Методические рекомендации к выполнению конкурсных заданий практического этапа Московского конкурса межпредметных навыков и знаний «Интеллектуальный мегаполис. Потенциал» в номинации «ИТ-класс» для направления «Робототехника»

Оглавление

1. Решение кейса в среде TRIK Studio

1.1. Знакомство с программной средой

Для написания программ для робота TRIK используется программная среда TRIK Studio. Давайте познакомимся с ней более подробно. При запуске программы появляется стартовot окно, в котором предлагается осуществить выбор: создать новый проект или открыть уже существующий (рисунок 1).

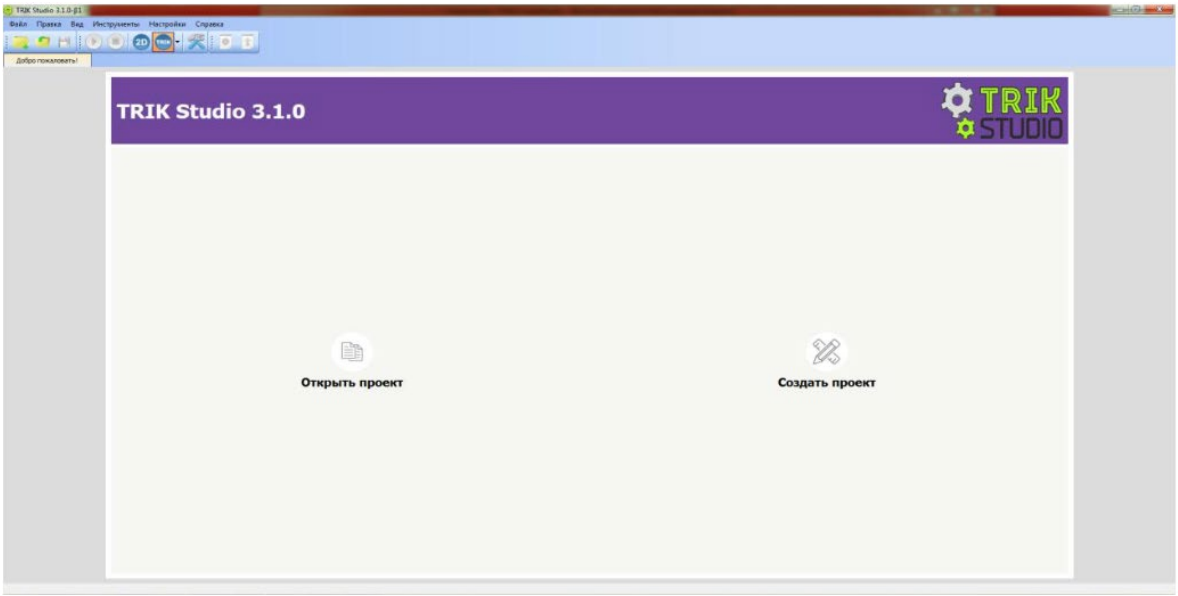


Рисунок 1. Стартовое окно TRIK Studio

После запуска программы открывается основное окно программы (рисунок 2). Центральная часть окна, размеченная клеткой, называется сценой. На сцене собирается программа, которую и будет выполнять робот.

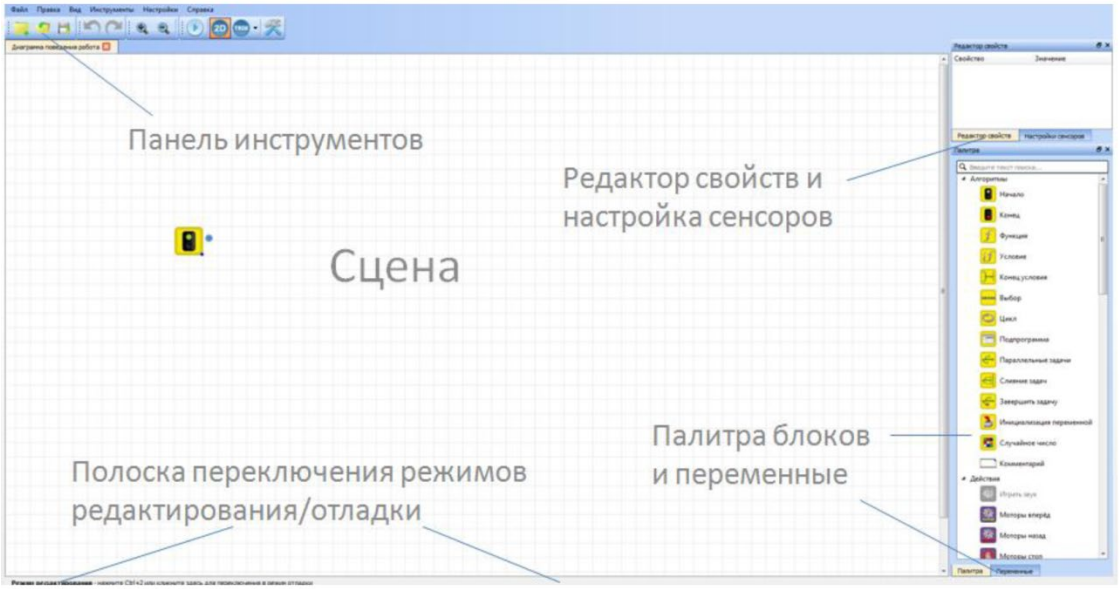


Рисунок 2. Основное окно программы

Справа находятся 4 окна: Палитра, Переменные, Редактор свойств, Настройки сенсоров. Окно «Палитра» содержит различные блоки, из которых можно собрать программу (рисунок 3).

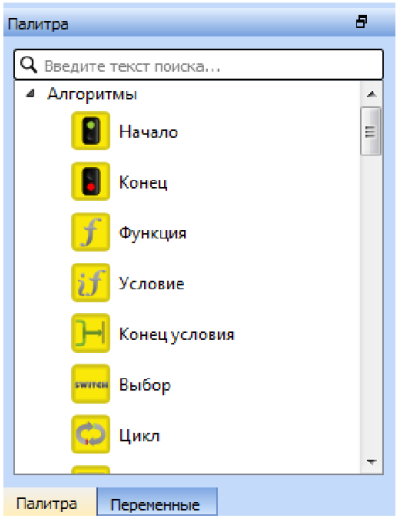


Рисунок 3. Окно «Палитра»

В программе TRIK доступны следующие группы блоков:

- Алгоритмы. Содержит различные алгоритмические структуры. Например, блоки «начало» и «конец», «цикл», «условие». В данной группе представлены блоки желтого цвета.

- Действия. Содержит блоки, описывающие действия механизмов робота. Например, «моторы вперед», «моторы стоп», «сказать», «светодиод». В данной группе представлены блоки фиолетового цвета.

- Ожидание. Содержит блоки, описывающие ожидание какого-то события: таймера, срабатывания датчика, нажатия кнопки. В данной группе представлены блоки оранжевого цвета.

- Рисование. Содержит блоки, позволяющие запрограммировать рисование как на сцене, так и на экране контроллера. В данной группе представлены блоки зеленого цвета.

В любом режиме в палитре могут присутствовать серые блоки. Это означает, что блок в данном режиме недоступен.

Окно «Редактор свойств» отображает свойства блоков и связей. В этом окне их можно редактировать (рисунок 4).

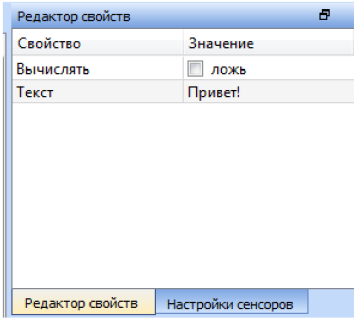


Рисунок 4. Окно «Редактор свойств»

Окно «Настройки сенсоров» отображает датчики (сенсоры), подключенные к различным портам, а также позволяет эти датчики подключить (рисунок 5).

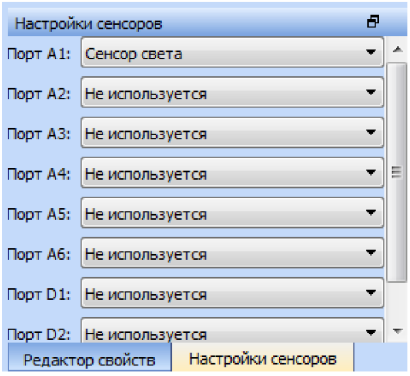


Рисунок 5. Окно «Настройки сенсоров»

В TRIK Studio возможно подключение датчика света, инфракрасного датчика и датчика касания. Более подробно эти сенсоры (датчики) будут изучаться в следующих параграфах.

Вкладка «Переменные» содержит представление значений, уже определенных в среде TRK Studio переменных, а также переменных, заданных пользователем (рисунок 6).

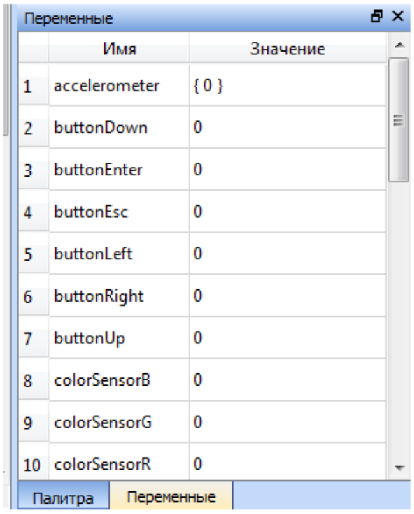


Рисунок 6. Окно «Переменные»

В TRIK Studio представлены две модели мира: виртуальная и реальная. Можно писать программу как для реального робота (созданного из конструктора TRIK, например), так и для виртуального. Виртуальный робот функционирует в 2D-модели, а для реального можно выбрать один из режимов: генерация (перевод программы на выбранный язык, например, Java Script), интерпретация (передача программы по Wi-Fi роботу и исполнение ее). Переключение между режимами выполняется нажатием соответствующих кнопок на панели инструментов (рисунок 7).

Рисунок 7. Кнопки переключения между режимами реального и виртуального робота

Также в TRIK Studio выделяют режим редактора (который предполагает написание и редактирование программ) и режим отладки (который предполагает исполнение программы реальным или виртуальным роботом). Переключение между режимами может выполняться нажатием на полоску в нижней части экрана (в режиме редактирования на ней написано «Режим редактирования») или нажатием горячих клавиш Ctrl+1 (Ctrl+2). В режиме отладки (если выбрана 2D-модель робота) появляется виртуальный робот. По умолчанию это простейшая трехколесная тележка конструктора TRIK (рисунок 8).

Рисунок 8. Виртуальный робот

Главное окно в режиме отладки представлено на рисунке 9.

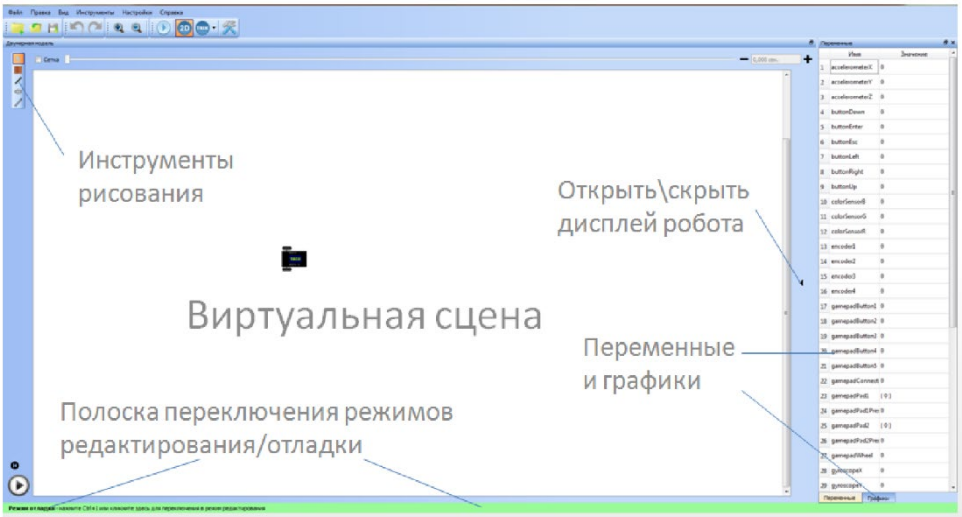


Рисунок 9. Главное окно в режиме отладки

В данном режиме можно отобразить дисплей (экран контроллера TRIK) (рисунок 10). Тогда при отображении команд на дисплее или управлении сенсором, результаты

выполнения команд можно будет наблюдать на экране. Кроме того, имеется возможность нажатия кнопок на контроллере.

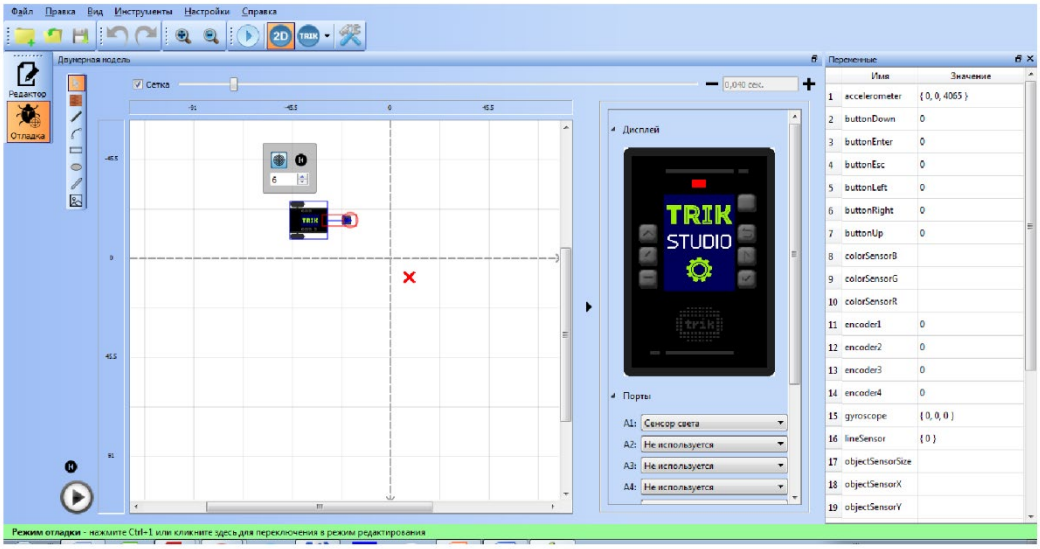


Рисунок 10. Главное окно в режиме отладки с отображением дисплея

В режиме отладки можно нарисовать на виртуальной сцене стену, разметку и т.д. Для этого используется панель инструментов «Рисование». При программировании робота можно предусмотреть реакцию робота на эти объекты (например, остановку у стены или движение вдоль черной линии).

Рассмотрим, какие настройки TRIK Studio можно задать. Для вызова окна настроек используется кнопка «Настройки» (рисунок 11).

Рисунок 11. Кнопка «Настройки»

В окне «Настройки» выделены раздела: поведение, разное, редактор, роботы и горячие клавиши.

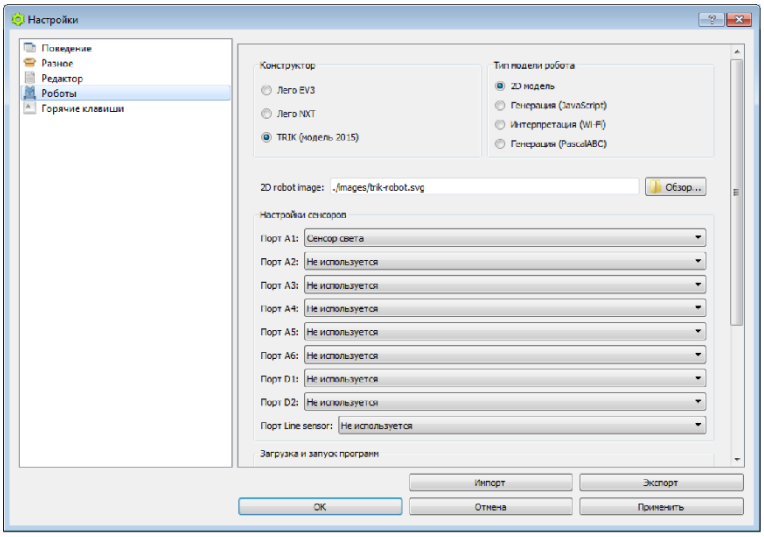


Рисунок 12. Окно «Настройки»

Во вкладке «Поведение» можно выбрать язык (по умолчанию русский), а также параметры автосохранения и реагирования на мышь. Во вкладке «Редактор» - настроить толщину сетки, размер ячеек, размер области масштабирования и другие параметры. Вкладка «Роботы» позволяет задать конструктор, для которого пишется программа. Кроме конструктора TRIK, поддерживаются конструкторы Лего EV3 и NXT. Также можно выбрать реальный или виртуальный режим и задать настройки сенсоров.

Используя программу TRIK Studio, можно писать самые различные программы. Стоит отметить, что любые программы могут быть написаны при помощи трех алгоритмических конструкций: следование, ветвление и цикл. В данном параграфе мы рассмотрим конструкцию следования.

Конструкция следования предполагает последовательное выполнение команд. Блок-схема конструкции следования представлена на рисунке 13.

Рисунок 13. Конструкция следования

В среде TRIK конструкция следования задается просто последовательностью блоков (рисунок 14).

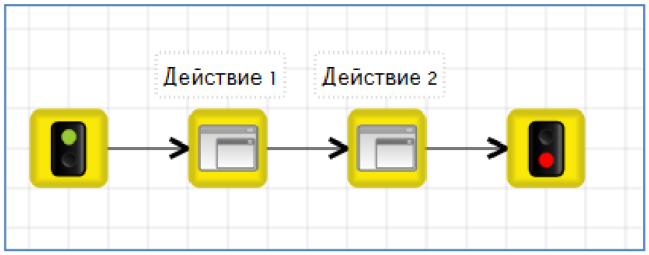


Рисунок 14. Конструкция следования в среде TRIK Studio

Первые программы будут написаны на основе именно этой конструкции.

1.2. Программирование движения робота (тайм-модели)

Движение вперед

Напомним, что базовая тележка (на основе которой ведется работа в TRIK Studio) содержит два колеса, которые подключены к моторам. Чтобы запрограммировать движение робота, необходимо на оба мотора подать одинаковое напряжение.

Для подачи напряжения на моторы используются блоки «Моторы вперед», «Моторы назад» раздела «Действия» панели инструментов (рисунок 15).

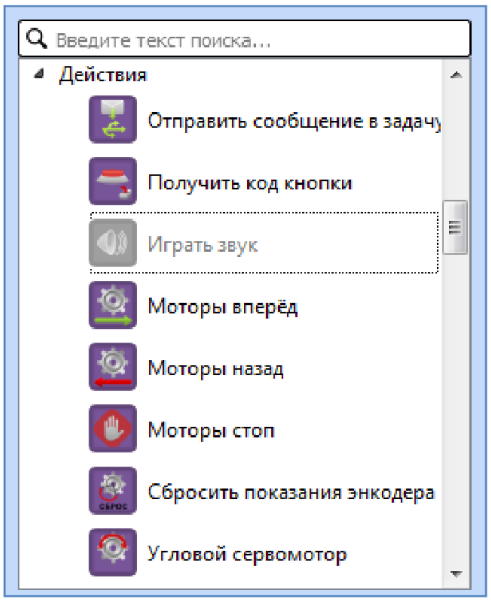


Рисунок 15. Раздел «Действия» панели инструментов

Для подключения силовых моторов у контроллера TRIK используются 4 порта, они обозначаются как M1, M2, M3 и M4. Любые из этих портов могут использоваться для подключения моторов. По умолчанию левое колесо подключается к порту M3, правое – к M4 (рисунок 16)..

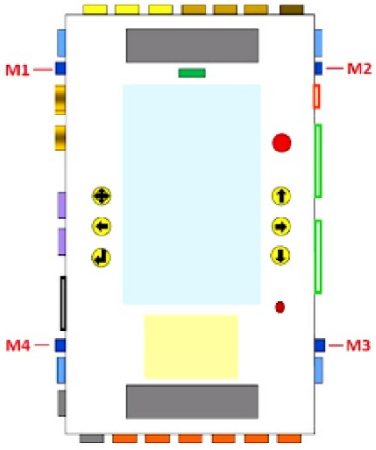


Рисунок 16. Схема подключения моторов к контроллеру TRIK

Чтобы в среде TRIK Studio узнать, к каким портам подключены моторы, можно использовать панель «Моторы». Эта панель в режиме отладки отражается под дисплеем.

Также в этом окне отображается панель «Параметры модели», которая показывает высоту, ширину, массу робота и панель «Порты», в которой отмечены датчики, подключенные к разным портам (рисунок 17).

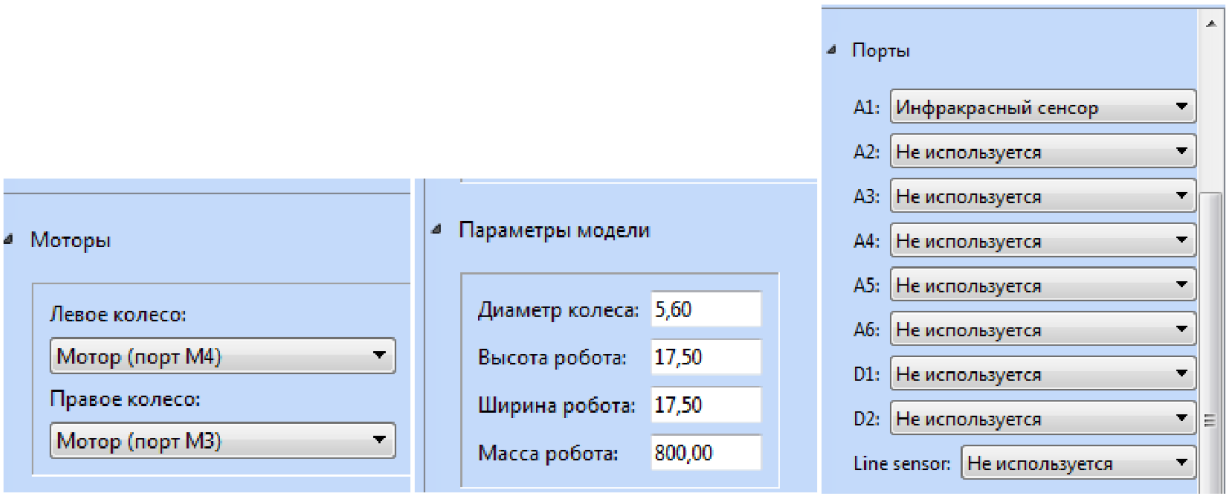


Рисунок 17. Меню «Моторы», «Параметры модели», «Порты»

Чтобы задать движение вперед, нужно использовать блок «Моторы вперед». У блока имеются следующие параметры: порты, на которые подается напряжение, и само напряжение. Напряжение указывается в процентах (от -100 до 100%) и влияет на скорость движения. После блока «Моторы» необходимо обязательно использовать один из блоков ожидания, который будет задавать условие движения (например, движение в течение некоторого времени, движение на заданное количество оборотов и т.п.) (рисунок 18).

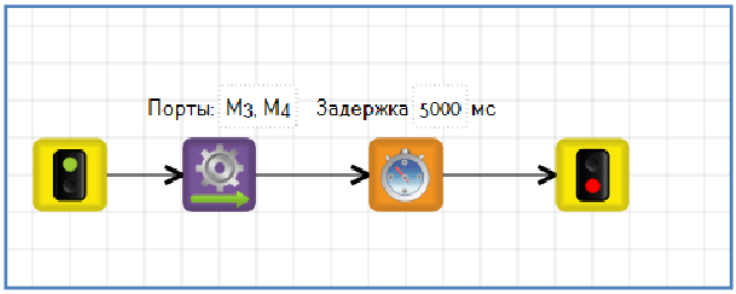


Рисунок 18. Программа движения вперед по таймеру

Кроме движения вперед и назад, робот должен уметь выполнять повороты. Существуют следующие виды поворотов:

- *Резкий поворот*

Для выполнения резкого поворота одно колесо робота движется вперед, а второе стоит на месте (рисунок 19).

Рисунок 19. Резкий поворот

Для остановки мотора используется блок «Моторы стоп». Программа для выполнения резкого поворота представлена на рисунке 20.

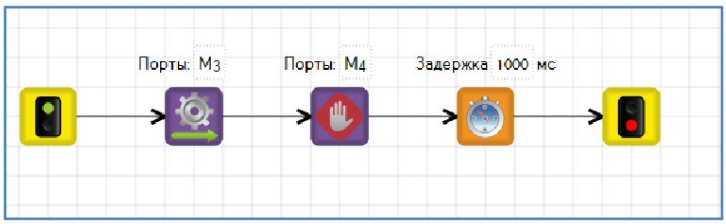


Рисунок 20. Программа для выполнения резкого поворота

Траектория движения робота представлена на рисунке 21.

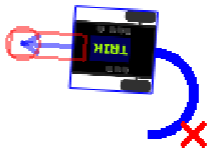


Рисунок 21. Траектория движения робота при резком повороте

- Плавный поворот

Для выполнения плавного поворота оба колеса должны двигаться вперед: одно с большей скоростью, другое – с меньшей (рисунок 22).

Рисунок 22. Плавный поворот

Программа для выполнения плавного поворота представлена на рисунке 23.

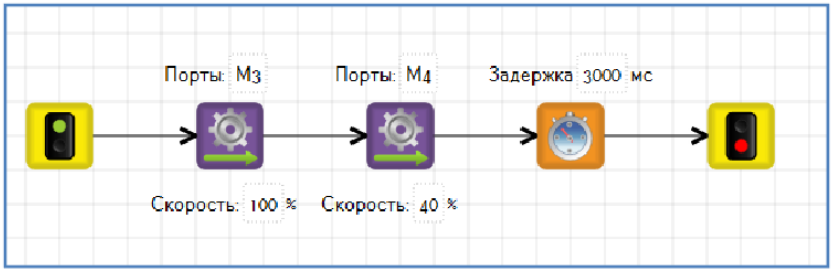


Рисунок 23. Программа для выполнения плавного поворота

Разница между мощностями определяет плавность поворота. Чем больше разница, тем поворот резче, чем меньше – тем плавнее. Траектория движения робота при плавном повороте представлена на рисунке 24.

Рисунок 24. Траектория движения робота при плавном повороте

- *Поворот на месте*

Для выполнения поворота на месте одно колесо робота движется вперед, а второе – назад (рисунок 25).

Рисунок 25. Поворот на месте

Программа для выполнения поворота на месте представлена на рисунке 26.

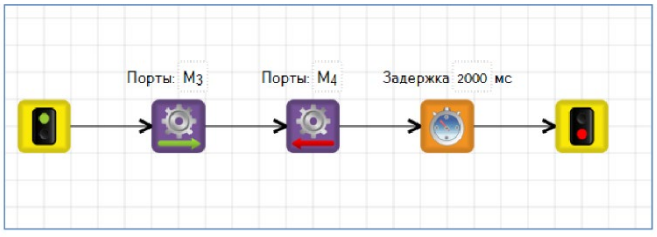


Рисунок 26. Программа для выполнения поворота на месте

Обратите внимание, что мощности, подаваемые на разные колеса, должны быть одинаковыми. Т.к. робот поворачивается на месте, то траектория движения будет представлять собой точку. Траектория движения робота (точка) при повороте на месте представлена на рисунке 27.

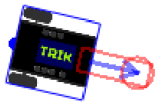


Рисунок 27. Траектория движения робота при повороте на месте

Такие модели называются тайм-моделями, т.к. движутся в течение определенного времени (по таймеру).

1.3. Программирование движения робота (энкодерные модели)

При использовании тайм-моделей возникают определенные проблемы. При разряде аккумулятора при той же мощности тележка будет проходить меньшее расстояние, а значит, поведение робота становится непредсказуемым. Чтобы избежать этой ситуации, используются энкодеры – датчики количества оборотов, которые имеются у каждого мотора. Если задать движение до достижения определенного количества оборотов, то независимо от уровня заряда робот пройдет одинаковое расстояние.

В программе моторы обозначаются M3, M4, а энкодеры – E3, E4. При этом мотору M3 соответствует энкодер E3, а мотору M4 – энкодер E4. Важно помнить, что энкодеры накапливают значение, т.е. количество оборотов будет увеличиваться при каждом движении робота. Соответственно, необходимо осуществлять сброс показаний энкодеров перед очередным движением. Кроме того, желательно выполнять команду «моторы стоп» после завершение очередного «элементарного действия» (движения вперед, назад, поворота). Программа движения вперед будет выглядеть следующим образом (рисунок 28):

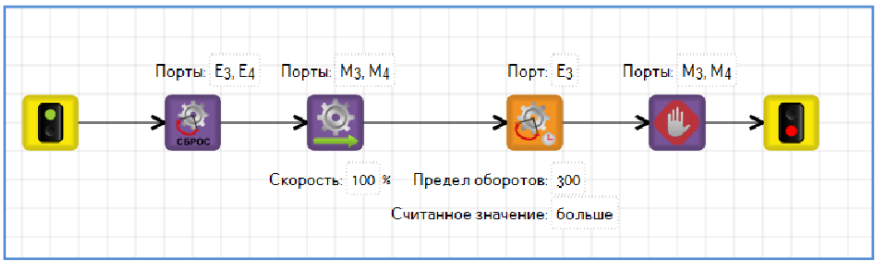


Рисунок 28. Движение вперед по энкодерной модели

Обратите внимание на свойства блока «Ждать показания энкодера»: считывается значение только одного энкодера, т.е. анализируется количество оборотов одного мотора. В свойствах устанавливаем значение «больше 500», т.е. робот будет выполнять движение до тех пор, пока количество оборотов не станет больше 500.

Таким образом, алгоритм выполнения простейшего действия (движения вперед, назад, поворота) будет следующим (рисунок 29):

1. Сброс энкодера.
2. Установка моторов (один или два блока в зависимости от того, что нужно выполнить: движение или поворот).
3. Ожидание значения энкодера.
4. Остановка моторов.

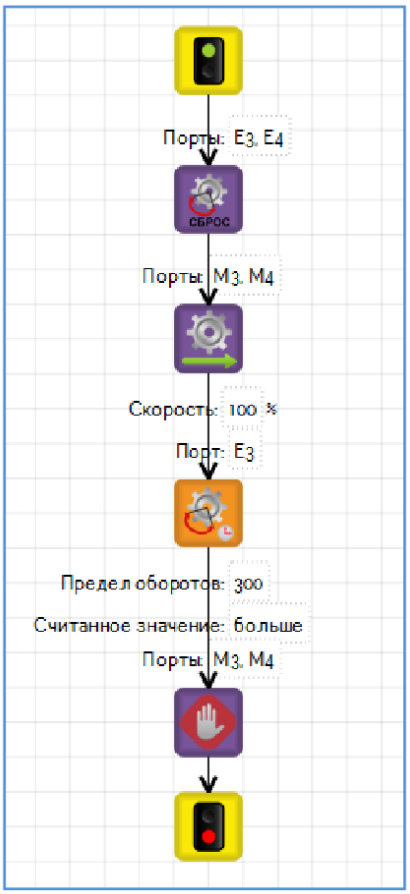


Рисунок 29. Алгоритм движения

Повороты будут программироваться аналогично, но будут использоваться два блока (в зависимости от выбранного типа поворота «Моторы вперед», «Моторы назад», «Моторы стоп»).

1.4. Условный оператор

Часто возникают ситуации, когда действие робота зависит от какого-либо условия. Например, нужно решить такую задачу: робот движется в течение 3 секунд. Затем выполняет поворот: если количество оборотов больше 2000, робот поворачивается направо, иначе – налево.

Для проверки используется условный оператор (оператор ветвления). Блок-схема обычного (полного) условного оператора представлена на рисунке 30.

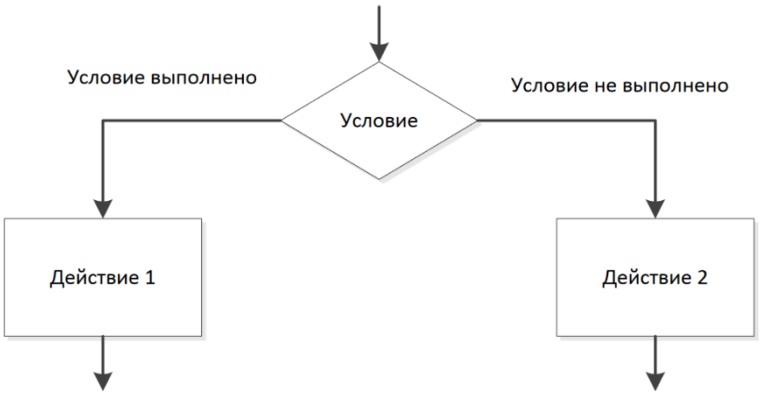


Рисунок 30. Полный условный оператор

В среде TRIK используется блок «Условие» ( ), который находится в разделе «Алгоритмы» панели инструментов. У блока «Условие» имеется один параметр «Условие» (рисунок 31).

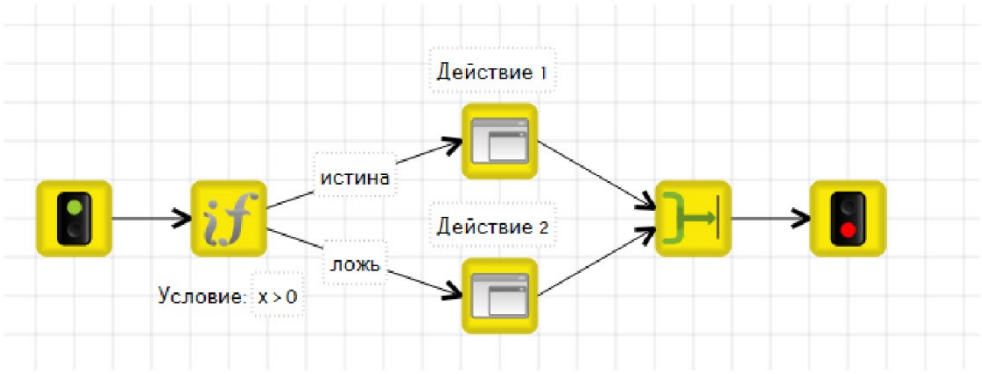


Рисунок 31. Условный оператор в среде TRIK.

Обратите внимание, что при выполнении условия выполняется действие, подключенное к стрелке «истина», иначе – действие, подключенное к стрелке «ложь». Очень важно не забывать подписывать стрелки, иначе программа работать не будет. Для того, чтобы было понятно, где заканчивается условный оператор, используют блок «Конец условия». Кроме стандартного (полного) условного оператора может использоваться неполный условный оператор. Блок-схема неполного условного оператора представлена на рисунке 32.

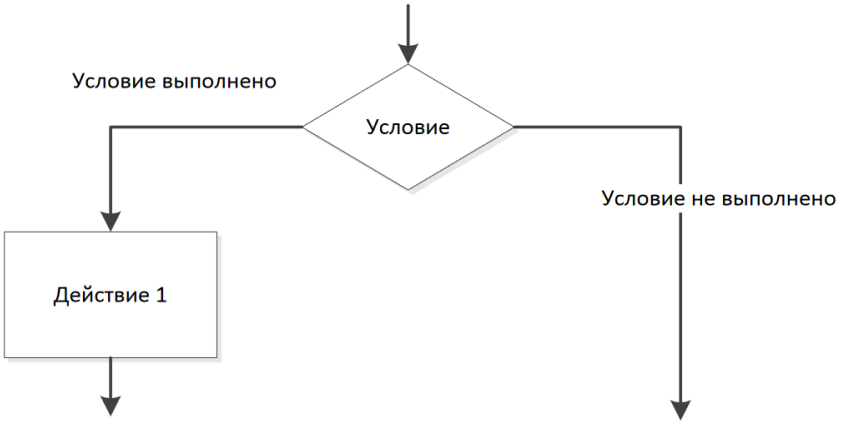


Рисунок 32. Неполный условный оператор

В среде TRIK алгоритм будет выглядеть следующим образом (рисунок 33):

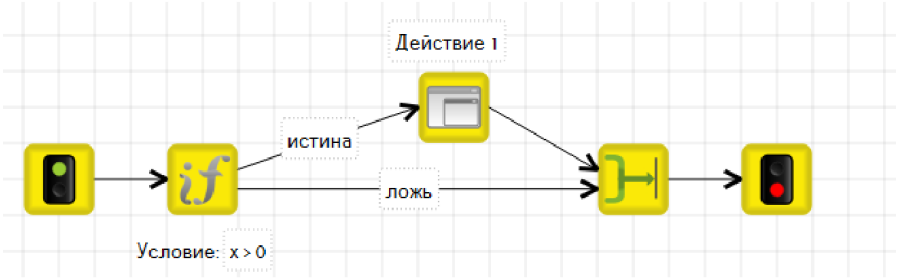


Рисунок 33. Неполный условный оператор в среде TRIK.

Кроме того, могут использоваться вложенные условные операторы, когда при выполнении (или невыполнении) условия, снова задается некоторое условие (рисунок 34).

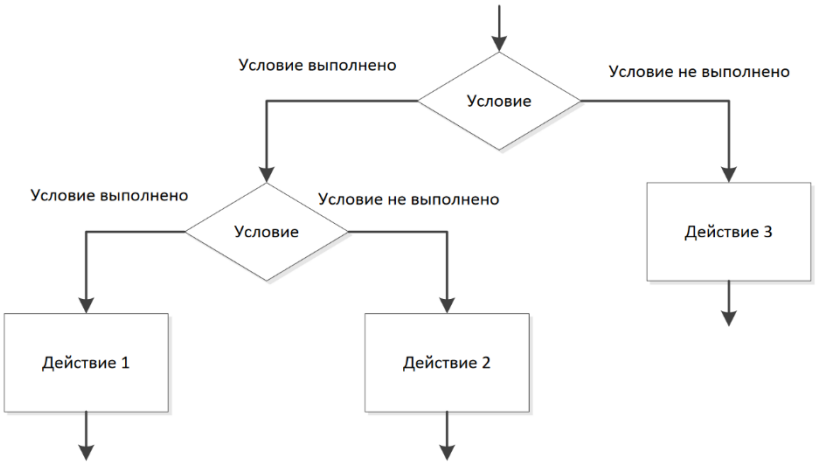


Рисунок 34. Вложенный условный оператор

В среде TRIK вложенный оператор будет выглядеть следующим образом (рисунок 35):

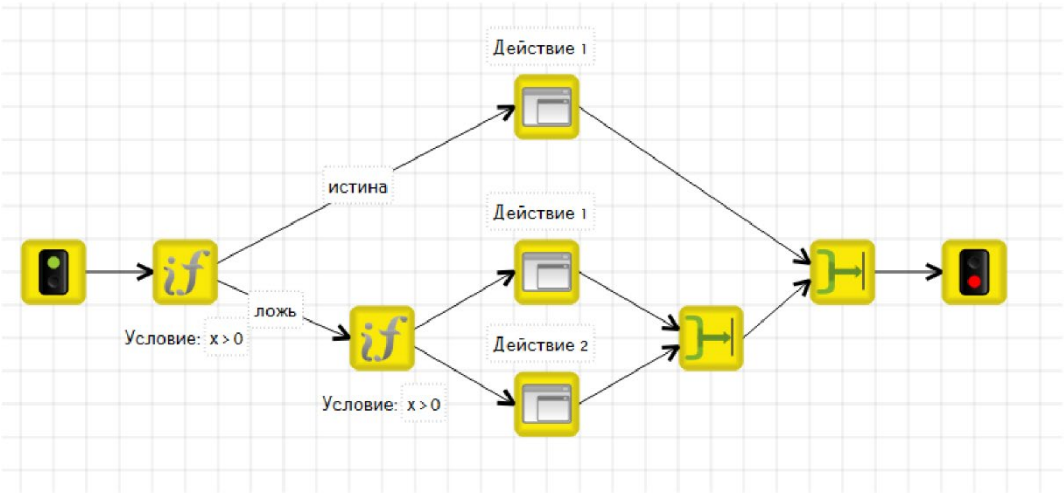


Рисунок 35. Вложенный условный оператор в среде TRIK.

В качестве примера использования составного оператора можно предложить решение такой задачи: определить, какое из двух чисел больше. Если числа равны, то сообщить об этом. Пример блок-схемы решения представлен на рисунке 36.

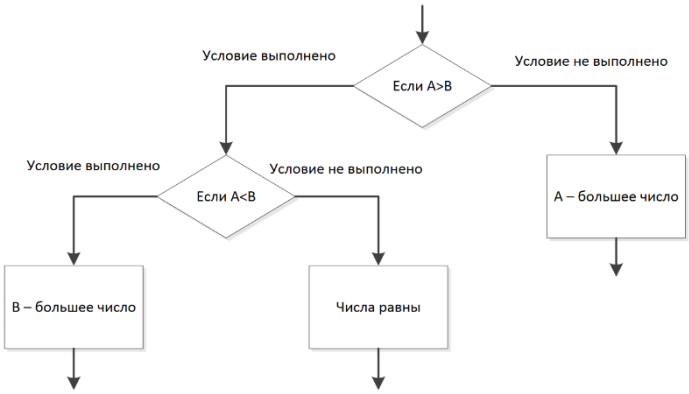


Рисунок 36. Часть блок-схемы решения задачи

Также при использовании условного оператора могут использоваться составные условия. Для этого используются слова – связки (рисунок 37).

Связка	Пример составного условия	Выполнение условия
and	(условие 1) and (условие 2)	Составное условие выполняется тогда, когда одновременно выполняется и первое, и второе условие
or	(условие 1) or (условие 2)	Составное условие выполняется тогда, когда выполняется хотя бы одно из условий
not	not (условие 1)	Составное условие выполняется тогда, когда первое условие не выполнено

Таблица 37. Составные условия

Составное условие в среде TRIK задается в окне свойств условного оператора (рисунок 38, рисунок 39).

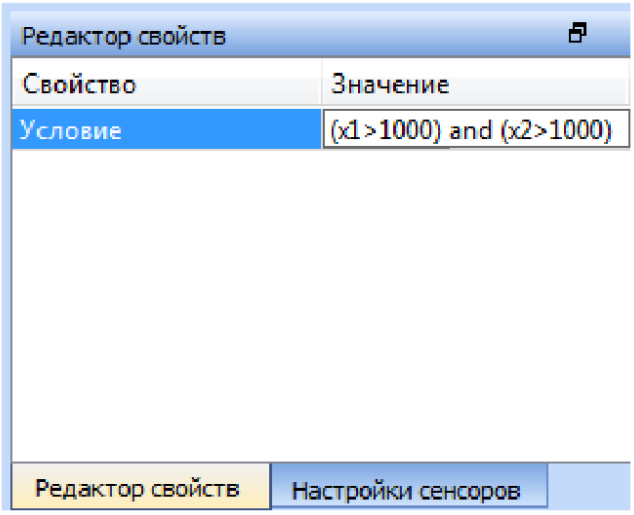


Рисунок 38. Задание составного условия

оператор	синтаксис	пример
равенство	==	enterButton == 1
неравенство	!=	rightButton != 0
больше	>	sensorA1 > 50
меньше	<	sensorA2 < 30
больше или равно	>=	sensorA3 >= 50
меньше или равно	<=	sensorA4 <= 50

Таблица 39. Операторы сравнения

1.5. Подпрограммы

Подпрограмма – это отдельная часть программы, которая объединяет некоторую последовательность действий, имеет собственное имя и может быть неоднократно вызвана из основной программы.

Подпрограммы обычно используются тогда, когда в программе часто используются повторяющиеся последовательности команд. Для создания подпрограммы в среде TRIK Studio используется блок «Подпрограмма» (рисунок 40).

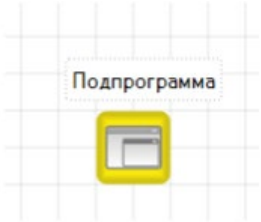


Рисунок 40. Блок «Подпрограмма»

При добавлении этого блока на рабочее поле появляется служебное окно, в которое нужно ввести название подпрограммы (рисунок 41).

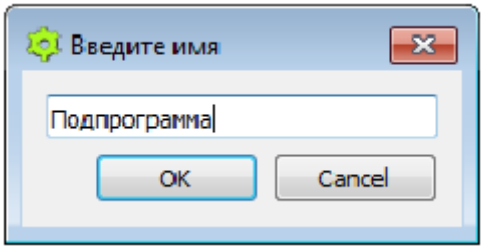


Рисунок 41. Служебное окно для ввода названия подпрограммы

Чтобы задать последовательность команд, которые объединены в подпрограмму, достаточно дважды щелкнуть по значку подпрограммы в основном окне, после чего откроется окно с названием подпрограммы.

Пусть была создана подпрограмма с названием «Вперед» (рисунок 42).

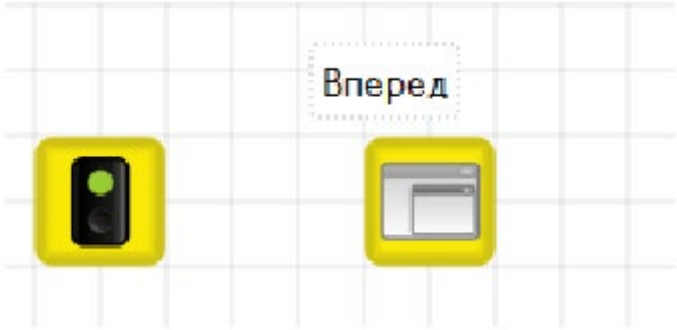


Рисунок 42. Подпрограмма «Вперед»

При двойном щелчке по подпрограмме откроется окно с названием «Вперед» (рисунок 43).

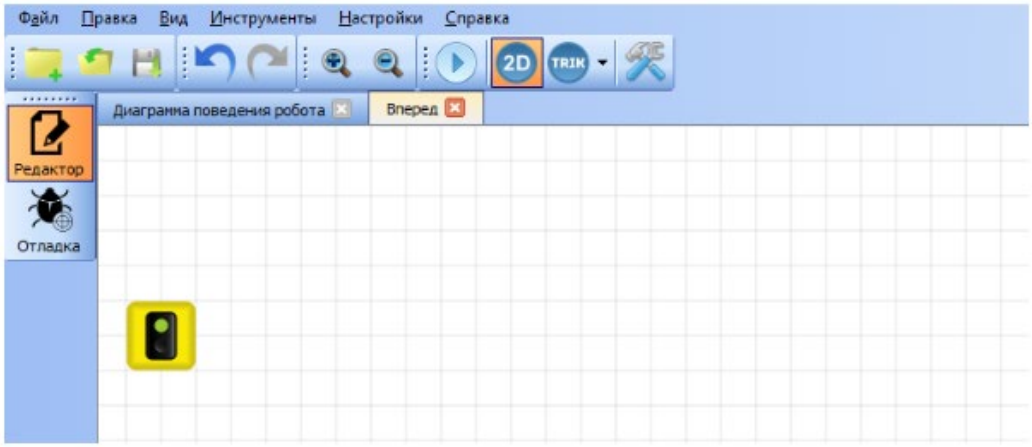


Рисунок 43. Окно подпрограммы «Вперед»

В этом окне размещаются команды подпрограммы. Обратите внимание, что в окне подпрограммы автоматически появляется блок «Начало». Соответственно, в подпрограмме обязательно должен быть и блок «Конец» (за исключением ситуации использования бесконечного цикла). Если нажать кнопку «Выполнить», когда будет открыто окно подпрограммы, то будут выполняться только команды подпрограммы. Это удобно при проверке работы отдельных подпрограмм. Рекомендуется после написания подпрограммы проверить ее работоспособность, а только после этого использовать в основной программе. На рисунке 44 приведены команды подпрограммы «Вперед», которые задают движение робота вперед на 1000 оборотов.

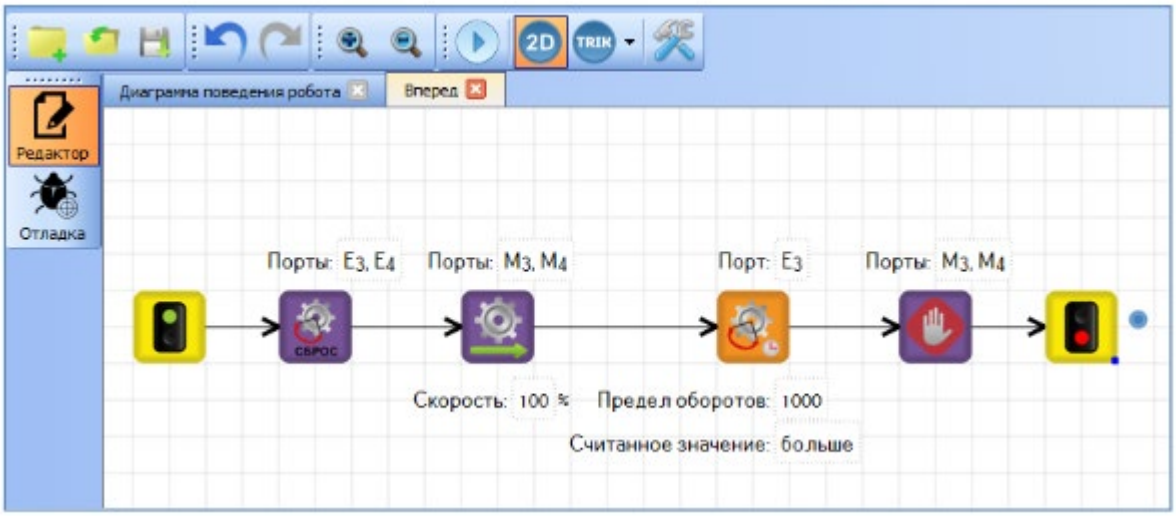


Рисунок 44. Подпрограмма «Вперед»

Когда подпрограмма написана и отлажена, ее можно использовать в основной программе. После добавления подпрограммы в правой части основного окна отображается окно «Подпрограммы», где показаны все подпрограммы, которые могут использоваться в данной программе (рисунок 45).

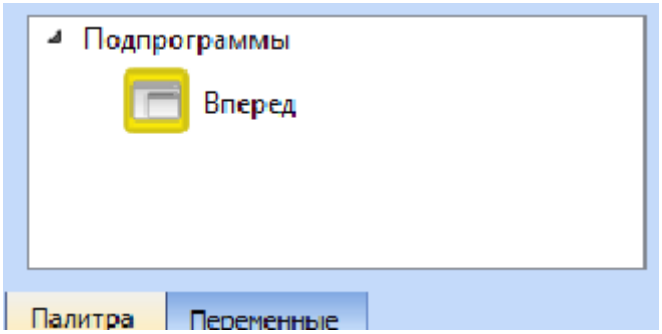


Рисунок 45. Окно «Подпрограммы»

Для добавления подпрограммы в текст основной программы достаточно перетащить блок с названием подпрограммы в основное окно. При этом нужно помнить, что одна и та же подпрограмма может быть использована многократно. На рисунке 46 представлена программа с использованием подпрограммы «Вперед».

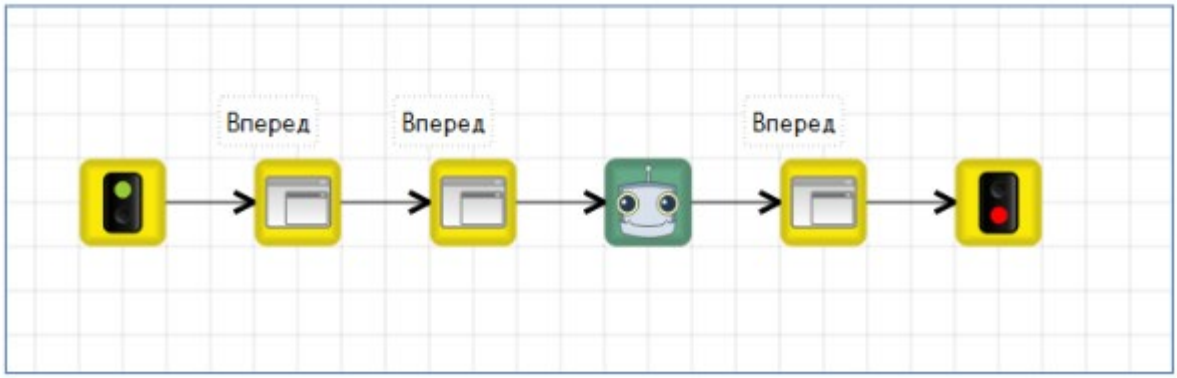


Рисунок 46. Внешний вид программы

Обратите внимание, что подпрограмма «Вперед» используется 3 раза. При выполнении программы робот проедет на расстояние 2000 оборотов, покажет смайлик и проедет еще на 1000 оборотов.

1.6. Прохождение лабиринта.

Разберем алгоритм прохождения робота по лабиринту.

Для прохождения лабиринта роботу (рисунок 48) рекомендуется использовать правило правой руки.

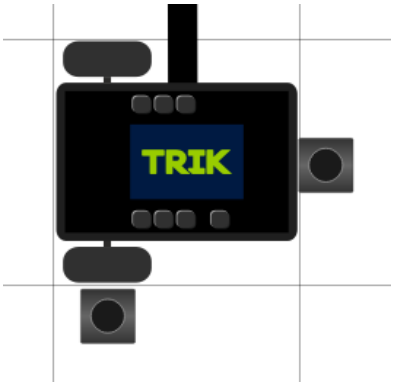
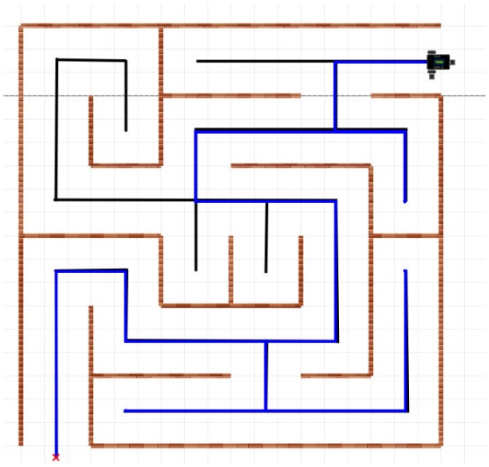


Рисунок 47. Робот для прохождения лабиринта

Попав в лабиринт, при выполнении условия односвязанности лабиринта (в задании оно выполняется), удерживая одну руку в контакте с одной стенкой лабиринта, вы гарантировано не потеряетесь и достигнете другого выхода, если он есть. В противном случае, вы вернетесь ко входу, пройдя каждый коридор рядом с этим соединенным участком стен хотя бы один раз.

Алгоритм реализуется следующим образом:

- Если справа есть стенка, а спереди нет - двигайтесь прямо;
- Если справа нет стенки - поверните направо и двигайтесь прямо;
- Если справа есть стенка, и спереди есть стенка - поверните налево (рисунок 48) .



-

- *Рисунок 48. Траектория, которую проходит робот*

-

То есть наша программу будет выглядеть следующим образом (рисунок 49):

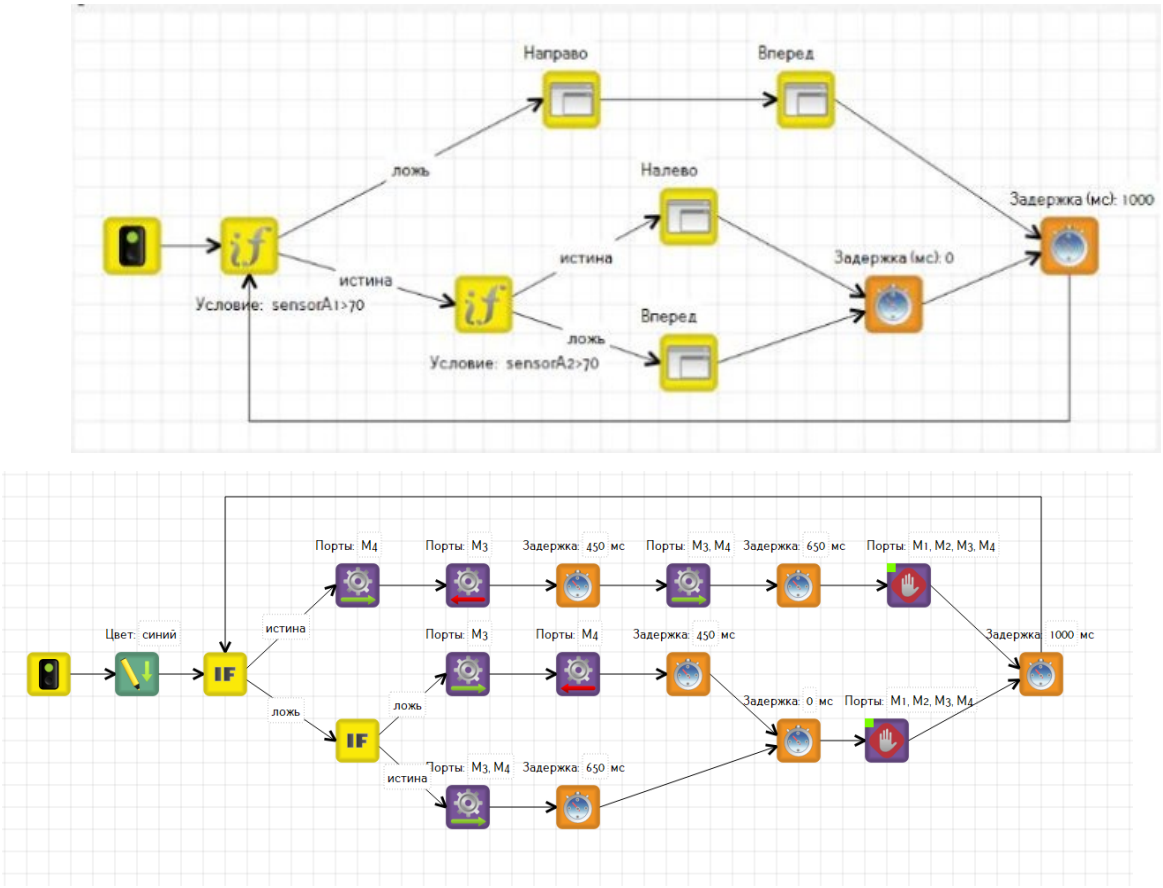


Рисунок 47. Программная реализация прохождения лабиринта по правильный правой руки

2. Решение кейса со сборкой конструктора

2.1. Общая информация о знаниях и навыках

Для выполнения задания уровня сложности «базовый» участнику необходимо обладать следующими знаниями и навыками.

Знания:

- Базовых принципов работы с инструментом и крепежом.
- Принципов составления и электрических схем и базовых электронных компонентов.
- Принципов составления и стандартов изображения алгоритмов.
- Синтаксиса языка С, принципов программирования Arduino, основ управления движением робота.

Умения:

- Следовать инструкции, навыки использования базовых инструментов и крепежа.
- Читать блок-схемы, принципиальные электрические схемы, соединять электронные модули в соответствии со схемами.
- Разрабатывать и изображать алгоритмы в соответствии с заданием, используя специализированное программное обеспечение.
- Разрабатывать и отлаживать программы, обеспечивающие движение робота в соответствии с алгоритмом и заданием.

Для выполнения задания уровня сложности «повышенный» участнику необходимо знать и уметь:

- Составлять алгоритмы и реализовывать программы, считывающие и обрабатывающие несколько видов датчиков.
- Уметь изображать алгоритмы повышенной сложности в соответствии с заданием, используя специализированное программное обеспечение.
- Знать синтаксис языка С и принципов программирования Arduino на уровне, достаточном для реализации одновременного управления движением робота и считывания двух видов датчиков.
- Знать принципы работы ИК и УЗ датчиков препятствий и особенности их подключения к плате Arduino.

2.2. Сборка робототехнического набора

Даны детали для сборки робота из набора (могут быть изменены модели датчиков и микроконтроллеров) (рисунок 50, рисунок 51):

1) Микроконтроллер Arduino Nano
2) Драйвер мотора TB6612
3) Два модуля датчика линии KY-033 (HW-511)
4) Два ультразвуковых датчика расстояния HC-SR04
5) Два мотора с колесами
6) Источник питания (аккумулятор или батарейки)
7) Макетная плата и набор проводов
8) ИК - датчик препятствия LM393

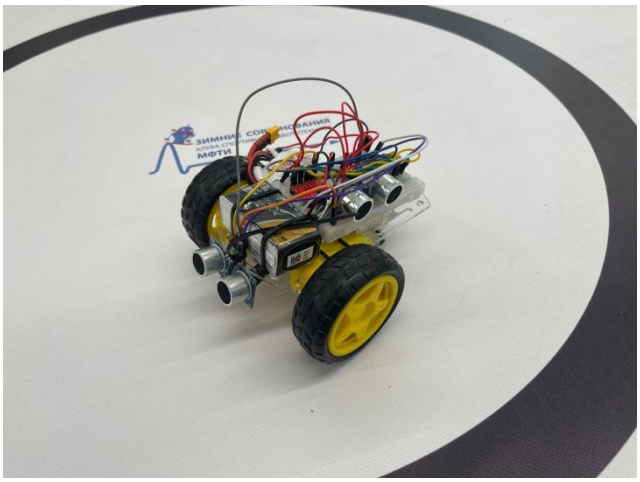


Рисунок 50. Сборка робота для лабиринта

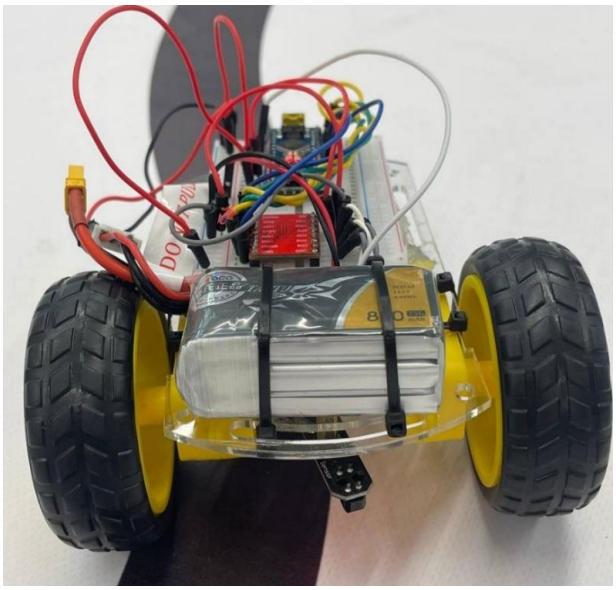


Рисунок 51. Сборка робота для езды по лини

2.3. Подключение драйвера моторов

Подключение моторов к Arduino с использованием драйвера TB6612 – это распространённая схема для управления электродвигателями, такими как DC-моторы (постоянного тока) или шаговые двигатели. Драйвер TB6612 позволяет легко управлять скоростью и направлением вращения моторов с использованием ШИМ (широтно-импульсной модуляции) и сигналов управления.

Основные компоненты:

1. Arduino – микроконтроллер, который управляет драйвером и отправляет команды на моторы.

2. Драйвер TB6612 – H-мост для управления двумя двигателями. Он поддерживает ток до 1.2A на канал и кратковременный ток до 3.2A. Также может быть использован любой из драйверов на рисунке 52.

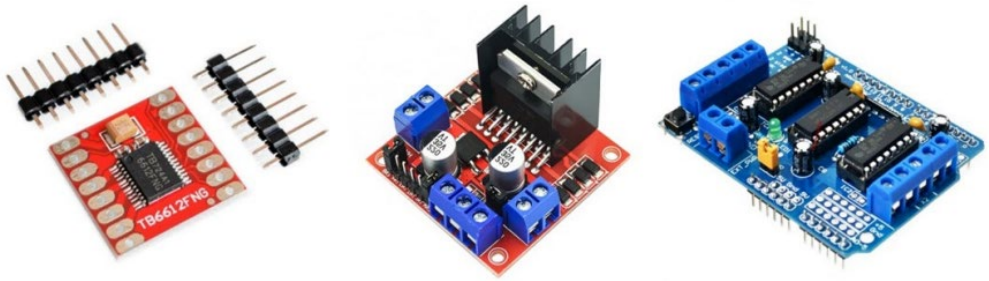


Рисунок 52. Драйверы двигателя с микросхемами TB6612, L298 и l293D

3. Моторы постоянного тока – обычные DC-моторы или шаговые двигатели, которые будут подключены через драйвер.

4. Источник питания – для питания моторов (например, батарея 6-12 В или адаптер).

Схема подключения:

1. Моторы: Драйвер TB6612 может управлять двумя моторами. На плате есть клеммы для подключения двигателей:

- AIN1, AIN2, BIN1, BIN2 – входы для управления направлением моторов.
- PWMA и PWMB – входы для управления скоростью моторов через ШИМ.
- A01, A02, B01, B02 – выходы, к которым подключаются сами моторы.

2. Питание:

- VCC – питание для логики драйвера (обычно 5В от Arduino).
- VM – питание для моторов (обычно 6-12 В в зависимости от типа моторов).
- GND – общий провод.

3. ШИМ-сигналы:

- Arduino генерирует сигналы ШИМ для управления скоростью вращения моторов. Эти сигналы передаются на входы PWMA и PWMB.
- Для изменения направления вращения используются сигналы на входах AIN1, AIN2 для мотора A и BIN1, BIN2 для мотора B.

Пример подключения (рисунок 53):

1. Подключите питание драйвера: VCC (5B) к 5V на Arduino, GND к GND.
2. Подключите питание для моторов (VM) к источнику питания.
3. Мотор A подключите к A01 и A02, мотор B – к B01 и B02.
4. AIN1, AIN2 подключите к цифровым выводам Arduino, например, к D2 и D3.
5. PWMA подключите к выводу, поддерживающему ШИМ, например, D9.
6. Повторите для второго мотора: BIN1, BIN2 подключите к D4 и D5, PWMB – к D10.

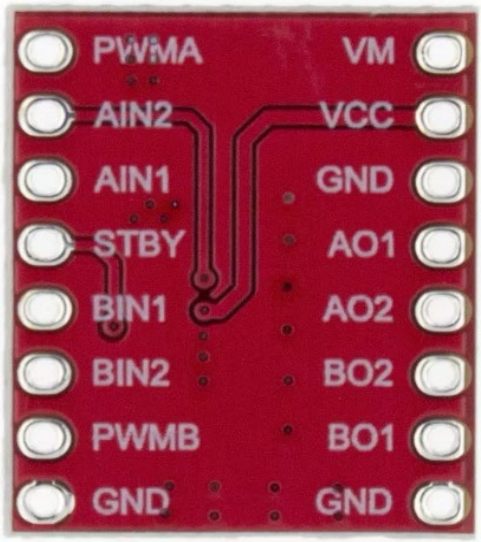


Рисунок 53. Обратная сторона TB6612

Пример кода для Arduino и TB6612:

```
// Пины для мотора A
int AIN1 = 2;
int AIN2 = 3;
int PWMA = 9; // ШИМ пин для управления скоростью мотора A

// Пины для мотора B
int BIN1 = 4;
int BIN2 = 5;
int PWMB = 10; // ШИМ пин для управления скоростью мотора B

void setup() {
  // Настройка пинов на выход
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(PWMB, OUTPUT);
}
void loop() {
  // Включаем мотор A вперёд на половину мощности
  digitalWrite(AIN1, HIGH);
  digitalWrite(AIN2, LOW);
  analogWrite(PWMA, 128); // Скорость 50% от полной

  // Включаем мотор B назад на полную мощность
  digitalWrite(BIN1, LOW);
  digitalWrite(BIN2, HIGH);
  analogWrite(PWMB, 255); // Скорость 100%

  delay(2000); // Работаем 2 секунды
  // Останавливаем оба мотора
  analogWrite(PWMA, 0);
  analogWrite(PWMB, 0);
  delay(2000); // Остановка на 2 секунды
}
```

Листинг 1. Управление моторами

Принцип работы:

1. Направление вращения: Установка комбинации сигналов на входах AIN1/AIN2 или BIN1/BIN2 определяет направление вращения моторов:

- AIN1 = HIGH, AIN2 = LOW – мотор A вращается вперёд.
- AIN1 = LOW, AIN2 = HIGH – мотор A вращается назад.
- Если оба пина в одном состоянии (оба HIGH или оба LOW), мотор останавливается.

2. Скорость вращения: Используя функцию `analogWrite()`, можно регулировать скорость мотора, передавая на PWMA или PWMB значения от 0 (стоп) до 255 (максимальная скорость).

Использование драйвера TB6612 с Arduino – это удобный и эффективный способ управления моторами в робототехнике или других проектах с электродвигателями.

2.4. Подключение датчиков расстояния

Датчик LM393 для обнаружения препятствий

Датчик LM393 для обнаружения препятствий — это инфракрасный (IR) модуль, который используется для распознавания объектов или препятствий на определённом расстоянии. Этот датчик часто применяется в робототехнике, системах безопасности и других автоматизированных устройствах для определения наличия объектов перед датчиком.

Основные компоненты датчика:

1. **Инфракрасный передатчик (светодиод)**: излучает инфракрасный свет в направлении перед датчиком.
2. **Инфракрасный приёмник (фотодиод)**: принимает отражённый инфракрасный свет, если он возвращается от какого-либо объекта.
3. **Чип LM393**: это ключевая микросхема модуля. LM393 — это двойной компаратор напряжения, который сравнивает сигнал, полученный от фотодиода, с эталонным значением и генерирует цифровой сигнал на выходе. Если объект присутствует, то на выходе генерируется логический ноль (низкий уровень), если препятствия нет — высокий уровень.
4. **Потенциометр**: используется для регулировки чувствительности датчика, то есть изменения расстояния, на котором датчик распознаёт объекты.
5. **Выходы**:
   - **VCC**: подача питания (обычно 3.3V или 5V).
   - **GND**: заземление.
   - **DO (Digital Output)**: цифровой выход. Если препятствие обнаружено, сигнал будет низким (0), если нет — высоким (1).

Как работает датчик LM393 (рисунок 54):

1. Инфракрасный светодиод излучает свет в прямом направлении.
2. Если на пути есть объект, инфракрасный свет отражается от него и возвращается к приёмнику (фотодиоду).
3. Если отражённый свет был принят, компаратор LM393 сравнивает напряжение от фотодиода с эталонным значением, установленным с помощью потенциометра.
4. Когда напряжение на входе превышает эталонное значение (т.е. объект находится на определённом расстоянии), компаратор активирует выходной сигнал (DO = 0).

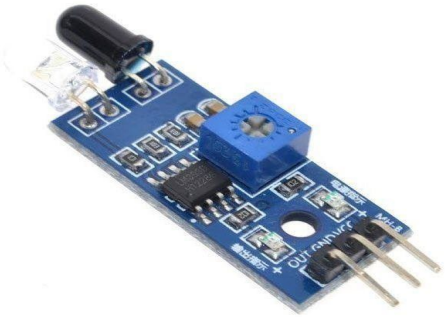


Рисунок 54. Датчик LM393

Описание и подключение двух датчиков LM393 к Arduino

Для работы с двумя датчиками на базе компаратора LM393, мы подключаем каждый из них к отдельному цифровому пину на Arduino. Это позволяет считывать данные с обоих датчиков и обрабатывать их независимо.

Подключение модулей на базе LM393 к Arduino

Датчик 1:

- VCC → 5V на Arduino
- GND → GND на Arduino
- OUT → D2 на Arduino

Датчик 2:

- VCC → 5V на Arduino
- GND → GND на Arduino
- OUT → D3 на Arduino

Код для Arduino и LM393

```
// Определение пинов для подключения датчиков
const int sensorPin1 = 2; // Датчик 1 (подключен к D2)
const int sensorPin2 = 3; // Датчик 2 (подключен к D3)

void setup() {
  // Настраиваем пины как входы
  pinMode(sensorPin1, INPUT);
  pinMode(sensorPin2, INPUT);

  // Инициализация последовательного порта для вывода данных в монитор порта
  Serial.begin(9600);
}
```

```
void loop() {
  // Чтение состояния каждого датчика
  int sensorState1 = digitalRead(sensorPin1);
  int sensorState2 = digitalRead(sensorPin2);

  // Вывод состояния первого датчика в монитор порта
  if (sensorState1 == HIGH) {
    Serial.println("Датчик 1 активен (HIGH)");
  } else {
    Serial.println("Датчик 1 не активен (LOW)");
  }

  // Вывод состояния второго датчика в монитор порта
  if (sensorState2 == HIGH) {
    Serial.println("Датчик 2 активен (HIGH)");
  } else {
    Serial.println("Датчик 2 не активен (LOW)");
  }
  // Небольшая задержка перед следующей проверкой
  delay(500);
}
```

Листинг 2. Считывание с датчиков

Описание кода

1. Определение пинов: Мы объявляем два пина для подключения выходов датчиков — один подключен к пину D2, другой к пину D3.

2. Настройка пинов: В функции setup() оба пина настраиваются как входы с помощью команд pinMode(sensorPin1, INPUT) и pinMode(sensorPin2, INPUT).

3. Чтение данных с датчиков: В цикле loop() с помощью функции digitalRead() считывается состояние с каждого датчика по очереди.

4. Вывод данных: Результаты работы каждого датчика выводятся в последовательный порт (Serial Monitor). Если состояние датчика HIGH — выводится сообщение о его активации, если LOW — о неактивном состоянии.

5. Задержка: добавляется небольшая задержка в 500 мс перед следующей проверкой.

Алгоритм использования двух датчиков линии на основе TCRT5000

Описание датчиков TCRT5000

Датчик TCRT5000 — это инфракрасный сенсор, состоящий из инфракрасного светодиода и фотоприёмника. Он используется для обнаружения линии (обычно черной линии на белом фоне) за счёт отражения инфракрасного света от поверхности (рисунок 55).

Рисунок 55. Датчик линии TCRT5000

Основные характеристики TCRT5000:

- **Рабочее напряжение**: 3.5V — 5.5V.
- **Дальность обнаружения**: до 10 см, в зависимости от цвета и материала поверхности.
- **Выходной сигнал**: аналоговый и цифровой (в зависимости от схемы подключения).

Подключение

Каждый датчик TCRT5000 подключается следующим образом:

- **VCC**: подключите к положительному напряжению (например, 5V).
- **GND**: подключите к земле.
- **OUT**: цифровой выход, который подключается к соответствующему пину на микроконтроллере (например, Arduino).
-

Алгоритм работы с двумя датчиками

1. **Инициализация**:
   - Подключите два датчика TCRT5000 к микроконтроллеру. Например, один датчик подключите к пину D2, а второй — к D3.
2. **Чтение значений**:
   - Считывайте состояния обоих датчиков в цикле. Если датчик видит черную линию (объект), его выходной сигнал будет LOW, если белую — HIGH.

3. **Определение положения линии**:

- Если оба датчика видят черную линию (оба сигнала LOW), значит, робот находится на линии и должен двигаться прямо.
- Если только левый датчик видит линию (LOW), значит, робот должен поворачивать влево.
- Если только правый датчик видит линию (LOW), значит, робот должен поворачивать вправо.
- Если оба датчика не видят линию (оба сигнала HIGH), робот должен искать линию (например, поворотом или движением назад).

4. **Движение**:

- В зависимости от состояния датчиков, управляйте моторами робота (движение вперёд, поворот налево, поворот направо или движение назад).

Пример кода на Arduino и двух датчиков линии на основе TCRT5000

```
const int leftSensor = 2;                    // Левый датчик
const int rightSensor = 3;        // Правый датчик
const int motorLeft = 9;                     // Левый мотор
const int motorRight = 10;       // Правый мотор

void setup() {
  pinMode(leftSensor, INPUT);
  pinMode(rightSensor, INPUT);
  pinMode(motorLeft, OUTPUT);
  pinMode(motorRight, OUTPUT);
}

void loop() {
  int leftValue = digitalRead(leftSensor);
  int rightValue = digitalRead(rightSensor);

  if (leftValue == LOW && rightValue == LOW) {
    // Двигаемся вперед
    digitalWrite(motorLeft, HIGH);
    digitalWrite(motorRight, HIGH);
  } else if (leftValue == LOW && rightValue == HIGH) {
    // Поворот налево
```

```
        digitalWrite(motorLeft, LOW);
        digitalWrite(motorRight, HIGH);
    } else if (leftValue == HIGH && rightValue == LOW) {
        // Поворот направо
        digitalWrite(motorLeft, HIGH);
        digitalWrite(motorRight, LOW);
    } else {
        // Оба датчика не видят линию
        digitalWrite(motorLeft, LOW);
        digitalWrite(motorRight, LOW); // Остановка или поиск линии
    }
}
```

Листинг 3. Считывание значений с датчиков линии

Описание ультразвукового датчика расстояния для Arduino

Ультразвуковой датчик расстояния — это устройство, позволяющее измерять расстояние до объекта с помощью ультразвуковых волн (рисунок 56). Он широко используется в проектах на базе Arduino благодаря своей простоте и точности. Датчик работает по принципу отправки ультразвукового импульса и измерения времени, необходимого для его отражения от объекта и возвращения обратно.

Основные характеристики:

- **Рабочий диапазон**: от 2 см до 4 м
- **Рабочая частота**: 40 кГц
- **Напряжение питания**: 5 В
- **Ток потребления**: менее 15 мА
- **Угол обнаружения**: около 15°

Основные компоненты:

1. **Ультразвуковой преобразователь (Трансимитер и Ресивер)**:
   - Трансимитер излучает ультразвуковые волны.
   - Ресивер принимает отраженные волны.
2. **Микроконтроллер**: Обрабатывает сигналы и рассчитывает расстояние.
3. **Контрольные выводы**:

- **Trig**: Входной вывод для активации ультразвукового импульса.
- **Echo**: Вывод, который отправляет сигнал обратно, указывая время, за которое сигнал прошел.

Подключение к Arduino:

1. Подключите вывод **VCC** к 5 В.
2. Подключите вывод **GND** к земле.
3. Подключите вывод **Trig** к любому цифровому пину (например, 9).
4. Подключите вывод **Echo** к другому цифровому пину (например, 10).

Рисунок 56. Ультразвуковой датчик расстояния

Пример кода для Arduino и ультразвукового датчика расстояния:

```
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop() {
  long duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
```

```
  distance = (duration * 0.034 / 2); // делим на 2, так как сигнал идет туда и обратно

  Serial.print("Расстояние: ");
  Serial.print(distance);
  Serial.println(" см");
  delay(500);
}
```

Листинг 4. Измерение расстояния до препятствия

Применение:

- Измерение расстояния до препятствий.
- Автоматизация и робототехника.
- Системы безопасности.
- Интерактивные проекты.

Ультразвуковые датчики являются универсальным инструментом в мире электроники и идеальны для множества приложений, связанных с измерением расстояния.

Езда по линии с объездом препятствия

Для создания робота, который может ехать по линии и объезжать препятствия с использованием двух датчиков линии и одного ультразвукового датчика расстояния, необходимо реализовать код для управления двигателями через драйвер TB6612 и логики работы с датчиками. Пример кода на Arduino, который реализует это поведение, выглядит следующим образом:

Пример кода машинки с датчиками линии и ультразвуковым датчиком расстояния:

```
// Подключения
#define LINE_SENSOR_LEFT A0  // Левый датчик линии
#define LINE_SENSOR_RIGHT A1 // Правый датчик линии
#define TRIG_PIN 7           // Пин триггера ультразвукового датчика
#define ECHO_PIN 6           // Пин эха ультразвукового датчика
#define DIST_THRESHOLD 20    // Пороговое расстояние для обнаружения препятствий (в сантиметрах)

// Пины драйвера TB6612 для моторов
#define AIN1 4   // Направление мотора A
#define AIN2 5   // Направление мотора A
```

```
#define PWMA 9   // ШИМ мотор A
#define BIN1 10  // Направление мотора B
#define BIN2 11  // Направление мотора B
#define PWMB 3   // ШИМ мотор B

// Функция для настройки направления движения и скорости моторов
void moveMotors(int leftSpeed, int rightSpeed) {
  if (leftSpeed > 0) {
    digitalWrite(AIN1, HIGH);
    digitalWrite(AIN2, LOW);
  } else {
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
  }

  if (rightSpeed > 0) {
    digitalWrite(BIN1, HIGH);
    digitalWrite(BIN2, LOW);
  } else {
    digitalWrite(BIN1, LOW);
    digitalWrite(BIN2, HIGH);
  }

  analogWrite(PWMA, abs(leftSpeed));
  analogWrite(PWMB, abs(rightSpeed));
}

// Функция для измерения расстояния с ультразвукового датчика
long getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = duration * 0.034 / 2;
```

```
  return distance;
}

void setup() {
  pinMode(LINE_SENSOR_LEFT, INPUT);
  pinMode(LINE_SENSOR_RIGHT, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(PWMB, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  // Чтение значений с датчиков линии
  int leftSensor = digitalRead(LINE_SENSOR_LEFT);
  int rightSensor = digitalRead(LINE_SENSOR_RIGHT);

  // Чтение расстояния до препятствия
  long distance = getDistance();

  // Если препятствие ближе порогового значения, остановка и объезд
  if (distance < DIST_THRESHOLD) {
    moveMotors(0, 0); // Остановка
    delay(500); // Пауза

    // Поворот для объезда
    moveMotors(-150, 150); // Поворот вправо
    delay(100); // Время для поворота
    moveMotors(150, 150); // Отъезд от линии
```

```
    delay(100); // Время для отъезда от линии
    moveMotors(150, -150); // Поворот влево
    delay(100); // Время для поворота
    moveMotors(150, 150); // Объезд препятствия
    delay(100); // Время для объезда препятствия
    moveMotors(150, -150); // Поворот влево
    delay(100); // Время для поворота
    moveMotors(150, 150); // Возвращение на линию
    delay(100); // Время для озвращения на линию
    moveMotors(-150, 150); // Поворот вправо
    delay(100); // Время для поворота
  }

  // Логика движения по линии
  if (leftSensor == LOW && rightSensor == LOW) {
    // Оба датчика на линии — ехать прямо
    moveMotors(200, 200);
  } else if (leftSensor == LOW && rightSensor == HIGH) {
    // Левый датчик на линии, правый вне — поворот влево
    moveMotors(100, 200);
  } else if (leftSensor == HIGH && rightSensor == LOW) {
    // Правый датчик на линии, левый вне — поворот вправо
    moveMotors(200, 100);
  } else {
    // Если оба датчика вне линии — остановка
    moveMotors(0, 0);
  }
}
```

Листинг 5. Код для езды по линии и объезда

2.5. Оборудование

1. **Датчики линии (2 шт.)**: для определения черной линии на светлой поверхности.
2. **Ультразвуковой датчик (например, HC-SR04)**: для измерения расстояния до препятствий.
3. **Драйвер двигателей TB6612**: для управления двумя моторами робота.

Подключение:

1. **Датчики линии** подключены к аналоговым пинам Arduino.
2. **Ультразвуковой датчик** подключен к цифровым пинам (один для триггера, другой для эха).
3. **Моторы** подключены к драйверу TB6612, который управляется Arduino через цифровые пины для направления и ШИМ для скорости.

Алгоритм работы:

1. Датчики линии проверяют наличие черной линии (возвращают логический 0 на черной линии и логическую 1 на светлой поверхности).
2. Если оба датчика находятся над линией, робот движется вперед.
3. Если один из датчиков покидает линию, робот корректирует курс, замедляя один из моторов, чтобы вернуться на линию.
4. Ультразвуковой датчик измеряет расстояние до препятствий. Если обнаружено препятствие на определенном расстоянии, робот останавливается, поворачивает и продолжает движение.
5. Управление моторами осуществляется через драйвер TB6612 с использованием ШИМ для регулировки скорости.

Описание алгоритма:

1. **Чтение датчиков линии**: Два датчика линии читают значения с поверхности. Если один из датчиков обнаруживает черную линию (низкий сигнал), то робот корректирует свое движение, чтобы оставаться на линии.
2. **Измерение расстояния**: Ультразвуковой датчик измеряет расстояние до объектов. Если расстояние до препятствия меньше порогового значения (например, 20 см), робот останавливается и поворачивает, чтобы объехать препятствие.

3. **Управление двигателями**: Управление двигателями осуществляется через драйвер TB6612. В зависимости от состояния датчиков линии и ультразвукового датчика, скорости и направления двигателей изменяются, чтобы поддерживать движение по линии и объезжать препятствия.

Приведенные в качестве примера программы упрощены для понимания и наглядности работы. Для увеличения скорости движения по линии и плавности хода робота необходимо реализовывать ПИД-регулятор и более сложные алгоритмы обработки как датчиков линии, так и датчиков препятствий.

Указанных алгоритмов и программ достаточно для того, чтобы робот начал двигаться, но для того, чтобы добиться успеха необходимо их серьезно усовершенствовать.