем цикл, условием выполнения которого является достижение такого значения угла энкодера, которое будет меньше, чем заданное нами в качестве аргумента функции значение переменной angle:

```
while(nMotorEncoder[port1] < angle)
```

Это условие будет выполняться до тех пор, пока мотор не «накрутит» нужное нам значение угла, и, соответственно, не пройдет определенное расстояние. Как только условия цикла нарушатся, цикл прекратится и программа продолжит работу дальше — код после цикла. Следующая часть кода (тело цикла) представляет собой уже известный нам вызов встроенных функций setMotor и wait1Msec с целью управления моторами робота. Соответственно, в теле основной функции task main() мы просто вызываем созданную нами функцию и указываем нужные нам значения аргументов:

```
moving(50, 50, 600, 2000);
```

Описанная выше программа будет работать следующим образом. Робот будет ехать вперед со скоростью, равной 50, в течение 600 миллисекунд до тех пор, пока значение угла поворота оси мотора не достигнет 2000.

Задание

Скорректируйте программу так, чтобы робот проехал вперед ровно на 40 см при условии, что диаметр колеса равен 4.7 см.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup»: два мотора (правый и левый) в соответствии с моделью робота Clawbot (рис.5), а также встроенный энкодер для одного из них.
3. Напишите программу, которая позволит роботу проехать ровно 1 метр при условии, что диаметр колеса известен – 10 см. Подумайте, как можно рассчитать пройденное роботом расстояние на основании приведенных данных?
4. Запустите симулятор и проверьте работу программы.

Рисунок 5: Модель робота Clawbot

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №5 «Пропорциональный регулятор»

Цель

Изучение правил создания и использования пропорционального регулятора в RobotC.

Краткие теоретические сведения

Прежде, чем мы начнем рассматривать особенности создания регулятора на языке RobotC и, вообще, что это такое, обратимся к одной из реальных задач робототехники. Как известно, у робота Clawbot есть «рука» (рис.1), на конце которой размещен механизм захвата («клешня»), с помощью которых он может перемещать различные предметы.

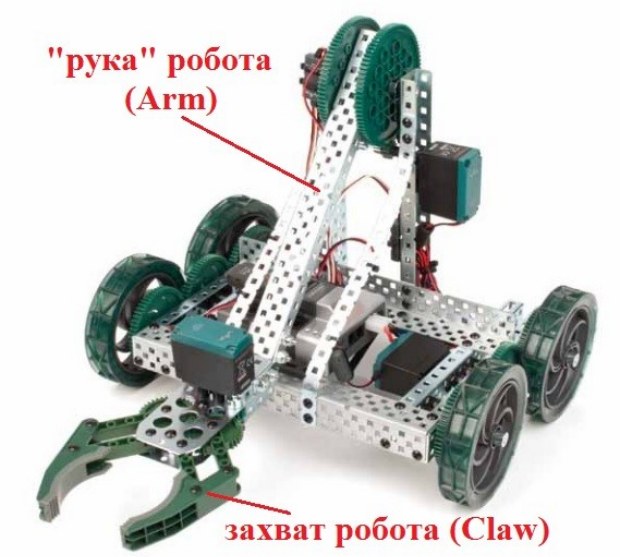

Рисунок 1: Внешний вид робота Clawbot

Очевидно, для этого роботу нужно выбранный предмет зафиксировать («схватить»), поднять на некоторую высоту, затем переместить и поставить. Но есть один нюанс, из-за которого такая простая (с виду) задача становится достаточно сложно реализуемой программой. «Руку» нужно не просто поднять на определенную высоту, а еще и удержать на этой высоте. Казалось бы, нужно всего лишь подключить мотор и, «покрутив» этим мотором, поднять «руку» на нужную высоту. Нет. В таком случае «рука», конечно же, будет поднята, но после того, как мотор остановится, она упадет обратно под собственной

тяжестью. Тем более, это произойдет в том случае, если в захвате робота будет находиться перемещаемый предмет (который добавит веса для всей конструкции). Для решения такой задачи нужно управлять мотором особым образом и такое решение называется «пропорциональный регулятор». Суть решения заключается в следующем. На первом шаге пользователь задает целевое значение угла поворота «руки» – с помощью значения этого угла можно косвенным путем задать высоту подъема захвата робота. Конечно же, угол поворота контролируется с помощью энкодера.

Далее организуется цикл, условием выполнения которого является достижение целевого значения угла. Причем, скорость мотора указывается не постоянной, а переменной – она прямо пропорционально зависит от разницы текущего положения энкодера и целевого значения угла. Этот параметр называется с помощью определенного термина – «ошибки». Чем больше ошибка – тем больше скорость мотора. По мере того, как мотор поднимает «руку» и приближает ее положение к целевому значению угла, значение ошибки уменьшается и, соответственно, скорость мотора замедляется. Но при достижении целевого значения мотор не сможет остановиться мгновенно и захват «перескочит» заданный угол – в этот момент ошибка сменит знак и, соответственно, мотор начнет перемещать «руку» в противоположном направлении. Таким образом, «рука» будет совершать колебания относительно целевого значения угла, но размер этих колебания будет настолько мал, что визуально будет казаться, что захват стоит на определенной высоте и никуда не смещается. В этом случае можно нагрузить «руку» каким-либо предметом и она не будет при этом падать – мотор не прекращает свою роботу и держит конструкцию непрерывно.

Наконец, последнее, на что следует обратить внимание. Как было отмечено выше, для того, чтобы «рука» не падала под собственным весом необходимо, чтобы мотор не прекращал свою работу и постоянно работал, контролируя ее положение. То есть у нас возникает необходимость в использовании особой

конструкции на языке RobotC, которая называется «параллельной программой». По сути, такое программное решение представляет собой функцию, которая по своему наполнения ничем не отличается от пользовательской функции, но при этом не имеет аргументов (ведь это же просто фрагмент кода) и вызывается с помощью ключевого слова task. Имя такой функции, как и в случае с пользовательской, задается пользователем произвольно. Ниже приведем пример такой функции, которую мы подробно разберем.

```
task ArmUp()
{
  while(nMotorEncoder(Arm) > -200)
  {
                motor[Arm] = -2*(-200 - nMotorEncoder(Arm));
                wait1Msec(10);
  }
}
```

Итак, функция с именем «ArmUp» отнесена нами к категории программ, выполняемых параллельно с остальной частью кода, и имеет своей целью подъем и удержание «руки» робота на определенной высоте. Для контроля угла поворота оси мотора, управляющего «рукой», используется встроенный энкодер, который подключен к мотору с именем «Arm» – для этого мы устанавливаем соответствующий переключатель в меню Motors and Sensors Setup (рис.2).

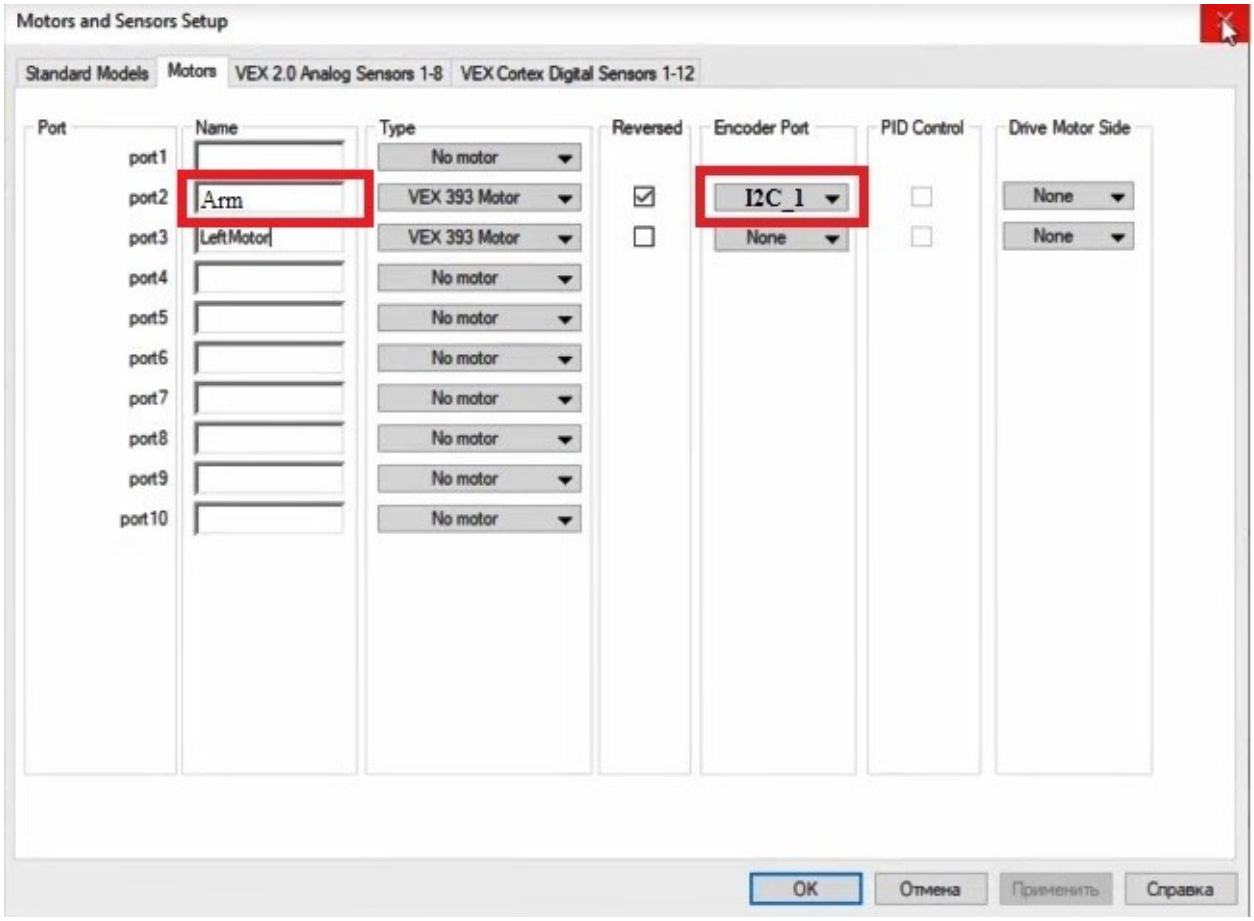

Рисунок 2: Настройка подключения энкодера

Соответственно, после того, как в порт port2 был подключен мотор, управляющий «рукой», и для него было задано имя «Arm» мы можем внутри всей программы обращаться к этому мотору не через номер порта, а через заданное нами имя (что, несомненно, проще). До тех пор, пока «рука» робота не достигла целевого значения угла поворота, равного -200 условие цикла while выполняется, и мы попадаем в тело цикла, которое состоит из двух строк – обращение к мотору Arm и задержки. Причем, скорость вращения мотора Arm не является фиксированной, наоборот, она «привязана» к показаниям энкодера. В начальный момент времени показания энкодера равны нулю, следовательно, мотор будет вращать ось со скоростью, равной -400. Поскольку показания энкодера изменяются в отрицательную строну, то в следующий момент времени, когда «рука» поднимется на небольшую высоту, показания энкодера будут уже отличны от нуля и, после преобразования согласно приведенной формуле, скорость мотора уменьшится. Так будет происходить до тех пор, пока «рука» не достигнет положения целевого угла – в этом случае скорость мотора

станет равна нулю. Множитель «-2» является коэффициентом усиления регулятора, с помощью которого можно настраивать скорость завершения переходного процесса т.е. можно отрегулировать, насколько быстро и резко (или медленно и плавно) «рука» достигнет положения целевого угла.

Обратите внимание: в описанном выше случае «рука» будет зафиксирована только снизу т.е. мотор не будет давать ей упасть. В том случае, если необходимо зафиксировать «руку» с обеих сторон, следует написать несколько иную функцию:

```
task ArmUp()
{
 if(nMotorEncoder(Arm) > -200)
 {
                motor[Arm] = -1*(-400 - nMotorEncoder(Arm))/10;
                wait1Msec(10);
 }
 else
 {
                motor[Arm] = 1*(-400 - nMotorEncoder(Arm))/10;
                wait1Msec(10);
 }
}
```

Так же, как и в прошлом случае, мы в качестве скорости мотора задаем функцию, зависящую от показаний энкодера. Отличие заключаются в том, что мы реализуем регулятор не через цикл while, а с помощью ветвления if-else. Если показания энкодера говорят о том, что «рука» еще не достигла целевого положения, то мы «крутим» мотор в сторону подъема. Если же «рука» пересекла пороговое значение, то мотор должен вращаться в противоположную сторону. В этом случае можно увидеть, что «рука» будет совершать рывки около целевого положения, поэтому, чтобы сделать ее движения более плавными, коэффициент усиления был уменьшен до «-1». Кроме того, был введен дополнительный коэффициент, равный 10, а с учетом этого, первое слагаемое в скобках (-400) теперь отличается от целевого значения (-200). В совокупности, перечисленные меры привели к тому, что теперь «рука» поднимается на определенную высоту и фиксируется с обеих сторон.

Полный текст программы приведен ниже:

```
#pragma config(I2C_Usage, I2C1, i2cSensors)
#pragma config(Sensor, I2C_1, sensorQuadEncoderOnI2CPort, AutoAssign )
#pragma config(Motor, port1, MotorRight, tmotorVex393_HBridge, openLoop)
#pragma config(Motor, port6, Claw, tmotorVex393_MC29, openLoop)
#pragma config(Motor, port7, Arm, tmotorVex393_MC29, openLoop,
encoderPort, I2C_1)
#pragma config(Motor, port10, MotorLeft, tmotorVex393_HBridge, openLoop,
reversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
task ArmUp()
{
 if(nMotorEncoder(Arm) > -200)
 {
                motor[Arm] = -1*(-400 - nMotorEncoder(Arm))/10;
                wait1Msec(10);
 }
 else
 {
                motor[Arm] = 1*(-400 - nMotorEncoder(Arm))/10;
                wait1Msec(10);
 }
}
task main()
{
 resetMotorEncoder(Arm);
 startTask(ArmUp);
}
```

Последнее, что нужно отметить, это то, что перед вызовом функции task нужно обязательно сбросить показания энкодера с помощью команды resetMotorEncoder(Arm). В противном случае, в качестве нулевого, стартового значения энкодера будет использоваться то значение, которое сохранилось в его памяти от прошлой программы.

Задание

Напишите программу, реализующую пропорциональный регулятор на оптическом энкодере, с помощью которого робот будет ехать на заданное расстояние с ускорением.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов и датчиков в меню «Motors and Sensors setup» в соответствии с моделью робота Clawbot (рис.3).
3. Напишите программу, которая позволит роботу проехать 50 см при условии, что движение должно не равномерным, а с ускорением.
4. Запустите симулятор и проверьте работу программы.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Рисунок 3: Модель робота Clawbot

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.

2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №6 «Использование датчиков в RobotC: ультразвуковой дальномер»

Цель

Изучение правил подключения и использования ультразвукового дальномера в RobotC.

Краткие теоретические сведения

Ультразвуковой дальномер (Ultrasonic Range Finder) (рис.1) представляет собой датчик, выполняющий функцию по определению расстояния до препятствия бесконтактным способом. Это позволяет роботу избегать препятствий на его пути с помощью высокочастотных звуковых волн. Ультразвуковой дальномер производит звуковую волну частоты 40 кГц, которая, при столкновении с препятствием, отражается и возвращается к сенсору, но не слышна человеческом ухом. Затем, зная скорость распространения звука в воздухе, и на основании времени, которое потребовалось волне, чтобы вернуться, вычисляется расстояние до предполагаемого объекта. Этот сенсор можно применять в качестве инструмента отслеживания любых объектов на пути следования робота.

Рисунок 1: Ультразвуковой дальномер
Ultrasonic Range Finder

Первое, что нужно сделать при работе с ультразвуковым дальномером – выполнить предварительную настройку в меню «Motor and Sensors Setup» (рис.2). В соответствии с моделью робота выбираем один из цифровых портов, задаем имя для датчика (в данном случае «Sonar») и выбираем из раскрывающегося списка тип датчика («Sonar») и единицы измерения, которые будут использоваться дальномером во время работы (в данном случае выбраны миллиметры).

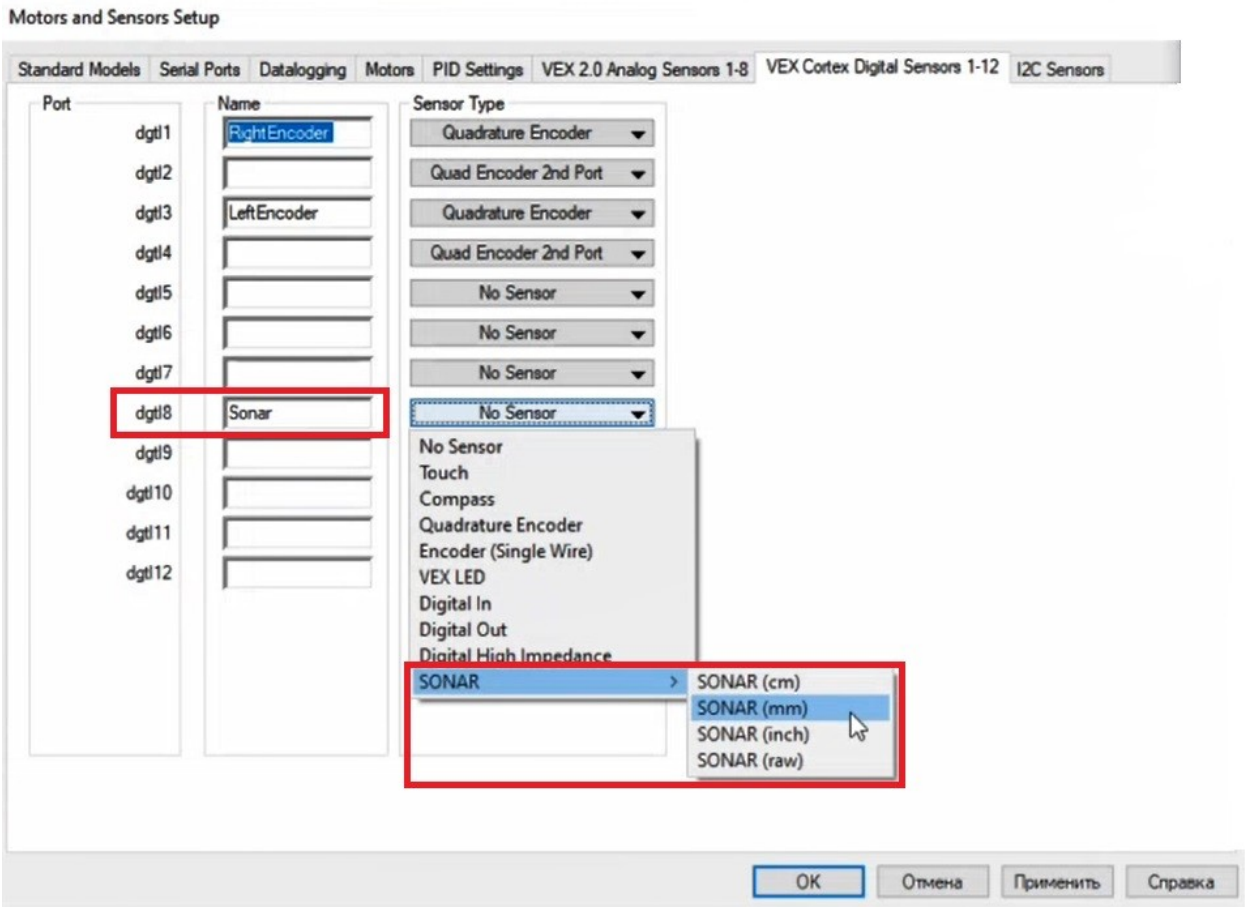

Рисунок 2: Настройка подключения датчика

Обращение к датчику происходит через команду SensorValue с указанием номера порта, к которому он подключен, либо через заданное имя. Таким образом, программа, которая использует данные с дальномера для контроля расстояния до препятствия, выглядит следующим образом:

```
if (SensorValue[dgtl8] > 30)
{
  //выполнение_первого_действия;
}
```

```
else
{
  //выполнение_второго_действия;
}
```

В качестве альтернативы можно реализовать конструкцию с помощью цикла:

```
while (SensorValue[dgtl8] > 30)
{
  //выполнение_действия;
}
```

В обоих случаях использование показаний дальномера дает возможность для реализации автономного режима работы робота: можно закрепить несколько датчиков в разных местах и реализовать автоматический объезд препятствий.

Очевидно, что при работе с дальномером (как, впрочем, и с любым датчиком) неизбежно возникает необходимость отладки кода для устранения возникающих ошибок. И в этом случае крайне важно видеть содержимое дальномера в каждый момент: какое расстояние он измерил в разных ситуациях, где находятся «мертвые» зоны, насколько быстро он успевает выполнить измерение. Для ответов на эти и подобные вопросы в RobotC есть возможность запуска отладчика – специального окна, с помощью которого можно увидеть содержимое всех переменных или датчиков и выполнить программу по шагам (рис.3). Отладчик запускается автоматически при управлении роботом по USB-кабелю (при управлении с помощью джойстика такая возможность отсутствует). Соответственно, перед тем, как начать управление с джойстика, необходимо полностью устранить все возможные ошибки в коде.

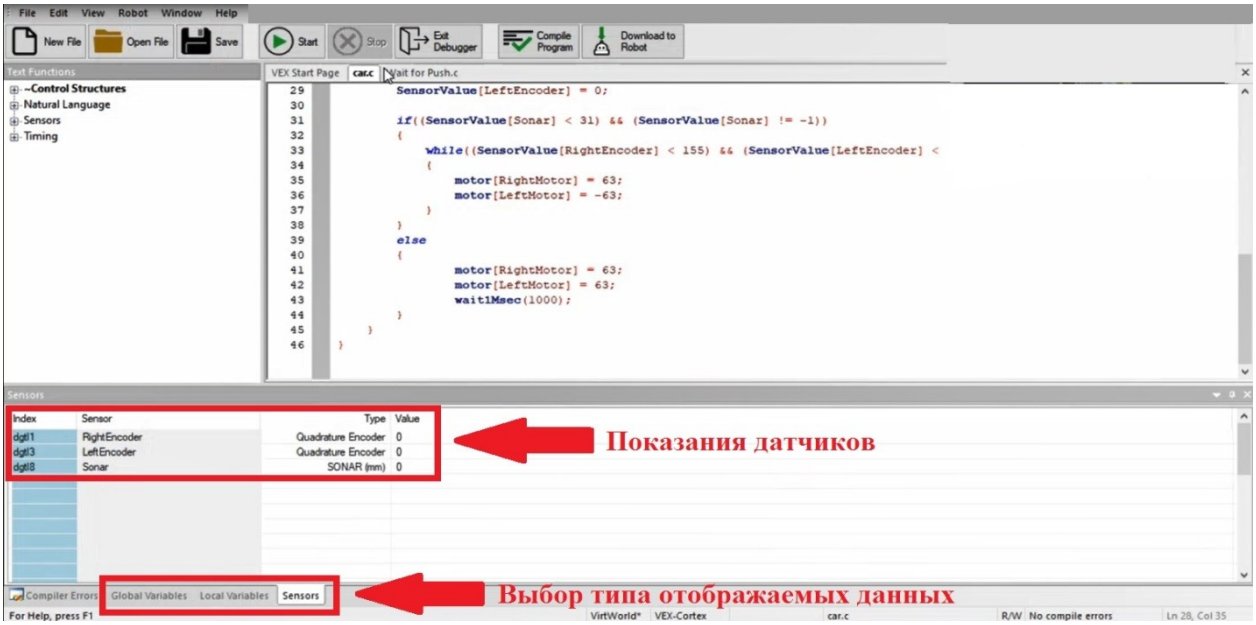

Рисунок 3: Работа с отладчиком

Обратите внимание – для выбора типа отображаемых данных доступны 3 вкладки: «Global Variables» (глобальные переменные), «Local Variables» (локальные переменные) и «Sensors» (датчики). Глобальные переменные – переменные, которые «видны» из любой части программы, локальные переменные доступны только внутри локальной области программы (например, внутри тела функции). Соответственно, показания датчиков в реальном времени можно увидеть двумя способами: напрямую, через вкладку «Sensors», либо косвенным путем, через глобальную переменную. Этот способ заключается в том, что мы «кладем» в заранее объявленную глобальную переменную содержимое нужного нам датчика и затем в отладчике видим ее содержимое во вкладке «Global Variables». Такой способ подходит в том случае, если невозможно увидеть содержимое датчика напрямую, например, при использовании встроенного энкодера.

Рассмотрим пример программы, которая решает следующую задачу. Робот Clawbot движется вперед до тех пор, пока расстояние до стены не станет равно 30 мм. После достижения указанного расстояния он должен остановиться.

Текст готовой программы:

```
#pragma config(Sensor, dgtl8,  Sonar, sensorSONAR_mm)
```

```
#pragma config(Motor,  port1, LeftMotor, tmotorVex393_HBridge, openLoop,
reversed)
#pragma config(Motor,  port6, Claw, tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port7, Arm, tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10, RightMotor, tmotorVex393_HBridge,
openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration
wizard               !!*//
void moving (int LeftMotor, int RightMotor, int time)
{
setMotor (port1, LeftMotor);
setMotor (port6, RightMotor);
 wait1Msec (time);
setMotor (port1, 0);
setMotor (port6, 0);
wait1Msec (30);
}
task main()
{
  while(SensorValue(Sonar) > 30)
  {
          moving(45, 45, 100);
  }
}
```

Задание

Скорректируйте программу так, чтобы робот совершил «вальс»: он должен делать круговые движения, сканировать пространство на предмет приближения к препятствиям (стены и столбики) и при их обнаружении отъезжать назад с одновременным поворотом в сторону.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup» в соответствии с моделью робота Clawbot (рис.4). В данной работе следует выбрать поле Driving Straight I (рис.5).
3. Напишите программу, которая позволит роботу совершить «вальс»: он должен двигаться вперед до тех пор, пока расстояние до препятствия (стены и столбики) не станет равно 30 мм, после чего необходимо отъехать назад и совершать поворот вправо до тех пор, пока расстояние

до стены не станет больше 30 мм. Затем опять проехать вперед с контролем расстояния. В результате робот должен совершать вальсирующие движения вокруг своей оси, не задевая при этом препятствия. Главной целью является написание такой программы, которая позволит роботу вернуться в исходное положение после совершения полного оборота на 360°. Финальное положение робота должно быть таким же, как и на старте.

4. Запустите симулятор и проверьте работу программы, также вы можете запустить отладчик и посмотреть как меняются показания датчиков.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

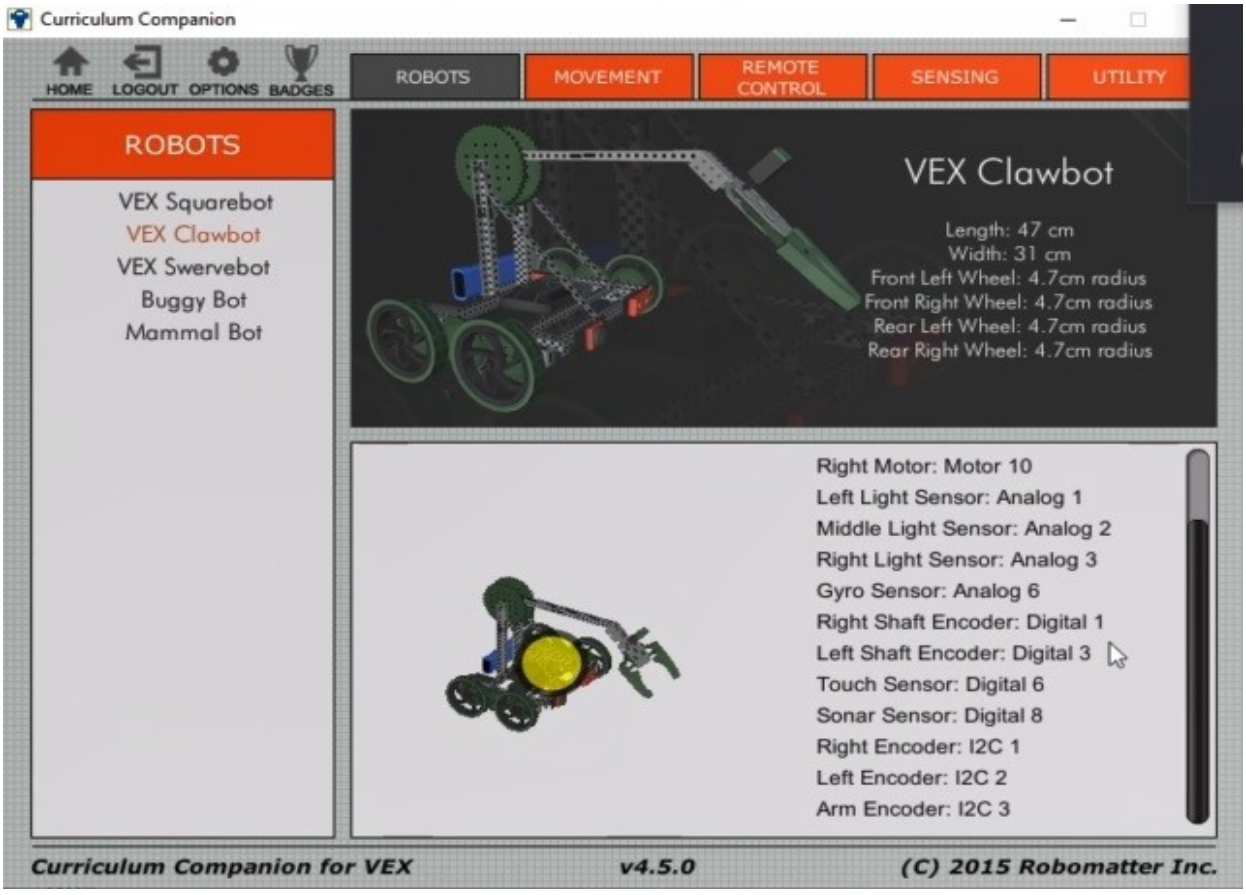

Рисунок 4: Модель робота Clawbot

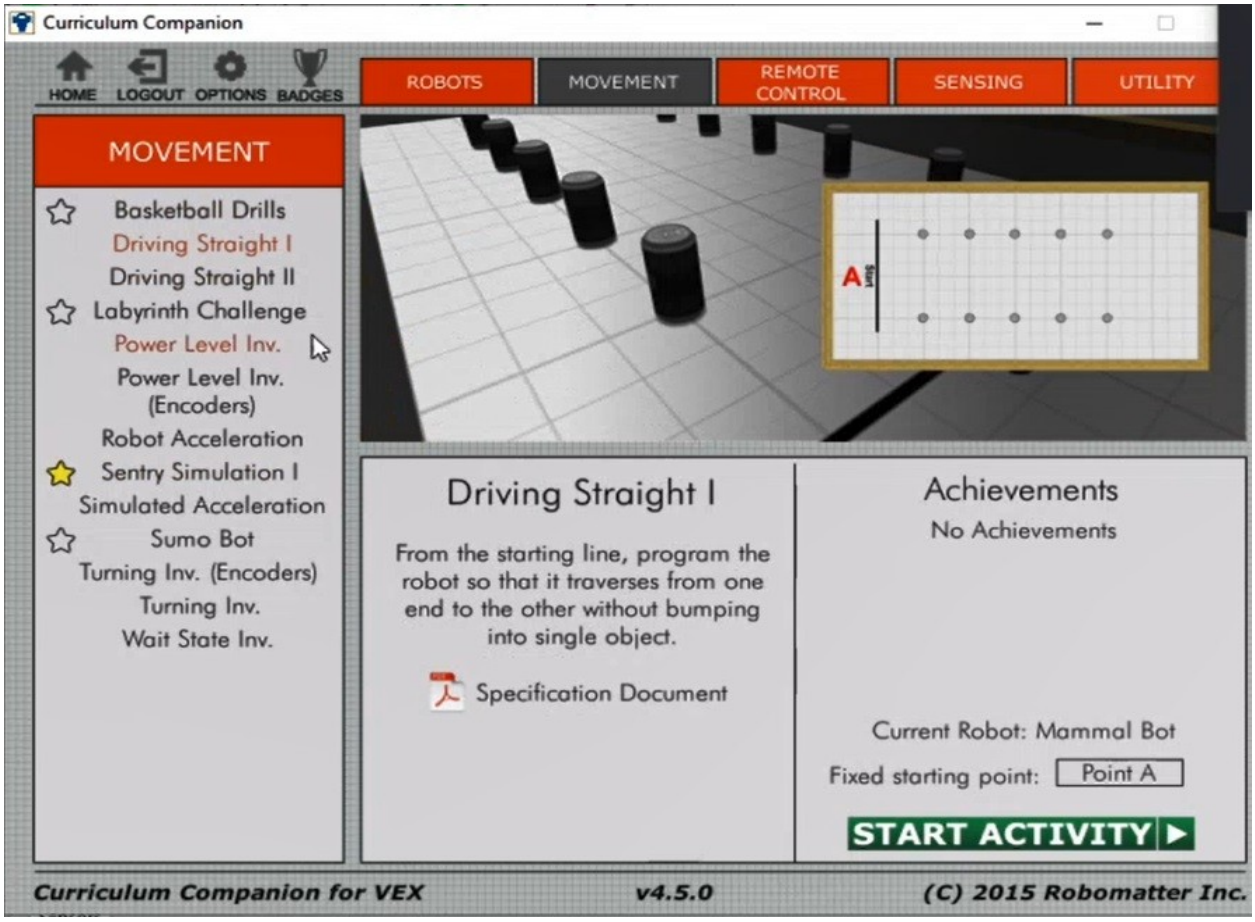

Рисунок 5: Схема поля для робота

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.

2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №7 «Использование датчиков в RobotC: датчик цвета»

Цель

Изучение правил подключения и использования цветового датчика в RobotC.

Краткие теоретические сведения

Датчик цвета (Line Follower) (рис.1) представляет собой датчик, выполняющий функцию по определению цвета анализируемой поверхности бесконтактным способом. В первую очередь, такой датчик находит применение в роботе, задачей которого является движение по траектории, нарисованной на полу в виде линии определенного цвета. Отсюда и второе название датчика – «Line Follower». Такой датчик крепится к основанию робота «лицом вниз» максимально близко к полу с тем, чтобы распознавание цвета поверхности было наиболее точным. Также, для повышения точности, используют не один, а сразу несколько таких датчиков – их ставят рядом друг с другом, и в этом случае зона контроля становится больше.

Рисунок 1: Датчик цвета Line Follower

Первое, что нужно сделать при работе с датчиком цвета – выполнить предварительную настройку в меню «Motor and Sensors Setup» (рис.2). В соответствии с моделью робота выбираем один из цифровых портов (например, порт «in4»), задаем имя для датчика (в данном случае «Tracker») и выбираем из раскрывающегося списка тип датчика («Line Follower»).

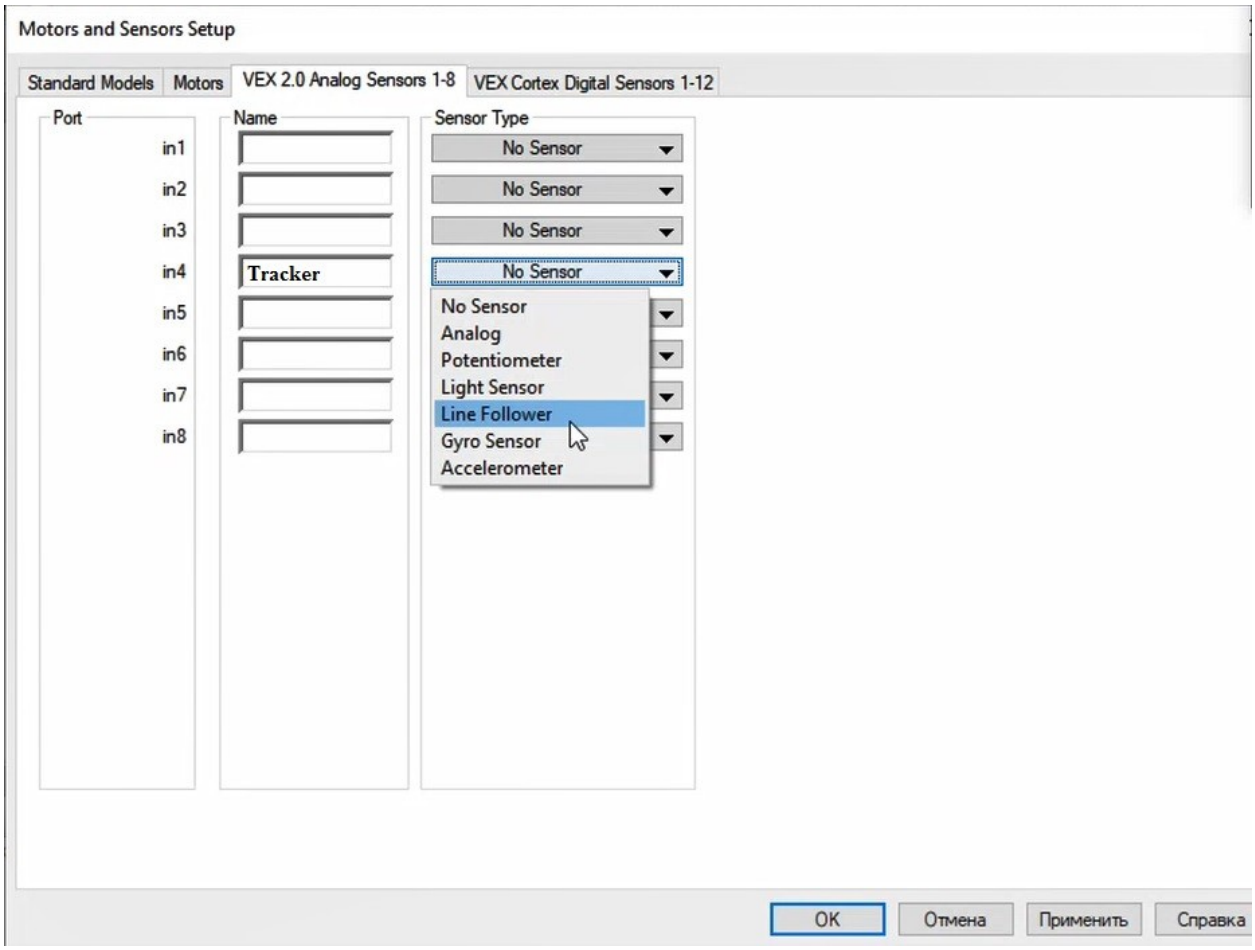

Рисунок 2: Настройка подключения датчика

Обращение к датчику происходит через команду SensorValue с указанием номера порта, к которому он подключен, либо через заданное имя. Таким образом, программа, которая использует данные с датчика цвета для движения по линии, выглядит следующим образом:

```
if (SensorValue[Tracker] > 3000)
{
  //выполнение_первого_действия;
}
else
{
  //выполнение_второго_действия;
```

```
}
```

В качестве программы, реализующей выполнение_первого_действия можно написать код, согласно которому робот будет двигаться вперед со смещением вправо. После того, как условие для ветвления перестанет выполняться, мы попадем в ветку else и, соответственно, начнет работать программа, реализующая выполнение_второго_действия. Таким образом, робот сможет двигаться по траектории любой, даже самой сложной формы.

Пример программы, с помощью которой можно решить подобную задачу, приведен ниже:

```
#pragma config(Sensor, in4, LightSensor,    sensorReflection)
#pragma config(Motor,  port2, RightMotor, tmotorVex393_MC29, openLoop,
reversed)
#pragma config(Motor,  port3, LeftMotor, tmotorVex393_MC29, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
task main()
{
while(1)
{
 if(SensorValue(Tracker) > 3000)
 {
        motor[RightMotor] = 20;
        motor[LeftMotor] = 20;
        wait1Msec(100);
 }
 else
 {
        motor[RightMotor] = -20;
        motor[LeftMotor] = 20;
        wait1Msec(100);
        if(SensorValue(Tracker) < 3000)
        {
                motor[RightMotor] = 20;
                motor[LeftMotor] = -20;
                wait1Msec(150);

                motor[RightMotor] = 20;
                motor[LeftMotor] = 20;
                wait1Msec(150);
        }
```

```
        else
        {
                motor[RightMotor] = 20;
                motor[LeftMotor] = 20;
                wait1Msec(100);
        }
  }
}
}
```

Разберем программу более подробно. Так же, как и в прошлых лабораторных работах, необходимо, помимо настройки подключения датчика цвета, настроить подключение используемых моторов (см. Лабораторную работу №2). В теле основной функции создан бесконечный цикл while(1) и уже внутри этого цикла происходит постоянный опрос датчика цвета Tracker. В том случае, если показания датчика превышают 3000 безразмерных единиц, мы попадаем в первую ветку if, и оба мотора вращают оси колес робота с тем, чтобы он двигался вперед. Робот будет ехать вперед до тех пор, пока условие для попадания в первую ветку if остается справедливым. В какой-то момент робот съедет с линии, условие перестанет выполняться, и мы попадем в первую ветку else. В этом месте робот начнет разворачиваться вправо – для того, чтобы вернуться на линию обратно. И вот здесь возникает интересный момент. Мы же не можем заранее знать, в какую сторону повернула линия и, соответственно, в какую сторону нам сейчас нужно поворачивать: вправо или влево! Не исключена такая ситуация, при которой поворот вправо (собственно, то, что было сделано в программе) не только не вернет робота обратно на линию, но и уведет его совсем в другую сторону! Для того, чтобы отработать подобную ситуацию внутри первого ветвления if-else было введено второе – «вложенное»: именно в с его помощью мы сможем оперативно решить возникшую проблему:

```
if(SensorValue(Tracker) < 3000)
{
motor[RightMotor] = 20;
motor[LeftMotor] = -20;
wait1Msec(150);
```

```
motor[RightMotor] = 20;
motor[LeftMotor] = 20;
wait1Msec(150);
}
else
{
motor[RightMotor] = 20;
motor[LeftMotor] = 20;
wait1Msec(100);
}
```

Учитывая, что показания датчика у нас и так были меньше 3000, следовательно, мы сразу же попадем первую ветку вложенного if – робот повернет в обратную сторону (влево) и затем поедет по прямой. А поскольку вся программа у нас реализована в виде бесконечного цикла, то опрос датчика происходит непрерывно – в совокупности, это и даст возможность проехать по сложной траектории без отклонений.

Результат выполнения приведенной программы показан на рисунке 3.

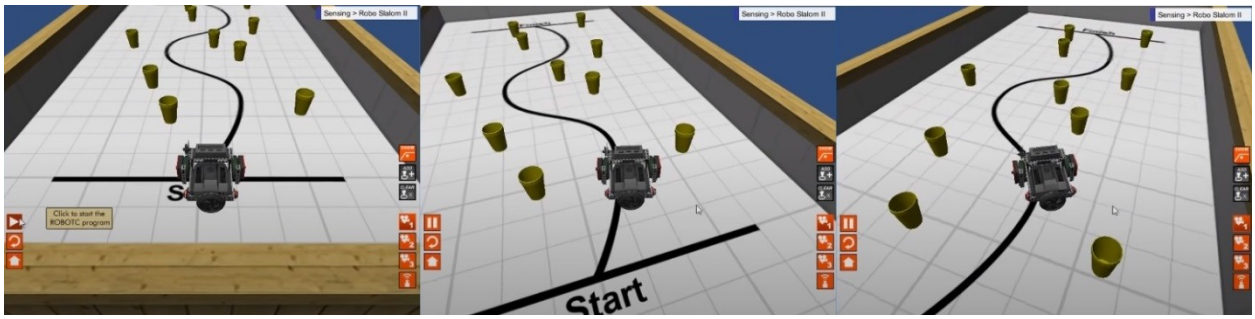

Рисунок 3: Пример работы программы

Задание

Скорректируйте программу так, чтобы робот, после того, как доедет до конца траектории, сделал разворот, затем сумел правильно определить местоположение линии и вернулся в исходную позицию.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup» в соответствии с моделью робота Buggy Bot (рис.4). В данной

работе следует выбрать поле Robo Slalom II. Обратите внимание: в качестве датчика цвета в этой модели робота используется Light Sensor, подключенный в порт «Analog 4».

3. Напишите программу, с помощью которой робот, после того как достигнет конца траектории, сможет не просто развернуться на 1800, а при этом так же сможет «найти» начало траектории для движения в обратном направлении. Как нужно скорректировать программу? Что нужно сделать, чтобы робот вернулся в исходную позицию?
4. Запустите симулятор и проверьте работу программы.

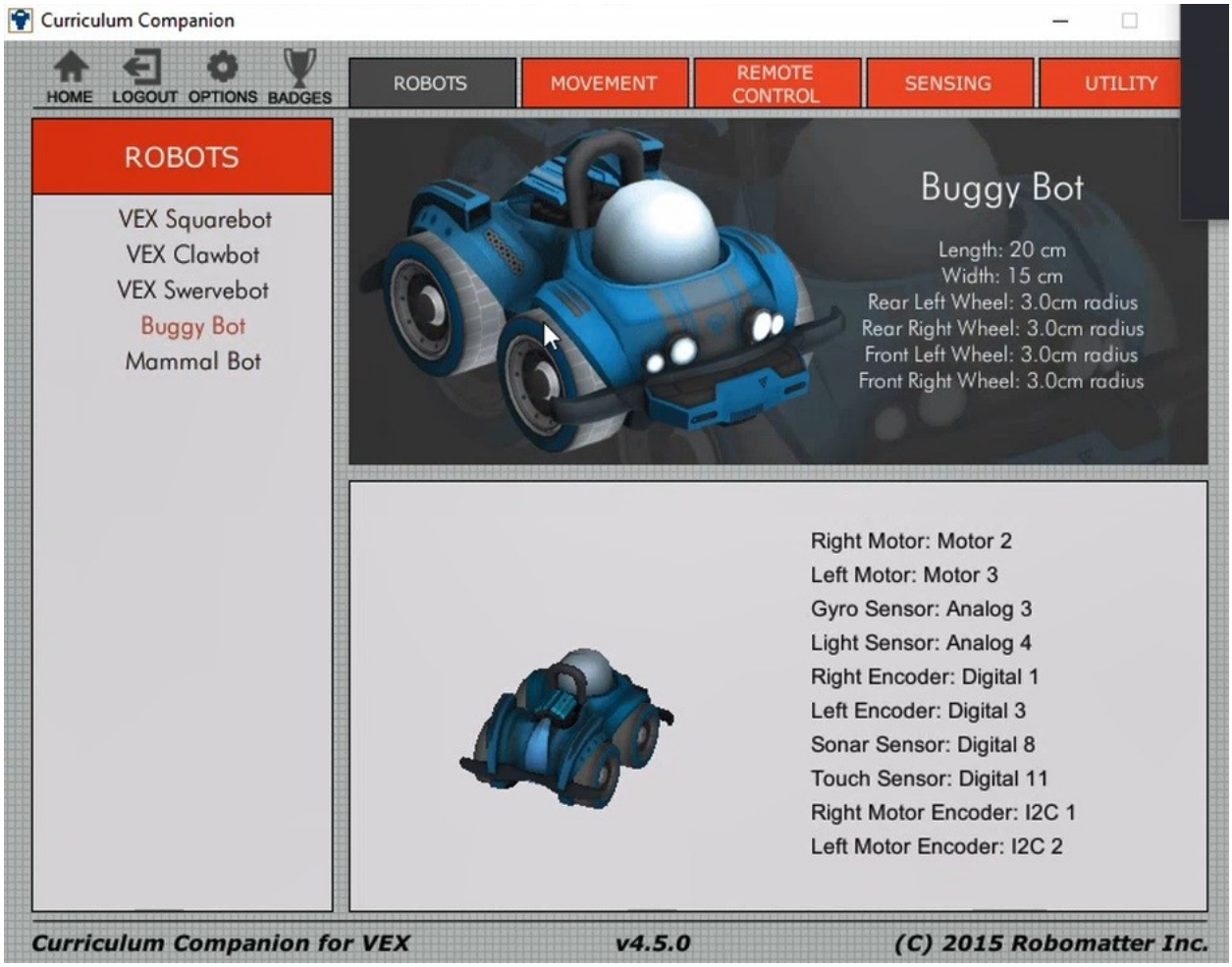

Рисунок 4: Модель робота Buggy Bot

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №8 «Использование джойстика в RobotC»

Цель

Изучение правил подключения и использования джойстика для управления роботом в RobotC.

Краткие теоретические сведения

Джойстик (рис.1) представляет собой самостоятельное устройство, с помощью которого можно дистанционно управлять роботом. Во всех предшествующих лабораторных работах предполагалось, что, даже после прошивки робота программой, соединение между роботом и компьютером остается по USB-кабелю. С одной стороны, только при таком режиме возможно использование отладчика и отображение содержимого всех переменных и датчиков, с другой – это создает определенные неудобства при работе т.к. USB-кабель имеет фиксированную длину и неудобен при маневрировании робота.

Рисунок 1: Джойстик VEX

Именно для обеспечения дистанционного управления роботом и применяется джойстик – предполагается, что код предварительно отлажен по проводному соединению (с применением отладчика), и уже после того, как были устранены все ошибки, можно переходить к управлению джойстиком.

Первое, что нужно сделать, при работе с джойстиком – выбрать режим управления «VEXnet or USB» (рис.2). Этот режим является комплексным:

предполагается, что управлять роботом можно и по USB-кабелю, и с помощью джойстика. В качестве альтернативы можно выбрать другой режим – «Competition (VEXnet)». В этом случае управление по USB-кабелю становится недоступным.

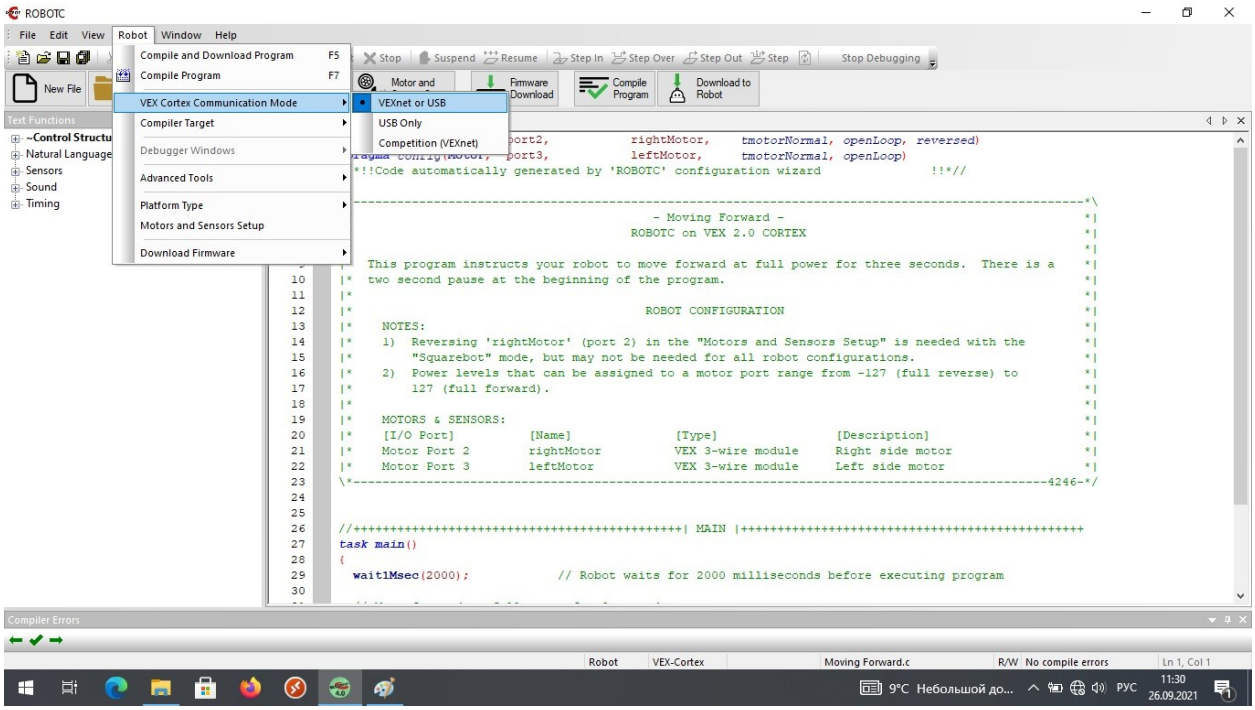

Рисунок 2: Выбор режима управления роботом

Работа с джойстиком практически ничем не отличается от работы с кнопкой (см. Лабораторная работа №3 «Использование датчиков в RobotC: датчик касания») – просто в джойстике мы имеем не одну, а достаточно большое количество кнопок, разделенных на группы, каждая из которых имеет свое имя (рис.3).

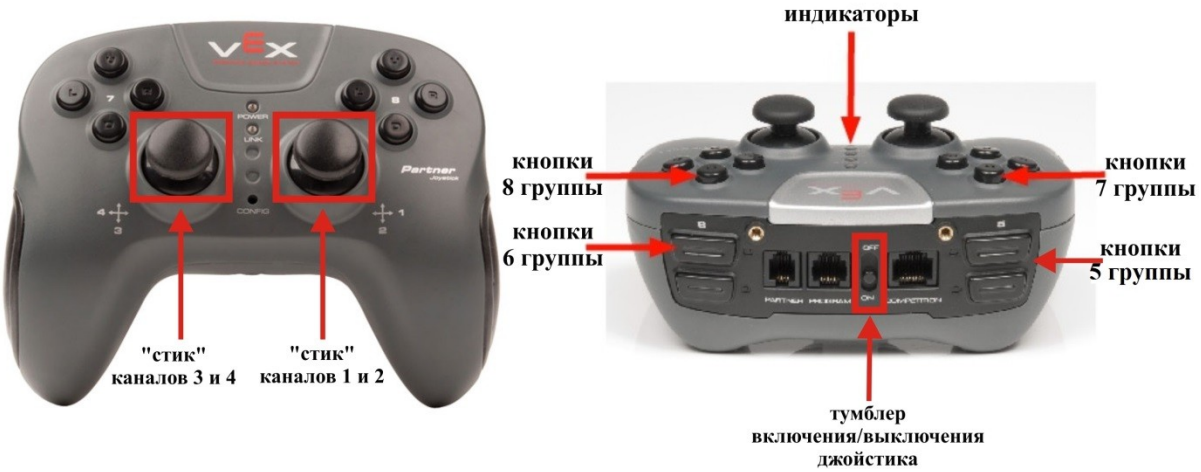

Рисунок 3: Элементы управления джойстика

Каждая кнопка внутри группы имеет свое имя, например, в группе 8 имеется 4 кнопки: «8U» (верхняя), «8D» (нижняя), «8L» (левая) и «8R» (правая). Соответственно, обращение к той или иной кнопки происходит по ее имени с приставкой «Btn». В отличие от обращения к обычной кнопке («Bumper Switch») используется не ключевое слово «SensorValue», а «vexRT»:

```
if (vexRT[Btn8U] == 1)
{
 //выполнение_первого_действия, если нажата 8U;
}
if (vexRT[Btn8U] == 0)
{
 выполнение_второго_действия;
}
```

Логику работы с кнопками джойстика рассмотрим на примере: если кнопка «8U» нажата (т.е. выполняется первое условие ветвления: vexRT[Btn8U] == 1), то происходит выполнение_первого_действия. Если кнопка «8U» не нажата (т.е. выполняется второе условие ветвления: vexRT[Btn8U] == 0), то происходит выполнение_второго_действия.

Таким образом, можно запрограммировать джойстик для дистанционного управления роботом. Например, назначить четырем кнопкам 8 группы совершение следующих действий: при нажатии на кнопку «8U» робот едет вперед, при нажатии на кнопку «8D» робот едет назад, при нажатии на кнопку

«8L» робот поворачивает влево и при нажатии на кнопку «8R» робот поворачивает вправо.

Пример программы, с помощью которой можно решить подобную задачу, приведен ниже:

```
#pragma config(Motor,  port1, MotorLeft, tmotorVex393_HBridge, openLoop,
driveLeft)
#pragma config(Motor,  port6, MotorRight, tmotorVex393_MC29, openLoop,
reversed, driveRight)
//*!!Code automatically generated by 'ROBOTC' configuration
wizard               !!*//
void moving(int LeftMotor, int RightMotor, int time)
{
setMotor(port1, LeftMotor);
setMotor(port6, RightMotor);
wait1Msec(time);
}
task main()
{
while(1)
{
if(vexRT[Btn8U] == 1) {
 moving(100,100,25);
 }
 if(vexRT[Btn8D] == 1)
 {
   moving(-100,-100,25);
 }
 if(vexRT[Btn8L] == 1)
 {
    moving(-100,100,25);
 }
 if(vexRT[Btn8R] == 1)
 {
 moving(100,-100,25);
 }
 else
 {
 moving(0,0,25);
 }
}
}
```

Разберем программу более подробно. Так же, как и в прошлых лабораторных работах, необходимо, помимо выбора режима управления роботом, настроить подключение используемых моторов (см. Лабораторную работу №2). В теле основной функции создан бесконечный цикл while(1) и уже внутри этого цикла происходит постоянный опрос всех кнопок джойстика. В том случае, если нажата кнопка «8U», мы попадаем в первую ветку if и оба мотора вращают оси колес робота с тем, чтобы он двигался вперед. Если будет нажата кнопка «8D», мы попадаем во вторую ветку if и оба мотора вращают оси колес робота с тем, чтобы он двигался назад. Если будет нажата кнопка «8L», мы попадаем в третью ветку if и оба мотора вращают оси колес робота с тем, чтобы он совершил поворот налево. Если будет нажата кнопка «8R», мы попадаем в четвертую ветку if и оба мотора вращают оси колес робота с тем, чтобы он совершил поворот направо. Если же не нажата ни одна кнопка – мы попадаем в else, внутри которого вызывается функция moving с нулевыми значениями аргументов скорости т.е. робот будет стоять на месте.

Задание

Скорректируйте программу так, чтобы по нажатию на кнопку «7U» робот Clawbot поднял «руку» и удерживал ее на заданной высоте до тех пор, пока не будет нажата кнопка «7D»: в этом случае робот должен опустить «руку» обратно.

Порядок выполнения работы

1. Запустите RobotC.
2. Выполните настройку подключения моторов в меню «Motors and Sensors setup» в соответствии с моделью робота Clawbot (рис.4).
3. Напишите программу, с помощью которой робот под управлением джойстика будет поднимать и опускать «руку» на заданную высоту: при нажатии на кнопку «7U» «рука» должна быть поднята, а при нажатии на кнопку «7D» «рука» должна быть опущена. В лабораторной работе №5 были рассказано о применении пропорционального регулятора и о параллельно выполняемых функциях. Для выполнения предложенного

задания нужно использовать навыки из этой лабораторной работы. Подумайте, что нужно сделать для того, чтобы при нажатии на определенную кнопку джойстика происходил запуск параллельно выполняемой функции, реализующей работу пропорционального регулятора и, соответственно, подъем «руки» робота? Вспомните про то, что «рука» падает под собственным весом в случае отсутствия управления – значит, программа для опускания «руки» может быть значительно проще, чем программа для ее подъема?

4. Запустите симулятор и проверьте работу программы.

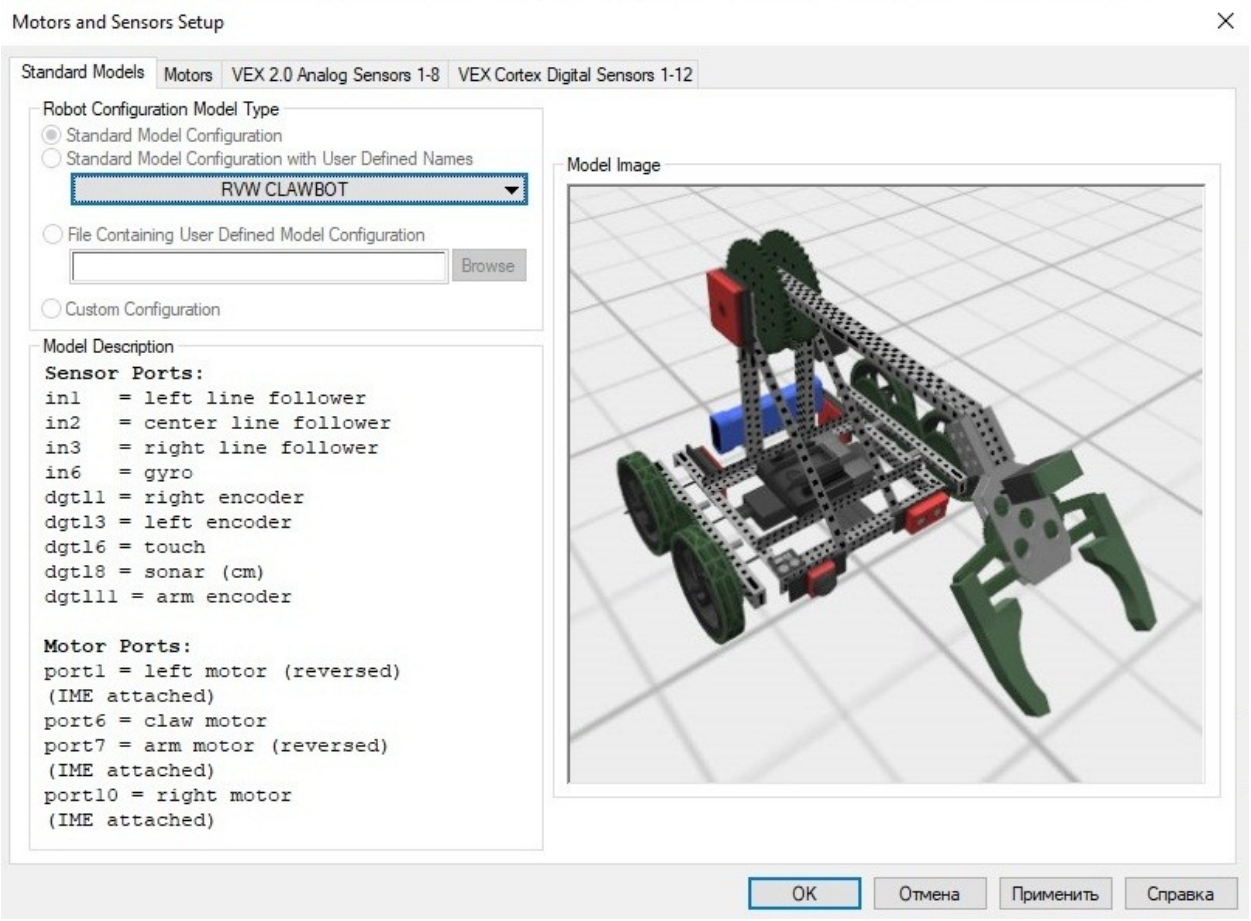

Рисунок 4: Модель робота Clawbot

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере RobotC.

2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить симулятор и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №9 «Основы программирования в Arduino»

Цель

Изучить основы программирования языка C/C++ в среде аппаратной платформы Arduino.

Краткие теоретические сведения

Arduino — инструмент для проектирования электронных устройств (электронный конструктор) более плотно взаимодействующих с окружающей физической средой, чем стандартные персональные компьютеры, которые фактически не выходят за рамки виртуальности.

Arduino применяется для создания электронных устройств с возможностью приема сигналов от различных цифровых и аналоговых датчиков, которые могут быть подключены к нему, и управления различными исполнительными устройствами. Проекты устройств, основанные на Arduino, могут работать самостоятельно или взаимодействовать с программным обеспечением на компьютере. Язык программирования Arduino является реализацией Wiring, схожей платформы для «physical computing», основанной на мультимедийной среде программирования Processing.

Устройство Arduino Mega 2560

Рисунок 1: Распиновка Arduino Mega 2560

Цифровой сигнал передаётся в виде единиц и нулей, он дискретный, аналоговый сигнал непрерывно изменяется во времени. Но аналоговые сигналы чувствительны к воздействию шумов и помех, подвержены искажению и затуханию при передаче или записи. Поэтому какие либо данные передают с помощью цифрового сигнала, а аналоговый сигнал, как правило, используют для получения некоторой физической величины. Например, колебания звука. На рисунке 2 наглядно представлена разница между аналоговым и цифровым сигналом.

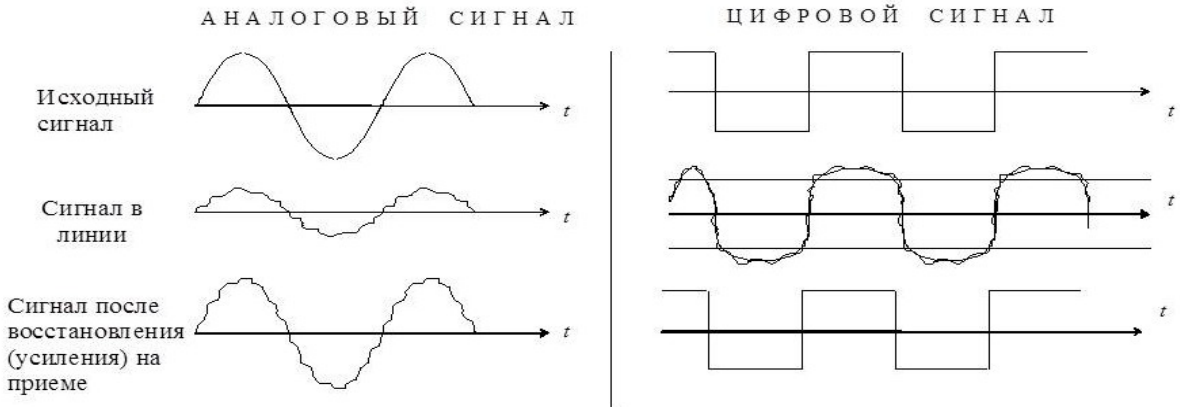

Рисунок 2: Разница между аналоговым и цифровым сигналом

Язык программирования в среде Arduino IDE

Язык программирования Arduino похож на C/C++ с некоторым особенностями, облегчающими новичкам написание первой программы.

Программы, написанные программистом называются скетчи и сохраняются в файлах с расширением .ino. Эти файлы перед компиляцией обрабатываются препроцессором Arduino.

Arduino обладает долговременной энергонезависимой памятью. Соответственно любой скетч, выгруженный в Arduino, при следующей инициализации останется и начнёт действовать.

Начало работы

Для начала работы необходимо иметь последнюю версию среды программирования ArduinoIDE, которую можно скачать на официальном сайте. После подключения Arduino к USB-выходу (COM-порту) на персональном компьютере и запуска среды программирования, откроется пустой скетч (рис 3).

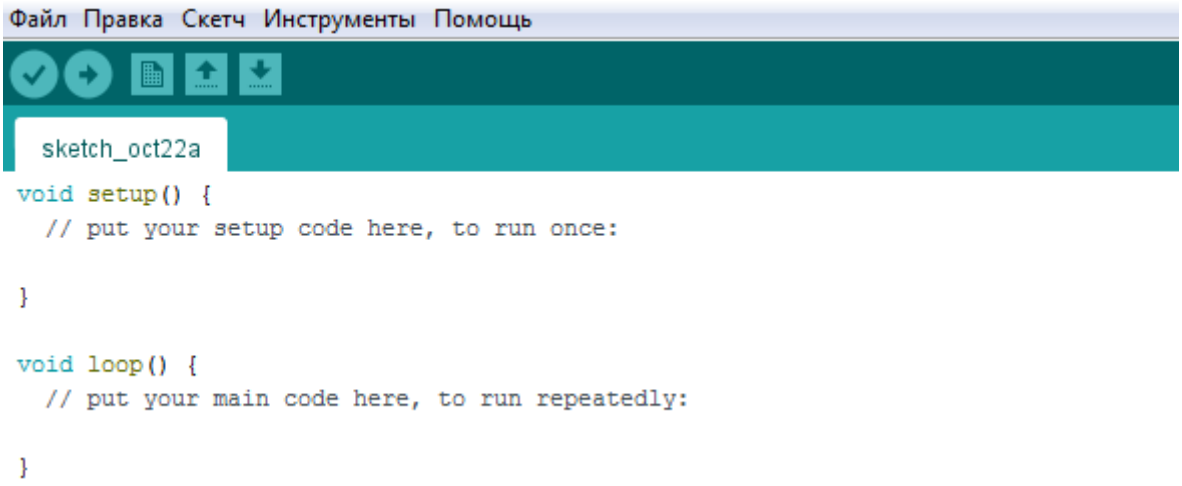

Рисунок 3: Пустой скетч

Далее следует инициализация рабочей платы. Следует открыть инструменты, выбрав пункт Плата, выбрать плату, которая подключена к компьютеру. (рис 4).

Затем выставить верхний процессор из списка предложенных (рис 5).

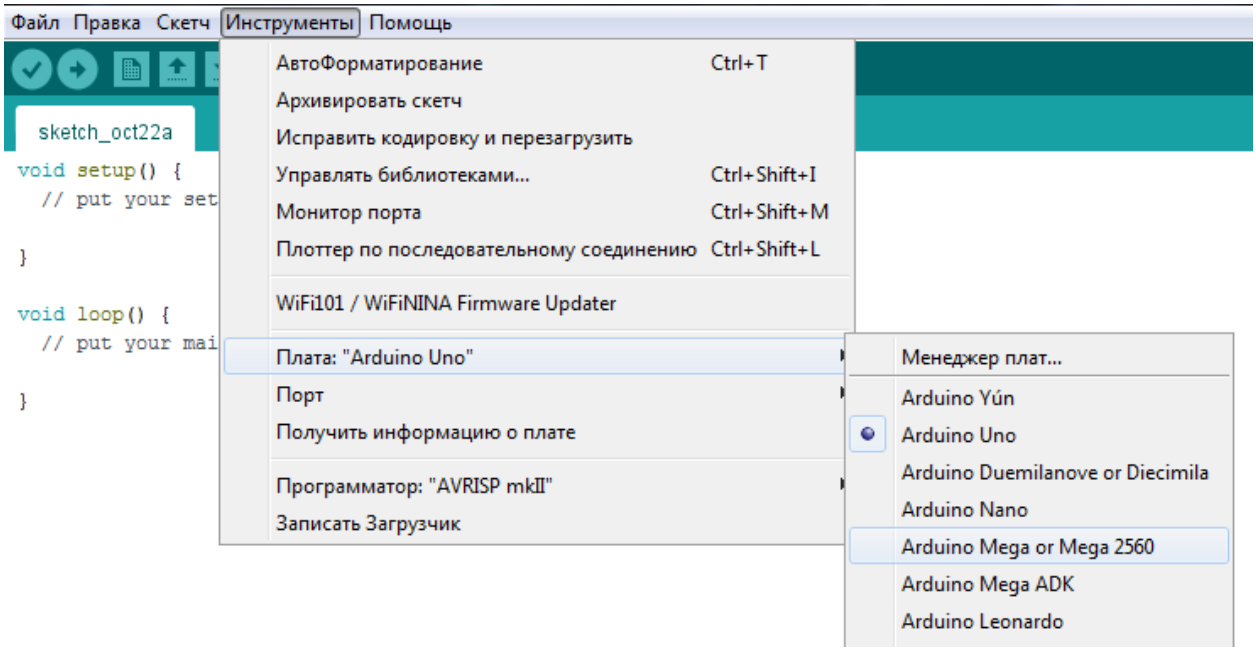

Рисунок 4: Инициализация рабочей платы

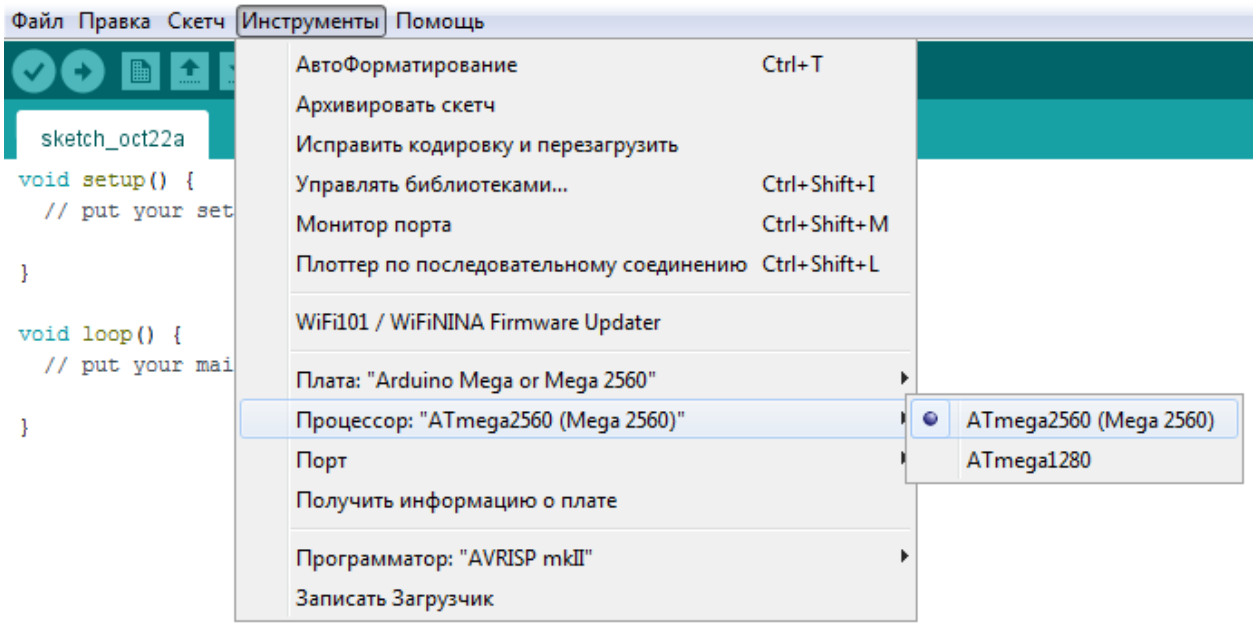

Рисунок 5: Инициализация процессора

Правильность подключения платы можно определить по горящему зелёному светодиоду-индикатору питания. Убедившись в правильном включении платы нужно выбрать порт с идентифицированным названием платы.

Первая программа

Простейшая программа состоит из двух функций

- *setup()*: функция вызывается один раз при старте микроконтроллера и служит панелью управления для периферии Arduino;

- *loop()*: функция вызывается после *setup()* и выполняется в бесконечном цикле и является основной исполняемой программой.

Одна из простейших схем на Arduino - включение встроенного в плату светодиода (рис 1).

Ниже приведён полный текст простейшей программы (скетча) мигания светодиода, на этапе производства платы подключённого к 13 выводу, с периодом 2 секунды.

```
void setup(){
pinMode(13, OUTPUT); // Назначение 13 вывода Arduino выходом
}
void loop(){
digitalWrite(13, HIGH); // Включение 13 вывода
delay (1000);                  // Задержка на 1000 мс
    digitalWrite(13, LOW); // Выключение 13 вывода
delay (1000);                 // Задержка на 1000 мс (1 секунда)
}
```

Скетч готов и остаётся лишь загрузить программу в саму Arduino нажатием на горизонтальную *стрелку*, перед загрузкой код скомпилируется и проверится на наличие ошибок (рис 6).

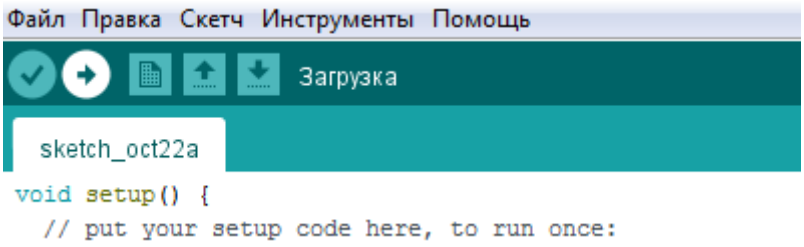

Рисунок 6: Выгрузка программы в плату

Задание

Напишите программу, по которой мигание светодиода станет пульсирующим: два быстрых мигания, большая задержка в выключенном состоянии и снова два быстрых мигания.

Порядок выполнения работы

1. Запустите ArduinoIDE.
2. Выполните настройку подключения платы в инструменты -> Плата -> Arduino Mega or Mega 2560 (рис 4). Далее, убедитесь, что в пункте Процессор выделена строка ATmega2560 (Mega2560), в пункте Порт выделен порт с названием платы.
3. Напишите программу по заданию.
4. Загрузите скетч в плату (рис 6).

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Загрузить программу в плату и продемонстрировать её работу.

5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу программы на плате – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №10 «Управление яркостью светодиода с помощью джойстика»

Цель

Научиться использовать джойстик в программах, изменять яркость светодиода с его помощью.

Краткие теоретические сведения

Устройство джойстика

Использование джойстика – это один из способов обмена информацией между человеком и устройством (компьютер, микроконтроллер). Чаще всего их используют для управления механизмами или роботами. Существует большое количество видов джойстиков по количеству степеней свободы, частоте считывания информации и используемой технологии. В данной лабораторной работе рассматривается наиболее популярный вариант - аналоговый джойстик.

Аналоговый джойстик выглядит как ручка, которая закрепляется на шарнире с двумя потенциометрами (переменными резисторами), определяющими оси X и Y. Наклон ручки вращает специальный подвижный контакт, из-за чего изменяется внутреннее сопротивление, и как результат выходное напряжение. Сам джойстик оснащен пружиной, благодаря которой плавно возвращается в первоначальное центральное состояние после отпускания его с какой-либо позиции.

Используемые в джойстике потенциометры изменяют входное напряжение, выдавая на порт Arduino другое напряжение, что является аналоговым сигналом в диапазоне значений от 0 до 1023. На рисунке 1 представлено деление этих диапазонов для управления “назад” и “вперёд” по обеим осям.

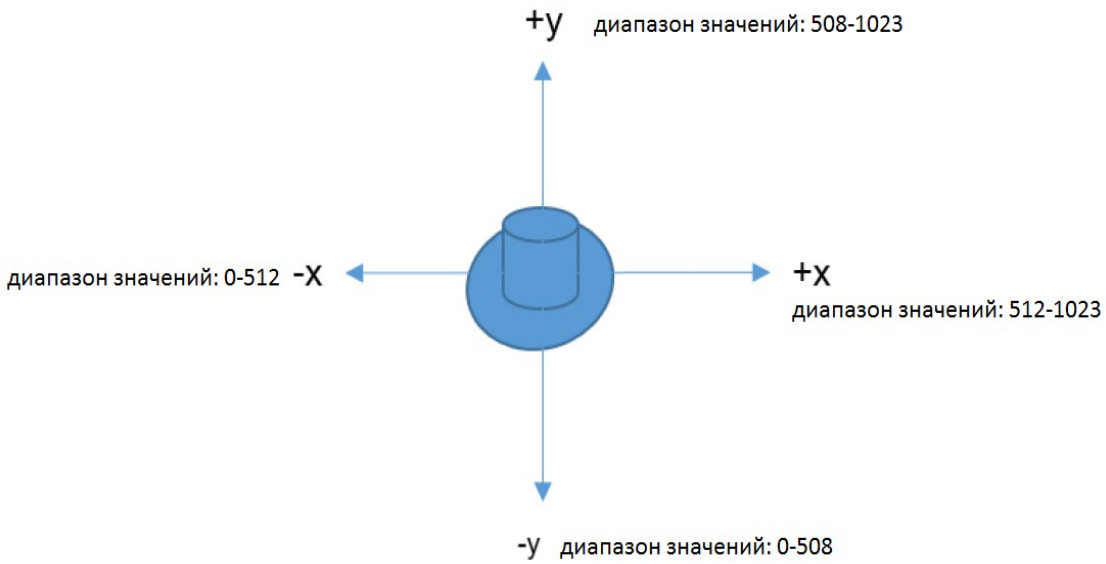

Рисунок 1: Диапазон значений джойстика

Плата расширения

Одним из ключевых преимуществ платформы Arduino является наличие огромного ассортимента плат расширения (Arduino Shields). Плата расширения Ардуино – это законченное устройство, предназначенное для выполнения определенных функций. На плате расширения установлены все необходимые электронные компоненты, а взаимодействие с микроконтроллером и другими элементами основной платы происходят через стандартные пины Arduino. Чаще всего питание на плату расширения подается с основной платы arduino, однако во многих других моделях есть возможность подключения питания с внешних источников.

Для данной лабораторной работы также имеется своя плата расширения, позволяющая удобно соединить все необходимые электронные компоненты для работы манипулятора с микроконтроллером.

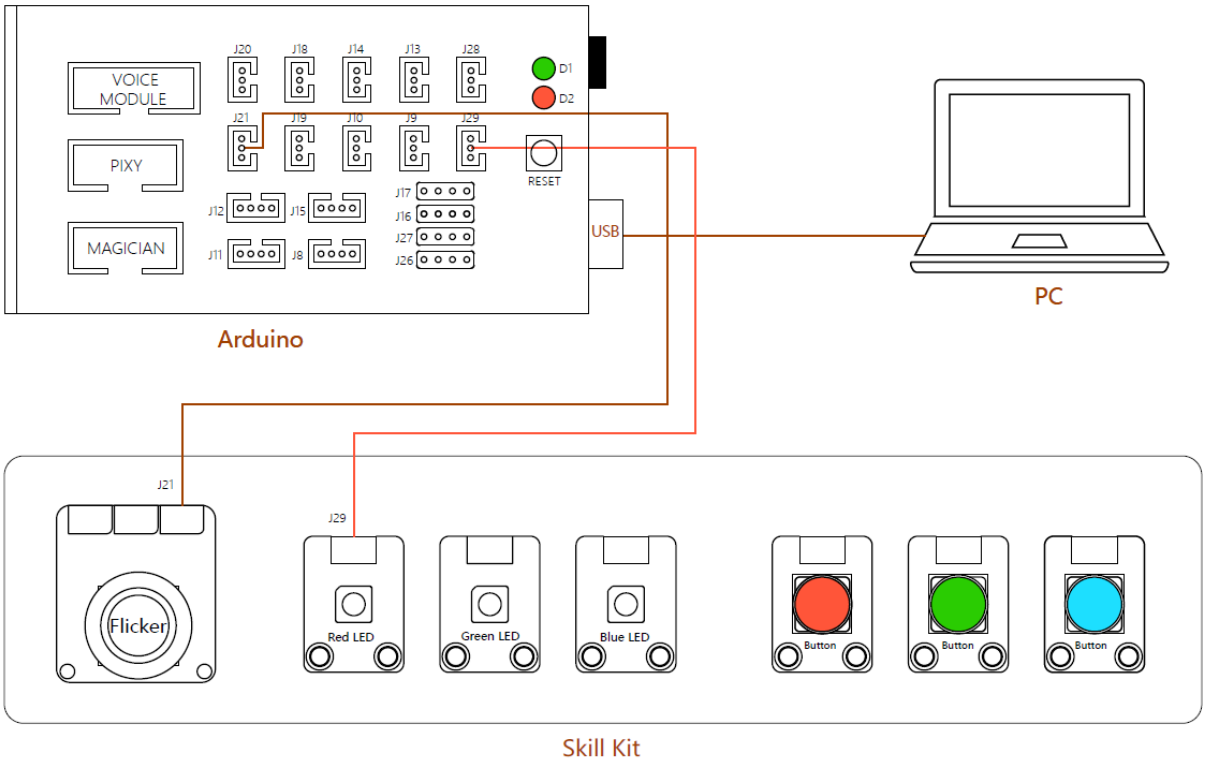

Рисунок 2: Схема подключения джойстика и светодиода

Для начала работы с джойстиком и внешним светодиодом необходимо подключить их к микроконтроллеру. Плата расширения имеет уже подготовленные для работы с ним провода и выведенные необходимые пины, поэтому достаточно подключить нужные компоненты, как представлено на рисунке 2.

Составление программного кода

Как уже известно из описания джойстика, его взаимодействие с микроконтроллером происходит с помощью подачи на пин Arduino аналогового сигнала. Работа с этим сигналом в коде программы осуществляется за счёт нескольких функций:

analogRead() : считать значение с указанного в скобках аналогового входа. Напряжение, поданное на аналоговый вход, обычно от 0 до 5 вольт, будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт.

analogWrite() : Выдает аналоговую величину (ШИМ волну) на порт вход/выхода в диапазоне от 0 до 255.

Очень часто в робототехнике возникает необходимость плавно управлять каким-то процессом, например, яркостью светодиода, мощностью обогревателя или скоростью вращения мотора. Вполне очевидно, что управление напрямую связано с изменением напряжения на потребителе: и светодиод будет по-другому светить, и мотор крутиться с другой скоростью. Но проблема в том, что управлять напряжением может только ЦАП – цифро-аналоговый преобразователь, а в микроконтроллере Arduino встроенного ЦАПа нет, только цифровой сигнал, т.е. либо вкл, либо выкл. Как раз ШИМ и является решением данной проблемы.

ШИМ - Широко-импульсная модуляция (PWM - Pulse-width modulation). После вызова analogWrite() на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова analogWrite (или вызова digitalWrite или digitalRead на том же порту вход/выхода) с частотой сигнала приблизительно 490 Hz.

Таким образом можно регулировать яркость светодиода, быстро включая и выключая его. Несмотря на то, что светодиод практически не имеет задержки включения/выключения, при более высокой частоте ШИМ сигнала переключение светодиода будут незаметны человеческому глазу, создавая эффект меньшей яркости. Рисунок 3 наглядно показывает процесс регулировки яркости, чем ниже, тем ярче.

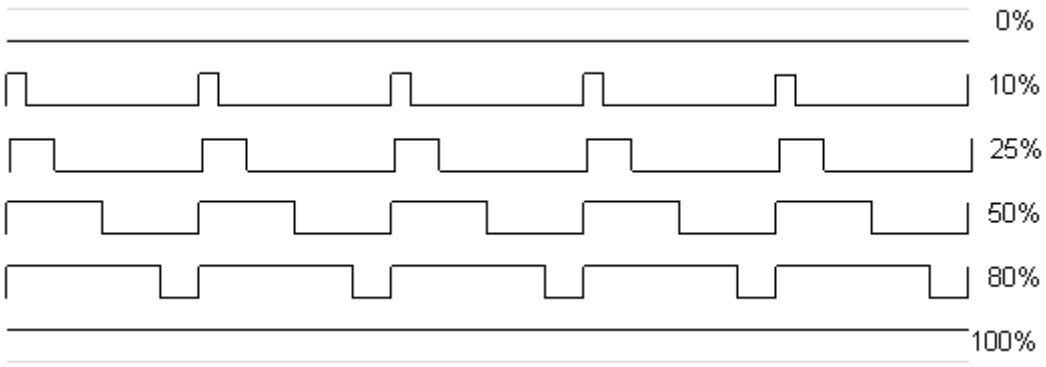

Рисунок 3: Пример рабочего цикла ШИМ

Пример программы:

```
#define led 9                          // Наименование пинов 9 и A7
#define JoyStick_x A7
int i;                                 // Объявление переменных для хранения
int x;                                 // значения яркости (i) светодиода и текущего
                                       // положения ручки джойстика (x)
void setup(){
pinMode(led, OUTPUT);                  // Назначение 9 пина выходом
pinMode(JoyStick_x, INPUT);            // Назначение 7 пина входом
}
void loop(){
x = analogRead(JoyStick_X);            // Считывание положения ручки

if( x > 520 ){
i++;                                   // увеличить значение i на 1
} else if ( x < 500 ) {
i --;                                  // уменьшить значение i на 1
    }


if( i >= 255 ){                        // предотвращение
i = 255;                               // переменной i выхода за предел
}else if ( i < 0 ){                    //  допустимых значений
i = 0;
}
analogWrite(led, i);                   // вывод на светодиод
}
```

Задание

Напишите программу, по которой яркость светодиода с помощью оси X джойстика будет изменять свою яркость, а по оси Y изменять интервал мигания.

Для работы с осью Y необходимо подключить вторым проводом джойстик в слот J10, для чтения положения второй оси использовать пин A4.

Порядок выполнения работы

1. Запустите ArduinoIDE.
2. Выполните настройку подключения платы в инструменты -> Плата -> Arduino Mega or Mega 2560. Далее, убедитесь, что в пункте Процессор выделена строка ATmega2560 (Mega2560), в пункте Порт выделен порт с названием платы.
3. Напишите программу по заданию.
4. Загрузите скетч в плату.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Загрузить программу в плату и продемонстрировать её работу.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу программы на плате – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №11 «Включение светодиода нажатием кнопки»

Цель

Научиться использовать кнопку для взаимодействия с контроллером.

Краткие теоретические сведения

Кнопка, подключение к плате и проблемы при её использовании

Кнопка – один из самых популярных и простых видов датчиков. В основе работы любой кнопки лежит механический способ смыкания-размыкания контактов. Нажимая на любую, даже самую простую тактовую кнопку, формируется определенное давление на внутренние механизмы (пластины или пружины), в результате чего происходит сближение или расхождение металлических пластин, замыкание и размыкание контакта.

Для чтения текущего состояния кнопки (нажата или не нажата) используется функция digitalRead(). Сама же кнопка должна быть подключена к питанию, земле с подтягивающим резистором и пину, объявленному как вход - pinMode(PIN, INPUT).

Однако даже такой элементарный механизм имеет свой недостаток - “дребезг.” Дело состоит в том, что микроскопические неровности на поверхности пластин, многократные неконтролируемые замыкания и размыкания цепи, вызванные подпрыгиванием контактов при соударениях, не позволяют пластинам мгновенно соединиться. Из-за этого в короткий промежуток времени на границе пластинок меняется и сопротивление, и взаимная емкость, из-за чего возникают масса разнообразных изменений уровня тока и напряжения (переходный процесс) (Рис 1).

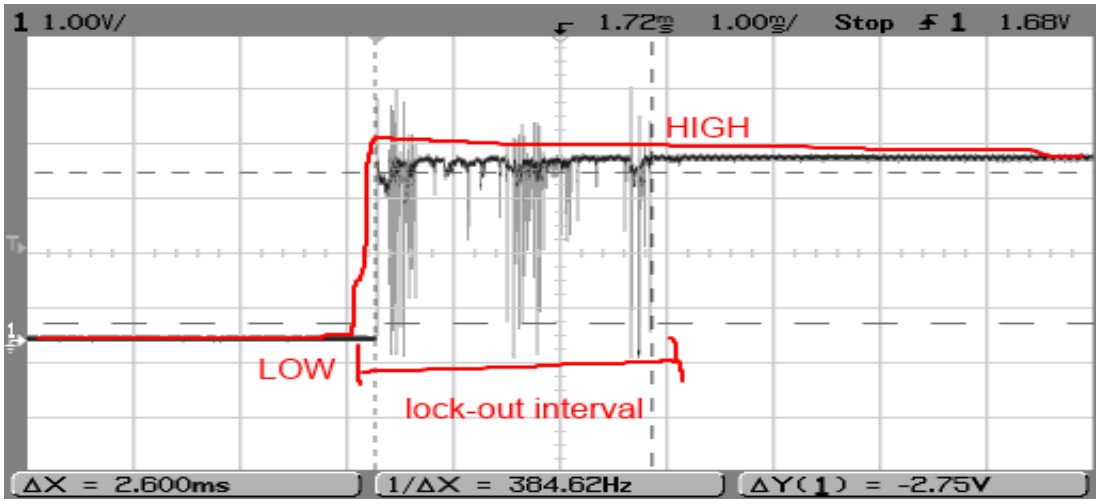

Рисунок 1: Дребезг сигнала при замыкании контакта

Переходные процессы протекают очень быстро и исчезают за доли миллисекунд. Поэтому человек редко их замечает, например, когда включается свет в комнате. Лампа накаливания не может менять свою яркость с такой скоростью, и тем более не может реагировать на эти изменения человеческий мозг. Но, обрабатывая сигнал от кнопки на таком быстром устройстве как Arduino, вполне можно столкнуться с такими переходными эффектами и нужно их учитывать при программировании.

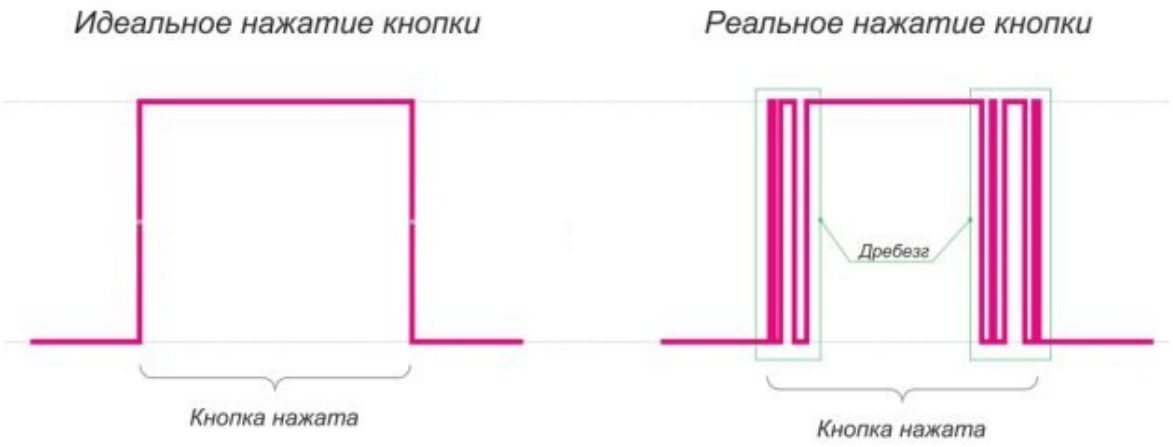

Рисунок 2: Нажатие кнопки на практике

Методы борьбы с дребезгом

Программный способ заключается в добавлении задержки. Процесс выполнения программы останавливается на время от 10 мс до 50 мс функцией delay(). В скобках функции указывается время в миллисекундах.

С помощью библиотеки Bounce. Проблема с дребезгом настолько популярна, что созданы специальные библиотеки, благодаря которым не нужно организовывать ожидание и паузы вручную – это все делается внутри специального класса.

Пример использования:

```
#include <Bounce2.h>;
#define PIN_BUTTON 2
#define PIN_LED 13
Bounce debouncer = Bounce();                    // Создание объекта

void setup() {
 pinMode(PIN_BUTTON, INPUT_PULLUP);  // Установка типа пина

 debouncer.attach(PIN_BUTTON);            // Даем библиотеке знать,
                                                        // к какому пину
                                                        // подключена кнопка

 debouncer.interval(5);                           // Интервал, в течение
                                                        // которого значения с пина
                                                        // не будет приходить

 pinMode(PIN_LED, OUTPUT);
}

void loop() {
  debouncer.update();                          // Даем объекту библиотеке знать,
                                                        // что мы вошли в цикл loop()

  int value = debouncer.read();               // Получаем значение кнопки

  if ( value == LOW ) {
   digitalWrite(PIN_LED, HIGH );
  }
  else {
   digitalWrite(PIN_LED, LOW );
  }
}
```

Аппаратный способ подавления.

Подавление дребезга кнопки с помощью задержек в скетче – способ очень распространенный и не требующий изменения самой схемы. Но далеко не всегда его можно использовать – ведь 10 миллисекунд – это целая вечность для многих процессов в электронном мире. Также программный способ невозможно применять при использовании прерываний – дребезг приведет к многократному вызову функций и повлиять на этот процесс в скетче мы не сможем.

Более правильный способ борьбы с дребезгом – использование аппаратного решения, сглаживающего импульсы, посылаемые с кнопки.

Аппаратный способ устранения дребезга основан на использовании сглаживающих фильтров. Сглаживающий фильтр, как следует из названия, занимается сглаживанием всплесков сигналов за счет добавления в схему элементов, имеющих своеобразную “инерцию” по отношению к таким электрическим параметрам как ток или напряжение. Самым распространенным примером таких “инерционных” электронных компонентов является конденсатор. Он поглощает всплески, медленно накапливая и отдавая энергию.

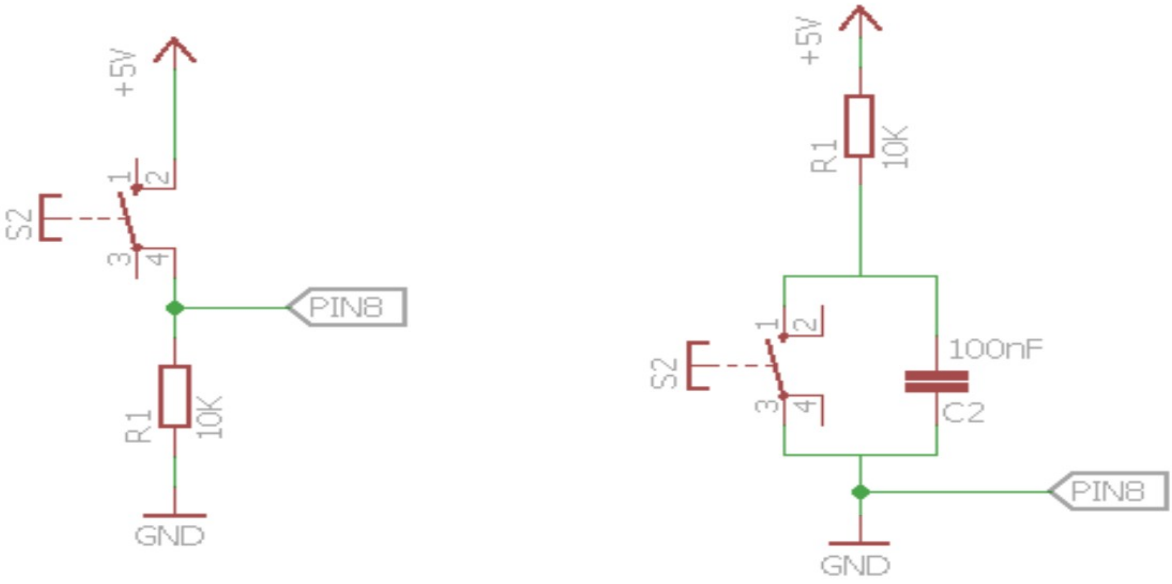

Рисунок 3: Схема подключения конденсатора к кнопке

За счет инерции устройство сглаживает сигнал с большим количеством пиков и впадин, создавая неидеальную, но вполне гладкую кривую, у которой легче определить уровень срабатывания.

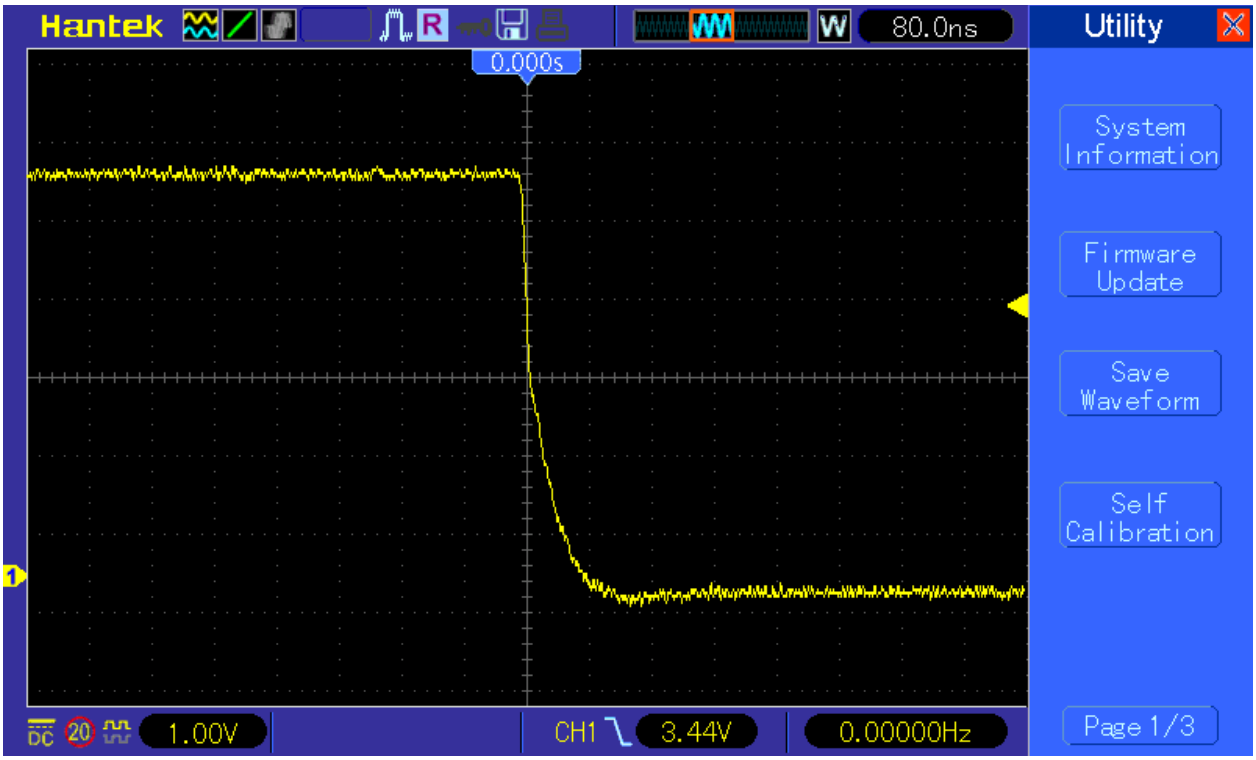

Рисунок 4: Форма сигнала после отжатия кнопки

Триггера Шмитта.

Сделать квадратную форму сигнала с помощью простой RC цепочки невозможно (цепочки с конденсатором). Для “огранения” сглаженных форм

используется специальный компонент, который называется триггер Шмидта. Его особенностью является срабатывание при достижении определенного уровня сигнала. На выходе Триггера Шмитта получится или высокий, или низкий уровень сигнала, никаких промежуточных значений. Выход триггера инвертированный: при спаде входного сигнала он выдает на выходе включение и наоборот. На рисунках 5, 6 представлена схема подключения и результат работы с триггером шмидта.

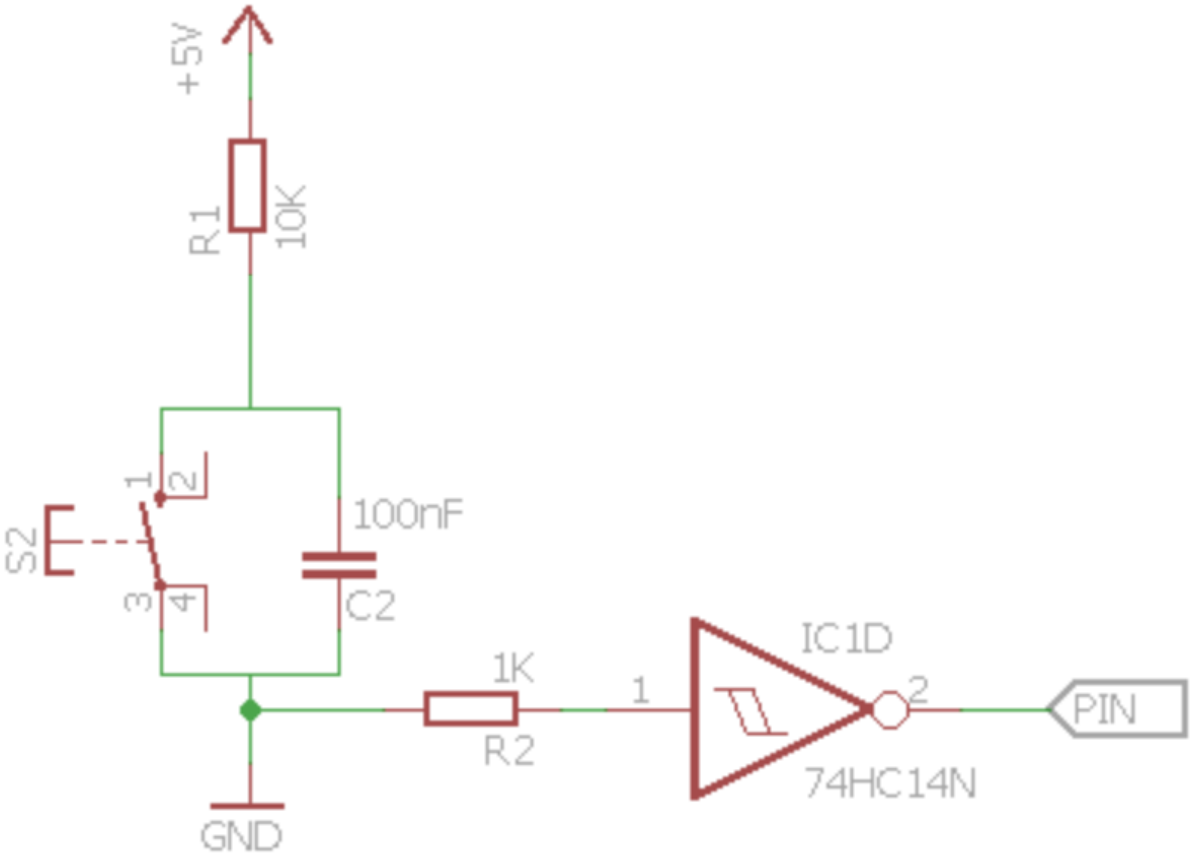

Рисунок 5: Схема подключения Триггера Шмитта

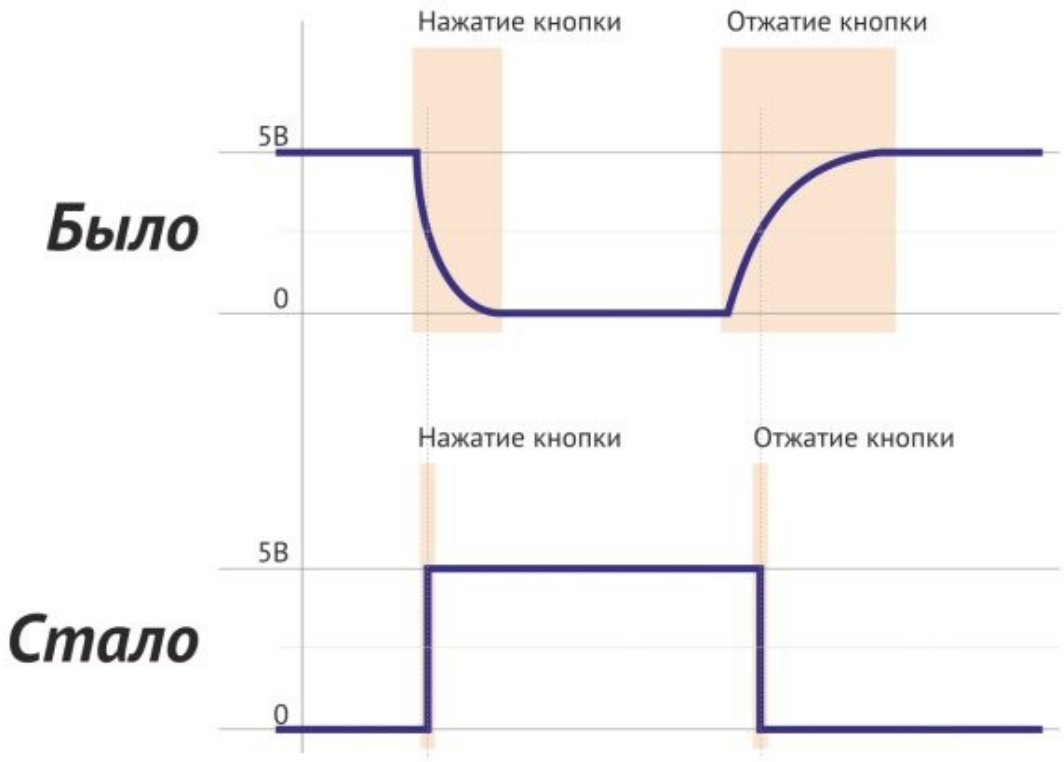

Рисунок 6: Результат работы схемы с Триггером Шмитта

Задание

Подключите три кнопки и три светодиода к микроконтроллеру как изображено на рисунке 7 и напишите программу, по которой включение светодиода будет происходить с помощью кнопки. Для каждого светодиода должна быть отдельная кнопка. Устраните дребезг с помощью программного способа.

Переменные и пины:

```
#define Red_LED 9
#define Green_LED A1
#define Blue_LED A3

#define Red_button A0
#define Green_button A2
#define Blue_button A4
```

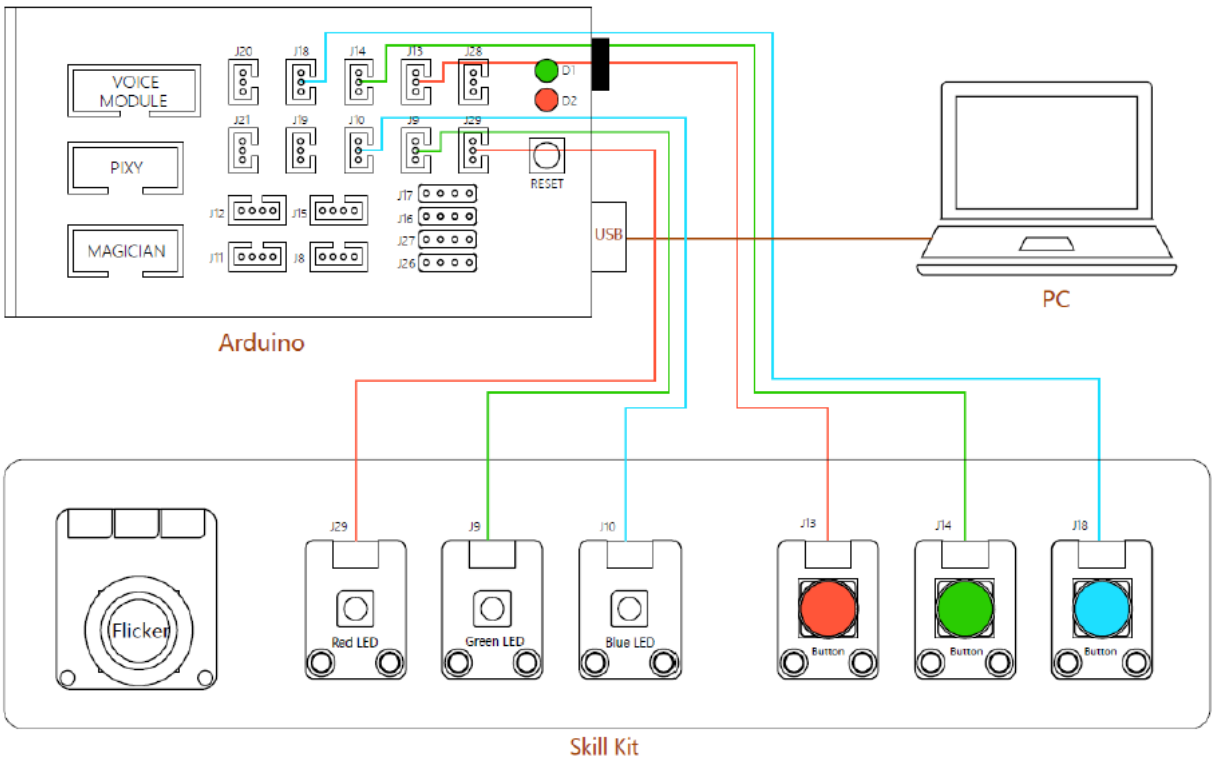

Рисунок 7: Схема подключения кнопок и светодиодов

Порядок выполнения работы

1. Запустите ArduinoIDE.
2. Выполните настройку подключения платы в инструменты -> Плата -> Arduino Mega or Mega 2560. Далее, убедитесь, что в пункте Процессор выделена строка ATmega2560 (Mega2560), в пункте Порт выделен порт с названием платы.
3. Напишите программу по заданию.
4. Загрузите скетч в плату.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
2. Озвучить задание лабораторной работы.

3. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу программы на плате – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа № 12 «Модуль голосового управления»

Цель

Научиться использовать в программах модуль голосового управления.

Краткие теоретические сведения

Распознавание речи и его применение

В настоящее время все больший интерес приобретают системы распознавания речи, что обусловлено широким распространением мобильных устройств. Применение голосового управления к данным устройствам, ввиду ограниченности интерфейсов ручного ввода привело бы к расширению их возможностей и упрощению в использовании. Существующие технологии распознавания, как правило, не имеют широкого распространения ввиду ограничения их по одному или нескольким параметрам. Например, основная доля систем, позволяющих распознать большинство пользовательских запросов требуют подключения к сети Интернет (к удаленным серверам, на которых происходит обработка запросов), поскольку недостаточно их вычислительной мощности и ограниченности памяти, выделенной под словарь.

К одним из первых программ для коммерческого использования можно отнести Voice Navigator, Dragon NaturallySpeaking и другие. Начало использования таких программ датируется началом девяностых годов. В основном, их используют люди с ограниченными возможностями, набор текста большого объема для которых является проблематичным. Преобразуя голосовое сообщение диктора в текст, данные программы позволяют не использовать руки при печатании. Качество перевода у данных программ довольно низкое, однако со временем оно существенно улучшается.

Одним из этапом внедрения технологий распознавания отмечается применение интерфейсов безмолвного доступа (SSI). Данные системы основаны на обработке речи на начальной стадии артикулирования. Однако

можно выделить значительный недостаток систем распознавания речи, это: слишком высокая чувствительность к воздействию шумов, т.е необходимо довольно разборчиво и четко произносить голосовые команды во время обращении к системе по распознаванию. Важно отметить, что SSI основан на подходе заключающимся в использовании сенсоров не подверженных влиянию шумов в качестве дополнения к обработанным акустическим сигналам.

Можно выделить пять основных способов распознавания речи:

1. Распознавание отдельных команд – в данном способе требуется раздельно произносить слово\словосочетание, а затем проводится его распознавание. Качество распознавания данного способа ограничена размером имеющегося словаря.

2. Распознавание по грамматике – в данном способе проводится распознавание уже по фразам, которое соответствует определенному набору правил. Грамматика здесь задается путем использования стандартных XML8 языков, а обмен данными между системой распознавания и приложением осуществляется по протоколу MRCP.

3. Поиск ключевых слов в потоке слитной речи – в данном способе распознавание проводится по отдельным участкам речи. Здесь речь вполне может быть, как соответствующая набору неких правил, так и соответствующей определённым правилам. В данном методе нет необходимости переводить весь текст - в ней находятся лишь те участки, которые содержат заданные слова или словосочетания.

4. Распознавание слитной речи на большом словаре – в данном методе, сказанная фраза, дословно преобразуется в текст. При этом достоверность данного метода достаточно высока.

5. Распознавание речи с помощью нейронных систем – является довольно сложным методом, однако на основе нейронных сетей появляется возможность создания обучаемых и самообучающиеся систем, что является важной предпосылкой для их применения в системах распознавания (и синтеза) речи.

Используемый в лабораторной работе голосовой модуль по большей части работает с четвёртым способом распознавания речи. Прослушивание команды активируется словом “Hicell” и сопровождается включением светодиода на модуле. Далее, слово, полученное с микрофона, разбивается на буквы для распознавания. Например, “позвонить” (Рис 1).

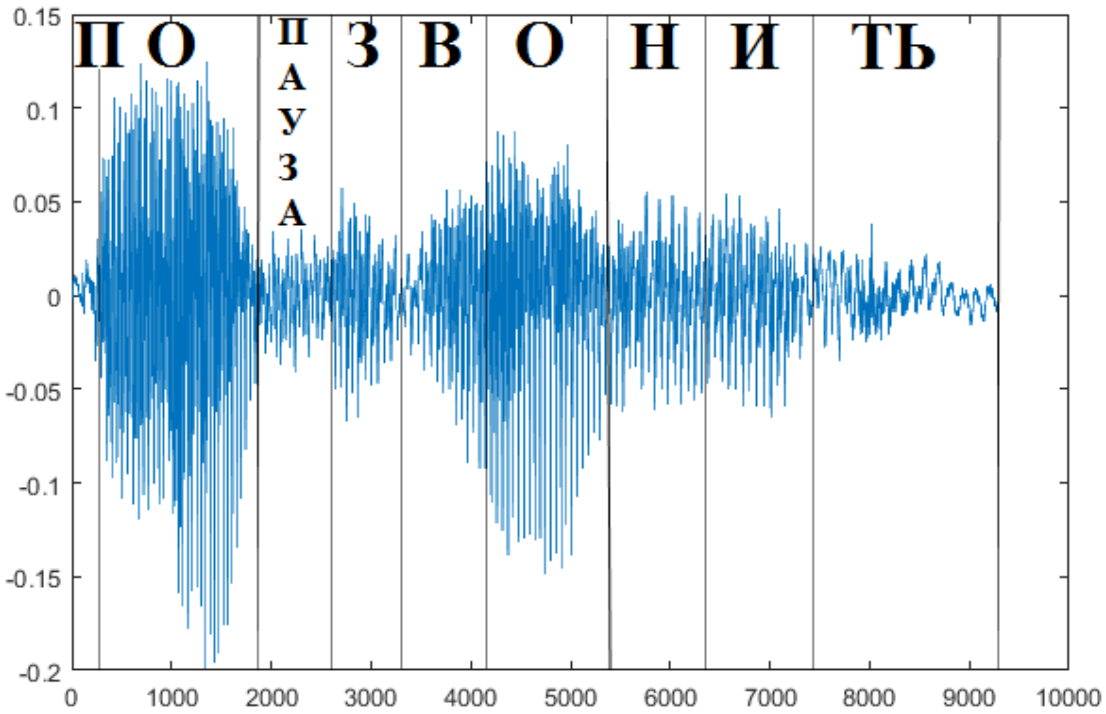

Рисунок 1: График отсегментированного слова «позвонить»

Полученный порядок букв сравнивается с записанными в памяти модуля командами и на выходе возвращается значение, соответствующее схожему слову.

В данном голосовом модуле прописаны алгоритмы определения двадцати двух команд на английском языке:

- "Turn on the light",

- "Turn off the light",
- "Play music",
- "Pause",
- "Next",
- "Previous",
- "Up",
- "Down",
- "Turn on the TV",
- "Turn off the TV",
- "Increase temperature",
- "Decrease temperature",
- "What's the time",
- "Open the door",
- "Close the door",
- "Left",
- "Right",
- "Stop",
- "Start",
- "Mode 1",
- "Mode 2",
- "Go",

Пример программного кода

Так как голосовой модуль возвращает числовые значения, их можно использовать в качестве триггера для активации каких либо алгоритмов. Например, для включения и отключения светодиодов.

Также из-за количества доступных команд при написании кода будет неудобно на каждую создавать отдельную переменную и использовать сочетание операторов if и else. Гораздо удобнее воспользоваться массивом и оператором switch().

char voicebuffer[a] = {} : Массив voicebuffer длинной a. Char - символьный тип данных, предназначенный для хранения одного символа (управляющего или печатного) в определённой кодировке. Может являться как однобайтовым (для стандартной таблицы символов), так и многобайтовым (к примеру, для Юникода). Основным применением является обращение к отдельным знакам строки. Если не указывать количество переменных, входящих в массив, то его длина будет зависеть от количества данных в фигурных скобках, перечисленных через запятую.

Switch() – оператор, который сравнивает последовательно значение переменной, находящейся в скобках, со всеми вариантами значений, находящимися после каждого ключевого слова case.

```
#include <SoftwareSerial.h>                // библиотека для работы с
                                           // голосовым модулем
#define SOFTSERIAL_RX_PIN A11
#define SOFTSERIAL_TX_PIN 14

#define Red_LED 9
#define Green_LED A1
#define Blue_LED A3

const char *voiceBuffer[] =                //массив с командами
{
"Turn on the light",                       //возвращает 1
"Turn off the light",                      //возвращает 2
"Play music",                              //возвращает 3
"Pause",                                   //возвращает 4
"Next",                                    //возвращает 5
"Previous",                                //возвращает 6
"Up",                                      //возвращает 7
"Down",                                    //возвращает 8
"Turn on the TV",                          //возвращает 9
"Turn off the TV",                         //возвращает 10
"Increase temperature",                    //возвращает 11
"Decrease temperature",                    //возвращает 12
"What's the time",                         //возвращает 13
"Open the door",                           //возвращает 14
"Close the door",                          //возвращает 15
"Left",                                    //возвращает 16
```

```
"Right",                                  //возвращает 17
"Stop",                                   //возвращает 18
"Start",                                  //возвращает 19
"Mode 1",                                 //возвращает 20
"Mode 2",                                 //возвращает 21
"Go",                                    //возвращает 22
}

void setup()
{
   Serial.begin(9600);                  // Подключение монитора порта,
   softSerial.begin(9600);             // вывода данных с голосового модуля
   softSerial.listen();                // на этот порт
}



void loop()
{
char cmd;
if(softSerial.available())            // Если была подана активирующая модуль
                                      //  команда “Hicell”, то выполнить:
 {
   cmd = softSerial.read();          // запись полученного номера в переменную
                                     // для хранения
   switch(cmd)
 {
     case 1:
       digitalWrite(Red_LED,HIGH);
       digitalWrite(Green_LED,HIGH);
       digitalWrite(Blue_LED,HIGH);
       Serial.println(voiceBuffer[cmd - 1]);
       break;
     case 2:
      digitalWrite(Red_LED,LOW);
      digitalWrite(Green_LED,LOW);
      digitalWrite(Blue_LED,LOW);
      Serial.println(voiceBuffer[cmd - 1]);
      break;
 }
}
}
```

Задание

Повторите код из примера и усовершенствуйте его, добавив возможность включать и отключать каждый светодиод независимо от других. Схема подключения - рисунок 2.

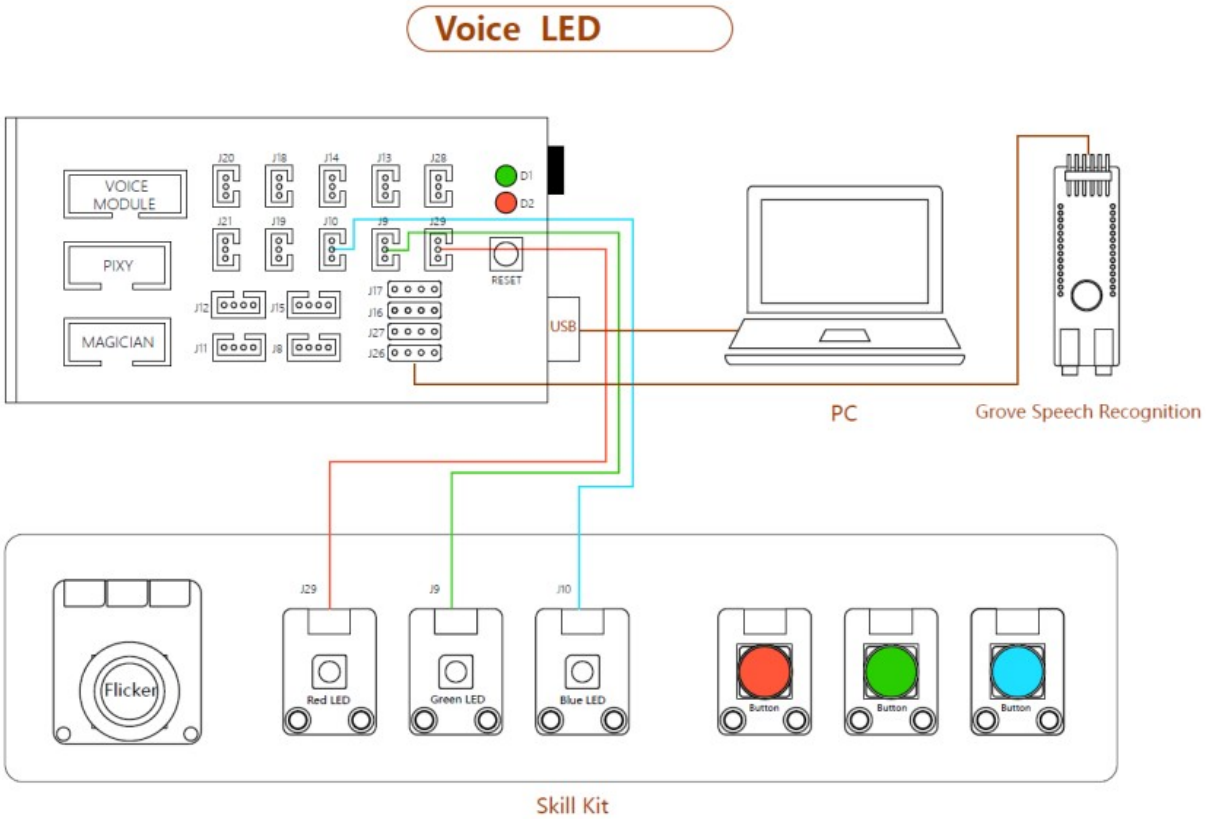

Рисунок 2: Схема подключения

Порядок выполнения работы

1. Запустите ArduinoIDE.
2. Выполните настройку подключения платы в инструменты -> Плата -> Arduino Mega or Mega 2560. Далее, убедитесь, что в пункте Процессор выделена строка ATmega2560 (Mega2560), в пункте Порт выделен порт с названием платы.
3. Напишите программу по заданию.
4. Загрузите скетч в плату.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Загрузить программу в плату и продемонстрировать её работу.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу программы на плате – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №13 «Управление манипулятором»

Цель

Научиться управлять манипулятором, автоматизировать его работу.

Краткие теоретические сведения

Роботизированный манипулятор DOBOT Magician является универсальной платформой для углубленного изучения промышленной робототехники и разработки собственных производственных линий с полной автоматизацией процессов, обеспечивая развитие востребованных инженерных компетенций у учеников, а комплект сменных инструментов позволяет изучать не только проектирование и программирование, но и современные технологии аддитивного и лазерного производства.

Состав и устройство манипулятора

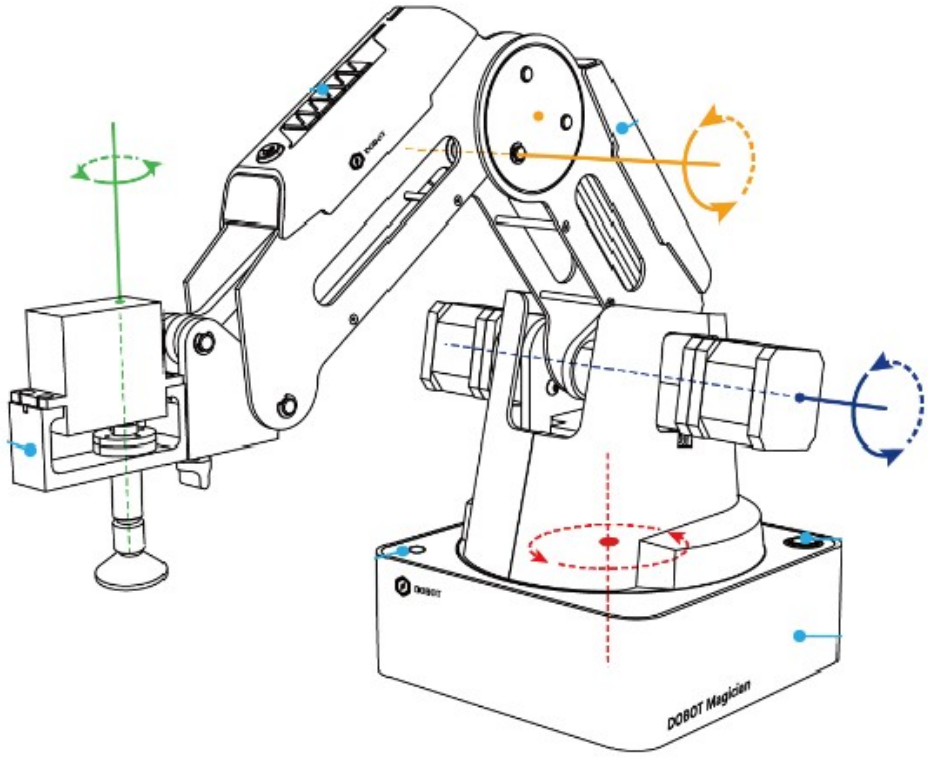

Рисунок 1: Состав и устройство манипулятора

Опорная конструкция (или основание) снизу снабжена противоскользящими накладками, чтобы Dobot надежно фиксировался на поверхности и не скользил во время работы. При этом основание достаточно массивное, чтобы Dobot обладал устойчивостью, даже если необходимо добраться до далеко отстоящих объектов. На верхней части основания располагаются кнопка включения и индикатор состояния. Красный цвет индикатора означает, что манипулятор не включился или рабочий инструмент находится вне рабочей зоны, зеленый — манипулятор готов к работе. На боковой поверхности размещены разъемы для подключения питания (Power)(5), кабеля для программирования (USB)(4), пульта дистанционного управления (Communication Interface)(3), сторонних устройств и датчиков (Peripheral Interface)(6), кнопки Key(2) (выполнить загруженную программу) и Reset (перезагрузить)(1) (Рис 2).

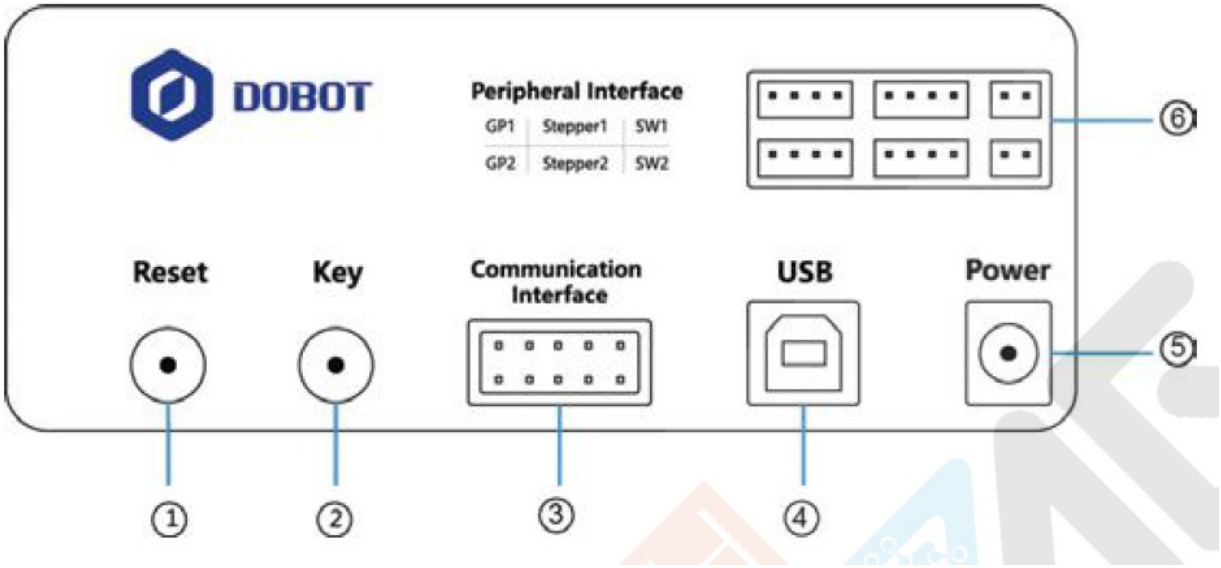

Рисунок 2

Соединение элементов манипулятора между собой

Основание с колонной соединяется при помощи вала, двигателя и подшипника. Плечо к колонне крепится также посредством вала, подшипника и двигателя. Стрела и плечо соединены рычагом первого рода (рис.1). Для того чтобы рабочий инструмент всегда находился под одним и тем же углом к плоскости горизонта, используются рычаги, которые соединены между собой в виде параллелограмма. Один параллелограмм крепится к площадке рабочего инструмента и находится внутри стрелы, второй размещен внутри плеча. Второй параллелограмм (внутри плеча) заставляет первый (внутри стрелы) двигаться, то есть при любом движении стрелы и площадки рабочего инструмента стороны рычагов всегда параллельны друг другу. Рабочий инструмент вставляется в специальное отверстие в площадке и фиксируется винтом-бабочкой. Сервопривод крепится к рабочему инструменту при помощи болтов, к манипулятору — проводом GP3.

Рабочий инструмент манипулятора Dobot Magician имеет несколько модификаций: вакуумный захват (присасывает предметы с относительно гладкой поверхностью), механический захват (захватывает предметы с рельефной поверхностью), захват для пишущего инструмента (позволяет писать

и рисовать на ровной поверхности), экструдер (печатает 3D-элементы), лазерный гравер (гравирует или отрезает лазером).

Движение элементов манипулятора

Внутри колонны и на ней крепится 3 шаговых двигателя, один из которых (внутри колонны) соединен с основанием и отвечает за вращение в горизонтальной плоскости всего манипулятора относительно основания (ось вращения перпендикулярна основанию). Еще два двигателя расположены снаружи колонны: один отвечает за движение плеча (ось вращения параллельна основанию), другой — за движение стрелы относительно плеча (ось вращения параллельна основанию). Рабочий инструмент вращается сервоприводом.

Для того чтобы провода не путались и не попадали под рабочий инструмент, разъемы для датчиков и исполнительных устройств, в том числе для сервопривода рабочего инструмента, находятся на стреле.

Важно! На стреле расположена кнопка разблокировки шаговых двигателей. Когда манипулятор находится во включенном состоянии, попытка смены положения рабочего инструмента вручную с приложением усилий может привести к поломке двигателей. Чтобы этого избежать, необходимо нажимать кнопку.

Три шаговых двигателя и их предназначение

Один двигатель дает возможность рабочему инструменту попасть на любую точку окружности, два двигателя — на любую точку сферы, три двигателя — на любую точку внутри шара. Таким образом, количество шаговых двигателей определяет рабочую зону манипулятора. Не обязательно всегда использовать сразу три двигателя. Например, если исходная точка (откуда взять объект) и конечная точка (куда его переложить) находятся на одной плоскости, будет достаточно одного двигателя.

Для того чтобы включить манипулятор, к нему необходимо присоединить блок питания. После этого подключить его к сети 220V.

Для манипуляций с объектами используется воздушная помпа, которая откачивает воздух (в вакуумном захвате создает вакуум между присосом и объектом; в механическом захвате двигает поршень, который управляет элементами захвата). Воздушная помпа подключается к основанию манипулятора проводом питания SW1 и проводом управления GP1 (соответствующие разъемы SW1 и GP1 на основании робота-манипулятора). Она соединяется непосредственно с рабочим инструментом при помощи воздушной трубки (крепится к штуцеру на вакуумном захвате).

Для дистанционного управления манипулятором применяется пульт дистанционного управления. Пульт работает от аккумуляторов, которые заряжаются через micro USB.

Для связи пульта с манипулятором к основанию Dobot подключается USB-контроллер (USB Host), который по протоколу Bluetooth принимает сигналы от пульта. USB Host имеет два индикатора: синий и зеленый. Если горит только синий индикатор, значит, пульт готовится к работе. Если включены сразу оба (синий + зеленый), Dobot готов принимать сигналы от пульта. Для соединения пульта с USB Host используется USB Bluetooth-модуль.

Рабочая зона манипулятора ограничена и не позволяет захватывать предметы, находящиеся на удалении. Для перемещения объектов применяется конвейерная лента. Основным элементом конвейерной ленты является непосредственно лента и мотор, управляющий ее движением. Подключение мотора к манипулятору осуществляется при помощи провода, который вставляется в разъем STEPPER1 на обратной стороне основания манипулятора.

Для перемещения манипулятора в горизонтальной плоскости используется рельс. Для соединения устройств применяется специальная подставка, которая крепится болтами к основанию манипулятора с одной стороны и к рельсу — с другой. Движение манипулятора по рельсу обеспечивает шаговый двигатель,

который соединяется с манипулятором шиной с проводами (разъем STEPPER1). Чтобы привести двигатель в действие, рельс подключается к сети 220V.

Программное обеспечение DobotStudio и подключение к компьютеру

Управление манипулятором может осуществляться при помощи компьютера.

Необходимо иметь программное обеспечение с сайта Dobot.cc (ПО) DobotStudio на компьютере.

Язык ПО по умолчанию английский. Для того чтобы сменить его, вызовите меню языковых параметров. Оно находится в верхней части окна (буквы EN). Выберите русский язык и перезапустите ПО.

Подключение манипулятора к компьютеру.

1. Включите ПК и запустите ПО DobotStudio.
2. Подключите адаптер питания к основанию манипулятора, включите питание кнопкой, расположенной на основании. Дождитесь, когда манипулятор издаст короткий звуковой сигнал.
3. Выполните подключение манипулятора к компьютеру при помощи USB-кабеля.
4. В окне ПО нажатие на кнопку «Подключить», которая расположена в верхнем левом углу, должно изменить вид кнопки на «Отключить», в таком случае подключение выполнено успешно. (Рис 3)

Рисунок 3: Интерфейс подключения манипулятора

В верхнем правом углу окна ПО расположены три кнопки: «Аварийная остановка», «Домой» и «Настройки».

- При нажатии на кнопку «Аварийная остановка» производится немедленная остановка манипулятора. (рисунок 4).
- При нажатии на кнопку «Домой» манипулятор автоматически возвращается в стартовое положение. Данное положение можно дополнить или изменить в меню «Настройки».
- Меню «Настройки» позволяет выполнять управление множеством параметров манипулятора.

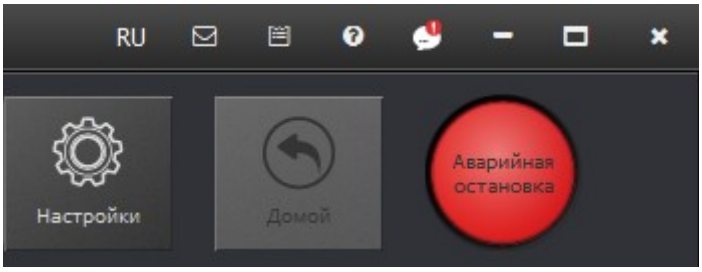

Рисунок 4: Кнопка аварийной остановки

В левой части экрана размещены кнопки основных функциональных модулей манипулятора и кнопка для самостоятельного добавления модулей.

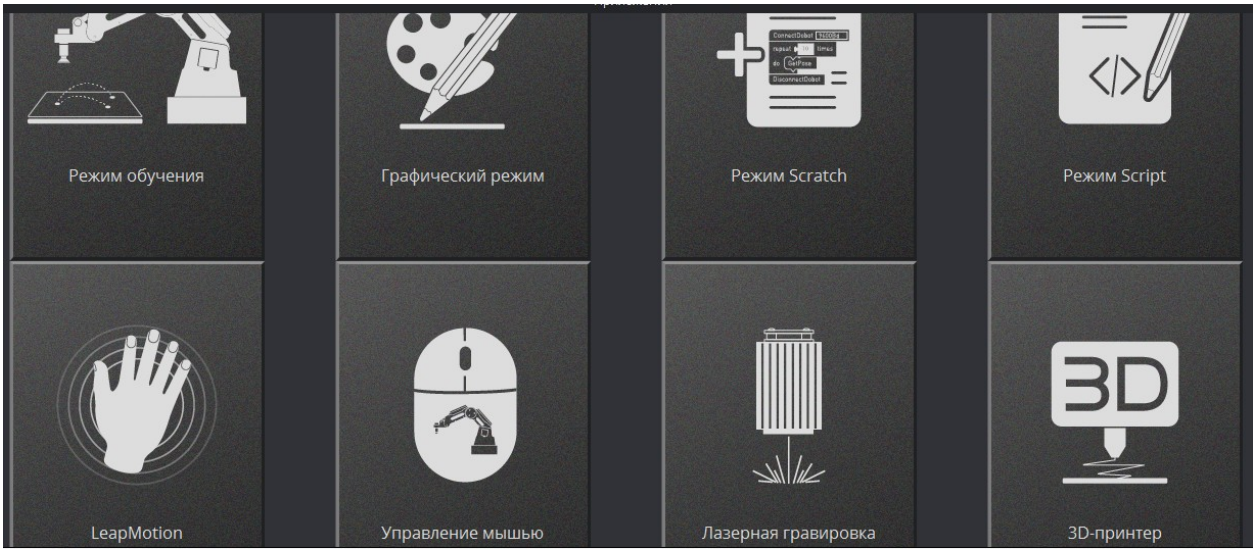

Рисунок 5: Основные функциональные модули

- Режим обучения — это режим составления простых линейных программ для перемещения рабочего инструмента манипулятора между точками. В этом режиме Dobot повторяет действия за оператором или следует его указаниям.
- Графический режим предназначен для рисования загруженных в векторном формате изображений.
- Режим Blockly — это режим программирования манипулятора в пиктографической среде программирования Dobot Blockly.

- Режим Script — в объектно-ориентированном языке программирования Python.
- LeapMotion — это технология управления манипулятором при помощи специального сенсора, считывающего движения рук.
- Режим управления мышью — это режим управления манипулятором в сферической системе координат при помощи использования компьютерной мыши.
- Лазерная гравировка — это режим для управления таким рабочим инструментом, как лазерный гравер. Позволяет вырезать (выжигать) загруженные в растровом формате рисунки при помощи лазера.
- 3D-принтер — это режим, позволяющий манипулятору работать в качестве 3D-принтера. Требует установки специального ПО Repetier Host.

Управление Dobot Magician при помощи панели управления

Для того чтобы манипулятором можно было управлять при помощи компьютера, недостаточно просто объединить манипулятор с компьютером посредством ПО. Необходим некий посредник, принимающий и передающий команды оператора и позволяющий ему извне осуществлять управление режимом работы манипулятора. В ПО DobotStudio в качестве такого посредника выступает панель управления.

Панель управления расположена в правой части экрана и может находиться в скрытом или открытом состоянии. Переход между ними осуществляется при помощи стрелки, направленной влево.

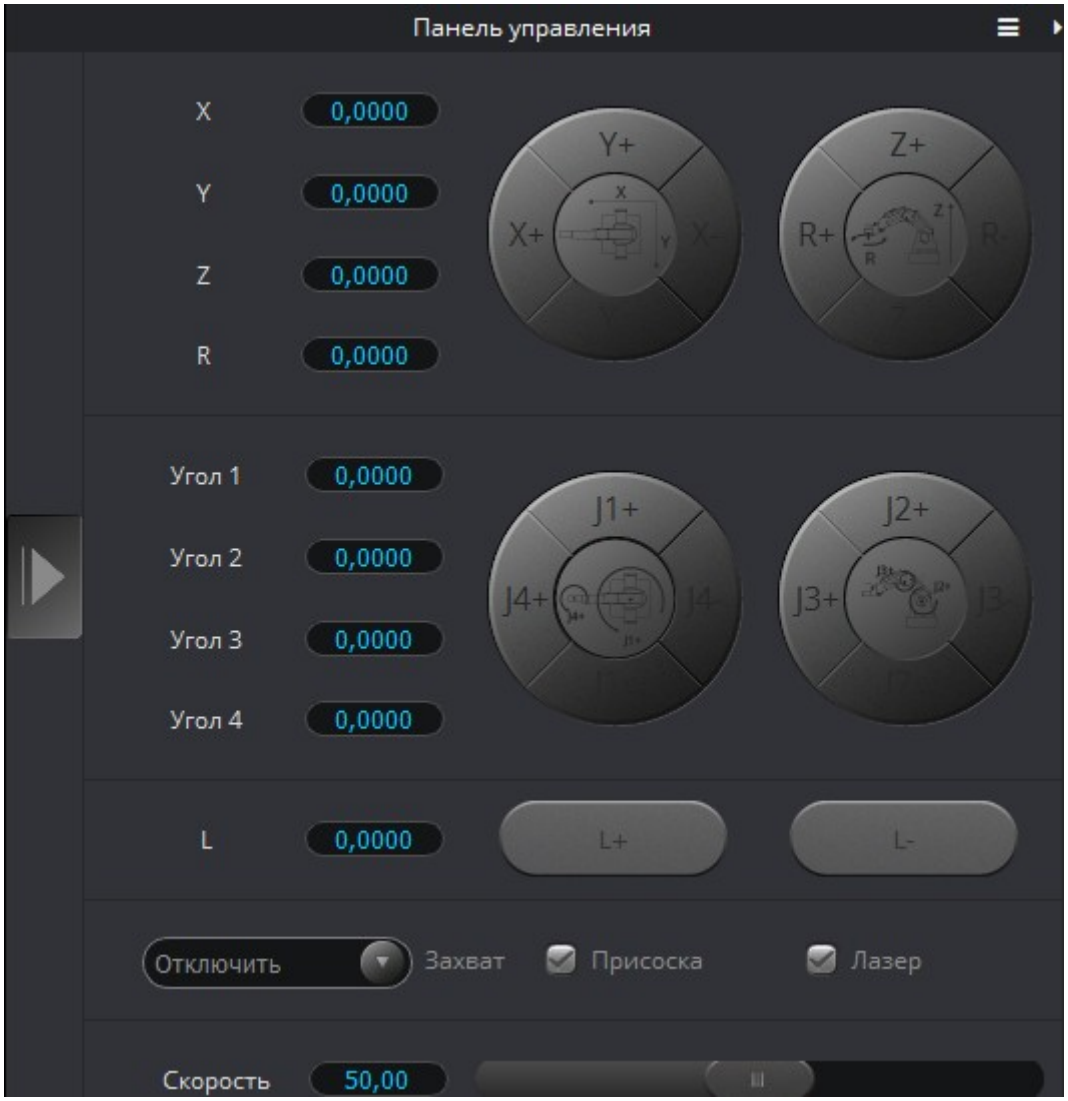

Рисунок 6: Панель управления

Внешний вид панели управления очень напоминает вид пульта дистанционного управления. Но в отличие от пульта на панели располагаются специальные окошки, отображающие числовые характеристики совершаемых манипулятором движений. Таким образом, у оператора появляется возможность отслеживать текущие координаты рабочего инструмента в декартовой системе координат (параметры X, Y, Z, R) и в сферической системе координат (Углы 1–4).

Каждому режиму управления соответствует свой набор кнопок, что позволяет избежать путаницы при переключении режимов.

Пример программы

Задачей примера является перемещение кубика из пункта A в пункт B.

Для выполнения задачи необходимо освоить две функции, связанные с манипулятором и позволяющие управлять его местоположением в пространстве.

- Dobot_SetPTPCmdEx(JUMP_XYZ, X, Y, Z, R) - оператор перемещения захвата манипулятора в точку с координатами X, Y, Z и R.
- Dobot_SetEndEffectorSuctionCupEx() - оператор для захвата предмета с помощью помпы. Указание в скобках true или false - включает откачку воздуха для присасывания предмета (захвата предмета) или отключает для освобождения предмета.

Далее, нужно опытным путём определить координаты пункта A, где находится кубик, и координаты пункта B, куда нужно переместить кубик. Справится с этим поможет панель управления манипулятором. (Рис 1.7)

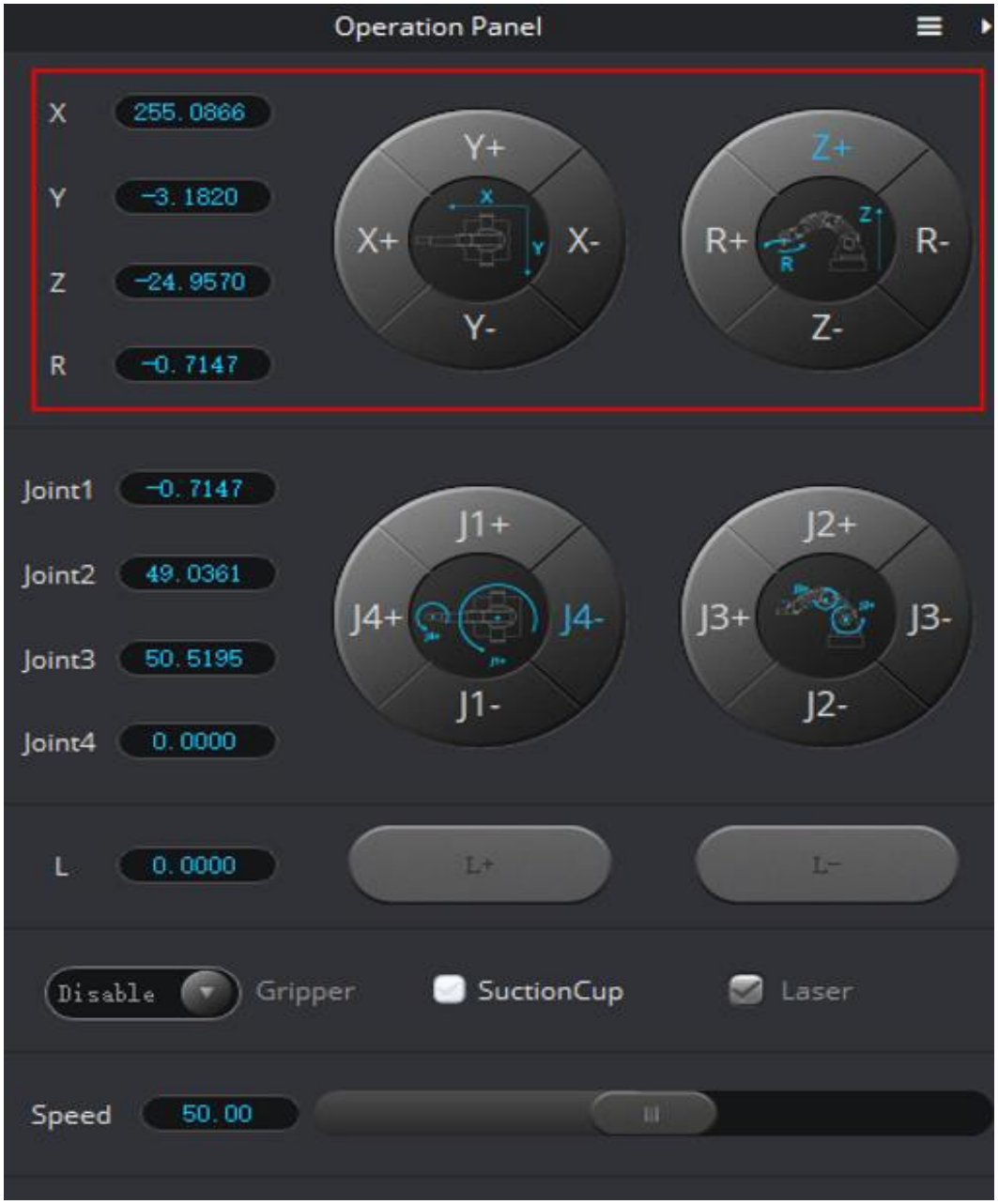

Рисунок 7

После установления нужных координат остаётся записать их в программе и приступить к написанию кода:

```
#include "SmartKit.h"       // Библиотека для работы с манипулятором
#define b_X 263                          //Координаты точки A
#define b_Y 3
#define b_Z -40
#define b_R 0

#define D_X 207                          //Координаты точки B
#define D_Y -171
#define D_Z -46
```

```
#define D_R 0

void setup() {
   Serial.begin(115200);
   Dobot_Init();                              // Активация манипулятора
   Serial.println("start...");

   // Перемещение захвата в стартовую позицию
   Dobot_SetPTPCmd(MOVJ_XYZ, 178, -4, 40, 0);
}

int count  = 4;

void loop(){
while(count > 0)
{
// Переместить захват в точку A и осуществить захват предмета
Dobot_SetPTPCmdEx(JUMP_XYZ,b_X, b_Y , b_Z , b_R );
Dobot_SetEndEffectorSuctionCupEx(true);

// Переместить захват повыше, чтобы предмет не касался земли
Dobot_SetPTPCmdEx((JUMP_XYZ,b_X, b_Y ,-4, b_R );

// Переместить захват в точку B и освободить предмет
Dobot_SetPTPCmdEx(JUMP_XYZ, D_X , D_Y , D_Z , D_R );
Dobot_SetEndEffectorSuctionCupEx(false);

// Переместить захват повыше, чтобы не касаться предмета
Dobot_SetPTPCmdEx(JUMP_XYZ, D_X , D_Y , -20, D_R );

// Вернуть захват в стартовую позицию
Dobot_SetPTPCmdEx(MOVJ_XYZ, 178, -4, 40, 0);

count--;
}
}
```

Задание

Подключите модуль голосового управления и измените программу из примера так, чтобы каждое действие манипулятора выполнялось по отдельной голосовой команде. Местоположения точки A и точки B выбрать самостоятельно, выбор голосовых команд на ваше усмотрение.

Порядок выполнения работы

1. Запустите ArduinoIDE и DobotStudio.
2. Подключите адаптер питания к основанию манипулятора, включите питание кнопкой, расположенной на основании. Дождитесь, когда манипулятор издаст короткий звуковой сигнал.
3. Выполните подключение манипулятора к компьютеру при помощи USB-кабеля.
4. В окне ПО DobotStudio нажмите на кнопку «Подключить», которая расположена в верхнем левом углу в окне программы.
5. Подключите плату Arduino к компьютеру. В программе ArduinoIDE выполните настройку подключения платы в инструменты -> Плата -> Arduino Mega or Mega 2560. Далее, убедитесь, что в пункте Процессор выделена строка ATmega2560 (Mega2560), в пункте Порт выделен порт с названием платы.
6. Напишите программу по заданию.
7. Загрузите скетч в плату.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE и DobotStudio.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Загрузить программу в плату и продемонстрировать её работу.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу программы на плате – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №14 «Сборка робота и подключение моторов»

Цель

Изучение функций драйвера моторов, подключение моторов, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

Управление миниатюрным электродвигателем (вибромотор), возможно при подключении его к выходу Arduino, однако, дискретный выход не сможет питать двигатели, потребляющие больше 40 мА. Наиболее правильным будет управление любыми нагрузками мощнее светодиода при помощи транзисторов или специальных микросхем – драйверов. Это нужно для того, что бы защитить ножку микроконтроллера от возможного повреждения из-за чрезмерно мощной нагрузки. Выход заключается в использовании простого усиливающего устройства, транзистора, чтобы иметь возможность управлять электродвигателями постоянного тока любой мощности. Наиболее удобный вариант для работы с электродвигателями – использование специализированной микросхемы – драйвера состоящего из специальной сборки транзисторов. На картинке далее приведены варианты драйверов с микросхемами tb6612, l298 и l293D.

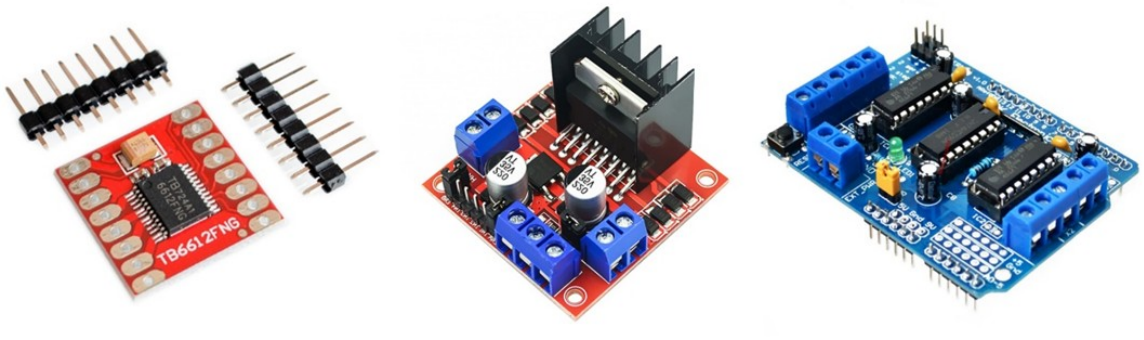

Рисунок 1: Примеры драйверов для двигателя.

Настройка и подключение драйвера

Моторы робота управляются контроллером не напрямую, а через микросхему-драйвер TB6612, для управления одним мотором необходимо 3 управляющих вывода микроконтроллера. (Для большинства микросхем – драйверов для управления мотором нужно 3 вывода микроконтроллера)

Проверьте подключение моторов. При сборке робота и программировании проверьте полярность подключения моторов и разъемы для подключения, что бы левый мотор был подключен к выводам драйвера, управляемым при помощи AIN1 и AIN2, а правый – выводам, управляемым к BIN1 и BIN2. В зависимости от платы драйвера, могут быть изменения в названии входов управления драйвера.

Проверьте подключение моторов! Левый мотор должен быть подключен к разъему M2, а правый - к M1.

Таблица 1: Соответствие ножек микроконтроллера на плате расширения ножкам микросхемы TB6612

ШИМ мотора 1	PWMA	D5
ШИМ мотора 2	PWMB	D13
Мотор 1	AIN1	D12
Мотор 1	AIN2	D4
Мотор 2	BIN1	D8
Мотор 2	BIN2	D10

Примечание: Для различных вариантов робота или разных версий плат возможно свое сочетание ножек МК и выводов микросхемы драйвера. Для различных драйверов соблюдается логика – два вывода микроконтроллера для задания направления вращения и один вывод для регулирования скорости вращения при помощи ШИМ сигнала.

Задание №1 Движение вперед

Первым делом необходимо заставить робота ездить по прямой, для этого нам понадобится открыть Arduino IDE, подключить плату RoBoard к компьютеру и написать следующий код:

```
//Для удобства создаем переменные, в которые записываем номера портов
#define LP 5 //Скорость левого мотора
#define L1 12 //Направление вращения левого мотора
#define L2 4 //Направление вращения левого мотора
#define RP 13 //Скорость вращения правого мотора
#define R1 8 //Направление вращения правого мотора
#define R2 10 //Направление вращения правого мотора
void setup() {
pinMode(LP,OUTPUT);
pinMode(L1,OUTPUT);
pinMode(L2,OUTPUT);
pinMode(RP,OUTPUT);
pinMode(R1,OUTPUT);
pinMode(R2,OUTPUT);
}
void loop() {
//Левое колесо вращается с ШИМом 100
analogWrite(LP,100);
digitalWrite(L1,LOW);
digitalWrite(L2,HIGH);
//Правое колесо вращается с ШИМом 100
analogWrite(RP,100);
digitalWrite(R1,LOW);
digitalWrite(R2,HIGH);
}
```

Рисунок 2: Движение вперед [код]

Задание №2 Реализация функции для движения вперед, выравнивание скорости вращения моторов

Вы можете заметить, что двигатели вращаются с разной скоростью и робот не может двигаться по прямой. Наша задача сделать замеры и заставить двигатели крутиться с одинаковой скоростью.

Напишем функцию движения:

```
void loop() {
  moveForward();

}

int speedRobot = 70; // Создаём переменную скорости

void moveForward(){ //Создаём функцию движения вперёд

  digitalWrite(L1, LOW);
  digitalWrite(L2, HIGH);
  digitalWrite(R1, LOW);
  digitalWrite(R2, HIGH);

  analogWrite(LP, speedRobot+5); // +5 Прибавляем скорости левому двигателю
                                 // чтобы правый и левый двигатели имели одинаковые скорости
  analogWrite(RP, speedRobot);  // Скорость правого двигателя
}
```

Рисунок 3: Функция движения [код]

Теперь изменим скорость моторов speedRobot и увидим, что моторы опять вращаются с разной скоростью. Зависимость сигнала ШИМ от скорости вращения моторов нелинейная и мы не можем точно предсказать, с какой скоростью вращается наш двигатель, поэтому мы сделаем замеры вручную, чтобы добиться максимально одинаковых скоростей вращения двигателей. Соотношения уровня ШИМ для моторов представлены в таблице 2.

Изменим значение скорости speedRobot несколько раз и для каждого замера заставим робота двигаться прямо.

Таблица 2: Пример соотношения ШИМ сигнала для левого и правого мотора и общей скорости робота.

speedRobot	LP	RP
50	50 '+ 0	50
70	70 '+ 5	70
90	90 '+ 20	90
100	100 '+ 10	100
120	120 '+ 14	120

На графике (рис.4) видно, что для одинаковых скоростей моторов на них нужно подавать разный уровень ШИМ для получения одинакового числа оборотов в секунду.

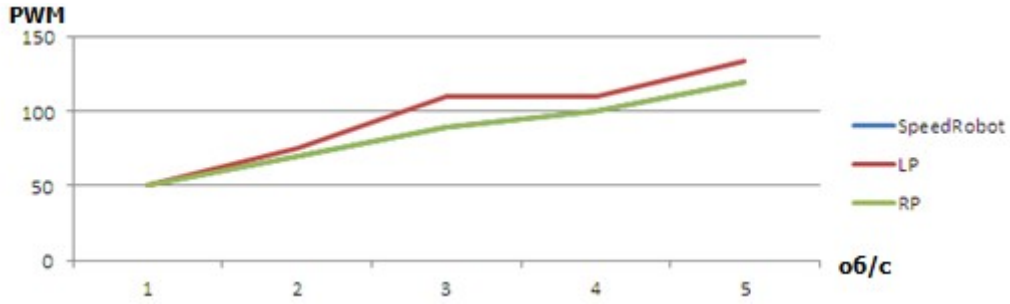

Рисунок 4: График соотношения скоростей вращения моторов

Перепишем код с учётом выравнивания скоростей:

```
void moveForward(){ //Создаём функцию движения вперёд
  int delta = 0;
  digitalWrite(L1, LOW);
  digitalWrite(L2, HIGH);
  digitalWrite(R1, LOW);
  digitalWrite(R2, HIGH);

  if (speedRobot < 70){                    // Если скорость в пределах от 0 до 70
    delta = 0;
  } else if (70 <= speedRobot < 90 ){      // Если скорость в пределах от 70 до 90
    delta = 5;                             // то левый мотор крутиться на 5 ед ШИМ быстрее
  } else if (90 <= speedRobot < 100 ){     // Если скорость в пределах от 90 до 100
    delta = 20;
  } else if (100 <= speedRobot < 120 ){    // Если скорость в пределах от 100 до 120
    delta = 10;
  } else {
    delta = 0;                             // Если скорость в пределах от 120 до 255 ничего не меняем
  }

  analogWrite(LP, speedRobot + delta); // +5 Прибавляем скорости левому двигателю
  analogWrite(RP, speedRobot);
}

void loop() {
  for (int i=50; i<140; i++){ // Постепенно увеличиваем скорость от 50 до 140
    speedRobot = i;           // смотрим, чтобы робот двигался прямо
    moveForward();
    delay(50);
  }
}
```

Рисунок 5: Выравниване скоростей [код]

Робот должен двигаться прямо с ускорением.

Задание №3 Функции для поворота на 180^o и 90^o

Для того, чтобы робот развернулся нам нужно, чтобы одно колесо вращалось в одну сторону, а другое в противоположную. Установив стандартную скорость поворота, например, равную 70, нам остаётся только замерить время, за которое робот совершит разворот. Теперь напишем функцию поворота и поэкспериментируем со временем.

```
void turnRobot(){
  digitalWrite(L1, HIGH); // Меням местами HIGH и LOW
  digitalWrite(L2, LOW);  // Меняется направление вращения левого мотора
  digitalWrite(R1, LOW);
  digitalWrite(R2, HIGH);

  analogWrite(LP, 70); // Задаём стандартную скорость поворота
  analogWrite(RP, 70); // Робот всегда будет поворачиваться с такой скоростью

  delay(980);          // Меняем время и смотрим, чтобы робот совершил полный оборот
}

void loop() {
    moveForward();    // Движемся прямо пол секунды
    delay(500);       // Задержка
    turnRobot();      // Разворачиваемся
}
```

Рисунок 6: Функция поворота

Меняя время задержки delay(980) мы смотрим, чтобы робот совершил разворот на 180^{o}.

Точно таким же образом напишите функцию поворота на 90^{o}.

Задание №4

Напишите программу для робота, что бы он двигался по восьмерке

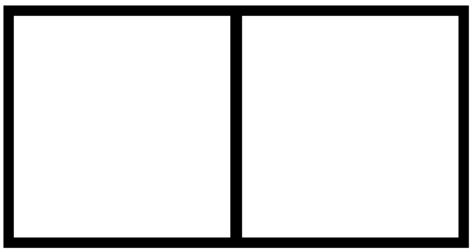

Рисунок 7: Пример траектории движения по восьмерке

Задание №5

Напишите программу для робота, что бы он двигался по кругу

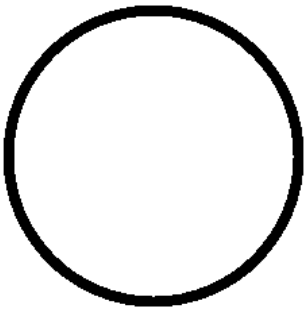

Рисунок 8: Пример траектории "Окружность"

Задание №6

Напишите программу для робота, что бы он двигался на определенное расстояние

Робот должен проехать вперёд на 70 см, затем повернуть направо и проехать ещё 30 см.

Задание №7

Напишите программу для робота, что бы он двигался по спирали

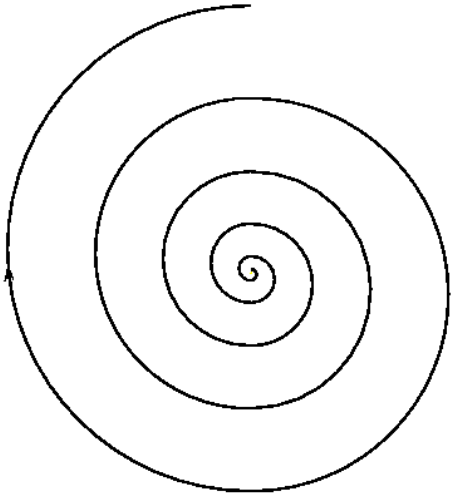

Рисунок 9: Пример траектории "Спираль"

Пример программы:

```
int tourtime=2000;         // время на один оборот.области пересечения (наложения) спиралей друг с другом
                           // оно должно быть меньше чем время проохождения круга
int increasetime=10;       // время следующего оборота, чем больше спираль, тем больше нужно времени на оборот
int numberofspiral=10;     // число кругов
int minspeed=50;           // начальная скорость для спирали.
int maxspeed=80;           // максимальная скорость

void spiralling() {
     digitalWrite(L1,HIGH);
     digitalWrite(L2,LOW);
     digitalWrite(R1,HIGH);
     digitalWrite(R2,LOW);

     for(int i=0;i<=numberofspiral;i++) {
       analogWrite(LP,maxspeed);
       analogWrite(RP,minspeed);
       minspeed=minspeed+1;              // увеличиваем скорость, чтобы увеличить радиус
       delay(tourtime);
       tourtime=tourtime+increasetime;  // увеличиваем время на оборот
     }
     analogWrite(RP,120);
     delay(150);
}

void loop() {
  spiralling();
  }
```

Рисунок 10: Движение по спирали [код]

Задание №7

Измените программу для робота, так что бы радиус спиралей уменьшался (Робот двигался к центру спирали)

Задание №8

Измените программу для робота, так что бы радиус спирали уменьшался, а затем увеличивался.

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение моторов к плате. В программе уделите внимание подключению правого мотора «Right Motor» и выводам, которым его подключение соответствует, повторите операцию для левого мотора «Left Motor». Заполните соответствующие строчки в программе..
3. Напишите функцию, реализующую движение робота вперед. Код есть в задании №1. Проведите несколько экспериментов, проверьте, что робот едет и моторы работают нормально.
4. Воспользовавшись материалами и рекомендациями из задания №2, попробуйте настроить скорости вращения для моторов.

5. Для дальнейшей работы вам понадобятся функции из задания №3. Перепишите функцию в ArduinoIDE и проверьте ее работу. Напишите функцию поворота на 90^o
6. Проверьте, что бы все функции работали и робот ездил по произвольной траектории.
7. После этого модифицируйте программу таким образом, что бы выполнить задания №4, 5, 6.
8. Для выполнения задания №7, изучите пример программы, загрузите ее в робота и посмотрите на его траекторию движения. Добавьте новые элементы из примера в ваш код.
9. Используя новые знания, выполните задания №8 и 9.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.

2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №15 «Подключение ИК датчиков поверхности»

Цель

Изучение принципов работы и функций ИК датчиков поверхности, их подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

Принцип работы датчика - светодиод излучает в инфракрасном диапазоне на длине волны 950 нм. Свет отражается от поверхности и попадает на фототранзистор. Сигнальный светодиод загорается, когда датчик находится над светлой (по его мнению) поверхностью, на выходе датчика устанавливается высокий логический уровень – «1» 3.3 или 5В в зависимости от напряжения питания датчика.

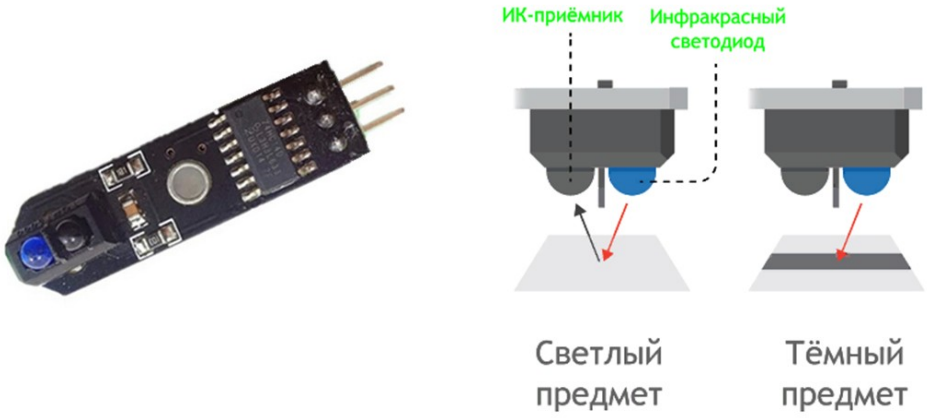

Рисунок 1: Общий вид ИК датчика и принцип его работы.

Для запуска программы примера вам понадобится собранный робот и два ИК датчика поверхности. В примере приводится алгоритм работы с датчиками при подключении левого датчика к «11» порту, а правого датчика - к «7». При использовании различных плат или конфигураций датчиков можно выбрать другие порты.

При наезде датчика на черную линию, он возвращает плате логический «0», и наоборот, при наезде на белое поле – «1».

Подключение датчиков и работа с данными

- Питание (V) — красный провод. На него должно подаваться напряжение 5 В (или 3,3 В).
- Земля (G) — чёрный провод. Должен быть соединён с землёй микроконтроллера.
- Сигнальный (OUT) — жёлтый провод. Подключается к цифровому входу микроконтроллера. Через него датчик передает микроконтроллеру бинарное значение, ноль или единицу.

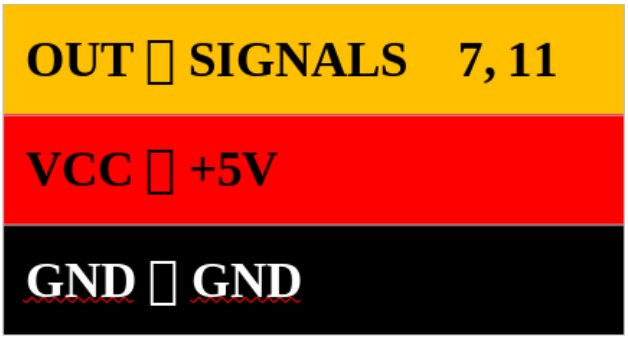

Рисунок 2: Схема подключения ИК датчика при помощи цветных проводов.

Снимем данные с датчиков и направим их в serial port

```
#define LEFTSENSOR 11 // Подключаем левый датчик к 11 порту на плате
#define RIGHTSENSOR 7 // Подключаем правый датчик к 7 порту на плате

void setup() {
  pinMode(LEFTSENSOR, INPUT);  // Настраиваем порты на вход
  pinMode(RIGHTSENSOR, INPUT);
  Serial.begin(9600);
}

void loop() {
  // Считываем значения с датчиков
  byte left = digitalRead(LEFTSENSOR);       // 0 - черный
  byte right = digitalRead(RIGHTSENSOR);     // 1 - белый

  Serial.print(left);
  Serial.println(right);

}
```

Рисунок 3: Вывод данных в последовательный порт [код]

Задание №1 Вывод данных в последовательный порт

Подведите датчики на чёрную и на белую линию и посмотрите, в каком виде они присылают данные.

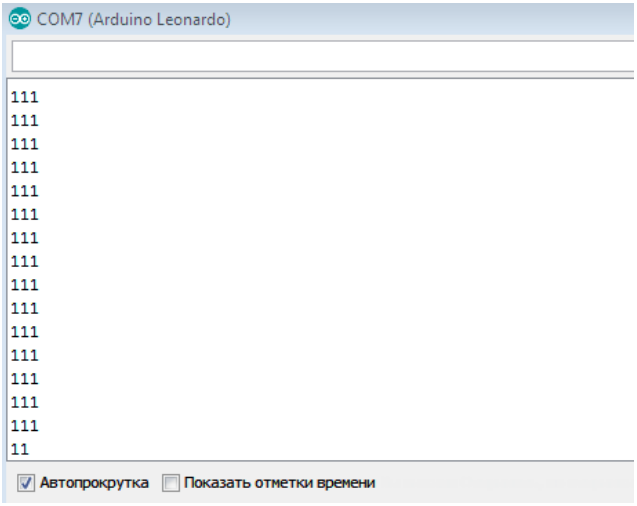

Рисунок 4: Пример вывода данных в терминал Arduino IDE

Движение по линии при помощи двух ИК датчиков.

В основном цикле программы реализуем алгоритм, проверяющий данные с датчиков и вызывающий соответствующие функции :

1. Считываем показания с датчиков, определяем чёрный 0 или белый 1 цвет зафиксировали датчики.

2. Приписываем условия. Какие функции будут выполняться при комбинации данных с датчиков.

Левый датчик	Правый датчик	Выполняемая функция
		go();
		stp();
		rght();
		lft();

Функция go()

```
void go() // Движемся вперёд
{
  analogWrite(LP, speedRobot);
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, speedRobot);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 5: Функция go() [код]

Функция stp()

```
void stp() // Останавливаемся
{
  analogWrite(LP, 0);
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, 0);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 6: Функция stp() [код]

Функция rght() и lft()

Прохождение плавных поворотов

```
void rght() // Поворачиваем направо
{
  analogWrite(LP, turnSpeed);  //Левое колесо движется
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, 0);           //Правое колесо стоит на месте
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}

void lft() // Поворачиваем налево
{
  analogWrite(LP, 0); //Левое колесо стоит на месте
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, turnSpeed);  //Правое колесо движется
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 7: Функция rght() и lft() [код]

Основная программа:

```
#define L1 12
#define L2 4
#define RP 13
#define R1 8
#define R2 10

#define LEFTSENSOR 11 // Подключаем левый датчик к 11 порту на плате
#define RIGHTSENSOR 7 // Подключаем правый датчик к 7 порту на плате

void setup() {
  pinMode(LP, OUTPUT);
  pinMode(L1, OUTPUT);
  pinMode(L2, OUTPUT);
  pinMode(RP, OUTPUT);
  pinMode(R1, OUTPUT);
  pinMode(R2, OUTPUT);

  pinMode(LEFTSENSOR, INPUT);  // Настраиваем порты на вход
  pinMode(RIGHTSENSOR, INPUT);
}


 int speedRobot = 40;  // Создаём переменную скорости
 int turnSpeed = 20;   // Скорость поворотов
```

Рисунок 8: Основная программа [код]

```
void loop() {
  // Считываем значения с датчиков
  byte left = digitalRead(LEFTSENSOR);       // 0 - черный
  byte right = digitalRead(RIGHTSENSOR);     // 1 - белый

  if ((left==1)&&(right==1))      //Если левый и правый датчики попадают на белое поле
  {                               //мы двигаемся вперёд
   go();
  }
  else if ((left==0)&&(right==0))  //Если левый и правый датчики попадают на чёрное поле поле
  {                                //мы останавливаемся
   stp();
  }
  else if ((left==1)&&(right==0))  //Если правый датчик попадает на чёрное поле поле
  {                                //мы поварачиваем направо
   rght();
  }
   else if ((left==0)&&(right==1))  //Если левый датчик попадает на чёрное поле поле
  {                                 //мы поварачиваем налево
   lft();
  }

}
```

Рисунок 9: Основная программа [код]

Задание № 2

Протестируйте робота на тестовом полигоне. Поэкспериментируйте со скоростями (максимальная, минимальная), с расположением датчиков на роботе (уже, шире). Найдите участки трассы, которые робот не может преодолеть.

Задание №3

Разработайте алгоритм плавного прохождения поворота, для прохождения данного участка трассы.

Рисунок 10: Траектория движения

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение датчиков к плате. В программе уделите внимание подключению левого и правого датчика. Посмотрите на плате номер порта и ножек и впишите соответствующие цифры в код программы.

3. Проверьте подключение датчиков,- необходимо убедиться, что питание подключено правильно.
4. Включите питание и перепишите код программы примера. Попробуйте его в работе.
5. Напишите функцию, реализующую задание №1. Проведите несколько экспериментов, проверьте, что все работает нормально.
6. Дополните код функциями для движения по линии при помощи двух датчиков. Проверьте корректность работы новой программы.
7. Испытайте робота на треке, проведите несколько экспериментов, со скоростями (максимальная, минимальная), с расположением датчиков на роботе (уже, шире). Найдите участки трассы, которые робот не может преодолеть.
8. Модифицируйте программу таким образом, чтобы робот мог преодолевать плавные повороты и повороты на 90° градусов.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №16 «Движение робота по датчикам»

Цель

Изучение функций и работы ИК датчиков, их подключение, доработка программы для управления движением робота при помощи трех датчиков.

Краткие теоретические сведения

Увеличение количества датчиков позволяет реализовать плавное движение робота по траектории. С помощью трёх датчиков мы можем описать большее количество возможных ситуаций, и, следовательно, лучше преодолевать трассу. Для понимания рассмотрим все возможные ситуации движения робота по линии

Предположим, что наш трек выглядит следующим образом:

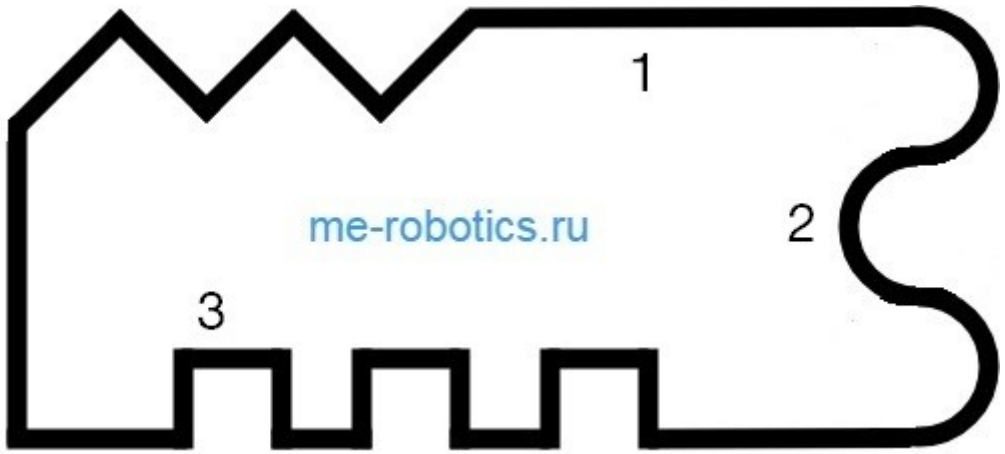

Рисунок 1: Общий вид трассы для движения робота

Нам потребуются следующие функции:

Участок	Что делаем	Функция
1	Едем вперёд	go();
2	Поворачиваем медленно направо/налево	rght(); и lft();
3	Поворачиваем на 90^0 направо/налево	turnLft(); и turnRght();
4	Останавливаемся	stp();

Теперь определяем, при каких показаниях датчиков нам следует выполнять наши функции.

Левый датчик	Центральный датчик	Правый датчик	Выполняемая функция
			go();
			go();
			rght();
			lft();
			stp();
			turnlft();
			turnrght();
			go();

Функция go()

```
void go() // Движемся вперёд
{
  analogWrite(LP, speedRobot);
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, speedRobot);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 2: Функция go() [код]

Функции fastright() и fastleft()

```
void fastright()
{
  analogWrite(LP,speedRobot);
  digitalWrite(L1,HIGH);
  digitalWrite(L2,LOW);
  analogWrite(RP,speedRobot);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

```
void fastleft() //Резко поворачиваем направо
{
analogWrite(LP, turnSpeed);
digitalWrite(L1,LOW); //Левый мотор вперёд
digitalWrite(L2,HIGH);
analogWrite(RP, turnSpeed);
digitalWrite(R1,HIGH); //Правый мотор назад
digitalWrite(R2,LOW);

}
```

Рисунок 3: Крутой поворот направо/налево

Одно колесо крутим вперёд, другое назад. Робот разворачивается относительно центральной точки.

Алгоритм прохождения резкого поворота

Как сделать крутой поворот робота на 90° или больше градусов?

1. Робот зафиксировал резкий поворот

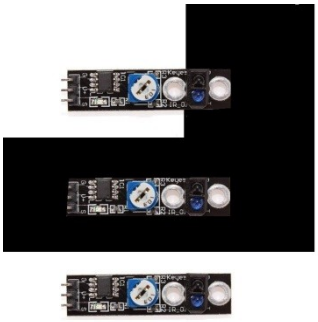

Рисунок 4: Прохождение поворота в 90 градусов

2. Робот начинает функцию поворота на 90^{o}
3. Робот проезжает вперёд и останавливается центром на повороте

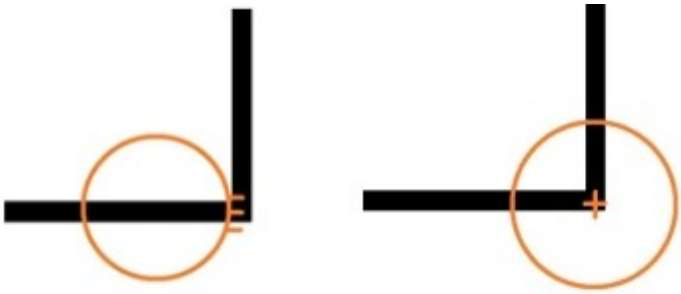

Рисунок 5: Прохождение поворота в 90 градусов

4. Робот разворачивается относительно центральной точки (одно колесо вперёд, другое назад)
5. Робот поворачивает до тех пор, пока центральный датчик не зафиксирует чёрную линию

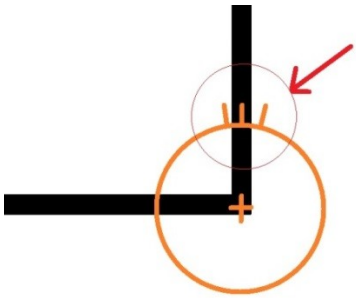

Рисунок 6: Завершение поворота

6. Поворот завершён

Функции turnLft() и turnRght(), реализующие ранее описанный алгоритм.

```
void turnrght(){
  go();
  delay(500);//700
  while(1){
    if(digitalRead(CENTERSENSOR)==0)
      break;

    fastright();
  }
  stp();
}

void turnlft(){
  go();
  delay(500);//700
  while(1){
    if(digitalRead(CENTERSENSOR)==0)
      break;

    fastleft();
  }
  stp();
}
```

Рисунок 7: Функции turnLft() и turnRght() [код]

В исходном коде программы отмечены новые строчки, позволяющие усовершенствовать алгоритм работы робота.

Основная программа:

```
#define LP 5
#define L1 12
#define L2 4
#define RP 13
#define R1 8
#define R2 10

#define LEFTSENSOR 11 // Подключаем левый датчик к 11 порту на плате
#define CENTERSENSOR 3
#define RIGHTSENSOR 7 // Подключаем правый датчик к 7 порту на плате

void setup() {
  pinMode(LP, OUTPUT);
  pinMode(L1, OUTPUT);
  pinMode(L2, OUTPUT);
  pinMode(RP, OUTPUT);
  pinMode(R1, OUTPUT);
  pinMode(R2, OUTPUT);

  pinMode(LEFTSENSOR, INPUT);  // Настраиваем порты на вход
  pinMode(CENTERSENSOR, INPUT);
  pinMode(RIGHTSENSOR, INPUT);
  Serial.begin(9600);
}

int speedRobot = 40;  // Создаём переменную скорости
int turnSpeed = 40;   // Скорость поворотов
```

Рисунок 8: Основная программа

```
void lft() // Плавно поворачиваем налево
{

void rght() // Плавно поворачиваем направо
{

void fastright() //Резко поворачиваем налево
{

void fastleft() //Резко поворачиваем направо
{

void turnrght(){ //Поворот больше чем на 90 градусов направо

void turnlft(){ //Поворот больше чем на 90 градусов налево
```

Рисунок 9: Основная программа

```
void loop() {

  // Считываем значения с датчиков
  byte left = digitalRead(LEFTSENSOR);        // 0 - черный
  byte right = digitalRead(RIGHTSENSOR);      // 1 - белый
  byte center = digitalRead(CENTERSENSOR);
  Serial.print(left);
  Serial.print(center);
  Serial.println(right);
```

Рисунок 10: Основная программа

```
 if ((left==1)&&(center==0)&&(right==1)) //Если левый и правый датчики попадают на белое поле,
                                         //а средний на чёрное
 {                                       //Мы двигаемся вперёд
  go();
 }
 else if ((left==1)&&(center==1)&&(right==1))  //Если все датчики попадают на белое поле
 {                                             //Мы двигаемся вперёд
  go();
 }
 else if ((left==1)&&(center==1)&&(right==0))  //Если левый и средний датчики попадают на белое поле
 {                                             //Мы двигаемся направо
  rght();
 }
 else if ((left==0)&&(center==1)&&(right==1))  //Если правый и средний датчики попадают на белое поле
 {                                             //Мы двигаемся налево
  lft();
 }
 else if ((left==0)&&(center==0)&&(right==0))  //Если все датчики попадают на чёррное поле
 {                                             //Мы останавливаемся
  stp();
 }
 else if ((left==0)&&(center==0)&&(right==1))  //Если левый и средний датчики попадают на чёрное поле
 {                                             //РЕЗКИЙ ПОВОРТ НАЛЕВО
  turnlft();
 }
 else if ((left==1)&&(center==0)&&(right==0))  //Если правый и средний датчики попадают на чёрное поле
 {                                             //РЕЗКИЙ ПОВОРТ НАПРАВО
  turnrght();
 }
 else {                                        //В любом другом случае
   go();                                       //Мы двигаемся вперёд
 }
}
```

Рисунок 11: Основная программа

Задание №1

Протестируйте робота на полигоне.

Задание №2

Увеличьте скорость. Доработайте робота для скоростного прохождения полигона.

Задание №3

Устройте соревнования на скоростное прохождение полигона.

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение дополнительного датчика к плате. В программе уделите внимание подключенным датчикам. Посмотрите на плате номер порта и ножек и впишите соответствующие цифры в код программы.
3. Проверьте подключение датчиков,- необходимо убедиться, что питание подключено правильно.
4. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.
5. Попробуйте модифицировать алгоритм, для того что бы робот мог поворачивать на месте и двигаться не только по плавным траекториям.
6. Добавьте новые фрагменты кода для реализации поворотов в свою программу. Пример модификации базовой программы приведен в теоретической части.
7. Испытайте робота на треке. Добейтесь того, что робот проехал трассу полностью. Попробуйте увеличивать скорость движения робота.
8. Попробуйте посоревноваться с товарищами. Узнайте насколько быстро можно проехать всю трассу. Модифицируйте код программы для увеличения скорости и алгоритмов движения по трассе.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.

3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №17 «Подключение ИК датчиков препятствий»

Цель

Изучение функций и работы ИК датчиков, их подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

В этой лабораторной работе рассмотрим один из самых распространенных датчиков препятствия, который работает по принципу отражения. Устроен он очень просто. Датчик содержит направленный источник света и детектор света. Источником часто служит инфракрасный светодиод с линзой, а детектором — фотодиод или фототранзистор.

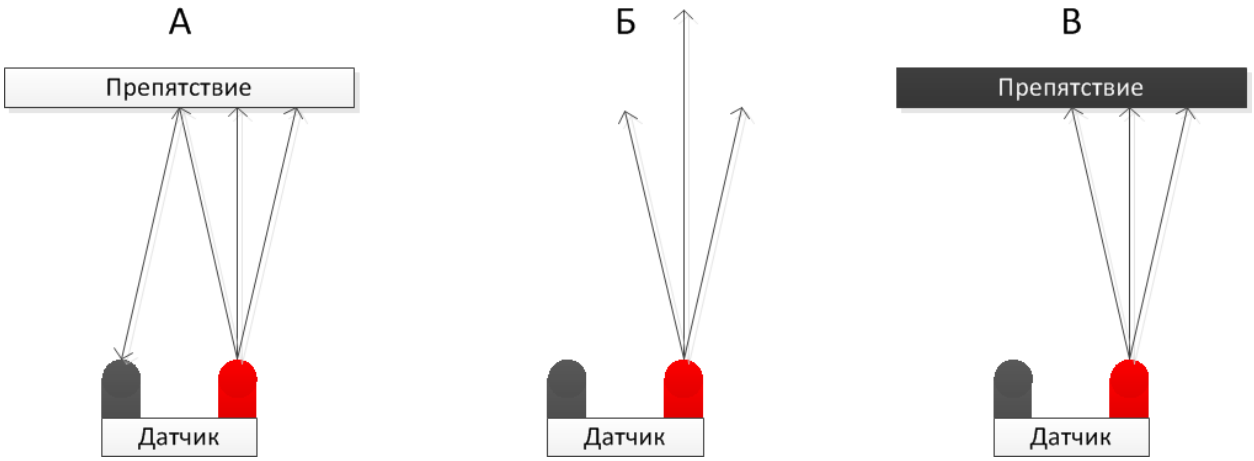

Рисунок 1: Принцип работы ИК датчика препятствия

А. Если перед датчиком есть препятствие, то на детектор попадает отраженное ИК излучение от источника, и на выходе датчика появляется высокий логический уровень.

Б. Если препятствия нет, то ИК излучение на датчик не попадает и логический уровень на выходе низкий.

В. Есть и третий вариант, когда препятствие есть, но свет от него не отражается! Получается, матовую черную поверхность робот не увидит.

При встрече с препятствием датчик возвращает «1» контроллеру, если же препятствия нет, то «0».

Рисунок 2: Общий вид ИК датчика препятствия.

Подключение датчика препятствия

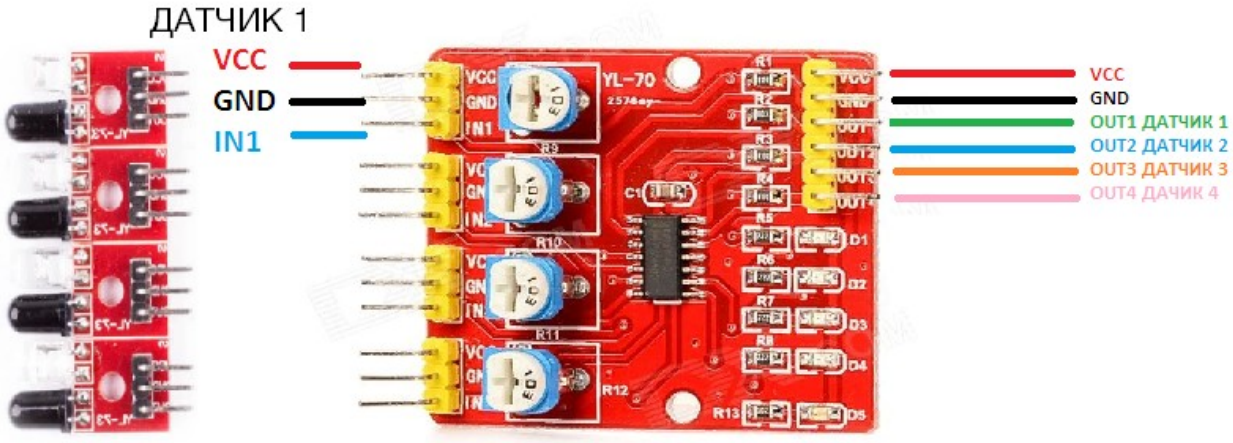

Рисунок 3: Схема подключения ИК датчика препятствия

Подключаем датчик, считываем сигнал с помощью последовательного порта и смотрим, на каком расстоянии срабатывает датчик. С помощью резистора на плате управления можно регулировать порог срабатывания датчика, тем самым изменяя расстояние срабатывания.

Задание № 1

Проехать по чёрной линии, если на пути робота появляется препятствие, её нужно объехать и продолжить движение по полигону.

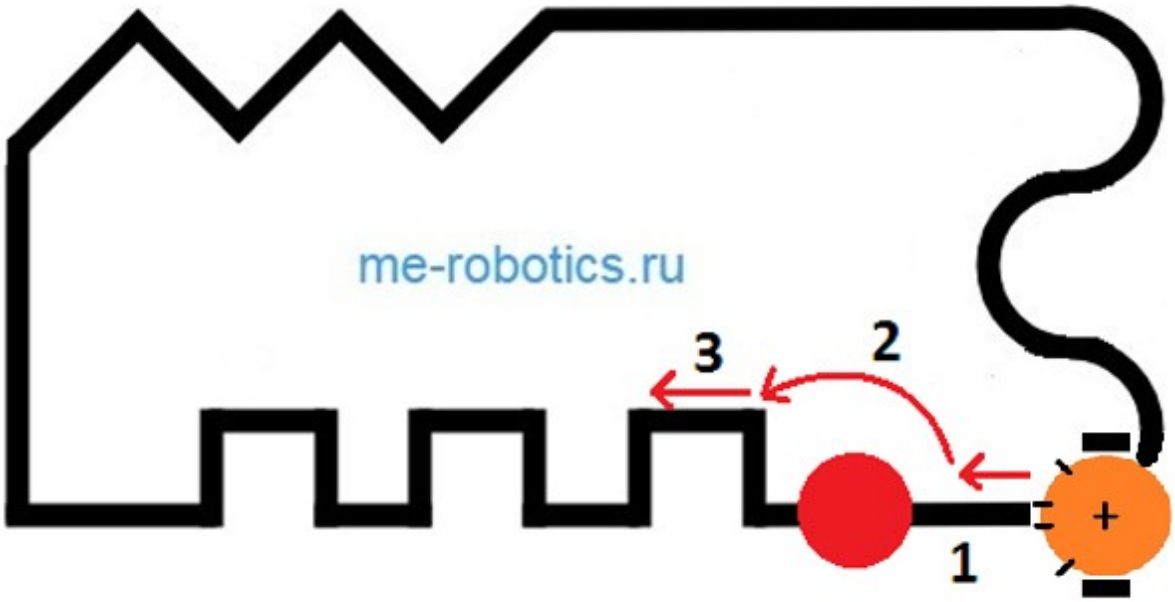

Рисунок 4: Схема проезда трассы с препятствием

1. Робот увидел препятствие (поворот 90 градусов направо);
2. робот начал объезд препятствия (движение по окружности);
3. робот вернулся на трек (фиксирует центральным датчиком чёрную линию).

Дополним нашу программу с передвижением по чёрной линии.

```
#define OBST_SEN 1 // Подключим датчик к порту 1
pinMode(OBST_SEN, INPUT); //Объявление порта 1 как входного в setup()
```

Добавляем новую функцию объезда препятствия.

```
void obstSens(){
  if (digitalRead(OBST_SEN)==0){
    fastright(); // Если сработал датчик
    delay(1000); // То поворачиваем направо 1000 мс (90 градусов)

    while(1){ //Движемся по окружности, объезжая препятствие
      if(digitalRead(CENTERSENSOR)==0) // Если срабатывает центральный датчик черной линии
        break;      // то мы заканчиваем объезжать препятствие и переходим к основному алгоритму
      rndleft();
    }

    while(1){ // Доворачиваем направо, чтобы выровнить робота относительно линии
      if((digitalRead(LEFTSENSOR)==1) && (digitalRead(CENTERSENSOR)==0) && (digitalRead(RIGHTSENSOR)==1))
        break;
      fastright();
    }
  }
}
```

Рисунок 5: Функция для объезда препятствия

Функция rndLeft – робот движется по окружности некоторого радиуса (участок 2 на рисунке).

```
void rndleft(){
  analogWrite(LP, 2*speedRobot);
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, (speedRobot+diff));
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 6: Функция rndLeft

Добавим функцию obstSens в основной цикл программы. Таким образом, перед тем как начать следовать линии, робот сначала всегда будет проверять есть ли препятствие.

```
void loop() {
  obstSens();
}
```

```
  if ((left==1)&&(center==0)&&(right==1)) //Если левый и правый датчики попадают на белое поле,
                                          //а средний на чёрное
  {                                       //Мы двигаемся вперёд
   go();
  }
  else if ((left==1)&&(center==1)&&(right==1))  //Если все датчики попадают на белое поле
  {                                             //Мы двигаемся вперёд
   go();
  }
  else if ((left==1)&&(center==1)&&(right==0))  //Если левый и средний датчики попадают на белое поле
  {                                             //Мы двигаемся направо
   rght();
  }
  else if ((left==0)&&(center==1)&&(right==1))  //Если правый и средний датчики попадают на белое поле
  {                                             //Мы двигаемся налево
   lft();
  }
  else if ((left==0)&&(center==0)&&(right==0))  //Если все датчики попадают на чёррное поле
  {                                             //Мы останавливаемся
   stp();
  }
  else if ((left==0)&&(center==0)&&(right==1))  //Если левый и средний датчики попадают на чёрное поле
  {                                             //РЕЗКИЙ ПОВОРТ НАЛЕВО
   turnlft();
  }
  else if ((left==1)&&(center==0)&&(right==0))  //Если правый и средний датчики попадают на чёрное поле
  {                                             //РЕЗКИЙ ПОВОРТ НАПРАВО
   turnrght();
  }
  else {                                        //В любом другом случае
    go();                                       //Мы двигаемся вперёд
  }
}
```

Рисунок 7: Основной цикл программы

Задание №2

Направить датчики препятствия вниз, чтобы робот смог чувствовать край стола. Написать программу, которая предотвращает падение робота со стола. (Для этой цели также можно использовать датчики чёрной линии, т.к. при виде края стола сигнал не будет возвращаться к приёмнику после отражения).

Задание №3

Направить датчики в стороны, и заставить робота двигаться вдоль стены.

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение датчиков препятствий к плате. В программе уделите внимание подключенным датчикам. Посмотрите на плате номер порта и ножек и впишите соответствующие цифры в код программы.

3. Проверьте подключение датчиков,- необходимо убедиться, что питание подключено правильно. Проверьте питание и правильность подключения платы с потенциометрами.
4. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.
5. Модифицируйте код и положение датчиков таким образом, чтобы робот смог «чувствовать» край стола. Написать программу, которая предотвращает падение робота со стола.
6. Модифицируйте код и положение датчиков таким образом, чтобы робот смог ехать вдоль стены.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №18 «Объезд препятствий при помощи ИК датчиков препятствий»

Цель

Изучение функций и работы ИК датчиков, их подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

Для того, что бы реализовать движение робота необходимо написать и применить в коде программы несколько функций. Ранее мы разработали и использовали функцию go(), на основе которой можно разработать функцию reverse().

```
void go() // Движемся вперёд
{
  analogWrite(LP, speedRobot);
  digitalWrite(L1,LOW);
  digitalWrite(L2,HIGH);
  analogWrite(RP, speedRobot);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

Рисунок 1: Функция go [код]

Функции fastright() и fastleft()

```
void fastright()
{
  analogWrite(LP,speedRobot);
  digitalWrite(L1,HIGH);
  digitalWrite(L2,LOW);
  analogWrite(RP,speedRobot);
  digitalWrite(R1,LOW);
  digitalWrite(R2,HIGH);
}
```

```
void fastleft() //Резко поворачиваем направо
{
analogWrite(LP, turnSpeed);
digitalWrite(L1,LOW); //Левый мотор вперёд
digitalWrite(L2,HIGH);
analogWrite(RP, turnSpeed);
digitalWrite(R1,HIGH); //Правый мотор назад
digitalWrite(R2,LOW);

}
```

Рисунок 2: Функции fastright() и fastleft()

Можно модифицировать меняя скорость или время выполнения функции.

Для того, что бы реализовать постоянное движение робота вы должны вызывать функции движения в зависимости от сигналов, получаемых с датчиков. Изучим варианты алгоритма для решения этой задачи. Ехать нужно

осторожно, так как банки разных цветов определяются датчиками на разном расстоянии.

Задача №1 Объезд препятствий

Задача: ехать, избегая столкновений с препятствиями.

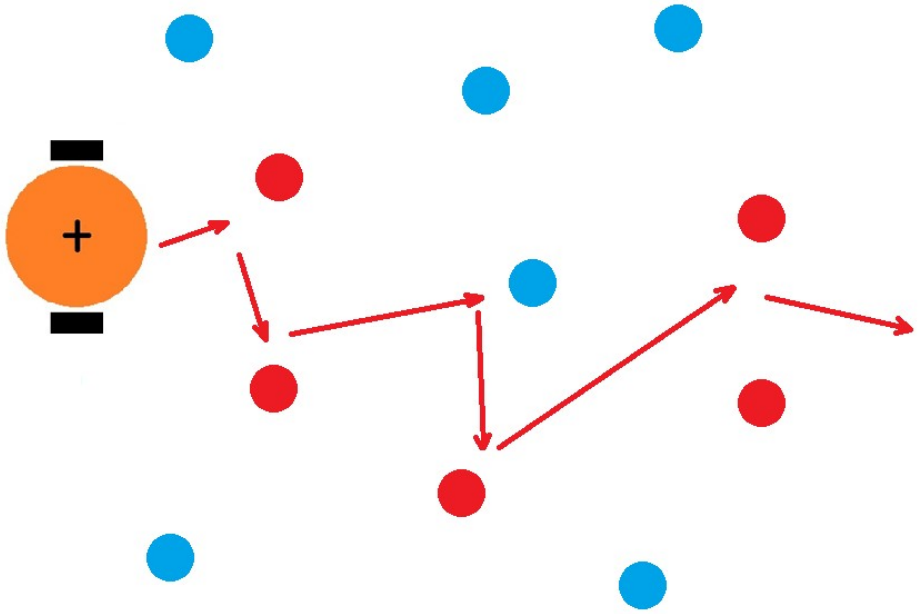

Рисунок 3: Схема расположения препятствий

В качестве датчиков будем использовать уже знакомый модуль:

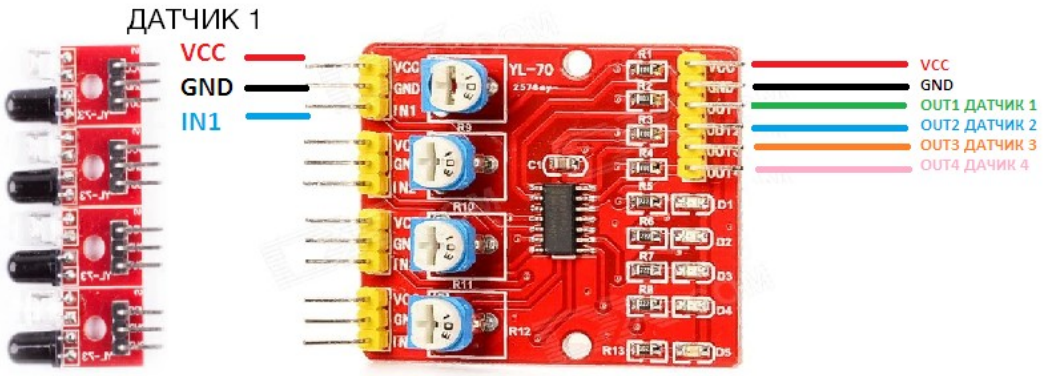

Рисунок 4: ИК датчики препятствий

Подключаем и настраиваем 4 датчика.

Испытание будет проходить на столе, подключите 3 датчика чёрной линии и напишите программу, чтобы робот не падал со стола. Для того что бы это сделать необходимо расположить датчики линии так, что бы они срабатывали и

давали на выходе логическую «1» пока робот стоит на столе. При подъезде к краю – ИК излучение не будет отражаться и датчики вернут «0»

Возможное решение:

Устанавливаем и настраиваем 4 датчика препятствий. Прописываем 5 функций поворота:

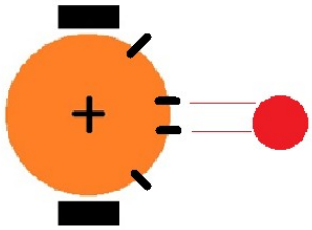

Рисунок 5: Робот видит препятствие двумя центральными датчиками.

Отъезжаем назад и поворачиваем в сторону, чтобы объехать препятствие.

Отъезжаем назад и объезжаем.

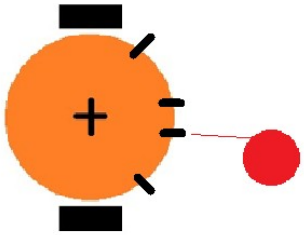

Рисунок 6: Робот видит препятствие одним из центральных датчиков

Немного поворачиваем в сторону.

Поворот на малый угол.

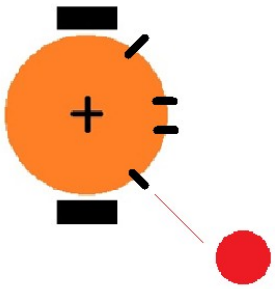

Рисунок 7: Робот видит препятствие одним из крайних датчиков

Сильно поворачиваем в сторону.

Поворот на большой угол.

Задача № 2 Объезд препятствий в течение заданного времени

Робот должен ездить по столу и избегать столкновений в течение 30 сек. При этом он не должен упасть со стола.

Задача № 3 Объезд препятствий на прямой

Робот должен проехать строго в одну сторону слава направо, преодолевая все препятствия по пути.

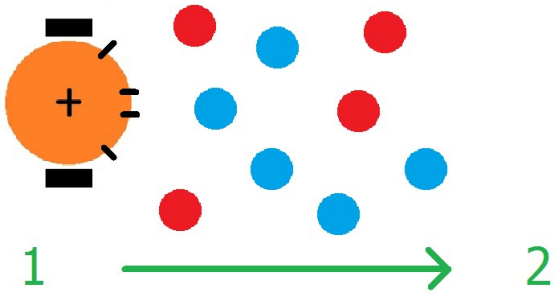

Рисунок 8: Объезд препятствий на прямой

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Проверьте подключение датчиков, номера портов, питание и землю, правильность подключения платы с потенциометрами.
3. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.

4. Модифицируйте код, так что бы робот мог ехать, избегая столкновений с препятствиями.
5. Объедините код, отвечающий за определение края стола и объезд препятствий. Испытание будет проходить на столе, подключите и правильно расположите все датчики.
6. Модифицируйте код, так что бы робот мог двигаться по алгоритмам, описанным в заданиях 1 и 2.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №19 «Подключение УЗ датчика препятствий»

Цель

Изучение функций и работы УЗ датчиков, их подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

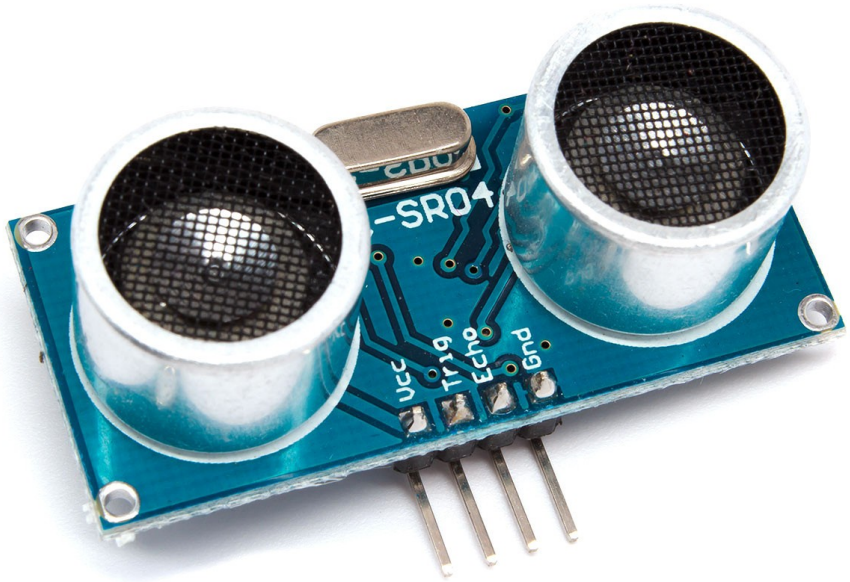

Рисунок 1: Ультразвуковой датчик

Этот дальномер может служить прекрасным датчиком для робота, благодаря которому он сможет определять расстояния до объектов, объезжать препятствия, или строить карту помещения. Его можно также использовать в качестве датчика для сигнализации, срабатывающего при приближении объектов.

Принцип действия

Ультразвуковой дальномер определяет расстояние до объектов точно так же, как это делают дельфины или летучие мыши. Он генерирует звуковые импульсы на частоте 40 кГц и слушает эхо. По времени распространения звуковой волны туда и обратно можно однозначно определить расстояние до объекта.

В отличие от инфракрасных дальномеров, на показания ультразвукового дальномера не влияют засветки от солнца или цвет объекта. Но могут возникнуть трудности с определением расстояния до пушистых или очень тонких предметов.

В отличие от ультразвукового дальномера URM37, этот дальномер не обладает таким большим выбором интерфейсов и режимов работы. Но этот «недостаток» компенсируется простотой работы с ним. Если вы планируете использовать его с Arduino вы можете воспользоваться существующими библиотеками:

- Ultrasonic — самая популярная библиотека для HC-SR04.
- NewPing — отличается большей точностью и скоростью работы.

Подключение к Arduino

Контакты датчика можно соединить с макетной платой или Arduino проводами «мама-папа».

Распиновка

1. Vcc — положительный контакт питания.
2. Trig — цифровой вход. Для запуска измерения необходимо подать на этот вход логическую единицу на 10 мкс. Следующее измерение рекомендуется выполнять не ранее чем через 50 мс.
3. Echo — цифровой выход. После завершения измерения, на этот выход будет подана логическая единица на время, пропорциональное расстоянию до объекта.
4. GND — отрицательный контакт питания.

Характеристики

1. Напряжение питание: 5 В
2. Потребление в режиме тишины: 2 мА
3. Потребление при работе: 15 мА
4. Диапазон расстояний: 2–400 см
5. Эффективный угол наблюдения: 15°

6. Рабочий угол наблюдения: 30°

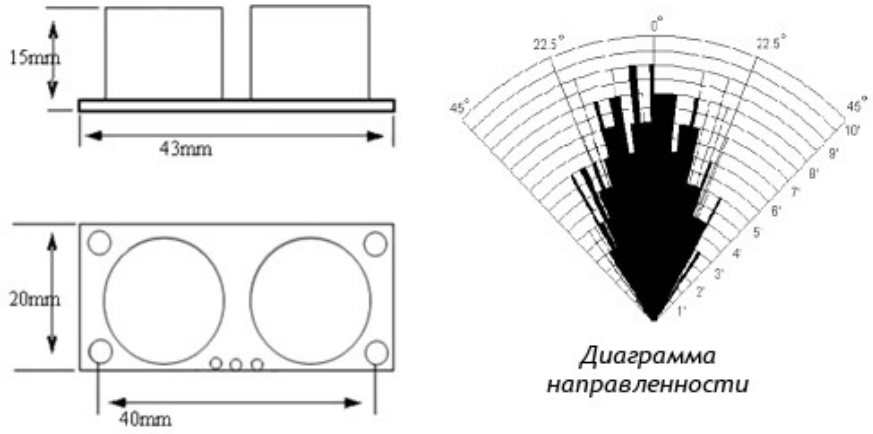

Рисунок 2: Ультразвуковой датчик, размеры и угол обзора

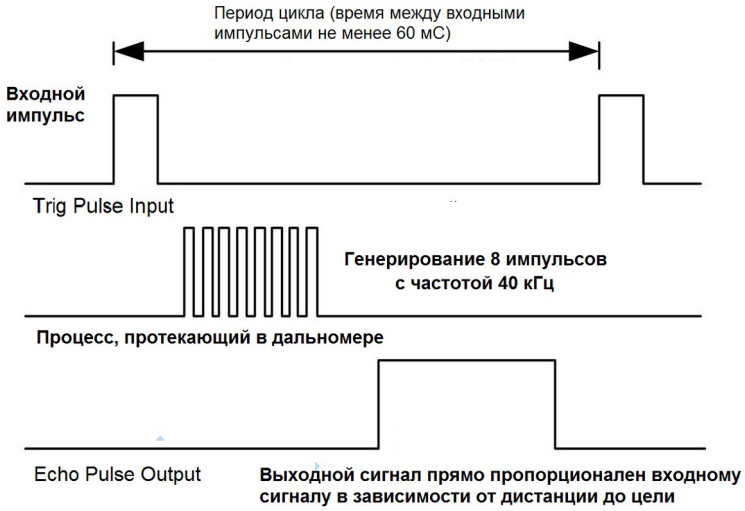

Рисунок 3: Ультразвуковой датчик, временные диаграммы сигналов

Пример программного кода, реализующий работу с ультразвуковым датчиком без использования библиотеки:

```
#define Trig 9
#define Echo 8
#define ledPin 13

void setup()
{
 pinMode(Trig, OUTPUT); //инициируем как выход
```

```
 pinMode(Echo, INPUT); //инициируем как вход
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
 /* задаем скорость общения. В нашем случае с компьютером */
}
unsigned int impulseTime=0;
unsigned int distance_sm=0;

void loop()
{
 digitalWrite(Trig, HIGH);
 /* Подаем импульс на вход trig дальномера */
 delayMicroseconds(10); // равный 10 микросекундам
 digitalWrite(Trig, LOW); // Отключаем
 impulseTime=pulseIn(Echo, HIGH); // Замеряем длину импульса
 distance_sm=impulseTime/58; // Пересчитываем в сантиметры
 Serial.println(distance_sm); // Выводим на порт
 if (distance_sm<30) // Если расстояние менее 30 сантиметром
 {
  digitalWrite(ledPin, HIGH); // Светодиод горит
 }
 else
 {
  digitalWrite(ledPin, LOW); // иначе не горит
 }
 delay(100);
 /* ждем 0.1 секунды, Следующий импульс может быть излучён, только после
исчезновения эха от предыдущего.
  Это время называется периодом цикла (cycle period).
  Рекомендованный период между импульсами должен быть не менее 50 мс. */
}
```

Пример с библиотекой Ultrasonic, реализующий считывание показаний датчика и их отправку в терминал.

```
#include "Ultrasonic.h"

// sensor connected to:
// Trig - 12, Echo — 13
Ultrasonic ultrasonic(12, 13);

void setup()
{
  Serial.begin(9600); // start the serial port
}
```

```
void loop()
{
  float dist_cm = ultrasonic.Ranging(CM); // get distance
  delay(100); // arbitary wait time.
}
```

Задача №1 Объезд препятствий

Модифицируйте код добавив к нему работу с ультразвуковым датчиком. Ультразвуковой датчик одинаково хорошо определяет банки разных цветов, в отличии от ИК датчиков.

Задача: проехать серию препятствий, избегая столкновений.

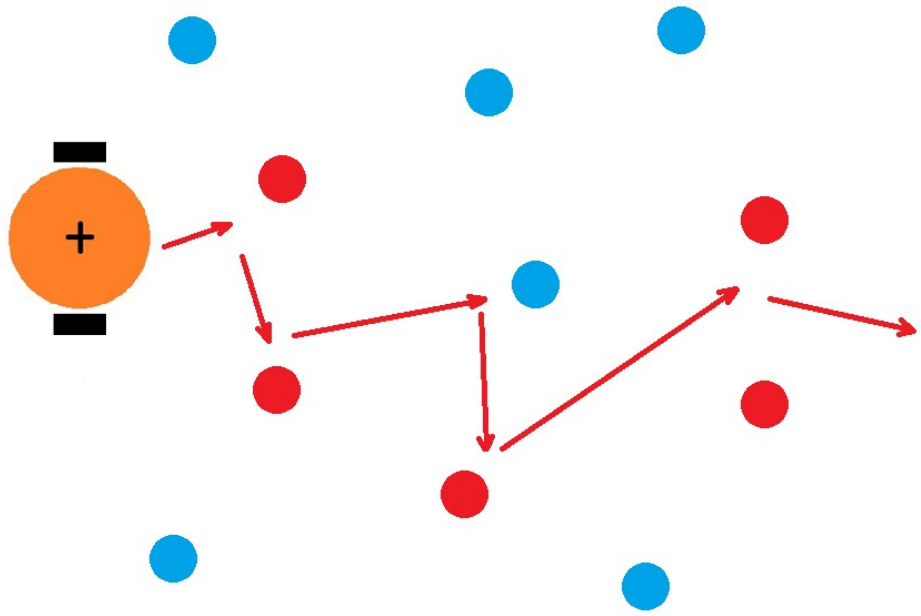

Рисунок 4: Схема расположения препятствий

Часть алгоритма останется прежней, но так как УЗ датчик расположен по центру робота – надо будет переработать часть кода отвечающую за работу с центральными датчиками.

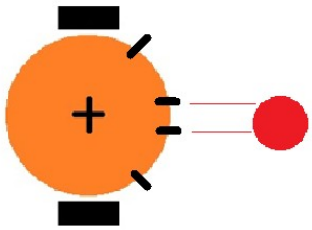

Рисунок 5: Робот видит препятствие УЗ дальномером

Отъезжаем назад и поворачиваем в сторону, чтобы объехать препятствие. Центральные ИК датчики можно расставить шире.

Отъезжаем назад и объезжаем.

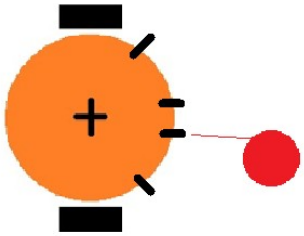

Рисунок 6: Робот видит препятствие одним из центральных датчиков

Немного поворачиваем в сторону.

Поворот на малый угол.

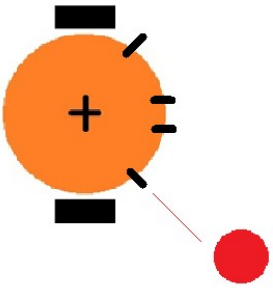

Рисунок 7: Робот видит препятствие одним из крайних датчиков

Сильно поворачиваем в сторону.

Поворот на большой угол.

Задача №2 Объезд препятствий с ограничением по времени

Робот должен ездить по столу и избегать столкновений в течение 30 сек. Используем ИК датчики и УЗ дальномер.

При этом он не должен упасть со стола.

Задача №3 Объезд препятствий на прямой

Робот должен проехать строго в одну сторону слава направо, преодолевая все препятствия по пути. Проверим работу алгоритма после добавления в него обработчик для УЗ датчика.

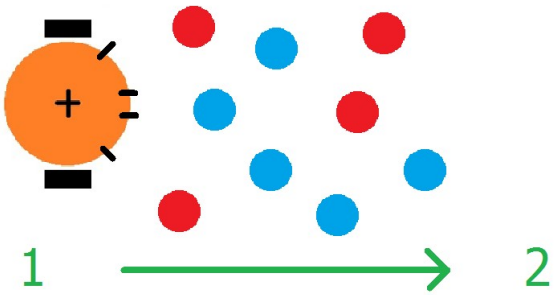

Рисунок 8: Схема расположения препятствий

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение УЗ датчика к плате. В программе уделите внимание подключению датчика. Посмотрите на плате номера портов и ножек и впишите соответствующие цифры в код программы.
3. Проверьте подключение датчика,- необходимо убедиться, что питание подключено правильно.
4. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.
5. Модифицируйте код, добавив к нему работу с ультразвуковым датчиком. Добейтесь того, что бы робот объезжал препятствия, используя новый датчик. Выполните задания 1, 2 и 3, используя УЗ датчик.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №20 «Подключение энкодеров для регулирования скорости колес»

Цель

Изучение функций и работы энкодреров, их подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

Из-за разных скоростей вращения колес ваш робот будет отклоняться в ту или иную сторону во время движения, это связано с тем, что моторы, используемые в наборе, хоть и из одной партии, но так или иначе будут немного отличаться. Это можно исправить с помощью энкодеров, установленных на вашем роботе.

Энкодер

Сигнал энкодера имеет цифровой а не аналоговый характер. Имеются два вида энкодеров: инкрементальные и абсолютные.

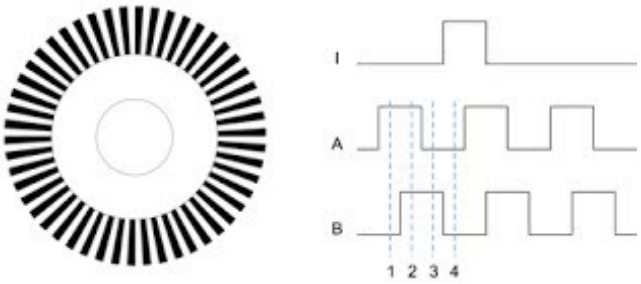

Рисунок 1: Инкрементальный энкодер

Оптический инкрементальный энкодер представляет собой тонкий диск с нанесенными на него чередующимися прозрачными и черными участками. Диск закреплен на валу двигателя, а на его краю размещается фото датчик. При вращении диска происходит последовательное перекрывание щели фото датчика. Получая такой сигнал, контроллер может определить скорость вращения диска и величину угла на который повернулся вал.

Помимо фото датчиков, в инкрементальном энкодере могут применяться датчики холла. В этом случае вместо диска с рисунком применяется магнит. Разрешающая способность у такого энкодера значительно ниже чем у оптического. Также возможна фиксация положения диска при помощи щеточной системы.

Для определения направления вращения диска в систему добавляется еще один фото датчик. Такой энкодер называется квадратурным. Второй датчик сдвигается относительно первого таким образом, чтобы можно было зафиксировать четыре состояния: первый открыт, второй закрыт; оба открыты; первый закрыт, второй открыт; оба закрыты (10-11-01-00). При этом канал одного датчика называется синусом (A) а канал другого косинусом (B). Таким образом, если при движении диска сначала открывается датчик A, а затем датчик B (т.е. последовательность 10-11) то это движение в одну сторону. Если же сначала B а затем A (01-11) то имеет место обратное вращение.

Помимо возможности определить направление вращения, квадратурный энкодер позволяет увеличить точность позиционирования вдвое. Недостатком инкрементальных энкодеров является тот факт, что после включения устройства невозможно определить положение вала двигателя без проведения дополнительной процедуры инциализации. В рамках этой процедуры происходит вращение двигателей до обнаружения специальной метки (referencemark). Нахождение метки отдельным фото датчиком означает что двигатель прибыл в начальное положение и готов к работе.

Рисунок 2: Абсолютный энкодер

Для устранения главного недостатка инкрементальных энкодеров - потеря позиции при выключении питания, был разработан абсолютный энкодер. Кодовый диск с метками в этом энкодере устроен несколько сложнее. Начиная от края диска, на нем размещаются несколько слоев меток. Каждый слой отвечает за одну позицию в бинарном выходном коде. Для снятия сигнала с каждого слоя, напротив него размещается свой отдельный фото датчик. При этом, в каждом фиксированном положении диска, на выходе имеется строго уникальный бинарный код.

Главный недостаток кодового диска, что при переходе между кодами, в которых сразу несколько бит переключаются, например 7-8 это 0111 — 1000, может случиться, что под одним датчиком чуть больше черного, а на другом — белого, и получится промежуточный код, например 0000, где старший (левый) бит еще не успел переключиться, а остальные уже переключились.

Часто, для кодирования угла на диске используется код Грея. Последовательность состояний датчика с таким кодом выглядит следующим образом: 000-001-011-010-110-111-101-100. В коде Грея, каждое последующее состояние отличается от предыдущего только на один бит, это помогает избежать проблемы описанной ранее. Применяются также и другие системы кодирования состояний. Например, на рисунке представлен диск энкодера с обычным двоичным кодом.

Следует отметить, что принципы заложенные в датчиках поворота применяются и в датчиках линейного перемещения. В частности, рисунок энкодера может наноситься не на диск а на подвижный элемент линейного актуатора. То же самое касается и потенциометров.

В нашем случае энкодер представляет собой простое устройство со светодиодом и фотоприемником. Фотоприемник срабатывает при засветке светодиодом, а это происходит, когда в дополнительном колесе появляется отверстие, таким способом можно посчитать скорость вращения колеса, зная количество отверстий в дополнительном колесе.

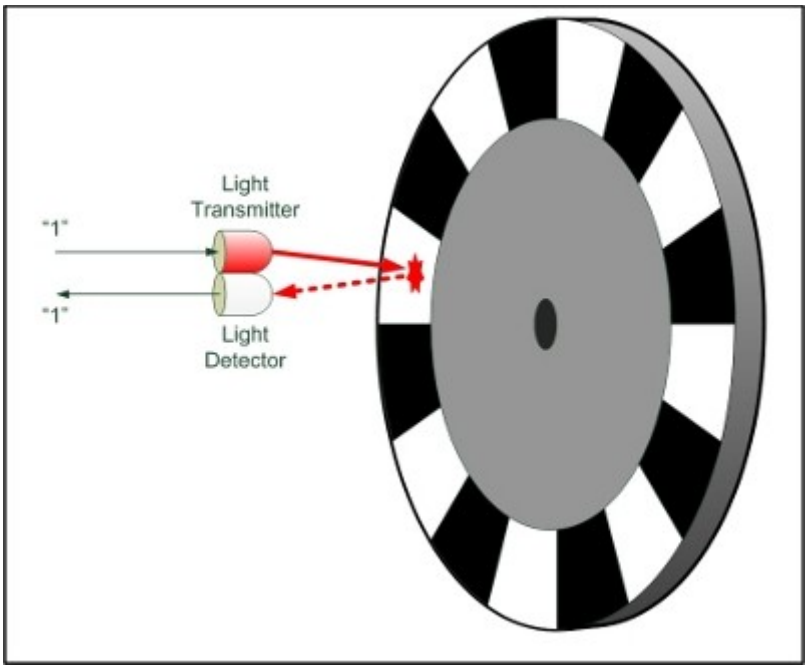

Рисунок 3: Энкодер, используемый в роботе

Пример программы, регулирующей вращение колес.

```
#include <TimerOne.h>
unsigned int counter=0;

int b1a = 6; // L9110 B-1A
int b1b = 9; // L9110 B-1B

void docount() // Считываем срабатывания энкодера
{
 counter++; // увеличиваем счетчик оборотов
}
```

```
void timerIsr()
{
 Timer1.detachInterrupt(); //останавливаем таймер
 Serial.print("Motor Speed: ");
 int rotation = (counter / 20); // делим число на количество отверстий в диске
 Serial.print(rotation,DEC);
 Serial.println(" Rotation per seconds");
 counter=0; // reset counter to zero
 Timer1.attachInterrupt( timerIsr ); //включаем таймер
}

void setup()
{
 Serial.begin(9600);

pinMode(b1a, OUTPUT);
pinMode(b1b, OUTPUT);

 Timer1.initialize(1000000); // запускаем таймер на 1 секунду
 attachInterrupt(0, docount, RISING); // Счетчик +1 при «1» на ножке
энкодера
 Timer1.attachInterrupt( timerIsr ); // включаем таймер
}

void loop()
{
 int potvalue = analogRead(1); // потенциометр подключаем к  Pin A1
 int motorspeed = map(potvalue, 0, 680, 255, 0);
 analogWrite(b1a, motorspeed); // устанавливаем скорость мотора  (0-255)
 digitalWrite(b1b, 1); // устанавливаем вращение мотора по часовой стрелке
}
```

Работа с энкодерами осуществляется через внешние прерывания, на плате ArduinoUno внешние прерывания присутствуют только на двух выводах, это D2 и D3.

attachInterrupt(interrupt, function, mode)

interrupt: номер прерывания (int)

function: функция, которую необходимо вызвать при возникновении прерывания; эта функция должна быть без параметров и не возвращать никаких значений. Такую функцию называют обработчиком прерывания.

mode: определяет условие, при котором должно срабатывать прерывание. Может принимать одно из четырех предопределенных значений:

LOW - прерывание будет срабатывать всякий раз, когда на выводе присутствует низкий уровень сигнала

CHANGE - прерывание будет срабатывать всякий раз, когда меняется состояние вывода

RISING - прерывание сработает, когда состояние вывода изменится с низкого уровня на высокий

FALLING - прерывание сработает, когда состояние вывода изменится с высокого уровня на низкий

HIGH - прерывание будет срабатывать всякий раз, когда на выводе присутствует высокий уровень сигнала.

Задача №1 Добавление обработки второго колеса

Запустите представленный код и разберитесь как он работает. Добавьте к нему код обрабатывающей вращение второго колеса.

Задача №2 Проезд по прямой

Робот должен проехать строго в одну сторону слева направо, траектория движения робота должна быть как можно ближе к прямой. Возможно, код потребует дополнительной настройки.

Задача №3 Модификация программы для езды по линии

Робот должен ездить по линии при помощи ИК датчиков. Код для работы с энкодером должен помочь роботу лучше удерживать траекторию.

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение энкодеров к плате. В программе уделите внимание подключению модулей. Посмотрите на плате номера портов и ножек и впишите соответствующие цифры в код программы.
3. Проверьте подключение энкодеров,- необходимо убедиться, что питание подключено правильно.
4. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.
5. Модифицируйте код, добавив к нему работу с энкодерами. Добейтесь того, что бы робот мог двигаться, используя энкодеры. Выполните задания 1, 2 и 3.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.
3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №21 «Подключение Bluetooth модуля для управления роботом»

Цель

Изучение функций и работы Bluetooth, подключение, разработка программ для управления движением робота на языке C/C++.

Краткие теоретические сведения

Нередко в проектах возникает необходимость дистанционного управления или передачи данных с телефона или другого устройства. Одним из самых популярных и удобных способов является обмен данных через Bluetooth. Для связи платы Arduino и компьютера используется интерфейс UART (Serial). Так как любая плата Arduino имеет хотя бы один последовательный порт UART, для подключения Bluetooth модуля не требуются специализированные библиотеки и схемы.

Модуль HP-05

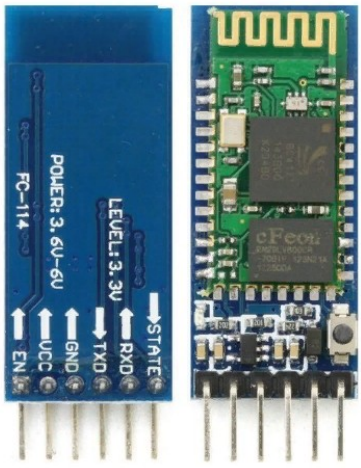

Рисунок 1: Вид модуля Bluetooth

Основные характеристики HC-05:

1. рабочие частоты 2,4 – 2,48 ГГц;
2. мощность передачи 0,25 – 2,5мВт;
3. дальность 10 м;

4. максимальная скорость обмена данными 115200 бод;
5. питание 3,3В;
6. ток 30-40 мА;
7. рабочие температуры от -25С до 75С.

Таблица 1: Схема подключения

Arduino Leonardo	Bluetooth HP-05
Pin 6 (TX)	RXD
Pin 9 (RX)	TXD
GND	GND
5V	VCC

Будьте внимательны, подключать нужно TX → RXD ,RX → TXD.

Пример программы для моргания светодиодом с использованием Bluetooth.

```
#include <SoftwareSerial.h> //Подключаем библиотеку
SoftwareSerial Serial_1(9, 6); // RX, TX  Настраиваем порты вывода

int val;
int LED = 13;
void setup()
{
  Serial_1.begin(9600); //Устанавливаем скорость передачи
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH);
}
void loop()
{
  if (Serial_1.available()) //Смотри что передалась информация
  {
    val = Serial_1.read(); //Читаем по 1 биту

    // При символе "1" включаем светодиод
    if (val == '1')
    {
      digitalWrite(LED, HIGH);
    }
    // При символе "0" выключаем светодиод
    if ( val == '0')
    {
      digitalWrite(LED, LOW);
    }
  }
}
```

Рисунок 2: Пример программы для работы с Bluetooth

Теперь напишем программу для управления роботом через Bluetooth.

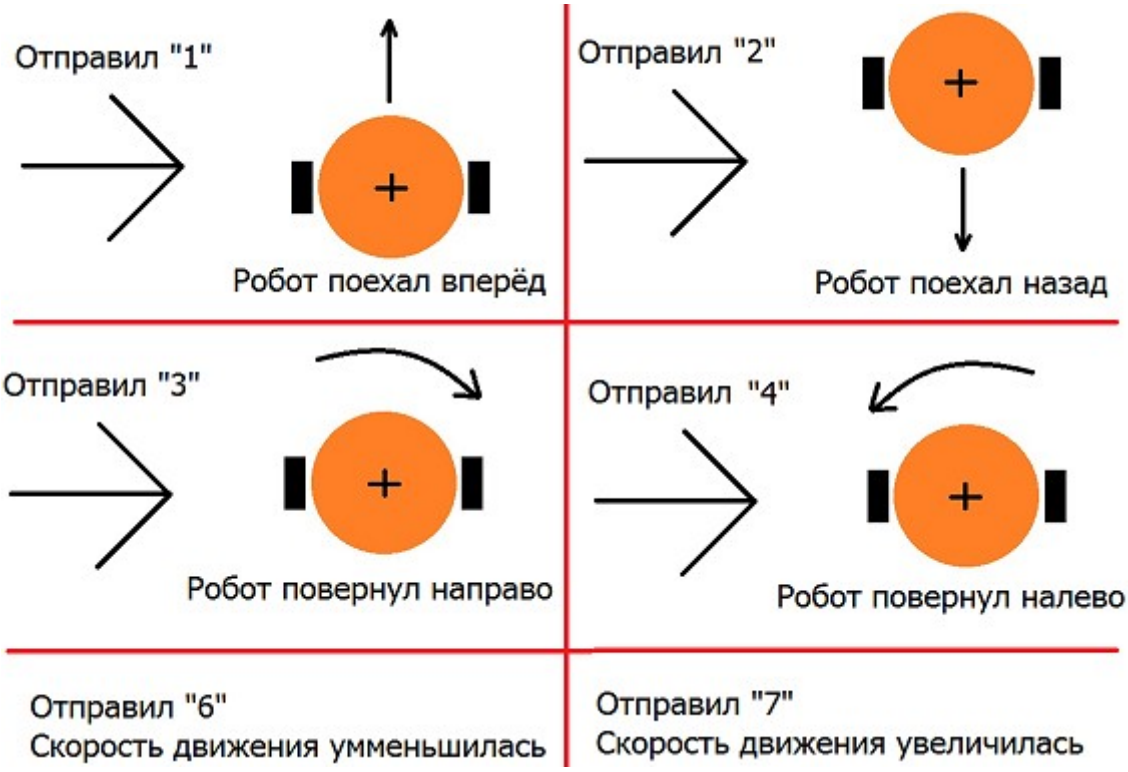

Рисунок 3: Алгоритм движения

Код программы:

```
#include <Servo.h>
#include <SoftwareSerial.h> //Подключаем библиотеку
// Инициализируем пины ардуино для работы с модулем 9 - RX, 6 - TX
SoftwareSerial Serial_1(9, 6); // RX, TX (14 (MISO), 16 (MOSI))

Servo grab;

const uint8_t M1A = 12;
const uint8_t M1B = 4;
const uint8_t M1S = 5;

const uint8_t M2A = 10;
const uint8_t M2B = 8;
const uint8_t M2S = 13;

void setup()
{
  pinMode(M1A, OUTPUT);
  pinMode(M1B, OUTPUT);
  pinMode(M1S, OUTPUT);

  pinMode(M2A, OUTPUT);
  pinMode(M2B, OUTPUT);
  pinMode(M2S, OUTPUT);

  // устанавливаем скорость передачи данных для последовательного порта, созданного
  // библиотекой SoftwareSerial
  Serial_1.begin(9600);             // Устанавливаем скорость передачи 9600 бод
  Serial_1.println("I'm alive!");
}

void loop()
{
  listen();
}
```

Рисунок 4: Управление роботом через Bluetooth

```
#include "movement.h"

int sp = 40;

void listen() // Функция ждёт передачи информации по bluetooth
  if(Serial_1.available()) // Если в буфер что то есть, тогда true (Если что то приняли, то тога продолжаем)
  {
    Serial_1.print("got: ");
    switch(Serial_1.read())  // Прочитать 1 бит информации
    {
      case '0': Stop();     // Передаём 0 - выполняется функция Stop()
        break;
      case '1': Forward();  // Передаём 1 - выполняется функция Forward()
        break;
      case '2': Back();     // Передаём 2 - выполняется функция Back()
        break;
      case '3': Left();     // Передаём 3 - выполняется функция Left()
        break;
      case '4': Right();    // Передаём 4 - выполняется функция Right()
        break;
      case '5': Grab();     // Передаём 5 - выполняется функция Grab()
        break;
      case '6':             // Передаём 6 - уменьшаем скорость движения
        if(sp >= 10)
          sp-=10;
        Serial_1.print("Speed = ");
        Serial_1.println(sp);
        break;
      case '7':
        if(sp <= 240)        // Передаём 7 - увеличиваем скорость движения
          sp+=10;
        Serial_1.print("Speed = ");
        Serial_1.println(sp);
    }
  }
}
```

Рисунок 5: Управление роботом через Bluetooth

Отдельно прописываем как двигается робот в наших функциях

Подключение к bluetooth со смартфона:

1. включаем Bluetooth на телефоне и ищем новые устройства;
2. находим в списке расстройств "HC-06" и подключаемся к нему;
3. телефон спросит пин-код, необходимо ввести "1234" или "0000";
4. устройство подключено;
5. теперь нужно скачать bluetooth terminal на ваш телефон.

Рассмотрим на примере платформы Android.

Скачиваем приложение в Android Play Market.

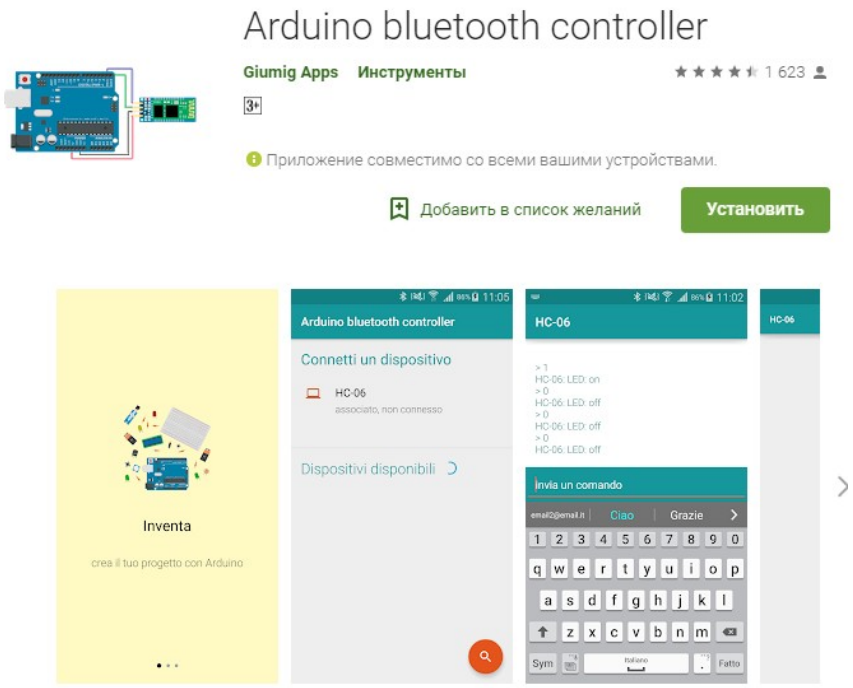

Рисунок 6: Вид приложения управления по Bluetooth

Включаем режим управления.

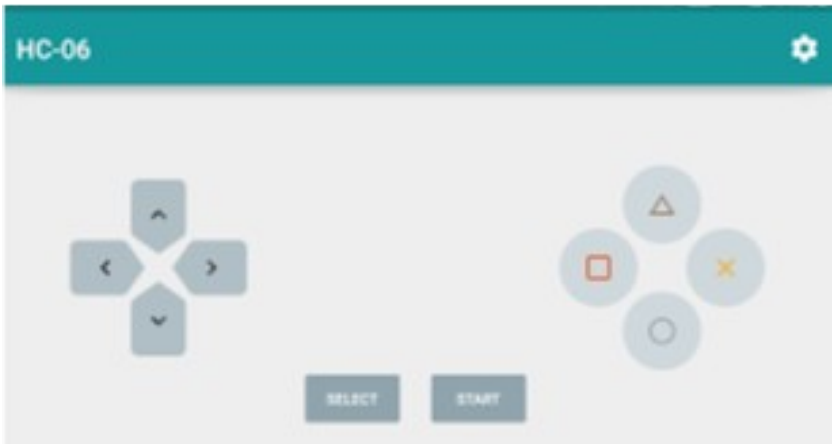

Рисунок 7: Вид приложения в режиме пульта

В настройках устанавливаем код для каждой кнопки

Например ВПЕРЁД = 1, тогда при нажатии на кнопку мы передаём и принимаем единичку, а в программе выполнится функция Forward().

Задание №1

Заставить робота двигаться с управлением через Bluetooth

Задание №2

Настроить остальные кнопки в контроллере:

1. добавить алгоритм для «Разворота на 180»;
2. добавить алгоритм для случайного движения или вращения на месте для Робота;
3. робот переключается в режим терминатора и едет на ближайшее препятствие.

Порядок выполнения работы

1. Запустите Arduino IDE.
2. Выполните подключение Bluetooth модуля к плате. В программе уделите внимание подключению линий RX и TX. Будьте внимательны, подключать нужно TX -> RXD ,RX -> TXD. Посмотрите на плате номера портов и ножек и впишите соответствующие цифры в код программы.
3. Проверьте подключение модуля,- необходимо убедиться, что питание подключено правильно.
4. Включите питание и перепишите код программ из примера. Попробуйте новые функции в работе.
5. Скачайте и установите приложение на телефон.
6. Модифицируйте код, добавив к нему работу с Bluetooth модулем. Выполните задания 1 и 2.

Содержание отчета по работе

Отчет должен содержать полный текст программы с подробным описанием каждой строки кода.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере ArduinoIDE.
2. Открыть программу из раздела «Краткие теоретические сведения», отобразив экран своего монитора с помощью проектора на большом экране.

3. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
4. Запустить загрузить программу в плату управления робота и продемонстрировать работу программы.
5. Озвучить задание лабораторной работы.
6. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу робота – в случае ошибок, исправить программу вместе с учеником.

Лабораторная работа №22 «Измерение электрических величин»

Цель

Изучение основных способов измерения электрических величин.

Краткие теоретические сведения

В зависимости от способа обработки экспериментальных данных для нахождения результата различают прямые, косвенные, совместные и совокупные измерения.

При прямом измерении искомое значение величины находят непосредственно из опытных данных в результате выполнения измерения; например, измерение амперметром тока в ветви цепи.

При косвенном измерении искомое значение величины находят на основании известной зависимости между измеряемой величиной и величинами, подвергаемыми прямым измерениям; например, определение сопротивления R резистора из уравнения $R = U/I$, в которое подставляют измеренное значение напряжения U на зажимах резистора и протекающего через него постоянного тока I.

Совместные измерения - одновременные измерения нескольких неодноименных величин для нахождения зависимости между ними; например, определение зависимости сопротивления резистора от температуры по формуле $R_t = R_0 (1 + at + bt^2)$ посредством измерения сопротивления резистора R_t при трех различных температурах t. Составив систему из трех уравнений, находят параметры R_0, a и b зависимости сопротивления резистора от температуры.

Совокупные измерения одновременные измерения нескольких одноименных величин, при которых искомые значения величин находят решением системы уравнений, составленных из результатов прямых измерений различных сочетаний этих величин; например, определение сопротивлений

резисторов, соединенных треугольником, посредством измерения сопротивлений между различными вершинами треугольника. По результатам трех измерений по известным соотношениям определяют сопротивления резисторов треугольника.

В данной лабораторной работе будет рассмотрен косвенный метод измерения сопротивления участка цепи. Исследуемая цепь представлена на рисунке 1.

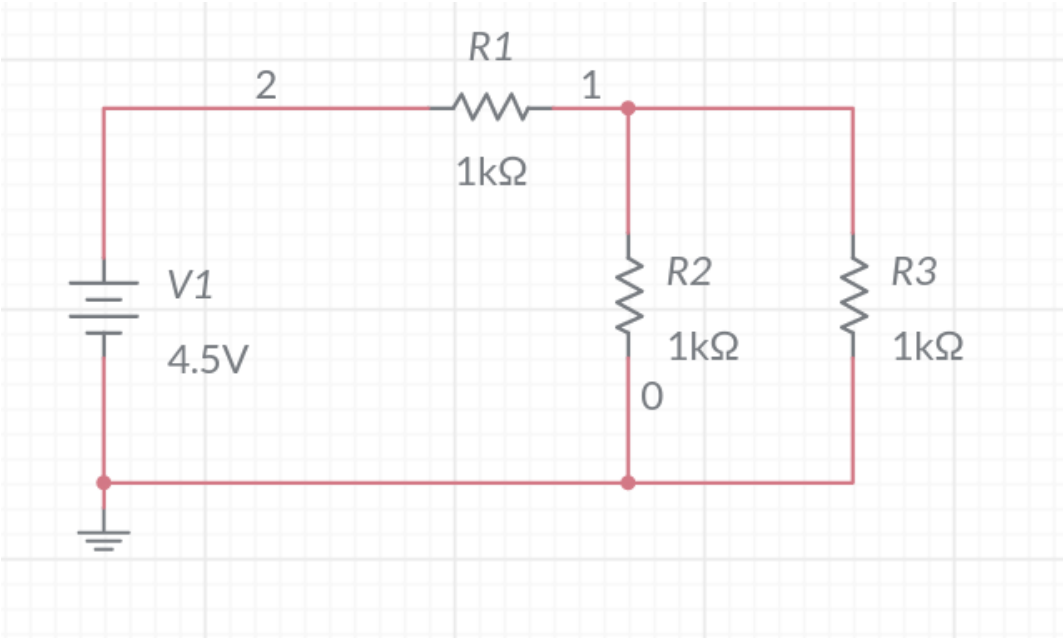

Рисунок 1: Исследуемая цепь

Multisim Online

Для самостоятельной сборки цепи необходимо воспользоваться Online симулятором программы Multisim, доступным по ссылке https://www.multisim.com.

Задание №1 Расчет параметров

Для теоретического расчета сопротивления участка цепи с резисторами R_1, R_2, R_3 необходимо воспользоваться формула расчета сопротивления для параллельного и последовательного участков:

Для параллельного:

$1 / R_{общ} = 1 / R_1 + 1 / R_2$

Для последовательного

$R_{общ} = R_1 + R_2$

Для данной цепи эквивалентное сопротивление может быть рассчитано по формуле:

$R_{экв} = R_1 + R_2 * R_3 / (R_2 + R_3)$.

Задание №2 Сборка цепи в Multisim Online

Для сборки цепи на рабочем поле среды Multisim, необходимо воспользоваться библиотекой компонентов, расположенной слева от рабочего поля.

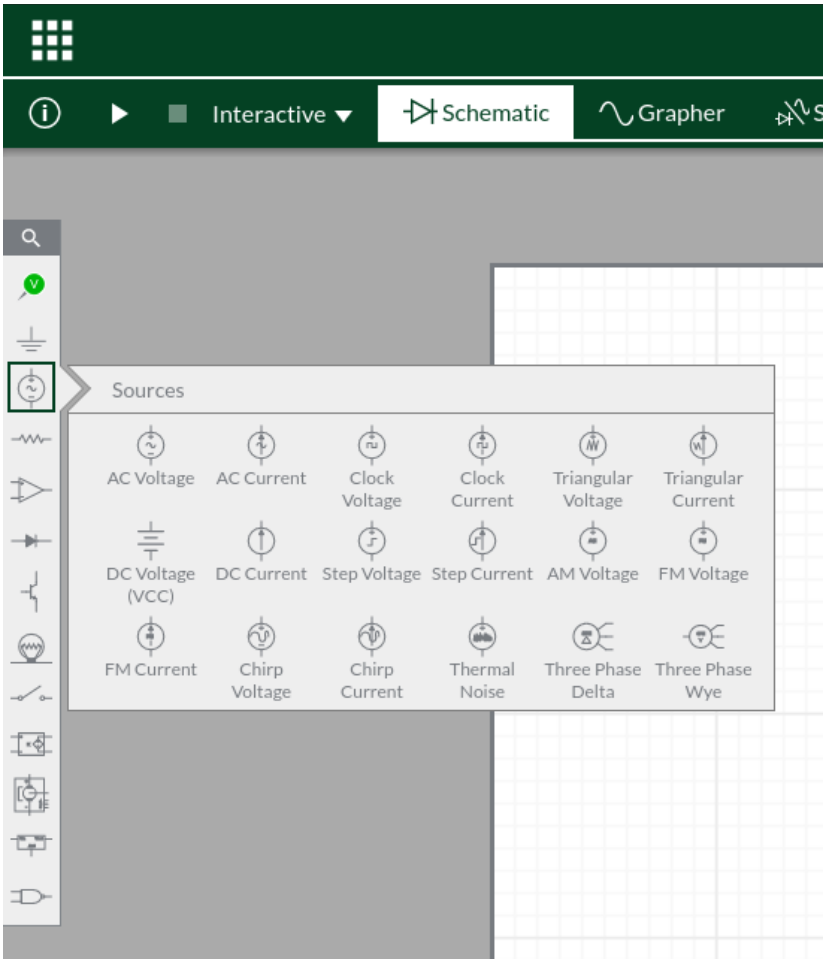

Рисунок 2: Библиотека компонентов

Для сборки цепи представленной на рисунке 1 необходимо поместить на рабочее поле следующие элементы:

- Ground (земля) — 1 шт;
- DC Voltage (VCC) (источник ЭДС) — 1 шт;
- Resistor (резистор) — 3 шт.

Далее требуется соединить все элементы проводниками (проводники считаются идеальными, т. е. не имеют внутреннего сопротивления) и выставить номиналы элементов:

- $E_{ист}$ = 4,5 В;
- R1 = R2 = R3 = 1кОм.

Задание №3 Измерение сопротивления

Для того, чтобы произвести косвенное измерение сопротивления участка цепи с тремя резисторами, необходимо добавить в цепь измерительный прибор: амперметр (пробник). Пробник находится также слева в библиотеке компонентов.

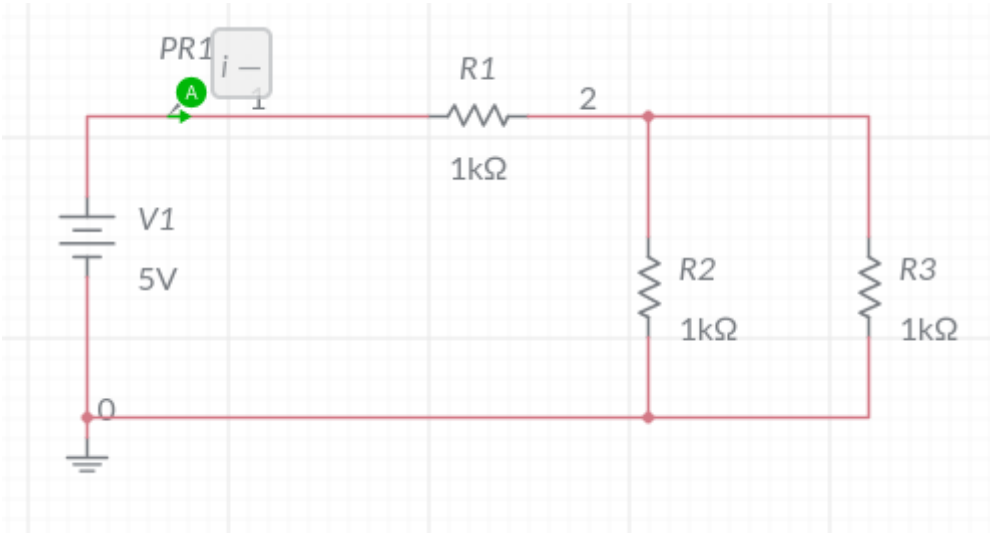

Рисунок 3: Цепь с установленным пробником тока

Далее необходимо запустить моделирование цепи, путем активации кнопки с изображением треугольника (Play) в верхней левой части окна.

После запуска моделирования пробник отобразит значение силы тока, протекающей в ветви.

Для получения значения сопротивления участка цепи с тремя резисторами необходимо воспользоваться законом Ома:

$R = U / I$, где $U = E_{ист}$, I — измеренное значение силы тока.

Сравните значение эквивалентного сопротивления полученного в задании 2 с полученным экспериментальным значением.

Порядок выполнение работы

1. Запустить симулятор Multisim Online.
2. Произвести расчеты в соответствии с заданием 1.
3. Произвести измерения в соответствии с заданиями 2 и 3.
4. Сравнить результаты расчетов и измерений.

Содержание отчета по работе

Отчет должен содержать снимки экрана выполнения заданий 2 и 3 и расчетные формулы из заданий 1 и 3.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере Multisim Online.
2. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
3. Запустить симулятор и продемонстрировать работу программы.
4. Озвучить задание лабораторной работы.
5. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить вместе с учеником.

Лабораторная работа №23 «Делители напряжения и тока»

Цель

Изучение принципов работы делителя напряжения и тока в цепях постоянного тока.

Краткие теоретические сведения

Делитель напряжения — это простая схема, которая позволяет получить из высокого напряжения пониженное напряжение.

Используя только два резистора и входное напряжение, можно получить выходное напряжение, составляющее определенную часть от входного.

Схема и формула представлены ниже.

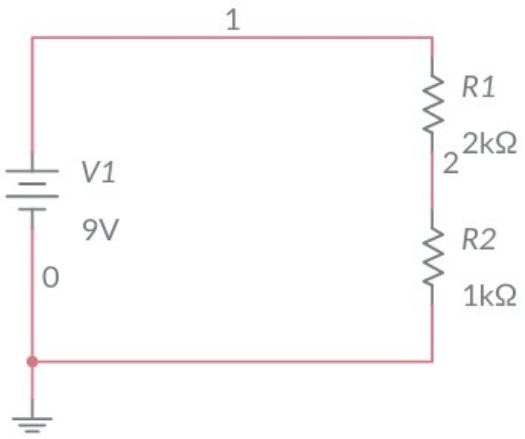

Рисунок 1: Схема делителя напряжения

$U_{вых} = U_{вх} * R_2 / (R_1 + R_2)$

Делитель тока — это схема, позволяющая разделить ток на две части, чтобы в дальнейшем использовать одну из них. Схема применяется, когда устройство не работает с большим током и необходимо отделить его меньшее количество, необходимое для использования аппаратуры.

В основе принципа действия, лежит первый закон Кирхгофа: сумма втекающих токов в узле равна сумме вытекающих.

Схема и формулы расчета представлены ниже.

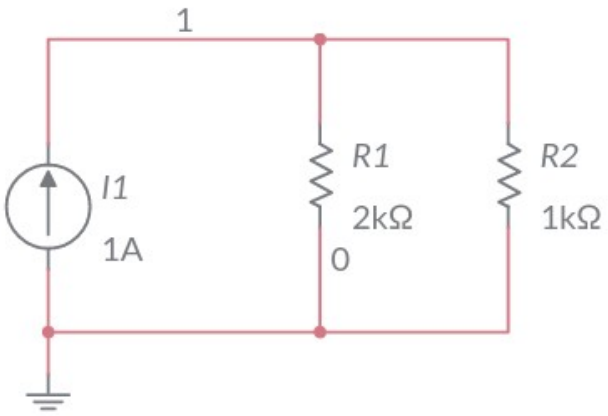

Рисунок 2: Схема делителя тока

$I_1 = I * R_2 / (R_1 + R_2)$

$I_2 = I * R_1 / (R_1 + R_2)$

Задание №1 Делитель напряжениям

Собрать на рабочем поле Multisim Online цепь в соответствии с рисунком 3.

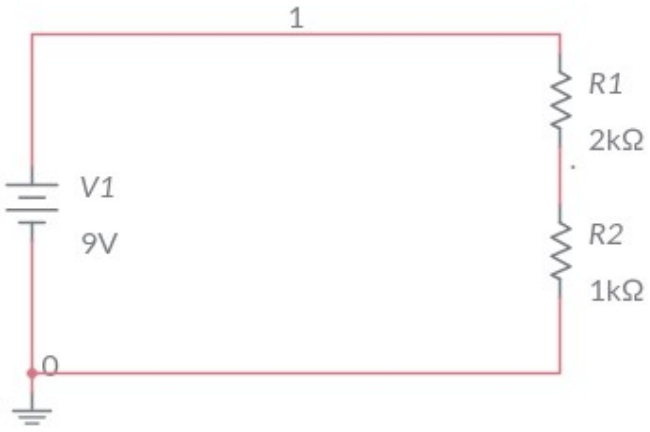

Рисунок 3: Цепь делителя напряжения

Установить следующие номиналы элементов:

- $E_{ист}$ = 9 В;
- R_1 = 2 кОм;
- R_2 = 1 кОм.

Установить пробники для измерения напряжения на резисторах.

Пробник для измерения напряжения по умолчанию использует заземленный узел как опорную точку для измерения разницы потенциалов. Для задания точки, относительно которой должно производиться измерение напряжения, необходимо активировать кнопку V- на пробнике и установить вспомогательный пробник в нужную точку цепи (рис. 4).

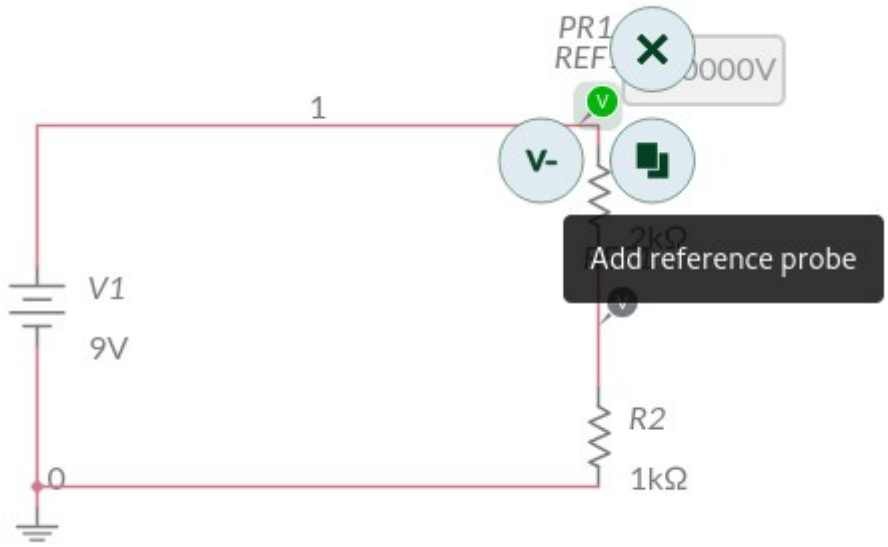

Рисунок 4: Установка пробника для измерения напряжения

Произведите измерения напряжения на резисторах R_1 и R_2.

Произведите расчет напряжения на резисторах R_1 и R_2 по формулам из теоретического раздела.

Сравните полученные значения.

Задание №2 Делитель тока

Собрать на рабочем поле Multisim Online цепь в соответствии с рисунком 5.

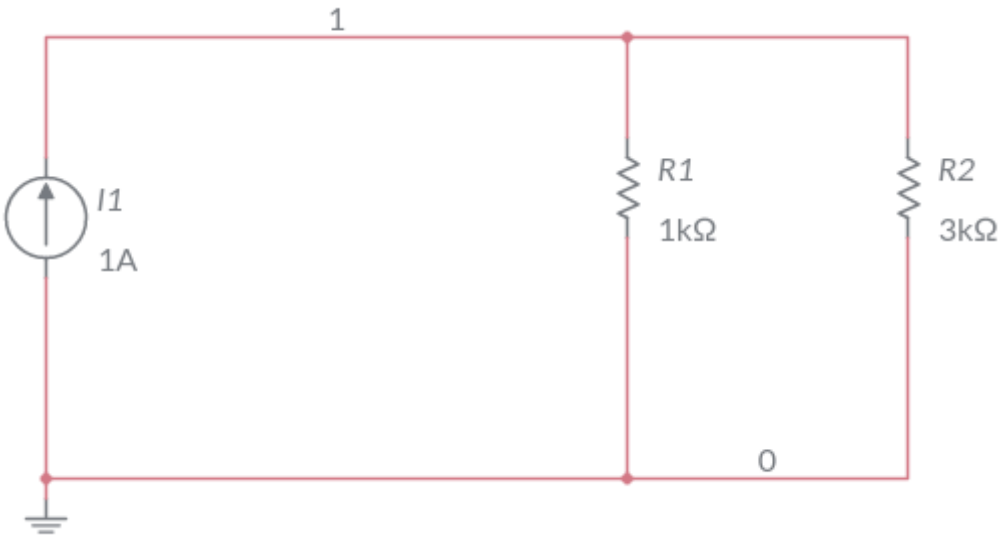

Рисунок 5: Цепь делителя тока

Установить следующие номиналы элементов:

- $J_{ист}$ = 1А;
- R_1 = 1 кОм;
- R_2 = 3 кОм.

Установить пробники для измерения силы тока на ветвях с резисторами.

Произведите измерения силы тока на ветвях с резисторами R_1 и R_2.

Произведите расчет силы тока в ветвях с резисторами R_1 и R_2 по формулам из теоретического раздела.

Сравните полученные значения.

Порядок выполнение работы

1. Запустить симулятор Multisim Online.
2. Произвести расчеты в соответствии с заданием 1 и 2.
3. Произвести измерения в соответствии с заданиями 1 и 2.

4. Сравнить результаты расчетов и измерений.

Содержание отчета по работе

Отчет должен содержать снимки экрана выполнения заданий и расчетные формулы.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере Multisim Online.
2. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
3. Запустить симулятор и продемонстрировать работу программы.
4. Озвучить задание лабораторной работы.
5. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить вместе с учеником.

Лабораторная работа №24 «Реактивные элементы в цепях постоянного тока»

Цель

Изучить работу конденсатора и катушки индуктивности в цепях постоянного тока в установившимся режиме.

Краткие теоретические сведения

Катушка индуктивности

Индуктивность — это свойство электронного компонента противодействовать изменениям тока, протекающего через данный компонент.

Индуктивностью обладают компоненты, которые называются катушками индуктивности, соленоидами или дросселями. Если ток, протекающий в катушке индуктивности, изменяется, свойство индуктивности противодействует такому изменению тока. Если ток увеличивается, катушка индуктивности препятствует росту тока. Если ток уменьшается, катушка индуктивности снова пытается сохранить ток без изменения. Эффект индуктивности заметен в первую очередь в тех схемах, где используется переменный ток. Противодействие переменному току, оказываемое катушкой индуктивности, называется индуктивным сопротивлением. Подобно сопротивлению резистора индуктивное сопротивление оказывает фиксированное противодействие, которое контролирует уровень тока в схеме.

В схемах постоянного тока, в которых ток имеет фиксированное значение, определяемое сопротивлениями и напряжениями, катушки индуктивности обычно имеют лишь незначительный эффект или вообще не имеют никакого эффекта, т. е. их свойства приравниваются к свойствам обычного проводника.

Конденсатор

Поскольку между обкладками конденсатора находится диэлектрик, то электрический ток от одной пластинки к другой протекать не может, следовательно, образуется разрыв электрической цепи для постоянного и для переменного тока. От сюда следует, что конденсатор не пропускает постоянный ток. Переменный ток он также не пропускает, однако переменный ток постоянно перезаряжает накопитель, что создает картину того, что переменный тока проходит сквозь обкладки конденсатора.

Если к обкладкам разряженного конденсатора приложить постоянное напряжение, то в цепи начнет протекать электрический ток. По мере его заряда ток будет снижаться и при равности напряжений на пластинках и источника питания, ток перестанет протекать – образуется разрыв электрической цепи.

Задание №1 Катушка индуктивности

Собрать на рабочем поле Multisim Online схему в соответствии с рисунком 1.

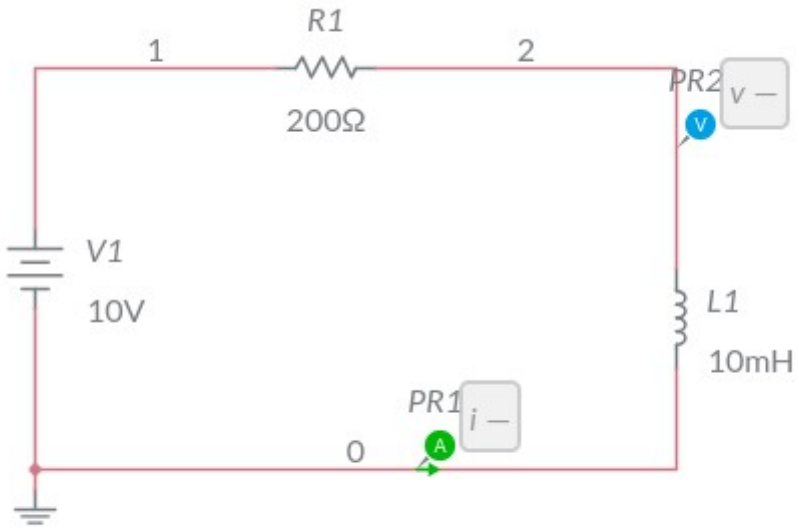

Рисунок 1: Цепь с катушкой индуктивности

Задать ЭДС источника напряжения $E_{ист}$ = 10 В. Значения сопротивления нагрузки и индуктивности катушки задать: R = 200 Ом, L = 10 мГн.

Исследовать поведение характеристик индуктивности на постоянном токе. Схема для измерений представлена на рисунке 1. Для индуктивности: убедиться, что разность потенциалов равна нулю при наличии тока, т.е. сопротивление индуктивности равно "0".

Задание №2 Конденсатор

Собрать на рабочем поле Multisim Online схему в соответствии с рисунком 2.

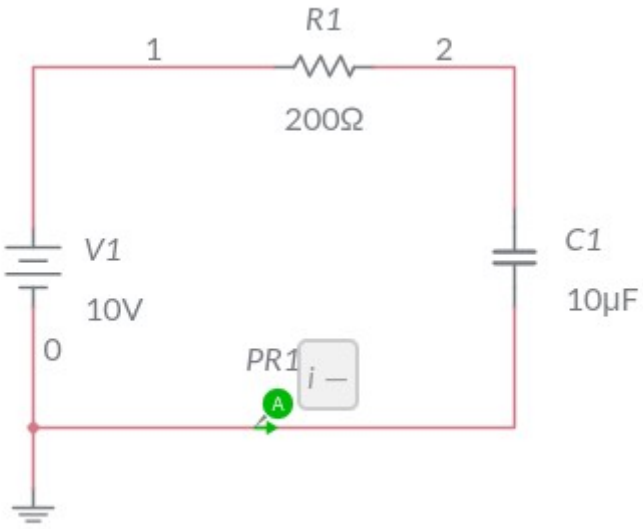

Рисунок 2: Цепь с конденсатором

Задать ЭДС источника напряжения $E_{ист}$ = 10 В. Значения сопротивления нагрузки и индуктивности катушки задать: R = 200 Ом, C = 10 мкФ.

Исследовать поведение характеристик индуктивности на постоянном токе. Схема для измерений представлена на рисунке 1. Для индуктивности: убедиться, что разность потенциалов равна нулю при наличии тока, т.е. сопротивление индуктивности равно "0".

Порядок выполнение работы

1. Запустить симулятор Multisim Online.
2. Произвести расчеты в соответствии с заданием 1 и 2.
3. Произвести измерения в соответствии с заданиями 1 и 2.

4. Сравнить результаты расчетов и измерений.

Содержание отчета по работе

Отчет должен содержать снимки экрана выполнения заданий и расчетные формулы.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере Multisim Online.
2. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
3. Запустить симулятор и продемонстрировать работу программы.
4. Озвучить задание лабораторной работы.
5. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить вместе с учеником.

Лабораторная работа №25 «Метод контурных токов»

Цель

Изучить метод контурных токов для расчета электрических цепей

Краткие теоретические сведения

Первый закон Кирхгофа

Формулировка 1: Сумма всех токов, втекающих в узел, равна сумме всех токов, вытекающих из узла.

Формулировка 2: Алгебраическая сумма всех токов в узле равна нулю.

Поясним первый закон Кирхгофа на примере рисунка 1.

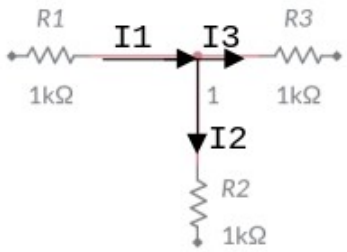

Рисунок 1: Узел электрической цепи

Здесь ток I_1- ток, втекающий в узел , а токи I_2 и I_3 — токи, вытекающие из узла. Тогда применяя первую формулировку , можно записать: $I_1 = I_2 + I_3$.

Что бы подтвердить справедливость второй формулировки, перенесем токи I_2 и I_3 в левую часть выражения, тем самым получим: $I_1 - I_2 - I_3 = 0$.

Знаки «минус» и означают, что токи вытекают из узла.

Знаки для втекающих и вытекающих токов можно брать произвольно, однако в основном всегда втекающие токи берут со знаком «+», а вытекающие со знаком «-».

Второй закон Кирхгофа

Формулировка: Алгебраическая сумма ЭДС, действующих в замкнутом контуре, равна алгебраической сумме падений напряжения на всех резистивных элементах в этом контуре.

Здесь термин «алгебраическая сумма» означает, что как величина ЭДС так и величина падения напряжения на элементах может быть как со знаком «+» так и со знаком «-». При этом определить знак можно по следующему алгоритму:

1. Выбираем направление обхода контура (два варианта либо по часовой, либо против).
2. Произвольно выбираем направление токов через элементы цепи.
3. Расставляем знаки для ЭДС и напряжений, падающих на элементах по правилам:

- ЭДС, создающие ток в контуре, направление которого совпадает с направление обхода контура записываются со знаком «+», в противном случае ЭДС записываются со знаком «-».
- напряжения, падающие на элементах цепи записываются со знаком «+», если ток, протекающий через эти элементы совпадает по направлению с обходом контура, в противном случае напряжения записываются со знаком «-».

Например, рассмотрим цепь, представленную на рисунке , и запишем выражение согласно второму закону Кирхгофа, обходя контур по часовой стрелке, и выбрав направление токов через резисторы, как показано на рисунке 2.

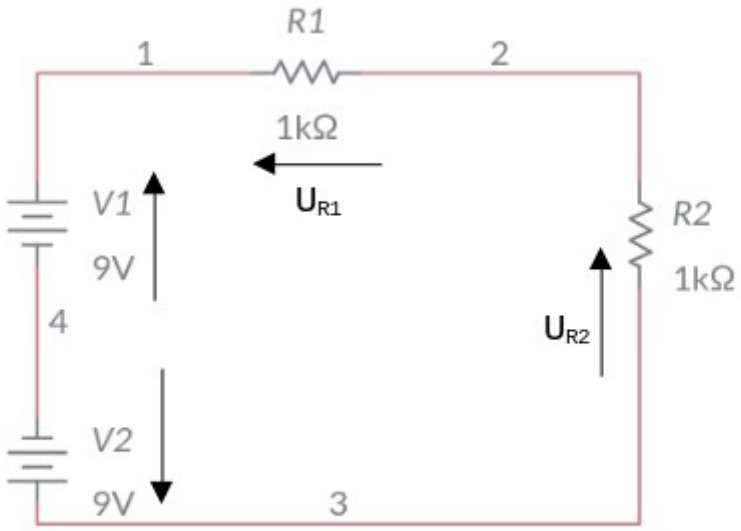

Рисунок 2: Электрическая цепь, для пояснения второго закона Кирхгофа

Формула описывающая второй закон Кирхгофа для цепи представленной на рисунке 2: E_1- E_2 = -U_{R1} — U_{R2}.

Метод контурных токов

Один из методов анализа электрической цепи является метод контурных токов. Основой для него служит второй закон Кирхгофа. Главное его преимущество это уменьшение количества уравнений до $N_в$ – $N_у$ +1, $N_в$ — количество ветвей, а $N_у$ — количество узлов. На практике такое уменьшение существенно упрощает расчет.

Контурный ток - это величина, которая одинакова во всех ветвях данного контура. Обычно в расчетах они обозначаются двойными индексами, например I_{11}, I_{22} и тд.

Действительный ток в определенной ветви определяется алгебраической суммой контурных токов, в которую эта ветвь входит. Нахождение действительных токов и есть первоочередная задача метода контурных токов.

Контурная ЭДС - это сумма всех ЭДС входящих в этот контур.

Собственным сопротивлением контура называется сумма сопротивлений всех ветвей, которые в него входят.

Общим сопротивлением контура называется сопротивление ветви, смежное двум контурам.

План составления уравнений:

1. Выбор направления действительных токов.
2. Выбор независимых контуров и направления контурных токов в них.
3. Определение собственных и общих сопротивлений контуров
4. Составление уравнений и нахождение контурных токов
5. Нахождение действительных токов

Рассмотрим пример.

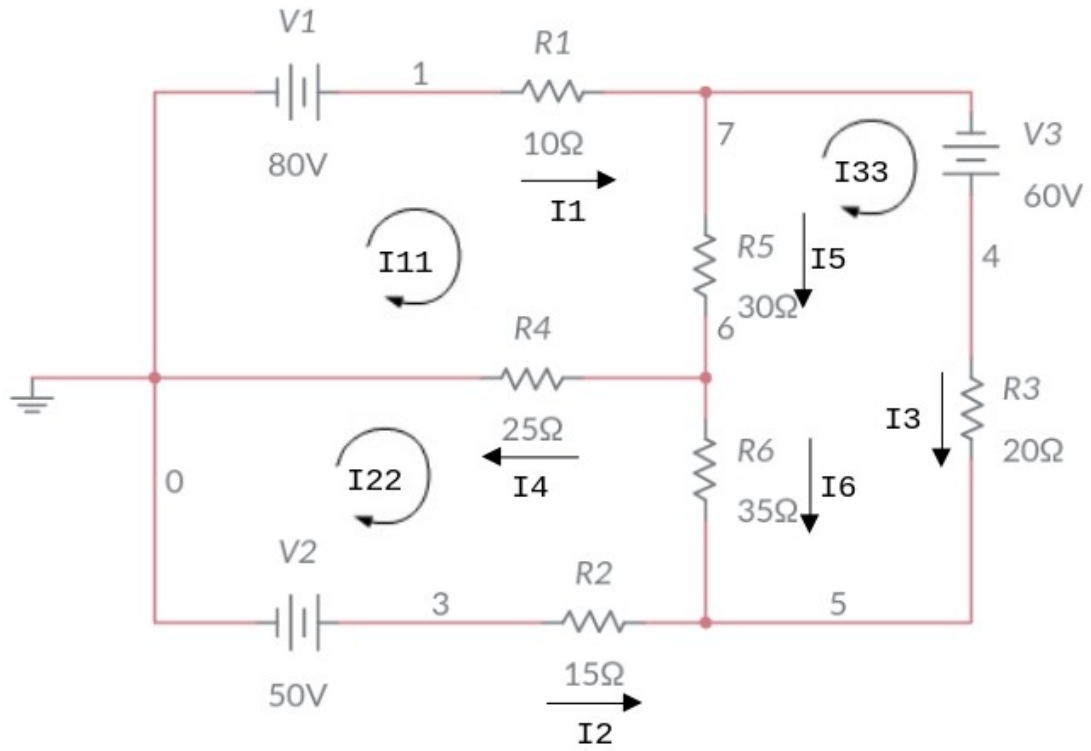

Рисунок 3: Схема

1. Произвольно выбираем направления действительных токов I_1-I_6.
2. Выделяем три контура, а затем указываем направление контурных токов I_{11},I_{22},I_{33}. Направление можно выбрать по часовой стрелке.

3. Определяем собственные сопротивления контуров. Для этого складываем сопротивления в каждом контуре.

   $R_{11}=R_1+R_4+R_5=10+25+30=$ 65 Ом

   $R_{22}=R_2+R_4+R_6=15+25+35=$ 75 Ом

   $R_{33}=R_3+R_5+R_6=20+30+35=$ 85 Ом

   Затем определяем общие сопротивления, общие сопротивления легко обнаружить, они принадлежат сразу нескольким контурам, например сопротивление R4 принадлежит контуру 1 и контуру 2. Поэтому для удобства обозначим такие сопротивления номерами контуров к которым они принадлежат.

   $R_{12}=R_{21}=R_4=$ 25 Ом

   $R_{23}=R_{32}=R_6=$ 35 Ом

   $R_{31}=R_{13}=R_5=$ 30 Ом

4. Далее следует составление системы уравнений контурных токов. В левой части уравнений входят падения напряжений в контуре, а в правой ЭДС источников данного контура.

   Так как контура три, следовательно, система будет состоять из трех уравнений. Для первого контура уравнение будет выглядеть следующим образом:

   $I_{11} * R_{11} - I_{22} * R_{21} - I_{33} * R_{31} = E_1$

   Ток первого контура I_{11}, умножаем на собственное сопротивление R_{11} этого же контура, а затем вычитаем ток I_{22}, умноженный на общее сопротивление первого и второго контуров R_{21} и ток I_{33}, умноженный на общее сопротивление первого и третьего контура R_{31}. Данное выражение будет равняться ЭДС E1 этого контура. Значение ЭДС берем со знаком плюс, так как направление обхода (по часовой стрелке) совпадает с направление ЭДС, в противном случае нужно было бы брать со знаком минус.

   Те же действия проделываем с двумя другими контурами и в итоге получаем систему:

   $I_{11} * R_{11} - I_{22} * R_{21} - I_{33} * R_{31} = E_1$

   $-I_{11} * R_{12} + I_{22} * R_{22} - I_{33} * R_{32} = -E_2$

   $-I_{11} * R_{13} - I_{22} * R_{23} + I_{33} * R_{33} = E_3$

В полученную систему подставляем уже известные значения сопротивлений и решаем её любым известным способом.

$I_{11} = 2,726$

$I_{22} = 1,264$

$I_{33} = 2,189$

5. Последним этапом находим действительные токи, для этого нужно записать для них выражения.

   Контурный ток равен действительному току, который принадлежит только этому контуру. То есть другими словами, если ток протекает только в одном контуре, то он равен контурному.

   $I_1 = I_{11} = 2,726$

   Необходимо учитывать направление обхода, например, ток I_2 не совпадает с направлением, поэтому он будет со знаком минус.

   $-I_2 = -I_{22} = -1,264$

   $I_3 = I_{33} = 2,189$

   Токи, протекающие через общие сопротивления определятся как алгебраическая сумма контурных, учитывая направление обхода.

   Через резистор R_4 протекает ток I_4, его направление совпадает с направлением обхода первого контура и противоположно направлению второго контура. Следовательно, для него выражение будет выглядеть:

   $I_4 = I_{11} - I_{22} = 2,726 - 1,264 = 1,462$

   Для остальных:

   $I_5 = I_{11} - I_{33} = 0,537$

   $I_6 = I_{33} - I_{22} = 0,925$

Задание

Найти токи во всех ветвях с помощью метода контурных токов.

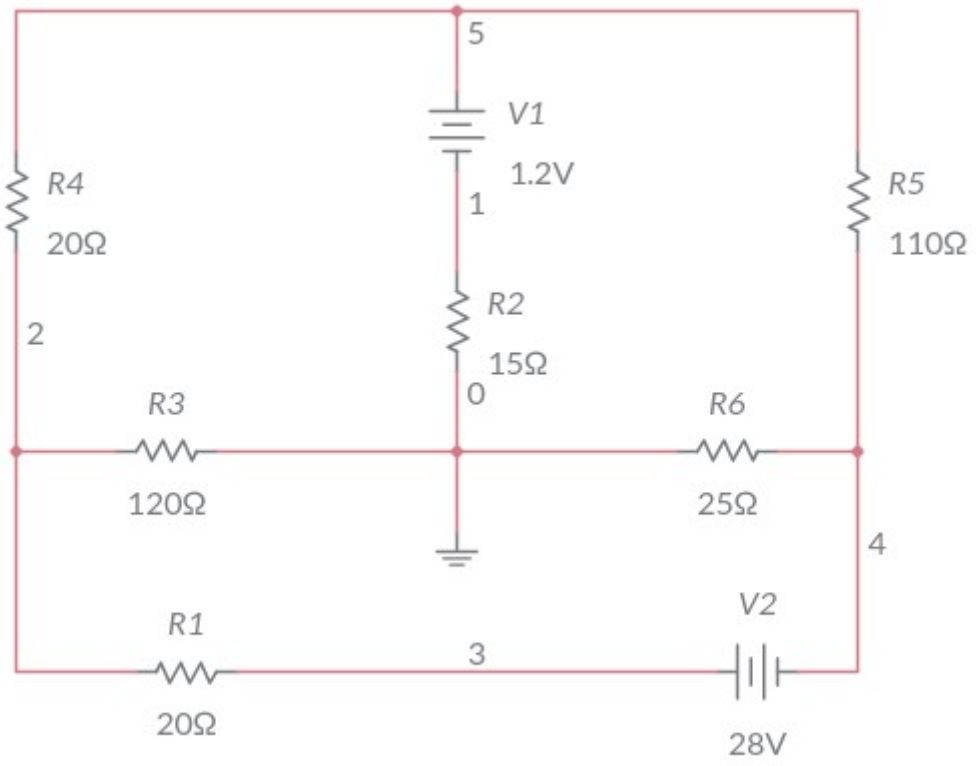

Рисунок 4: Схема для расчета

Номиналы элементов представлены в таблице:

Таблица 1

E1, В	E2, В	R1, Ом	R2, Ом	R3, Ом	R4, Ом	R5, Ом	R6, Ом
1,2	28	20	15	120	20	110	25

Порядок выполнение работы

1. Запустить симулятор Multisim Online.
2. Произвести расчеты в соответствии с заданием.
3. Произвести измерения в соответствии с заданиями.
4. Сравнить результаты расчетов и измерений.

Содержание отчета по работе

Отчет должен содержать снимки экрана выполнения заданий и расчетные формулы.

Инструкция по организации работы и проверке работы для преподавателя

Работа преподавателя организуется следующим образом:

1. Запустить на своем компьютере Multisim Online.
2. Изложить материал раздела «Краткие теоретические сведения» перед учениками и ответить на вопросы.
3. Запустить симулятор и продемонстрировать работу программы.
4. Озвучить задание лабораторной работы.
5. После завершения работы принять ее результаты.

Проверка работы преподавателя происходит следующим образом:

1. Изучить отчет по лабораторной работе на предмет ошибок. При их наличии – исправить совместно с учеником.
2. Проверить работу модели в симуляторе – в случае ошибок, исправить вместе с учеником.